GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521 356.

Department of Computer Science and Engineering

HANDOUT

on

MOBILE APPLICATION DEVELOPMENT

IV Year – I Semester 2018-19 CSE

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

Program Educational Objectives

- Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.
- Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations
- Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with professional clients and the public.

HANDOUT ON MOBILE APPLICATION DEVELOPMENT

Class & Sem. :IV B.Tech – I Semester Year :2018-19
Branch : CSE & IT Credits: 3

==

1. Brief History and Scope of the Subject

As feature phones got faster the possibilities for phone apps expanded and it was Java Micro Edition that won the race to provide a platform for developing them.Java ME started life as JSR 68, replaced Personal Java and quickly became so popular that it evolved into numerous standards for use across phones, PDAs and other embedded devices like set top boxes. Devices implement profiles (like the Mobile Information Device Profile) which are subsets of configurations (like the Connected Limited Device Configuration).CLDC, designed for devices with total memory of 160KB to 512KB, contains the bare minimum of Java-class libraries required for operating a virtual machine.

MIDP, designed for mobile phones, includes a GUI, an API for data storage and even (in MIDP 2.0) a basic 2D gaming API. Applications here are called MIDlets. MIDP pretty much became an industry standard for mobile phones.Java ME spawned an open source implementation, Mika VM, which contains the class libraries for implementing the Connected Device Configuration.JME was the undisputed king of mobile platforms, it's used in the Bada and Symbian operating systems and implementation existed for Windows CE, Windows Mobile and Android.

2. Pre-Requisites

- OOP Concepts
- Basic knowledge in core java
- XML

3. Course Objectives:

- To prepare students with skills and knowledge of mobile application development using J2ME technology.
- Understand the Android OS architecture and able to develop the applications for mobile devices.

4. Course Outcomes:

At the end of the course, the students will be able to

CO1: Configure a J2ME environment for development

CO2: Plan and design of J2ME applications

CO3: Access and work with database under J2ME

CO4: Reproduce the installation of the Android Eclipse SKD.

CO5: Implement the user interface for android applications

CO6: Use best design practices for mobile development, designing applications for performance and responsiveness and also implement communication between the mobile devices.

5. Program Outcomes:

Graduates of the Computer Science and Engineering Program will have an ability to

a. apply knowledge of computing, mathematics, science and engineering fundamentals to solve complex engineering problems.

b. formulate and analyze a problem, and define the computing requirements appropriate to its solution using basic principles of mathematics, science and computer engineering.

c. design, implement, and evaluate a computer based system, process, component, or software to meet the desired needs.

d. design and conduct experiments, perform analysis and interpretation of data and provide valid conclusions.

e. use current techniques, skills, and tools necessary for computing practice.

f. understand legal, health, security and social issues in Professional Engineering practice.

g. understand the impact of professional engineering solutions on environmental context and the need for sustainable development.

h. understand the professional and ethical responsibilities of an engineer.

i. function effectively as an individual, and as a team member / leader in accomplishing a common goal.

j. communicate effectively, make effective presentations and write and comprehend technical reports and publications.

k. learn and adopt new technologies, and use them effectively towards continued professional development throughout the life.

l. understand engineering and management principles and their application to manage projects in the software industry.

6. Mapping of Course Outcomes with Program Outcomes:

	a	b	c	d	e	f	g	h	i	j	k	l
CO1	L				H						H	
CO2			H	L	M		M					
CO3					M						H	H
CO4					H							
CO5	H										H	M
CO6			M	M	H							

7. Prescribed Text Books

1. James Keogh J2ME: The Complete Reference,McGraw-Hill/Osborne.
2. James C Sheusi Android Application development for java programmers, Cengage Learning.

8. Reference Text Books

1. John W. Muchow, Core J2ME Technology by Prentice Hall PTR; 1st edition.
2. Michael juntao yuan, Enterprise J2ME : developing mobile java applications pearson Education ,2004.

3. Ray Richpater, Beginning java ME platform, après,2009.
4. Wallace Jackson, Android apps for absolute Beginners Apress.
5. Wei-meng lee,wiley Begining android 4 application development.
6. Ziguord Mednieks, Laired Dornin, G.Blake Meike &Masumi Nakameera, Programming android, Orelly

9. URLs and Other E-Learning Resources

URLs:

- **http://freevideolectures.com/blog/2011/07/mobile-application-development-courses/**
- **http://web.stanford.edu/class/cs193a/lectures.shtml**
- **https://www.youtube.com/watch?v=1g2Pdge3-88**
- **https://j2meprograms.blogspot.in/2016/08/video-lecture-on-mobile-application.html**

E-Learning Materials:

Journals:

INTERNATIONAL JOURNALS:

- **IEEE Conference Publications**
- **IJCSNS International Journal of Computer Science and Network Security**
- **International journal of Interactive mobile technologies.**

NATIONAL JOURNALS:

- **Journal of Information Technology and software Engineering**
- **Indian journal of science and Technology**

10. Digital Learning Materials:

a. SONET CDs – J2ME platform
b. IIT CDs – Andriod App Development

11. Lecture Schedule / Lesson Plan

Topic	No. of Periods	
	Theory	Tutorial
UNIT –1: J2ME Overview & Architecture J2ME Overview		
Inside J2ME, How J2ME Is Organized	1	1
J2ME and Wireless Devices	2	
What J2ME Isn't, Other Java Platforms for Small Computing Devices?	1	
J2ME Architecture ,Small Computing Device Requirements,	1	1
Run-Time Environment, MIDlet Programming	2	
Java Language for J2ME ,J2ME Software Development Kits	2	
Hello World J2ME Style Multiple MIDlets in a MIDlet Suite	2	
J2ME Wireless Toolkit	1	
UNIT – 2: Event Processing & Canvas Commands, Items, and Event Processing		
J2ME User Interfaces ,Display Class	2	1
The Palm OS Emulator ,Command Class	2	
Item Class ,Exception Handling	2	2
High-Level Display: Screens :Screen Class , Alert Class	2	
Form Class ,Item Class ,List Class	2	
Text Box Class, Ticker Class.	2	
Canvas: The Canvas, User Interactions Graphics	2	
Clipping Regions, Animation	2	
UNIT – 3: Database concepts Record Management System		
Record Storage ,Writing and Reading Records, Writing and Reading Mixed Data Types	2	1
Record Enumeration ,Sorting Records	2	
Searching Records Record Listener	2	1
J2ME Database Concepts: Data, Databases, Database Schema	2	
Overview of the JDBC Process, Database Connection	2	
UNIT – 4: Introduction to Android Installation and Configuration of android starting an android application project		
Components, debugging with eclipse	2	1
Application design: the screen layout and Main.xml file	2	
Components ids, controls	1	1
Creating and configuring android Emulator	2	
Communication with emulator	1	
UNIT – 5: User Interface controls and user interface		
Radio buttons, radio group	1	1
The spinner, data picker	1	
Buttons, array adapter	2	1
View class: combining graphics with a touch listener	1	
Canvas, bitmap, paint ,motion event	2	
UNIT – 6: Android Applications working with images		
Display images ,using images stored on android devices	2	1
Image view, working with text files, working with data tables	2	

Using sqlite ,using xml for data exchange	2	2
Cursor, content values ,XML PUL Parser, XML Resource parser	2	
Client -server applications: socket, server socket	1	
HTTPURL connection ,URL	1	
Total No.of Periods:	**62**	**14**

Learning Material

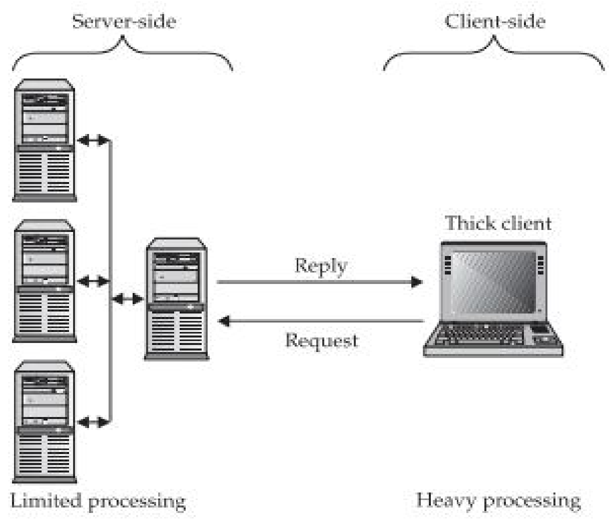
Different editions of JAVA Platforms:

- **J2SE (Java Platform, Standard Edition):**
  - J2SE also known as Core Java, this is the most basic and standard version of Java and a basic foundation for all other editions.
  - It consists of a wide variety of general purpose API's (like java.lang, java.util) as well as many special purpose APIs.
  - J2SE is mainly used to create applications for Desktop environment.
- **J2EE (Java Platform, Enterprise Edition)**
  - J2EE stands for Java 2 Enterprise Edition for applications which run on servers.
  - J2EE uses many components of J2SE, as well as, has many new features like Servlets, JavaBeans, Java Message Services to support distributed/Web development.
  - J2EE uses HTML, CSS, JavaScript etc., so as to create web pages and web services. It is also one of the most widely accepted web development standard.
- **J2ME (Java Platform, Micro Edition)**
  - J2ME stands for Java 2 Micro Edition which is mainly concentrated for the applications running on embedded systems, mobiles and small devices.(which was a constraint before it's development)
  - Constraints included limited processing power, battery limitation, small display etc.
  - The basic aim of this edition was to work on mobiles, wireless devices, set top boxes etc.

- J2ME apps help in using web compression technologies, which in turn, reduce network usage, and hence cheap internet accessibility.
- J2ME uses many libraries and API's of J2SE, as well as, many of its own.

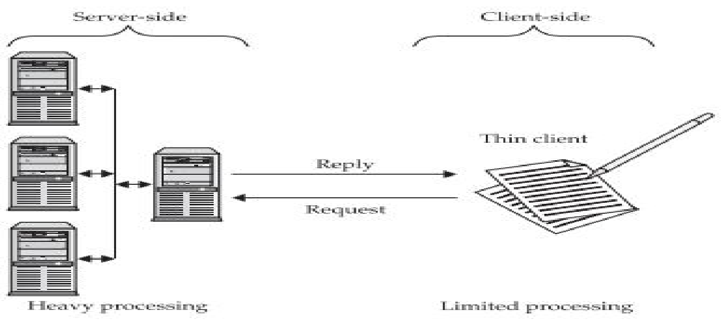
INSIDE J2ME:

- J2ME is targeted to developers of intelligent wireless devices and small computing devices that need to incorporate cross-platform functionality in their products.
- Consumers of mobile and small computing devices expect for quick response time, compatibility with companion services, and full-featured applications in a small computing device.
- Developers need to harness the power of existing front-end and back-end software found on business computers and transfer this power onto small, mobile, and wireless computing devices. J2ME enables this transformation to occur with minimal modifications, assuming that applications are scalable in design so that an application can be custom-fitted to resources available on a small computing device.
- To build applications that run on cell phones, personal digital assistants, and various consumer and industrial appliances, we have to balance between a thick client and a thin client.
- A thick client is front-end software that contains the logic to handle a sizable amount of data processing for the system.



Thick client applications handle most processing locally

- A thin client is front-end software that depends on back-end software for much of the system processing.



Thin client applications rely on server-side software for nearly all processing

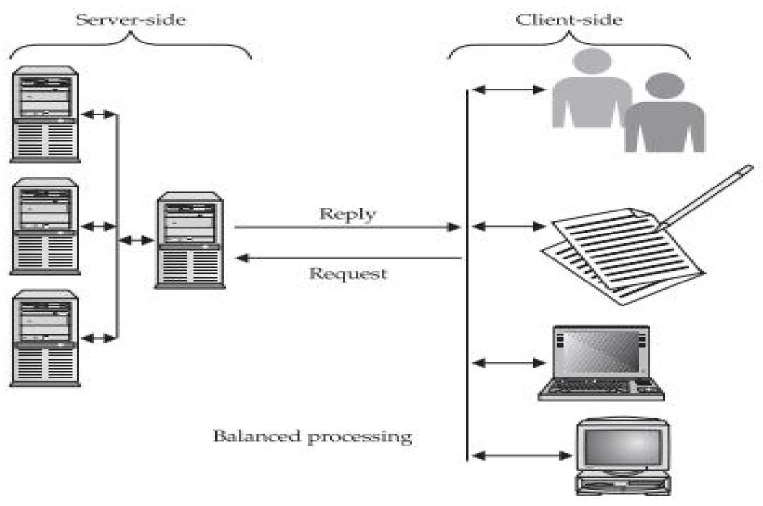
- Example: Consider a wireless small computing device is used to transact orders on the floor of a stock exchange. The wireless device has software to handle user interactions such as displaying an electronic form on the screen, collecting user input, processing the input, and displaying results of the processing on the screen. The order form is displayed on the screen, and the user enters information into the order form using various input conventions commonly found in small wireless devices. The device collects the order information and then processes the order using a combination of software on the wireless device and software running on a back-end system that receives the order through a wireless connection.

- Processing on the wireless device might involve two steps:

  - ❖ First the software performs a simple validation process to assure that all fields on the form contain information. Next the order is transmitted to the back-end system.

  - ❖ The back-end system handles adjusting account balances and other steps involved in processing the order. A confirmation notice is returned by the back-end system to the wireless device, which displays the confirmation notice on the screen.

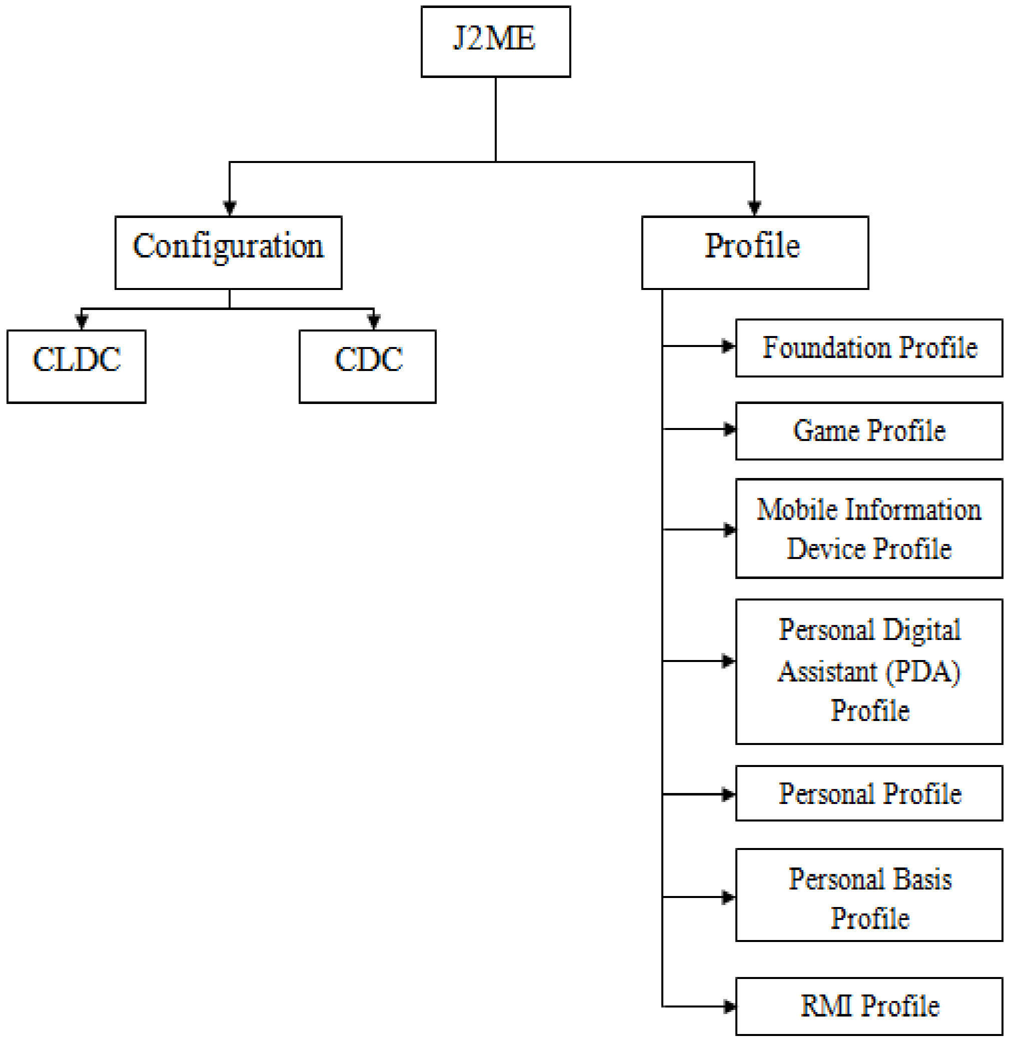
- A key benefit of using J2ME is that J2ME is compatible with all Java-enabled devices. A Java-enabled device is any computer that runs the Java Virtual Machine.



A J2ME application is a balance between local and server-side processing.

<u>How J2ME is organized:</u>

- T
  he lack of uniform hardware configuration among the small computing devices poses a formidable challenge for the Java Community Process Program, which is charged with developing standards for the JVM and the J2ME for small computing devices.
- J
  2ME must service many different kinds of small computing devices, including screen phones, digital set-top boxes used for cable television, cell phones, and personal digital assistants.
- T
  he challenge for the Java Community Process Program is to develop a Java standard that can be implemented on small computing devices that have nonstandard hardware configurations.



J2ME Configurations: The configuration defines the basic run-time environment as a set of core classes and a specific JVM that run on specific types of devices. Currently, two configurations exist for J2ME, though others may be defined in the future:

Connected Limited Device Configuration (CLDC):

- The CLDC is designed for 16-bit or 32-bit small computing devices with limited amounts of memory.
- CLDC devices usually have between 160KB and 512KB of available memory and are battery powered.
- They also use an inconsistent, small-bandwidth network wireless connection and may not have a user interface.

- CLDC devices use the KJava Virtual Machine (KVM) implementation, which is a stripped-down version of the JVM.
- CLDC devices include pagers, personal digital assistants, cell phones, dedicated terminals, and handheld consumer devices with between 128KB and 512KB of memory.

Connected Device Configuration (CDC):

- CDC devices use a 32-bit architecture, have at least two megabytes of memory available, and implement a complete functional JVM.
- CDC devices include digital set-top boxes, home appliances, navigation systems, point-of-sale terminals, and smart phones.

J2ME Profiles:

- The profile defines the type of devices supported by your application. Specifically, it adds domain-specific classes to the J2ME configuration to define certain uses for devices. Profiles are built on top of configurations.
- Currently, seven Profiles exist for J2ME, though others may be defined in the future:

  Foundation Profile:

  ❖ The Foundation Profile is used with the CDC configuration and is the core for nearly all other profiles used with the CDC configuration because the Foundation Profile contains core Java classes.

  Game Profile:

  ❖ The Game Profile is also used with the CDC configuration and contains the necessary classes for developing game applications for any small computing device that uses the CDC configuration.

  Mobile Information Device Profile (MIDP):

  ❖ The Mobile Information Device Profile (MIDP) is used with the CLDC configuration and contains classes that provide local storage, a user interface, and networking capabilities to an application that runs on a mobile computing device such as Palm OS devices. MIDP is used with wireless Java applications.

PDA Profile:

❖ T

he PDA Profile (PDAP) is used with the CLDC configuration and contains classes that utilize sophisticated resources found on personal digital assistants. These features include better displays and larger memory than similar resources found on MIDP mobile devices (such as cell phones).

Personal Profile:

❖ T

he Personal Profile is used with the CDC configuration and the Foundation Profile and contains classes to implement a complex user interface. The Foundation Profile provides core classes, and the Personal Profiles provide classes to implement a sophisticated user interface, which is a user interface that is capable of displaying multiple windows at a time.

Personal Basis Profile:

❖ T

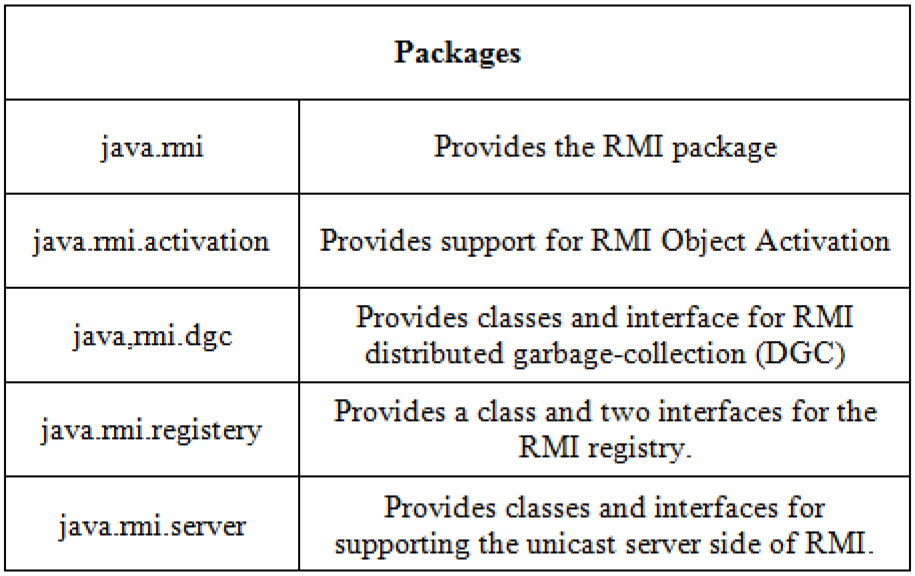
he Personal Basis Profile is similar to the Personal Profile in that it is used with the CDC configuration and the Foundation Profile. However, the Personal Basis Profile provides classes to implement a simple user interface, which is a user interface that is capable of displaying one window at a time.

RMI Profile:

❖ T

he RMI Profile is used with the CDC configuration and the Foundation Profile to provide Remote Method Invocation classes to the core classes contained in the Foundation Profile.

Packages:



Packages	
java.rmi	Provides the RMI package
java.rmi.activation	Provides support for RMI Object Activation
java.rmi.dgc	Provides classes and interface for RMI distributed garbage-collection (DGC)
java.rmi.registery	Provides a class and two interfaces for the RMI registry.
java.rmi.server	Provides classes and interfaces for supporting the unicast server side of RMI.

J2ME and Wireless Devices:

- The wireless devices such as cell phones keep their end users connected to the outside world at anytime from anywhere.
- They offer great connectivity that other types of devices couldn't offer.
- Application development for these wireless devices is going to be in great demand. Because mobile communication devices or wireless devices utilize a number of different application platforms and operating systems.
- Without changing the code, an application written for one device cannot run on another device.
- Mobile communication devices lack a standard application platform and operating system, which is a concern for developing applications for these devices.

➢ **WAP (Wireless Application Protocol)**

- WAP forum became the initial group that provides standards for wireless technology.
- The WAP forum created mobile communications device standards referred to as the WAP standard,
- The WAP standard is an enhancement of HTML, XML, and TCP/IP.

- The WAP standard provides Wireless Markup Language specification, which consists of a mix of HTML and XML and is used by developers to create documents that can be displayed by a **micro browser**.
- A **micro browser is a** tiny web browser that operates on a mobile communications device.
- The WAP standard also includes wireless Telephony Application Interface (WTAI) specification and the WMLScript specification.
    - ❖ WTAI is used to create an interface for applications that run on a mobile communications device.
    - ❖ WMLScript is a simple version of JavaScript.
- The WAP forum provided the framework for the developers to build applications for the mobile communication devices; they still had to overcome a problem.
    - ❖ The complexity of mobile communications devices, rapid growth of the market, and high demand for industrial-strength mobile communications applications.

➢ J2ME

- Many applications designed for mobile communications devices require the device to process information beyond the capabilities of the WAP specification.
- J2ME provided the standard to process the information which cannot be handled by WAP standard.
- J2ME applications referred to as a MIDlet can run on any mobile communication device that implements a JVM and MIDP (Mobile Information Device Profile).
- This encourages the developers to build applications for mobile communication devices without the risk that the application is device independent.
- But J2ME is not a replacement for the WAP specification because both are opposite technologies.
    - ❖ Developers whose applications are light-client based they use WML ad WMLScript.
    - ❖ Developers whose applications are heavy that requires complicated processing on the device they turn to J2ME.

<u>What J2ME isn't:</u>

- **Misunderstandings of J2ME**
  - Although J2ME is J2SE without some classes, developer's shouldn't assume that existing J2SE applications would run in the J2ME environment without requiring modification to the code, because of the resource constraints imposed by small computing devices.
  - Some J2SE applications require classes that are not available in J2ME. Likewise, resources required by the J2SE application may not be available on the small computing device.
  - Another misconception of J2ME is the Java Virtual Machine implementation on the small computing device.
  - Small computing devices use one of two Java Virtual Machine implementations.
    - Devices that use the CDC configuration use the full Java Virtual Machine implementation.
    - While devices that use the CLDC configuration use the KJava Virtual Machine implementation.
  - A MIDlet is not invoked the same way as a J2SE application is invoked because many small computing devices don't have a command prompt.
  - MIDlets are controlled by Application Management Software (AMS).
    - This AMS is provided by the manufacturer of small computing devices or third-party vendors might also create.
  - AMS (Application Management Software) interacts with native operations of a small computing device and controls the life cycle of a MIDlet.

- ❖ The life cycle consists of installation and upgrades as well as version management and uninstalling the application. So, AMS is responsible for starting, managing execution, and stopping the MIDlet.

Other Java Platforms for Small Computing Devices

J2ME isn't the only Java platform designed for small computing devices.

Other Java platforms for small computing devices are

- Embedded Java,
- Java Card,
- PersonalJava

Embedded Java:

- Embedded Java is the Java platform used for small computing devices that are dedicated to one purpose and have a 32-bit processor and 512KB of ROM and RAM.
- Embedded Java is based on JDK 1.1 and is being replaced by the CDLC configuration.

Java Card:

- Java Card is the Java platform used for smart cards, the smallest computing device that supports Java.
- The Java Card VM runs on small computing devices that have 16KB of nonvolatile memory and 512 bytes of volatile memory.
- However, unlike the Embedded Java platform, there isn't any movement to replace Java Card with J2ME because of the resource constraints of the current generation of smart cards.
- Future smart card generations will probably have great resources available and be compatible with the CDLC configuration.

PersonalJava:

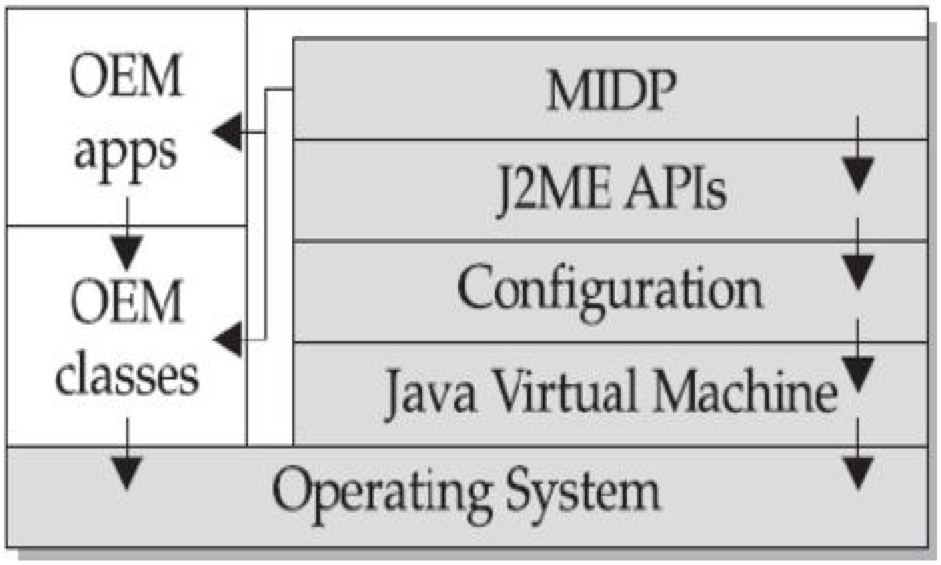
- PersonalJava is the Java platform used for small computing devices that have a maximum of 2MB of ROM and a minimum of 1MB of RAM, such as large PDAs and mobile communications devices.
- PersonalJava uses JDK 1.1.8 and the JVM and will be replaced by the CDC configuration and the Personal Basis Profile and Personal Profile.

J2ME Architecture:

The J2ME architecture is designed for small computing devices that have limited memory and limited computational capability. J2ME architecture doesn't replace the operating system of a small computing device. Instead J2ME architecture consists of layers located above the native operating system, collectively referred to as the **Connected Limited Device Configuration (CLDC).** The CLDC, which is installed on top of the operating system, forms the run-time environment for small computing devices.

The J2ME architecture comprises three software layers.

- The first layer is the **configuration** layer that includes the Java Virtual Machine (JVM), which directly interacts with the native operating system. The configuration layer also handles interactions between the profile and the JVM.
- The second layer is the **profile layer**, which consists of the minimum set of application programming interfaces (APIs) for the small computing device.
- The third layer is the **Mobile Information Device Profile (MIDP),** which contains Java APIs for user network connections, persistence storage, and the user interface. It also has access to CLDC libraries and MIDP libraries.



Layers of the J2ME Architecture

A small computing device has two components supplied by the original equipment manufacturer (OEM). These are classes and applications.

- OEM classes are used by the MIDP to access device-specific features such as sending and receiving messages and accessing device-specific persistent data.
- OEM applications are programs provided by the OEM, such as an address book. OEM applications can be accessed by the MIDP.

Small Computing Device Requirements:

There are minimum resource requirements for a small computing device to run a J2ME application.

Every device must have both minimal requirements for hardware and native operating system.

Hardware Requirements:

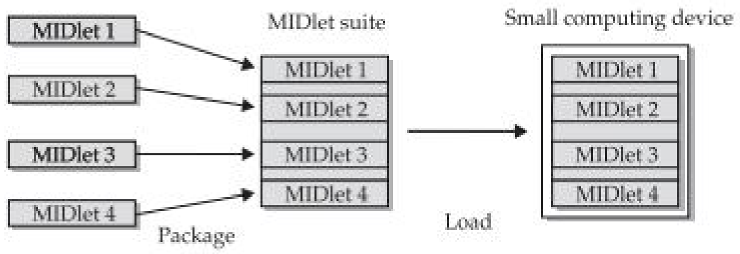
- The device must have minimum of 96 × 54 pixel display that can handle bitmapped graphics and have a way for users to input information, such as a keypad, keyboard, or touch screen.
- At least 128 Kilobytes (KB) of nonvolatile memory is needed to run Mobile Information Device (MID), and 8 KB of nonvolatile memory is needed for storage of persistent application data.
- To run JVM, 32 KB of volatile memory must be available.
- The device must also provide two-way network connectivity.

Operating System Requirements:

- The native operating system must implement exception handling, process interrupts, be able to run the JVM, and provides schedule capabilities.
- Furthermore, all user input to the operating system must be forwarded to the JVM; otherwise the device cannot run a J2ME application.
- The native operating system doesn't need to implement a file system to run a J2ME application, it must be able to write and read persistent data (data retained when the device is powered down) to nonvolatile memory.

Run-Time Environment:

- A MIDlet is a J2ME application designed to operate on an MIDP small computing device. A MIDlet is defined with at least a single class that is derived from the
  javax .microedition.midlet.MIDlet abstract class.
- Related MIDlets are bundled into a **MIDlet suite**, which is contained within the same package and implemented simultaneously on a small computing device. All MIDlets within a MIDlet suite are considered a group and must be installed and uninstalled as a group.



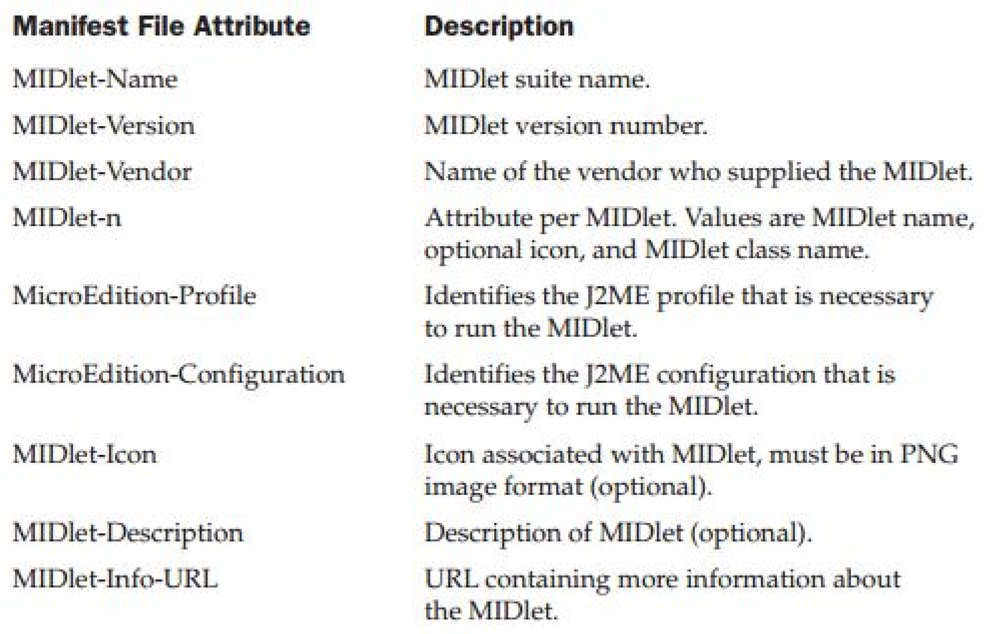
MIDlets are packaged into MIDlet suites, which are loaded in a small computing device.

- Members of a MIDlet suite share resources of the host environment and share the same instances of Java classes and run within the same JVM.
- If three MIDlets from the same MIDlet suite run the same class, only one instance of the class is created at a time in the Java Virtual Machine. MIDlet suite members share the same data, including data in persistent storage such as user preferences.
- Sharing data among MIDlets exposes each MIDlet to data errors caused by concurrent read/write access to data. This risk is reduced by **synchronization primitives** on the MIDlet suite level that restrict access to volatile data and persistent data.
- Data cannot be shared between MIDlets that are not from the same MIDlet suite because the MIDlet suite name is used to identify data associated with the suite.
- A MIDlet suite is installed, executed, and removed by the application manager running on the device. The manufacturer of the small computing device provides the application manager.

- Once a MIDlet suite is installed, each member of the MIDlet suite is given access to classes of the JVM and CLDC by the application manager.
- A MIDlet can access classes defined in the MIDP to interact with the user interface, network, and persistent storage. The application manager makes the **Java archive (JAR) file and the Java application descriptor (JAD) file** available to members of the MIDlet suite.

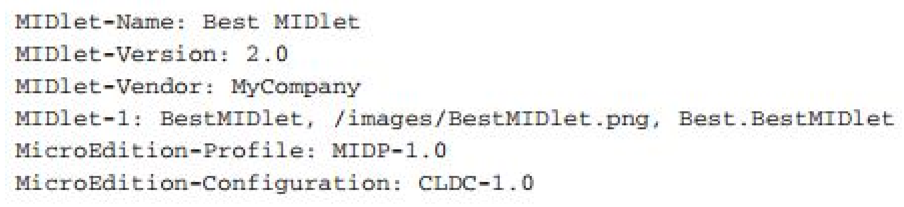
Java Archive File

- All the files necessary to implement a MIDlet suite must be contained within a production package called a Java archive (JAR) file.
- The files include **MIDlet classes, graphic images and the Manifest file.**
- The **manifest file** contains a list of attributes and related definitions that are used by the application manager to install the files contained in the JAR file onto the small computing device.
- Nine attributes are defined in the manifest file; but six of these attributes are optional.



Manifest File Attribute	Description
MIDlet-Name	MIDlet suite name.
MIDlet-Version	MIDlet version number.
MIDlet-Vendor	Name of the vendor who supplied the MIDlet.
MIDlet-n	Attribute per MIDlet. Values are MIDlet name, optional icon, and MIDlet class name.
MicroEdition-Profile	Identifies the J2ME profile that is necessary to run the MIDlet.
MicroEdition-Configuration	Identifies the J2ME configuration that is necessary to run the MIDlet.
MIDlet-Icon	Icon associated with MIDlet, must be in PNG image format (optional).
MIDlet-Description	Description of MIDlet (optional).
MIDlet-Info-URL	URL containing more information about the MIDlet.

- The first six attributes are required for every manifest file. Failure to include them in the manifest file causes the application manager to halt the installation of the JAR file.
- Example of a MIDlet:



```
MIDlet-Name: Best MIDlet
MIDlet-Version: 2.0
MIDlet-Vendor: MyCompany
MIDlet-1: BestMIDlet, /images/BestMIDlet.png, Best.BestMIDlet
MicroEdition-Profile: MIDP-1.0
MicroEdition-Configuration: CLDC-1.0
```

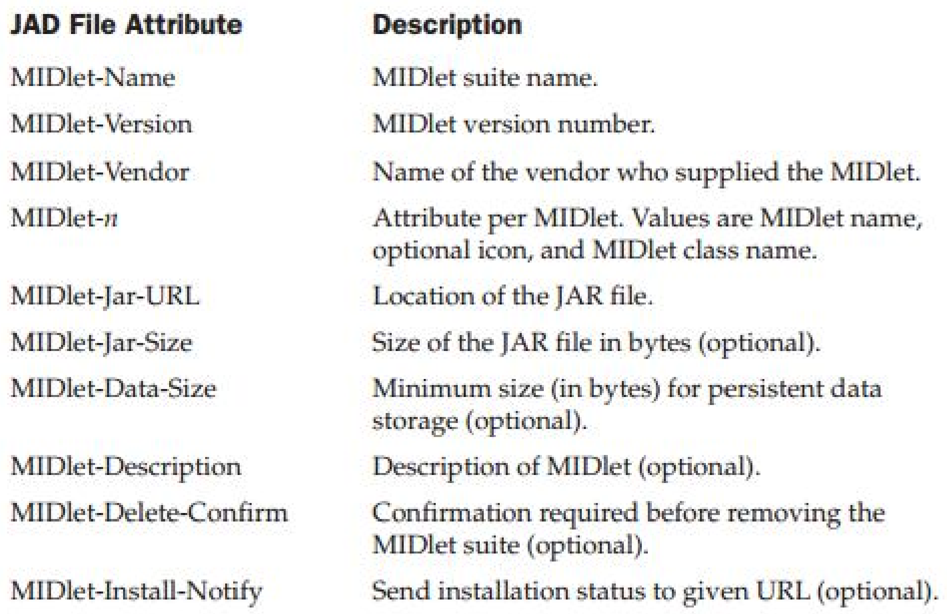
- Entries in the manifest are **name:value** pairs and therefore can appear in any order within the manifest file. Each pair must be terminated with a carriage return. Whitespace between the colon and the attribute value is ignored when the application manager reads the manifest file.
- The **MIDlet-Name** attribute specifies the name of the MIDlet suite, which is **Best MIDlet** in this example.
- The **MIDlet-Version** and **MIDlet-Vendor** attributes identify the **version number** of the MIDlet suite and the **company or person** who provided the MIDlet suite.
- The **MIDlet-n** attribute contains **information about each MIDlet** that is in the JAR file. The number of the MIDlet replaces the letter n. In this example, the n is replaced with the digit 1 because there is only one MIDlet in the MIDlet suite. The MIDlet-n attribute can contain three values that describe the MIDlet. A comma separates each value. The first value is the name of the MIDlet, which is BestMIDlet. Next is an optional value that specifies the icon that will be used with the MIDlet. In this example, BestMIDlet.png is the icon. The icon must be in the PNG image format. And the last value for the MIDlet-n attribute is the MIDlet class name, which is Best.BestMIDlet. The application manager uses the class name to load the MIDlet.
- The next MIDlet-n attribute is the **MicroEdition-Profile** whose value is the J2ME profile that is required to run the MIDlet. In this example the MIDP-1.0 profile is required.
- Last MIDlet-n attribute is the **MicroEdition-Configuration**. The MicroEdition-Configuration attribute identifies the J2ME configuration that is necessary to run the MIDlet.

Java Descriptor File

- A JAD file is also used to provide the application manager with additional content information about the JAR file to determine whether the MIDlet suite can be implemented on the device.

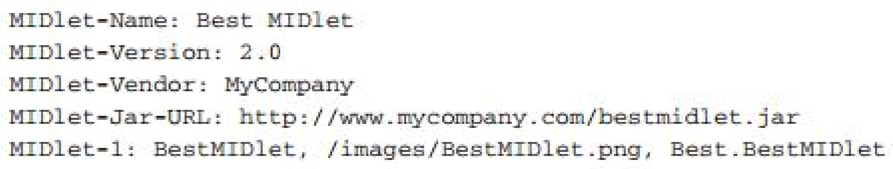
- A JAD file is similar to a manifest in that both contain attributes that are name: value pairs. Name: value pairs can appear in any order within the JAD file. There are five required system attributes for a JAD file:

  - ❖ **MIDlet-Name**
  - ❖ **MIDlet-Version**
  - ❖ **MIDlet-Vendor**
  - ❖ **MIDlet-n**
  - ❖ **MIDlet-Jar-URL.**
- A system attribute is an attribute that is defined in the J2ME specification. All JAD files must have the .jad extension.



JAD File Attribute	Description
MIDlet-Name	MIDlet suite name.
MIDlet-Version	MIDlet version number.
MIDlet-Vendor	Name of the vendor who supplied the MIDlet.
MIDlet-*n*	Attribute per MIDlet. Values are MIDlet name, optional icon, and MIDlet class name.
MIDlet-Jar-URL	Location of the JAR file.
MIDlet-Jar-Size	Size of the JAR file in bytes (optional).
MIDlet-Data-Size	Minimum size (in bytes) for persistent data storage (optional).
MIDlet-Description	Description of MIDlet (optional).
MIDlet-Delete-Confirm	Confirmation required before removing the MIDlet suite (optional).
MIDlet-Install-Notify	Send installation status to given URL (optional).

- **Example:**



```
MIDlet-Name: Best MIDlet
MIDlet-Version: 2.0
MIDlet-Vendor: MyCompany
MIDlet-Jar-URL: http://www.mycompany.com/bestmidlet.jar
MIDlet-1: BestMIDlet, /images/BestMIDlet.png, Best.BestMIDlet
```

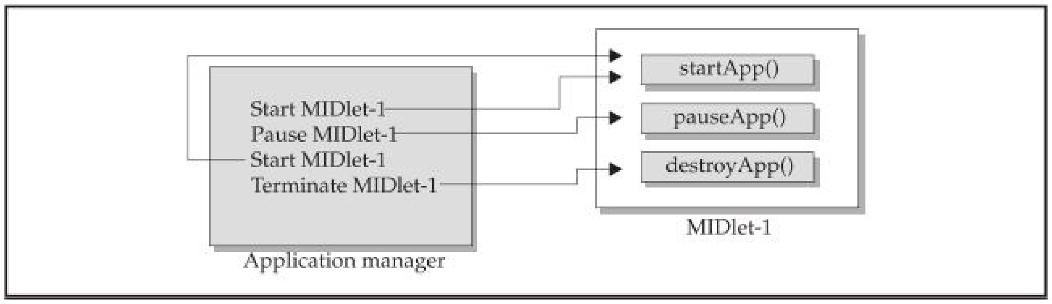
- The first three attributes in the JAD file are identical to attributes in the manifest file. The MIDlet-Jar-URL attribute contains the URL of the JAR file, which in this example is called **bestmidlet.jar.** And the last required attribute in the JAD file is the MIDlet-n attribute that defines a MIDlet of the MIDlet suite identical to the MIDlet-n attribute of the manifest. A MIDlet-n attribute is required for each MIDlet in the MIDlet suite.

- The values of the **MIDlet-Name, MIDlet-Version, and MIDletVendor** attributes in the JAD file **must match the same attributes** in the manifest. If the values are different, the JAR file is not installed. Other attributes that are not the same are overridden by attributes in the descriptor.

MIDlet Programming:

- Programming a MIDlet is similar to creating a J2SE application in that we define a class and related methods. However, a MIDlet is less robust than a J2SE application because of the restrictions imposed by the small computing device.
- A MIDlet is a class that extends the MIDlet class and is the interface between application statements and the run-time environment, which is controlled by the application manager.
- A MIDlet class must contain three abstract methods that are called by the application manager to manage the life cycle of the MIDlet. These abstract methods are **startApp(), pauseApp(),** and **destroyApp().**
- The **startApp()** method is called by the application manager when the MIDlet is started and contains statements that are executed each time the application begins execution.
- The **pauseApp()** method is called before the application manager temporarily stops the MIDlet. The application manager restarts the MIDlet by recalling the **startApp()** method.
- The **destroyApp()** method is called prior to the termination of the MIDlet by the application manager.

```
public class BasicMIDletShell extends MIDlet

{

        public void startApp()

        {

        }

        public void pauseApp()

        {
        }
        public void destroyApp( boolean unconditional)
        {
        }
}
```



- Both the **startApp()** and **pauseApp()** methods are public and have no return value nor parameter list.
- The **destroyApp()** method is also a public method without a return value. However, the destroyApp() method has a boolean parameter that is set to true if the termination of the MIDlet is unconditional, and false if the MIDlet can throw a **MIDletStateChangeException** telling the application manager that the MIDlet does not want to be destroyed just yet.
- User interactions are managed by user interface MIDP API classes.
- These APIs enable a developer to display screens of data and prompt the user to respond with an appropriate command. The command causes the MIDlet to execute one of three routines: perform a computation, make a network request, or display another screen.

Event Handling:

- A MIDlet is an event-based application. All routines executed in the MIDlet are invoked in response to an event reported to the MIDlet by the application manager.
- The initial event that occurs is when the MIDlet is started and the application manager invokes the startApp() method.

- A Command object is used to present a user with a selection of options to choose from when a screen is displayed. Each screen must have a CommandListener.
    - A CommandListener monitors user events with a screen and causes the appropriate code to execute based on the current event.

User Interfaces:

- The design of a user interface for a MIDlet depends on the restrictions of a small computing device. Some small computing devices contain resources that provide a rich user interface, while other more resource-constrained devices offer a modest user interface.
- A Form is the most commonly invoked user interface element found in a MIDlet and is used to contain other user interface elements. Text is placed on a form as a StringItem, a List, a ChoiceGroup, and a Ticker.
    - A StringItem contains text that appears on a form that cannot be changed by the user.
    - A List is an itemized options list from which the user can choose an option.
    - A ChoiceGroup is a related itemized options list. And a Ticker is text that is scrollable.
- A user enters information into a form by using the Choice element, TextBox, TextField, or DateField elements.
- The Choice element returns an option that the user selected.
- TextBox and TextField elements collect textual information from a user and enable the user to edit information that appears in these user interface elements.
- The DateField is similar to a TextBox and TextField except its contents are a date and time.
- An Alert is a special Form that is used to alert the user that an error has occurred.
- An Alert is usually limited to a StringItem user interface element that defines the nature of the error to the user.

Device Data:

- Small computing devices don’t have the resources necessary to run an onboard database management system (DBMS).

- Some of these devices lack a file system. Therefore, a MIDlet must read and write persistent data without the advantage of a DBMS or file system.
- A MIDlet can use an MIDP class—RecordStore—and two MIDP interfaces— RecordComparator and RecordFilter—to write and read persistent data.
    - ➢ A RecordStore class contains methods used to write and read persistent data in the form of a record.
    - ➢ Persistent data is read from a RecordStore by using either the RecordComparator interface or the RecordFilter interface.

Java Language for J2ME

- CDC implements the full J2SE available, but CLDC implements a stripped-down J2SE because of the **limited resources in small computing devices**.

- **Floating-point math** is a missing feature of J2ME. Floating point math requires special processing hardware to perform floating-point calculations. But most small computing devices lack such hardware and therefore are unable to process floating-point calculations. So our MIDlet cannot use any floating-point data types or calculations.

- The most notable difference between the Java language used in J2SE and J2ME is the **absence of support for the finalize() method**. The finalize() method in J2SE is automatically called before an instance of a class terminates and typically contains statements that free previously allocated resources. However, resources in a small computing device are too scarce to process the finalize() method.

- There are **reduced number of error-handling exceptions** that are supported in J2ME. Exception handling drains system resources, which are precious in a small computing device. This is the primary reason for trimming the number of error-handling exceptions. Run-time errors are automatically responded to by the native operating system by restarting the small computing device.

- Changes were also made in the **Java Virtual Machine** that runs on a small computing device because of resource constraints. One such change occurs with the class loader. JVM for small computing devices requires a custom class loader that is supplied by the device manufacturer and cannot be replaced or modified.
- Another feature lacking in the JVM is the **ThreadGroup class**. You cannot group threads. All threads are handled at the object level,
- Two other features of J2SE that are missing from J2ME are: **weak references and the Reflection classes.**
- The standard JVM uses class file verification to protect applications from malicious code through the use of a security manager. However, this process is replaced with a two-step process because of the limited resources available on small computing devices.
  - ➢ The first step is called **preverification** and occurs outside the small computing device prior to loading the MIDlet. Preverification requires that additional attributes called stack maps are inserted into a class file by software before the second step runs. Stack maps describe the MIDlet's variables and operands located on the interpreter stack.
  - ➢ After preverification is completed, the **MIDlet class is loaded into the device**, and the verifier within the small computing device validates each instruction in the MIDlet class. The MIDlet class is automatically rejected if the verifier detects an error.

<u>J2ME Software Development Kits</u>

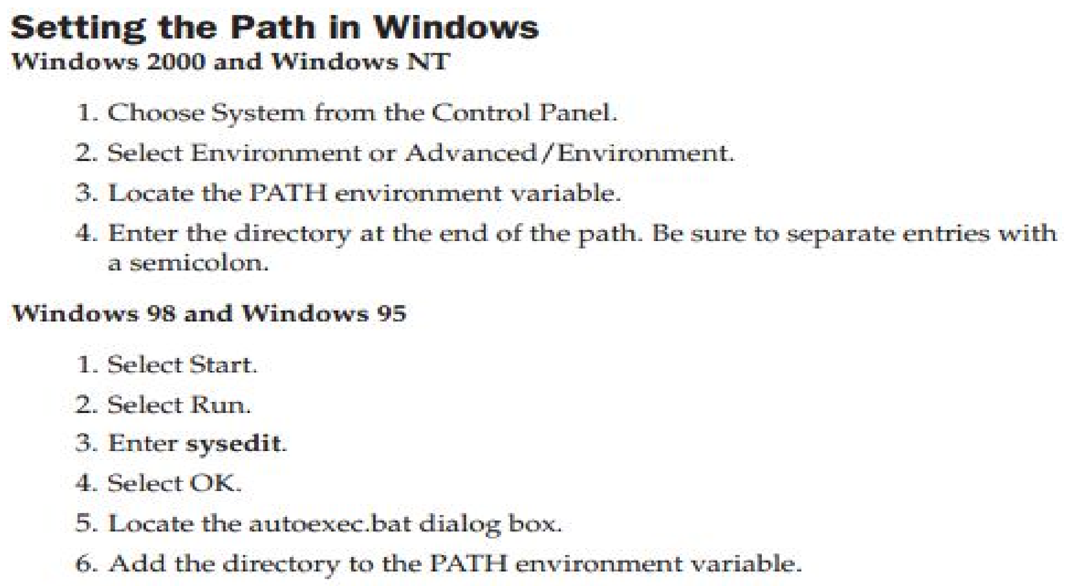
- A MIDlet is built using free software packages that are downloadable from the java.sun .com web site
- Three software packages need to be downloaded from java.sun.com.
  - ➢ Java Development Kit (1.3 or greater) (java.sun.com/ j2se/downloads.html),
  - ➢ Connected Limited Device Configuration (CLDC) (java.sun. com/products/cldc/),

  - ➢ Mobile Information Device Profile (MIDP) (java.sun.com/ products/midp/).

- We need the J2ME Wireless Toolkit to develop MIDlets for handheld devices (java.sun.com/products/j2mewtoolkit/download.html).
- Each of these software packages contains installation instructions that need to followed to assure proper installation of each package.

Steps for Installation

- First, install the Java development kit. The Java development kit contains the Java compiler and the jar.exe, which is used to create Java archive files
- After downloading the Java development kit package, unzip the package and run the installation program. Choose default directory.
- Once the Java development kit is installed, place the c:\jdk\bin directory, or whatever directory you selected for the Java development kit, on the PATH environment variable.
- This enables you to invoke the Java compiler from anywhere on your computer.



Setting the Path in Windows

Windows 2000 and Windows NT

1. Choose System from the Control Panel.
2. Select Environment or Advanced/Environment.
3. Locate the PATH environment variable.
4. Enter the directory at the end of the path. Be sure to separate entries with a semicolon.

Windows 98 and Windows 95

1. Select Start.
2. Select Run.
3. Enter **sysedit**.
4. Select OK.
5. Locate the autoexec.bat dialog box.
6. Add the directory to the PATH environment variable.

- Install the CLDC once the Java development kit is installed. Unzip the downloaded CLDC files from the java.sun.com web site onto the d:\j2me directory (J2ME_HOME) on your computer. We have to create the j2me directory if one doesn't exist. Unzipping the CLDC package creates the j2me_cldc subdirectory below the j2me directory.

- The j2me_cldc has a bin subdirectory that contains the K Virtual Machine and the preverifier executable files for an assortment of platforms such as win32. Each platform is in its own subdirectory under j2me_cldc. Add the j2me\j2me_cldc\bin\win32 subdirectory to the PATH environment variable .You should substitute win32 subdirectory with the appropriate subdirectory for your platform.
- Next, download and unzip the MIDP file. Be sure to use \j2me as the directory for the MIDP file. Unzipping the MIDP file creates a midp directory. The name of this directory might vary depending on the version that you download.
- The midp1.0.3fcs directory also contains a bin subdirectory. We need to include the \j2me\midp1.0.3fcs\bin subdirectory in the PATH environment variable.
- Next, create two environment variables. These are CLASSPATH and MIDP_HOME. The CLASSPATH environment variable identifies the path to be searched whenever a class is invoked. The MIDP_HOME environment variable identifies the location of the \lib directory that contains the internal.config file and the system.config file. Set the CLASSPATH to **d:\j2me\midp1.0.3fcs\classes;.** CLASSPATH terminates with a period. The period implies the current directory and will cause the current directory to be searched if a class is not found in the \j2me\midp1.0.3fcs\classes directory.
- Set the MIDP_HOME environment variable to **d:\j2me\midp1.0.3fcs**

Hello World J2ME Style

- We can create your first MIDlet once the Java development kit, Connected Limited Device Configuration (CLDC), and Mobile Information Device Profile (MIDP) are installed.
- Create a directory structure within which you can create and run MIDlets. The directories that are used for examples are:

- j2me
- j2me\src
- j2me\src\greeting
- j2me\tmp_classes
- j2me\midlets

Creating HelloWorld

- Let us create two MIDlets in this application. The first MIDlet is called **HelloWorld** and the other MIDlet is **GoodbyeWorld.** The HelloWorld MIDlet shows how to create a simple MIDlet that can be invoked directly from the class and from a Java archive file. Create a MIDlet suite that contains two MIDlets. These are HelloWorld and GoodbyeWorld.

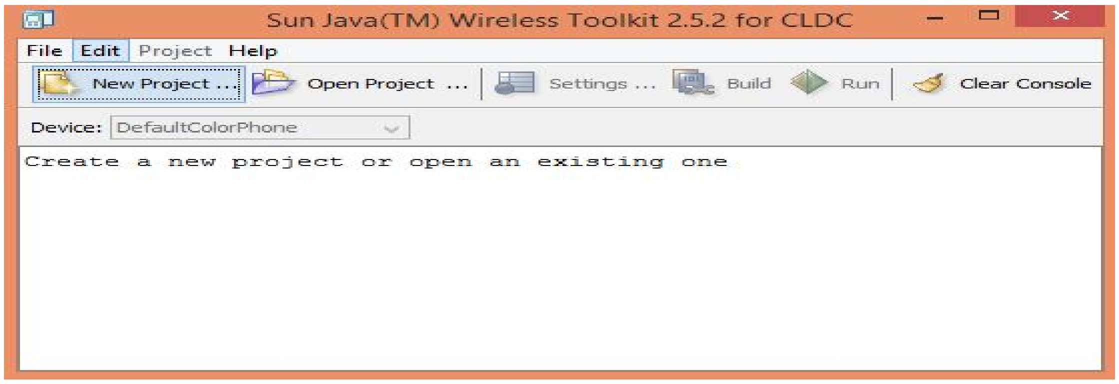
The HelloWorld MIDlet can be created as follows:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public    class    HelloWorld    extends    MIDlet    implements
CommandListener
{
        private Display display ;
        private TextBox textBox ;
        private Command quitCommand;
        public void startApp()
        {
                display = Display.getDisplay(this);
                quitCommand      =      new      Command("Quit",
Command.SCREEN, 1);
                textBox  =  new  TextBox("Hello  World",  "My  first
MIDlet", 40, 0);
                textBox .addCommand(quitCommand);
                textBox .setCommandListener(this);
                display .setCurrent(textBox );
        }
        public void pauseApp()
        {
        }
        public void destroyApp(boolean unconditional)
        {
        }
        public      void      commandAction(Command      choice,
Displayable displayable)
        {
                if (choice == quitCommand)
                {
                        destroy     App(false);
                        notifyDestroyed();
                }
        }
}
```

- Enter the above code into a text editor such as Notepad, and save the file in the j2me\src\greeting directory as HelloWorld.java.
- The MIDlet performs three basic functions:
    - ➢ To display a text box
    - ➢ To display a command on the screen
    - ➢ Then listen to events that occur while the MIDlet is running.
- The HelloWorld MIDlet is created by defining a class called HelloWorld that extends the MIDlet class and implements a CommandListener. The HelloWorld class contains three private data members and four methods. The data members are a Display object, a text box, and a command. The methods are startApp(), pauseApp(), and destroyApp() and the fourth method is called commandAction() is invoked by the application manager whenever an event occurs.
- Two packages must be imported at the beginning of the MIDlet to access MIDlet classes and lcdui classes.
- MIDlet classes are screen oriented and create a Display object and then place components of the screen into the Display object. The Display object is then invoked later in the MIDlet to display the screen on the small computing device.

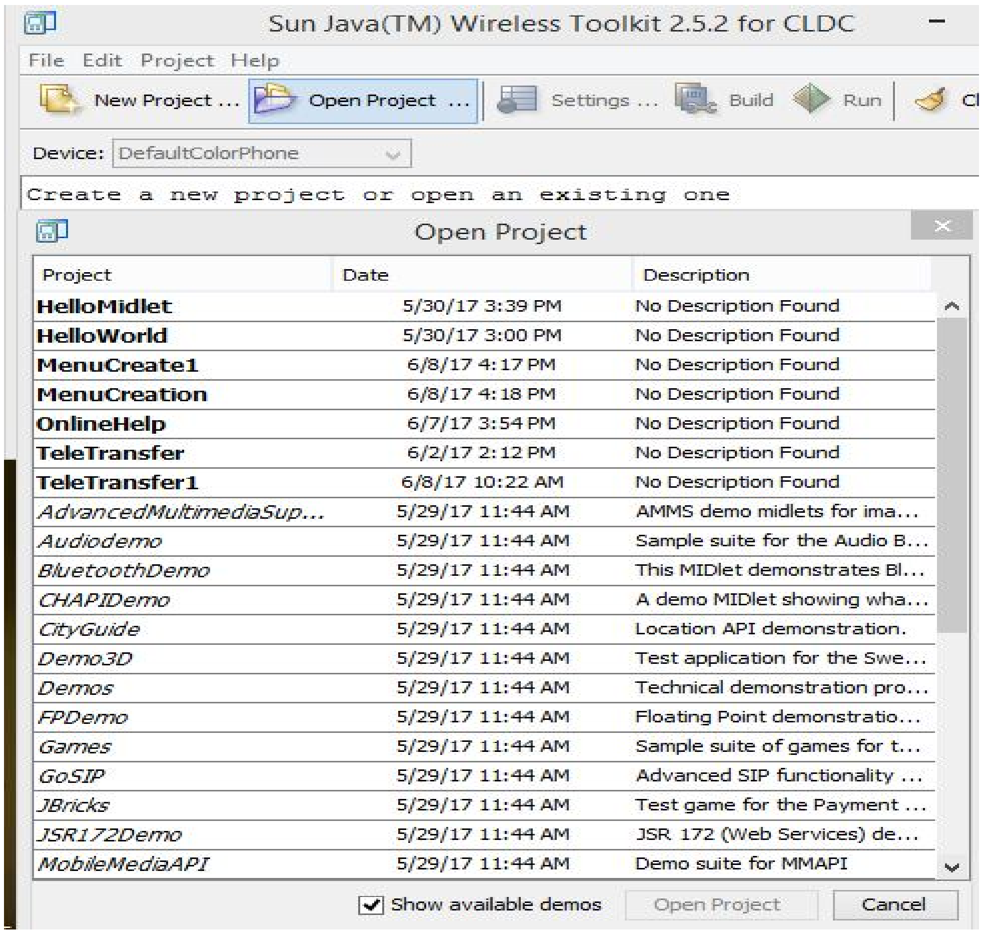
Compiling Hello World

- The Hello World source code files should be saved in the new j2me\src\greeting directory as HelloWorld.java.
- Next, we'll need to compile the HelloWorld MIDlet.
- Then open J2Me wireless tool-kit


Sun Java(TM) Wireless Toolkit 2.5.2 for CLDC
File Edit Project Help
New Project ... Open Project ... Settings ... Build Run Clear Console
Device: DefaultColorPhone

```
Create a new project or open an existing one
```

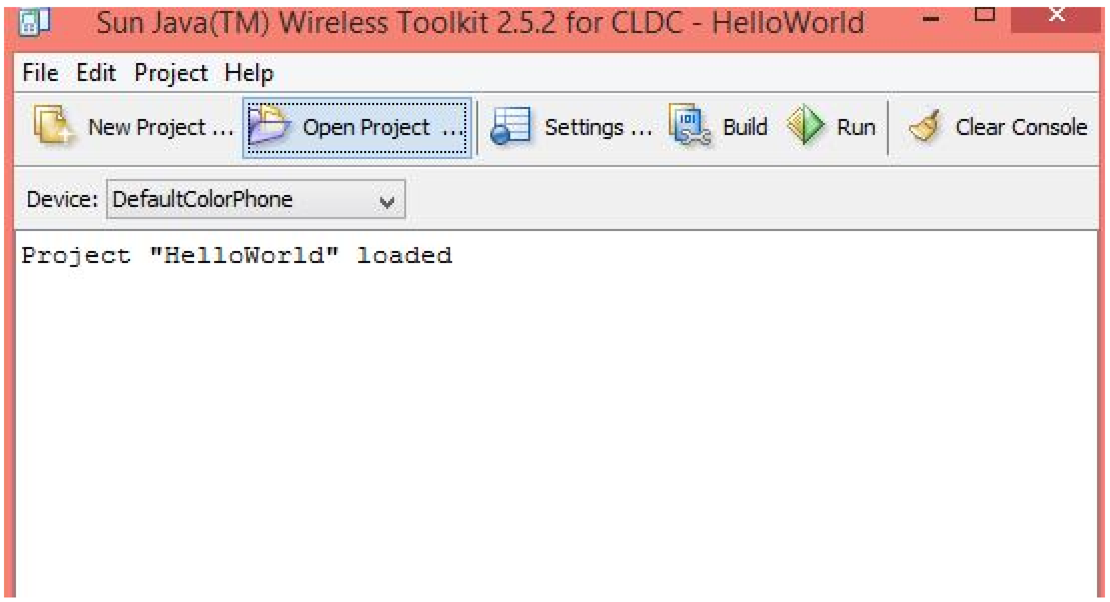
- Click on open project , then we will see all the saved projects


Sun Java(TM) Wireless Toolkit 2.5.2 for CLDC
File Edit Project Help
New Project ... Open Project ... Settings ... Build Run Cl
Device: DefaultColorPhone
Create a new project or open an existing one
Open Project

Project	Date	Description
HelloMidlet	5/30/17 3:39 PM	No Description Found
HelloWorld	5/30/17 3:00 PM	No Description Found
MenuCreate1	6/8/17 4:17 PM	No Description Found
MenuCreation	6/8/17 4:18 PM	No Description Found
OnlineHelp	6/7/17 3:54 PM	No Description Found
TeleTransfer	6/2/17 2:12 PM	No Description Found
TeleTransfer1	6/8/17 10:22 AM	No Description Found
AdvancedMultimediaSup...	5/29/17 11:44 AM	AMMS demo midlets for ima...
Audiodemo	5/29/17 11:44 AM	Sample suite for the Audio B...
BluetoothDemo	5/29/17 11:44 AM	This MIDlet demonstrates Bl...
CHAPIDemo	5/29/17 11:44 AM	A demo MIDlet showing wha...
CityGuide	5/29/17 11:44 AM	Location API demonstration.
Demo3D	5/29/17 11:44 AM	Test application for the Swe...
Demos	5/29/17 11:44 AM	Technical demonstration pro...
FPDemo	5/29/17 11:44 AM	Floating Point demonstratio...
Games	5/29/17 11:44 AM	Sample suite of games for t...
GoSIP	5/29/17 11:44 AM	Advanced SIP functionality ...
JBricks	5/29/17 11:44 AM	Test game for the Payment ...
JSR172Demo	5/29/17 11:44 AM	JSR 172 (Web Services) de...
MobileMediaAPI	5/29/17 11:44 AM	Demo suite for MMAPI

[✓] Show available demos

- Select the project we want to run, then it is loaded into run-time environment



```
Project "HelloWorld" loaded
```

Running Hello World:

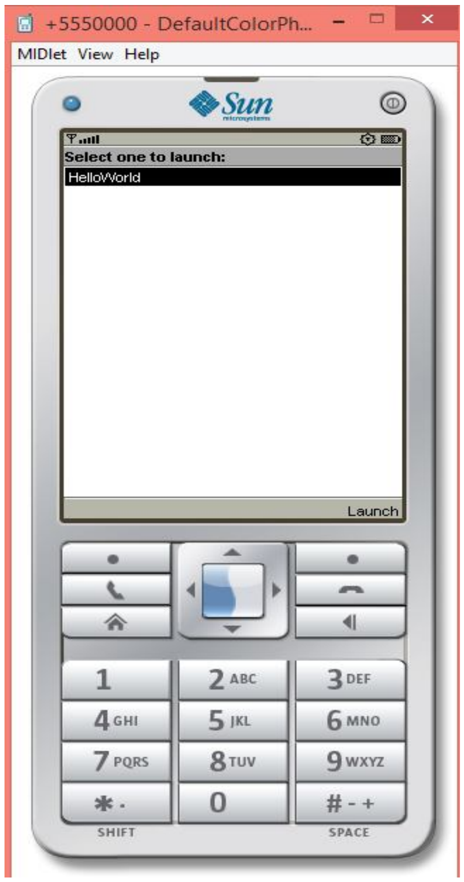
- A MIDlet should be tested in an emulator before being downloaded to a small computing device.
- An *emulator* is software that simulates how a MIDlet will run in a small computing device.
- There are two ways to run a MIDlet.
  - ❖ These are either by invoking the MIDlet class OR
  - ❖ By creating a JAR file,

  Then run the MIDlet from the JAR file. Let's begin by running the MIDlet class without the need of a JAR file. Make sure that j2me\src\ greeting is the current directory, and then enter the following command.

  midp -classpath d:\j2me\classes greeting.HelloWorld

Deploying Hello World

- After we click on Run the application will be deployed in emulator like this.



What to Do When Your MIDlet Doesn't Work Properly:

- Sometimes a MIDlet won't compile or run properly. Although each MIDlet is unique, there are a few common problems that cause a MIDlet to fail.
- Here are some of those problems
  - ❖ If the compiler, preverifier, JAR program, or emulator doesn't run from the command line, review the value of the PATH, CLASSPATH, and MIDP_HOME environment variables to be sure you have included the exact path to these programs. Also make sure that the current directory reference (a period) is included in the CLASSPATH environment variable.
  - ❖ Running out of environment space is a common problem on some platforms. This results in not enough room to store the complete value of an environment variable such as the PATH. You can work around this problem by creating an executable file, such as a batch file in Windows that sets the environment variables for J2ME components. Run this executable file before compiling and testing your MIDlet to

temporarily reset environment variables. The environment variables return to their original values the next time you restart your computer or log in.

- ❖ Many types of errors can occur during the compiling and packaging process. Some are syntax errors, which you'll be able to fix quickly by reviewing the source code.
- ❖ Other errors can be caused by poorly formed command line options and arguments, such as failing to insert a space between an option and a period when referencing the current directory.
- ❖ Another common occurrence is for a MIDlet suite to run fine in test but fail to run after downloaded to the small computing device. In this case, the application manager on the small computing device might reject the MIDlet suite because the MIDlet suite cannot be run on the device. An oversize MIDlet suite is a likely suspect.

Multiple MIDlets in a MIDlet Suite

- Multiple MIDlets are distributed in a single MIDlet suite. The application manager then displays each MIDlet as a menu option, enabling the user to run one of the MIDlets.
- Let's create a MIDlet to illustrate how to deploy a multiple MIDlet suite.
- The new MIDlet is called GoodbyeWorld. Enter the code into a text editor and save the file as GoodbyeWorld.java in the j2me\src\greeting directory. Make the j2me\src\greeting directory the current directory

```
  package greeting;

 import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class GoodbyeWorld extends MIDlet implements CommandListener
{
        private Display display ;
        private TextBox textBox ;
        private Command quitCommand;
        public void startApp()
 {
display = Display.getDisplay(this);
quitCommand = new Command("Quit", Command.SCREEN, 1);
textBox = new TextBox("Goodbye World", "My second MIDlet", 40, 0);
textBox .addCommand(quitCommand);
textBox .setCommandListener(this);
display .setCurrent(textBox );
 }
 public void pauseApp()
 {
 }
 public void destroyApp(boolean unconditional)
 {
 }
 public void commandAction(Command choice, Displayable displayable )
 {
if (choice == quitCommand)
{
        destroyApp(false);
        notifyDestroyed();
}
    }
}
```

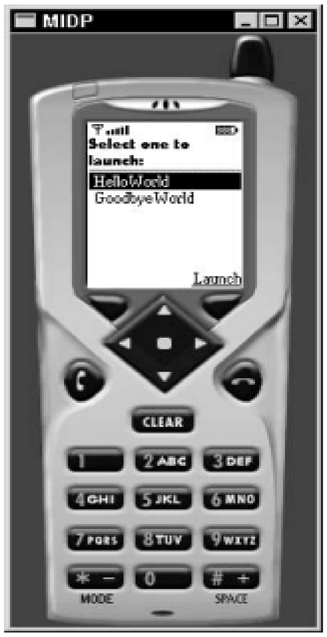
Compile both the HelloWorld.java and GoodbyeWorld.java files by entering the following command at the command line:

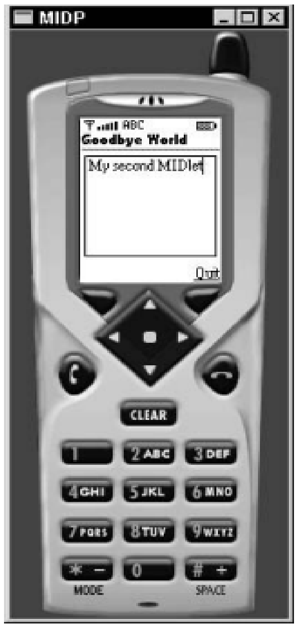
```
javac -d d:\j2me\tmp_classes -target 1.1 -bootclasspath

d:\j2me\midp1.0.3fcs\classes *.java
```

- **Preverify** these files by entering the following command at the command line:

preverify -d d:\j2me\classes -classpath d:\j2me\midp1.0.3fcs\classes d:\j2me\tmp_classes

- Move the cursor to the GoodbyeWorld MIDlet and select the center button on the emulator, the emulator's application manager launches the GoodbyeWorld MIDlet.
-





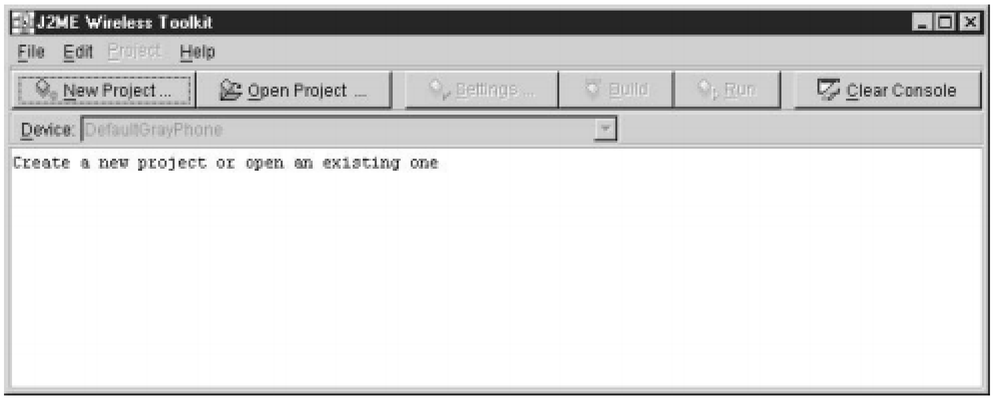
J2ME Wireless Toolkit

- Building and running a J2ME application at the command line is difficult, when you are creating a robust application consisting of several MIDlets.
- Creating your application within an integrated development environment is more productive than developing applications by entering commands at the command line.
- An integrated development environment is the **J2ME Wireless Toolkit** that is downloadable from java.sun.com/products/j2mewtoolkit/download.html.
- The J2ME Wireless Toolkit is used to develop and test J2ME applications by selecting a few buttons from a toolbar. The J2ME Wireless Toolkit is a stripped-down integrated development environment .It does not include an editor, a full debugger, and other amenities .

Building and Running a Project

- Download the J2ME Wireless Toolkit from the Sun web site. The Toolkit file is a selfextracting executable file. Run this executable after downloading the file, and the installation program creates all the directories required to run the Toolkit.

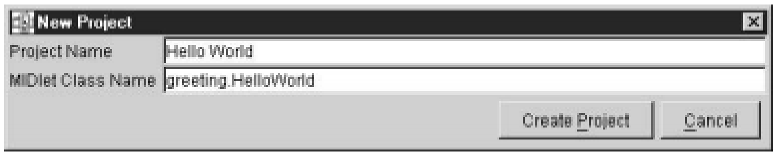
- The installed J2ME Wireless Toolkit is placed in the WTK104 directory, although the directory might have a variation of this name depending on the version of the Toolkit that you download.
- Ktoolbar is the executable within the directory that launches the Toolkit. The main window is displayed when you run ktoolbar.



```
Create a new project or open an existing one
```

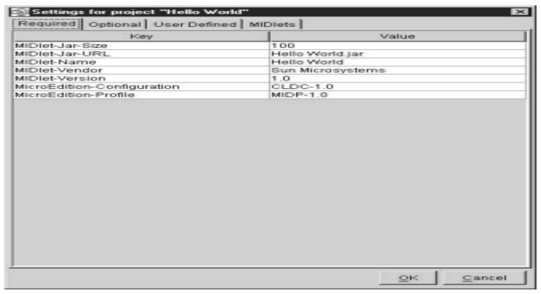
Main window of the J2ME Wireless Toolkit

- Create a new project by selecting the New Project button from the toolbar. You'll be prompted to enter a project name and class name Enter Hello World as the project name and greeting. HelloWorld as the class name, which is the name of the first MIDlet that is associated with the project.



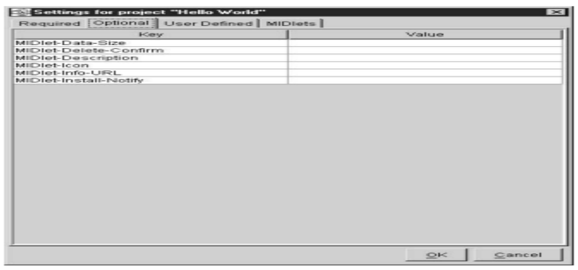
Project Name: Hello World
MIDlet Class Name: greeting.HelloWorld

- After selecting the Create Project button, the J2ME Wireless Toolkit automatically creates a directory structure for the project and also creates the manifest file and JAD file. We can see and modify attributes of these files by selecting the Settings option, which displays a dialog box containing a series of tabs.
  - The **Required tab** contains a list of attributes that are necessary for the manifest file and JAD file, as previously discussed in this chapter.
  - The **Optional tab** contains attributes that are common to many projects but not required to build and deploy a J2ME application.



Key	Value
MIDlet-Jar-Size	100
MIDlet-Jar-URL	Hello World.jar
MIDlet-Name	Hello World
MIDlet-Vendor	Sun Microsystems
MIDlet-Version	1.0
MicroEdition-Configuration	CLDC-1.0
MicroEdition-Profile	MIDP-1.0

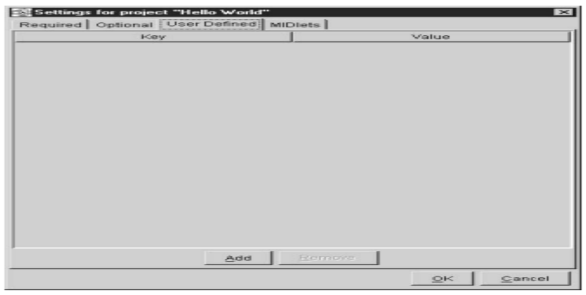
Required Tab

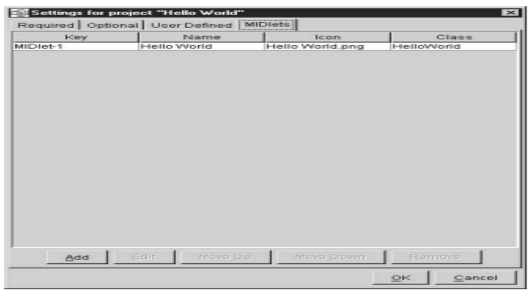


Key	Value
MIDlet-Data-Size	
MIDlet-Delete-Confirm	
MIDlet-Description	
MIDlet-Icon	
MIDlet-Info-URL	
MIDlet-Install-Notify	

Optional Tab

- ➢ The **User Defined tab** contains optional attributes specific to your application. This tab will be empty until you select the Add button and insert your own attributes.
- ➢ The **MIDlets tab** lists MIDlets of your project.





Key	Name	Icon	Class
MIDlet-1	Hello World	Hello World.png	HelloWorld

User Defined Tab **MIDlet Tab**

- A well-organized file structure is automatically created for our project as a result of starting a new project. Within the WTK104 directory, we see an apps subdirectory in which the projects you create are stored. Browse the apps subdirectory to see a subdirectory, which is the name that we gave to your project. A subdirectory of the apps directory is created for every project. And within the project's subdirectory is another set of subdirectories. These are:
  - ➢ src, containing source code
  - ➢ bin, containing the manifest.mf file, JAD file, and JAR file
  - ➢ classes, containing the compiled classes
  - ➢ tmpclasses, containing the preverify classes
  - ➢ res, containing image, data, and other files required by the application

UNIT-I

Assignment-Cum-Tutorial Questions

SECTION-A

I.*Objective Questions*

1. J2SE is mainly used for developing________________.
2. The most widely accepted web development standard is _____________.
3. The main aim of J2ME is to develop _____________, ___________________ and _____________.
4. CLDC stands for___________________.
5. CDC stands for ______________________.
6. MIDP stands for _______________________.
7. MIDlet is controlled by ______________________.
8. The two components supplied by the original equipment manufacturer (OEM) are ________and __________________.
9. The Java development kit contains the ________ and the ________, which is used to create Java archive files.
10. What is a MIDlet?
11. A platform, on which developers can build and implement programs to control small computing devices, is called ____________.
12. Two packages that are to be imported at the beginning of MIDlet programming are ___________ and _____________.
13. What is a Micro Browser?
14. J2ME architecture comprises of _________________,__________________ and _________________ layers.
15. What are the abstract methods in a MIDlet?
16. What are the five system attributes of a JAD file?
17. A MIDlet is __________ based application. [ ]
    a) App b) event c)architecture d) all
18. The method that is invoked by application manager when the MIDlet is started [ ]
    a) startApp() b) pauseApp() c) destroyApp() d) none
19. All the files necessary to implement a MIDlet suite must be contained within a production package called a _________ file.
    a) JAR b) JAD c) Both d) none [ ]
20. The most commonly invoked user interface element in a MIDlet is_______ [ ]
    a) Form b)Ticker c) List d) Choice Group
21. Which of the following are JAD file attributes: [ ]
    a) MIDlet-Name b) MIDlet-Version c) MIDlet –Vendor d) All

22. The CLDC is designed for _____ bit small computing devices.[ ]
    a) 16 b) 32 c) 38 d) both a & b
23. The PDA profile is used with the ________ configuration. [ ]
    a) CLDC b) CDC c) both a & b d) CPDC
24. ________ contains API used to create applications for small computing devices including wireless JAVA applications. [ ]
    a) J2SE b) J2EE c)J2 ME d) core Java
25. ________ File contains a list of attributes and related definitions that are used by the application manager to install the files contained in the JAR file onto the small computing device. [ ]
    a) Manifest b) JAD c) Text d) Deployment

SECTION-B

II) Descriptive Questions

1. Show the manifest file with six attributes.
2. Show the JAD file with its attributes.
3. Distinguish between Servlets and MIDlets
4. Explain J2ME architecture
5. What is a Profile? Explain J2ME profile
6. Explain MIDlet suite?
7. Explain about J2ME configurations
8. Explain about any three profiles in J2ME.
9. Explain about the runtime environment of J2ME.
10. Differentiate between J2SE, J2EE, and J2ME.
11. Identify the features of Java that are not available in J2ME.
12. Develop a MIDlet to print “Hello World”.
13. Discuss the misunderstandings about J2ME.
14. Interpolate the requirements needed for small computing devices.
15. Discuss about the abstract methods used in MIDlet programming with an example.
16. Demonstrate how J2ME is organized?
17. Elaborate the features of MIDlet programming?

UNIT – II

Learning Material

Syllabus:

Event Processing & Canvas Commands, Items, and Event Processing: J2ME User Interfaces ,Display Class ,The Palm OS Emulator ,Command Class ,Item Class ,Exception Handling .High-Level Display: Screens :Screen Class , Alert Class, Form Class ,Item Class ,List Class, Text Box Class, Ticker Class. Canvas: The Canvas, User Interactions Graphics, Clipping Regions, Animation

Objective:

To plan, design of j2me applications.

Learning Outcomes:

Student will be able to:

- Design User interfaces.
- Understand the various methods in High level Display classes.
- Understand the various methods in Low level Display classes.
- Manage Canvas on the screen.

J2ME User Interfaces:

User-interface requirements for small handheld devices are different from personal computers. Because the display size of handheld devices is smaller. In J2ME, the **CLDC** itself does not define any GUI functionality. The official GUI classes for the J2ME are included in profiles such as **MIDP** and are defined by **Java Community Process (JCP).**

In J2ME a developer can use one of three kinds of user interfaces for an application. These are

- Command
- Form
- Canvas

Command-based User Interface:

- ➢ A **Command-based** user interface consists of instances of the “Command” class.
- ➢ An instance of the Command class is a button that the user presses on the device to do a specific task.

- For example, Exit is an instance of the Command class associated with an Exit button on the keypad to terminate the application.

Form-based User-Interface:

- A Form-based user interface consists of an instance of the **Form** class that contains instances derived from the Item class such as text boxes, radio buttons, check boxes, lists and other conventions used to display information on the screen and to collect input from the user.

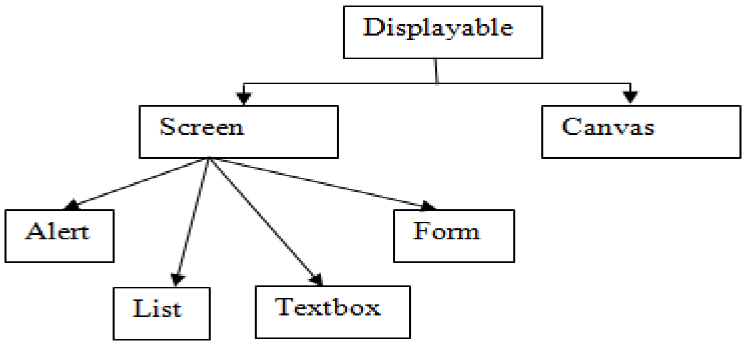
Canvas-based User Interface

- A canvas-based user interface consists of instances of the Canvas class within which the developer creates images such as those used in a game.

Display Class

Display class is used to manage the objects that can be displayed on the screen.

- The device's screen is referred to as the display, and we interact with the display by obtaining a reference to an instance of the MIDlets **Display** class.
- MIDlets can be pure background applications or applications interacting with the user.
- Interactive applications can get access to the display by obtaining an instance of the **Display** class.
- Every J2ME MIDlet that displays anything on the screen must obtain a reference to its **Display** instance. This instance is used to show instances of **Displayable** class on the screen.
- The Displayable class has two subclasses. These are the Screen class and the Canvas class.



- The **Screen** class contains a subclass called the Item class, Item class, which has its own subclasses used to display information or collect information from a user (such as forms, List, radio buttons, Textbox, Alerts).
- The **Screen** class and its derived classes are referred to as high-level user interface components.
- The **Canvas** class is used to display graphical images such as used for games.
- Displays created using the **Canvas** class are considered a low-level user interface and are used whenever we need to display a customized screen.
- A MIDlet can get its Display instance by calling

  Display.getDisplay (MIDlet midlet),

  where the MIDlet itself is given as parameter.
- The Display class and all other user interface classes of MIDP are located in the package **javax.microedition.lcdui**.
- The Display class provides a **setCurrent ()** method that sets the current display content of the MIDlet. The object that is to be displayed is passed to the **setCurrent ()** method as a parameter.
- The difference between Display and Displayable is that the Display class represents the display hardware, whereas Displayable is something that can be shown on the display.
- The **getCurrent ()** method of the Display class is used by a MIDlet to retrieve information about the instances of derivatives of the Displayable class.

A simple example to understand the Display and Displayable class
Step 1: First, we need to import the necessary **midlet** and **lcdui** packages

```
          import javax.microedition.midlet.*;
          import javax.microedition.lcdui.*;
```

Step 2: Every MIDP application is required to extend the MIDlet class.

```
     public class HelloMidp extends MIDlet {
```

Step 3: In the constructor, we obtain the Display and create a Form

```
          Display display;
          Form mainForm;
     public HelloMidp ()
     {
       mainForm = new Form ("HelloMidp");
```

```
}
```

A **Form** is a specialized **Displayable** class. The Form has a title that is given in the constructor. We do not add content to the form yet, so only the title will be displayed.

Step 4:

When MIDlet is started the first time, or when the MIDlet resumes from a paused state, the **startApp()** method is called by the program manager. Here, we set the display to form, thus requesting the form to be displayed:

```
public void startApp()
{
   display =Display.getDisplay (this);
   display.setCurrent (mainForm);
 }
```

Step 5: We need to provide an empty implementation because implementation of **pauseApp()** is mandatory:

```
public void pauseApp()
{
 }
```

Step 6: Like pauseApp(), implementation of destroyApp() is mandatory.

```
        public void destroyApp(boolean unconditional)
         {
        }
}
```

Program:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class HelloMidp extends MIDlet
{
        Display display;
        Form mainForm;
        public HelloMidp ()
        {
                 mainForm = new Form ("HelloMidp");
        }
        public void startApp()
        {
                display =Display.getDisplay (this);
                display.setCurrent (mainForm);
        }
        public void pauseApp()
        {
                    }
                    public void destroyApp(boolean unconditional)
                     {
                    }
        }
```

The Palm OS Emulator:

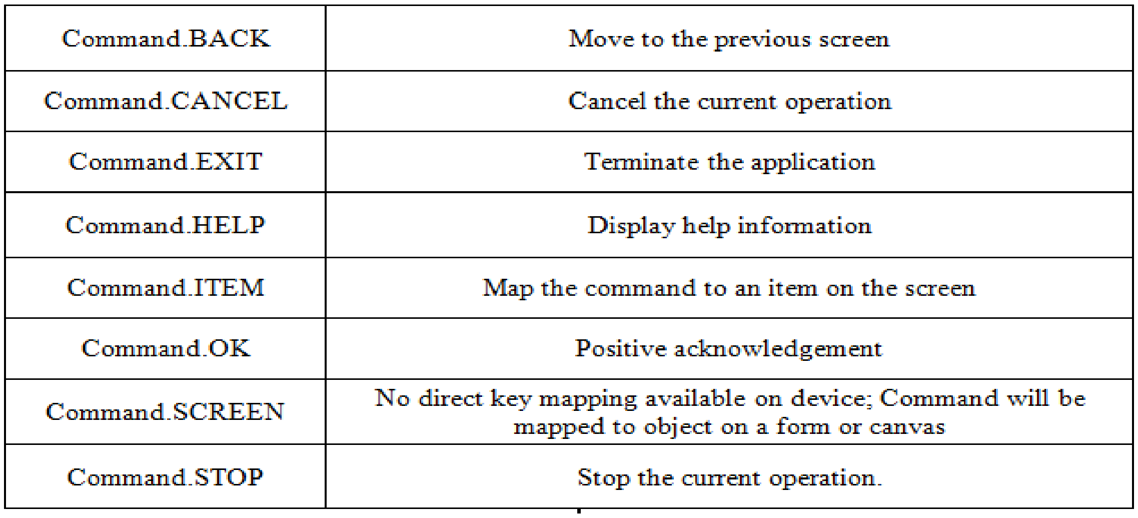
- Before we can run the Palm OS emulator in the J2ME Wireless Toolkit, we'll need to download Palm OS ROM files from the Palm web site (www.palmos.com/dev).
- The ROM file contains the Palm OS required for the emulator to properly perform like a Palm PDA.
- We'll also need to join the Palm OS Developer Program (free) and agree to the online license (free) for ROM files before you are permitted to download them.
- If our MIDlet is Palm device specific, we'll need to download the ROM file that corresponds to the Palm OS that runs on that Palm device.
- If we download the wrong ROM, because the Palm OS emulator displays an error when running your MIDlet, indicating the proper version of the Palm OS that is required to run your MIDlet on the Palm device that is being tested in the emulator.

Command Class:

- In contrast to desktop computers, which have plenty of screen space for displaying buttons or menus, a different approach is necessary for mobile devices.
- The **lcdui** package does not provide buttons or menu, but an abstraction called **"Command".**
- The Command class is used to set command buttons on the display screen. By clicking on these commands the applications will perform a pre-defined action.
- Commands can be added to all classes derived from the **Displayable** class. These classes are **Screen** and its subclasses such as **Form, List,** and **Textbox** for the high-level API and **Canvas** for the low-level API.
- No positioning or layout information is passed to the Command, the **Displayable** class itself is completely responsible for arranging the visible components corresponding to Commands on a device.
- We create an instance of the **Command** class by using the **Command** class constructor within the J2ME application.
- The Command class constructor requires three parameters. These are
  - command label
  - command type

- command priority

- The Command class constructor returns an instance of the Command class.
- The different predefined command types available in Command class are



Command.BACK	Move to the previous screen
Command.CANCEL	Cancel the current operation
Command.EXIT	Terminate the application
Command.HELP	Display help information
Command.ITEM	Map the command to an item on the screen
Command.OK	Positive acknowledgement
Command.SCREEN	No direct key mapping available on device; Command will be mapped to object on a form or canvas
Command.STOP	Stop the current operation.

- A command type is mapped to a key on the device's keypad, but the device does not process the command. When the user selects the command, the application manager detects the event and passes the selected command to application for processing.

CommandListener

- Every J2ME application that creates an instance of the Command class must also create an instance that implements the CommandListener interface.
- The CommandListener is notified whenever the user interacts with a command by way of the commandAction () method.
- Classes that implement the CommandListener must implement the commandAction () method, which accepts two parameters.
    - The first parameter is a reference to an instance of the Command class
    - Second parameter is a reference to the instance of the Displayable class
- The device's application manager calls the commandAction () method and passes the command selected by the user.
- The commandAction () method must contain all the processing that is to occur when the user selects a command.

Available Methods in Command Class:

Method	Description
Command(String label, int commandType, int priority)	Create an instance of the Command class that displays the specified label and is of the specified commandType and has the specified priority
addCommand(Command command)	Associate a command to an instance of the Displayable class
removecommand(Command command)	Disassociate a command from an instance of the Displayable class
setCommandListener(CommandListener commandlistener)	Associate a CommandListener to an instance of the Displayable class
boolean isShown()	Determine whether an instance of the Displayable class is shown on the screen
getCommandType()	Retrieve the commandType of a command
getLabel()	Retrieve the label of a command
getPriority()	Retrieve the priority of a command

Complete Program:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class OnlineHelp extends MIDlet implements CommandListener
{
        private Display display;
        private Command back;
        private Command exit;
        private Command help;
        private Form form;
        private TextBox helpMesg;

        public OnlineHelp()
        {
                display = Display.getDisplay(this);
                back = new Command("Back", Command.BACK, 2);
                exit = new Command("Exit", Command.EXIT, 1);
                help = new Command("Help", Command.HELP, 3);
                form = new Form("Online Help Example");
                helpMesg = new TextBox("Online Help", "Press Back to return
                to the previous screen or press Exit to close this program.", 81, 0);
```

```
		helpMesg.addCommand(back);
		form.addCommand(exit);
		form.addCommand(help);
		form.setCommandListener(this);
		helpMesg.setCommandListener(this);
	}
	public void startApp()
	{
		display.setCurrent(form);
	}
	public void pauseApp()
	{
	}
	public void destroyApp(boolean unconditional)
	{
	}
	public void commandAction(Command command, Displayable displayable)
	{
		if (command == back)
		{
			display.setCurrent(form);
		}
		else if (command == exit)
		{
			destroyApp(false);
			notifyDestroyed();
		}
		else if (command == help)
		{
			Displayable displayable)
			display.setCurrent(helpMesg);
		}
	}
}
```

Exception Handling:

- The small computing device's application manager controls the operation of a MIDlet.
- The application manager calls the **startApp()**, **pauseApp()**, and **destroyApp()** methods whenever the user or the device requires a MIDlet to begin, pause, or terminate.
- However, there are times when the interruption of processing by complying with the application manager's request might cause permanent harm.

- For example, a MIDlet might be in the middle of a communication session or saving persistent data when the **destroyApp()** method is called by the device's application manager, the request would break off communications or corrupt data.
- We can regain a little control of the MIDlet's operation by using a **MIDletStateChangeException**.
- A **MIDletStateChangeException** is used to temporarily reject a request from the application manager either to start the MIDlet (startApp()) or to destroy the MIDlet (destroyApp()).
- A **MIDletStateChangeException** cannot be thrown within the **pauseApp()** method.
- We should incorporate routines that throw a MIDletStateChangeException whenever MIDlet has processing that should not be interrupted by the application manager.

Example:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ThrowException extends MIDlet implements CommandListener
{
        private Display display;
        private Form form;
        private Command exit;
        private boolean isSafeToQuit;
        public ThrowException()
        {
                isSafeToQuit = false;
                display = Display.getDisplay(this);
                exit = new Command("Exit", Command.SCREEN, 1);
                form = new Form("Throw Exception");
                form.addCommand(exit);
                form.setCommandListener(this);
        }
        public void startApp()
        {
                display.setCurrent(form);
        }
        public void pauseApp()
        {
        }
public void destroyApp(boolean unconditional) throws MIDletStateChangeException
{
        if (unconditional == false)
        {
                throw new MIDletStateChangeException();
        }
}
public void commandAction(Command command,Displayable displayable)
{
        if (command == exit)
        {
                try
                {
                        if (exitFlag == false)
                        {
                        StringItem msg = new StringItem ("Busy", "Please try again.");
                        form.append(msg);
                        destroyApp(false);
                        }
                        else
                        {
                                destroyApp(true);
                                notifyDestroyed();
                        }
                }
                catch (MIDletStateChangeException exception)
                {
                        isSafeToQuit = true;
                }
        }
}
}
```

In this program the user to select the Exit command twice to terminate the MIDlet. When the user selects the Exit command the first time, the device's application manager calls the destroyApp() method where a MIDletStateChangeException is thrown, causing the message "Busy Please try again." to be displayed on the screen. The MIDlet successfully terminates the second time the user selects the Exit button.

High Level Display

- The display is a crucial component of every J2ME application since it contains objects used to present information to the user using the application and in many cases prompts the user to enter information that is processed by the application.
- The J2ME **Display** class is the parent of **Displayable** class.
- The **Displayable** class has two subclasses
  - ➢ **Screen**
  - ➢ **Canvas**
  - ▪ The **Screen class** is used to create **high-level J2ME displays** in which the methods of its subclasses handle details of drawing objects such as radio buttons and check boxes.
  - ▪ The **Canvas class** and its subclasses are used to create **low-level J2ME displays**. The methods give you pixel-level control of the display, enabling us to draw your own images and text such as those used to create games.

Screen class:

- The **Screen class** and its derived classes are used to create high-level J2ME displays.
- These classes contain methods that generate radio buttons, check boxes, lists, and other familiar objects that users expect to find on the screen when interacting with the application.

Display class hierarchy:

```
public class Display
    public abstract class Displayable
        public abstract class Screen extends Displayable
            public class Alert extends Screen
            public class Form extends Screen
            public class List extends Screen implements Choice
        public abstract class Item
            public class ChoiceGroup extends Item implements Choice
            public class DateField extends Item
            public class TextField extends Item
            public class Gauge extends Item
            public class ImageItem extends Item
            public class StringItem extends Item
        public class TextBox extends Screen
            public class Command
            public class Ticker
            public class Graphics
            public interface Choice
    public abstract class Canvas extends Displayable
        public class Graphics
```

- We already know that the Displayable class has two derived classes, **Screen** and **Canvas**.
- The **Screen** class has its own set of derived classes.
    - ➢ These are TextBox, List, Alert, Form, and Item classes.
    - ▪ The **TextBox** class is used to display multi-line text on the screen.
    - ▪ The **List** class is used to display a list of items, as a menu, and enables the user to choose one of those items.
    - ▪ The **Alert** class displays a dialog box containing a message such as a warning.
    - ▪ The **Form** class is a container class that can display multiple classes derived from the Item class.
    - ▪ The **Item** class has six derived classes, any number of which can be displayed within a Form object on the screen.
        - ➢ **ChoiceGroup** class used to display radio buttons and check boxes
        - ➢ **DateField** class used for inputting a date into an application
        - ➢ **TextField** class used for inputting text into an application
        - ➢ **Gauge** class used to graphically show progress
        - ➢ **ImageItem** class used to display an image stored in a file
        - ➢ **StringItem** class used to display text on the screen

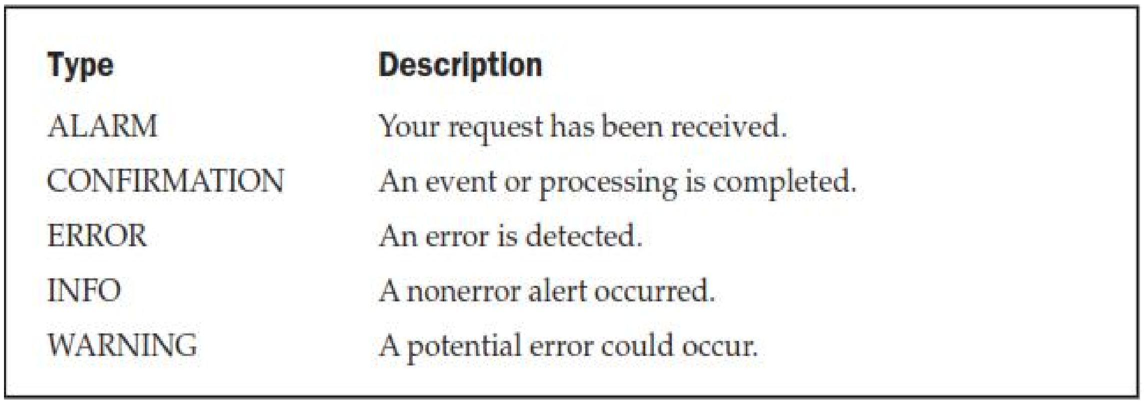
Alert Class:

- An alert is a dialog box displayed by the program to warn a user of a potential error such as a break in communication with a remote computer.
- An alert can also be used to display any kind of message on the screen, even if the message is not related to an error.

For example, an alert is an ideal way of displaying a reminder on the screen. We implement an alert by creating an instance of the Alert class in the program using the following statement. Once created, the instance is passed to the setCurrent () method of the Display object to display the alert dialog box on the screen.

```
alert = new Alert("Failure", "Lost communication link!", null, null);
display.setCurrent(alert);
```

- The Alert constructor requires four parameters.
  - ➢ The first parameter is the title of the dialog box, which is “Failure” in this example.
  - ➢ The next parameter is the text of the message displayed within the dialog box. “Lost communication link!” is the text that appears when the Failure dialog box is shown on the screen.
  - ➢ The third parameter is the image that appears within the dialog box. If we don’t use an image, the third parameter is set to null.
  - ➢ The last parameter is the AlertType. The AlertType is a predefined type of alert. If we don’t use any predefined AlertType, the fourth parameter is set to null.



Type	Description
ALARM	Your request has been received.
CONFIRMATION	An event or processing is completed.
ERROR	An error is detected.
INFO	A nonerror alert occurred.
WARNING	A potential error could occur.

An alert dialog box reacts in one of two ways depending on the value of the default timeout for the Alert object. The alert dialog box can remain visible until the user selects the OK button, or the alert dialog box can be visible for a specified number of milliseconds. An alert dialog box is referred to as a *modal* dialog box if the user must select the OK button to terminate the dialog box. Otherwise, it is considered a *timed* dialog box that terminates when the default timeout value is reached.

Methods in Alert Class:

setTimeout() method: The **setTimeout()** method determines whether an alert dialog box is a modal dialog box or a timed dialog box. The **setTimeout()** method has one parameter, which is the default timeout value. Use **Alert.FOREVER** as the default timeout value for a modal alert dialog box, or pass a time value in milliseconds indicating time to terminate the alert dialog box.

Example:

```
alert = new Alert("Failure", "Lost communication link!", null, null);
alert.setTimeout(Alert.FOREVER);
display.setCurrent(alert);
```

getDefaultTimeout() method: The **getDefaultTimeout()** method returns the integer value of Alert.FOREVER or the default timeout in milliseconds.

Alert Sound:

- Each **AlertType** has an associated sound that automatically plays whenever the alert dialog box appears on the screen. The sound, which is different for each AlertType, is used as an audio cue to indicate that an event is about to occur.
- An audio cue can be sounded without having to display the alert dialog box. We do this by calling the **playSound()** method and passing it reference to the instance of the Display class.

  Example:

```
if (exitFlag == false)
{
        AlertType.WARNING.playSound(display);
        destroyApp(false);
}
```

Form Class:

- A **Form** is a **Screen** that contains an arbitrary mixture of items: images, read-only text fields, editable text fields, editable date fields, gauges, choice groups, and custom items.
- In general, any subclass of the **Item** class may be contained within a form.
- Syntax to create an instance of the Form class is

  private Form form;
- An instance is placed with the instance of the Form class by calling one of two methods.

  These are **insert()** method and **append()** method.
  - ➢ The **insert()** method places the instance in a particular position on the form as specified by parameters passed to the insert() method.
  - ➢ The **append()** method places the instance after the last object on the form.

Example:

```
private Form form;
private StringItem message;
form = new Form("My Form");
message = new StringItem("Welcome, ", "glad you could come.");
form.append(message);
```

The above code segment illustrates how to create an instance of the **Form class** and call the **append()** method to place an instance of the **StringItem** class onto the form. After declaring referencing for the instance of the **Form class** and for the instance of the **StringItem** class, a **new Form** instance is created and given the title “My Form.” Next, a **StringItem** instance with the message “Welcome, glad you could come.” is created. The **append()** method is called once when both instances are created. Reference to the **StringItem** instance is then passed to the form, thereby placing the **StringItem** instance as the last object on the form.

Methods in Form Class:

Adding items to a Form:

- **append(Item item)** - Adds an Item into the Form.
- **insert(int itemNum, Item item) -** Inserts an item into the Form just prior to the item specified.

- **set(int itemNum, Item item)** - Sets the item referenced by itemNum to the specified item, replacing the previous item.

Adding strings and images to a Form:

- **append(String str)** - Adds an item consisting of one String to the Form.
- **append(Image img)** - Adds an item consisting of one Image to the Form.

Deleting items from a Form:

- **delete(int itemNum)** - Deletes the Item referenced by itemNum.
- **deleteAll()** - Deletes all the items from this Form, leaving it with zero items.

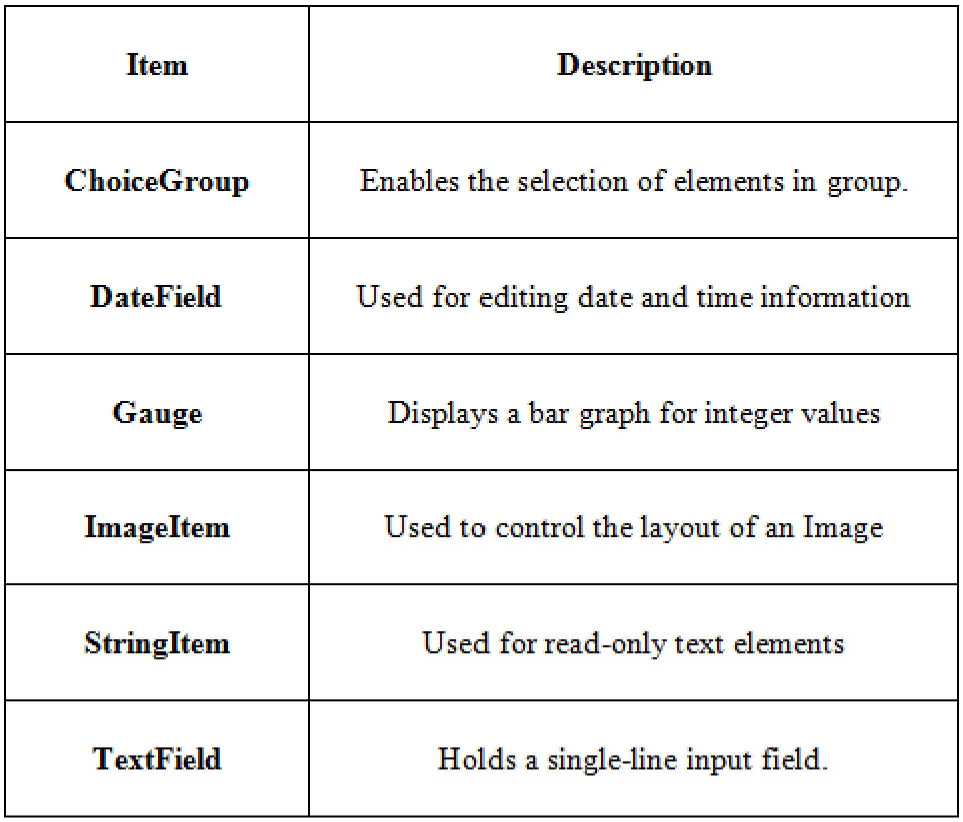
The getter methods:

- **get(int itemNum)** - Gets the item at given position.
- **getHeight()** - Returns the height in pixels of the displayable area available for items.
- **getWidth()** - Returns the width in pixels of the displayable area available for items.
- **size()** - Gets the number of items in the Form.4

<u>Item Class:</u>

- The **Item** class is derived from the Form class, and that gives an instance of the Form class character and functionality by implementing **text fields, images, date fields, radio buttons, check boxes**, and other features common to most graphical user interfaces.
- The Item class itself is an abstract base class that cannot be instantiated.
- Items can neither be placed freely nor can their size be set explicitly. Unfortunately, it is not possible to implement Item subclasses with a custom appearance. The Form handles layout and scrolling automatically.

Subclasses of Item:



Item	Description
ChoiceGroup	Enables the selection of elements in group.
DateField	Used for editing date and time information
Gauge	Displays a bar graph for integer values
ImageItem	Used to control the layout of an Image
StringItem	Used for read-only text elements
TextField	Holds a single-line input field.

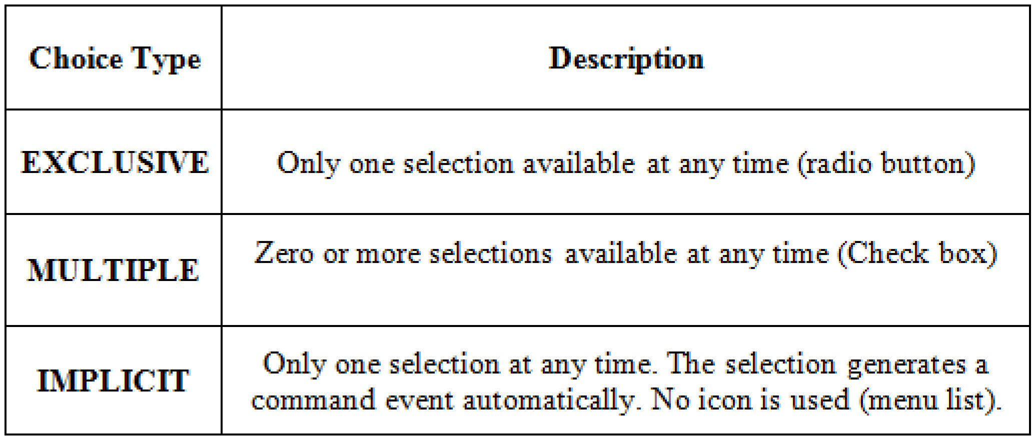
Item Listener:

- Each MIDlet that utilizes instances of the Item class within a form must have an **itemStateChanged()** method to handle state changes in these instances.
- The **itemStateChanged()** method contains one parameter, which is an instance of the Item class.
    - ➢ The instance passed to the **itemStateChanged()** method is the instance whose state was changed by the user.
- The **itemStateChanged()** method would have appropriate logic to process a change in each instance rather than displaying a statement at the command line.

<u>ChoiceGroup Class:</u>

- The ChoiceGroup is an MIDP UI widget enabling the user to choose between different elements in a Form.
- J2ME classifies check boxes and radio buttons as the ChoiceGroup class.

- The primary difference between a set of check boxes and a set of radio buttons, besides their obvious appearance, is the number of check boxes or radio buttons that users can select. Users can choose multiple check boxes within a set of check boxes, while they can choose only one radio button within a set of radio buttons.
- An instance of the ChoiceGroup class can be one of two types: exclusive or multiple.
  - ➢ An **exclusive** instance appears as a set of radio buttons, and a **multiple** instance contains one or a set of check boxes.

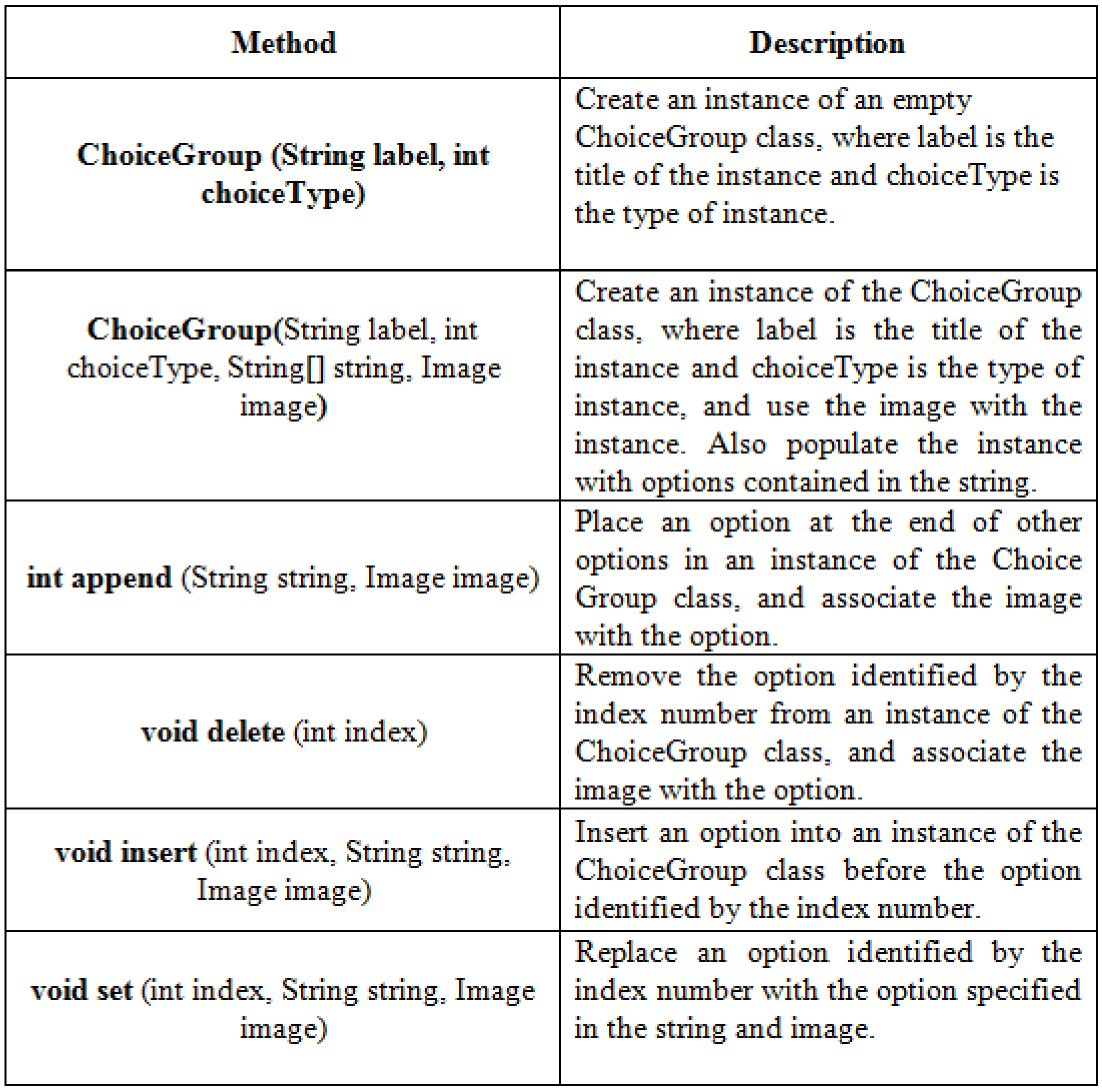


Choice Type	Description
EXCLUSIVE	Only one selection available at any time (radio button)
MULTIPLE	Zero or more selections available at any time (Check box)
IMPLICIT	Only one selection at any time. The selection generates a command event automatically. No icon is used (menu list).

Choice Types for ChoiceGroup Object and List Object

- When the user selects either a radio button or check box, the device's application manager detects the event and calls the **itemStateChanged()** method of the MIDlet.
- The **itemStateChanged()** method determines whether the item selected is an instance of the ChoiceGroup. If so, then either the **getSelectedFlags()** method or **getSelectedIndex()** method must be called to retrieve the item selected by the user.
  - ➢ The **getSelectedFlags()** method returns an array that contains the status of the selected flag for each member of the instance of the ChoiceGroup class. The MIDlet must step through each element of the array to determine whether the selected flag status is true or false. If true, the radio button or check box that corresponds to the index of the array element was selected by the user. If false, the user did not make a selection.
  - ➢ The **getSelectedIndex()** method returns the index number of the item selected by the user, such as a radio button. The index number is typically passed to the

getString() method, which returns the text of the selected radio button or check box.



Method	Description
ChoiceGroup (String label, int choiceType)	Create an instance of an empty ChoiceGroup class, where label is the title of the instance and choiceType is the type of instance.
ChoiceGroup(String label, int choiceType, String[] string, Image image)	Create an instance of the ChoiceGroup class, where label is the title of the instance and choiceType is the type of instance, and use the image with the instance. Also populate the instance with options contained in the string.
int append (String string, Image image)	Place an option at the end of other options in an instance of the Choice Group class, and associate the image with the option.
void delete (int index)	Remove the option identified by the index number from an instance of the ChoiceGroup class, and associate the image with the option.
void insert (int index, String string, Image image)	Insert an option into an instance of the ChoiceGroup class before the option identified by the index number.
void set (int index, String string, Image image)	Replace an option identified by the index number with the option specified in the string and image.

String getString (int index)	Retrieve the string associated with the option identified by the index number.
Image getImage(int index)	Retrieve the image associated with the option identified by the index number.
int getSelectedIndex()	Retrieve the index associated with an option.
void setSelectedIndex(int index, boolean selected)	Select the option identified by the index and whether the option is selected (true) or unselected (false).
int getSelectedFlags (boolean[] array)	Retrieve the selection status of options and store them in an array.
void setSelectedFlag (boolean[] array)	Set the selection status of options stored in an array
boolean isSelected (int index)	Determine whether the user selected the option identified by the index number.
int size()	Determine the number of options there are in an instance of the ChoiceGroup.

DateField Class:

- The DateField class is used to display, edit, or input date and/or time into a MIDlet.
- A DateField class is instantiated by specifying a label for the field, a field mode, and a time zone (time zone is optional).

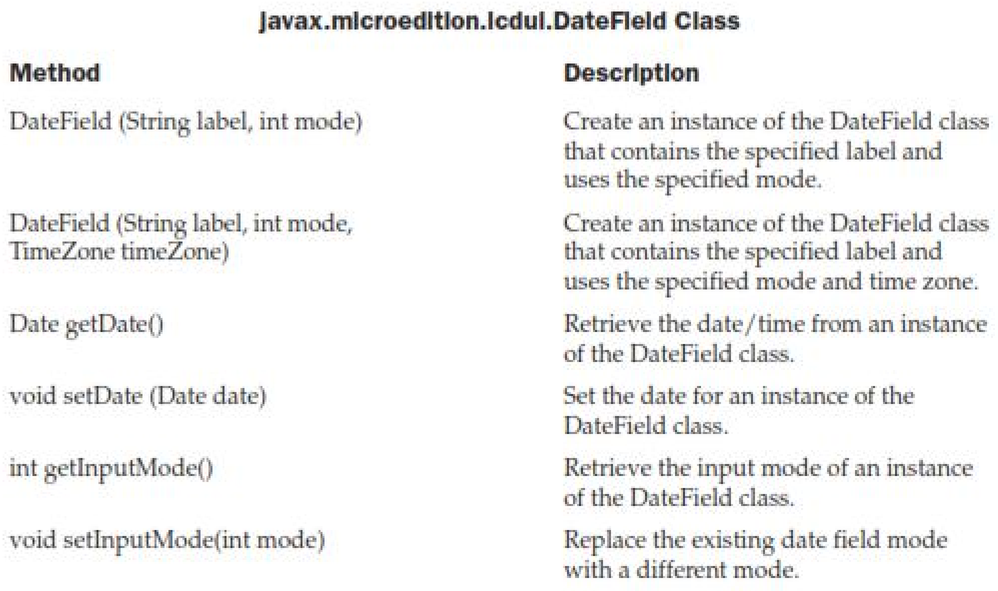
  DateField datefield = new DateField("Today", DateField.DATE);

  DateField datefield = new DateField("Time", DateField.TIME, timeZone);
- Once a DateField class is instantiated, we can use DateField class methods to enter a date and time into the date field and retrieve the date and time value that has already been entered into the date field.

Mode	Description
DATE	Display, edit, and input a date
TIME	Display, edit, and input a time
DATE_TIME	Display, edit, and input both date and time

DateFiled Modes

DateField Methods:



javax.microedition.lcdui.DateField Class

Method	Description
DateField (String label, int mode)	Create an instance of the DateField class that contains the specified label and uses the specified mode.
DateField (String label, int mode, TimeZone timeZone)	Create an instance of the DateField class that contains the specified label and uses the specified mode and time zone.
Date getDate()	Retrieve the date/time from an instance of the DateField class.
void setDate (Date date)	Set the date for an instance of the DateField class.
int getInputMode()	Retrieve the input mode of an instance of the DateField class.
void setInputMode(int mode)	Replace the existing date field mode with a different mode.

Example:

```
import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class DateToday extends MIDlet implements CommandListener
{
        private Display display;
        private Form form;
        private Date today;
        private Command exit;
        private DateField datefield;
        public DateToday()
        {
                display = Display.getDisplay(this);
                form = new Form("Today's Date");
                today = new Date(System.currentTimeMillis());
                datefield = new DateField("", DateField.DATE_TIME);
                datefield.setDate(today);
                exit = new Command("Exit", Command.EXIT, 1);
                form.append(datefield);
                form.addCommand(exit);
                form.setCommandListener(this);
        }
        public void startApp ()
        {
                display.setCurrent(form);
        }
        public void pauseApp()
        {
        }
        public void destroyApp(boolean unconditional)
        {
        }
        public void commandAction(Command command, Displayable displayable)
        {
            if (command == exit)
            {
                destroyApp(false);
                notifyDestroyed();
            }
        }
}
```

Gauge Class:

- The Gauge class creates an animated progress bar that graphically represents the status of a process.
- It is initialized with a label, a flag indicating whether it is interactive, and a maximum and an initial value.
- If a Gauge is interactive, the user is allowed to change the value using a device-dependent input method.
- Creating an instance of the Gauge class

  Gauge gauge = new Gauge ("Like/Dislike Gauge", true, 100, 0);

  This statement creates an interactive gauge with the caption “Like/Dislike Gauge” and a scale of zero to 100.

  - The first parameter passed to the constructor of the Gauge class is a string containing the caption that is displayed with the gauge.
  - The second parameter is a Boolean value indicating whether or not the gauge is interactive.
  - The third parameter is the maximum value of the gauge, and the last parameter is the gauge’s initial value.

Methods in Gauge Class:

Method	Description
Gauge(String label, boolean interactive, int maxValue, int initialValue)	Create an instance of the Gauge class, where label is the caption for the instance, interactive is a boolean value indicating whether the instance is interactive, maxValue is the maximum value displayed in the gauge, and the initialValue is the beginning value displayed in the gauge.
int getValue()	Retrieve the current value of the gauge.
void setValue (int value)	Set a new value for the gauge.
int getMaxValue()	Retrieve the maximum value of the gauge.
void setMaxValue(int maxValue)	Set the maximum value of the gauge.
boolean isInteractive()	Determine whether the gauge is interactive.

StringItem Class:

- The purpose of using a StringItem class is to display a text that cannot be modified or deleted by the user of the MIDlet.
- A StringItem class is different from other classes derived from the Item class in that a StringItem class does not recognize events.
- This means that an instance of a StringItem class can never cause an event because the user cannot modify the text of the StringItem.
- We create an instance of a StringItem class by passing the StringItem class constructor two parameters.
    - ➢ The first parameter is a string representing the label of the instance.
    - ➢ The other parameter is a string of text that will appear on the screen.

Methods used in StringItem class:

- **getText():**
    - ➢ We can retrieve the text of the instance of a StringItem class by calling getText () method.
- **setText():**
    - ➢ We can replace the text by calling setText(0 method.
    - ➢ The setText() method require one parameter, which is the new text that replaces the current text of instance.
- **setLabel():**
    - ➢ The label of the instance can be changed by calling the setLabel() method.
    - ➢ The setLabel() method require one parameter, which is replacement of the label.
- **getLabel():**
    - ➢ The getLabel() method returns a String consisting of the label of the instance.

ImageItem Class:

- There are two types of images that can be displayed. These are immutable images and mutable images.
- An immutable image is loaded from a file or other resource and cannot be modified once the image is displayed.
- A mutable image is drawn on the screen using methods available in the Graphics class.
- An immutable image is drawn on a screen, and a mutable object is drawn on a canvass.
- Mutable images are displayed using the Graphics class, which is derived from the Canvas class.
- The first step in displaying an immutable image is to create an instance of the Image class by calling the **createImage()** method.
- The createImage() method requires one parameter that contains the name of the file containing the image.(Make sure that you include the full path to the file in the parameter.)
- The next step is to create an instance of the ImageItem class.
- The constructor of the ImageItem class requires four parameters.

- ➢ The first is a string that becomes the label for the image.
- ➢ The second parameter is reference to the instance of the Image class created in step one.
- ➢ The third parameter is the layout directive.
- ➢ And the last parameter is a string referred to as alternate text that is displayed in place of the image if for some reason the image cannot be displayed by the device.

- Some applications won't require you to specify a label or alternate text; therefore, use a null as the value of the parameter in place of a string.

Value	Description
LAYOUT_DEFAULT	Use the device's default layout
LAYOUT_LEFT	Place image left
LAYOUT_RIGHT	Place image right
LAYOUT_CENTER	Center image
LAYOUT_NEWLINE_BEFORE	Start a new line and then draw the image
LAYOUT_NEWLINE_AFTER	Draw the image and then start a new line

Image Layout Directives

- The layout directive is a request to the device's application manager to position the image at a particular location on the screen. The device's application manager determines the actual location where the image appears.

Methods used in ImageItem class:

setLabel ():

- It replaces the current Label, with new Label which is passed as parameter.

getLabel():

- It returns the current Label of the ImageItem class instance.

setLayout():

- The setLayout() method replaces the current layout with a new layout whose directive is passed as a parameter to the setLayout() method.

getLayout():

- The getLayout () method returns the current layout directive of an instance of an ImageItem.

setImage():

- It replaces the instance of the Image class, with new instance of the Image class which is passed as parameter.

getImage():

- The getImage() method fetches the current image associated with the instance of the ImageItem

setAltText():

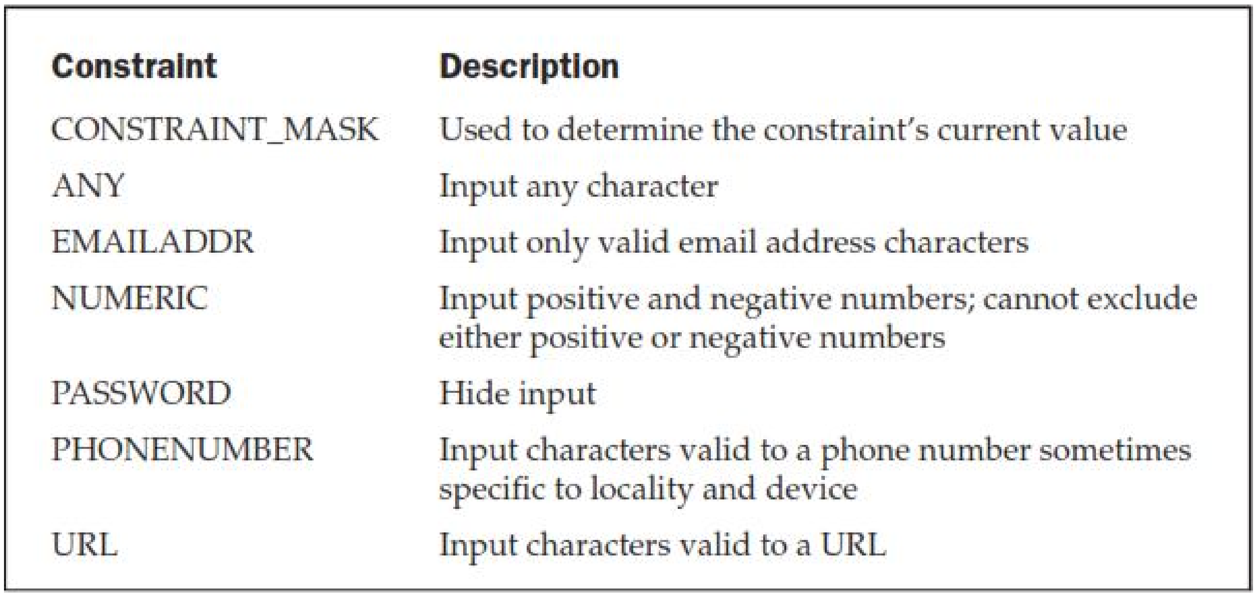
- It replaces the Alternative text with new text, which is passed as parameter.

getAltText():

- It returns Alternative text of the ImageItem instance.

TextField Class:

- The TextField class is used to capture one line or multiple lines of text entered by the user. The number of lines of a text field depends on the maximum size of the text field when you create an instance of the TextField class.
- The instance of the TextField class is created using TextField constructor.
  Textfield_instance=new TextField("1st parameter","2nd parameter","3rd parameter","4th parameter")
- The constructor requires four parameters
  - ➢ The first parameter is the label that appears when the instance is displayed on the screen.
  - ➢ The second parameter is text that you want to appear as the default text for the instance, which the user can edit.
  - ➢ The third parameter is the maximum number of characters that can be held by the instance.
  - ➢ The last parameter passed to the constructor of the TextField class is the constraint (if any) that is used to restrict the type of characters that the user can enter into the text field.



Constraint	Description
CONSTRAINT_MASK	Used to determine the constraint's current value
ANY	Input any character
EMAILADDR	Input only valid email address characters
NUMERIC	Input positive and negative numbers; cannot exclude either positive or negative numbers
PASSWORD	Hide input
PHONENUMBER	Input characters valid to a phone number sometimes specific to locality and device
URL	Input characters valid to a URL

Methods used TextField class:

getString():

- The getString() method returns the content of the text field as a string

getChars():

- the getChars() method returns the text field content as a character array.
- The getChars() method requires that you pass it a character array as a parameter.

setString():

- We can place text into a text field by calling either the setString() method
- The setString() method requires one parameter, which is the string containing text that should appear in the text field.

setChars():

- We can place text into a text field by calling either the setString() method
- The setChars() method requires three parameters.
    - The first is the character array whose data will populate the text field.
    - The second is the position of the first character within the array that will be placed into the text field.
    - The last parameter is the length of characters of the character array that will be placed into the text field.

Insert():

- We can insert characters within the text field without overwriting the entire content of the text field by calling the **insert()** method.
- The insert() method has two signatures, one for strings and the other for character arrays.
- The insert() method used to insert a string into the contents of a text field requires two parameters.
  - The first parameter is the string that will be inserted into the text field.
  - The other parameter is the character position of the current string where the new text is inserted.
- The insert() method used to insert a character array requires four parameters.
  - The first parameter is reference to the array.
  - The second parameter is the position of the first character within the array that will be placed into the text field.
  - The third parameter is the number of characters contained in the array that will be placed into the text field.
  - The last parameter is the character position of the current text that will be shifted down to make room for the inserted text.

delete():

- Text can be removed from the text field by calling the delete() method, which requires two parameters. The first is the position of the first character to be deleted. The other parameter is the length of characters that are to be deleted.

setConstraints():

- The constraint of a text field can be changed after the instance is created by calling the setConstraints() method. The setConstraints() method requires you to pass the new constraint as a parameter to the setConstraints() method.

getConstraints():

- We can determine the current constraint by calling the getConstraints() method.

setMaxSize():

- We can also change the maximum size by calling the setMaxSize() method.

- ➢ The setMaxSize() method requires one parameter, which is the new value for the maximum size for the text field.

size ():

- ➢ If we need to know the length of the text in the text field you can call the size() method, which returns an integer representing the number of characters existing in the text field.

<u>List Class:</u>

- The List class is used to display a list of items on the screen from which the user can select one or multiple items.
- There are three formats for the List class:
  - ➢ radio buttons
  - ➢ check boxes
  - ➢ an implicit list that does not use a radio button or check box icon
- List class is functionally similar to the ChoiceGroup class, but it differs from the ChoiceGroup class by the way events of each instance are handled by a MIDlet.
- An **ItemStateListener** is used to listen to events generated by an instance of a ChoiceGroup class. Those events are then passed along to the **itemStateChanged()** method for processing.
- In contrast, a list does not generate an item state change event therefore; a Command needs to be added to initiate processing.
- A List class is derived from the Screen class and does not require a container. We can create an instance of the List class with or without list items.
- An instance is created without list items by passing the constructor of the List class two parameters.
  - ➢ The first parameter is a string that contains the titles of the list
  - ➢ other parameter is the format of the list
- We can include list items when creating the instance of a List class by passing two additional parameters to the List class constructor.
  - ➢ The first two parameters are title and list Type.
  - ➢ The third parameter is a string array whose elements contain list items that can be selected by the user of your MIDlet.
  - ➢ The fourth parameter is an array of instances of the Image class, each associated with a corresponding list item.
- List items can be added to an instance of a List object by calling the **append()** method or **insert()** method.
- The **append()** method requires two parameters.
  - ➢ The first parameter is the string that contains the new list item,

- ➢ The second parameter is an instance of the Image class of an image that is associated with the new list item.
- ➢ The new list item is appended to the end of the list.

Methods in List Class:

- **getSelectedIndex():**

  The getSelectedIndex() method returns the index number of the selected list item.

- **getSelectedFlag():**

  If the instance of the List class is a check box, then call the getSelectedFlag() method. The getSelectedFlag() method requires one parameter, which is a boolean array.

- **getSelectedIndex();**

  The setSelectedIndex() sets the selected status for all list items. The setSelectedIndex() requires one parameter, which is a boolean array containing the selected status for the entire list.

- **setSelectedFlags():**

  The setSelectedFlags() method is used to set the selected flag of one list item and requires two parameters. The first parameter is the index number of the list item being selected, and the other parameter is a boolean value, where true signifies that the list item is selected and false signifies unselected.

- **set() method:**

  Any list item can be replaced by calling the set() method. The set() method requires three parameters.

  - ➢ The first is the index number of the list item being replaced.
  - ➢ The second parameter is the string replacing the string of the specified list item.
  - ➢ The last parameter is an Image object that contains the image associated with the replacement list item.

- **delete() method:**

  A list item can be removed from the list by calling the delete() method. The delete() method requires one parameter, which is the index number of the list item being deleted.

List implicit Example:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ListImplicit extends MIDlet implements CommandListener
{
private Display display;
private List list;
private Command exit;
Alert alert;
public ListImplicit()
{
        display = Display.getDisplay(this);
        exit = new Command("Exit", Command.EXIT, 1);
        list = new List("Menu:", List.IMPLICIT);
        list.append("New",null);
        list.append("Open",null);
        list.addCommand(exit);
        list.setCommandListener(this);
}
}
public void startApp()
{
        display.setCurrent(list);
}
public void pauseApp()
{
        }
public void destroyApp(boolean unconditional)
{
}
public void commandAction(Command command, Displayable displayable)
{
        if (command == List.SELECT_COMMAND)
        {
                String selection = list.getString(list.getSelectedIndex());
                alert = new Alert("Option Selected", selection, null, null);
                alert.setTimeout(Alert.FOREVER);
                alert.setType(AlertType.INFO);
                display.setCurrent(alert);
        }
        else if (command == exit)
        {

                destroyApp(false);
                notifyDestroyed();
        }
}
}
```

Check-Box Formatted List Class:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ListCheckBox extends MIDlet implements CommandListener
{
        private Display display;
        private Command exit;
        private Command submit;
        private List list;
        public ListCheckBox()
        {
                display = Display.getDisplay(this);
                list = new List("Select Media", List.MULTIPLE);
                list.append("Books", null);
                list.append("Movies", null);
                list.append("Television", null);
                list.append("Radio", null);
                exit = new Command("Exit", Command.EXIT, 1);
                submit = new Command("Submit", Command.SCREEN,2);
                list.addCommand(exit);
                list.addCommand(submit);
                list.setCommandListener(this);
        }
        public void startApp()
        {
                display.setCurrent(list);
        }
        public void pauseApp()
        {
        }
        public void destroyApp(boolean unconditional)
        {
        }
        public void commandAction(Command command, Displayable Displayable)
        {
                if (command == submit)
                {
                        boolean choice[] = new boolean[list.size()];
                        StringBuffer message = new StringBuffer();
                        list.getSelectedFlags(choice);
                        for(int x=0;x<choice.length; x++)
                        {
                                if (choice[x])
                                {
                                        message.append(list.getString(x));
                                        message.append(" ");
                                }
                        }
                Alert alert = new Alert("Choice", message.toString(),null, null);
                        alert.setTimeout(Alert.FOREVER);
                        alert.setType(AlertType.INFO);
                        display.setCurrent(alert);
                        list.removeCommand(submit);
                }
                else if (command == exit)
                {
                        destroyApp(false);
                        notifyDestroyed();
                }
        }
}
}
```

An alert is displayed on the screen containing the text of each item selected by the user. The Submit command is then removed from the screen, leaving the user to close the alert dialog box and then terminate the MIDlet by selecting the Exit command.

<u>Radio Button–Formatted List Class:</u>

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class ListRadioButtons extends MIDlet implements CommandListener
{
        private Display display;
        private Command exit;
        private Command submit;
        private List list;
        public ListRadioButtons()
        {
                display = Display.getDisplay(this);
                list = new List("Select one", List.EXCLUSIVE);
                list.append("Male", null);
                list.append("Female", null);
                exit = new Command("Exit", Command.EXIT, 1);
                submit = new Command("Submit", Command.SCREEN,2);
                list.addCommand(exit);
                list.addCommand(submit);
                list.setCommandListener(this);
        }
        public void startApp()
        {
                display.setCurrent(list);
        }
        public void pauseApp()
        {
        }
        public void destroyApp(boolean unconditional)
        {
        }
        public void commandAction(Command command, Displayable Displayable)
        {
                if (command == submit)
                {
        Alert alert = new Alert("Choice",list.getString(list.getSelectedIndex()),null,
null);
                        alert.setTimeout(Alert.FOREVER);
                        alert.setType(AlertType.INFO);
                        display.setCurrent(alert);
                        list.removeCommand(submit);
                }
                else if (command == exit)
                {
                        destroyApp(false);
                        notifyDestroyed();
                }
        }
}
```

TextBox Class:

- The TextBox class is very similar to a TextField class; both are used to receive multiple lines of textual data from a user.
- The TextBox class and TextField class differ in that the TextBox class is derived from the Screen class, while the TextField class is derived from the Item class.
    - This means that an instance of the Form class cannot contain an instance of the TextBox class, while an instance of a TextField class must be contained within an instance of the Form class.
- Another important difference between the TextBox class and the TextField class is that the TextBox class uses a CommandListener and cannot use an ItemStateListener. An ItemStateListener is used with an instance of the TextField class.
- An instance of the TextBox class is created by passing four parameters to the TextBox class constructor.
    - The first parameter is the title of the text box.
    - The second parameter is text used to populate the instance.
    - The third parameter is the maximum number of characters that can be entered into the instance.
        - This parameter is a request and may not be fulfilled by the device.
        - The device determines the maximum number of characters for an instance of the TextBox class.
    - The last parameter is the constraint used to limit the types of characters that can be placed within the instance.

Constraint	**Description**
CONSTRAINT_MASK	Used to determine the constraint's current value
ANY	Input any character
EMAILADDR	Input only valid email address characters
NUMERIC	Input positive and negative numbers; cannot exclude either positive or negative numbers
PASSWORD	Hide input
PHONENUMBER	Input characters valid to a phone number sometimes specific to locality and device
URL	Input characters valid to a URL

TextBox Constraints

Example:

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
public class TextBoxCapture extends MIDlet implements CommandListener
{
        private Display display;
        private TextBox textbox;
        private Command submit;
        private Command exit;
        public TextBoxCapture()
        {
                display = Display.getDisplay(this);
                submit = new Command("Submit", Command.SCREEN, 1);
                exit = new Command("Exit", Command.EXIT, 1);
                textbox = new TextBox("First Name:", "", 30, TextField.ANY);
                textbox.addCommand(exit);
                textbox.addCommand(submit);
                textbox.setCommandListener(this);
        }
        public void startApp()
        {
                display.setCurrent(textbox);
        }
        public void pauseApp()
        {
        }
        public void destroyApp(boolean unconditional)
        {
        }
        public void commandAction(Command command, Displayable displayable)
        {
                if (command == submit)
                {
                        textbox.setString("Hello, " + textbox.getString());
                        textbox.removeCommand(submit);
                }
```

```
                    else if (command == exit)
                    {
                              destroyApp(false);
                              notifyDestroyed();
                    }
          }
}
```

Ticker class:

- The Ticker class is used to scroll text horizontally on the screen.
- An instance of the Ticker class can be associated with any class derived from the Screen class and be shared among screens.
- An instance of the Ticker class can be associated with any class derived from the Screen class and be shared among screens.
- An instance of a Ticker object is created by passing the constructor of the Ticker class a string containing the text that is to be scrolled across the screen.
- We cannot control the location on the screen where scrolling occurs. Likewise, there is no control over the speed of the scrolling.

Methods:

getString ():

- We can retrieve the text associated with an instance of the Ticker class by calling the getString () method.

setString ():

- We can replace the text currently scrolling across the screen by calling the **setString ()** method.
- The **setString ()** method requires one parameter, which is a string containing the replacement text.

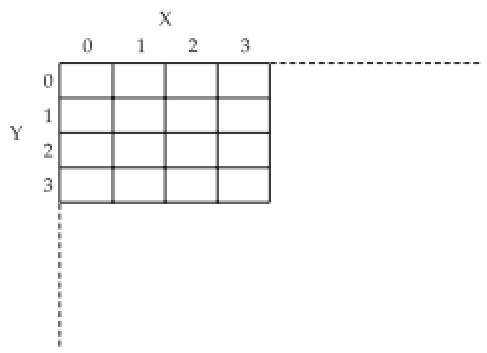
Low-Level Display

Canvas class

- The Displayable class has two subclasses**: Screen and Canvas**.
- The Canvas class and its derivatives are used to gain low-level access to the display, which is necessary for graphic- and animation-based applications. A graphic is used with a canvas.
- An instance of the Graphics class is similar to the artist's tools that are used to draw an image.
- The Canvas class and the Graphics class give you pixel control over everything that appears on the canvas.
- If we want to display text using the high-level user interface, first, you create an instance of a text field, text box, or string item and then associate text with the instance. Next, the **setCurrent()** is called and passed the instance (or a container such as a form that contains the instance). We are not concerned about describing how the device's application manager forms each character of the text on the screen.
- But **displaying text using the Graphics class** requires you to **specify the height, width, and other characteristics** that describe how each character of the text is to be drawn on the screen.

The Layout of a Canvas:

- The canvas is divided into a virtual grid in which each cell represents one pixel.
- Coordinates mark the column and row of a cell within the grid .The **x coordinate represents the column, and the y coordinate represents the cell's row**. The first cell located in the upper-left corner of the grid has the coordinate location of 0, 0, where the first zero is the x coordinate and the other zero is the y coordinate.



- The size of the canvas is **device dependent**, since canvas size and the screen size are the same.
- The screen size of a mobile telephone might be different from the screen size of a PDA, but both devices are capable of running the same MIDlet.
- Our MIDlet should determine the canvas size of the device that implements our graphic application before drawing on the screen. The canvas size is measured in pixels.
- The MIDlet should determine the canvas size of the device by calling the following methods of the Canvas class:
    - **getWidth()**
    - **getHeight()**

Proportional Coordinates:

- The values (in pixels) returned by the getWidth() and getHeight() methods can be used to draw an image at a given location that is proportional to the size of the canvas by using relative coordinates rather than exact coordinates on the canvas.
- If we want to draw an image on the canvas in the **centre of the canvas**, we need to calculate the centre .This depends on the canvas size, which depends on the small computing device that runs our MIDlet. The MIDlet calculates the centre coordinate based on the return value of the getWidth() and getHeight() methods.

x = getWidth()/2

y = getHeight()/2

- Suppose if a MIDlet is running on a small computing device with a canvas size of 400 pixels wide by 400 pixels high. The calculation determines the center coordinate as 199, 199.
- The MIDlet can use the calculation to determine the center coordinate of any size canvas, which means that the image will appear in the same canvas location when the MIDlet runs on any device. Thus, calculating a specific coordinate rather than specifying a fixed coordinate solves one problem faced by a developer of a J2ME application.

- Another problem is **scaling an image to fit a canvas size** that is device dependent. If you know the size of the canvas, you could plot each pixel that is required to draw an image. The image will be symmetrical within the screen. However, the symmetry is disrupted when the size of the canvas changes and the image size remains the same.
- So we need to use **relative coordinates** to draw an image rather than specific coordinates .The MIDlet must calculate the specific coordinate of each element of your image.

The Pen:

- An image is drawn on a canvas using a virtual pen.
- The pen is dropped on the canvas at a specified coordinate, filling the cell with the colour of ink used in the pen.
- Cells change from their present colour to the colour of the ink as the pen is repositioned on the canvas.
- For example, a horizontal line forms on the canvas when the virtual pen is dragged horizontally across the canvas. Dragging the virtual pen vertically down the canvas draws a vertical line.
- A virtual pen is used by instances of the Graphics class to draw rectangles, arcs, and other graphical image components on the canvas. You don't directly create and use a virtual pen.

Painting:

- Graphical components used to create an image on a canvas are drawn on the canvas when the **paint() method of the Displayable class** is called. This is referred to as **painting**.
- The paint() method is an abstract method that is used both by instances and derivatives of the Screen class and Canvas class.
- The contents of the paint() method are statements that draw images on the screen.
- Derivatives from the Screen class have **two predefined methods** used to paint the screen:
    - ➢ **paint ():** which contains instructions that set parameters for drawing an image, such as defining the virtual pen.
    - ➢ **paintContent():** which is called at the end of the paint() method and contains statements to actually draw the image.

The paint() Method:

- We cannot call the **paint() method** directly. Instead, the **paint() method is called automatically by the setCurrent()** method when the MIDlet is started. We call the repaint() whenever the canvas or a portion of the canvas must be refreshed.
- The paint() method requires one parameter, which is reference to the instance of the Graphics class. For example-a paint() method that draws a rectangle on the canvas.

```
protected void paint(Graphics graphics)

{

        graphics.drawRect(12, 6, 40, 20));

}
```

The repaint() Method:

- There are two versions of the **repaint() method**.
    - ➢ One version requires no parameters and repaints the entire canvas.
    - ➢ The other version requires four parameters that define the region of the canvas that is to be repainted. The first two parameters are the x and y coordinates for the upper-left corner of the region, and the last two parameters are the width and height of the region.
- We specify a region of the canvas to repaint whenever only a portion of the canvas has changed and when you don't want to waste time repainting the entire canvas, such as when an animated image is displayed on the screen. This is known as **clipping.**
- **Animation** is the illusion of movement caused by rapidly changing images on the screen, where each image is slightly different from the previous image. Each image displayed on the screen is referred to as a **frame**. A key to successful animation is **speed**. We must change frames in such a way that users don't notice the change.
- If a small portion of a frame changes in an animated image, the repaint() method is capable of repainting only the portion of the frame that changed rather than the entire frame, which dramatically reduces the time that is necessary to change a frame on the screen.

The servicePaint() Method:

- . A paint request is one of many requests a MIDlet can make to the application manager of a small computing device. Other requests can be made to store data or to communicate with a remote computer.
- Sometimes outstanding requests can be given a higher process priority by the device's application manager than a paint request. We need to override outstanding requests to have the canvas repainted whenever an image is being animated; otherwise, a delay in repainting the canvas destroys the effect of animation.
- The **serviceRepaints()** method directs the device's application manager to override outstanding requests for service with the repaint request. The repaint request becomes the next request to be processed by the application manager.

showNotify() and hideNotify():

- **showNotify() method**: called by the application manager immediately before the application manager displays the canvas. It consists of statements that prepare the canvas for display, such as initializing resources by beginning threads or assigning values to variables as required by the application.
- **hideNotify() method:** is called by the application manager after the canvas is removed from the screen. It consists of statements that free resources that were allocated when the showNotify() method was called. This includes deactivating threads and resetting values assigned to variables as necessary.

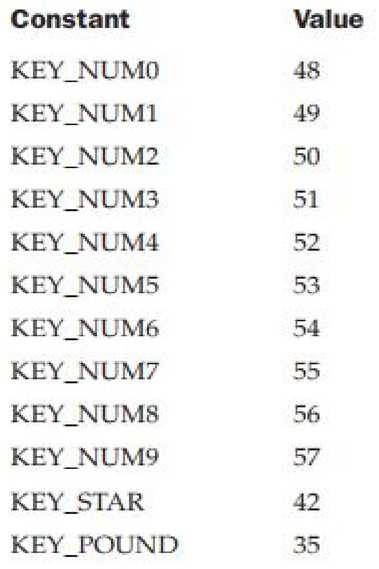
User Interactions

- Two techniques can be used to receive user input into your low-level J2ME application:
  - ➢ **To create one or more instances of the Command class.** Once an instance of a command is created, the instance is associated with the instance of the Canvas class by calling the addCommand() method. If we associate a command with a canvas, we need to associate a CommandListener to the canvas in order to monitor command events generated by the user selecting a command. So we need to define a commandAction() method that is called by the device's application manager to process the command event.
  - ➢ **To use low-level user input components that generate low-level user events**. These components are **key codes, game actions, and pointers.**

- A **key code** is a numerical value sent by the small computing device when the user of your application selects a particular key. Each key on the device's keypad is identified by a unique key code.
- A **game action** is a keystroke that a person uses to play a game on the small computing device. MIDP defines a set of constants that represent keystrokes common to game controllers.
- A **pointer** event is input received from a pointer device attached to the small computing device, such as a touch screen or mouse

Working with Key Codes:

- Each key on keypad, which is used on cellular telephones, is mapped to a standard set of key codes. J2ME associates key code values with constants; however, use the constant instead of the constant value.
- Using constants within the code clarifies the reference to a key because the name of the constant contains the name of the key associated with the key code.



Constant	Value
KEY_NUM0	48
KEY_NUM1	49
KEY_NUM2	50
KEY_NUM3	51
KEY_NUM4	52
KEY_NUM5	53
KEY_NUM6	54
KEY_NUM7	55
KEY_NUM8	56
KEY_NUM9	57
KEY_STAR	42
KEY_POUND	35

Keycode constants and keycode values

- There are three empty methods that are called when a particular key event occurs while the MIDlet is running. We should override these methods if your application needs to call them. These methods are:
  - **keyPressed()-**method is called by the application manager whenever a key is pressed by the user.

- **keyReleased()**-method is called when the key selected by the user is released.
- **keyRepeated()-**method is called by the application manager when the user holds down the key, causing the key to be automatically repeated. Not all devices support repeated keys. Your MIDlet can inquire whether or not the repeated key feature is supported by calling the **hasRepeatEvents()** method.

- All of these methods have empty implementation. We must override each method if our application needs to process the related key events.
- Many of the applications we create that implement a low-level user interface, will only need to override the keyPressed() method because we need to know which key was selected by the user.
- The keyRelease() method and the keyRepeated() method are overridden only for applications that have special processing whenever a person releases a key or holds down a key for an extended period.
- All three methods require **one parameter**, which is an integer that represents the value of the key code passed to the method by the device's application manager. An if statement or switch case statement is used to compare the incoming key code with key code constants that are processed by the MIDlet.

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
 public class KeyCodeExample extends MIDlet
{
             private Display display;
            private MyCanvas canvas;
             public KeyCodeExample ()
            {
                        display = Display.getDisplay(this);
                         canvas = new MyCanvas(this);
            }

            protected void startApp()
             {
                         display.setCurrent(canvas);
            }
             protected void pauseApp()
             {
             }
            protected void destroyApp( boolean unconditional )
            {
            }
```

```
     public void exitMIDlet()
     {
                destroyApp(true);
                notifyDestroyed();
        }
 }
class MyCanvas extends Canvas implements CommandListener
{
                private Command exit;
                private String direction;
                private KeyCodeExample keyCodeExample;
                public MyCanvas (KeyCodeExample keyCodeExample)
                {
                          direction = "2=up 8=dn 4=lt 6=rt";
                          this.keyCodeExample = keyCodeExample;
                          exit = new Command("Exit", Command.EXIT, 1);
                          addCommand(exit);
                          setCommandListener(this);
                }
                protected void paint(Graphics graphics)
                 {
                          graphics.setColor(255,255,255);
                          graphics.fillRect(0, 0, getWidth(), getHeight());
                           graphics.setColor(255, 0, 0);
                graphics.drawString(direction, 0, 0, Graphics.TOP | Graphics.LEFT);
                 }
 public void commandAction(Command command, Displayable displayable)
 {
                 if (command == exit)
                 {
                           keyCodeExample.exitMIDlet();
                 }
 }
protected void keyPressed(int key)
{
                switch ( key )
                {
                          Case: KEY_NUM2: direction = “up”;
                           break;
                          case: KEY_NUM8: direction = “down”;
                           break;
                          case: KEY_NUM4: direction = “left”;
                           break;
                          case: KEY_NUM6: direction = “right”;
                          break;
                }
```

```
            repaint();
             }
    }
```

- **Detecting and Processing Key Codes:** The MIDlet begins by declaring two variables. The first variable references an **instance of the Display class**, the other variable references a developer-defined class called **MyCanvas**.

- Two instances of classes are created within the KeyCodeExample constructor. These are the **Display class and the MyCanvas class**.

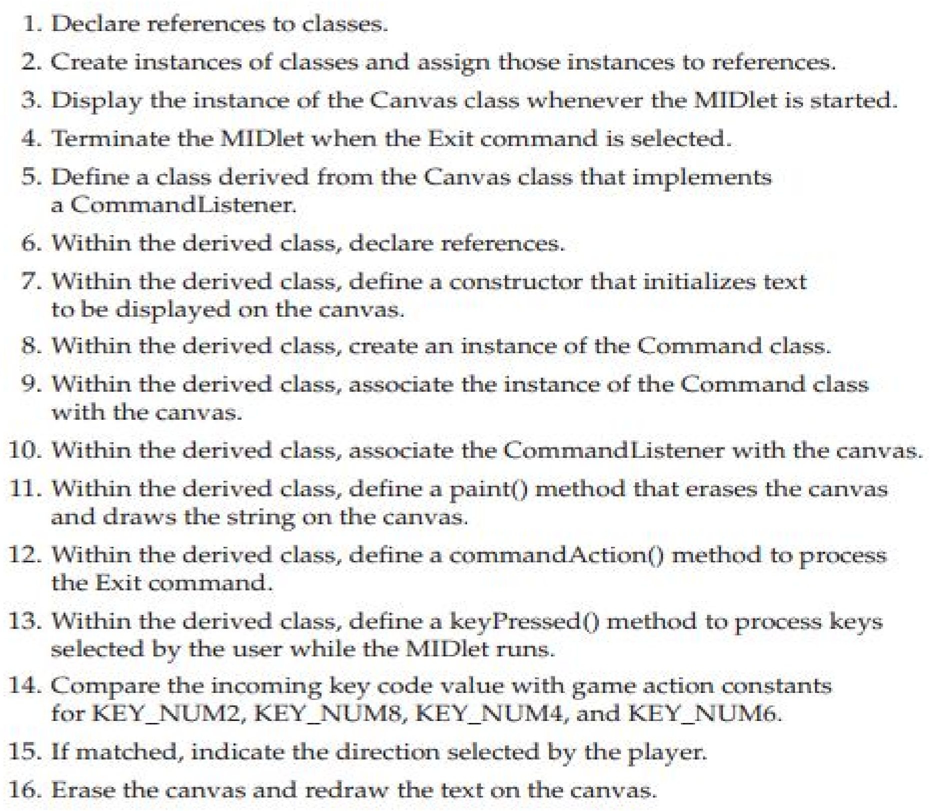
- In **MyCanvas class**, the first statement declares a **reference to an instance of the Command class** that is used to terminate the MIDlet.

- Next, **reference to a String** called direction is declared, followed by the creation of an **instance of the KeyCodeExample class**.

- The constructor of MyCanvas is passed an instance of the KeyCodeExample class that is referenced internally to the constructor.

- The direction string is initialized with text that describes directional keys; the text is displayed on the screen when the device's application manager calls the paint() method.

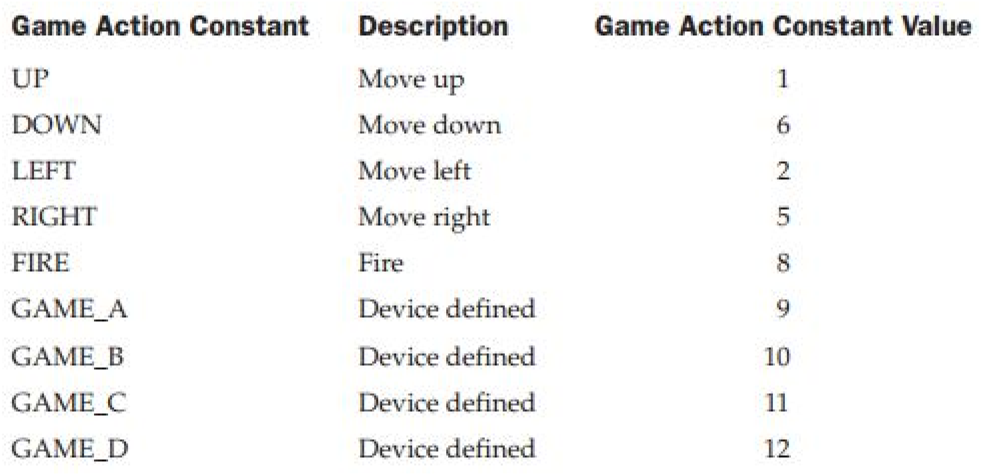
- The following are the steps:



1. Declare references to classes.
2. Create instances of classes and assign those instances to references.
3. Display the instance of the Canvas class whenever the MIDlet is started.
4. Terminate the MIDlet when the Exit command is selected.
5. Define a class derived from the Canvas class that implements a CommandListener.
6. Within the derived class, declare references.
7. Within the derived class, define a constructor that initializes text to be displayed on the canvas.
8. Within the derived class, create an instance of the Command class.
9. Within the derived class, associate the instance of the Command class with the canvas.
10. Within the derived class, associate the CommandListener with the canvas.
11. Within the derived class, define a paint() method that erases the canvas and draws the string on the canvas.
12. Within the derived class, define a commandAction() method to process the Exit command.
13. Within the derived class, define a keyPressed() method to process keys selected by the user while the MIDlet runs.
14. Compare the incoming key code value with game action constants for KEY_NUM2, KEY_NUM8, KEY_NUM4, and KEY_NUM6.
15. If matched, indicate the direction selected by the player.
16. Erase the canvas and redraw the text on the canvas.

Working with Game Actions:

- The theme may differ among computer games, but the way players interact with a game is fairly constant across all computer games. Players can move up, down, left, right, and they can fire.
- The directional movement causes a game piece to move in a corresponding direction or changes the viewpoint of the player, depending on the nature of the game. Fire causes an event to occur within the game, such as releasing a bullet from a gun.
- **Directional movement and fire** are called as **"game actions"**, and MIDP game action defines constants that enable you to utilize game actions within our MIDlet without being concerned about the appropriate key code that is assigned to each action.
- Each game action is associated with one or more keys on the keypad. For example, the down game action might be associated with a down directional key and a number on the keypad. Pressing either key causes the same game action to occur.
- Each key can be assigned to only one game action. This means pressing the down game action key doesn't also generate an up game action.
- The following list contains game action constants that are used when developing a game for a small computing device. We can refer either the name of the constant or the value of the constant within your MIDlet to determine the game action selected by the player. It is always best to reference the constant rather than the constant value.



Game Action Constant	Description	Game Action Constant Value
UP	Move up	1
DOWN	Move down	6
LEFT	Move left	2
RIGHT	Move right	5
FIRE	Fire	8
GAME_A	Device defined	9
GAME_B	Device defined	10
GAME_C	Device defined	11
GAME_D	Device defined	12

- A game action causes the keyPressed() method, keyReleased() method, and keyRepeated() method to be called, depending on the key pressed by the player.
- We can detect which game action occurred by calling the **getGameAction() method**.
- **getGameAction() method :** requires one parameter—the key code of the key selected by the player—which is passed as a parameter to the keyPressed(), keyReleased(), or keyRepeated() method.
- An if statement or a switch case statement can be used to compare the incoming key code to game action constants.
- Each game action constant is a data member of the Canvas class and is referenced by using the name of the game action constant, such as **Canvas.LEFT, Canvas.RIGHT, Canvas.UP, Canvas.DOWN, and Canvas.FIRE.**
- There are two alternatives to detect the game action key selected by the player:
  - **getKeyCode() method:** requires **one parameter**, which is the name of the game action constant. It **returns the key code value** associated with the game action constant that can then be directly compared to the incoming key code value passed to the keyPressed(), keyReleased(), or keyRepeated() method.

    ```
    if (getKeyCode(FIRE) == keycode)
     {
            //fire
     }
    ```
  - **getKeyName() method:** to determine the player's selection is to retrieve the name of the key that is associated with the incoming key code. It requires **one parameter**, which is the **key code value**. It **returns the name of the key** represented by the key code value.
- To use the getKeyName() method to detect the game action key selected by the player, we must first determine the key name for each of the game action keys. This can be done by calling the getKeyCode() method, passing it the game action constant name, and then passing the return value from the getKeyCode() method to the getKeyName() method, which returns the name of the key associated with the key code value. This technique is illustrated in the following code segment:

  ```
  if (getKeyName(getKeyCode(FIRE).equals(getKeyName(keycode))))
  {
          //fire
  ```

```
}
```

Working with Pointer Devices:

- A pointer device is something other than a keyboard or keypad that is used to interact with an application. The most commonly used pointer devices are **a touch screen and a mouse**.
- In a J2ME application we are not involved in the details of how a pointer device interfaces with a small computer or how someone uses the pointer device to interact with your MIDlet. The **device manufacturer and the implementation of the Java Virtual Machine** handle those details.
- We have to develop routines within your MIDlet to process pointer events.
- A pointer event occurs whenever the person uses a pointer device to interact with your MIDlet. There are three pointer events that your MIDlet must process. When the person:
  - ➢ presses a pointer device,
  - ➢ releases a pointer device,
  - ➢ drags a pointer device.
- A person presses a pointer device by applying pressure to a portion of a touch screen or by clicking the mouse button. This causes a press event. A release event occurs once pressure is removed from the touch screen or the mouse button. And your MIDlet is notified of a drag event whenever the person moves the pointer device during a press event.
- The MIDlet processes pointer events by defining three methods that are automatically called by the device's application manager when a pointer event occurs. These methods are:
  - ➢ the pointerPressed() method,
  - ➢ the pointerReleased() method,
  - ➢ the pointerDragged() method.
- All three methods require two parameters.
  - ➢ an integer representing the x coordinate of the pointer device
  - ➢ an integer representing the y coordinate.

- **Detecting and Processing Pointer Events:** Let a MIDlet prompts the user to draw a line across the screen using a pointer device. A line is drawn by pressing the pointer device at a particular location on the canvas, then while pressed (or while holding down the mouse button), the person drags the pointer device to another position on the canvas before releasing the pointer device.

- A press event is detected and the **pointerPressed()** method is called when the pointer device is pressed. The pointerPressed() method receives the coordinate of the pointer on the canvas.That is the **starting coordinate**.

- Dragging the pointer device is a drag event and causes the **pointerDragged()** method to be invoked continuously until the person stops dragging the pointer device. The pointerDragged() method is called each time the pointer device is dragged and is passed the coordinate of the pointer device when the drag event occurs. These coordinates are the **current coordinates.**

- Finally, the release event occurs when the person removes pressure from the pointer device (removes the finger or implement from the touch screen or releases the mouse button). The **pointerReleased()** method is then called and passed the pointer device's coordinate on the canvas where the person released the pointer device. These are the **end coordinates.**

Graphics

- The canvas is organized into a grid in which each cell of the grid is a pixel. Coordinates identify each cell.

- An image is drawn on the canvas by using a virtual graphical device called a **graphic context**, such as the rectangle and line. A graphic context is an instance of the **Graphics class**.

- Reference to the graphic context is passed to the paint() method.

- Once the MIDlet leaves the paint() method, the graphic context goes out of scope. The graphic context can no longer be used to draw on the canvas, even if reference to the graphic context is retained.

- But, a graphic context created in association with a mutable image remains available to the MIDlet as long as reference to the image and the image itself remains in scope.

1. Stroke Style and Color:

- Every graphic context has two characteristics you can control from within the MIDlet. These are :

- **Stroke style:**
  - Stroke style defines the appearance of lines used to draw an image on the canvas.
  - We use **two kinds of stroke styles** when drawing images on the canvas:
    - ✧ **Solid:** As the names imply, the solid stroke style causes the graphic context to use a solid line when drawing the image. The solid stroke style is the default.
    - ✧ **Dotted:** the dotted stroke style results in the image being drawn using a dotted line. Skipping pixels along the lines of the image creates the dotted stroke. The small computing device determines the number of pixels skipped.
- **Color:**
  - It specifies the background and foreground color of the image.
  - Combining degrees of red, green, and blue creates the foreground and background color of a graphic context. The degree of each color is specified as an integer value within the range of 0 to 255. Zero produces the darkest possible value of the color, and 255 produces the lightest possible value.
  - Color values 0, 0, 0 (red, green, blue) produce black, and color values 255, 255, 255 produce white.
  - All integers in Java are 32 bits. Of those 32 bits, 8 bits are used to represent each color-red, blue, and green. All color values are stored in one integer. The 8 highest order bits are not used.

- **Methods Used in stroke style:**
  - **setStrokeStyle() method** :
    - determines the stroke style that will be used by a graphic context.
    - **void setStrokeStyle(int style)-** Set the stroke style of a graphic context, where style is either SOLID or DOTTED.
    - A stroke style setting is particular to each graphic context and does not affect other graphic contexts.

- For example, one graphic context can be set to a dotted stroke style and another set to a solid stroke style. Both graphic contexts can be used to draw images on the same canvas without affecting each other's stroke style.
- It requires one parameter, which is a constant that represents a stroke style. There are two constants: **SOLID and DOTTED**, both of which are members of the Graphics class.
- You can change the stroke style of a graphic context anytime within your MIDlet by calling the setStrokeStyle() and passing the setStrokeStyle() the constant that represents a different stroke style.

➢ **getStrokeStyle():**

- We can determine the current stroke style of a graphic context by calling this method. It returns an integer that can be compared within your MIDlet to the stroke style constants.
- **int getStrokeStyle()**

- **Methods Used in Color:**

➢ **isColor():**

- We have to determine whether a device supports color and the number of colors or shades of gray that are supported by calling the appropriate Display class method within your MIDlet.
- It returns a boolean value that is true if color is supported; otherwise a false value is returned, indicating that the device supports the gray scale instead of color.

➢ **numColors():**

- It returns an integer representing the number of colors or shades of gray supported by the device.

➢ **setColor():**

- we can set the color of a graphic context by calling the setColor() method of the Graphics class.
- The setColor() method requires either one parameter or three parameters depending on how you represent your choice of color. A color can be

represented as one integer or three integers, where each of the three integers represents a color value of red, green, and blue.

- **void setColor(int RGB)** -Change the current color to the integer represented by RGB. Red, green, and blue color values are consolidated into one integer value and passed to the setColor() method.
- **void setColor(int red, int green, int blue)-** Change the current color to integers represented by red, green, and blue color values.

➢ **getColor():**

- Retrieve the integer value that represents the current color.
- retrieves the 32-bit color value, then using a bit mask , we have to extract each component of the color.
- **int getColor()**

➢ **int getBlueComponent()**:Retrieve the blue color value.

➢ **int getGreenComponent()**: Retrieve the green color value.

➢ **int getRedComponent()** : Retrieve the red color value.

➢ **void setGrayScale(int value):** Change the value of the current gray scale.

➢ **int getGrayScale() :**Retrieve the value of the current gray scale.

<u>2. Lines</u>

- Lines are drawn on the canvas by calling the drawLine() method.
- The thickness of the line, referred to as the **weight**, is typically measured in point size, where zero is the thinnest possible line.
- We cannot easily change the weight of a line drawn on the canvas because the weight is always one pixel. The only way to create a heavier (thicker) line is to draw multiple, abutting lines, which appear as one thicker line on the screen.
- **Method:**

  ➢ **drawLine() method :**

  - creates a line from a starting coordinate to an ending coordinate. Four parameters are required by the drawLine() method. The first two parameters are integers representing the starting x, y coordinate of the line. The other two parameters are integers representing the ending x, y coordinate of the line.

- **Void drawLine(int x1, int y1, int x2, int y2)**

- The color of the line is determined by the color setting of the graphic context used to draw the line. We can set the color of the line by calling the **setColor()** method before invoking the drawLine() method.

3. Rectangles

- A rectangle is an area of the canvas defined by four corners. We define a rectangle's dimensions by identifying coordinates for the **upper-left corner and the lower-right corner**.
- Four types of rectangles can be drawn on a canvas. These are :
  - **an outlined rectangle**-line segments connecting the corners are drawn. The inside of the rectangle remains the same color as the outside of the rectangle
  - **filled rectangle**-draws line segments to connect corners, but the inside of the rectangle is filled with the same color as the drawn line segments
  - outlined rectangle with rounded corners
  - a filled rectangle with rounded corners.
- The color used to draw a rectangle must be set using the **setColor() method** before drawing the rectangle. Otherwise the current color of the graphic context is used both to color the outline and fill the inside of the rectangle, depending on the type of rectangle being drawn.
- **Methods used with Rectangle:**
  - **drawRect() method:**
    - **void drawRect(int x1, int y1, int x2, int y2)**
    - Draw a outlined rectangle, where x1, y1 represents the coordinate of the upper-left corner of the rectangle, and x2, y2 represents the width and height of the rectangle.
    - The first two parameters are the coordinates of the upperleft corner of the rectangle (x1, y1), and the last two parameters are the width and height of the rectangle (x2, y2).

- **fillRect() method:**
  - **void fillRect(int x1, int y1, int x2, int y2)**
  - Draw a filled rectangle, where x1, y1 represents the coordinate of the upper-left corner of the rectangle, and x2, y2 represents the width and height of the rectangle.
- **void drawroundRect() method:**
  - **void drawroundRect(int x1, int y1, int x2, int y2, int arcW, int arcH).**
  - We must specify the horizontal and vertical diameter of the arc used to create the round corners. The horizontal diameter is referred to as the **arc width,** and the vertical diameter is referred to as the **arc height**.
  - Used to draw a rounded rectangle, where x1, y1 represents the coordinate of the upper-left corner of the rectangle, and x2, y2 represents the width and height of the rectangle. arcW is the angle for the width of the arc, and arcH is the angle for the height of the arc..
  - The **diameter** represents the **sharpness of the corner**, where the smaller the diameter, the sharper the corner appears. Both the horizontal and vertical diameters are defined as integers.
- **fillRoundRect() method:**
  - **void fillRoundedRect (int x1, int y1, int x2, int y2, int arcW, int arcH)**
  - Used to draw a rounded filled rectangle, where x1, y1 represents the coordinate of the upper-left corner of the rectangle, and x2, y2 represents the width and height of the rectangle. arcW is the angle for the width of the arc, and arcH is the angle for the height of the arc.

4. Arcs

- An arc is a curved line segment that is used to draw circles, ovals, and other curved images.
- Drawing an arc is a bit difficult because you must define the area of the canvas that will be covered by the arc and the angle used to draw the arc.
- The first step in drawing an arc is to **decide the area of the canvas** that will be covered by the arc. The area is defined as a rectangle rather than the circumference of the arc.This is a box in which an arc is drawn.

- A rectangle is defined by specifying two sets of coordinates. The first set of coordinates (x1, y1) set the upper-left corner of the rectangle. The other set of coordinates (x2, y2) set the lower-right corner of the rectangle.
- Once the rectangle is defined, you must **define two angles used to draw the arc**. An angle is defined in degrees from 0 to 360 degrees. The first angle is the starting point of the arc, and the other angle is the end point of the arc.
- Picture a clock. The 3 o'clock position is 0 degree. Degrees are incremented as you move counterclockwise. The 12 o'clock position is 90 degrees, 9 o'clock is 180 degrees, and 6 o'clock is 270 degrees. Degrees decrement as you move clockwise. Based on the picture of the clock, you must select the angle where the arc begins to be drawn. Likewise, you select the angle where the arc terminates.
- You can draw two kinds of arcs:
    - ➢ **an outlined arc**: In an outlined arc, only the circumference of the arc is drawn (like a smile).
    - ➢ **a filled arc**- In a filled arc, the circumference of the arc is drawn, and the area within the center and the circumference is filled with the color of the graphic context used to draw the arc (a colored circle, for example).
- **Methods used in arc:**
    - ➢ **void drawArc():**
        - ❖ **Void drawArc(int x1, int y1, int x2, int y2, startAngle, endAngle)**
        - ❖ Draw an outline arc within the rectangle defined as the first four parameters beginning with the startAngle and terminating with the endAngle.
    - ➢ **Void fillArc():**
        - ❖ **Void fillArc(int x1, int y1, int x2, int y2, startAngle, endAngle)**
        - ❖ Draw a filled arc within the rectangle defined as the first four parameters beginning with the startAngle and terminating with the endAngle

<u>5. Text:</u>

- Displaying text using the low-level user interface differs from displaying text with the high-level user interface.

- Using the high-level user interface, text is displayed by calling one of several methods and passing the text as a parameter to those methods. Each method determines how to display the text without requiring any direction from the user.

- When displaying text using the low-level user interface, the user controls the details of how text is displayed.

- The user can determine the appearance of each letter of the text, the height and width of every character, and the size of the space between characters, and other such details.

- The font used to display text determines the appearance of text on the screen. We identify fonts by name, such as Times Roman and Arial. **A font name** represents a set of font metrics that determine the pixels necessary to generate alphanumeric characters and symbols on the screen and on a printed page.

- The J2SE specification defines the **FontMetrics class** that is used to specify every detail of the font; but the **J2ME specification does not support the FontMetrics class.**There are three font metrics that are controllable by a MIDlet. These are :

    - ➢ **The font face**- font face is similar to selecting the font name in a word processing. The selections are:

        - ▪ default system font face- is the font face that the device chooses

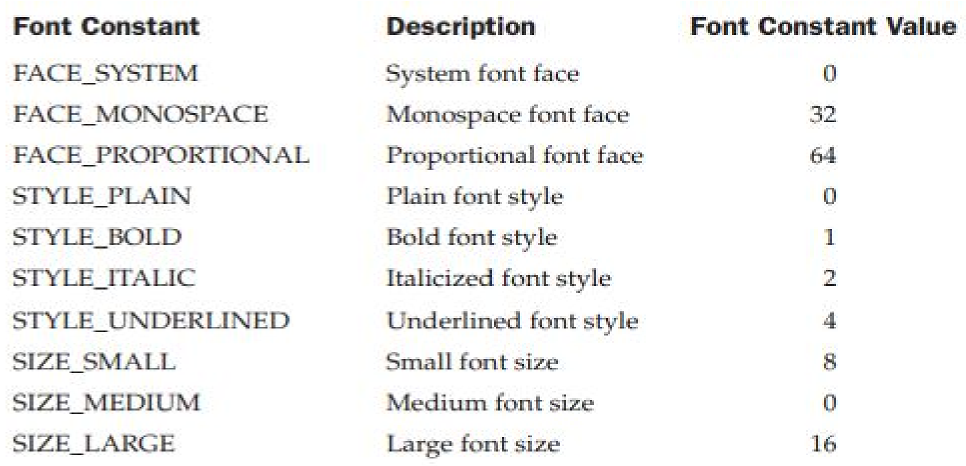
        - ▪ monospace font face- e is a font face in which all characters are the same width.

        - ▪ proportional font face- is a font face in which the width of a character is determined by the nature of the character. For example, the letter W is wider than the letter A, and the letter I has a smaller width than A.

    - ➢ **the font style-** There are four font styles to choose from, which are identical to styles available in a word processor. These are plain, bold, italic, and underlined. You can apply multiple font styles to text by using the OR (|) operator.

    - ➢ **the font size**- Font sizes are small, medium, and large. The small computing device determines the actual size of the font.

- Font faces, font styles, and font sizes are associated with font constants that are used to identify your font request.



Font Constant	Description	Font Constant Value
FACE_SYSTEM	System font face	0
FACE_MONOSPACE	Monospace font face	32
FACE_PROPORTIONAL	Proportional font face	64
STYLE_PLAIN	Plain font style	0
STYLE_BOLD	Bold font style	1
STYLE_ITALIC	Italicized font style	2
STYLE_UNDERLINED	Underlined font style	4
SIZE_SMALL	Small font size	8
SIZE_MEDIUM	Medium font size	0
SIZE_LARGE	Large font size	16

Font Constants

- The selection of a font is a request and not a directive to the device. The device will match your request to available fonts, but there is no guarantee that your request will be fulfilled.

- **Methods used in text:**

  - **void drawChar(char character, int x, int y, int anchor)-**

    - Used for drawing one character on the canvas.
    - It requires four parameters. The first parameter is the character. The next two parameters are the x, y coordinates of the upper-left corner of the boundary box. And the last parameter is the anchor point.
    - It draws a character at the x, y coordinate on the canvas using the specified anchor point.

  - **void drawChars(char[] data, int offset, int len, int x, int y, int anchor)**

    - It is used to draw an array of characters or a subset of a character array.
    - The drawChars() method requires six parameters. The first parameter is the character array. The second parameter is an integer representing the offset in the array where the first character to be drawn is located. The third parameter indicates how many characters are to be drawn starting at the offset specified by the second parameter. The fourth and fifth parameters are the x, y coordinates of the boundary box's anchor point.

    - Draw a subset of a character array the length specified by len and beginning with the character indicated by the offset. Draw the subset at the x, y coordinate on the canvas using the specified anchor point.

- **void drawString (String str, int x, int y, int anchor)**
    - It requires four parameters. The first parameter is the string. The second and third parameters are the x, y coordinates that specify the anchor point coordinate. And the last parameter specifies the part of the boundary box to locate the anchor point.
    - Draw a string at the x, y coordinate on the canvas using the specified anchor point.

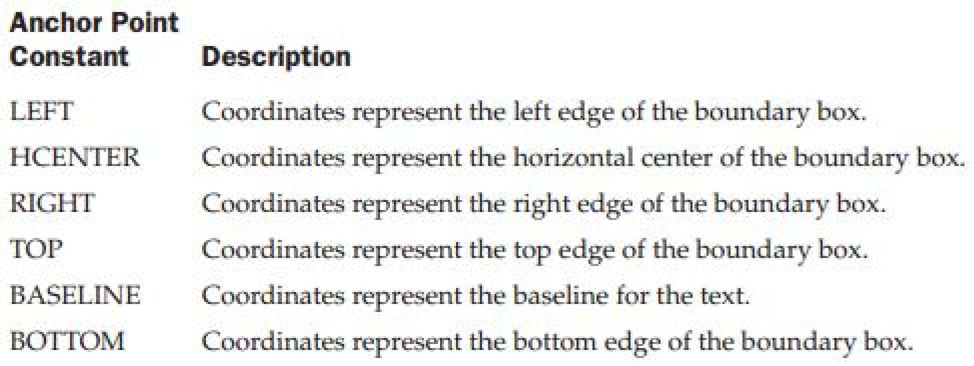
- **void drawSubstring(String str, int offset, int len, int x, int y, int anchor)-** Draw a substring the length specified by len and beginning with the character indicated by the offset. Draw the substring at the x, y coordinate on the canvas using the specified anchor point.

- **Font getFont()-**Return the font of the graphic context.

- **void setFont(Font font)-**
    - It is a member of the Graphics class. It is used to set the font of the graphic context, where font is the new font.
    - It requires one parameter, which is an instance of the Font class. obtain the instance of the Font class by calling the getFont() method. The getFont() method requires three parameters. The first parameter is the font face, the second parameter is the font style, and the last parameter is the font size.

graphics.setFont(Font.getFont(Font.PROPORTIONAL,Font.BOLD|Font.ITALIC, Font.SMALL);

- **Aligning Text:**
    - When drawing text on the canvas we must know measurements of text that is already on the canvas as well as measurements of text being drawn.
    - Text is drawn within a virtual bounding box, which is an invisible box that defines the boundaries of the text.
    - First you specify a position on the screen by setting coordinates. Let’s call them x, y. Next, you specify an anchor point that identifies the relationship of the coordinate to the bounding box.



Anchor Point Constant	Description
LEFT	Coordinates represent the left edge of the boundary box.
HCENTER	Coordinates represent the horizontal center of the boundary box.
RIGHT	Coordinates represent the right edge of the boundary box.
TOP	Coordinates represent the top edge of the boundary box.
BASELINE	Coordinates represent the baseline for the text.
BOTTOM	Coordinates represent the bottom edge of the boundary box.

Anchor Point Constants

- Suppose you want the coordinates to be the upper-left corner of the boundary box, then we specify the anchor point TOP | LEFT. If we want the coordinates to represent the lower-right corner of the boundary box, then use the BOTTOM | RIGHT anchor point.
- The width and height of the text determine the coordinate of the opposite corner of the boundary box. **There are three horizontal values, LEFT, HCENTER, and RIGHT, and three vertical values, TOP, BASELINE, and BOTTOM.** Horizontal and vertical values define the location within the bounding box of the specified coordinate. The values are combined to define an anchor point. We pick a location on the screen, and if we want that position to be the upper-right of the bounding box, we set the anchor point to GRAPHICS.TOP | GRAPHICS.RIGHT.
- **Text is measured** by following parameters:
  - **Ascent-** is the measurement from the baseline of the text to the top of the highest character in the text.
    - The ascent is measured by calling the **getBaselinePosition() method**. No parameters are required by this method because the method analyzes text already associated with the graphic context used to draw text on the canvas. It returns an integer that represents the pixels between the baseline and the top character within the text.
  - **Descent-** is from the baseline to the lowest character in the text. Let's examine the following text to identify the ascent and descent: "We work together." The ascent is the distance between the bottom and the top of the W because the W is the highest character in the text. The bottom of the W is the baseline. The descent is the distance between the bottom of the g

and the bottom of the W, or the bottom of any of the other characters of the text, because the g is the lowest character within the text

- **Leading-** is the distance between the descent and ascent of abutting lines of text.

- **font height-** is the sum of the ascent, leading, and descent.

  - ✧ The font height is measured by calling the **getHeight() method**. The getHeight() method does not require any parameters and returns an integer representing the pixel measurement of the font height

- **Advance-** is the text length, including spaces between characters. We can determine the advance by calling the following methods:

  - ✧ The **charWidth() method of Font class** measures the width of one character and requires one parameter, which is a character. This method returns an integer representing the width of the character in pixels.

  - ✧ The **charsWidth() method** measures a series of characters in a character array. This method requires three parameters. The first parameter is the character array. The second parameter is an integer representing the first character of the series being measured. The last parameter is an integer representing the length of the series.

  - ✧ The **substringWidth() method** measures a substring of characters within a string and also requires three parameters. The first parameter is the string. The second parameter is an integer representing the first character of the substring, and the last parameter is an integer representing the length of the substring.

6. Images

- An image is an instance of an Image object that has been previously created either by your MIDlet using a graphic context, or by graphics software.

- There are two kinds of images:

  - ➢ a mutable image can be modified by your MIDlet

  - ➢ an immutable image cannot be modified by your MIDlet

- **Methods used in Image:**
  - **createImage() method:**
    - It requires one parameter or two parameters depending on whether you are drawing a mutable or immutable image.
      - **One parameter** is required for the createImage() method if the instance is used to draw an immutable image. The parameter is the file name of the image, including the full directory path.

Image image = Image.createImage("/myImage.png");

- **Two parameters** are required for the createImage() method if the instance is used to draw a mutable image. These parameters define the height and width in pixels of the memory block used to store the mutable image as it is being drawn. The following code segment creates a block of memory 20 pixels high and 10 pixels wide for a total image size of 200 pixels.

Image tmpImg = Image.createImage(20, 10);

- **drawImage() method:**
  - **void drawImage(Image img, int x, int y, int anchor)**
  - Draw the image specified in img, where the upper-left corner of the image is positioned at coordinate x, y using anchor point referenced by anchor.
  - We can create a mutable image by calling the getGraphics() method of the Image class to return an instance of the Graphics class, which is the graphic context used to draw the mutable image.

Image image = Image.createImage(20, 10);

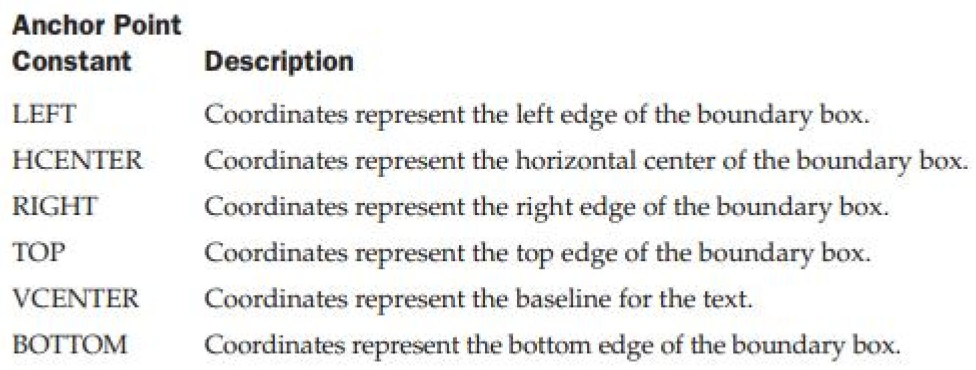
Graphics graphic = image.getGraphics();

graphic.drawLine(5, 5, 20, 20);

- The drawImage() requires four parameters.
  - The first parameter is the instance of the Image class that references the image.
  - The next two parameters are integers that represent the coordinate used to position the image on the canvas. The image is drawn

within a virtual boundary box similar to the boundary box used to draw text on the canvas. The coordinate represents the upper-left corner of the boundary box.

- ✧ The last parameter is an integer that represents the portion of the image bounding box that is anchored at the specified coordinate. The image anchor point is used to finely adjust the location of the image within the boundary box.



Anchor Point Constant	Description
LEFT	Coordinates represent the left edge of the boundary box.
HCENTER	Coordinates represent the horizontal center of the boundary box.
RIGHT	Coordinates represent the right edge of the boundary box.
TOP	Coordinates represent the top edge of the boundary box.
VCENTER	Coordinates represent the baseline for the text.
BOTTOM	Coordinates represent the bottom edge of the boundary box.

6. Repositioning Text and Images

- Each position on a canvas is organized by a row and column grid, where each coordinate identifies a pixel. The upper-left corner of the canvas is always coordinate 0, 0.
- Coordinates are used with methods of a graphic context to identify locations on the canvas for drawing and positioning an image.
- Coordinates are passed explicitly to these methods by providing exact coordinates, such as 5, 10, or implicitly by referencing an offset of an explicit coordinate, such as 5 + 3, 10 + 3. In either case, coordinates are based on the 0, 0 coordinate being the upper-left corner of the canvas.
- Sometimes we need to proportionally shift all text and images to a new location on the canvas. There are two techniques we can use to make this move:
    - ➢ We can change all coordinates of an image to reflect its new position,
    - ➢ We can move the entire grid and let the device adjust all the coordinates based on the new position of the upper-left corner of the grid on the canvas. This technique, called **translating coordinates**.It is a more efficient way of moving text and images than modifying coordinates within your MIDlet.
    - ➢

- **Methods used for translate():**
    - **void translate (int x, int y)-** Translate the specified by x, y coordinate.
    - **int getTranslateX()-** Retrieve the translated x coordinate.
    - **int getTranslateY() -**Retrieve the translated y coordinate.

Clipping Regions:

- Clipping is a region in which we can define a region on the screen
- Anything included or passing in the clipping region is been displayed and the rest is rejected.
- Clipping is mostly used while creating animations.
- A clipping region is a rectangular box on the canvas that is used to draw image.
- The **entire canvas** is the default clipping region, so the image is always drawn fully when we use the **drawImage ()** method.
- If we define a clipping region than the portion of the image that appears within the clipping region is drawn on the canvas, other portion of image still exist but are not drawn.

Methods:

setClip():

- We set the clipping region by calling the setClip**()** method of the Graphics class.
- The setClip() method requires four parameters.
    - The first two parameters are integers representing the upper-left corner coordinates of the clipping region,
    - The third and fourth parameters are integers representing the width and height of the clipping region.

clipRect():

- We can reduce the size of the clipping region by calling the clipRect() method, which requires four parameters which we used in setClip() method.

There are another four methods available to get the attribute information about the clipping region. Those are

getClipX(),getClipY(),getClipHeight(),getClipWidth()

- The **getClipX()** method and **getClipY()** method return upper-left coordinates of the existing clipping region.
- The **getClipHeight()** method returns the height and **getClipWidth()** method returns the width of the existing clipping region.

<u>Animation:</u>

- Animation is the simulation of motion on the screen caused by the timed drawing of a series of related images.
- Each image is referred to as a ***cell*** in animation terminology but, we'll use the term ***image*** instead because a cell also refers to an intersection of the grid used for positioning objects on the canvas.
- Each image in the animation must relate to the image currently displayed and the next image to be displayed.
- The first step in animation is to carefully lay out the progression of images that we'll need to display in order to create the illusion of movement.
- Next, create each image so that each is slightly different from the previous image.
- We can create these images as either mutable or immutable.
    - ➢ A mutable image is created using methods described in this chapter.
    - ➢ An immutable image is created using graphics software or digital photography.
- Once images are drawn or loaded from a file, you display each image by first calling the **createImage ()** method to create an instance of the Image class and then calling the **drawImage ()** method.
- Timing the display of each image is controlled by a timing loop (sometimes within a while loop) so that the animation recycles to the first image after the last image is displayed.

UNIT-II

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1) Which of the following is a low level display [ ]
a) Alert b) TextBox c) Canvas d) Screen

2) The three parameters required by Command class constructor are ___________, ____________, ___________.

3) Classes that implement Command Listener must implement ___________ method.

4) The method itemStateChanged() is in the _____ class. [ ]
a) ChoiceGroup b) Item c) Canvas d) Command

5) commandAction() takes instances of __________ as parameters. [ ]
a) Command Class b) Displayable Class c) both a & b d) none

6) __________ class is used for displaying error and warning messages on the screen. [ ]
a) Alert b) Canvas c) Item d) Ticker

7) _______class is used to display message on the screen. [ ]
a) Alert b) Canvas c) Item d) StringItem

8) An immutable image is drawn on a ________ and immutable image is drawn on a _________. [ ]
a) Screen, canvas b) canvas, screen c) screen, screen d) Canvas, Canvas

9) TextBox class is derived from ________ where as TextField class is derived from _________. [ ]
a) Screen, Item b) Item, Screen c) Item, Item d) Screen, Screen.

10) __________ class is used to scroll the text horizontally. [ ]
a) Alert b) Canvas c) Item d) Ticker

11) What is the purpose of paint() method of Displayable class?

12) What are the low level components that generate low level events?

13) The Item class is derived from the________ class. [ ]
a) Canvas b) Screen c) Form d) All

14) An exclusive instance of ChoiceGroup class appears as a set of ___________. [ ]
a) Radio Buttons b) Check Boxes c) TextField d) ImageItem

15) Every graphic context has two characteristics___________ and ____________. [ ]
a) Stroke-style b) color c) Paint d) both a & b

16) The ______________ class creates an animated progress bar that graphically represents the status of a process. [ ]
a) Gauge b) Canvas c)Bar d) none

17) The screen class and its derived classes are referred to as __________ user interface components. [ ]
a) High level Display b) Low level Display c) Graphical Display d) None

18) An image is drawn on a canvas using ___________________. [ ]
a) Pen b) Virtual Pen c) Pointer d) both a&b

19) The action performed by "Priority" Parameter in Command Class is ___________ [ ]
a) To set the Place of Commands b) To resolve conflicts when priorities equal
c) Application Manager dependent d) All the above

20) Alert dialogue box is designed to ______________ [ ]
a) To display any type of message b) Retrieve input from the user
c) To display error Message d) None

21) By using Ticker class we can control______________ [ ]
a)Location on the screen where Scrolling occurs b) Speed of Scrolling
c) both a&b d) None

22)The instance of the TextBox class______________ [ ]
a) Must be contained in an instance of Form class
b) Must not be contained in an instance of Form class
c) May or may not be contained in an instance of Form class
d) None

23) The DateField class is used to _______date/time into a MIDlet. [ ]
a) Display b) edit c) input d) All the above

24) A Command event is automatically generated when the user selects an item from an instance of ___ [ ]
a) ChoiceGroup class b) List class c) Radio Group class d) All of the above

SECTION-B

Descriptive Questions

1. Explain about the methods used in Display class with an example?
2. Explain about Command class with an example
3. Explain about Ticker class with an example.
4. Write the Display class hierarchy?

5. Explain about methods used in a) Form class b) Alert class
6. Differentiate between TextField and TextBox.
7. Write about any 10 methods used inn Graphics Class?
8. Write short notes on

    a) Canvas Layout   b) User interactions

9. Differentiate between List class and ChoiceGroup class?
10. Demonstrate working of CommandListener Interface with an example?
11. Discuss how warning message and error messages can be displayed on the screen?
12. Discuss how StringItem and ImageItem class differs from other classes derived from Item class?
13. Discuss how TextBox class differs from TextField class?
14. Show how the user interactions will be done in Low-Level Display?
15. Discuss how Displaying Text in Low-Level User Interface differs from displaying Text with High-Level User Interface?
16. Create an animated progress bar that graphically represents the status of a process.
17. Develop a MIDlet that prints your name on emulator.

Unit-III

Objective:

- To access and work with database under the J2ME.

Syllabus:

Database concepts Record Management System: Record Storage, Writing and Reading Records, Writing and Reading Mixed Data Types, Record Enumeration, Sorting Records, Searching Records, Record Listener. J2ME Database Concepts: Data, Databases, Database Schema, Overview of the JDBC Process, Database Connection.

Learning Outcomes:

Student will be able to:

- Organize records in Record store.
- Search and sort single data type records in a Record store.
- Search and sort Mixed data type records in a Record store.
- Explain about database schema.
- Load the JDBC driver and establish a connection to database.
- Create and Execute an SQL statement.

Record Management System

Record Storage:

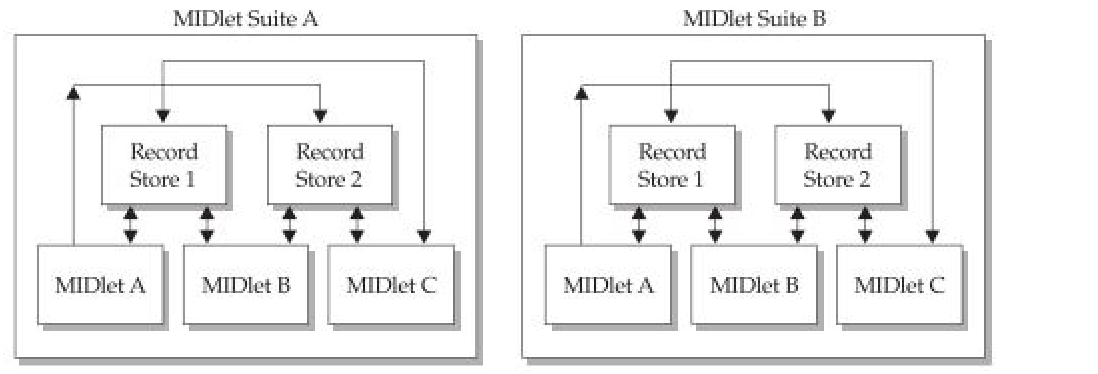
- Many operating environments contain a file system that is used to store information in nonvolatile resources such as a CD-ROM and disk drive.
- Not all small computing devices have a file system and therefore are unable to store information.
- The **Record Management System (RMS)** provides a file system–like environment that is used to store and maintain persistence in a small computing device.
- **RMS** is a combination file system and database management system that enables to store data in columns and rows similar to the organization of data in a table of a database.
- We can use RMS to perform the functionality of database management software (DBMS). That is, we can insert records, read records, search for particular records, and sort records stored by the RMS.
- Although RMS provides database functionality, but RMS is not a relational database, and therefore we cannot use SQL to interact with the data.
- Instead, we'll use the RMS application programming interface and the enumeration application programming interface to sort, search, and otherwise manipulate information stored in persistence.

The Record Store:

- RMS stores information in a **record store.**
- A **record store** compares to a flat file used for data storage in a traditional file system and to a table of a database.
- A record store is a collection of records organized as rows (records) and columns (fields).
- RMS assigns to each row a unique integer that identifies the row in the record store, which is called the record ID.
- Conceptually we can visualize a record store as rows and columns, technically there are two columns. The first column is the record ID, and the other column is an array of bytes that contains the persistent data.

Record Store Scope:

- We can create multiple record stores as required by our MIDlet as long as the name of each record store is unique.
- The name of a record store must be a minimum of one character and not more than 32 characters. Characters are Unicode, and the name is case sensitive.
- Record stores can be shared among MIDlets that are within the same MIDlet suite.
- Record stores must be uniquely named within a MIDlet suite, although duplicate names can be used for record stores in other MIDlet suites.



-1. *MIDlets within MIDlet suite A cannot access record stores collected in MIDlet suite B.*

<u>Setting up a Record Store:</u>

- The **openRecordStore ()** method is called to create a new record store and to open an existing record store.
- This method creates or opens a record store depending on whether the record store already exists within the MIDlet suite.
- The **openRecordStore ()** method requires two parameters. The first parameter is a string containing the name of the record store.
- The second parameter is a boolean value indicating whether the record store should be created if the record store doesn't exist.
  - ➢ A true value causes the record store to be created if the record store isn't in the MIDlet suite and also opens the record store.
  - ➢ A false value does not create the record store if the record store isn't located.
- We close a record store by calling the **closeRecordStore()** method. The **closeRecordStore()** method does not require any parameters.
- A record store remains in nonvolatile memory even after the small computing device is powered down.
- We can manage nonvolatile memory by removing all record stores that are no longer being used by MIDlets running on the device.
- A record store can be deleted by calling the deleteRecordStore() method.
  - ➢ This method requires one parameter, which is a string containing the name of the record store that is to be removed from the device.

<u>Writing and Reading Records:</u>

- Once our MIDlet opens a record store, the MIDlet can write records to the record store and read information already stored there using one of two techniques for writing and reading records.

- ➢ The first technique is used to write and read a string of data and is used primarily whenever you have one data column in the record store.
- ➢ The other technique is used to write and read multiple columns of data of different types such as string, integer, and boolean.

Let’s we discuss the technique for **writing a string to a record store**.

addRecord():

- The addRecord() method is used to write a record to the record store.
- The addRecord() method requires three parameters.
  - ➢ The first parameter is a byte array containing the byte value of the string being written to the record store.
  - ➢ The second parameter is an integer representing the index of the first byte of the byte array that is to be written to the record store.
  - ➢ The third parameter is the total number of bytes that is to be written to the record store.
- The first step in writing a string to a record store is to create an instance of a String and assign text to the instance. Next, the string must be converted to a byte array by calling the getBytes() method, as shown here. The getBytes() method returns a byte array.

string.getBytes()

- The second parameter of the addRecord() method is usually zero, and the third parameter is the length of the byte array, indicating that the entire byte array should be written to the record store.

getNumRecords():

- Our MIDlet needs to know the number of records in a record store in order to read all the records from the record store.
- The **getNumRecords()** method of the RecordStore class returns an integer that represents the total number of records in the record store.

getRecord():

- The **getRecord()** method returns bytes from the RecordStore, which are stored in a byte array that you create.
- The getRecord() method requires three parameters.

- The first parameter is the record ID, as described earlier in this chapter.
- The second parameter is the byte array that you create for storing the record.
- The third parameter is an integer representing the position in the record from which to begin copying into the byte array.

Creating a New Record and Reading an Existing Record:

Writing and Reading String-Based Records

Here are the steps required to write and read string-based records:

1. Declare references to classes.
2. Create instances of classes and assign those instances to references.
3. Open a record store and create a new record store if the record store doesn't exist.
4. Display any errors that occur when opening/creating a record store.
5. Create data in the form of a string.
6. Convert data to a byte array.
7. Write the byte array to the record store.
8. Create a byte array before reading data from the record store.
9. Determine the number of records in the record store.
10. Loop through each record in the record store.
11. Determine whether the size of the current record exceeds the length of the byte array. If so, then create a new byte array large enough to hold the record.
12. Copy the current record from the record store to the byte array.
13. Convert the byte array to a string and display the string on the screen.
14. Close the record store and display any errors that occur as the record store is closed.
15. Remove the record store and display any errors that occur as the record store is being removed.

```
import javax.microedition.rms.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
public class WriteReadExample extends MIDlet implements CommandListener
{
```

```
private Display display;
private Alert alert;
private Form form;
private Command exit;
private Command start;
private RecordStore recordstore = null;
public WriteReadExample()
{
        display = Display.getDisplay(this);
        exit = new Command("Exit", Command.SCREEN, 1);
        start = new Command("Start", Command.SCREEN, 1);
        form = new Form("Record");
        form.addCommand(exit);
        form.addCommand(start);
        form.setCommandListener(this);
}
public void startApp()
{
        display.setCurrent(form);
}
public void pauseApp()
{
}
public void destroyApp( boolean unconditional )
{
}
public void commandAction(Command command, Displayable displayable)
{
        if (command == exit)
        {
                destroyApp(true);
                notifyDestroyed();
        }
        else if (command == start)
        {
        try
        {
        recordstore= RecordStore.openRecordStore("myRecordStore", true );
        }
        catch (Exception error)
```

```
                {
            alert=new Alert("Error Creating",error.toString(), null, AlertType.WARNING);
            alert.setTimeout(Alert.FOREVER);
            display.setCurrent(alert);
                }
                try
                {
                        String outputData = "First Record";
                        byte[] byteOutputData = outputData.getBytes();
                recordstore.addRecord(byteOutputData, 0,byteOutputData.length);
                }
                catch ( Exception error)
                {
            alert = new Alert("Error Writing",error.toString(), null, AlertType.WARNING);
            alert.setTimeout(Alert.FOREVER);
            display.setCurrent(alert);
                }
                try
                {
                        byte[] byteInputData = new byte[1];
                        int length = 0;
                        for(intx=1;x<=recordstore.getNumRecords(); x++)
                        {
                        if (recordstore.getRecordSize(x) > byteInputData.length)
                        {
                        byteInputData = new byte[recordstore.getRecordSize(x)];
                        }
                        length = recordstore.getRecord(x, byteInputData, 0);
                        }
        alert = new Alert("Reading", new String(byteInputData, 0,
                                                length), null, AlertType.WARNING);
            alert.setTimeout(Alert.FOREVER);
            display.setCurrent(alert);
        }
        catch (Exception error)
        {
        alert = new Alert("Error Reading", error.toString(),
        null, AlertType.WARNING);
        alert.setTimeout(Alert.FOREVER);
        display.setCurrent(alert);
```

```
}
try
{
recordstore.closeRecordStore();
}
catch (Exception error)
{
alert = new Alert("Error Closing", error.toString(),
null, AlertType.WARNING);
alert.setTimeout(Alert.FOREVER);
display.setCurrent(alert);
}
if (RecordStore.listRecordStores() != null)
{
try
{
RecordStore.deleteRecordStore("myRecordStore");
}
catch (Exception error)
{
alert = new Alert("Error Removing", error.toString(),
null, AlertType.WARNING);
alert.setTimeout(Alert.FOREVER);
display.setCurrent(alert);
}
}
}
}
```

Writing and Reading Mixed Data Types:

- It is common for records to consist of mixed data types such as string, boolean, and integer.
- For example, We might store the customer name, customer number, and gender. Astring is used to store a customer name, an integer to store the customer number, and a boolean to indicate gender.

Writing and Reading Mixed Data Type Records

Here are the steps required to write and read mixed data type records:

1) Declare references to classes.
2) Create instances of classes and assign those instances to references.
3) Open a record store and create a new record store if the record store doesn’t exist.
4) Display any errors that occur when opening/creating a record store.
5) Create data in the appropriate data type.
6) Convert data to a byte array output stream.
7) Create a data output stream using the byte array output stream.
8) Write each column of the record to the data output stream.
9) Convert the data output stream to a byte array.
10) Write the record to the record store.
11) Close the output byte array output stream and the data output stream.
12) Display any errors that might occur while writing to the record store.
13) Create a buffer of bytes sufficient to hold a record.
14) Create a byte array input stream and a data input stream.
15) Loop through the record store, copying each column from the record store to a variable.
16) Display the data in a dialog box.
17) Display any errors that occur when reading records from the record store.
18) Close and remove the record store.

```
import javax.microedition.rms.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
public class WriteReadMixedDataTypesExample extends MIDlet implements
                                                            CommandListener
{
        private Display display;
        private Alert alert;
        private Form form;
        private Command exit;
        private Command start;
        private RecordStore recordstore = null;
        public WriteReadMixedDataTypesExample ()
        {
                display = Display.getDisplay(this);
                exit = new Command("Exit", Command.SCREEN, 1);
                start = new Command("Start", Command.SCREEN, 1);
                form = new Form("Mixed Record");
                form.addCommand(exit);
                form.addCommand(start);
                form.setCommandListener(this);
        }
        public void startApp()
        {
                display.setCurrent(form);
        }
        public void pauseApp()
        {
        }
        public void destroyApp( boolean unconditional )
        {
        }

        public void commandAction(Command command, Displayable displayable)
        {
                if (command == exit)
                {
                        destroyApp(true);
                        notifyDestroyed();
                }
                else if (command == start)
                {
                try
                {
                recordstore = RecordStore.openRecordStore( "myRecordStore", true );
                }
                catch (Exception error)
                {
        alert = new Alert("Error Creating", error.toString(), null, AlertType.WARNING);
        alert.setTimeout(Alert.FOREVER);
        display.setCurrent(alert);
        }
        try
        {
                byte[] outputRecord;
```

```
                String outputString = "First Record";
                int outputInteger = 15;
                boolean outputBoolean = true;
                ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        DataOutputStream outputDataStream =new DataOutputStream(outputStream);
        outputDataStream.writeUTF(outputString);
        outputDataStream.writeBoolean(outputBoolean);
        outputDataStream.writeInt(outputInteger);
        outputDataStream.flush();
        outputRecord = outputStream.toByteArray();
        recordstore.addRecord(outputRecord, 0, outputRecord.length);
        outputStream.reset();
        outputStream.close();
        outputDataStream.close();
        }
        catch ( Exception error)
        {
                alert = new Alert("Error Writing",
                error.toString(), null, AlertType.WARNING);
                alert.setTimeout(Alert.FOREVER);
                display.setCurrent(alert);
        }
        try
        {
                String inputString = null;
                int inputInteger = 0;
                boolean inputBoolean = false;
                byte[] byteInputData = new byte[100];
ByteArrayInputStream inputStream = new ByteArrayInputStream(byteInputData);
DataInputStream inputDataStream =new DataInputStream(inputStream);
for (int x = 1; x <= recordstore.getNumRecords(); x++)
{
        recordstore.getRecord(x, byteInputData, 0);
        inputString = inputDataStream.readUTF();
        inputBoolean = inputDataStream.readBoolean();
        inputInteger = inputDataStream.readInt();
        inputStream.reset();
}
inputStream.close();
inputDataStream.close();
```

```
alert = new Alert("Reading", inputString + " " +inputInteger + " " +inputBoolean, null,
                                                    AlertType.WARNING);
alert.setTimeout(Alert.FOREVER);
display.setCurrent(alert);
}
catch (Exception error)
{
alert = new Alert("Error Reading",
error.toString(), null, AlertType.WARNING);
alert.setTimeout(Alert.FOREVER);
display.setCurrent(alert);
}
try
{
recordstore.closeRecordStore();
}
catch (Exception error)
{
alert = new Alert("Error Closing",
error.toString(), null, AlertType.WARNING);
alert.setTimeout(Alert.FOREVER);
display.setCurrent(alert);
}
if (RecordStore.listRecordStores() != null)
{
try
{
RecordStore.deleteRecordStore("myRecordStore");
}
catch (Exception error)
{
        alert = new Alert("Error Removing",
        error.toString(), null, AlertType.WARNING);
        alert.setTimeout(Alert.FOREVER);
        display.setCurrent(alert);
}
}
}
}
}
```

Record Enumeration:

- A record store is more like a flat file than a database management system and therefore lacks many sophisticated features that you find in a database management system.
- For example, we cannot send an SQL query to a record store, nor can we ask a record store to search for keywords or sort records, which is commonly performed by a database management system.
- However, we can still perform searches and sorts of records in a record store by using the RecordEnumeration interface.
  - ➢ An Enumeration provides a way to traverse data elements.
  - ➢ The Enumeration object manages how data is retrieved from the record store.
  - ➢ Changes to the record store are reflected when the record store's content is iterated.
- We obtain a record enumeration by calling the enumerateRecords() method.
- The enumerateRecords() method requires three parameters.
  - ➢ The first is the record filter used to exclude records returned from the record store.
  - ➢ The second is reference to the record comparator, which is a method used to compare records returned from the record store.
  - ➢ The last parameter is a boolean value indicating whether or not the enumeration is automatically updated when changes are made to the underlying record store.

RecordEnumeration recordEnumeration= Recordstore.enumerateRecords (null, null, false);

- There isn't any filter or comparator method, and the record enumeration is not automatically updated when a change is made to the record store**.**
- You then use methods of the RecordEnumeration to interact with records in the RecordEnumeration.
- One of the most common interactions that you'll have with a RecordEnumeration is to step through each record of the RecordEnumeration.
- The **hasNextElement()** method is called to evaluate whether or not there is another record in the RecordEnumeration.

- Aboolean true is returned if another record exists; otherwise, a boolean false is returned.

```
while ( recordEnumeration.hasNextElement())
{
        //do something
}
```

- You can retrieve a record from the RecordEnumeration using one of two techniques.
  - The first technique is designed to read a record that has a single data type such as a string from the RecordEnumeration.
  - The other technique reads a record that has a compound data type.
- The **nextRecord()** method, which returns a copy of the next record in the RecordEnumeration. The record is passed to the constructor of the String class and is assigned to the string variable.

```
String string = new String(recordEnumeration.nextRecord());
```

- You can move forward or back within the RecordEnumeration by calling the nextRecord() method or previousRecord()
  - The nextRecord() method is used to move to the next record,
  - The previousRecord() method, which moves back one record.
- Both the nextRecord() method and the previousRecord() method return a byte array containing a copy of the record.

Record Enumeration:

- A record store is more like a flat file than a database management system and therefore lacks many sophisticated features that we find in a database management system.
  - For example, we cannot send an SQL query to a record store, nor can we ask a record store to search for keywords or sort records.
- However, we can still perform searches and sorts of records in a record store by using the **RecordEnumeration interface.**
- An **Enumeration** provides a way to traverse data elements.
- Changes to the record store are reflected when the record store’s content is iterated.
- We obtain a record enumeration by calling the **enumerateRecords ()** method.

- The **enumerateRecords ()** method requires three parameters.
    - ➢ The first is the record filter used to exclude records returned from the record store.
    - ➢ The second is reference to the record comparator, which is a method used to compare records returned from the record store.
    - ➢ The last parameter is a boolean value indicating whether or not the enumeration is automatically updated when changes are made to the underlying record store.

RecordEnumeration recordEnumeration = recordstore.enumerateRecords (null, null, false);

- The enumerateRecords() method returns a RecordEnumeration.
- There isn't any filter or comparator method, and the record enumeration is not automatically updated when a change is made to the record store.
- You then use methods of the RecordEnumeration to interact with records in the RecordEnumeration.
- One of the most common interactions that you'll have with a RecordEnumeration is to step through each record of the RecordEnumeration.
- The **hasNextElement()** method is called to evaluate whether or not there is another record in the RecordEnumeration.
- You can retrieve a record from the RecordEnumeration using one of two techniques.
    - ➢ The first technique is designed to read a record that has a single data type such as a string from the RecordEnumeration.
    - ➢ The other technique reads a record that has a compound data type.

nextRecord():

- The nextRecord() method, which returns a copy of the next record in the RecordEnumeration.
- The record is passed to the constructor of the String class and is assigned to the string variable.

String string = new String(recordEnumeration.nextRecord());

- The record is passed to the constructor of the String class and is assigned to the string variable.

- We can move forward or back within the RecordEnumeration by calling either the **nextRecord()** method, which moves to the next record, or the **previousRecord()** method, which moves back one record. Both the **nextRecord()** method and the **previousRecord()** method return a byte array containing a copy of the record.

When we create a RecordEnumeration, it is positioned at the top. The top is not the first record.

- We must call the **nextRecord()** method to move to the first record.
- We can move to the last record by calling the **previousRecord()** method while at the top of the RecordEnumeration.
- We can return to the top of the RecordEnumeration by calling the **reset()** method.
- **numRecords()** method is used to determine the number of records there are in the RecordEnumeration.
    - ➢ The **numRecords()** method returns an integer representing the total number of records.
- **nextRecordId()** method is used to determine the record ID of the next record.
- **previousRecordId()** method is used to determine the record ID of the previous record.
- The **keepUpdated()** method is used to set automatic updating of the RecordEnumeration.
- The **keepUpdated()** method has one parameter, which is a boolean value indicating whether or not the RecordEnumeration is automatically updated.
- We can check the status of the automatic updating feature by calling the **isKeptUpdated()** method. This method returns a boolean value indicating whether or not the RecordEnumeration is automatically updated.
- We can manually cause the RecordEnumeration to be rebuilt by calling the **rebuild()** method. The **rebuild()** method should be called whenever records in the underlying record store change and the automatic update feature is deactivated.
- We can call the **destroy** method to empty the contents of a RecordEnumeration and release resources used by the RecordEnumeration. This should be done as

soon as the MIDlet no longer requires the RecordEnumeration in order to free those resources for other purposes.

Reading a Record of a Simple Data type into a RecordEnumeration:

Here are the steps required to read a record of a string data type into a RecordEnumeration:

- Declare references to classes.
- Create instances of classes and assign those instances to references.
- Open a record store and create a new record store if the record store doesn't exist.
- Display any errors that occur when opening/creating a record store.
- Create data in the appropriate data type.
- Convert data to a byte array.
- Write the record to the record store.
- Display any errors that might occur while writing to the record store.
- Create a RecordEnumeration.
- Loop through the RecordEnumeration, copying each record to a variable.
- Display the data in a dialog box.
- Display any errors that occur when reading records from the RecordEnumeration.
- Close and remove the RecordEnumeration and the record store.

```
import javax.microedition.rms.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;
public class RecordEnumerationExample extends MIDlet implements CommandListener
{
        private Display display;
        private Alert alert;
        private Form form;
        private Command exit;
        private Command start;
        private RecordStore recordstore = null;
        private RecordEnumeration recordEnumeration = null;
        public RecordEnumerationExample ()
        {
```

```
        display = Display.getDisplay(this);
        exit = new Command("Exit", Command.SCREEN, 1);
        start = new Command("Start", Command.SCREEN, 1);
        form = new Form("RecordEnumeration");
        form.addCommand(exit);
        form.addCommand(start);
        form.setCommandListener(this);
}
public void startApp()
{
        display.setCurrent(form);
}
public void pauseApp()
{
}
public void destroyApp( boolean unconditional )
{
}
public void commandAction(Command command,   Displayable displayable)
{
        if (command == exit)
        {
        }
        destroyApp(true);
        notifyDestroyed();
        else if (command == start)
        {
        try
        {
        recordstore = RecordStore.openRecordStore("myRecordStore", true );
        }
        catch (Exception error)
        {
                alert = new Alert("Error Creating",
                error.toString(), null, AlertType.WARNING);
                alert.setTimeout(Alert.FOREVER);
                display.setCurrent(alert);
```

```
          }
          try
          {
          String outputData[] = {"First Record","Second Record", "Third Record"};
          for(intx=0;x<3;x++)
          {
                    byte[] byteOutputData = outputData[x].getBytes();
          recordstore.addRecord(byteOutputData, 0, byteOutputData.length);
          }
          }
          catch ( Exception error)
          {
alert = new Alert("Error Writing",error.toString(), null, AlertType.WARNING);
alert.setTimeout(Alert.FOREVER);
display.setCurrent(alert);
          }
          try
          {
                    StringBuffer buffer = new StringBuffer();
          recordEnumeration =recordstore.enumerateRecords(null, null, false);

          while (recordEnumeration.hasNextElement())
          {
                    buffer.append(new String(recordEnumeration.nextRecord()));
                    buffer.append("\n");
          }
alert = new Alert("Reading",buffer.toString(), null, AlertType.WARNING);
alert.setTimeout(Alert.FOREVER);
display.setCurrent(alert);
}
catch (Exception error)
```

```
        {
alert = new Alert("Error Reading",    error.toString(), null, AlertType.WARNING);
        alert.setTimeout(Alert.FOREVER);
        display.setCurrent(alert);
        }
        try
        {
                recordstore.closeRecordStore();
        }
        catch (Exception error)
        {
        alert = new Alert("Error Closing",
        error.toString(), null, AlertType.WARNING);
        alert.setTimeout(Alert.FOREVER);
        display.setCurrent(alert);
        }
        if (RecordStore.listRecordStores() != null)
        {
        try
        {
        RecordStore.deleteRecordStore("myRecordStore");
        recordEnumeration.destroy();
        }
        catch (Exception error)
        {
alert = new Alert("Error Removing", error.toString(), null, AlertType.WARNING);
        alert.setTimeout(Alert.FOREVER);
        display.setCurrent(alert);
        }
}
}
```

```
}
}
```

Reading a Mixed Data Type Record into a RecordEnumeration:

Here are the steps required to read a mixed data type record into a RecordEnumeration:

1. Declare references to classes.
2. Create instances of classes and assign those instances to references.
3. Open a record store and create a new record store if the record store doesn’t exist.
4. Display any errors that occur when opening/creating a record store.
5. Create data in the appropriate data type.
6. Convert data to a byte array output stream.
7. Create a data output stream using the byte array output stream.
8. Write each column of the record to the data output stream.
9. Convert the data output stream to a byte array.
10. Write the record to the record store.
11. Close the output byte array output stream and the data output stream.
12. Display any errors that might occur while writing to the record store.
13. Create a buffer of bytes sufficient to hold a record.
14. Create a byte array input stream and a data input stream.
15. Create a RecordEnumeration.
16. Loop through the RecordEnumeration, copying each column from the record store to a variable.
17. Display the data in a dialog box.
18. Display any errors that occur when reading records from the record store.
19. Close and remove the RecordEnumeration and the record store.

Program:

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.midlet.MIDlet;
import javax.microedition.rms.RecordStore;

public class J2MEWriteReadMixedDataTypesExample extends MIDlet implements
CommandListener {
  private Display display;

  private Alert alert;

  private Form form = new Form("Mixed Record");

  private Command exit = new Command("Exit", Command.SCREEN, 1);

  private Command start = new Command("Start", Command.SCREEN, 1);

  private RecordStore recordstore = null;

  public J2MEWriteReadMixedDataTypesExample() {
    display = Display.getDisplay(this);
    form.addCommand(exit);
    form.addCommand(start);
    form.setCommandListener(this);
  }

  public void startApp() {
    display.setCurrent(form);
  }

  public void pauseApp() {
  }

  public void destroyApp(boolean unconditional) {
  }

  public void commandAction(Command command, Displayable displayable) {
    if (command == exit) {
```

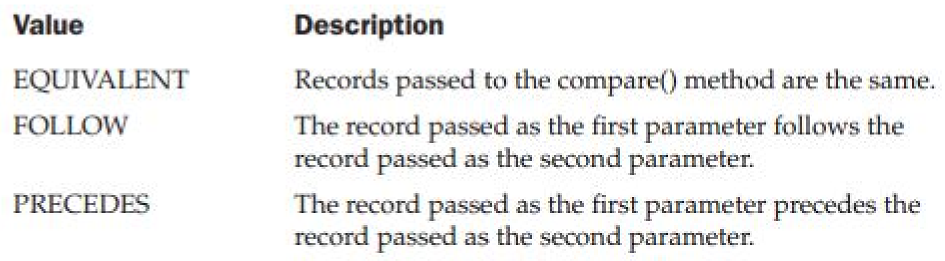
```
  destroyApp(true);
  notifyDestroyed();
 } else if (command == start) {
  try {
   recordstore = RecordStore.openRecordStore("myRecordStore", true);
   byte[] outputRecord;
   String outputString = "First Record";
   int outputInteger = 15;
   boolean outputBoolean = true;
   ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
   DataOutputStream outputDataStream = new DataOutputStream(outputStream);
   outputDataStream.writeUTF(outputString);
   outputDataStream.writeBoolean(outputBoolean);
   outputDataStream.writeInt(outputInteger);
   outputDataStream.flush();
   outputRecord = outputStream.toByteArray();
   recordstore.addRecord(outputRecord, 0, outputRecord.length);
   outputStream.reset();
   outputStream.close();
   outputDataStream.close();
   String inputString = null;
   int inputInteger = 0;
   boolean inputBoolean = false;
   byte[] byteInputData = new byte[100];
   ByteArrayInputStream inputStream = new ByteArrayInputStream(byteInputData);
   DataInputStream inputDataStream = new DataInputStream(inputStream);
   for (int x = 1; x <= recordstore.getNumRecords(); x++) {
    recordstore.getRecord(x, byteInputData, 0);
    inputString = inputDataStream.readUTF();
    inputBoolean = inputDataStream.readBoolean();
    inputInteger = inputDataStream.readInt();
    inputStream.reset();
   }
   inputStream.close();
   inputDataStream.close();
alert = new Alert("Reading", inputString + " " + inputInteger + " " + inputBoolean, null,
                                                      AlertType.WARNING);
   alert.setTimeout(Alert.FOREVER);
   display.setCurrent(alert);
   recordstore.closeRecordStore();
```

```
        if (RecordStore.listRecordStores() != null) {
          RecordStore.deleteRecordStore("myRecordStore");
        }
      } catch (Exception error) {
        alert = new Alert("Error Removing", error.toString(), null, AlertType.WARNING);
        alert.setTimeout(Alert.FOREVER);
        display.setCurrent(alert);
      }
    }
  }

}
```

Sorting Records

- Records within a RecordEnumeration are sorted by defining a **comparator class** that is an implementation of the RecordComparator interface.
- Within the comparator class, we define a method that has the logic to compare each record to determine whether the record is equal to the current record or should precede or follow the current record within the RecordEnumeration.
- The method compare**(),** requires two parameters, which are two byte arrays that contain the current record and the next record.
    - **int compare(byte[], rec1, byte[] rec2)-** Compare two records represented as byte array rec1 and byte array rec2 to determine the sort sequence of a RecordEnumeration.
- These byte arrays are then converted to two strings that are compared by using the **compareTo()** method of the String class:The compareTo() method returns an integer that is equal to zero, less than zero, or greater than zero.
    - A **zero** indicates that both strings are the same.
    - An integer **less than zero** indicates that the next record precedes the current record in the RecordEnumeration.
    - An integer **greater than zero** indicates that the next record follows the current record in the RecordEnumeration
- Based on the return value of the compareTo() method, the compare() method returns a predefined comparison value. These are:

- ➢ RecordComparator.EQUIVALENT
- ➢ RecordComparator.PRECEDES
- ➢ RecordComparator.FOLLOW

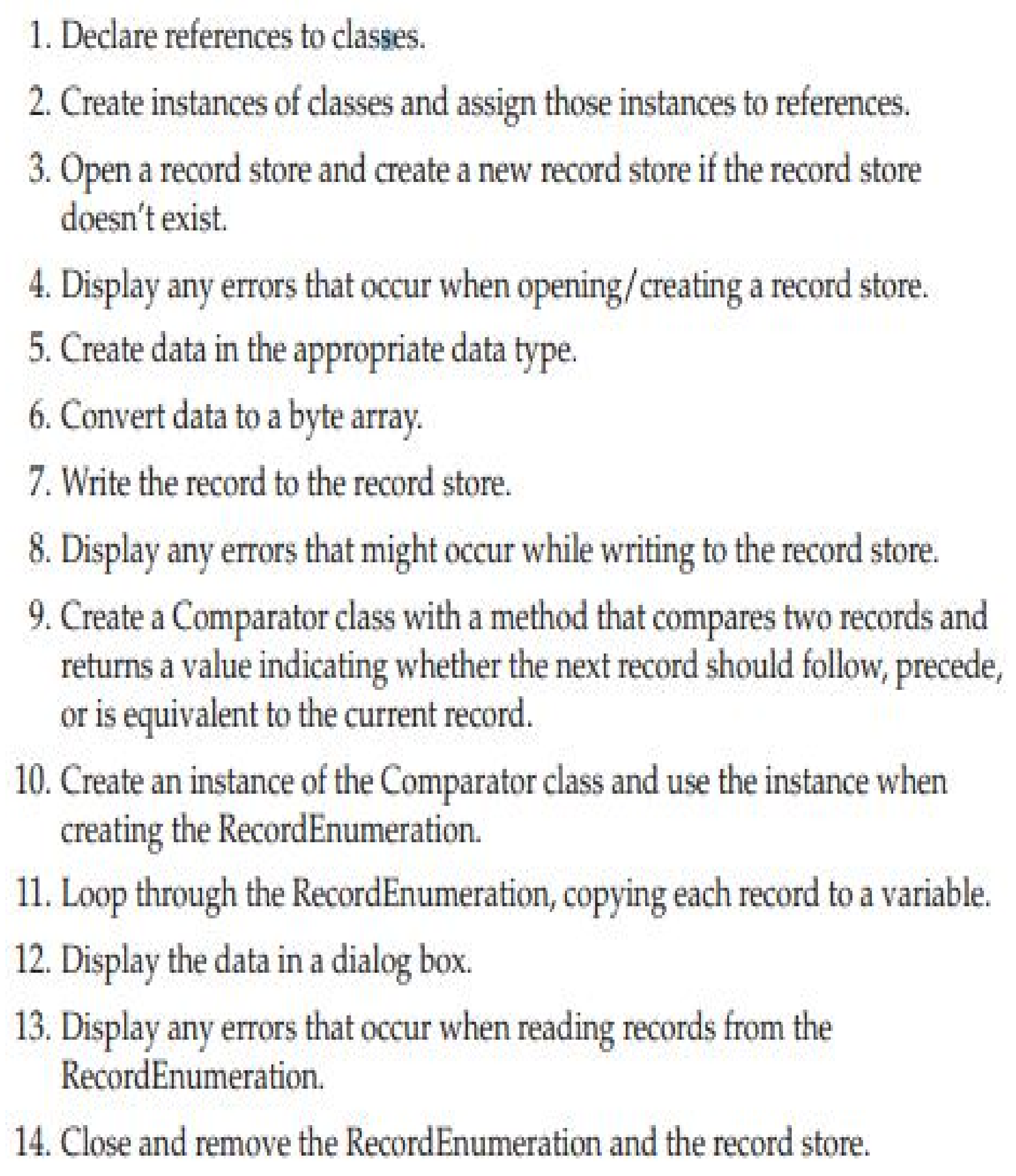


Value	Description
EQUIVALENT	Records passed to the compare() method are the same.
FOLLOW	The record passed as the first parameter follows the record passed as the second parameter.
PRECEDES	The record passed as the first parameter precedes the record passed as the second parameter.

- We pass reference to the instance of the RecordComparator() as the second parameter of the enumerateRecords() method. The enumerateRecords() then calls the compare() method whenever there is a need to sort records within the RecordEnumerator.
- The direction of the sort is controlled by the logic that you create within the compare() method.
  - ➢ If you want the sort to appear in ascending order, then return the **RecordComparator .PRECEDES when the return value of the compareTo() string is less than the current record , RecordComparator.FOLLOW when the return value is greater than the current record.**
  - ➢ If you want the sort to appear in descending order, then return the **RecordComparator.FOLLOW when the return value of the compareTo() string is less than the current record and RecordComparator.PRECEDES when the return value is greater than the current record**.

Sorting Single Data Type Records in a RecordEnumeration:

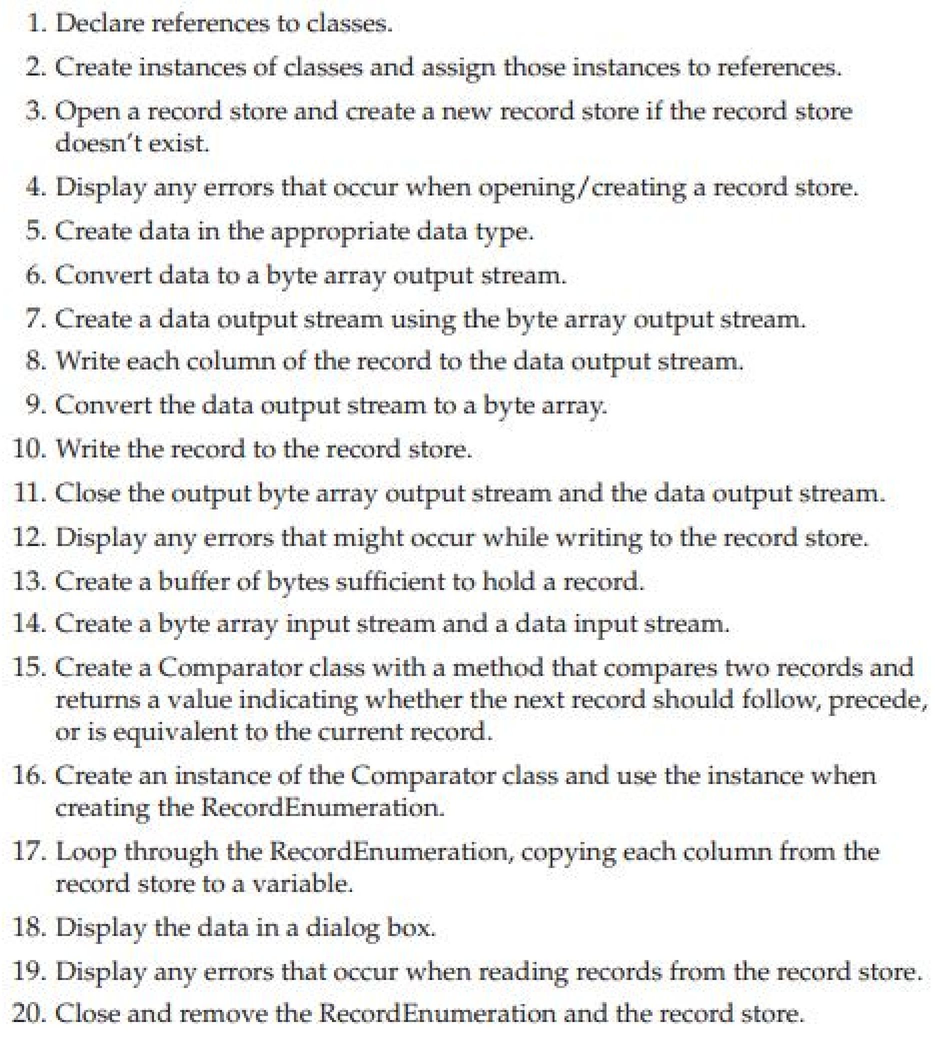
- The following are the steps to sort single type Records in a RecordEnumeration:



1. Declare references to classes.
2. Create instances of classes and assign those instances to references.
3. Open a record store and create a new record store if the record store doesn't exist.
4. Display any errors that occur when opening/creating a record store.
5. Create data in the appropriate data type.
6. Convert data to a byte array.
7. Write the record to the record store.
8. Display any errors that might occur while writing to the record store.
9. Create a Comparator class with a method that compares two records and returns a value indicating whether the next record should follow, precede, or is equivalent to the current record.
10. Create an instance of the Comparator class and use the instance when creating the RecordEnumeration.
11. Loop through the RecordEnumeration, copying each record to a variable.
12. Display the data in a dialog box.
13. Display any errors that occur when reading records from the RecordEnumeration.
14. Close and remove the RecordEnumeration and the record store.

Sorting Mixed Data Type Records in a RecordEnumeration:

- The following are the steps to sort Mixed type Records in a RecordEnumeration:



1. Declare references to classes.
2. Create instances of classes and assign those instances to references.
3. Open a record store and create a new record store if the record store doesn't exist.
4. Display any errors that occur when opening/creating a record store.
5. Create data in the appropriate data type.
6. Convert data to a byte array output stream.
7. Create a data output stream using the byte array output stream.
8. Write each column of the record to the data output stream.
9. Convert the data output stream to a byte array.
10. Write the record to the record store.
11. Close the output byte array output stream and the data output stream.
12. Display any errors that might occur while writing to the record store.
13. Create a buffer of bytes sufficient to hold a record.
14. Create a byte array input stream and a data input stream.
15. Create a Comparator class with a method that compares two records and returns a value indicating whether the next record should follow, precede, or is equivalent to the current record.
16. Create an instance of the Comparator class and use the instance when creating the RecordEnumeration.
17. Loop through the RecordEnumeration, copying each column from the record store to a variable.
18. Display the data in a dialog box.
19. Display any errors that occur when reading records from the record store.
20. Close and remove the RecordEnumeration and the record store.

Searching Records

- Searching is referred to as **filtering**, where the filter is defined by the search criteria.
- Records that match the search criteria are copied into the RecordEnumeration. Those not matching the search criteria are filtered from the RecordEnumeration.
- The RecordFilter interface is used when searching for a record. You must define two methods when defining an implementation of the RecordFilter interface.

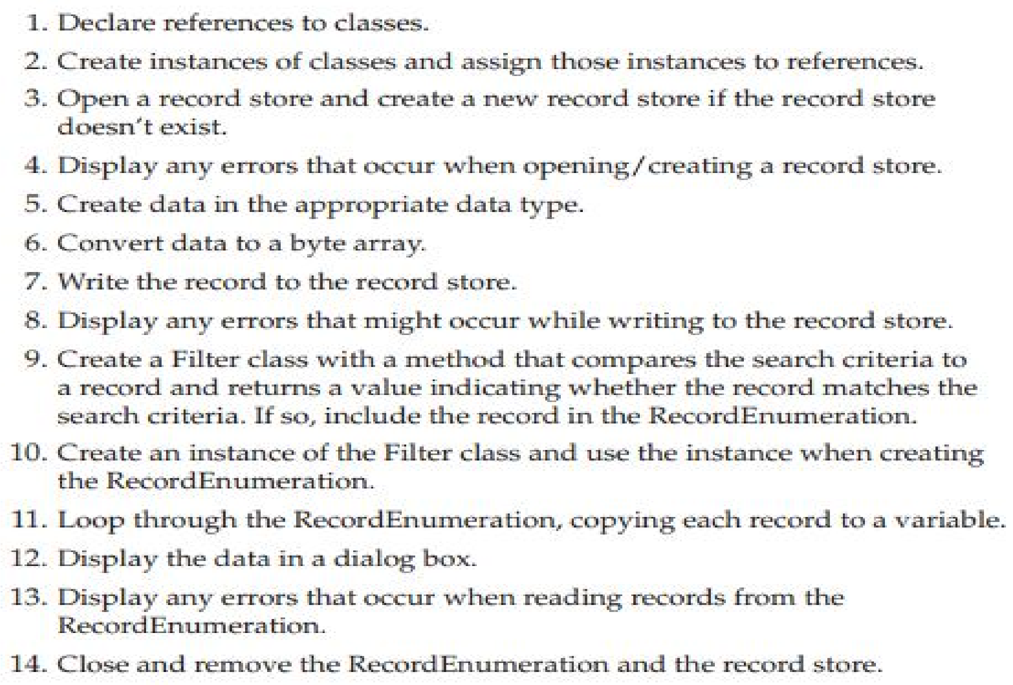
- **matches() method –**
  - contains the logic necessary to determine whether a column fits the search criteria and returns a boolean value indicating whether or not there is a match.
  - Logic contained in the matches() method reads one or multiple columns from the current record and then applies logical operators to determine whether the record meets the search criteria.
  - We determine the logic used to decide whether or not a record should or should not be included in the RecordEnumeration.
  - We can sort the filtered records by first searching for a subset of records in the record store, then sorting those records.

boolean matches(byte[] candidate)-Search a record for a specific value.

- **filterClose() method**-frees resources used by the implementation of the RecordFilter interface once the search is completed.

Searching Single Data Type Records:

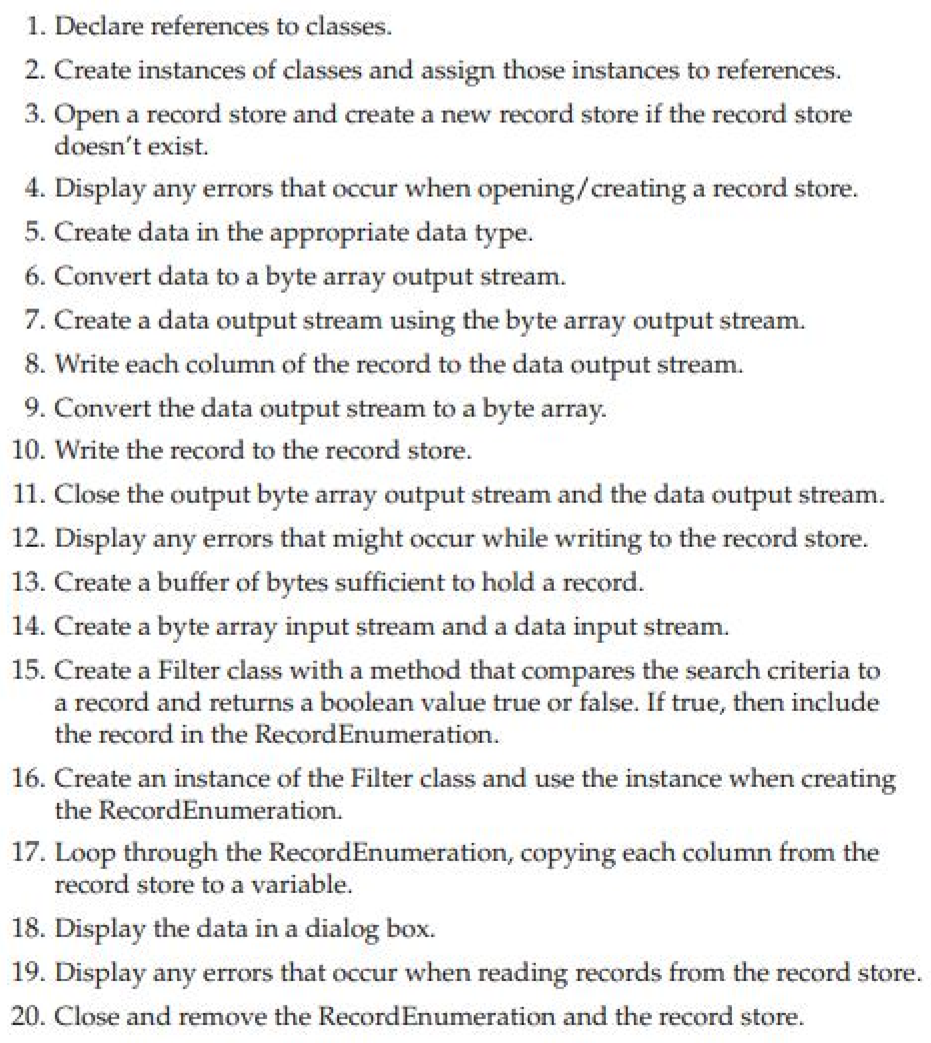
- The following are the steps required to search using a data type of a record:



1. Declare references to classes.
2. Create instances of classes and assign those instances to references.
3. Open a record store and create a new record store if the record store doesn't exist.
4. Display any errors that occur when opening/creating a record store.
5. Create data in the appropriate data type.
6. Convert data to a byte array.
7. Write the record to the record store.
8. Display any errors that might occur while writing to the record store.
9. Create a Filter class with a method that compares the search criteria to a record and returns a value indicating whether the record matches the search criteria. If so, include the record in the RecordEnumeration.
10. Create an instance of the Filter class and use the instance when creating the RecordEnumeration.
11. Loop through the RecordEnumeration, copying each record to a variable.
12. Display the data in a dialog box.
13. Display any errors that occur when reading records from the RecordEnumeration.
14. Close and remove the RecordEnumeration and the record store.

Searching Mixed Data Type Records:

- Here are the steps required to search a mixed data type record:



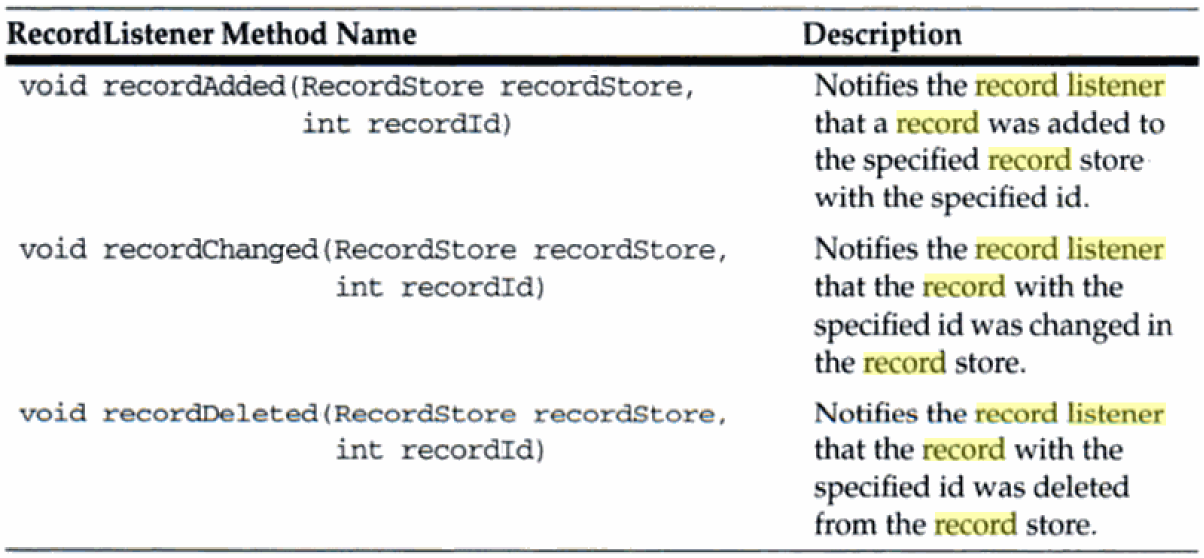
1. Declare references to classes.
2. Create instances of classes and assign those instances to references.
3. Open a record store and create a new record store if the record store doesn't exist.
4. Display any errors that occur when opening/creating a record store.
5. Create data in the appropriate data type.
6. Convert data to a byte array output stream.
7. Create a data output stream using the byte array output stream.
8. Write each column of the record to the data output stream.
9. Convert the data output stream to a byte array.
10. Write the record to the record store.
11. Close the output byte array output stream and the data output stream.
12. Display any errors that might occur while writing to the record store.
13. Create a buffer of bytes sufficient to hold a record.
14. Create a byte array input stream and a data input stream.
15. Create a Filter class with a method that compares the search criteria to a record and returns a boolean value true or false. If true, then include the record in the RecordEnumeration.
16. Create an instance of the Filter class and use the instance when creating the RecordEnumeration.
17. Loop through the RecordEnumeration, copying each column from the record store to a variable.
18. Display the data in a dialog box.
19. Display any errors that occur when reading records from the record store.
20. Close and remove the RecordEnumeration and the record store.

RecordListener:

- Applications have the ability to receive notifications whenever a record is added, removed or changed in a record store.
- The instance of the RecordListener interface is notified whenever one of three changes is made to the record store.
  - ➢ These are when a record is added, modified, or deleted from the record store.
- The RecordListener interface must define three methods:
  - ➢ recordAdded()

- ➢ recordChanged()
- ➢ recordDeleted()

- All three methods require two parameters.
  - ➢ The first parameter is reference to the record store that has changed,
  - ➢ The second is an integer indicating the record ID that was added, modified, or removed from the record store.



RecordListener Method Name	Description
`void recordAdded(RecordStore recordStore, int recordId)`	Notifies the record listener that a record was added to the specified record store with the specified id.
`void recordChanged(RecordStore recordStore, int recordId)`	Notifies the record listener that the record with the specified id was changed in the record store.
`void recordDeleted(RecordStore recordStore, int recordId)`	Notifies the record listener that the record with the specified id was deleted from the record store.

J2ME Database Concepts

DATA

- A CDC-based J2ME application interacts with commercial DBMSs by using a combination of Java data objects that are defined in the Java Database Connection (JDBC) specification and by using the Structured Query Language (SQL).
- The JDBC interface forms a communications link with a DBMS, while SQL is the language used to construct the message (called a query) that is sent to the DBMS to request, update, delete, and otherwise manipulate data in the DBMS.
- We use **data** in everyday life, such as when you dial a telephone number or log into a computer network using a user ID and password. The telephone number, user ID, and password are types of data.
- **Information** consists of one or more words that collectively infer a meaning, such as a person's address.

- Data refers to an atomic unit that is stored in a DBMS and is sometimes reassembled into information.
- Examples of data are a person's street address, city, state, and zip code. Each of these is an atomic unit that is commonly found in a DBMS.
- A J2ME application can access one or multiple atomic units as required by the application. Data is organized in a database so that a J2ME application can quickly find, retrieve, update, or delete one or more data elements.

DATABASES

- A database is a collection of data.
- Java and Java's IO classes can create a own database, or can interact with a commercially available DBMS.
- DBMSs use proprietary and public domain algorithms to assure fast and secure interaction with data stored in the database. Most DBMSs use widely accepted relational database model.
- A database model is a description of how data is organized in a database. In a relational database model, data is grouped into tables using a technique called normalization.
- Once a database and at least one table are created, a J2ME application can send SQL statements to the DBMS to perform the following:
  - ➢ Save data
  - ➢ Retrieve data
  - ➢ Update data
  - ➢ Manipulate data
  - ➢ Delete data

Tables:

- A table is the component of a database that contains data in the form of rows and columns.
- A row contains related data such as clients' names and addresses. A column contains like data such as clients' first names. Each column is identified by a unique name, called a column name that describes the data contained in the column.

- An attribute describes the characteristic of data that can be stored in the column. Attributes include size, data type, and format.

- Database name, table name, column name, column attributes, and other information that describe database components are known as **metadata**. Metadata is data about data.

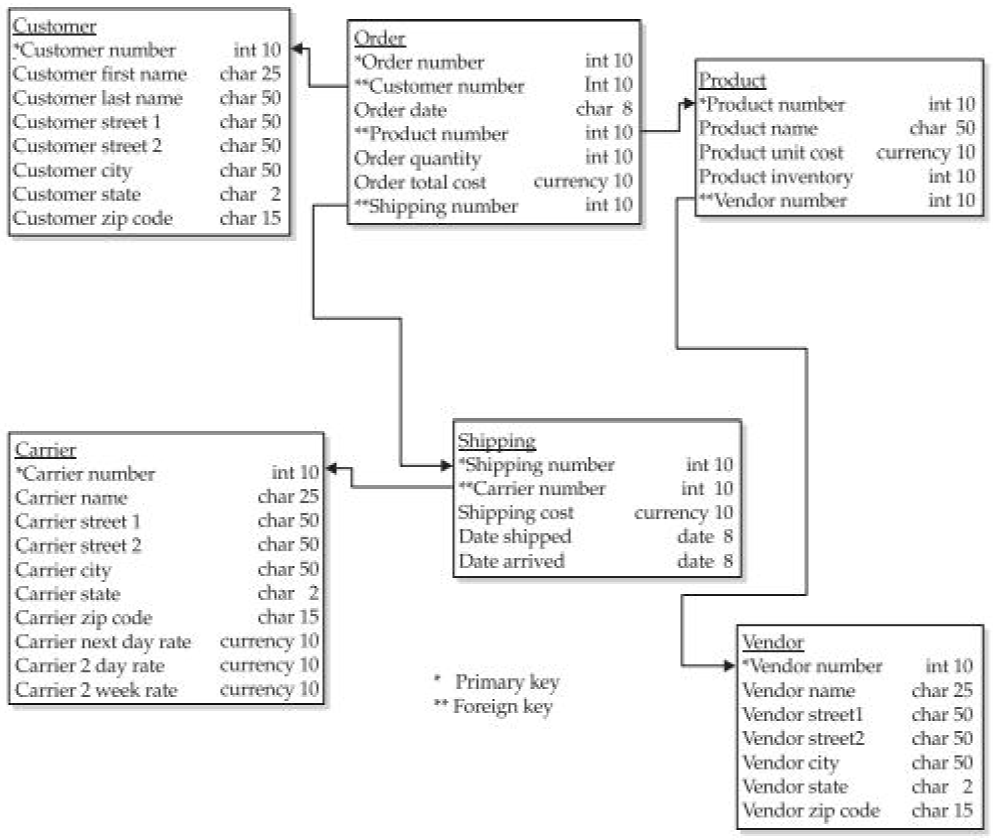
- Metadata is used by J2ME applications to identify database components without needing to know details of a column, table, or the database.

- A J2ME application can request from the DBMS the data type of a specific column. The column type is used by a J2ME application to copy data retrieved from the DBMS into a Java collection.

Database Schema

- A database schema is a document that defines all components of a database, such as tables, columns, and indexes.

- A database schema also shows relationships between tables; the relationships are used to join rows of two tables.



- To create a database schema, you must perform six steps:

1. Identify information used in the existing system or legacy system that is being replaced by the J2ME application.

2. Decompose this information into data.

3. Define data.

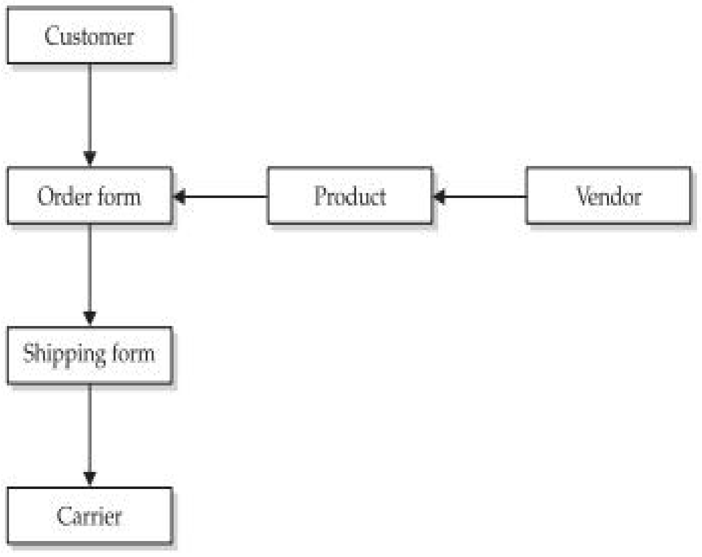
4. Normalize data into logical groups.

5. Create primary and foreign keys.

6. Group data together into logical groups.

1. Identifying Information:

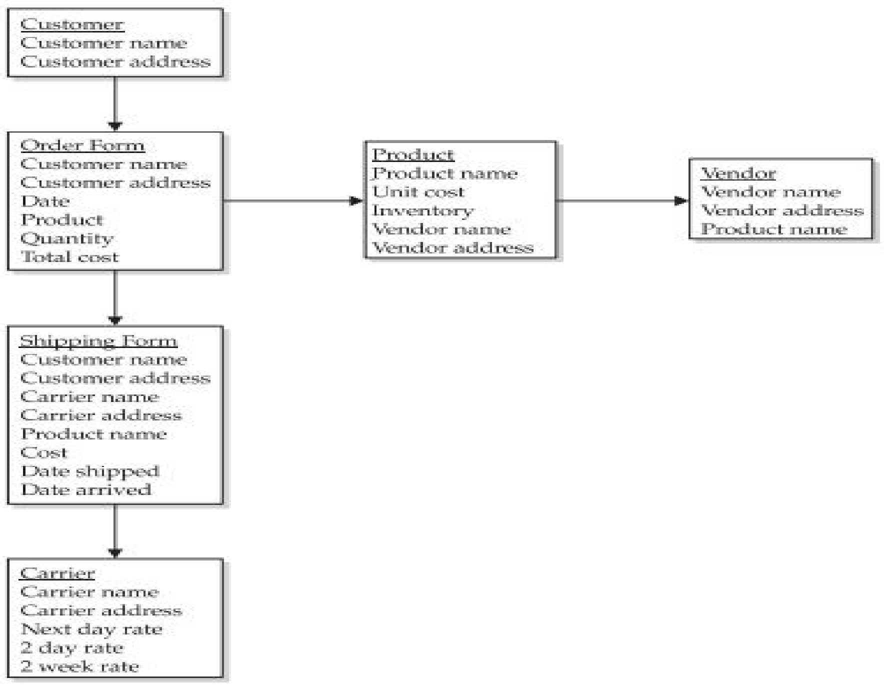
- The initial step in defining a database schema is to identify all information used by the system that is being converted to J2ME technology. Information is associated with objects—also known as **entities.**
- **Identifying Entity**. Each entity is defined by attributes. An attribute is information that describes an entity, such as a customer name for a customer entity. The following illustrates the entities for an order system:



- An entity attribute differs from data attributes .An entity attribute provides general information about an entity, while a data attribute provides information about data that is used by the entity.

- For example, a customer name is an entity attribute, and a customer first name and customer last name are data attributes.

- **Identifying attributes:** An attribute is information commonly used to describe an entity. For example, a customer name and address are information normally used to describe a customer. Therefore, customer name and address are easily recognizable as attributes of a customer entity.



- The best way to identify attributes of an entity is by analyzing instances of the entity. An entity is like an empty order form and an instance is an order form that contains order information. Looking at instances of an entity helps to identify attributes because we are viewing a real entity. Instead of looking at a blank order form, we are looking at an order form that represents a real order. We will find instances of an entity in the existing system.

- **Characteristics of an attribute:** Once attributes are identified, you must describe the characteristics of each attribute. The common characteristics found in many attributes:

- **Attribute name**: The name of the attribute uniquely distinguishes the attribute from other attributes of the same entity.
  - "First name" is an attribute name. Duplicate attribute names within the same entity are prohibited.
  - However, two entities can use the same attribute name. That is, the customer entity and the sales representative entity can both have an attribute called first name.
- **Attribute type:** An attribute type is nearly identical to the data type of a column in a table.
  - Common attribute types include numeric, character, alphanumeric, date, time, Boolean, integer, float, and double, among other attribute types.
- **Attribute size**: The attribute size describes the number of characters used to store values of the attribute. This is similar to the size of a column in a table.
- **Attribute range:** An attribute range contains minimum and maximum values that can be assigned to an attribute.
  - For example, the value of the "total amount" attribute of an order entity is likely to be greater than zero and less than 10,000, assuming that no order has ever been received that had a total amount of more than 9,999. This range is then used to throw an error should an order be received with a total amount outside this range.
- **Attribute default value:** An attribute default value is the value that is automatically assigned to the attribute if the attribute isn't assigned a value by the J2ME application.
  - For example, the J2ME application uses the default system date for the date of an order if a sales representative fails to date the order. The system date is the attribute default value.
- **Acceptable values**: An acceptable value for an attribute is one of a set of values established by the business unit and includes zip codes, country codes, methods of delivery, and simply "yes" or "no."
- **Required value**: An attribute may require a value before the attribute is saved to a table.
  - For example, an order entity has an order number attribute that must be assigned an order number.

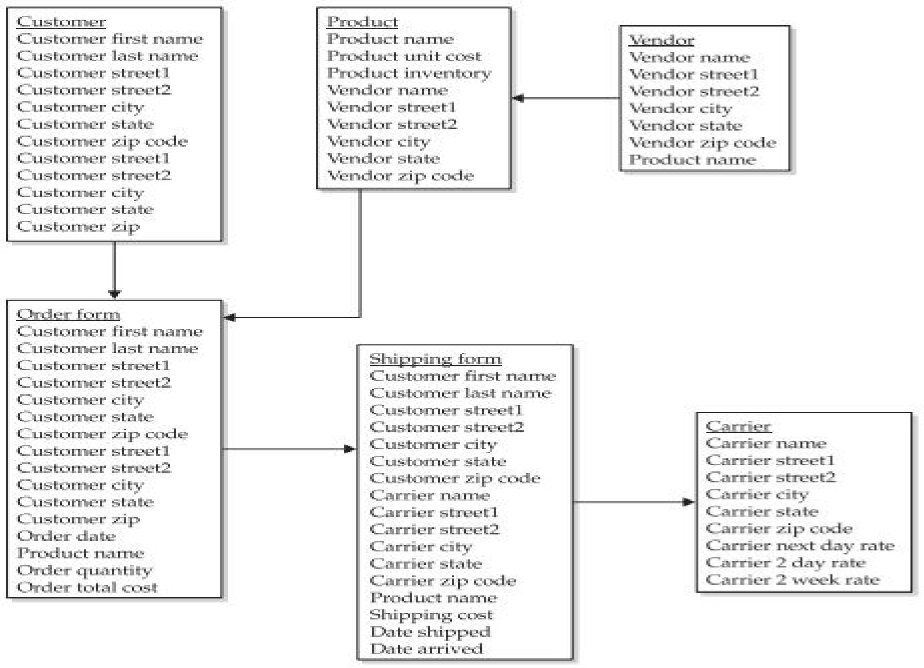
- **Attribute format:** The attribute format consists of the way an attribute appears in the existing system, such as the format of data.
- **Attribute source**: The attribute source identifies the origin of the attribute value. Common sources are from data entry and J2ME applications (such as using the system date as the value of the attribute).
- **Comments:** A comment is free-form text used to describe an attribute.

2. Decomposing Attributes to Data:

- Once attributes of entities are identified, they must be reduced to data elements. This process is called **decomposing**.
- In decomposing an attribute we can easily recognize whether an attribute is already at an atomic level. The nature of the system will determine whether or not additional decomposition is required for an attribute.
    - For example: customer name and customer address attributes. Both attributes are not atomic, but the attributes must be at atomic level.—first name, last name, city, state, and zip code.
    - For example, customer number attribute is already atomic. So there is no need of decomposing it. But sometimes it may be decomposed. If customer number consists of three numbered segments: 12-24-1001. The first segment (12) represents the sales region where the customer is located. The second segment (24) is the branch in the sales region that handles the relationship with the customer. And the final segment (1001) is the number that identifies the customer within the branch and region.
- **How to Decompose Attributes:** The process of decomposing attributes begins by analyzing the list of entities and their attributes. The list of attributes represents all the information used by the existing system. The objective is to reduce each attribute to a list of data that represents the atomic level of the attribute. It is as follows:

1. Look at each attribute and ask yourself if the attribute is atomic.

2. If the attribute isn’t atomic, it must be further decomposed. Create a list of data derived from the attribute.



3. If the attribute is atomic, no further decomposition is necessary for that attribute.

4. Place the name of the attribute on the data list.

5. Review the data list developed in step 2 and repeat the decomposition process until all attributes are atomic.

- **Decomposing by Example:**

The following are attributes for the customer entity:

- Customer number
- Customer name
- Customer address
- Customer telephone number

Once the list of attributes is assembled, each attribute on the list must be decomposed. The customer address attribute is decomposed in this example. The same process can be applied to the other attributes. The customer address attribute is not atomic, therefore the customer address attribute must be decomposed into the following data elements:

- ➢ Street address 1
- ➢ Street address 2
- ➢ City
- ➢ State
- ➢ Country
- ➢ Country code
- ➢ Postal code
- ➢ Address type (home or business)

3. Defining Data:

- Decomposing attributes results in the identification of data elements used by the existing system. Each data element can be defined as follows:
    - ➢ **Data name**: The unique name given to the data element, which should reflect the kind of data
    - ➢ **Data type:** A data type describes the kind of values associated with the data
    - ➢ **Data size**: The size of text data is the maximum number of characters required to represent values of the data. The size of numeric data is usually either the number of digits or the number of bytes for binary representation.
- **Data Types:** A data type describes the characteristics of data associated with a data element.
    - ➢ For example, a street address is likely to be an alphanumeric data type because a street address has a mixture of characters and numbers.
    - ➢ It is very important to select the data type of a data element because the data type that we choose typically becomes the data type of the column in the table that contains the data.
    - ➢ Many commercially available DBMSs have a common set of data types based on the SQL set of data types:
        - ▪ **Character-** also referred to as text Stores alphabetical characters and punctuation
        - ▪ **Alpha** -Stores only alphabetical characters

- **Alphanumeric -** Stores alphabetical characters, punctuation, and numbers
- **Numeric**- Stores numbers only
- **Date/Time** -Stores dates and time values
- **Logical (Boolean**) - Stores one of two values: true or false, 0 or 1, or yes or no
- **LOB (large object**) -Stores large text fields, images, and other binary data

4. Normalizing Data

- Normalization is the process of organizing data elements into related groups to minimize redundant data and to assure data integrity.
- Redundant data elements occur naturally since multiple entities have the same data elements.
- For example, an order form and invoice are both entities that contain a customer name and address. Therefore, customer name and address are redundant.
- Redundant data makes a database **complex, inefficient, and exposes the database to problems referred to as anomalies** when the DBMS maintains the database.
- Anomalies occur whenever new data is inserted into the database and when existing data is either modified or deleted, and can cause a violation to **referential integrity** of the database.
- For reporting data, the redundancy rules are violated a little because redundant data is more efficient. It minimizes the number of joins and allows data to be summarized into logical groups.
- Errors caused by redundant data are greatly reduced and possibly eliminated by applying the normalization process to the list of data elements that describe all the entities in a system. This is called **normalizing the logical data model of a system**.
- The normalization process consists of applying a **series of rules called normal forms** to the list of data elements to:
    - ➢ Remove redundant data elements.
    - ➢ Reorganize data elements into groups.

- Define one data element of the group (called a primary key) to uniquely identify the group. Often, two or more data elements make up the primary key, which is referred to as a composite key.
- Make other data elements of the group (called non-key data elements) functionally dependent on the primary key.
- Relate one group to another using the primary key.

- For example, a customer number is the primary key of a group that contains customer information. Other data contained in the group such as the customer first name and last name are referred to as non-key data elements. Non-key data elements are functionally dependent on the primary key. That is, a customer name, address, and related information cannot exist in the customer group without being assigned a customer number.
- **The Normalization Process:** There are five normal forms. But many industry leaders have concluded that the fourth and fifth normal forms are difficult to implement and unnecessary. The first three normal forms:
    - First normal form (1NF) requires that information is atomic.
    - Second normal form (2NF) requires data to be in the first normal form. In addition, data elements are organized into groups eliminating redundant data. Each group contains a primary key and non-key data, and non-key data must be functionally dependent on a primary key.
    - Third normal form (3NF) requires that data elements be in the second normal form, and non-key data must not contain transitive dependencies.

5. Grouping Data:

- A common way to organize data elements into groups is to first assemble a list of all data elements.
- When this is done, some data elements are duplicated because they are used by more than one entity.
- Duplicate data elements must be removed from the list. We must be careful, because not all data elements with similar sounding names are duplicates.
- For example, there are two data elements: zip code and postal code. These appear to have the same meaning. A zip code is another term for postal code. A zip code is a specific kind of postal code used in the United States. Postal code is a general term that also applies to postal codes used by countries other than the United States.

6. Creating Primary Keys & Foreign Keys:

- **Primary Keys:** A primary key is a data element that uniquely identifies a row of data elements within a group.
- The data selected to become the primary key may or may not exist in the data list .Sometimes a data element, such as an order number, alone is used as the primary key.
- DBMS can be requested to automatically generate a primary key whenever a column in the group isn't suitable to be designated the primary key.
- Consider **customer** entity. A customer has a **name and address** as attributes. These attributes decompose to first name, last name, street, city, state, and zip code.
  - None of these data elements are suited to become a primary key because individually and collectively none uniquely identify a customer.
  - The **customer first name and last name** seem to uniquely identify a customer, but upon closer analysis we see that more than one customer might have the same first name and last name.
- If neither a single data element nor a combination of data elements uniquely identifies a row, then we must create another data element to serve as the primary key of the table, which is what is required in the previous example. Alternatively, you can request the DBMS to generate a primary key automatically.
- Commercial DBMSs can generate primary keys to make the database thread safe and reliable.
- If a J2ME application that generates a key must contain the logic to be sure that none of the components running on different servers accidentally generate the same key.
- **Foreign Keys: A** foreign key is a primary key of another group used to draw a relationship between two groups of data elements.
- Relationships between two groups are made using the value of a foreign key.
- For example: there are two groups, one contains customer information, and the other contains order information. The primary key in the customer information group is the customer number, and the primary key in the order information group is the order number. Each row in the order group contains the customer number of the customer who placed the order. The customer number in the order group is a foreign key. That is, the customer number in the order group is the primary key of the customer information group.

- The DBMS is able to join information about a customer along with information about orders placed by that customer by joining together the customer number in both the customer information group and the order group.

Overview of the JDBC Process

- The interaction of J2ME applications with DBMS is divided into five routines:
    1. loading the JDBC driver
    2. connecting to the DBMS
    3. creating and executing a statement
    4. processing data returned by the DBMS
    5. terminating the connection with the DBMS

1. Load the JDBC Driver:

- The JDBC driver must be loaded before the J2ME application can connect to the DBMS. The Class.forName() method is used to load the JDBC driver.
- If a developer wants to work offline and write a J2ME application that interacts with Microsoft Access on the developer's PC. The developer must write a routine that loads the JDBC/ODBC Bridge driver called **sun.jdbc.odbc.JdbcOdbcDriver**.
- The driver is loaded by calling the Class.forName() method and passing it the name of the driver:

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

2. Connect to the DBMS:

- Once the driver is loaded, the J2ME application must connect to the DBMS using the DriverManager.getConnection() method.
- The java.sql.DriverManager class is the highest class in the java.sql hierarchy and is responsible for managing driver information.
- The DriverManager.getConnection() method is passed the URL of the database, along with the user ID and password if required by the DBMS.
- The URL is a String object that contains the driver name and the name of the database that is being accessed by the J2ME application.

- The DriverManager.getConnection() returns a Connection interface that is used throughout the process to reference the database.
- The java.sql.Connection interface is another member of the java.sql package that manages communications between the driver and the J2ME application.
- It is the java.sql.Connection interface that sends statements to the DBMS for processing.
- The following code shows the DriverManager.getConnection() method to load the JDBC/ODBC Bridge and connect to the CustomerInformation database.

```
String url = "jdbc:odbc:CustomerInformation";
String userID = "jim";
String password = "keogh";
Statement DataRequest;
private Connection Db;
try {
   Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
   Db = DriverManager.getConnection(url,userID,password);
 }
```

3. Create and Execute an SQL Statement:

- The next step after the JDBC driver is loaded and a connection is successfully made with a particular database managed by the DBMS is to send an SQL query to the DBMS for processing.
- An SQL query consists of a series of SQL commands that direct the DBMS to do something, such as return rows of data to the J2ME application.
- The Connect.createStatement() is used to create a Statement object. The Statement object is then used to execute a query and return a ResultSet object that contains the response from the DBMS, which is usually one or more rows of information requested by the J2ME application.
- The query is assigned to a String object, which is passed to the Statement object's executeQuery() method.
- The close() method is called to terminate the statement.
- The following code retrieves all the rows and columns from the Customers table:

```
Statement DataRequest;
ResultSet Results;
try {
       String query = "SELECT * FROM Customers";
       DataRequest = Database.createStatement();
       DataRequest = Db.createStatement();
       Results = DataRequest.executeQuery (query);
       DataRequest.close();
 }
```

4. Process Data Returned by the DBMS:

- The java.sql.ResultSet object is assigned the results received from the DBMS after the query is processed.
- The java.sql.ResultSet object consists of methods used to interact with data that is returned by the DBMS to the J2ME application.
- The following code is an abbreviated example that gives a preview of a commonly used routine for extracting data returned by the DBMS:

```
ResultSet Results;
String FirstName;
String LastName;
String printrow;
boolean Records = Results.next();
if (!Records ) {
   System.out.println( "No data returned");
   return;
}
else
{
 do {
   FirstName = Results.getString (FirstName) ;
   LastName = Results.getString (LastName) ;
   printrow = FirstName + " " + LastName;
   System.out.println(printrow);
 } while ( Results.next() );
}
```

- J2ME application requested a customer's first name and last name from a table. The result returned by the DBMS is already assigned to the ResultSet object called Results. The first time that the next() method of the ResultSet is called, the ResultSet pointer is positioned at the first row in the ResultSet and returns a boolean value. If false, this indicates that no rows are present in the ResultSet.
- A true value returned by the next() method means at least one row of data is present in the ResultSet, which causes the code to enter the do…while loop. The getString() method of the ResultSet object is used to copy the value of a specified column in the current row of the ResultSet to a String object. The getString() method is passed the

name of the column in the ResultSet whose content needs to be copied, and the getString() method returns the value from the specified column.

5. **Terminate the Connection to the DBMS:**

- The connection to the DBMS is terminated by using the close() method of the Connection object once the J2ME application is finished accessing the DBMS.
- The close() method throws an exception if a problem is encountered when disengaging the DBMS.
- Although closing the database connection automatically closes the ResultSet, it is better to close the ResultSet explicitly before closing the connection.

```
Db.close();
```

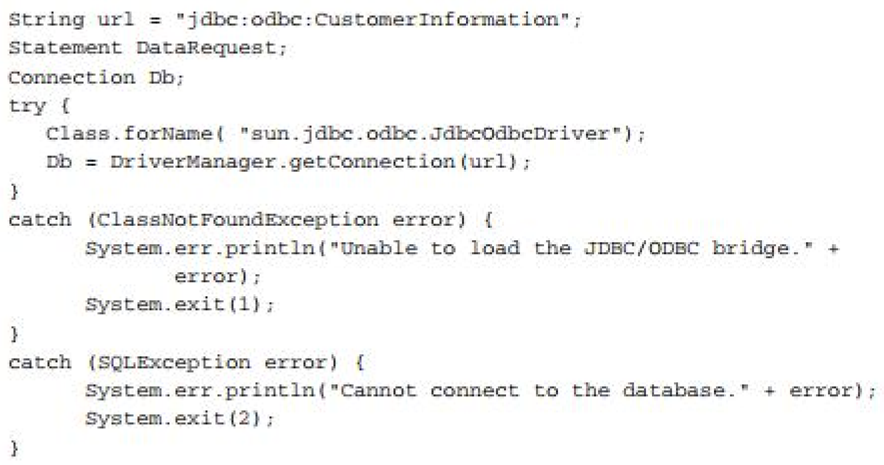
Database Connection

- A J2ME application does not directly connect to a DBMS.
- The J2ME application connects with the JDBC driver that is associated with the DBMS. Before this connection is made, the JDBC driver must be loaded and registered with the DriverManager.
- The purpose of loading and registering the JDBC driver is to bring the JDBC driver into the Java Virtual Machine (JVM).
- The JDBC driver is automatically registered with the DriverManager once it is loaded and is therefore available to the JVM and can be used by J2ME applications.
- The **Class.forName()**, is used to load the JDBC driver. The Class.forName() throws a ClassNotFoundException if an error occurs when loading the JDBC driver. Errors are trapped using the catch {} block whenever the JDBC driver is being loaded.

```
try
 {
 Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
}
 catch (ClassNotFoundException error)
 {
 System.err.println("Unable to load the JDBC/ODBC bridge." + error.getMessage());
System.exit(1);
}
```

The Connection:

- After the JDBC driver is successfully loaded and registered, the J2ME application must connect to the database. The database must be associated with the JDBC driver, which is usually performed by either the database administrator or the system administrator.
- The data source that the JDBC component will connect to is defined using the URL format. The URL consists of three parts:
    - jdbc, which indicates that the JDBC protocol is to be used to read the URL
    - which is the JDBC driver name
    - which is the name of the database
- The connection to the database is established by using one of three getConnection() methods of the DriverManager object.
- The getConnection() method requests access to the database from the DBMS. It is up to the DBMS to grant or reject access.
- A Connection object is returned by the getConnection() method if access is granted, otherwise the getConnection() method throws an SQLException.
- DBMS can grant access to a database :
    - **To anyone:** In this case, the J2ME application uses the getConnection(String url) method. One parameter is passed to the method because the DBMS only needs the database identified.



```
String url = "jdbc:odbc:CustomerInformation";
Statement DataRequest;
Connection Db;
try {
   Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
   Db = DriverManager.getConnection(url);
}
catch (ClassNotFoundException error) {
      System.err.println("Unable to load the JDBC/ODBC bridge." +
             error);
      System.exit(1);
}
catch (SQLException error) {
      System.err.println("Cannot connect to the database." + error);
      System.exit(2);
}
```

- **Limit access to authorized users:** This require the J2EE to supply a user ID and password with the request to access the database. In this case, the J2ME application uses the getConnection(String url, String user, String password) method.

```
String url = "jdbc:odbc:CustomerInformation";
String userID = "jim";
String password = "keogh";
Statement DataRequest;
Connection Db;
try {
   Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
   Db = DriverManager.getConnection(url,userID,password);
 }
catch (ClassNotFoundException error) {
      System.err.println("Unable to load the JDBC/ODBC bridge." +
             error);
      System.exit(1);
}
catch (SQLException error) {
      System.err.println("Cannot connect to the database." + error);
      System.exit(2);
}
```

- **when a DBMS requires information besides a user ID and password before the DBMS grants access to the database:** The additional information is referred to as “properties” and must be associated with a Properties object, which is passed to the DBMS as a getConnection() parameter. The properties used to access a database are stored in a text file, the contents of which are defined by the DBMS manufacturer. The J2ME application uses a FileInputStream object to open the file and then uses the Properties object load() method to copy the properties into a Properties object.

```
Connection Db;
Properties props = new Properties ();
try {
  FileInputStream propFileStream =
                new fileInputStream("DBProps.txt");
  props.load(propFileStream);
}
catch(IOException err) {
   System.err.print("Error loading propFile: ");
   System.err.println (err.getMessage());
   System.exit(1);
}
try {
   Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
   Db = DriverManager.getConnection(url, props);
}
catch (ClassNotFoundException error) {
      System.err.println("Unable to load the JDBC/ODBC bridge." +
             error);
      System.exit(2);
}
catch (SQLException error) {
      System.err.println("Cannot connect to the database." + error);
      System.exit(3);
}
```

Timeout:

- Competition to use the same database is a common occurrence and can lead to performance degradation.
- Multiple applications might attempt to access a database simultaneously. The DBMS may not respond quickly for a number of reasons, one of which might be that database connections are not available.
- Rather than wait for a delayed response from the DBMS, the J2ME application can set a timeout period after which the DriverManager will cease trying to connect to the database by using the following method:
    - **public static void DriverManager.setLoginTimeout(int seconds)-** establish the maximum time the DriverManager waits for a response from a DBMS before timing out.
    - **public static int DriverManager.getLoginTimeout()-**used to retrieve from the DriverManager the maximum time the DriverManager is set to wait until it times out. It returns an int that represents seconds.

Connection Pool:

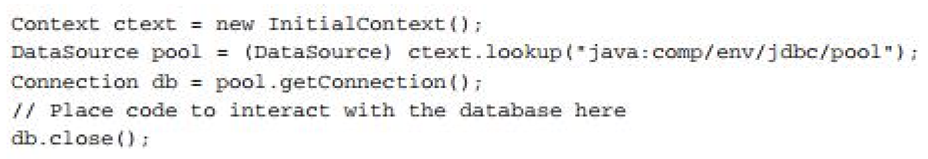
- Connecting to a database is performed on a per-client basis. That is, each client must open its own connection to a database, and the connection cannot be shared with unrelated clients.
- A client that needs to interact frequently with a database must either open a connection and leave the connection open during processing, or open or close and reconnect each time the client needs to access the database.
- Leaving a connection open might prevent another client from accessing the database because the DBMS have a limited number of connections available. Connecting and reconnecting is simply time consuming and causes performance degradation.
- The release of the JDBC 2.1 Standard Extension API introduced connection pooling to address the problem.
- A **connection pool** is a collection of database connections that are opened once and loaded into memory so these connections can be reused without having to reconnect to the DBMS. Clients use the DataSource interface to interact with the connection pool.

- The connection pool itself is implemented by the application server and other J2EE-specific technologies, which hide details on how the connection pool is maintained from the client.

- There are two types of connections made to the database:

    - **Physical connection**: which is made by the application server using PooledConnection objects. PooledConnection objects are cached and reused.

    - **Logical connection:** A logical connection is made by a client calling the DataSource.getConnection() method, which connects to a PooledConnection object that has already made a physical connection to the database.



```
Context ctext = new InitialContext();
DataSource pool = (DataSource) ctext.lookup("java:comp/env/jdbc/pool");
Connection db = pool.getConnection();
// Place code to interact with the database here
db.close();
```

UNIT-III

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. __________ is the process of organizing data elements into related groups to minimize redundant data.
2. ______________is a combination file system and database management system.
3. Within the same suite, there cannot be two RecordStores with the same name. **(T / F)**
4. _______________package contains a RecordStore class that provides basic access to data in a record store.
5. MIDP provides a mechanism for MIDlets to persist data so it can be used in later executions of the MIDlet, or to be shared among MIDlets. This mechanism is known as a ______________________
6. A____________ is the component of a database that contains data in the form of rows and columns.
7. RecordStore has the following properties:_______________,_______________ and ___________.
8. Searching of records is referred to as ________________.
9. _______________ method returns the number of records in the RecordEnumeration.
10. The method that is called whenever a record is added to the record store is ________________.
11. An ___________describes the characteristic of data that can be stored in the column.
12. The _______________ method is called to create a new record store and to open an existing record store. [ ]
    a) openRecordStore() b) openNewRecordStore()
    c) createRecordStore() d.none
13. Each RecordStore is composed of _____________ records. [ ]
    a) zero or more b) one or more c) two or more d)none
14. If an invalid record number was used, ____exception is raised. [ ]
    a) InvalidRecordIDException() b) InvalidRecordNumException()
    c) InvalidRecordException() d) none
15. _______________ retrieve the date of the last modification made to the record store. [ ]
    a) getLastModified() b) getLastDateModified() c) getLastTimeModified() d) none
16. ___________ method is used to insert a record into a record store.
    a) addRecord() b) setRecord() c) insertRecord() d)none. [ ]
17. Normalization is the process to assure _______________. [ ]

a) Data Integrity b) Data consistency c) Data Isolation d) Data Durability

18. ____________ requires that information is atomic. [ ]

a) 1NF b) 2NF c) 3NF d) BCNF

19. Each group must contain a primary key and non-key data, and non-key data must be functionally dependent on a primary key. This is constraint of ____________. [ ]

a) 1NF b) 2NF c) 3NF d) BCNF

20. ____________ functional dependency must be eliminated so that the table is in 2NF? [ ]

a) Trivial b) Partial c) non-trivial d) Transitional

21. ____________interface is used to search and sort records in a record store. [ ]

a) RecordState b) RecordListener c) RecordEnumeration d) none

22. -------------------- method is called to evaluate whether or not there is another record in the RecordEnumeration. [ ]

a) hasNextElement() b) hasOneElement() c) hasElement() d) none

SECTION-B

Descriptive Questions

1. Discuss about Record Listener?
2. Explain the characteristics of Attributes of an entity?
3. Explain about a) searching records b) sorting records
4. Define Normalization and discuss about different normal forms in normalization?
5. Discuss about :
   a) loading the JDBC driver
   b) connecting to the DBMS
6. Discuss about different record management exceptions?
7. Explain about the six steps that are used to create a database schema?
8. Explain about the common characteristics of an attribute?
9. Develop a program which illustrates the overview of JDBC process (loading, connecting, creating, processing and terminating).
10. Develop a program that establishes a database connection for a J2ME MIDlet?
11. Develop a program to create, open, close and remove a record from the RecordStore.
12. Develop a program to read and write a String-Based record from the RecordStore.
13. Develop a program to read a Mixed-Data type record into RecordEnumeration.
14. Illustrate the procedure to set up a Record Store?
15. Enumerate the procedure and the methods used, to manage records using Record Enumeration?
16. Illustrate how attributes can be decomposed to data with an example?

UNIT-IV

Objective:

To reproduce the installation of the Android Eclipse SDK.

Syllabus:

Introduction to Android Installation and configuration of android, starting an android application project: components, debugging with eclipse. Application design: the screen layout and Main.xml file, components ids, controls, creating and configuring android Emulator, communication with emulator.

Outcomes:

Student will be able to:

- **Explain the installation of Eclipse, java and ADK.**
- **Create and configure the android emulator.**
- **Communicate with emulator.**
- **Apply few simple controls.**
- **Illustrate Screen Layout and main.xml file.**
- **Explain component IDs**

What is Android?

- Android is an open source and Linux-based Operating System for mobile devices such as smart phones and tablet computers. Android was developed by the Open Handset Alliance, led by Google, and other companies.
- Android offers a unified approach to application development for mobile devices which means developers need only develop for Android, and their applications should be able to run on different devices powered by Android.

Installation and Configuration of Android

Installing and Eclipse and Java:

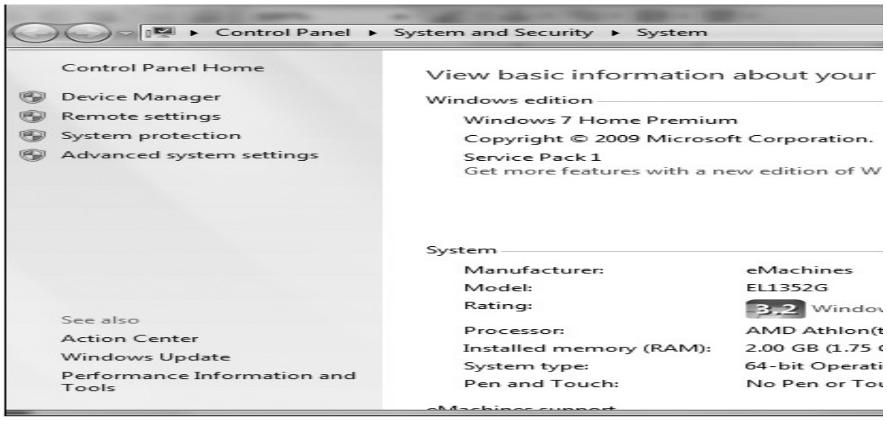
Eclipse is available from **www.eclipse.org**, and the JDK is available from **www.oracle.com/technetwork/java/javase/downloads/index.html.** The current version of the Java Standard Edition is version 7.

Step1: Download and install the JDK first. There are two possible downloads:

- JDK and the JRE.
- The JDK is the development software, and the JRE is simply the "run-time environment," a piece of software required by an operating system to host a Java application. Download and install the JDK. It contains and installs a copy of the JRE, so there is no need to install them separately.

Step 2: Set the path variable. After the JDK installs, you will want to set its location in your computer system's PATH variable. It is as follows:

1. Click the Start button. Then right-click on Computer on the right side of the Start menu and choose Properties.



View basic information about your

Windows edition

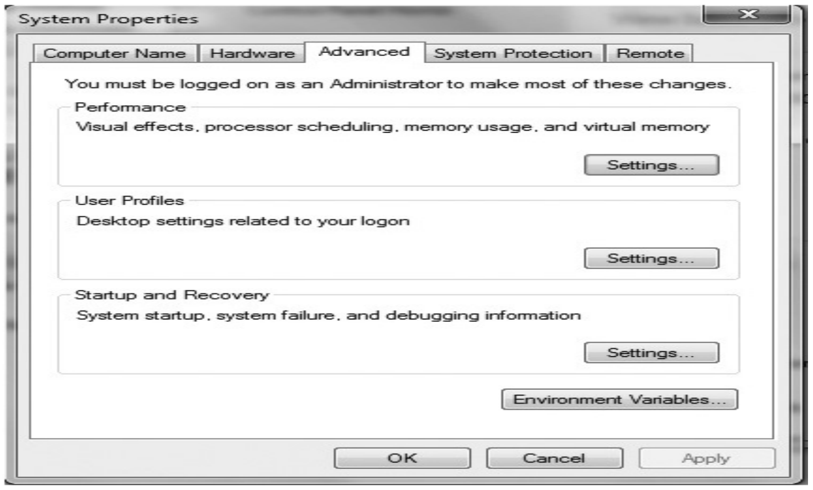
Windows 7 Home Premium
Copyright © 2009 Microsoft Corporation.
Service Pack 1
Get more features with a new edition of W

System

Manufacturer:	eMachines
Model:	EL1352G
Rating:	3.2 Windo
Processor:	AMD Athlon(t
Installed memory (RAM):	2.00 GB (1.75
System type:	64-bit Operati
Pen and Touch:	No Pen or Tou

Fig: Windows 7 System Settings panel

2. Choose Advanced System Settings on the left.



System Properties

You must be logged on as an Administrator to make most of these changes.

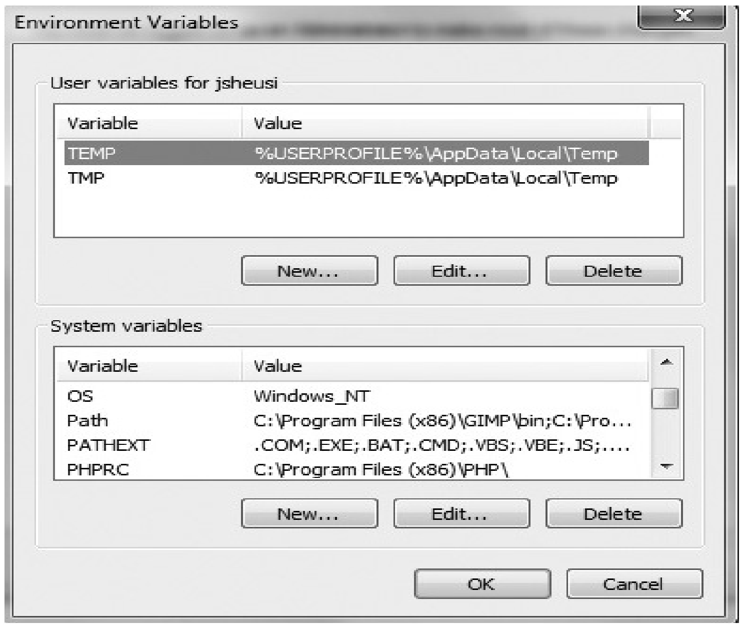
Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information

Fig: Windows 7 Environment Variables panel.

3. If it isn't selected already, select the advanced tab at the top. Then click the Environment Variables button at the bottom-right. You will see the window shown in Figure 1.3.



Environment Variables

User variables for jsheusi

Variable	Value
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
OS	Windows_NT
Path	C:\Program Files (x86)\GIMP\bin;C:\Pro...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;....
PHPRC	C:\Program Files (x86)\PHP\

Fig: Windows 7 Environment Variables panel.

4. In the bottom System Variables area, select Path from the list of variables, and click the Edit button. At the end of that line, insert a semicolon (;) followed by the path to your installed JDK, probably a line similar to the following:

c:\program files\java\jdk1.6.0_24\bin\

5. Finally, click OK and work your way out of the windows.

Step 3: Install Eclipse.

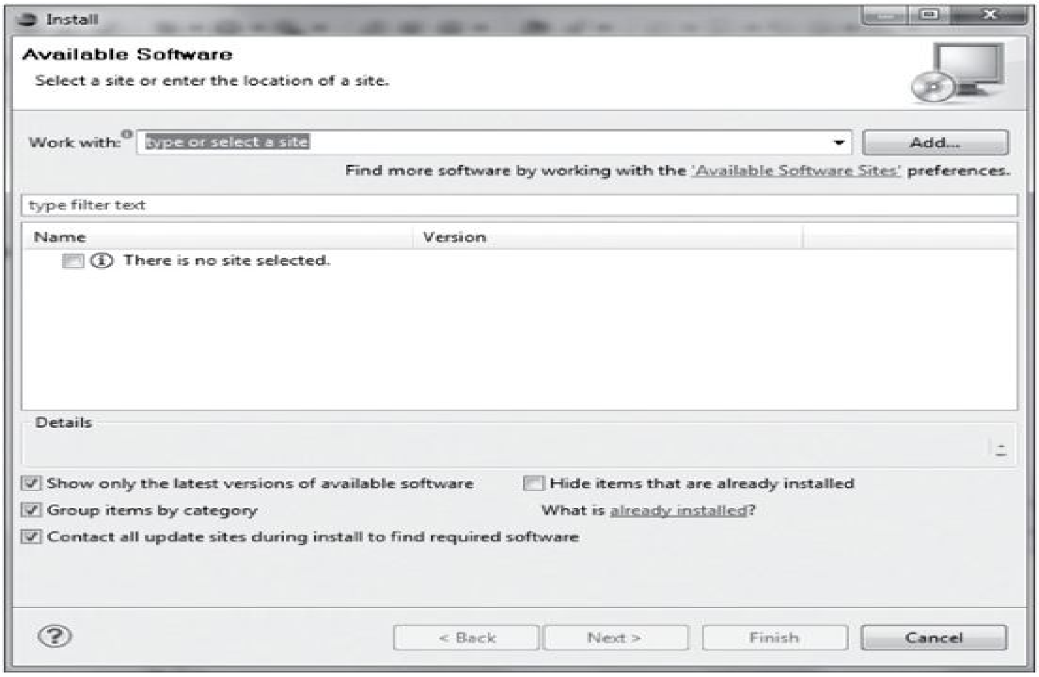
- You can download it at www.eclipse.org. The Eclipse website offers documentation to help you with the install.
- The Eclipse installation should use the system's path variable to find JDK. Eclipse does not install like most Windows software installs.
- It comes as a ZIP file that can be placed anywhere on the system and unzipped.
- Create a folder in the Program Files folder on a Windows system called Eclipse or something equally appropriate and unzip the ZIP file there. You might also want to create a shortcut for the Eclipse start icon and place it on your desktop.

INSTALLING THE ANDROID DEVELOPMENT KIT

Install the Android Development Kit (ADK). There are actually two steps to getting Android configured.

Step 1: Configure Eclipse. You can download the ADK at **http://developer.android.com/sdk.** Once you have it installed, you need to make a change to your PATH environment variable as follows: **c:\program files (x86)\android\android-sdk\tools\.** (Don't forget to separate entries with a semicolon.)

Step 2: Configure the Eclipse plug-in for Android. Start Eclipse and select Install New Software from the Help menu.


Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the 'Available Software Sites' preferences.

type filter text

Name	Version
There is no site selected.	

Details

[✓] Show only the latest versions of available software
[ ] Hide items that are already installed
[✓] Group items by category
What is already installed?
[✓] Contact all update sites during install to find required software

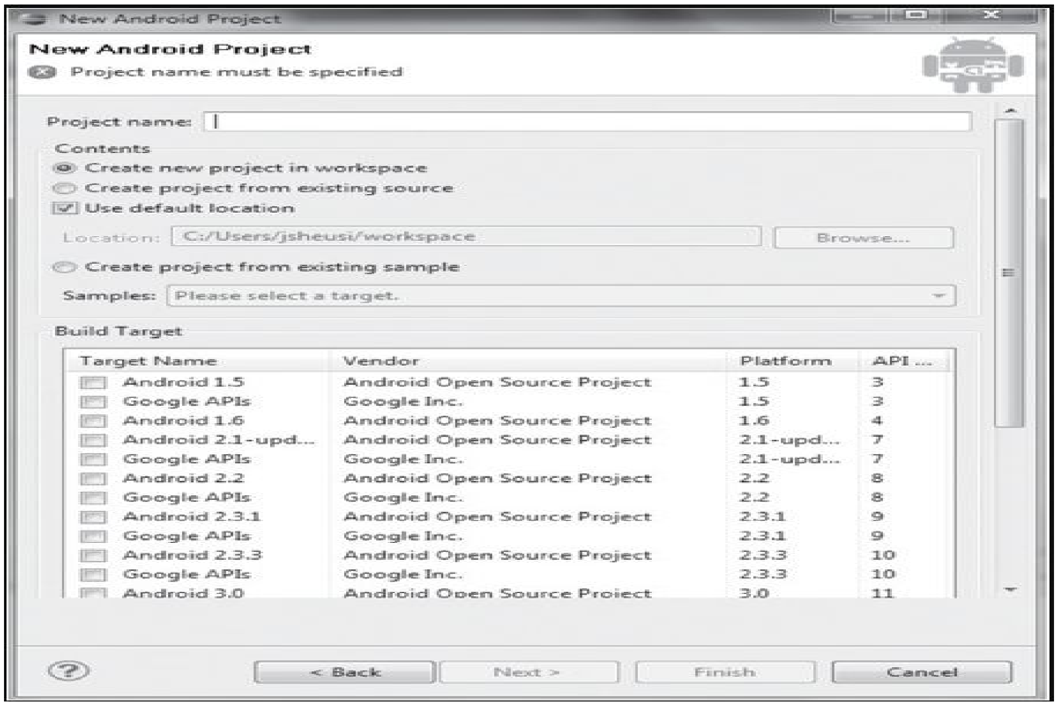
Fig: Eclipse Install New Software panel

- In the Work With field, enter the following website: **http://dl-ssl.google.com/android/eclipse/.** After clicking Add and waiting a moment, a Developer Tools line appears below. Put a check in the box that appears in the screen below, click next, and follow the prompts to the end of the process. You need to agree to all the licenses to get to the Finish button

Step 3: Testing the Android installation: It is similar to testing the Java installation. Again, select the File menu on Eclipse, select New, Project, and you should see Android Project as a choice. If it is there, you should be good to go. If not, check your steps and look for some online troubleshooting help.

Starting an Android Application Project:

- ➢ After we correctly configured Eclipse with Java and the Android Development Kit (ADK), it's time to write the first application.
- ➢ Eclipse creates a directory, or folder, to store our programming projects called Workspace.
- ➢ When Eclipse is installed, the installer is prompted for the desired location for this directory. After installation, Eclipse allows for the creation of new Workspace directories and allows us to change Workspace directories each time we open Eclipse.
- ➢ To start a project on the Eclipse desktop, select File, New, and Project. We are presented with a new dialog box. Select Android Project under the Android heading, and then select next.


New Android Project

New Android Project

Project name must be specified

Project name:

Contents

Create new project in workspace [✓]

Create project from existing source

Use default location [✓]

Location: C:/Users/jsheusi/workspace

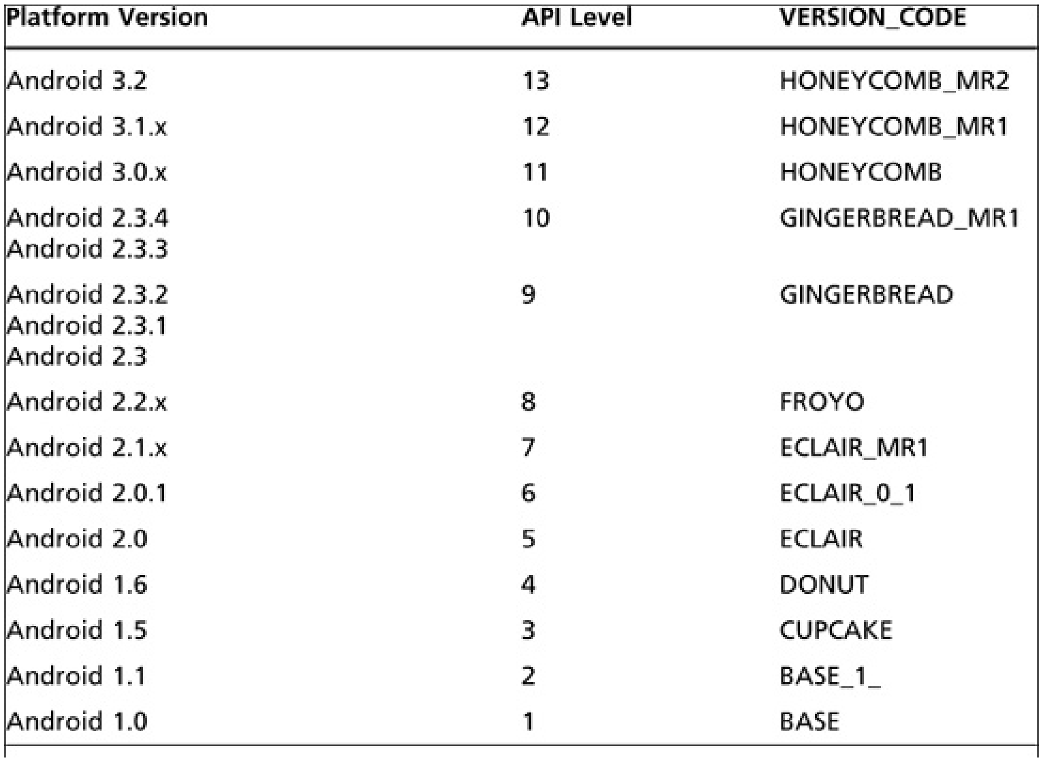
Create project from existing sample

Samples: Please select a target.

Build Target

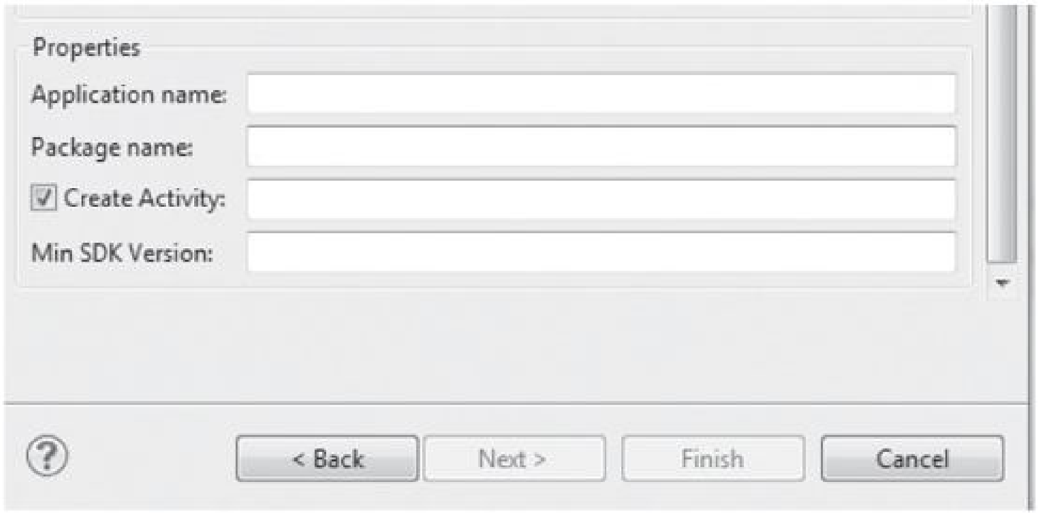
Target Name	Vendor	Platform	API ...
Android 1.5	Android Open Source Project	1.5	3
Google APIs	Google Inc.	1.5	3
Android 1.6	Android Open Source Project	1.6	4
Android 2.1-upd...	Android Open Source Project	2.1-upd...	7
Google APIs	Google Inc.	2.1-upd...	7
Android 2.2	Android Open Source Project	2.2	8
Google APIs	Google Inc.	2.2	8
Android 2.3.1	Android Open Source Project	2.3.1	9
Google APIs	Google Inc.	2.3.1	9
Android 2.3.3	Android Open Source Project	2.3.3	10
Google APIs	Google Inc.	2.3.3	10
Android 3.0	Android Open Source Project	3.0	11

- ➢ Choose a project name, such as Project_1. Notice the check box labelled Use Default Location. It is checked and contains a path with “workspace” as the final directory in the path.
- ➢ Next, the Build Target area refers to the version of the Android operating system (OS) you want your application to be used on.
    - The lower the application programming interface (API) we choose, the more inclusive you will be of devices that will run our application.
    - On the other hand, the lower the number we choose, the fewer features of devices and advances in the Android platform your application will be able to take advantage of.
- ➢ The complete list of versions and corresponding APIs



Platform Version	API Level	VERSION_CODE
Android 3.2	13	HONEYCOMB_MR2
Android 3.1.x	12	HONEYCOMB_MR1
Android 3.0.x	11	HONEYCOMB
Android 2.3.4	10	GINGERBREAD_MR1
Android 2.3.3		
Android 2.3.2	9	GINGERBREAD
Android 2.3.1		
Android 2.3		
Android 2.2.x	8	FROYO
Android 2.1.x	7	ECLAIR_MR1
Android 2.0.1	6	ECLAIR_0_1
Android 2.0	5	ECLAIR
Android 1.6	4	DONUT
Android 1.5	3	CUPCAKE
Android 1.1	2	BASE_1_
Android 1.0	1	BASE

- ➢ Next, we need to choose an application name. Android device owners who download your application will see this name and identify the application on their device by this name.



Properties

Application name:

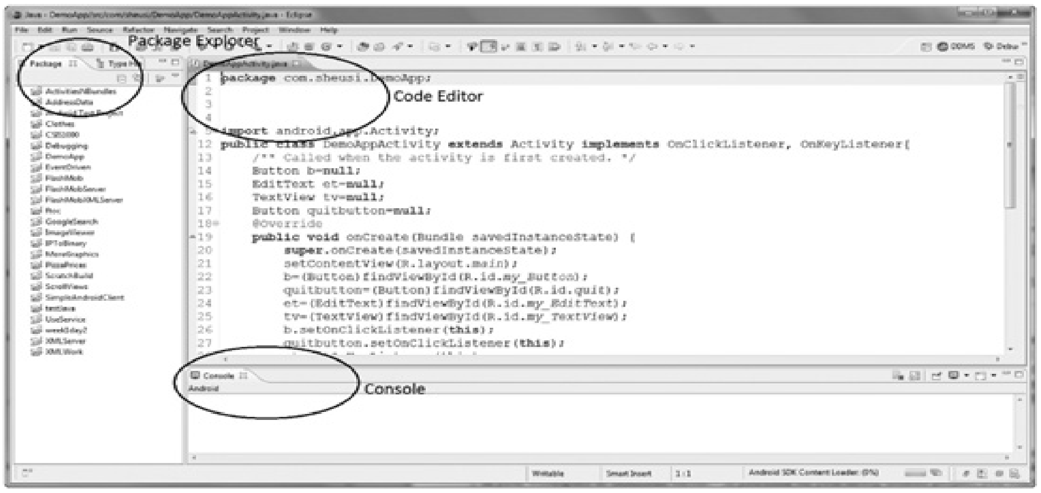
Package name:

☑ Create Activity:

Min SDK Version:

- ➢ The Package Name is a little more complicated and refers to the Java structure referred to as a package.
  - A package is a related set of classes.
  - Our project could contain many classes, related to each other by a single purpose: to provide the functionality of your Android app.
  - All the package names in the Android system must be unique.
  - Package is name must use a domain-style naming system.
  - Here we use reverse domain naming convention.(Example: com.sheusi.TipCalculator).

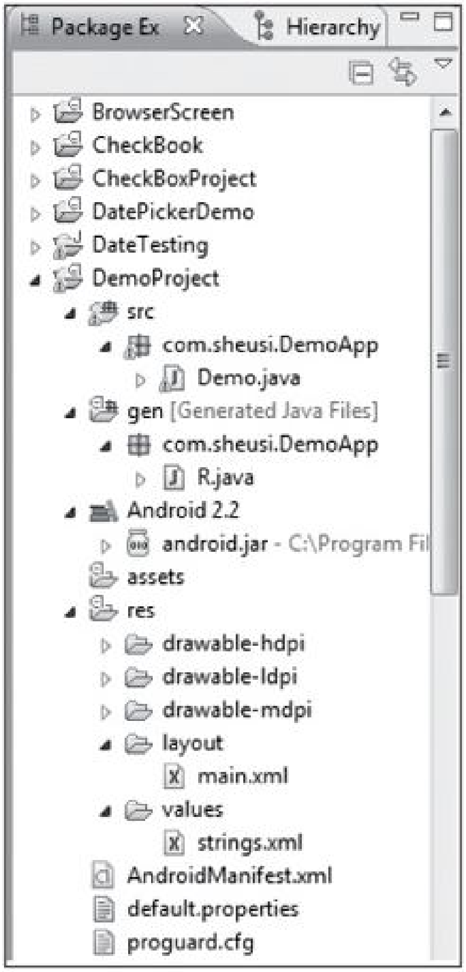
- ➢ The activity name is actually the class name for the primary Java class of our project, so we must follow the Java class naming conventions.
- ➢ We need to enter an integer to indicate the minimum SDK version for our project.
  - Entry of this value determines which versions of the Android OSs on devices that ultimately will want to download your application will be compatible, and which features our application will be able to use.
- ➢ After we click the Finish button, Eclipse creates an empty project framework for us.



```
package com.sheusi.DemoApp;



import android.app.Activity;
public class DemoAppActivity extends Activity implements OnClickListener, OnKeyListener{
    /** Called when the activity is first created. */
    Button b=null;
    EditText et=null;
    TextView tv=null;
    Button quitbutton=null;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        b=(Button)findViewById(R.id.my_Button);
        quitbutton=(Button)findViewById(R.id.quit);
        et=(EditText)findViewById(R.id.my_EditText);
        tv=(TextView)findViewById(R.id.my_TextView);
        b.setOnClickListener(this);
        quitbutton.setOnClickListener(this);
```

➢ The right side of the screen is the editor space and on the left side is the Package Explorer window.

Eclipse Package Explorer: (Android Application Components)



1) The first subdirectory is src.

This subdirectory or subfolder contains all the source code for the classes we intend to create for your application.

We should recognize the package name that we used in the dialog box in the first level below the src icon, and the activity name we chose with the **.**java extension added. This is the primary class of the application.

2) Next is the directory named gen.

The primary file here is called R.java, and it is created based on our configuration of the main.xml and strings.xml at a minimum.

3) The next component is a collection of API files based on the version we chose in the dialog box in the beginning.

4) The next directory is res**,** which is short for resources.

It contains folders whose names begin with drawable.

These contain graphics files such as the launch icons for the application. If we intend to use a launch icon other than the standard Android "robot" icon, we need to put a graphic of a specified size and type in each of these folders.

5) The layout and values subdirectories under res contain XML files.

➢ The **layout directory** contains XML files that configure the screens of the application.
➢ The **values** folder contains values for text strings and a couple of other data types we might use in our application.
➢ Assignments can be made here in the XML file, and they will be available throughout the coded application. These XML files together allow you to design the whole user interface without having to write a single line of Java code.

6) The next file is Android Manifest.xml file.

- This file can be viewed as the “instruction book” the target device uses to run the application.
- It contains things like permissions to use features on the device such as the GPS system, references to the files that should be included when the application is bundled up for deployment.
- This fie contains information about **version and revision numbers, API information.**

Following is a basic manifest file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
            xmlns:android="http://schemas.android.com/apk/res/android"
             package="com.sheusi.CheckBook"
            android:versionCode="1"
            android:versionName="1.0">
             <uses-sdk android:minSdkVersion="8" />

  <application android:icon="@drawable/icon" android:label="@string/app_name">
    <activity
            android:name=".CheckBook"
            android:label="@string/app_name">
        <intent-filter>
             <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
     </activity>

   </application>
 </manifest>
```

- Under the manifest tag is **the package** name, which is set at the beginning of the project.
- There’s also a version code; it’s a numeric value only, and it’s used internally to be sure that subsequent installs of the same application are newer versions.
- There’s also a version name and it is the version information that the user will see, and it is used only for this purpose.

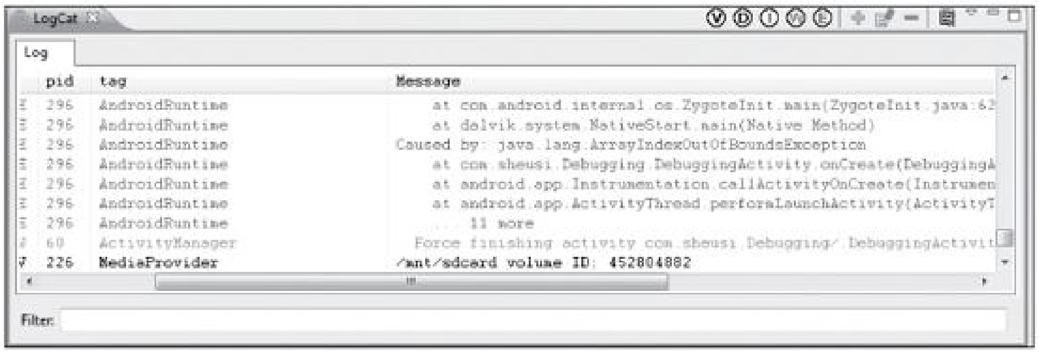
- **The <uses-sdk>** can contain more than one specification. The minSdk Version specification shown here prevents devices with a lower API than we specify from installing your application.

- Inside the **<application> tag**,
  - The parameter android: icon represents the graphic to be used as the launch icon on the device.
  - The android: label parameter is the name we gave the application when we started the project.
- Inside the **<activity>** tag,
  - The first parameter, android: name, which is the activity name we specified in the project setup. This will be the name of the Java class file that extends the Activity class.
  - The android: label parameter is the same as the previous android: label.
- Inside the **<intent-filter>** tag,
  - The parameter android: name is set to **.**MAIN.
  - It is an action built into the Android Intent class that instructs the device to open this application to the home screen.
  - The second parameter is android.name parameter, which is set to **.**LAUNCHER, defines the starting point.

Debugging with Eclipse:

- One of the biggest benefits to using an integrated development environment (IDE) such as Eclipse over using a simple editor to write source code is the extensive error detection and debugging facilities included in the IDE.
- The common errors that occur in application development fall into one of three common categories:
  - Syntax Error
  - Logical Error
  - Run-time Error
- **The logic error** is impossible for the IDE to detect and diagnose, because these errors are flaws in the approach to solving the problem that the application is meant to do.
- The other errors are different.
  - Take **syntax** errors, for example. These errors can include missing or incorrectly matching curly braces in code, missing semicolons at the end of a line, incorrect uppercase or lowercase letters, and so on.
  - In other words, they're what we could call "spelling and grammar."
  - These errors could also be use of classes without including the correct import statements, mistakes in variable scope, and other language-based errors.
  - In fact, these are the most obvious and easiest errors to fix.
- The error, indicated by a red circle, indicates a missing semicolon at the end of the line, a ***punctuation*** error.

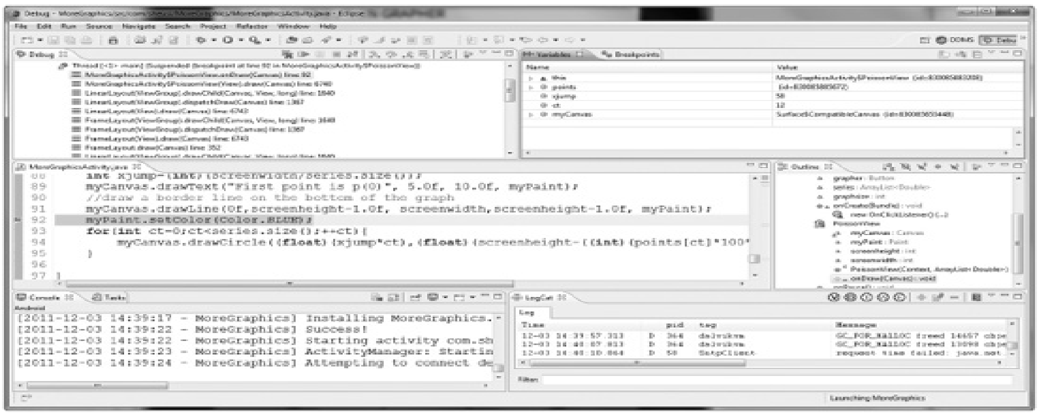
- The last type of error, the **run-time error**, can be the most frustrating. That is because this error occurs when the application is running, and there is no indication at compile time that anything is wrong. A run-time error turns up in the emulator when the application is running.
- Due to the nature of run-time errors, they don't occur every time the application runs. Unlike syntax errors, when run-time errors occur, the emulator gives no explanation.
- This is where the Debug perspective in Eclipse is the biggest help.
- In Eclipse there is a panel in lower-right corner with a tab marked LogCat. This is the first place to check for the source of the error. Any text in red is what We should examine.



```
pid  tag                Message
296  AndroidRuntime        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:62
296  AndroidRuntime        at dalvik.system.NativeStart.main(Native Method)
296  AndroidRuntime     Caused by: java.lang.ArrayIndexOutOfBoundsException
296  AndroidRuntime        at com.sheusi.Debugging.DebuggingActivity.onCreate(DebuggingA
296  AndroidRuntime        at android.app.Instrumentation.callActivityOnCreate(Instrumen
296  AndroidRuntime        at android.app.ActivityThread.performLaunchActivity(ActivityT
296  AndroidRuntime          ... 11 more
60   ActivityManager     Force finishing activity com.sheusi.Debugging/.DebuggingActivit
226  MediaProvider      /mnt/sdcard volume ID: 452804882
```

LogCat Panel

- We can also use the Eclipse debugger to check the values of variables at run-time by setting breakpoints in our code.
- The use of breakpoints to check values is useful during application design and testing and need not involve errors.
- We may want to check intermittent values during execution of a loop, or values of variables during calculation of a complex formula.
- To set a breakpoint, just double-click on the line number (or the left margin if we are not using line numbers) next to the line we would like execution to pause at.
- Next, switch to the debug perspective, and start the application by using the green bug icon instead of the green circle icon at the top of the screen.
- When execution reaches the line where you set the breakpoint, that line becomes highlighted in the code screen, and the variables and corresponding values appear one



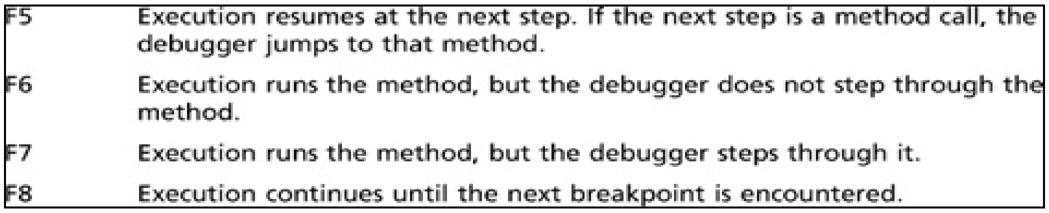
```
int xjump=(int)(screenwidth/series.size());
myCanvas.drawText("First point is p(0)", 5.0f, 10.0f, myPaint);
//draw a border line on the bottom of the graph
myCanvas.drawLine(0f,screenheight-1.0f, screenwidth,screenheight-1.0f, myPaint);
myPaint.setColor(Color.BLUE);
for(int ct=0;ct<series.size();++ct){
    myCanvas.drawCircle((float)(xjump*ct),(float)(screenheight-((int)(points[ct]*100*
}

}
```

```
[2011-12-03 14:39:17 - MoreGraphics] Installing MoreGraphics.
[2011-12-03 14:39:22 - MoreGraphics] Success!
[2011-12-03 14:39:22 - MoreGraphics] Starting activity com.sh
[2011-12-03 14:39:23 - MoreGraphics] ActivityManager: Startin
[2011-12-03 14:39:24 - MoreGraphics] Attempting to connect de
```

by one under the Variables tab on the upper right.

Eclipse debug screen showing a breakpoint

- Android applications are event driven, so we may need to bring up the emulator and perform the necessary actions on it for the debugger to reach the breakpoint, such as clicking a button onscreen.
- If we want execution to continue beyond the breakpoint, you can use the following function keys to obtain the corresponding results.

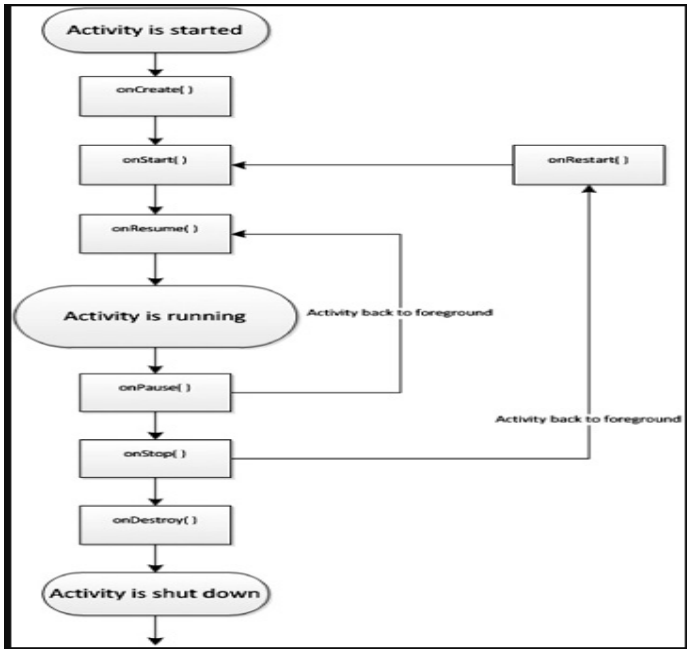


F5	Execution resumes at the next step. If the next step is a method call, the debugger jumps to that method.
F6	Execution runs the method, but the debugger does not step through the method.
F7	Execution runs the method, but the debugger steps through it.
F8	Execution continues until the next breakpoint is encountered.

Activity:

Class	Activity
Package	android.app
Extends	Android.view.ContextThemeWrapper

- An activity is generally a single-purpose screen and user interface. The activity takes care of creating the window on which the application designer places controls that allow the user to interact with the activity.



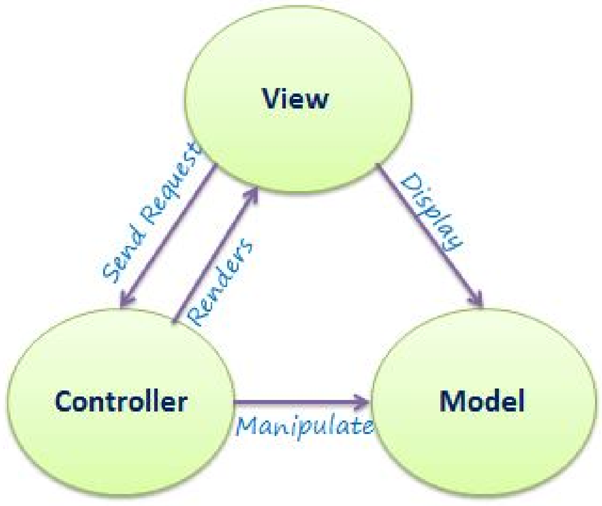
Simple Activity class state chart.

- The **onCreate ( )** method, which responds to the start of the application. Most of the screen configuration, initialization of variables, and design of event listeners take place in the **onCreate ( )** method.

onStart()	Called When the activity becomes visible to the user.
onResume()	Called when the Activity starts interacting with the user
onPause()	Called when the application/system resumes a previous activity. It is typically used to commit data to persistent storage.
onStop()	Called When the activity is no longer visible to the user.

Application Design:

- In Android, the Graphical User Interface (GUI) application paradigm used is "**Model-View Controller"**.



- The concept involves the developer considering three main areas when developing an application.
- The model represents what the application does and the coding behind what it is intended to do.
- The view is concerned with rendering the results on the display.
- The controller deals with how the user will interact with the application, including mouse movements, button clicks, and so on.
- When developing Android applications, we can easily isolate the view from the model and controller components.
- The view or layout of components is written in **XML** format in the main.xml file.
    - Once the programmer determines which controls are necessary for the application, such as lists, text fields, buttons, and so on, he can plan and code their arrangement, size, labels, fonts, and colors in the **main.xml file.**

The Rules to write XML files:

1. All XML elements must have opening and closing tags.
2. XML tags are case sensitive.
3. XML elements must be properly nested.

   ex.

   ```
   <tag1>
     <tag2>
     </tag2>
   </tag1>
   ```

4. XML documents must have a root element. A single tag pair must surround all other elements of the document. This is the root element.
5. XML attribute values must be quoted using either single or double quotes.

THE SCREEN LAYOUT AND THE MAIN.XML FILE

- Eclipse creates a functional application as soon as we create a new project.
- The screen configuration is controlled by the main.xml file that Eclipse generates.

Main.xml File:

```
<? xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
<TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/hello"
  />
</LinearLayout>
```

- The **LinearLayout** class is indeed a Java class found in the Android software development kit (SDK).
- It is one of many subclasses of the **ViewGroup** class.
- The attributes that are set in the XML file, namely orientation, **layout_width**, and **layout_length**, can be set in Java code by using methods that belong to the LinearLayout class.
- Using the main.xml file, we can separate form from functionality and design our user interface without writing a line of Java code.

- ➢ The outermost LinearLayout object normally represents the whole screen in an application, similar to the way a Frame class represents the application window in a PC Java application.
- ➢ The **orientation** attribute set to **Vertical** means that objects are added top to bottom.
- ➢ The two attributes, **layout_width** and **layout_height**, have two values
    - fill_parent
        - ✓ The fill_parent value is a special value that is always equal to the parent size. If it's used as the value for the **android:layout_width** attribute then it's the width of the parent view.
        - ✓ If it's used in the **android: layout_height** attribute, it would be equal to the height of the parent view instead.
    - wrap_content
        - ✓ The value **wrap_content** can be used much like a preferred size in Java AWT or Swing.
        - ✓ It says to the View object, "Take as much space as you need to, but no more".
        - ✓ The only valid place to use these special attribute values is in the **android: layout_width** and **android: layout_height** attributes.

- The “layouts” for Android applications are subclasses of the ViewGroup class. There are four layouts in Android:

    - ➢ FrameLayout,
    - ➢ AbsoluteLayout
    - ➢ TableLayout
    - ➢ RelativeLayout

<u>AbsoluteLayout</u>

- The AbsoluteLayout allows the programmer to specify the exact x,y coordinates of the components on the screen.
- Its use is limited because it doesn’t adjust for variations in the screen resolution of multiple target devices. So if the programmer is writing for only one type of device, say for a specific client, it may be useful.

<u>FrameLayout:</u>

- The FrameLayout layers multiple controls one on top of the other.
- This layout might be useful for graphics in an application.
- To see how the FrameLayout responds to text controls such as the Text-View control, we have to modify our main.xml file as follows and then restart the application:

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="horizontal"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
<TextView
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="@string/hello"

  />
  <TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="Text View number two"
  />
  <TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="Text View number three"
  />
</FrameLayout>
```

RelativeLayout:

- The RelativeLayout allows the programmer to position controls such as buttons on the screen relative to each other.

- To use a RelativeLayout, we have to modify our main.xml as follows, and then restart your application:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="horizontal"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 >
<TextView
 android:id="@+id/centertext"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="@string/hello"
 android:layout_centerHorizontal="true"
 android:layout_centerVertical="true"
 />
 <TextView
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Text View number two"
 android:layout_above="@id/centertext"
 />
 <TextView
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Text View number three"
```

```
  android:layout_below="@id/centertext"


  />
</RelativeLayout>
```

- The second and third TextViews appear at the top and bottom, respectively. In main.xml file, the android:id attribute for the first TextView is as follows:

android:id="@+id/centertext"

This is required because the other two TextView objects need to refer to the position of the first and need a way to identify it.

- The programmer can choose any ID for the control; We can choose a short name that identifies the control.
- **The coding @+id/:** When we start an Android Application Project, the R.java Java file generated by Eclipse. This file will contain several inner classes. One such inner class is called id. This prefix on the id name causes the actual identification value to be entered in that inner class.
- Double-click the R.java entry in the Package Explorer window on the left side of the Eclipse Screen (it's in the gen folder), and it opens in the editor. You will see the entry centertext in the id inner class.

<u>TableLayout:</u>

- The TableLayout is similar in appearance to the Java Frame's GridLayout.
- Within the TableLayout, the programmer makes one or more TableRow entries.
- Controls are placed inside the TableRow and appear in the order in which they are entered into the XML file.
- Columns are created as entries are made. To skip a column, the programmer can make a zero-based column specification.
- f only one row is specified, no gap appears between columns, even if there is a gap specified in the column indexes.
- If multiple rows are used, gaps appear where column index values are skipped.
- The column index is specified by the **android:layout_column** attribute:

(android:layout_ column="1").

- To use a TableLayout, we have to modify our main.xml as follows, and then restart your application:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
 android:orientation="horizontal"
 android:layout_width="fill_parent"
 android:layout_height="fill_parent"
 >
<TableRow>
<TextView

 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Column 1"
 android:layout_column="0"
 />
 <TextView
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Column 2"
 android:layout_column="1"
 />
 <TextView
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Column 3"
 android:layout_column="3"
 />
 </TableRow>
<TableRow>
<TextView
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Column 1"
 android:layout_column="0"
 />
 <TextView
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Column 2"
 android:layout_column="2"
 />
 <TextView
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="Column 3"
 android:layout_column="4"
 />
 </TableRow>
```

</TableLayout>

- We are using ***five* columns, indexed zero through four (0,1,2,3,4),** and they are specified by the attribute : android:layout_column. They have nothing to do with the attribute android:text, although we are using the word Column in the text.

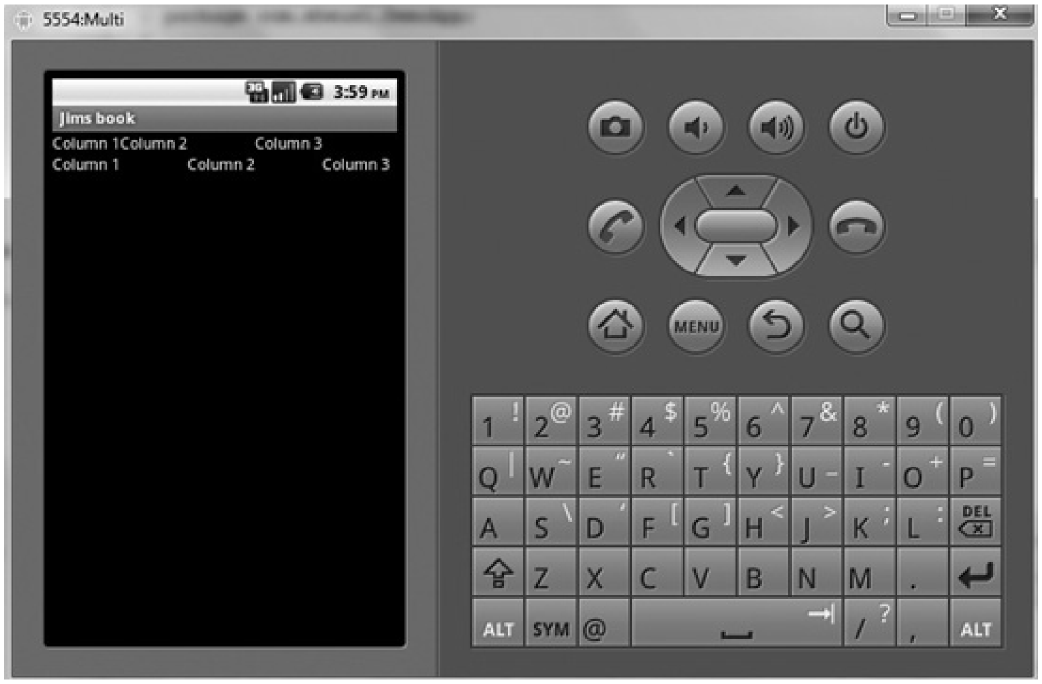


Fig: Emulator image showing rows and columns.

COMPONENT IDS

- We have used one of the controls a name with an android:id attribute. The line of code from the main.xml file:

android:id="@+id/centertext"

  - The **@ sign** in the ID is a signal to the XML parser how to deal with the ID string.
  - The + indicates that this is an ID the user has created; it is not part of the Android framework namespace.

- The android:id attribute is important to the RelativeLayout, but it is also important if the programmer wants to identify components from the XML layout file in the application's Java code.

- Suppose we write an application with a text field where the user would enter her name.
    - ➢ The programmer would need a way to identify that particular text field to extract the entered name to do something with it.
    - ➢ There must be a way to link a component from the static design of the screen during **development (the XML file)** to the application at **run-time (the Java code).**
- In the Java code, we declare and assign an instance of an object matching the class of the element in the XML. In the case we are describing, it would be an EditText control. The assignment step uses the ID from the XML file to make the connection.
- It is a good to put an android:id attribute on our layouts.
    - ➢ For example, you could identify the lowest level layout, the one represented with the first opening tag, as “base.” When we added two TableLayouts to the underlying LinearLayout, the LinearLayout could be IDed as “base,” the first TableLayout as level 2a or table_1.

CONTROLS

- We should build some functional controls into a main.xmlfile to actually watch an application doing something.
- We have used TextView objects to display some simple text. There are two more basic useful controls, or *widgets* :
    - ➢ EditText
    - ➢ Button.
- EditText controls are similar to TextFields. Android TextView is similar to the Java Label class.
- The following is a simple application that will take the contents of a predefined TextView and use a button to cause the application to take that text, convert it to uppercase, and display it in an EditText field. A basic main.xml file could look like the following:

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout_width="fill_parent"
```

```
android:layout_height="fill_parent"
android:id="@+id/base"
>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="My first android application"
android:id="@+id/my_TextView"
/>
<Button
 android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Touch me"
android:id="@+id/my_Button"
/>


<EditText
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:id="@+id/my_EditText"
/>
</LinearLayout>
```

- R.java is a file that is created based on the configuration of the main.xml. Let us see how the id attributes are interpreted in the R.java file:

```
package com.sheusi.DemoApp;
public final class R {
  public static final class attr {
  }
  public static final class drawable {
    public static final int icon=0x7f020000;
  }
  public static final class id {
    public static final int base=0x7f060000;
    public static final int my_Button=0x7f060002;
    public static final int my_EditText=0x7f060003;
    public static final int my_TextView=0x7f060001;
  }
  public static final class layout {
    public static final int main=0x7f030000;
  }
  public static final class string {
    public static final int app_name=0x7f040001;
    public static final int hello=0x7f040000;
  }
  public static final class style {
    public static final int shout=0x7f050000;
  }
}
```

- Eclipse builds a basic framework for the main Java file. It is found in the **src** subdirectory and has the name that we filled into the **Create Activity field**, when we started the project.
- There will be only enough code to put something into the emulator, so we just know that the application was created, and not much more.
- Our first step is **to decide what our application is designed to do**. This is our model in the MVC paradigm.

- Our application is meant to take the contents of a text field, convert them to uppercase, and place the results in another text field. Our view includes the original text in a text field, a button to start the conversion, and a text field to hold the results. The following is the java code:

```
package com.sheusi.DemoApp;


import android.app.Activity;

import android.os.Bundle;

import android.view.*;

import android.widget.*;

import android.view.View.OnClickListener;

public class Demo extends Activity implements nClickListener

{

/** Called when the activity is first created. */

            Button b=null;

EditText et=null;

TextView tv=null;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

 b=(Button)findViewById(R.id.my_Button);

et=(EditText)findViewById(R.id.my_EditText);

tv=(TextView)findViewById(R.id.my_TextView);


 b.setOnClickListener(this);

}

public void onClick(View v){
 String temp=tv.getText().toString();
 temp=temp.toUpperCase();
et.setText(temp);
 }
}
```

- The first statement starts with the word **package.**

package com.sheusi.DemoApp;

We chose the package name when you created the project.

- Next, the statements that begin with the word **import.** Java, like other object oriented languages, uses groups of predefined classes called ***packages***.
- Packages generally have multipart names separated by periods, such as android.view.View.Button.
- When packages are built, they have a tree-style organization, and the periods let the Java compiler navigate through the structure.
- The import statements are a convenience that relieves us from having to type the whole package name each time we use a class from the package. Here is an example.
- **Without the import statement**, if we wanted to declare and assign a couple of buttons, we would need these statements:

```
android.widget.Button b1;
android.widget.Button b2;
b1= new android.widget.Button("click me");
b2= new android.widget.Button("quit");
```

- **Using the import statement**,

import android.widget.Button;

If we want to indicate all classes in a given package, we can use an asterisk (*) as a wild card. For example:

import android.widget.*;

- The declarations and assignments need only the class name:

```
        Button b1;

        Button b2;

        b1= new Button("click me");

b2= new Button("quit");
```

- The next line is the **public class**. We have provided the name for the public class when you created the project:

public class Demo **extends** Activity **implements** OnClickListener{

- The core of an Android application is built on the **Activity class**. When we create an application, we customize the Activity class, or *extend* it .
- Eclipse generates the beginning of this statement. We add the words **implements OnClickListener** as we customize the application.
- The class **OnClickListener**, as the name implies, lets our application “listen” or check for actions on the user interface, namely the Android device screens, hardware buttons, and so on. Only certain objects can send click messages, but the Android device knows what these are, and the Android application programming interface (API) tells the programmer what they are.
- The next three lines declare the text areas and Button for the application’s user interface. A TextViewis similar to a Java Label class in that it contains text but cannot be edited by simply typing into it at run-time. For this, we use the EditText class.

    Button b=**null**;

    EditText et=**null**;

    TextView tv=**null**;

- The next section is where we define whatever we want to happen when the application starts, including laying out the screen, setting initial values to variables, and so on.
- In our application, we assign values to our control (widget) objects, and we connect the application’s “listener” code to the Button:

public void onCreate(Bundle savedInstanceState)

{

```
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
 b=(Button)findViewById(R.id.my_Button);
et=(EditText)findViewById(R.id.my_EditText);
 tv=(TextView)findViewById(R.id.my_TextView);
b.setOnClickListener(this);
}
```

- All the statements that include R.id.*** are the id attribute values we added to the sections of the main.xml file. These statements connect our screen objects to the Java code. Finally, the Button variable, b, connects the listener code.
- The last section defines what we want to happen when we touch the button on the screen

```
public void onClick(View v){
String temp=tv.getText().toString();
 temp=temp.toUpperCase();
et.setText(temp);
 }
```

- In the above code the following is the summary of what we have done:
    - ➢ We create a text string
    - ➢ Assign to it the contents of the TextView on the screen.
    - ➢ The TextView got its original text from the android:text statement in the main.xml file.
    - ➢ Then, through a built-in method of Java's String class, we convert all the letters to uppercase.
    - ➢ Finally, we assign the converted text string to the text property of the EditText field.
    - ➢ Edit your .java file to match the previous code, run the application, and see what happens.

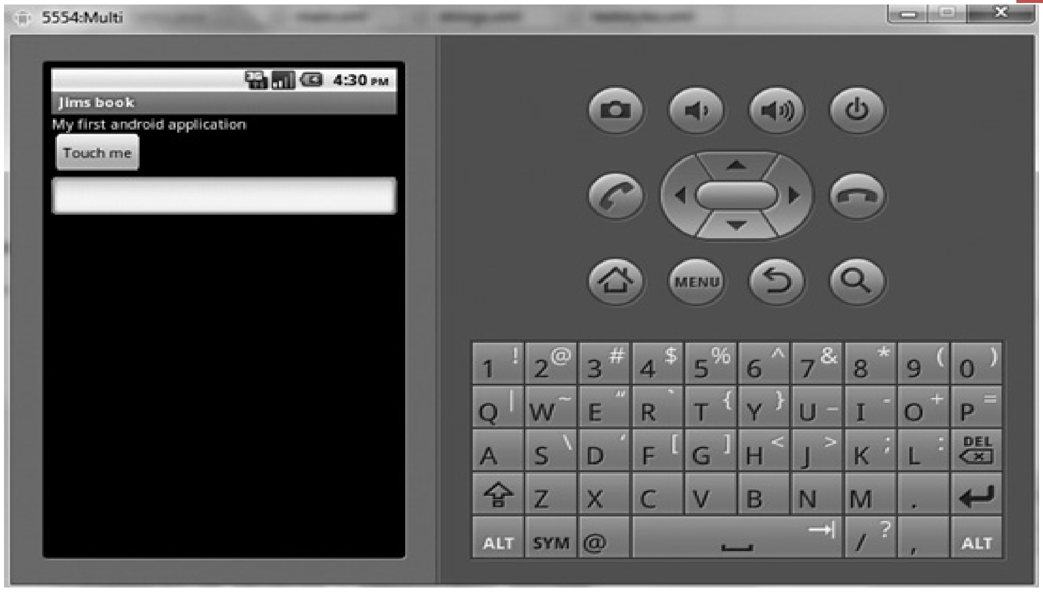


Fig:Emulator image of the application designed earlier.

- Let us add a second Button to close the application.
  - ➢ Go back to the main.xml file and add another button directly below the EditText set of tags.
  - ➢ Use the original button as our model. You can even copy and paste, but set the :
    - ▪ android:text attribute to Quit
    - ▪ android:id to @+id/quit.
  - ➢ Take care to align the opening and closing markers, < and />, correctly. Eclipse automatically makes corresponding changes to the R.java file, so there is no need to edit that manually. Finally, make the necessary changes to the Java file so that it matches the following:

```
package com.sheusi.DemoApp;

import android.app.Activity;

import android.os.Bundle;

import android.view.*;

import android.widget.*;

import android.view.View.OnClickListener;

public class Demo extends Activity implements OnClickListener{

 /** Called when the activity is first created. */
 Button b=null;

EditText et=null;
TextView tv=null;
Button quitbutton=null;
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.main);
  b=(Button)findViewById(R.id.my_Button);
  quitbutton=(Button)findViewById(R.id.quit);
  et=(EditText)findViewById(R.id.my_EditText);
  tv=(TextView)findViewById(R.id.my_TextView);
  b.setOnClickListener(this);
  quitbutton.setOnClickListener(this);
 }
public void onClick(View v){
 if(v==b){
 String temp=tv.getText().toString();
 temp=temp.toUpperCase();
 et.setText(temp);
 }
 if(v==quitbutton){
   this.finish();
 }
}
}
```

- We can declare an additional **Button object** and the same syntax as the original Button object. We can assign the same **OnClickListener** to the second Button.
- If we assign the same listener, the application would not know which Button was touched or clicked. We solve that problem with decision statements based on the parameter :

View v in the **onClick( ) statement**.

- Because the **Button class is a child class of the View class** in the Android SDK, the parameter can represent the buttons. The decision statements assess which Button was clicked or touched.
- The **.finish( ) method** is a method of the Activity class that ends the activity. Because our application extends the Activity class, we can use this method to end the application.
- The word this simply refers to the class using the .finish( ) method—the Demo class.
- When working with EditText controls on the screen, we may want the application to respond to a particular keystroke rather than requiring the user to touch a button or take some other action.
    - ➢ For instance, you may want to respond to keypad input as soon as the user touches the Enter key. We can do that by associating a **KeyListener** with a given input field.

```
package com.sheusi.DemoApp;

import android.app.Activity;

import android.os.Bundle;

import android.view.*;

import android.widget.*;

import android.view.View.OnClickListener;

import android.view.View.OnKeyListener;

public class DemoAppActivity extends Activity implements OnClickListener,
OnKeyListener
```

- First, there is a new **import statement** to add the **OnKeyListener interface** to our namespace.

```
import android.view.View.OnKeyListener;
```

- The implementation of interfaces requires that certain methods be defined in the code. In the case of the OnKeyListener interface, it is the **onKey( )method.** The method takes three arguments, or parameters,
  - a **View object**,
  - **an integer representing the key** you want to respond to.
    - The keyCode integer variable also represents a set of symbolic constants defined in the KeyEvent's documentation; there is a constant for each of the keys on the keypad.
  - a **KeyEvent** object. The KeyEvent object will represent one of two actions on a given key:
    - When it is touched or pushed, and when it is released. These are represented by symbolic constants that can be found in the KeyEvent's documentation: Symbolic constants are represented in all uppercase letters.

ACTION_DOWNand ACTION_UP.

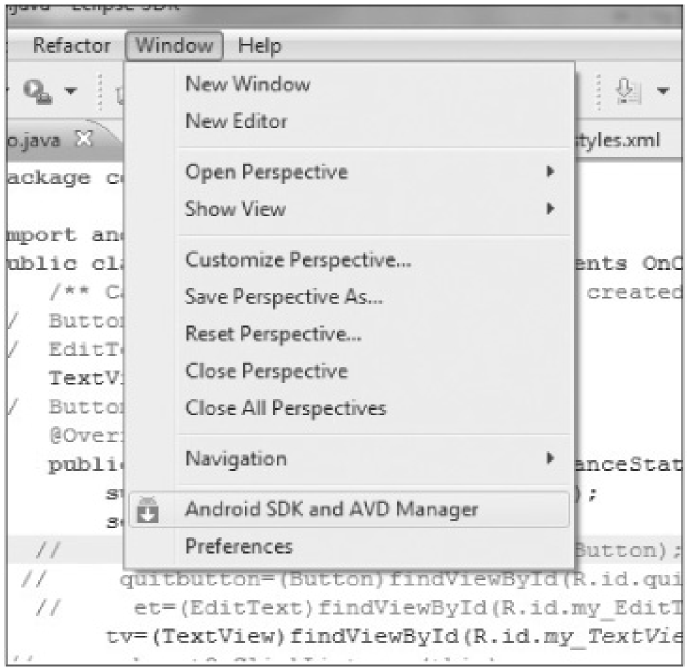
```
public boolean onKey(View v, int keyCode, KeyEvent event)

{

 if(event.getAction()==KeyEvent.ACTION_DOWN){



 if(keyCode==KeyEvent.KEYCODE_ENTER){

 String temp=et.getText().toString();

temp=temp.toUpperCase();

 et.setText(temp);

 } }     }
```

- The common non character keys on the Android devices and their corresponding symbolic constants are as follows:

Device's Key	KeyEvent Symbolic Constant
Power button	KEYCODE_POWER
Back key	KEYCODE_BACK
Menu key	KEYCODE_MENU
Camera button	KEYCODE_CAMERA
Home key	KEYCODE_HOME
Search key	KEYCODE_SEARCH

CREATING AND CONFIGURING AN ANDROID EMULATOR

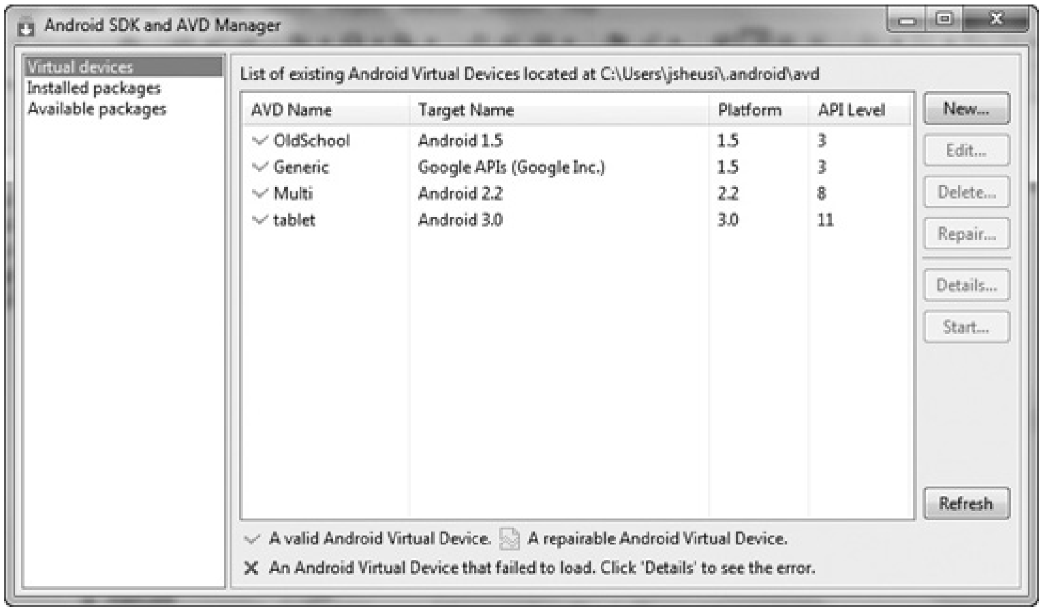
- There are different releases of the Android platform and the improvements and features added to the subsequent releases. Each of those versions had a version number starting at 1.0 and running to 3.2.
- We choose an API for our project to correspond with the target platform when the project is started.
- Finally, we need to configure an emulator that will properly simulate the use of our application. It is necessary to add features such as GPS and SD card for external storage.
- We can add features to existing emulators and create new emulators through **Eclipse**. Eclipse will maintain several emulators so the application designer can switch among several choices during development of a single application.
- Under the Window menu in Eclipse, choose Android SDK and AVD Manager. AVD stands for Android Virtual Device, the emulator.



```
ackage c
mport an
ublic cl                                          ents OnC
   /** C                                          created
/  Butto
/  EditT
   TextV
/  Butto
   @Over
   publi                                          anceStat
       s                                          );
       s
 //                                               Button);
//      quitbutton=(Button)findViewById(R.id.qui
 //      et=(EditText)findViewById(R.id.my_EditT
       tv=(TextView)findViewById(R.id.my_TextVie
```

Fig: Window menu open showing Android SDK and AVD Manager choice.

- Choosing that will give you the window, We will get:


Android SDK and AVD Manager

Virtual devices
Installed packages
Available packages

List of existing Android Virtual Devices located at C:\Users\jsheusi\.android\avd

AVD Name	Target Name	Platform	API Level
✓ OldSchool	Android 1.5	1.5	3
✓ Generic	Google APIs (Google Inc.)	1.5	3
✓ Multi	Android 2.2	2.2	8
✓ tablet	Android 3.0	3.0	11

✓ A valid Android Virtual Device. A repairable Android Virtual Device.
✗ An Android Virtual Device that failed to load. Click 'Details' to see the error.

Fig: Emulator configuration panel.

- Our screen may have only the **"default"** emulator in the list; there will be **New and Edit** buttons on the right, which can be used to **modify the chosen emulator or create new ones**.
- When we decide to create a new emulator, we have to remember the platforms and the features each offers. If we are writing an application that wants to take advantage of particular device features, we have to build an emulator at that level. Otherwise, our application will not run in the emulator as we might expect.
- To choose the emulator you want to use for your project development, you can click the down-facing arrow next to the green-circle icon and choose Run Configurations as indicated in Figure :

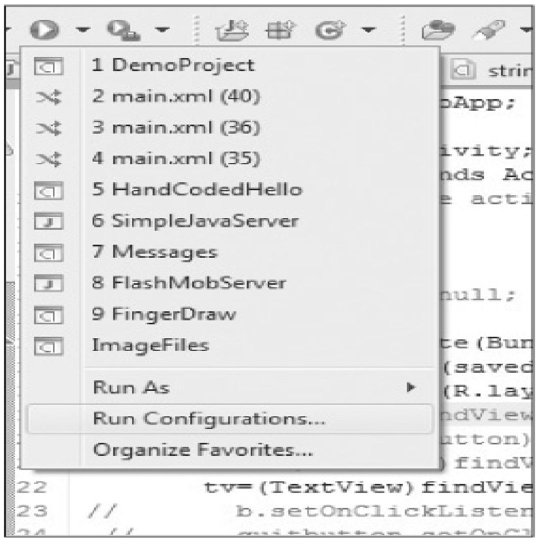
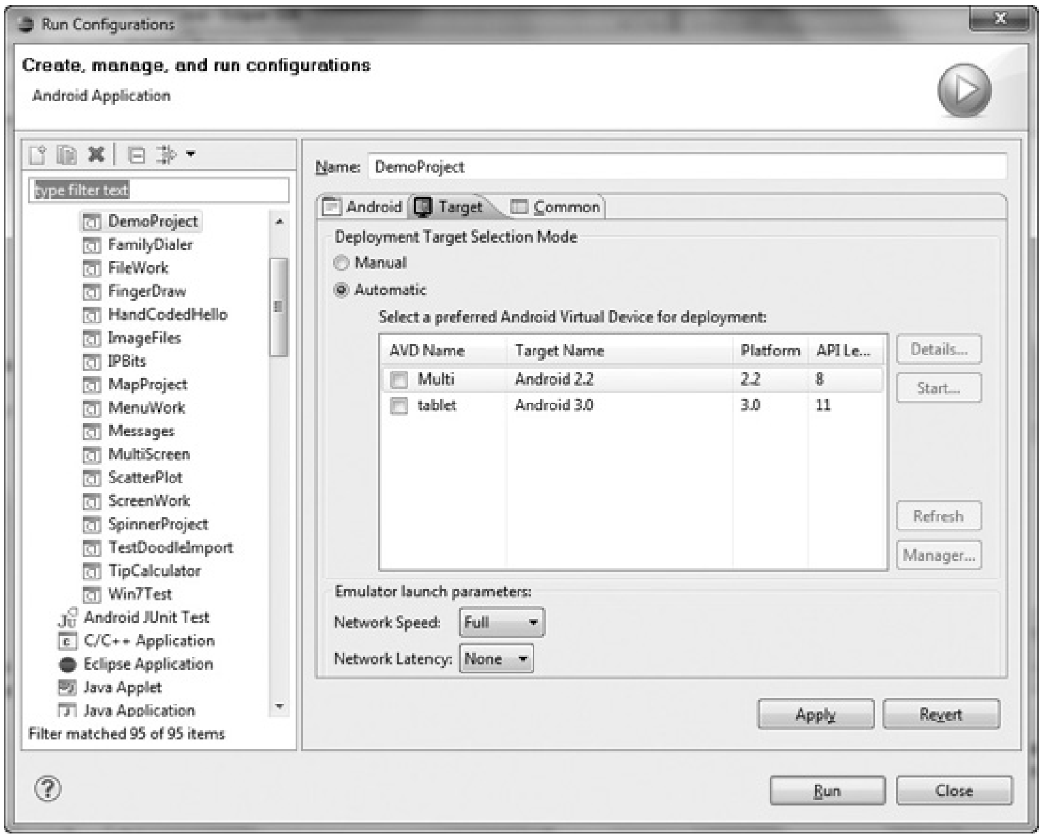


Fig: Run Configurations choice revealed in menu.

- Alternatively, you can choose Project, Properties, Run/Debug Settings. Then choose your project and choose the Edit button on the right. Either way, you will end up with the window shown in Figure :



Name: DemoProject

Deployment Target Selection Mode
( ) Manual
(●) Automatic

Select a preferred Android Virtual Device for deployment:

AVD Name	Target Name	Platform	API Le...
Multi	Android 2.2	2.2	8
tablet	Android 3.0	3.0	11

Emulator launch parameters:
Network Speed: Full
Network Latency: None

Fig: Run Configurations panel.

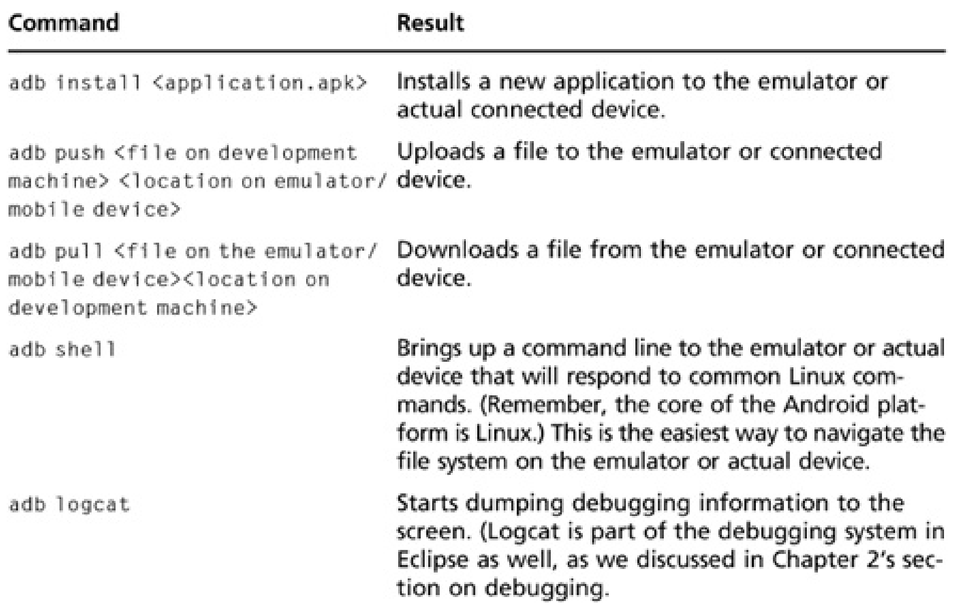
- On this screen, you would select the Target tab and pick your emulator.

COMMUNICATING WITH THE EMULATOR

- An application developer can communicate with a running emulator in two ways.
    - ➢ One way is through a **Telnet connection**, which is a network-type connection that requires a Telnet/SSH client on the development machine.
    - ➢ The second way to connect to a running emulator is through the **Android Debug Bridge (ADB).**
- The ADB is a stand-alone executable file that comes with the Android Development Kit (ADK).
    - ➢ We can run it by starting a console session on your development machine. If you are using windows:

    - Click the Start button or Windows icon (Windows 7).
    - In the search box, type **cmd**. This starts a terminal screen.
    - Navigate to the android-sdk directory, and in that directory we should find a subdirectory called platform-tools.
    - Navigate into the platform-tools directory and type **adb** (enter) on the command line to run the program.

- The **adb utility** also allows the user to connect to his actual Android device if it is connected through a cable.
- The **adb command** is used in conjunction with a keyword to cause a particular action. Some commands are as follows:



Command	Result
`adb install <application.apk>`	Installs a new application to the emulator or actual connected device.
`adb push <file on development machine> <location on emulator/ mobile device>`	Uploads a file to the emulator or connected device.
`adb pull <file on the emulator/ mobile device><location on development machine>`	Downloads a file from the emulator or connected device.
`adb shell`	Brings up a command line to the emulator or actual device that will respond to common Linux commands. (Remember, the core of the Android platform is Linux.) This is the easiest way to navigate the file system on the emulator or actual device.
`adb logcat`	Starts dumping debugging information to the screen. (Logcat is part of the debugging system in Eclipse as well, as we discussed in Chapter 2's section on debugging.

UNIT-IV

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. Android is ____________ and _______________ operating System.
2. To develop an Android application Eclipse must be configured with _____and _____.
3. R**.** java file is created based on _____________ and _______________ files.
4. **4.** Which method of the Activity class is called when the activity becomes visible to the user_________________?
5. **5.** Which method of the Activity class is called when the activity starts interacting with the user_________________?
6. **6.** Which method of the Activity class is called when the application resumes previous activity_________________?
7. **7.** Which method of the Activity class is called when the activity needs to be end_________________?
8. The two attributes necessary for any layout in android are__________, _________.
9. The two settings for layout attributes are _______________,________________.
10. Which classes need to be extended to use the functionality of Button and TextField Controls.
11. Which software need to be installed first to develop an android application______ [ ]
    a. a) Eclipse b) Android SDK c) JDK d)none
12. The directory created by Eclipse to store projects is named __[ ]
    a. a) Workbook b) Projects c) Eclipse d) Workspace
13. The "Install New Software" menu choice is found under which Eclipse menu? [ ]
    a. a) File b) Project c) Window d) Help
14. The base class for Android applications is the _____________.[ ]
    a. a) Applet class b) Activity Class c) Swing class d) none
15. Which XML files allow us to design the user interface without having to write a single line of java code? [ ]
    a. a) main.xml b) AndroidManifest.xml c) strings.xml d) both a&c e) both a&b
16. Which type of error is impossible for the IDE to detect and diagnose?
    a. a) Syntax b) Run-Time c) Logic d) Undefined [ ]
17. ______ Layout is used to design graphics in an application [ ]
    a. a) Absolute Layout b) Frame Layout c) Table Layout d) Relative Layout
18. _____ Layout is used to position the controls on the screen [ ]
    a. a) Absolute Layout b) Frame Layout c) Table Layout d) Relative Layout

19. If we want the layout component to stretch all the way from left to right, we have to set android:layout_width to_____________ [ ]
20. fill_parent b) wrap_content c) fill_content d) wrap_parent
21. If we do not want the layout component to stretch all the way from top to bottom, we have to set android:layout_height to_____________ [ ]
22. fill_parent b) wrap_content c) fill_content d) wrap_parent

SECTION-B

Descriptive Questions

1. Write the steps to install Eclipse and JAVA?
2. Draw and explain the Eclipse Package Explorer
3. Discuss about Debugging with Eclipse?
4. Draw and explain the Activity Class state Chart.
5. Explain about creating and configuring an Android Emulator?
6. Discuss about Communicating with the Emulator?
7. Explain about them following layouts:
   (i) Frame Layout (ii) Absolute Layout (iii) Table Layout (iv) Relative Layout
8. Explain about core components of Android?
9. Illustrate the Screen Layout and Main.xml File?

10 Manipulate the XML settings for **TextView** Control in the **main.xml** file.
11 Manipulate the XML settings for **Button** Control in the **main.xml** file.
12. Manipulate the XML settings for **EditText** Control in the **main.xml** file.
13. Develop an android application that implements **Button control**?
14. Illustrate the Emulator Responses based on the different **adb**Commands?
15. Develop an android application that implements Relative layout.
16. Develop an android application that places the controls by specifying x, y coordinates.

Objective:

To implement the user interface for android applications.

Syllabus:

User Interface controls and user interface:
Radio buttons, radio group, the spinner, date picker, buttons, array adapter. View class: combining graphics with a touch listener, canvas, bitmap, paint, motion event.

Outcomes:
Student will be able to:

- Understand the keyclasses and methods in RadioButtons
- Understand the keyclasses and methods in RadioGroup
- Understand the keyclasses and methods in Spinner
- Understand the keyclasses and methods in DatePicker
- Understand the keyclasses and methods in ArrayAdapter
- Combine the graphics with the Touch Listener
- Understand the keyclasses and methods in Canvas
- Understand the keyclasses and methods in Bitmap
- Understand the keyclasses and methods in Paint
- Understand the keyclasses and methods in MotionEvent

Radio Buttons:

- A radio button is a two-state button that can be either checked or unchecked. When the radio button is unchecked, the user can press or click it to check it.
- Radio buttons are normally used together in a **RadioGroup.** When several radio buttons live inside a radio group, checking one radio button unchecks all the others
- Radio Buttons are mutually exclusive (i.e) if one is selected, all the others are automatically deselected.

Key classes and Methods used in Radio Buttons:

Class	RadioButton
Package	android.widget
Extends	android.widget.CompoundButton
Overview	The radio button is a two-state widget similar to the CheckBox; however, once it is checked, the user cannot uncheck it.

Methods:

1) **Void setOnCheckChangedListener (CompundButton.OnCheck-ChangedListener occl)**
   Inherited from CompoundButton.
2) **toggle()**
   Forces the state to be changed from the current state to the opposite state.
3) **boolean isChecked()**
   This method returns the state of the Checkbox. Inherited from CompoundButton.
4) **boolean performClick()**
   Calls the view's onClickListener, if it is defined. Inherited from CompoundButton.

Radio Group:

- This class is used to create a multiple-exclusion scope for a set of radio buttons. Checking one radio button that belongs to a radio group uncheck any previously checked radio button within the same group.
- Initially, all of the radio buttons are unchecked. While it is not possible to uncheck a particular radio button, the radio group can be cleared to remove the checked state.

Key classes and Methods used in Radio Group:

Class	RadioButton
Package	android.widget

Extends	android.widget.LinearLayout
Overview	This class creates a multiple exclusion group of radio buttons. In other words. If one radio button is checked, all others will become unchecked. Only one radio button in a group can be checked at a time. Initially, all buttons are unchecked. RadioButtons are added to a RadioGroup in a Layout XML file similar to the way TableRows are added to a Table Layout.

Methods:

1) **Void setOnCheckChangedListener (CompundButton.OnCheck-ChangedListener occl)**
   Inherited from CompoundButton. Assigns a listener to the RadioGroup.
2) **Void clearCheck()**
   Clears all the RadioButtons in the group.
3) **Void Check (int ID)**
   Checks the RadioButton indicated by the ID.
4) **Void addView(View child, int index, ViewGroup.LayoutParams params)**
   Adds a child view with the specified layout parameters.
5) **Int getCheckedRadioButtonId()**
   Returns the identifier of the currently selected radio button in this group.
6) **Void clearCheck()**
   Clears the current selection.

Spinner:

- Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.
- The choices we provide for the spinner can come from any source, but must be provided through an **SpinnerAdapter**, such as an **ArrayAdapter** if the choices are available in an array or a **CursorAdapter** if the choices are available from a database query.
- If the available choices for our spinner are pre-determined, you can provide them with a string array defined in a string.xml

Key classes and Methods used in Spinner:

Class	Spinner

Package	android.widget
Extends	android.widget.AbsSpinner
Overview	A view that displays one child at a time and lets the user pick among them.

Methods:

1) **Void setOnItemSelectedListener (AdapterView.OnItemSelectedListener listener)**
   Assigns a listener to the Spinner. Inherited from AdapterView.
2) **setAdapter(SpinnerAdapter adapter)**
   Associates an adapter to the Spinner. The Adapter is the source of the items in the Spinner.
3) **setPrompt(CharSequence prompt)**
   Sets the prompt to display when the Spinner is shown.
4) **Int getCount()**
   Retrieves the prompt from the Spinner.
5) **CharSequence getPrompt()**
   Retrieves the prompt from the Spinner.
6) **Void setPromptId(int id)**
   Sets the prompt to display when the dialog is shown.
7) **Void setGravity()**
   Describes how the selected item view is positioned.
8) **Object getSelectedItem()**
   Returns selected item. Inherited from AdapterView.
9) **Long getSelectedItemID()**
   Returns selected item id. Inherited from AdapterView.

ArrayAdapter:

- An **adapter** is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and send the data to adapter view then view can takes the data from the adapter view and shows the data on different views like listview, gridview, spinner etc.
- **ArrayAdapter** is more simple and commonly used Adapter in android.
- Whenever you have a list of single type of items which is backed by an array, we can use ArrayAdapter. For instance, list of phone contacts, countries or names.

- By default, ArrayAdapter expects a Layout with a single TextView, If we want to use more complex views means more customization in grid items or list items, please avoid ArrayAdapter and use custom adapters.
- ArrayAdapter is an implementation of BaseAdapter so if we need to create a custom list view or a grid view then we have to create our own custom adapter and extend ArrayAdapter in that custom class. By doing this we can override all the function’s of BaseAdapter in our custom adapter.

Key classes and Methods used in Array Adapter:

Class	ArrayAdapter
Package	android.widget
Extends	android.widget.BaseAdapter
Overview	A BaseAdapter that is backed by an array of arbitrary objects. If the ArrayAdapter is used for anything other than a TextView, care must be taken to choose the correct constructor.

Methods:

1) **Void setDropDownViewResource (int resource)**
   Sets the layout resource to create the drop-down views.
2) **Void add(Object object)**
   Adds an object at the end of the array.
3) **Void remove(Object object)**
   Removes the specified object from the array.
4) **Void insert(Object object, int index)**
   Inserts the specified object at the specifiedindex position.
5) **Object getItem(int position)**
6) Returns the object at the specified position.
7) **Int getPosition(Object item)**
   Returns the position of the specified item in the array.

DatePicker:

- Android Date Picker allows you to select the date consisting of day, month and year in your custom user interface. For this functionality android provides DatePicker and DatePickerDialog components.

- DatePicker is a widget to select date. It allows you to select date by day, month and year. Like DatePicker, android also provides TimePicker to select time.

Key classes and Methods used in Date Picker:

Class	DatePicker
Package	android.widget
Extends	android.widget.FrameLayout
Overview	This class is a widget for selecting a date. The date is set by a series of Spinners. They date can be selected by a year, month, and day. Spinners or a CalenderView object.

Methods:

1) **init (int year, int month, int day, DatePicker.OnDateChangedListener onDateChangedListener)**
   This method sets the initial date for the Datepicker object and assigns onDateChangedListener.
2) **CalenderView getCalenderView()**
   Returns the day of month set on this object
3) **int getMonth()**
   Returns the month set on this object
4) **int getYear()**
   Returns the year set on this object.
5) **boolean getSpinnerShown()**
   Gets whether the Spinners are shown
6) **boolean isEnabled()**
   Gets whether the DatePicker is enabled.
7) **Void setEnabled(boolean enabled)**
   Sets the enabled state of this view.
8) **Void updateDate(int year, int month, int dayOfMonth)**
   Updates the current date.

DatePicker.OnDateChangedListener

Class	DatePicker.OnDateChangedListener (interface)
Package	android.widget
Extends	------

Overview	Signals that the user has changed the date on the DatePicker associated with this listener.

Methods:

Public abstract void onDateChanged (DatePicker view, int year, int month, int day)

Combining graphics with a touch listener

Up to now we have been using widgets included in the Android software development kit (SDK). Sometimes we want to build custom widgets that will actually extend a built-in base class.

<u>One Example for combining Graphics with a Touch Listener</u>

This example extends the View class and attaches a touch listener to become a "doodle pad." First, take a look at the main.xml for the project:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:id="@+id/base"
  >
<TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="My Doodle Pad"
  android:textSize="15pt"
  />
</LinearLayout>
```

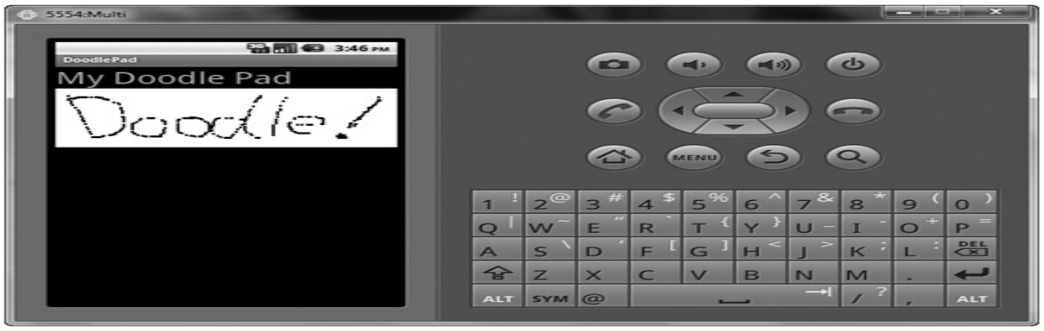
- ➢ In this XML file it contains only a LinearLayout in the background, and a single embedded TextView. What we should also notice is something it doesn't have: any kind of an element to represent a doodle pad on the screen.
- ➢ In this example, we will design that in the Java code and attach it dynamically to the screen.
- ➢ Here, android: id attribute is important because, we need to use the identifier in the Java code to attach our custom class object.

```
package com.sheusi.FingerDraw;

import android.app.Activity;
import android.os.Bundle;
import android.widget.*;
import android.content.Context;
import android.view.View.OnTouchListener;
import android.graphics.*;
import android.view.MotionEvent;
import android.view.View;
import java.util.ArrayList;

public class DoodlePad extends Activity
{
        /** Called when the activity is first created. */
        LinearLayout ll=null;
        DoodleView dv=null;
       @Override
        public void onCreate(Bundle savedInstanceState)
      {
                    super.onCreate(savedInstanceState);
                    setContentView(R.layout.main);
                    ll=(LinearLayout)findViewById(R.id.base);
                    dv=new DoodleView(this);
                    ll.addView(dv,1);
        }
}
class DoodleView extends TextView implements OnTouchListener
{
      ArrayList<dot> sketch =new ArrayList<dot>();
      Paint myPaint=new Paint();

      public DoodleView(Context myContext)
      {
                   super(myContext);
                   this.setBackgroundColor(Color.WHITE);
                   this.setHeight(120);
                   this.setFocusable(true);
                   this.setFocusableInTouchMode(true);
                   this.setOnTouchListener(this);
                   myPaint.setColor(Color.BLACK);
                  myPaint.setStyle(Paint.Style.FILL_AND_STROKE);
}
public void onDraw(Canvas c)
{
        for (dot d:sketch)
                c.drawCircle(d.x,d.y,2,myPaint);
}

public boolean onTouch(View v, MotionEvent me)
{
        if (me.getAction()==MotionEvent.ACTION_DOWN){}
        if (me.getAction()==MotionEvent.ACTION_MOVE)
        {
                sketch.add(new dot(me.getX(),me.getY()));
        }
        if (me.getAction()==MotionEvent.ACTION_UP){}
        invalidate();
        return true;
}
class dot
{
        private float x;
        private float y;
        //constructor
        public dot(float x, float y)
        {
                this.x=x;
                this.y=y;
        }
}
}
```

- Here we create a LinearLayout object variable and assign it using findViewById (R.id.base). This allows us to manipulate the object that appeared first in the main.xml file. You will notice that the source code actually has two classes: the "main" class that extends the Activity class and another class that extends the TextView class. This additional class defines our doodle pad.
- The doodle pad works by saving a series of dots and repainting the screen every time a dot is added.
- The "dot" class stores the x, y coordinates any time the user's finger is moved across the screen.

- Because **MotionEvent.ACTION_UP** and **MotionEvent. ACTION_DOWN** are empty methods, nothing will happen when the user touches the screen or the pad.
- The **invalidate( )** method causes the redraw.
- The **onDraw( )** method instructs the device to redraw all the dots that are in the ArrayList of dots called "sketch."
- Here we are adding a widget subclass to the screen in the Java code instead of the main.xml file. The following line does that:

  ll.addView (dv, 1);
- The name of the Linear Layout's variable is ll (el-el) and the addView ( ) method adds the widget represented by **dv** to position one in the list.
- The widget at position zero is the TextView specified in the main.xml file.



Our application in the emulator should look like

Canvas:

- The Canvas class holds the "draw" calls. To draw something, you need 4 basic components: A Bitmap to hold the pixels, a Canvas to host the draw calls (writing into the bitmap), a drawing primitive (e.g. Rect, Path, text, Bitmap), and a paint (to describe the colors and styles for the drawing).

Canvas	Paint
provides a method to draw a line	provides methods to define that line's color
has a method to draw a rectangle	defines whether to fill that rectangle with a color or leave it empty
defines shapes that you can draw on the screen	defines the color, style, font, and so forth of each shape you draw.

Key classes and Methods used in Canvas

Class	Canvas
Package	android.graphics
Extends	java.lang.Object

Overview	Holds the “draw” calls that write to a Bitmap

Methods:

1) **Void drawCircle(float cx, float cy, float radius, Paint paint)**
Draws a circle at the specified x,y coordinates with the specified radius and other features specified by the Paint object variable, such as the color.
2) **Void drawLine(float startX, float startY, float stopX, float stopY, Paint paint)**
Draws a Line from start x,y coordinates to stop x,y coordinates using features specified in the Paint object variable, such as the color.
3) **Void drawRect(float left, float top, float right. Float bottom, Paint paint)**
Draws a rectangle from top, left to bottom, right using features specified in the Paint object variable, such as the color
4) **Void drawCircle(float cx, flat cy, float radius, Paint paint)**
Draw the specified circle using the specified paint.
5) **Void drawColor(int color)**
Fill the entire canvas bitmap with the specified color.
6) **Void drawPicture(Picture picture)**
Save the canvas state, draw the Picture, and then restore the canvas state.
7) **Void drawPoint(float x, float y, Paint paint)**
Draws a single point at the coordinates indicated.
8) **int getWidth()**
Returns the width of the current drawing Layer
9) **int getHeight()**
Returns the Height of the current drawing Layer
10) **int save()**
Saves the current matrix and clip into a private stack.

Bitmap:

- We can say that the **bitmap** is the foundation of any use of Android graphics. The reason is that no matter how a graphic is specified or created, it is a bitmap that is eventually created and displayed on the screen.
- There are many ways of creating or obtaining a bitmap. We have already seen how a bitmap file, .jpg, gif or .png can be included in the drawable/ resource directory and displayed in an ImageView control.

Key classes and Methods used in Bitmap

Class	Bitmap
Package	android.graphics
Extends	java.lang.Object
Overview	Holds the “draw” calls that write to a Bitmap

Methods:

1) **boolean compress (Bitmap.CompressFormat format, int quality, OutputStream stream)**

   Writes a compressed version of the Bitmap to the output stream.

2) **Void copyPixelsFromBuffer (Buffer src)**

   Copy the pixels from the buffer, beginning at the current position, overwriting the Bitmap's pixels.

3) **Static Bitmap createBitmap(Bitmap source)**

   Returns an immutable Bitmap from the source Bitmap.

4) **final int getByteCount()**

   Returns the number of bytes used to store this Bitmap's pixels.

5) **int getDensity()**

   Returns the density for this Bitmap.

6) **Final int getHeight()**

   Returns the height of this Bitmap.

7) **Final int getWidth()**

   Returns the width of this Bitmap.

Paint:

The Paint class holds the style and color information about how to draw geometries, text and bitmaps.

Key classes and Methods used in Paint

Class	Paint
Package	android.graphics
Extends	java.lang.Object
Overview	Holds the "draw" calls that write to a Bitmap

Methods:

1) **setColor(int color)**

   Sets the Paint's color.

2) **setStyle(Paint.Style style)**

   Sets the Paint's style. Paint.Style sets how the line is filled, stroked, or both. Three choices are available: FILL, FILL_AND_STROKE, and Stroke.

3) **setTextSize(float size)**

   Sets the Paint's text size.

4) **Int getColor()**

   Returns the Paint's current color setting.

5) **Float measureText(String text)**

   Returns the width of the text.

6) **Void setStrokeWidth(float width)**

   Sets the width for stroking.

7) **Void setStyle(Paint.Style style)**

   Set the Paint's style, used for controlling how primitives' geometrics are interpreted.

Motion Event:

- Motion events describe movements in terms of an action code and a set of axis values. The action code specifies the state change that occurred such as a pointer going down or up. The axis values describe the position and other movement properties.
- For example, when the user first touches the screen, the system delivers a touch event to the appropriate View with the action code ACTION_DOWN and a set of axis values that include the X and Y coordinates of the touch and information about the pressure, size and orientation of the contact area.
- Some devices can report multiple movement traces at the same time. Multi-touch screens emit one movement trace for each finger. The individual fingers or other objects that generate movement traces are referred to as *pointers*. Motion events contain information about all of the pointers that are currently active even if some of them have not moved since the last event was delivered.

Key classes and Methods used in MotionEvent

Class	MotionEvent
Package	android.view
Extends	Android.view.InputEvent
Overview	Used to report movement

Symbolic constants used:

ACTION_DOWN	Signals the start of the gesture.Contans the starting location
ACTION_MOVE	Signals a change has happened since ACTION_DOWN
ACTION_UP	Signals a gesture has finished.Conatians the release location.

Methods:

1) **final long getDownTime(float size)**

Returns the time(in ms) when the user originally pressed Down

2) **final int getRawX()**

Returns the original raw X coordinate of the event.

3) **final int getActionX()**

Returns the kind of action being performed.

4) **final int getButtonState()**

Gets the state of all buttons that are presses, such as mouse or stylus button.

5) **final int getDeviceId()**

Gets the identification for the device that this event came from.

6) **final int getEventTime()**

Gets the time in milliseconds when this specific event was generated.

7) **final int getSource()**

Get the source of the event.

UNIT-V

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. The purpose of Motion event is____________________.
2. The class need to be extended to use the functionality of MotionEvent is ________________.
3. The purpose of ArrayAdapter is____________________.
4. The class need to be extended to use the functionality of ArrayAdapter is ________________.
5. The purpose of DatePicker is____________________.
6. The class need to be extended to use the functionality of DatePicker is ________________.
7. The purpose of Spinner is____________________.
8. The class need to be extended to use the functionality of Spinner is ________________.
9. The purpose of RadioGroup is____________________.
10. The class need to be extended to use the functionality of RadioGroup is ________________.
11. The important difference between checkboxes and radio buttons is _______________ [ ]
 a) their shape b) the shape of the checkmark
 c) that only one radio button can be checked at a time
 d) how many can go on a screen
12. Radio buttons on the same _____________ are mutually exclusive, i.e. only one can be used at once. [ ]
 a) screen b) LinearLayout c) RadioGroup d) column
13. There are actually two Java Date classes, one is found in the java.util package, the other is found in the java.____ package [ ]
 a) java.sql b) javax.swing c) java.io d) None of these
14. One feature of the Eclipse editor is showing or hiding blocks of code, which makes it easier to scroll through source code. Which of the following is the correct name for this in Eclipse? [ ]
 a) Code folding b) Expand/Collapse c) Section hiding d) Open/close
15. Many of the characteristics of coded graphics is determined by the _________ class. [ ]
 a) Paint class b) Brush class c) Pencil class d) Pen class
16. Testing the graphical user interface (GUI) during the early stages of development is best done using which of the following methods? [ ]
 a) Desk-checking b) On physical devices
 c) Using the emulator d) You can't test the GUI.
17. The DatePicker object responds to changes using ________________ listener. [ ]

a)OnDateChangedListener b) DateChangedListener.
C) onDateChanged() d) none

18. The TimePicker object responds to changes using ________________ listener. [ ]
a)OnTimeChangedListener b) TimeChangedListener.
C) onTimeChanged() d) none

19. _________ widget is ideal , when a programmer wants to contend with many items that must be displayed on the screen and many choices for a given item. [ ]
a) Spinner b) Radio Group c) CheckBox d) All the above

20. _______________ method writes a compressed version of the Bitmap to the output Stream. [ ]
a) int compress() b) string compress() c) boolean compress() d) none

SECTION-B

Descriptive Questions

1. Explain about the key classes and the methods used in Paint?
2. Explain about the key classes and the methods used in MotionEvent?
3. Explain about key classes and the methods used in Bitmap?
4. Explain about key classes and the methods used in RadioGroup?
5. Explain about key classes and the methods used in Spinner?
6. Explain about key classes and the methods used in Array Adapter?
7. Explain about the key classes and the methods used in DatePicker?
8. Explain about key classes and the methods used in Canvas?
9. Write a XML file to develop a TipCalculator application using Radio Buttons?
10. Write java source code for the TipCalculator using the XML file developed in the above question?
11. Write a XML file to develop a State& district Selection application using Spinner?
12. Write java source code for the State& district Selection using the XML file developed in the above question?
13. Write a XML file to select a date using DatePicker?
14. Write java source code to select a date using DatePicker using the XML file developed in the above question?

Objective:

To use best design practices for mobile development, designing applications for performance and responsiveness and also implement communication between the mobile devices.

Syllabus:

Android Applications working with images :display images ,using images stored on android devices ,image view, working with text files,working with data tables, using sqlite ,using xml for data exchange, cursor, content values ,XML PUL Parser, XML Resource parser. Client – server applications: socket, server socket ,HTTPURL connection ,URL.

Syllabus:

Student will be able to:

- Display image in an android application.
- Work with text files.
- Implement SQLite database.
- Use XML parsers to exchange data.
- Design client server applications.

Displaying Images:

- Displaying images on the Android device is fairly simple.
- Android supports the four common image formats: PNG, JPG, BMP, and GIF.
- Images that we want to bundle with our application are stored in the res directory.
- Users can have some control over the resolution of their images by storing three versions of the same image. The correct version will be displayed depending on the device the end user employs.
- We will notice that in the Package Explorer in Eclipse, the res directory has three subdirectories: drawable-ldpi (low resolution), drawable-mdpi (medium resolution), and drawable-hdpi (high resolution).
- Images are displayed in an ImageView object.
- Like a TextView object, an ImageView object can be added to the screen in the XML file that is used to lay out the screen, such as main.xml, or it can be declared and assigned in the Java code.
- A particular image can be assigned to an ImageView either in the XML file or in the Java code, the latter allowing the application to change the image while it is running.

Main.xml file for ImageView:

```
<?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
     >
    <TextView
              android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="@string/hello"
     />
     <ImageView
            android:id="@+id/myimage"
             android:layout_width="wrap_content"
             android:layout_height="wrap_content"
            android:src="@drawable/harbor"/>
    <ImageView
            android:id="@+id/myimage2"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
  </LinearLayout>
```

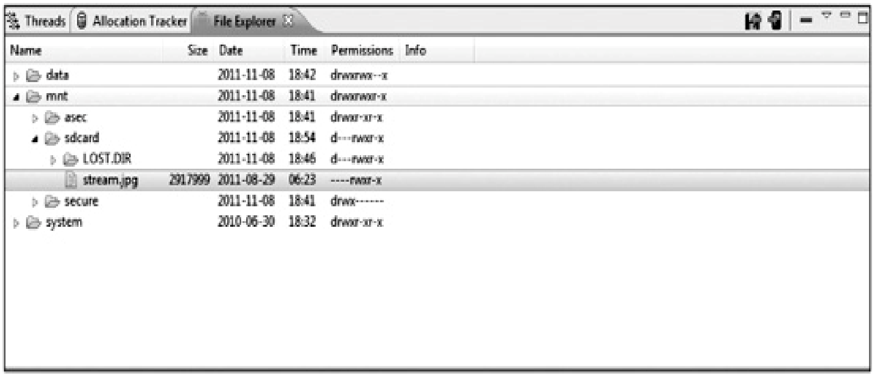
Java code:

```java
import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;

public class ImageDisplay extends Activity {
  /** Called when the activity is first created. */
  ImageView myImageView=null;
  ImageView myImageView2=null;
  @Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    myImageView=(ImageView)findViewById(R.id.myimage);
    myImageView2=(ImageView)findViewById(R.id.myimage2);
    myImageView2.setImageResource(R.drawable.lighthouse);
  }
}
```

USING IMAGES STORED ON THE ANDROID DEVICE

- Up to now we learned how to display default images with the application on the screen.
- Now we will learn how to pull images from the gallery on the mobile phone into our phone application.
- To select the photo from the gallery, we use the built-in activity ACTION_PICK.
- Android devices can store data items internally or externally (using an SD card).
- There are two constructs we will look at for retrieving data from these sources.

- The Android operating system (OS) can worry about where the data is stored internally on an actual device we just need to use the construct for internal storage. The same applies for the SD card. On the emulator we will use the SD card and the "external" code construct.
- To place images in the SD card
    - Under the Eclipse window menu, choose Android AVD and SDK Manager. Pick the emulator we wish to check and click the Edit button.
    - If we have already configured the emulator for an SD card, it will be in the list on the left. If we cannot find it in the list, select the New button to the right of the list and find SD card in the pick list seen here on the right; then click OK. After that, close all the pop-ups and get back to Eclipse.
    - Now take one or two small images we want to add to the SD card image in the emulator. Android will work with JPG, BMP, GIF, or PNG formats.
    - We can install images to the emulator, start it through the Android AVD and SDK Manager screens or just start an application in a project. When the emulator of choice is running, open the Window menu on Eclipse and choose Open Perspective. We have to choose DDMS.
    - DDMS stands for Dalvic Debug Monitor Server and is part of the Android plug-in for Eclipse; it's not part of the basic Eclipse configuration. If it is not on the list, open other and we should find it in the expanded list. When the perspective is open, find the tab named File Explorer. It will list the contents of the emulator just like the File Explorer does on our development machine.



Name	Size	Date	Time	Permissions	Info
data		2011-11-08	18:42	drwxrwx--x	
mnt		2011-11-08	18:41	drwxrwxr-x	
asec		2011-11-08	18:41	drwxr-xr-x	
sdcard		2011-11-08	18:54	d---rwxr-x	
LOST.DIR		2011-11-08	18:46	d---rwxr-x	
stream.jpg	2917999	2011-08-29	06:23	----rwxr-x	
secure		2011-11-08	18:41	drwx------	
system		2010-06-30	18:32	drwxr-xr-x	

- As you move in on the **mnt** folder, eventually you will arrive at the **sdcard** folder. Open the sdcard folder as you see in the figure.

- At the upper-right corner of the illustration, you will see two icons. One is an orange arrow pointing left on a floppy disk; the other is an arrow pointing right on a cell phone. The icon on the right is used to move files from an external source to the emulator.
- Select the latter and it will open a File Explorer on your development machine. Find the file(s) you want to transfer and load them to the emulator.

➢ Now we design image retrieving and viewing application.

➢ We need a simple main.xml file, which includes only a button to summon the image picker activity, and an ImageView element in which to load a chosen image.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="My Image Viewer"
/>
<Button
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="Touch to view gallery"
  android:id="@+id/gallerybutton"
/>
<ImageView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:id="@+id/imageview"
/>
```

```
        </LinearLayout>
```

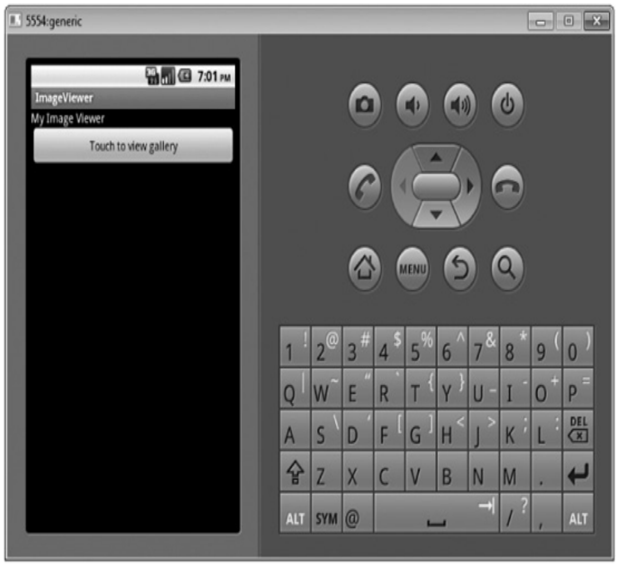
Java Code:

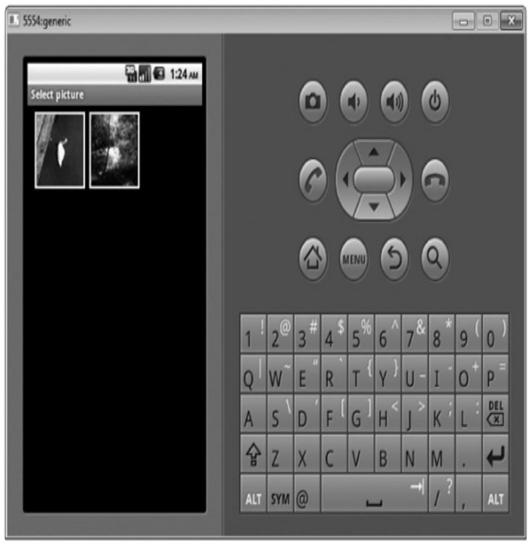
```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.widget.*;
import android.view.*;
import android.net.Uri;
public class ImageViewerActivity extends Activity {
  /** Called when the activity is first created. */
  Button b=null;
  ImageView iv=null;
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    b=(Button)findViewById(R.id.gallerybutton);
    iv=(ImageView)findViewById(R.id.imageview);
    b.setOnClickListener(new View.OnClickListener(){
      public void onClick(View v){
        Intent myIntent=new Intent(Intent.ACTION_PICK,
android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
       startActivityForResult(myIntent,2);
      }
    });
  }
   @Override
  public void onActivityResult(int requestID, int resultID, Intent i)
{
            super.onActivityResult(requestID,resultID, i);
   if (resultID==Activity.RESULT_OK){
    Uri selectedImage=i.getData();
     iv.setImageURI(selectedImage);
    }
  }
}
```

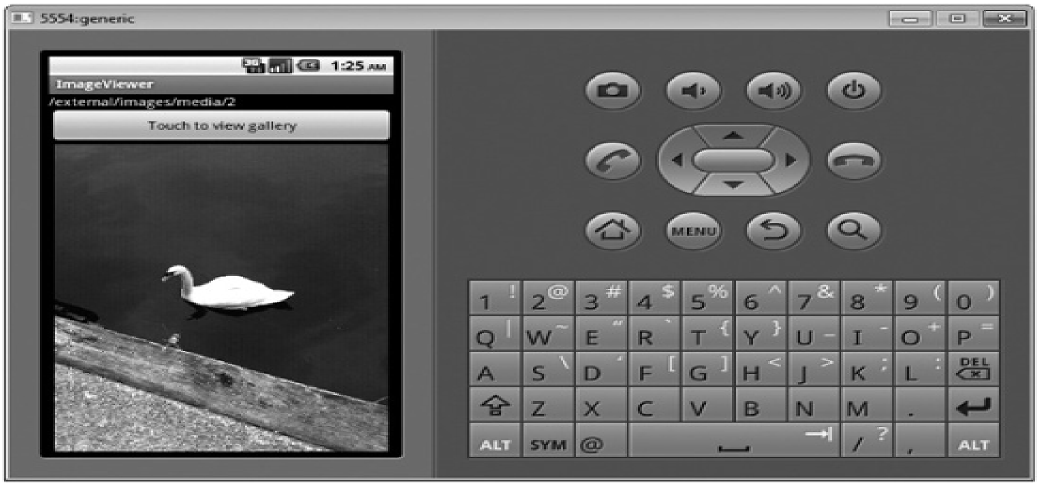
In these lines

```
Intent myIntent=new Intent(Intent.ACTION_PICK,
                      android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
       startActivityForResult(myIntent,2);
```

- The construct for sd card is EXTERNAL_CONTENT_URI.







Key class and methods used in ImageView:

Class	ImageView
Package	android.view
Extends	java.view.View
OverView	Displays an arbitrary image such as an icon. The ImageView class can load images from various sources, take care of computing its measurement from the image to properly display it, and show options such as scaling and tinting.

Methods:

1) Void setImageResource(int resid)

   Sets a Drawable object as this ImageView's content.
2) Void setImageUri(Uri uri)

   Sets the content for this ImageView to the specified uniform resource identifier.
3) Void setImageDrawable(Drawable drawable)

   Sets a Drawable object as the content of this ImageView.
4) Void setColorFilter(int color)

   Sets a tinting option for the image to be displayed.
5) Void setMaxHeight(int maxheight)

   Sets the maximum height for this ImageView.
6) Void setMAxWidth(int maxwidth)

   Sets the maximum width for this ImageView.
7) Void setBaseline(int baseline)

   Sets the offset of the widget's text baseline from the widget's top boundary.
8) Int getBaseline()

   Returns the offset of the widget's text baseline from the widget's top.

WORKING WITH TEXT FILES

- Our application design requires the ability to store and retrieve persistent data.
- We can do this in two ways:
  - a freeform text file, such as text notes,
  - Organized data such as a data table.
- In android, there are two choices for the application designer, regarding text files.
  - A text file can be built as part of an application and delivered with the application, but these text files will be read-only.
  - To build the application so the user can create, modify, and delete his own files.
- To create a text file that the application will use,
  - Create a new folder in the res folder by right-clicking the res folder in the Package Explorer and choosing New, Folder.
  - Call the folder raw, which indicates to Eclipse and the Android software development kit (SDK) that it will contain media and plain text files.

}

- Right-click on new folder name and select New, File. Name the file params.txt and place a few words of text in it.

- R.java file: The first run-through of the application will create an R.java file similar to the following code. The class called raw, came from our folder creation of the same name, and the field called params, based on our naming of the text file.

```
package com.sheusi.FileWerx;
public final class R {
  public static final class attr {
  }
  public static final class drawable {
    public static final int icon=0x7f020000;
  }
  public static final class id {
    public static final int FileViewer=0x7f060000;
  }
  public static final class layout {
    public static final int main=0x7f030000;
  }
  public static final class raw {
    public static final int params=0x7f040000;
  }
  public static final class string {
    public static final int app_name=0x7f050001;
    public static final int hello=0x7f050000;
  }
}
```

- Main.xml file: Next, we modify the main.xml file and add a TextView to display the contents of the text file. Generally, the text file would not necessarily be displayed, and it doesn't have to contain readable text; instead, it may contain data that is only important to the application. But now, we will read it and some parameters are to the TextView element .

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
<TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/hello"
  />
  <TextView
android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:minLines="3"
  android:maxLines="3"
  android:textColor="#000000"
  android:textSize="12px"
  android:background="#ff0000"
  android:text=""
  android:id="@+id/FileViewer"
  />
</LinearLayout>
```

- Java file: Finally, we can take a look at the Java file. First, we cannot simply link to a raw resource, in this case the text file, by using findViewByID( ), because a file is not an object of the View class. Instead, we use the openRaw-Resource( ) method of the Resources class. The following are the steps:
  - .
  - The getResources( ) method of the Activity class assigns the Resourses class variable.
  - The openRawResource( ) call returns an InputStream from the file
  - We wrap that in a DataInput-Stream object to make the file contents retrievable.
  - Finally, we assign the contents to a String and display it in the TextView we created for that purpose.

```
package com.sheusi.FileWerx;
import android.app.Activity;
import android.os.Bundle;
import android.widget.*;
import java.io.*;
import android.content.res.Resources;
public class FileWerx extends Activity {
  /** Called when the activity is first created. */
 InputStream is=null;
 DataInputStream dis=null;
 Resources myResources=null;
 TextView tv=null;
 @Override
public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
 myResources=this.getResources();
 tv=(TextView)findViewById(R.id.FileViewer);
 is=myResources.openRawResource(R.raw.params);
 dis=new DataInputStream(is);
 String someText=null;
 try{
   someText=dis.readLine();
 }catch(IOException ioe){
   someText="Couldn't read the file";
 }
 tv.setText(someText);
}
}
```

Working with SQLite Database:

- SQLite is an open source SQL database that stores data to a text file on a device.
- Android comes in with built in SQLite database implementation.
- SQLite supports all the relational database features. In order to access this database, we don't need to establish any kind of connections for it like JDBC, ODBC e.t.c.

Database - Package

- The main package is **android.database.sqlite** that contains the classes to manage our own databases.

Database – Creation:

- In order to create a database you just need to call **openOrCreateDatabase**() method with our database name and mode as a parameter.

Syntax:

SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",PRIVATE_MODE,null);

Database – Insertion:

- We can create table or insert data into table using **execSQL()** method defined in SQLiteDatabase class.

Syntax:

mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TableName(Username VARCHAR,Password VARCHAR);");

Database – Retrieving:

- We can retrieve anything from database using an object of the Cursor class.
- We will call a method of this class called **rawQuery()** and it will return a resultset with the cursor pointing to the table.
- We can move the cursor forward and retrieve the data.

Syntax:

Cursor resultSet=mydatabase.rawQuery("Select * from TableName",null);

resultSet.moveToFirst();

String username=resultSet.getString(0);

String password=resultSet.getString(1);

Key class and Methods used in SQLiteDatabase:

Class	SQLiteDatabase

Package	android.database.sqlite
Extends	android.databse.sqlite.SQLiteClosable
Overview	This class exposes methods used to manage a SQLite database.

Methods:

1) **openOrCreateDatabase("Database name", Mode, null);**

   creates a database with the specified name and specified mode. The third parameter is a Cursor factory object which can be set to null.

2) **Void execSQL(String sql)**

   Executes a single SQL statement that is not a SELECT statement or any other statement that returns data.

3) **Cursor query(String table, String[] columns, String selection, String[] selectionargs, String having, String groupBy, String orderBy,int length)**

   The query() method takes 8 parameters.

   Those are: Database name, String array of columns, Selection, Selection Criteria, GroupBy Specification, having Specification, Orderby Specification, number of rows to be retrieved.

4) **Cursor rawQuery(String sql, int limit)**

   Executes the SELECT statement and limit is the number of records to be returned.

5) **Void close()**

   Closes the databse.

6) **boolean isReadOnly()**

   Returns whether the database is opened read-only.

Key class and Methods used in Cursor:

Class	Cursor (Interface)

Package	android. Database
Extends	--------------------
Overview	The interface provides random access to the result set returned by a query.

Methods:

1) **int getCount()**

   Returns the number of rows in the Cursor (Result set)

2) **boolean moveToFirst()**

   Moves the cursor to the first row

3) **boolean moveToNext()**

   Moves the cursor to the next row.

4) **boolean moveToLast()**

   Moves the cursor to the last row.

5) **Boolean moveToPosition (int position)**

   Moves the Cursor to an absolute position.

6) **int getColumnCount()**

   Returns the number of columns in the Cursor (result set)

7) **int getInt(int columnIndex)**

   Returns the int value from the column specified by columnIndex.

8) **double getDouble(int columnIndex)**

   Returns the double value from the column specified by columnIndex.

9) **Void close()**

   Closes the cursor, releasing all of its resources and making it invalid.

10) **float getFloat(int columnIndex)**

    Returns the value of requested column index as a float.

11) **Long getLong()**

    Returns the value of the requested columns as a Long

12) **String getString(int columnIndex)**

    Returns the value of the requested column as a string

13) **Boolean isFirst()**

    Returns whether the cursor is pointing to the first row.

14) **Boolean isLast()**

    Returns whether the cursor is positioning to the last row.

Key Class and Methods used in ContentValues:

Class	ContentValues
Package	android.content
Extends	java.lang.Object
Overview	This class is used to store a set of values that the Content Resolver can process

Methods:

1) **Void put(string key, String value)**

   Adds a value of String type to the set

2) **Void put(String key, int value)**

   Adds a value of Integer type to the set.

3) **Void put(String key, Double value)**

   Adds a value of double type to the set.

4) **Void put(String key, Long value)**

   Adds a value of Long type to the set.

5) **Void put(String key, Float value)**

   Adds a value of Float type to the set.

6) **Void put(String key, Boolean value)**

   Adds a value of Boolean type to the set.

7) **Double getAsDouble(String key)**

   Retrieves a value and converts it to the Double type.

8) **Float getAsfloat(String key)**

   Retrieves a value and converts it to the Float type.

9) **Integer getAsInteger(String key)**

   Retrieves a value and converts it to the Integer type.

10) **Long getAsLong(String key)**

    Retrieves a value and converts it to the Long type.

11) **String getAsString(String key)**

    Retrieves a value and converts it to the String type.

12) **Void remove(String key)**

    Removes a single value.

13) **Int size()**

    Returns the number of values.

USING XML FOR DATA EXCHANGE

- The plain text files can be used to manage unorganized data and SQLite is used to manage the most structured data,

- We can turn our attention to a third alternative: XML files.
- XML, Extensible Markup Language, is a way to manage organized data while allowing us to create our own structure to suit our purpose.
- Anything from a document to an RSS feed to a web commerce site is structured with XML.
- XML distinguishes from an SQL data table in following ways:
    1. XML file is plain text. No database management overhead is needed. Because of this, we don’t have to worry about exchanging data between incompatible database management systems; we can convert data from one system to XML, and convert back at the other end.
- In XML, documents have to be structurally perfect, well-formed. All the tags have to be in perfect pairs, with perfect placement.
- XML does not display data.. Because they are simply text files, they need no special handling. They just organize data so the target application can make sense of what is being delivered. This is why formatting of these files must be done correctly.
- We have to write a utility that extracts data from an XML file (we call such a utility a *parser*). A programmer can use one of many prewritten utilities that are designed to handle the basic structure of an XML document and not be concerned with the specific tag names and document tag levels. An Android programmer can use packages that are written into the Java Development Kit (JDK), the Android Development Kit (ADK).
- There are two essential sides to using XML files:
    - ➢ assembling and exporting a file,
    - ➢ disassembling and displaying its contents.
- Several packages are available to handle all the operations necessary to process an XML file. One of the easiest to use is the **org.xmlpull.v1 package.**

- The following example shows a simple XML file and displays its contents with a small amount of formatting. The XML file, called book.xml, is a simple file consisting of two “book” elements with title and author elements in them, surrounded by a root tag called “inventory.” Following is a listing of book.xml:

```
<?xml version="1.0" encoding="utf-8"?>

<inventory>

<book>

<title>android tutorial</title>

<author>jim sheusi</author>

</book>


<book>

<title>info security</title>

<author>tom calabrese</author>

</book>

</inventory>
```

  - There is only one instance of the inventory tag pair: the root element.
  - Each book’s record would have values for the title and the author.
  - We would have two tag pairs, title and author, within each book pair. Each pair opens and closes within another open-close pair.

- XML file is processed as follows:
  - ➢ Include an XML file in the application. To do this, we create a sub-folder in the resources (res) folder in the Package Explorer of Eclipse. Just right-click

on the res folder, Choose New from the menu, and pick Folder. Name the new folder xml, in lowercase.

- ➢ Right-click on the new xml folder and choose New, then File, and you will get a clean editing window. Name the file book.xml and type the above code in that file.

- The application itself only needs an EditText field, so your **main.xml file** can look like the following example:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  >
<TextView
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="XML Parser"
/>
<EditText
  android:layout_width="fill_parent"

 android:layout_height="wrap_content"
 android:id="@+id/et1"
/>
```

```
</LinearLayout>
```

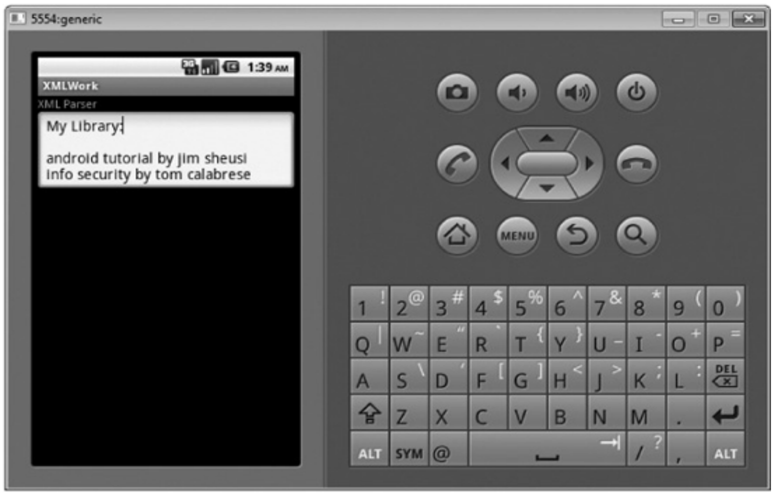
- The java code is as follows:

```
package com.sheusi.XMLWork;

import android.app.Activity;

import android.content.res.Resources;

import android.os.Bundle;

import java.io.*;

import android.widget.*;

import android.util.Log;

import org.xmlpull.v1.*;

import android.content.res.*;

public class XMLWorkActivity extends Activity {

  /** Called when the activity is first created. */

  EditText et1=null;

  InputStream is=null;

  Resources myRes=null;

  @Override

  public void onCreate(Bundle savedInstanceState){

    super.onCreate(savedInstanceState);

    setContentView(R.layout.main);

    et1=(EditText)findViewById(R.id.et1);

    this.getRecordsFromXML(this);
```

```
}
private void getRecordsFromXML(Activity activity){
  try{
    Resources res=activity.getResources();
   XmlResourceParser xrp=res.getXml(R.xml.book);
   xrp.next();// skips descriptor line in XML file
   int eventType=xrp.getEventType();
   while(eventType!=XmlPullParser.END_DOCUMENT){
   //while we haven’t reached the end of the xml file
     if(eventType==XmlPullParser.START_DOCUMENT){
       et1.append("My Library:\n");
     }
      if(eventType==XmlPullParser.START_TAG){
       if(xrp.getName().equals("book "))
      et1.append("\n"+xrp.getName()+":");
       if(xrp.getName().equals("author"))
         et1.append(" by ");
       if(xrp.getName().equals("title"))
      et1.append("\n");
     }
     if(eventType==XmlPullParser.END_TAG){}
     if(eventType==XmlPullParser.TEXT){
          et1.append(xrp.getText());
        }
        eventType=xrp.next();
      }
    }catch(Exception e){
      et1.append("App Error");
    Log.e("xml_error",e.getMessage());
   }
  }
 }
```

- ***"activity"*** is the parameter passed to the XML processing method. It represents the main activity of the application here. We create an instance of the XmlResourceParser class and assign it the file that we put in the xml subfolder of the res folder.
- The line xrp.next(); causes the code to skip this line in processing.
- the getName( ) method refers to what's in the tags, and getText( ) refers to the values between the tags.
- The XmlPullParser class has several symbolic constants that represent integer variables. The symbolic constants have names that adequately describe the events that can be encountered when processing the incoming XML document stream.
  - ❖ The START_DOCUMENT and END_DOCUMENT are associated with the outermost and innermost tags in the document, and the START_TAG and END_TAGevents are associated with other pairs as they are encountered. The parser uses the TEXT symbolic constant to represent the encounter with data between tags.

- **getName( ) method** is used to match the tags encountered with the various tags .
- We pull the data value from between the tags with the **getText( ) method.**



My Library:

android tutorial by jim sheusi
info security by tom calabrese

Key Class and Methods used in XML PullParser:

Class	XMLPullParser (interface)
Package	xmlpull.vl
Extends	------------
Overview	An interface that defines parsing functionality

Methods:

1) **int getAttributeCount()**

   Returns the number of attributes for the current start tag.

2) **String getAttributeName(int index)**

   Returns the local name of the attribute specified by the index parameter if namespaces are enabled, or just the attribute name if namespaces are disabled.

3) **String getAttributeValue(int index)**

   Returns the given attribute's value.

4) **String getAttributeType(int index)**

   Returns the type of the specified attribute. If the parser is nonvalidating it must return CDATA.

5) **int getDepth()**

   Returns the current depth of the element

6) **int getEventType()**

   Returns the type of the current event (START_TAG, END_TAG, and so an).

7) **String getInputEncoding()**

   Returns the input encoding, if known; null otherwise.

8) **int getLineNumber()**

   Returns the current line number stating at 1.

9) **String getText()**

   Returns the text content of the current event as a String.

Key Class and Methods used in XMLResourceParser:

Class	XMLResourceParser (Interface)
Package	android.content.res
Extends	Implements AttributeSet, XmlPullParser

Overview The XML parsing interface returned for an XML resource. This is the standard XmlPullParser interface.

Methods:

1) **int getAttributeCount()**

   Returns the number of attributes for the current start tag

2) **String getAttributeName(int index)**

Returns the local name of the attribute specified by the index parameter if namespaces are enabled, or just the attribute name if namespaces are disabled.

3) **String getAttributeValue(int index)**

   Returns the given attribute’s value

4) **int next()**

   Gets next parsing event

5) **Void close()**

   Closes this interface to the resource.

Key Class and Methods used in Socket:

Class:	Socket
Package:	java.net
Extends:	java.lang.Object
Overview	This class implements client sockets (also called just "sockets"). A socket is an endpoint for communication between two machines.

Methods:

1) Inputstream getInputStream()

   Returns an input stream for this socket.

2) **Outputstream getOutputStream()**

   Returns an output stream for this socket.

3) Void connect(socketAddress address, int timeout)

   Connects this socket to the server.

4) int getLocalPort()

   Returns the local port number to which this socket is bound.

5) **InetAddress getInetAdress()**

   Returns the address to which the socket is connected.

6) Void close()

   Closes this socket.

7) int getPort()

8) boolean isConnected()

Key Class and Methods used in ServerSocket:

Class:	ServerSocket
Package:	java.net
Extends:	java.lang.Object
Over View	This class implements server sockets. A server socket waits for requests to come in over the network. It performs some operation based on that request, and then possibly returns a result to the requester.

Methods:

1) **Socket accept()**

2) **int getLocalPort()**

   Returns the local port number to which this socket is bound.

3) **InetAddress getInetAdress()**

   Returns the address to which the socket is connected.

4) **int getSoTimeout()**

   Returns setting for SO_TIMEOUT.

5) **boolean isBound()**

   Returns the binding state of the socket.

6) **boolean isClosed()**

   Returns the closed state of the socket.

7) **void bind(SocketAddress endpoint)**

   Binds the socket to a local address.

Key Class and Methods used in HttpURLConnection:

Class:	HttpURLConnection
Package:	java.net
Extends:	java.lang.Object
Overview:	Each HttpURLConnection instance is used to make a single request but the underlying network connection to the HTTP server may be transparently shared by other instances.

Methods:

1) InputStream getInputStream()
2) String getRequestMethod()
3) Void disconnect()
4) Boolean usingProxy()
5) Permission getPermission()
6) String getResponseMessage()

Key Class and Methods used in URL:

Class:	URL
Package:	java.net
Extends:	java.lang.Object

Overview The **Java URLConnection** class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

Methods:

1) URLConnection openConnection()
2) InputStream openStream()
3) Uri toUri()
4) int getPort()
5) String getHost()
6) String getPath()
7) Object getConnect()

UNIT-VI

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. The three resolution folders available under res directory for image in android are ______, _________ and __________.
2. Images are displayed by using ___________ object.
3. The image formats supported by android are ________, __________, _________ and __________.
4. __________________________ method is called to set a drawable object as ImageView's content.
5. _________________ Built-in activity is used to select photo from the gallery.
6. The purpose of XML pull parser is____.
7. The purpose of XML resource parser is _______________________________________.
8. The purpose of ContentValues is ____.
9. The purpose of cursor is____.
10. The _________ folder under **res** directory indicates to Eclipse and SDK,

   that it will contain media and plain text files.
11. The ___________ file contains the words of text in raw file. [ ]

   a) Params.txt b) res.txt c) raw.txt d) all the above
12. If the cursor returns zero, it indicates that [ ]

   a) There is no such table b) there is a table

   c) there is table but no data d) none
13. The package that is to be imported for a sqlite database operations

is________ [ ]

a) android.sqlite b) android.database.sqlite

c) android.database d)both b & c

14. __________ seperates XML file from an SQL data table [ ]

a) XML file is a plain text b) No database management overhead is needed.

c) No problem with exchanging data between incompatible database management systems

d) all the above

15. The package that is to be extended for XML operations is_____ [ ]

a) org.xmlpull.v1 b) xmlpull.org.v1 c) xml.org.v1 d) pull.org.v1

16. ___________ interface defines parsing functionality. [ ]

a) XML pull parser b) XML resource parser c) XML parser d) both a & b

17. _________ interface provides random access to the result set returned by a query. [ ]

a) SQLite b) XML parser c) Cursor d) both a & c

18. ____________ method is called to execute a single SQL statement that is not a SELECT statement or any other statement that return data.

a) query() b) insert() c) execSQL() d) update() [ ]

SECTION-B

Descriptive Questions

1. Explain about Key classes and methods used in ImageView?
2. Explain about Key classes and methods used in SQLiteDatabase?
3. Explain about Key classes and methods used in Cursor?
4. Explain about Key classes and methods used in ContentValues?
5. Explain about Key classes and methods used in XML PullParser?
6. Explain about Key classes and methods used in XML ResourceParser?
7. Write main.xml file and java code for displaying image in android application.
8. Explain about the common image formats supported by Android.
9. Illustrate how images can be displayed in Android with an example program?
10. Illustrate how XML is used to exchange data?

11. Illustrate the procedure to create a text file that can be used by android application with main.xml file.
12. Illustrate the procedure to create the database and tables in SQLite database?
13. Illustrate the procedure to search a particular record in data table in SQLite database?
14. Illustrate the procedure to insert or edit a record in data table in SQLite database?
15. Develop an android application to retrieve an image from photo gallery on your phone and display it in the application.