

LAB MANUAL

Course: CSC303- Mobile Application Development



Department of Computer Science

Java Learning Procedure

- 1) Stage **J** (Journey inside-out the concept)
- 2) Stage **a₁** (Apply the learned)
- 3) Stage **V** (Verify the accuracy)
- 4) Stage **a₂** (Assess your work)

Table of Contents

Lab #	Topics Covered	Page #
Lab # 01	Java and Android Basics, A brief introduction to android	03-27
Lab # 02	GUI, Layouts, XML based approach	28-37
Lab # 03	Activity, Fonts and Colors, More GUI Components	38-51
Lab # 04	Menus, Layout Managers and Event Listeners	52-67
Lab # 05	Graphical Primitives and Databases	68-78
Lab # 06	Lab Sessional 1	
Lab # 07	Multithreading in Android	79-84
Lab # 08	Timer Application and Camera API	85-90
Lab # 09	ListView	91-94
Lab # 10	Services and Broadcast Receivers	95-110
Lab # 11	Audio and Video Player Development	111-119
Lab # 12	Lab Sessional 2	
Lab # 13	Playing with SDcard (File CRUD Operations)	120-130
Lab # 14	Working with RSS feed	131-140
Lab # 15	MAPs dealing in android	141-172
Lab # 16	App Publication	173-179
	Terminal Examination	

Statement Purpose:

The goal of this lab is to learn the fundamentals of developing Android Applications, from project creation to installation on a physical device. More specifically you should gain the knowledge of how to use basic development tools to support the application development process, as well as the key components of an Android application itself

Activity Outcomes:

After completing this chapter student will be able to understand the following topics.

- Android Development Environment
- Creating an App in Android Studio
- Anatomy of Android Project
- Running the First App
- Ingredients of an Android App

Instructor Note:

These labs assume that you know the Java Programming Language. If you are not yet a Java programmer, consider taking Java Complete Reference J2SE/J2EE. It will get you ready for completing these labs.

1) Stage J (Journey)

Introduction

Android is a mobile operating system that is based on a modified version of Linux. Android, as a system, is a Java-based operating system that runs on the Linux kernel. The system is very lightweight and full featured. It was originally developed by a startup of the same name, Android, Inc. In 2005, as part of its strategy to enter the mobile space, Google purchased Android and took over its development work. Google wanted Android to be open and free; hence, most of the Android code was released under the open source Apache License, which means that anyone who wants to use Android can do so by downloading the full Android source code.

The main advantage of adopting Android is that it offers a unified approach to application development. Developers need only develop for Android, and their applications should be able to run on numerous different devices, as long as the devices are powered using Android.



Android Versions






Android has gone through quite a number of updates since its first release. The follow table shows the various versions of Android and their codenames.



Features of Android

As Android is open source and can be customized by the manufacturers, there are no fixed hardware or software configurations. Android itself supports the following features:

- ✚ **Storage** — Uses SQLite, a lightweight relational database, for data storage.
- ✚ **Connectivity** — Supports GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, and WiMAX. **Messaging** — Supports both SMS and MMS.
- ✚ **Web browser** — Based on the open source WebKit, together with Chrome's V8JavaScript engine.
- ✚ **Media support** — Includes support for the following media: H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP

-  **Hardware support** — Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, and GPS
-  **Multi-touch** — Supports multi-touch screens
-  **Multi-tasking** — Supports multi-tasking applications
-  **Flash support** — Android 2.3 supports Flash 10.1.
-  **Tethering** — Supports sharing of Internet connections as a wired/wireless hotspot

Android Architecture Linux Kernel

Since the android architecture is built on top of Linux Kernel, the Kernel is used as a hardware abstraction layer. i.e, It can be used as an abstraction layer between hardware and software. For example, consider the battery function. The devices which run on android will have hardware component like camera, blue tooth, battery etc. The power management interface from the Kernel will interact with the battery (hardware). There is a list of low level interfaces provided by the Kernel such as camera driver, audio drivers, memory management, display driver etc. **Libraries** The libraries are the APIs which contains the features of android operating system. Android Runtime Contains two elements:

Dalvik Virtual Machine

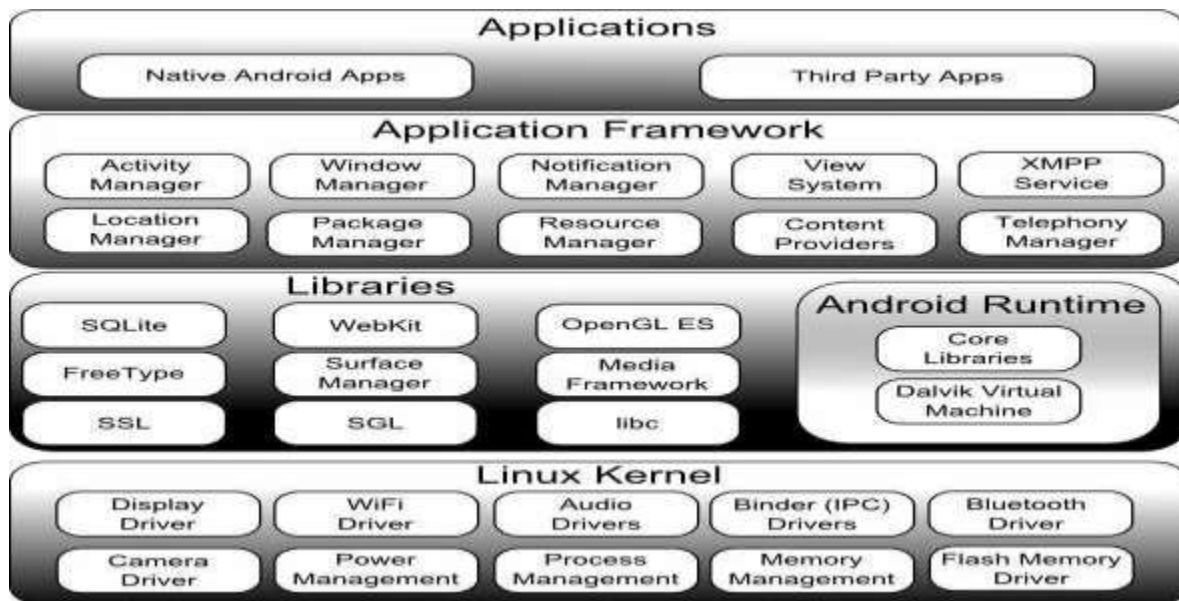
Core Libraries

Dalvik Virtual Machine

The Dalvik Virtual Machine (DVM) is available under Android runtime. The Dalvik virtual machine executes .dex files. The .dex files are provided by DX tool which works as a compiler for android SDK. This DX tool is explained in android SDK tools. The Dalvik virtual machine (DVM) does the job of Java virtual machine (JVM). The role of Dalvik virtual machine is similar to java virtual machine. While java virtual machine executes byte code, on the other side, in android programming Dalvik virtual machine executes Dalvik byte code (.dex files).

Application Framework

The application framework contains libraries for various areas like power management, multimedia, storage access and graphical user interface. The libraries in the framework are written in Java and run on top of the core libraries of the Android Runtime. The application framework of android provides managers for various purposes such as resources handling, content providers, system wide notification etc. The android applications are using the services of these managers. **Application** On Android, every application runs in its own process, each of which runs in its own virtual machine (VM). Applications are the android applications which are executed on a device. The applications are the top layer in the android architecture. This layer will consists of pre-installed and our own applications. The android application we develop goes into this layer.



2) Stage a1 (apply)

Lab Activities:

Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

- “Class” means a category of things
- “Object” means a particular item that belongs to a class
- Also called an “instance”

Activity 1: Example of ARRAYS is given below.

```

} public class TestArray
{
public static void main(String[] args) {
double[] myList= {1.9, 2.9, 3.4, 3.5};
// Printing all array elements
for (inti= 0; i< myList.length; i++) {
System.out.println(myList[i] + " "); }}

```

Activity 2: Creating class in OOP

Class Declaration

```

class Circle {
double radius = 1.0;
double findArea(){
return radius * radius * 3.14159; }}

```

Objects in a Single Step (Creating objects)

```

ClassName objectReference= new ClassName();

```

Accessing Objects

- Referencing the object's data:

objectReference.data

myCircle.radius

•Invoking the object's method:

objectReference.method


myCircle.findArea()

Constructors

- Constructors are a special kind of methods that are invoked using the new operator when an object is created.
- Constructors play the role of initializing objects.
- Constructors must have the same name as the class itself.
- Constructors do not have a return type—not even void.
- Constructor with no parameters is referred to as a *default constructor*.

Activity 3: An example of calculations of area of circles of different colors is given below

```
// Define the Circle class
public class Circle { // Save as "Circle.java"
    double radius; String color;
    // Constructors (overloaded)
    public Circle() { // 1st Constructor
        radius = 1.0;
        color = "red";
    }
    public Circle(double r) { // 2nd Constructor
        radius = r;
        color = "red";
    }
    public Circle(double r, String c) { // 3rd Constructor
        radius = r;
        color = c;
    }
    // Public methods
    public double getRadius() {
        return radius;
    }
    public String getColor() {
        return color;
    }
    public double getArea() {
        return radius*radius*Math.PI;
    }
}
```

 Output of Test Class

Radius is 2.0 Color is blue Area is 12.566370614359172

Radius is 2.0 Color is red Area is 12.566370614359172

Radius is 1.0 Color is red Area is 3.141592653589793

Keyword "this"

```
public class Circle {  
    double radius; // Member variable called "radius"  
    public Circle(double radius) { // Method's argument also called "radius"  
        this.radius = radius;  
        // "this.radius" refers to this instance's member variable  
        // "radius" resolved to the method's argument.  
    }  
}
```

Java -Inheritance

- Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

- The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

Activity 4: Calculator of two numbers

```
class Calculation{  
    int z;  
    public void addition(int x, int y){  
        z = x+y;  
        System.out.println("The sum of the given numbers:"+z);  
    }  
    public void Substraction(int x,int y){  
        z = x-y;  
        System.out.println("The difference between the given numbers:"+z);  
    }  
    public class My_Calculation extends Calculation{  
        public void multiplication(int x, int y){  
            z = x*y;  
            System.out.println("The product of the given numbers:"+z);  
        }  
        public static void main(String args[]){  
            int a = 20, b = 10;  
            My_Calculation demo = new My_Calculation();  
            demo.addition(a, b);  
            demo.Substraction(a, b);  
            demo.multiplication(a, b);  
        }  
    }  
}
```

OUTPUT

The sum of the given numbers:30
The difference between the given numbers:10
The product of the given numbers:200

Java -Interfaces

An interface is a reference type in Java, it is similar to class, it is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviour of an object. And an interface contains behaviours that a class implements.

Activity 5: an example of interface in java with two methods declaration

```
/* File name : Animal.java */  
interface Animal {  
    public void eat();  
    public void travel();}
```

Implementing Interfaces

- When a class implements an interface, you can think of the class as signing a contract, agreeing to perform the specific behaviors of the interface.
- A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Activity 6: Interface implementation

```
/* File name : MammalInt.java */  
public class MammalInt implements Animal{  
    public void eat(){  
        System.out.println("Mammal eats");  
    }  
    public void travel(){  
        System.out.println("Mammal travels");  
    }  
    public int noOfLegs(){  
        return 0;  
    }  
    public static void main(String args[]){  
        MammalInt m = new MammalInt();  
        m.eat();  
        m.travel();  
    }  
}
```

Activity 7: Instance Variables (“Fields” or “Data Members”)

```
class Ship1 {  
    public double x, y, speed, direction;  
    public String name;  
}  
public class Test1 {  
    public static void main(String[] args) {  
        Ship1 s1 = new Ship1();  
        s1.x = 0.0;  
        s1.y = 0.0;  
        s1.speed = 1.0;  
        s1.direction = 0.0; // East  
        s1.name = "Ship1";  
    }  
}
```

```

System.out.println(s1.name + " is initially at ("
+ s1.x + "," + s1.y + ").");
s1.x = s1.x + s1.speed
* Math.cos(s1.direction * Math.PI/ 180.0);
s1.y = s1.y + s1.speed
* Math.sin(s1.direction * Math.PI/ 180.0);
System.out.println(s1.name + " has moved to ("
+ s1.x + "," + s1.y + ").");
}
}

```

 **Output:**

Ship1 is initially at (1,0).
Ship2 has moved to (-1.41421,1.41421).

Activity 8: Methods Activity

```

class Ship2 {
public double x=0.0, y=0.0, speed=1.0, direction=0.0;
public String name = "UnnamedShip";
private double degreesToRadians(double degrees){
return(degrees * Math.PI/ 180.0);
}
public void move(){
double angle = degreesToRadians(direction);
x = x + speed * Math.cos(angle);
y = y + speed * Math.sin(angle);
}
public void printLocation(){
System.out.println(name + " is at ("
+ x + "," + y + ").");}}
public class Test2 {
public static void main(String[] args) {
Ship2 s1 = new Ship2();
s1.name = "Ship1";
Ship2 s2 = new Ship2();
s2.direction = 135.0; // Northwest
s2.speed = 2.0;
s2.name = "Ship2";
s1.move();
s2.move();
s1.printLocation();
s2.printLocation();
}
}Output:

```

Ship1 is at (1,0).

Ship2 is at (-1.41421,1.41421).

Activity 9: Setting Up the Development Environment

Download/Install the SDK

For in-depth instructions, visit [Android Installation Documentation](https://developer.android.com/sdk/index.html). Otherwise perform the following steps.

- Go to <http://developer.android.com/sdk/index.html>.
- Unpack to a convenient location - Remember the full path to this location, we will refer to it as **<android_sdk_dir>** for the rest of the lab.
 - For example, on Linux your home directory is a convenient location.
 - **<android_sdk_dir>** would then be `/home/<username>/android_dir`.
- Add the path to the **<android_sdk_dir>/tools** directory to your system PATH
 - Linux (Lab Machines Running Fedora):
 1. Using your favorite text editor, open the `.mycshrc` file in your home directory.
 2. Add the following text to the end of the file:

```
set path=($path <android_sdk_dir>/tools)
set path=($path <android_sdk_dir>/platform-tools
```
 3. Open up a terminal, navigate to your home directory and execute the following command:

```
source .mycshrc
```
 - Windows:
 1. Right-click **My Computer**.
 2. Click **Properties**.
 3. Click **Advanced** tab.
 4. Click **Environment Variables** button.
 5. Double Click **Path** under **System Variables**.
 6. Add `;<android_sdk_dir>/tools;<android_sdk_dir>/platform-tools` to the end of the **Variable Values** text field.
 - Mac:
 1. Using your favorite text editor, open the `.bash_profile` in your home directory.
 2. Add the following text to the end of the file:

```
export PATH=$PATH:<android_sdk_dir>/tools
export PATH=$PATH:<android_sdk_dir>/platform-tools
```
 3. Open up a terminal, navigate to your home directory and execute the following command:

```
source .bash_profile
```

- Navigate to your <android_sdk_dir>/tools directory and type **android**. Add the appropriate components. See step 4 in <http://developer.android.com/sdk/installing.html>.
- Test your installation by running **adb** from the command line. If you did everything right, you should get a long list of help instructions.

Download/Install the Eclipse Plugin

- It is recommended that you use Eclipse 3.4 or later
 - **Lab Machines** - Fedora Eclipse based on 3.4.2
The version of Eclipse used by the lab machines is missing a vital component and requires adding an additional Eclipse plugin in order to use the Android plugin:
 1. Click the menu **Help** -> **Software Updates**.
 2. Click the tab **Available Software** -> **Add Site** button.
 3. Enter <http://download.eclipse.org/releases/ganymede> into the **Location** field.
 4. Click **OK** button.
 5. Enter **WST Common UI** into the search/text box at the top of the window (give it a second, it tries to search as you type and its kind of slow).
 6. Click the checkbox next to **WST Common UI**.
 7. Click the **Install** button.
 8. Click the **Next** button.
 9. Accept the terms, click **Finish**.
 10. Restart Eclipse.
 11. Follow the steps in the next bullet **3.4 Ganymede**.
 - 3.4 Ganymede:
 1. Click the menu **Help** -> **Software Updates**.
 2. Click **Available Software** tab -> **Add Site** button.
 3. Enter <https://dl-ssl.google.com/android/eclipse/> into the "**Location**" field.
 4. Click **OK** button.
 5. Click the checkbox next to **Developer Tools**.
 6. Click the **Install** button.
 7. Click the **Next** button.
 8. Accept the terms, click **Finish**.
 9. Restart Eclipse.

- 3.5 Galileo:
 1. Click **Help** -> **Install New Software** .
 2. Click **Add...** button.
 3. Enter a name for the site into the **Name** field.
 4. Enter `https://dl-ssl.google.com/android/eclipse/` into the **Location** field.
 5. Click **OK** button.
 6. Click the checkbox next to **Developer Tools**.
 7. Click the **Next** button.
 8. Accept the terms, click **Finish**.
 9. Restart Eclipse.
- Point Eclipse to <android_sdk_dir>:
 1. Click the menu **Window** -> **Preferences**.
 2. Click **Android** from the Hierarchy view on the left hand side.
 3. Enter `<android_sdk_dir>` into the **SDK Location** field.
 4. Click the **Apply** button.
 5. Click the **OK** button.

Download/Install the SDK Platform Components

At the time of writing this lab there are eight different versions of the Android Platform available, ranging from 1.1 to 2.2. It is best practice to develop for the oldest platform available that still provides the functionality you need. This way you can be assured that your application will be supported by as many devices as possible. However, you will still want to download newer versions of the platforms so that you can test your applications against these as well. Due to the size of each platform component you will only be required to download and develop on one platform for the whole class. We will target the highest platform that the G1 phones support, Android 1.6 (API 4). Before we can begin developing we must download and install this platform:

- Select the menu **Window** -> "**Android SDK and AVD Manager**", or click on the black phone shaped icon in the toolbar.
- Select **Available Packages** on the left hand side.
- Expand the Google Android site in the "**Site, Packages, and Archives**" Tree.
- Check the following items:
 - **SDK Platform Android 1.6, API 4 Revision 3**
 - **Google APIs by Google Inc., Android API 4, Revision 2**

- **NOTE:** Those of you developing on Lab Machines should follow these instructions: <http://sites.google.com/site/androidhowto/how-to-1/set-up-the-sdk-on-lab-machines-linux>.
- Click **Install Selected**.
- Accept the Terms for all packages and click **Install Accepted**.

We're now ready to develop our application.

Activity 10: Create "Hello World" Application

Create a new Android Project

- Open Eclipse.
- Click the menu **File -> New -> Project**.
- Expand the **Android** folder and select **Android Project**.
- Name the project `lab1<userID>`
 - For instance, if you userID is *jsmith*, you would name your project `lab1jsmith`.
- You can change the location of where you would like to save the project by un-selecting the "Default Location" check box, and supplying your own location.
- Check "**Android 1.6**" from the Build Target List.
 - *This identifies that the project is being built to be compatible with Android versions 1.6 and later.*
 - *Its generally preferred that you choose the lowest build number possible, so as to be compatible with the largest number of existing systems in place.*
 - *This build target can be changed any time later on through the Project Properties menu.*
- Fill in the Properties:
 - **Application Name** = `Hello World!`
 - *This is the friendly name of the application, that shows up on the device.*
 - **Package Name** = `edu.calpoly.android.lab1<userID>`
 - *This is the namespace for the project, follows standard Java conventions.*
 - **Create Activity** = `HelloWorld`
 - *This optional field will automatically create a "Main Activity" class for the project. You can think of the Main Activity as the Home Page for your application.*
 - **Min SDK Version** = `4`
 - *This specifies the minimum API Level on which your application can run. By default this is set to the API Level of the Build Target Platform. As new API's are added to newer Versions, their API levels increase as well. A Program*

that uses an API Level of four won't be able to run on a platform that has a lower API Level.

- Click "**Finish**".

Activity 11: Take a Tour of the Application

The application you've just created is very similar to other java applications you may have created in Eclipse. Look in the **Package Explorer** side bar. Notice that the Android Development Toolkit(ADT) has generated a number of folders and files for you:

- **src:** If you expand this out you'll see the package hierarchy you previously entered. This is where your source code files will go.
 - **HelloWorld.java:** This is the auto-generated stub Activity Class with the name you entered into the project creation wizard. We'll add some code to this later.
- **Android 1.6:** This is the version of the library you had chosen in the project creation wizard. The application will be built using this version of 'android.jar'
- **res:** This folder will contain all of the resources (a.k.a. external data files) that your application may need. There are three main types of resources that you will be using and the ADT has created a subdirectory for each.
 - **drawable:** This folder will hold image and animation files that you can use in you application.
 - *It already contains a file called icon.png which represents the icon that Android will use for your application once it is installed*
 - **layout:** This folder will hold xml layout files that the application can use to construct user interfaces. You will learn more about this later, but using a layout resource file is the preferred way to layout your UI.
 - *It already contains a file called main.xml which defines the user interface for your 'HelloWorld.java' Activity class. Double clicking von this file will open up the Android UI Editor that you can use to help generate the xml layout files.*
 - **values:** This folder will hold files that contain value type resources, such as string and integer constants.
 - *It already contains a file called strings.xml. Double clicking on this file will open up the Android Resource Editor. Notice that there are two strings in there already, one of which is named 'app_name'. If you select this value, on the right hand side of the editor you should see the Application Name you entered in the project creation wizard. You can use this editor to add new resources to your application.*
- **gen:** This folder will contain Java files that get auto-generated by ADT. Notice that it already contains one file called "R.java".
 - **R.java:** This is a special static class that is used for referencing the data contained in your resource files. If you open this file you will see a number of static inner classes for each of the resource types, as well as static constant integers within them. Notice

that the names of the member variables are the same as the names of the values in your resource files. Each value in a resource file is associated with an integer ID, and that ID is stored in a member variable of the same name, within a static class named after its data type.

- *The 'app_name' resource value has an ID and is of value type 'string'. The ADT automatically adds an integer constant to the R.string class and names it 'app_name'.*
 - *A debugging hint: Occasionally, an Android project will report errors in Eclipse that do not show up in any source code file. Sometimes you can fix this by deleting R.java. When you rebuild your project, R.java gets generated, and perhaps your mysterious errors will disappear.*
 - *A second hint: I encourage you to turn on build automatically in Eclipse (Project menu).*
- **assets:** This folder is for asset files, which are quite similar to resources. The main difference being that anything stored in the 'assets' folder has to be accessed in the classic 'file' manipulation style. For instance, you would have to use the AssetManager class to open the file, read in a stream of bytes, and process the data. You will not be using assets quite as extensively as you will be using resources.
- **AndroidManifest.xml:** Every project has a file with this exact name in the root directory. It contains all the information about the application that Android will need to run it:
 - Package name used to identify the application.
 - List of Activities, Services, Broadcast Receivers, and Content Provider classes and all of their necessary information, including permissions.
 - System Permissions the application must define in order to make use of various system resources, like GPS.
 - Application defined permissions that other applications must have in order to interact with this application.
 - Application profiling information.
 - Libraries and API levels that the application will use.
- **default.properties:** This file contains all of the project settings, such as the build target you chose in the project creation wizard. If you open the file, you should see 'target=4', which is your build target. You should never edit this file manually. If you wish to edit the project properties, do so by right-clicking the project in the 'Package Explorer' panel, and selecting 'Properties'.

The project creation wizard has written the 'Hello World' application for you already. A string resource containing the display text has been placed into the res\values\strings.xml file. The value is named 'hello'.

The xml UI layout has been added to res\layout\main.xml. While you can use the Android Layout Editor to create your xml layout, you can also code them manually yourself. Let's take a look at this file:

- **Right-Click** on the file.
- Select **Open With -> Text Editor**.

Notice the Top Level node, **Linear Layout**, which defines the style of layout this file will be using. This 'Linear Layout' is perhaps the most basic, and specifies that UI elements will be laid out in a continuous line. It has three properties: orientation, width, and height.

Notice the Second Level node, **TextView**, which defines a UI element for displaying text. It has three properties: width, height, and the text to display. Notice that the text property is not set to "Hello World!". Instead it is set to reference the resource value which contains the text we want to display. In this case we are choosing to display the contents of the 'hello' value. We do this by using the '@' symbol, followed by the value type of the resource (which is a 'string'), followed by the name of the value (which is 'hello').

Activity 12: Run "Hello World" on the Emulator

On the Emulator

Before we can run the application, we need to setup an Android Virtual Device(AVD), or emulator, to run it on:

- Select the menu **Window** -> "**Android SDK and AVD Manager**", or click on the black phone shaped icon in the toolbar.
- Select **Virtual Devices** on the left hand side.
- Click the **New...** button.
- Give your AVD a name.
- Select the target build that we would like to run the application on, "**Android 1.6 - API Level 4**".
- Click **Create AVD** and close out the SDK/AVD Manager.

We're now ready to run our application.

- Select the menu **Run** -> **Run**.
 - *Note: The emulator may take a long time to start up.*
 - *Note: Another way to run your application is to right-click on the project in the Package Explorer, then select **Run As** -> **Android Application**.*
- You can interact with the emulator using the mouse just like you would with a device. To get started, press the Menu key to see the home screen.

Congratulations! You've just created and an Android Application.

On a Physical Device

Before we can run the application on a physical device we need to modify the project, make a configuration change on the phone, and install some drivers for the phone on our development machine. We begin by making your project declare itself as debuggable. It's possible to do this through the Android Manifest Editor that the ADT provides, however doing this manually helps you understand the manifest better:

- Project Modifications
 - From the Package Explorer, double-click the file **AndroidManifest.xml**.
 - Select the tab labeled **AndroidManifest.xml** along the bottom.

- Add `android:debuggable="true"` to the inside of the opening `<application>` tag.
- Save the file and close it.
- Phone Modifications
 - Turn the phone on.
 - Navigate to the **Home** screen.
 - Press **MENU** (the physical button on the device).
 - Select **Settings** -> **Applications** -> **Development**.
 - Enable the **USB debugging** option.
- Installing the Android USB drivers
 - Mac OS X: Don't need to install drivers, it should just work.
 - Windows: [Follow instructions here](#)
 - Linux: [Follow instructions here](#)

Ensure the device is properly connected. Run the application as you would normally. The "Hello World!" app should start on the phone.

Activity 13: Simple Activity Classes

There are four major types of component classes in any Android application:

- **Activities:** Much like a Form for a web page, activities display a user interface for the purpose of performing a single task. An example of an Activity class would be one which displays a Login Screen to the user.
- **Services:** These differ from Activities in that they have no user interface. Services run in the background to perform some sort of task. An example of a Service class would be one which fetches your email from a web server.
- **Broadcast Receivers:** The sole purpose of components of this type is to receive and react to broadcast announcements which are either initiated by system code or other applications. If you've ever done any work with Java Swing, you can think of these like Event Handlers. For example, a broadcast announcement may be made to signal that a WiFi connection has been established. A Broadcast Receiver for an email application listening for that broadcast may then trigger a Service to fetch your email.
- **Content Providers:** Components of this type function to provide data from their application to other applications. Components of this type would allow an email application to use the phone's existing contact list application for looking up and retrieving an email address.

In this lab, we will be focusing on what Activities are and how they are used. We will cover the other components in later labs. If you would like more information on these components, visit the Android overview page for [Application Components](#).

In case you haven't figured it out by now, you have already created one of these component classes. That's right, the HelloWorld class is an Activity Class. It's a simple user interface designed to greet the user. In the section that follows, we'll make our application more personal by adding a new Activity class to ask for the user's name. We'll then update the existing HelloWorld greeting Activity to display that name.

*Note: If you ever have problems getting things to compile in Eclipse, you might try **Project -> Clean**. You can also try to delete R.java under res, then **Project -> Build Project**.*

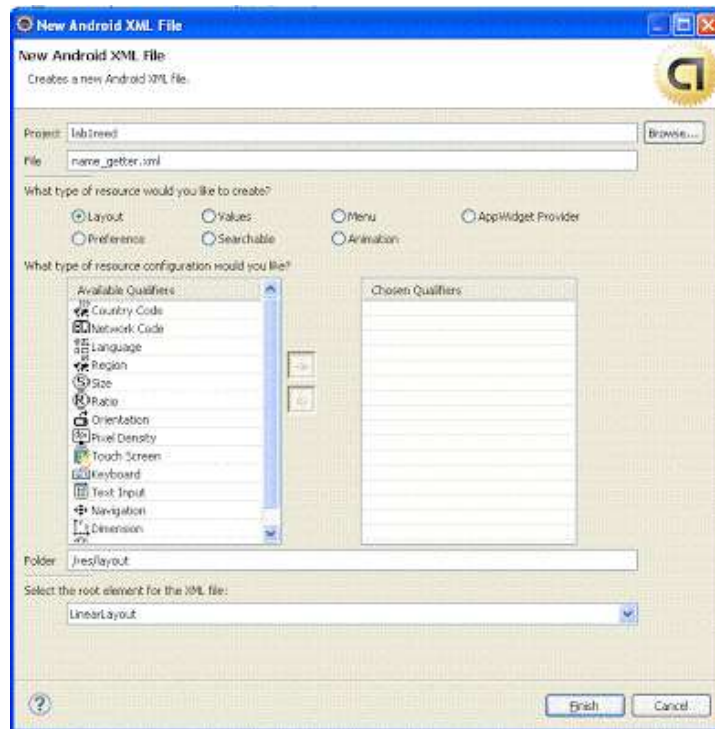
Getting the User's Name

To get the user's name, you will be creating an Activity class which will allow the user to enter their name into a text field and press a button when finished to proceed to the HelloWorld greeting Activity. There are three separate steps to accomplish here. You must first layout your user interface in XML. Then you must create the Activity class to parse the input from the user and initiate the HelloWorld Activity. Finally, you will have to reconfigure the application to use your new name retrieval Activity on startup.

5.1.1 Create the User Interface

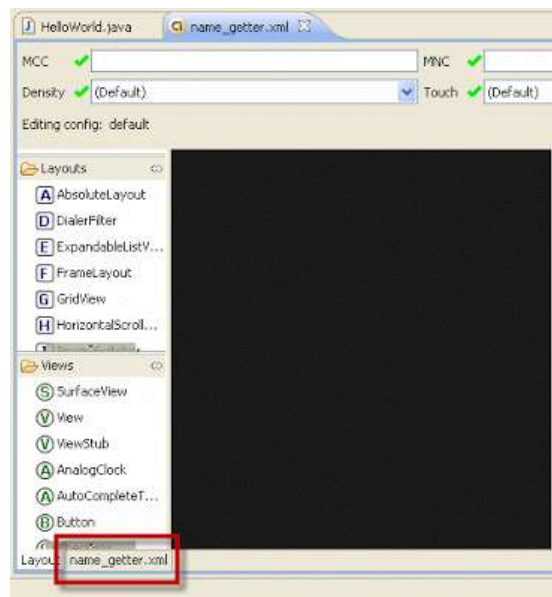
Android allows you to layout your user interfaces using a simple XML specification. We will go into more depth on this topic in the next lab, so for now you will be setting up a basic interface using four different GUI elements. Begin by creating a new Android XML file:

- Select the menu **File -> New -> Android XML File**.
*If **Android XML File** does not appear in the menu:*
 - Select **Other**.
 - Expand the **Android** folder.
 - Select **Android XML File** and click **Next**.
- Ensure the Project matches the name of your project and that the folder is **/res/layout**.
 - *Layout files should always go in this folder.*
- Enter "**name_getter.xml**" as the file name
 - *The name of your layout files must only contain lower case letters, the numbers 0-9, underscores '_', or periods '.'*
 - *[a-z0-9_.]*
- Select the **Layout** radio button.
- Select **LinearLayout** from the "Select the root element..." drop down and click **Finish**.



- By default, the file will be opened to the Layout Editor tab. Select the tab labeled **name_getter.xml** to switch to the XML Editor.
 - *This should be located in the bottom left corner of the Layout Editor.*

Each GUI element derives from the View base class. The first element was added for you when you created the XML layout file and selected LinearLayout from the dropdown menu. You should be able to see an XML opening and closing tag labeled LinearLayout in the editor. Each XML layout file must have a single root view tag, inside which all other view tags are nested. The LinearLayout tag tells Android to arrange elements contained inside it in a straight line in the order in which they appear. Let's make a few modifications to the LinearLayout by editing the attributes contained in the opening LinearLayout tag:

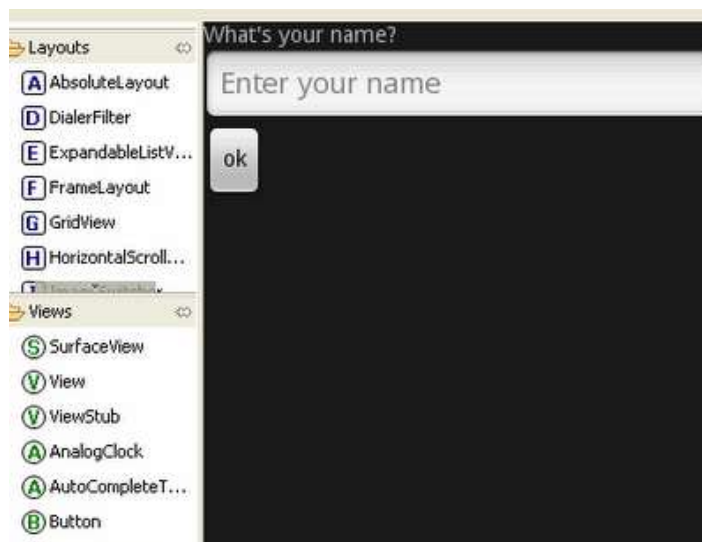


- Set the attributes labeled `android:layout_width` and `android:layout_height` to `"fill_parent"` (Include the quotes).

- *This tells Android that the `LinearLayout` should take up all the available width and height on the screen.*
- Add an attribute labeled `android:orientation` and set it to `"vertical"`.
 - *This tells Android that elements nested inside the `LinearLayout` should be laid out in a column, as opposed to a single row as indicated by `"horizontal"`.*

Lets add the other three UI elements to our XML layout file:

- Switch back to the Layout Editor tab.
 - *The three elements we will add all reside under the folder icon labeled "Views"*
 - *This can be seen along the left hand side of the previous figure, about halfway down*
- Scroll down to the item labeled **TextView**.
 - Click and drag the **TextView** onto the black canvas.
 - *The Layout Editor will pre-populate the label with its auto-generated id, which may look somewhat strange.*
- Repeat the previous step for the **EditText** and **Button** labels.
 - *Remember, order matters for the `LinearLayout`.*
- This is what you want your UI to look like. However it may not resemble this quite yet:



- Switch back to the XML Editor to change the attributes of the elements you just added.
 - *Notice that all the UI elements you added are nested within the `LinearLayout` Element.*

- *There will always be only one root element. You may, however, nest other Layout elements within each other.*
- Editing the **TextView** element:
 - This is the label that prompts the user to enter their name. It displays the value contained in the `android:text` attribute.
 - Set this attribute to ask the user what their name is.
 - The `android:id` attribute provides a variable name for referencing this element from within the code.
 - Id's are specified with the syntax of `@+id/MyId01`.
 - MyId01 is the handle used to identify the element from within your application code via the Static R class.
- Editing the **EditText** element:
 - This is the text field where the user will input their name. It will default to contain the value set by the `android:text` attribute.
 - Remove this attribute, we don't need it.
 - The `android:hint` attribute provides a hint to the user when the field is empty, and disappears when text is entered.
 - Set this attribute to instruct the user to enter their name.
 - Either make a mental note of the `android:id` attribute or provide your own variable name which you will use to reference this element from within the code.
- Editing the **Button** element:
 - This is the button that will allow the user to continue to the next HelloWorld greeting screen.
 - It displays the value contained in the `android:text` attribute.
 - Set this attribute to something like "ok", "next", or "submit".
 - Either make a mental note of the current value for the `android:id` attribute or provide your own variable name which you will use to reference this element from within the code.

Create the Activity Class

Using the HelloWorld Class from section 2.1 as an example, create a new class that extends `android.app.Activity` class and implements `android.view.View.OnClickListener` interface.

- Implement the `OnClickListener` interface by creating a method stub with the following signature: `public void onClick(View v)`.
 - *We'll fill in this method later*
- Declare a member variable of the type `android.widget.EditText`
 - *This will hold a reference to the text field in which the user will enter their name, the same one that you added to the `name_getter.xml` layout file.*
- Add a method with the following signature: `public void onCreate(Bundle savedInstanceState)`.
 - *This method will be called when the Activity starts and is where initialization of local and member data will be done.*
- Inside this method perform the following:
 - make a call to `super.onCreate(savedInstanceState)`
 - This should always be done and is to ensure that any necessary parent class initializations are performed.
 - make a call to `this setContentView(R.layout.name_getter)`
 - When you created the XML layout file earlier, the Android Eclipse Plugin automatically added a static constant to the static `R.layout` class in the `R.java` file under the `/gen` folder. This constant variable has the same name of the file and its value is used to identify the layout file.
 - This call tells Android to create a screen based off of the layout file.
 - Make a call to `this.findViewById(R.id.<EditText id>)` and set your `EditText` member variable equal to the return value.
 - `<EditText id>` should be replaced with the `android:id` that was specified in the `name_getter.xml` layout file for the `EditText` element.
 - You will have to explicitly cast the return value to the type of `EditText` as this method only returns objects of type `Object`.
 - *This static constant value was added in the same way as it was done for the `R.layout.name_getter` value and serves the same purpose.*
 - Make a call to `this.findViewById(R.id.<Button id>)` and set a local `android.widget.Button` reference equal to the return value.
 - `<Button id>` should be replaced with the `android:id` that was specified in the `name_getter.xml` layout file for the `Button` element.
 - Make a call to `button.setOnClickListener(this)`.

- **button** should be replaced with the local Button reference you just retrieved.
- Fill in the onClick method stub:
 - Retrieve the user entered text from the text field and keep it in a local String variable
 - Create an android.content.Intent object: `new Intent(this, HelloWorld.class)` .
 - We'll use the Intent object to start the HelloWorld greeting activity and pass it information
 - *We'll discuss Intents in more depth in later labs, essentially we use them to interact with other application components.*
 - You will use the Intent.putExtra(<key>, <value>) method to pass the user entered name to the HelloWorld greeting Activity. This method functions like a hashmap, where values can be stored by associating them with a string key and later retrieved with the same key. You can retrieve this hashmap later by calling getExtras().
 - Make a call to `<intent>.putExtra(<key>, <value>)` ,
 - `<intent>` should be replaced with the intent object you just created.
 - `<key>` should be replaced with a string you will use to access the user's name later.
 - `<value>` should be replaced with the user's name obtained from the text field. To obtain the text from the EditText object, you must call to `<editText object>.getText().toString()` .
 - Make a call to `this.startActivity(<intent>)`
 - *This command will initiate the switch to the HelloWorld greeting Activity.*

Reconfigure the HelloWorld Application

The Android Manifest contains information on all of the Application's components, including which component should be used in different scenarios. We need to tell our application to use the new activity on startup instead of the HelloWorld Activity which it was previously using. Begin by Double-Clicking the AndroidManifest.xml file to open it

- Like the Layout file there is also a Manifest Editor. Switch to the XML Editor by clicking the **AndroidManifest.xml** tab along the bottom of the editor
- Find the first opening `<activity ... >` tag and change the attribute labeled `android:name` equal from `".HelloWorld"` to `"<New Activity Class>"`
 - Replace `<New Activity Class>` with the full name of your new Activity Class.
 - Look at the `package` attribute in the `<manifest>` tag, this declares the top level package for the application. If your activity resides in a sub-package, then you must also include the sub-package in the name of your Activity class.

- For example:
 - application package name in the manifest tag: `android:package="my.app.basepackage"`
 - "Activity1"
 - fully qualified classpath: `my.app.basepackage.Activity1`
 - activity tag: `<activity android:package="Activity1"></activity>`
 - "Activity2"
 - fully qualified classpath: `my.app.basepackage.subpackage.Activity2`
 - activity tag: `<activity android:package="subpackage.Activity2"></activity>`
- *The Intent Filter tag you see nested inside the Activity tag is what tells the application that this Activity is the Main, or startup, Activity.*
- Add the following opening and closing Activity tag pair underneath the Activity tag pair you just modified, nested inside the Application tag:
 - `<activity android:name="HelloWorld" ></activity>`
 - *This declares to the Android device that the application has an Activity component named HelloWorld. If you don't add this tag, Android will not let you launch this Activity*

At this point, you should be able to try running the application. The application should start up and display the name retrieval activity you just created. You should be able to enter your name in the text field and hit the button, after which the old HelloWorld greeting activity should appear.

Greeting the User

Now that we've got the name retrieval activity completed, let's update the HelloWorld greeting Activity to print out the name the user entered. In the onCreate method of HelloWorld.java:

- Make a call to `this.getIntent().getExtras()` to retrieve the hashmap in which you placed the user entered name.
 - This will return an `android.os.Bundle` object which acts like a hashmap.
 - You should check to ensure that this value is not null.
- Retrieve the user entered name from the Bundle object.
 - Make a call to the bundle's `getString(<key>)` method.
 - `<key>` should be replaced with the key String you used to put the user entered name into the hashmap in the name retrieval Activity's onClick method.
 - You should check to ensure that this value is not null.

- Get a reference to the TextView object in your main.xml layout file (you may need to add an id field).
- Set the text of the TextView object to say `Hello <name>!`
 - `<name>` should be replaced with the user entered name that you retrieved from the bundle object.
- Run your application.

Activity 14: Exporting Your Application

In order for your application to be run on a physical device, it must be digitally signed with a certificate. The certificate does not need to be signed by a certificate authority like Verisign. It's acceptable to use a self-signed certificate. As of now, you've only been executing your application on your physical device by launching it through Eclipse. By doing it this way, the Eclipse ADT Plugin has been signing your application with its own "debug" certificate. Signing with the "debug" certificate is not acceptable for release to the general public. In this short section you will learn how to generate a new keystore, compile your application as a ".apk" file, create a certificate, and sign it. For more details on the Application signing you can view the documentation on the Android Developer Site [\[click here\]](#).

For those of you who haven't taken a Security class or are unfamiliar with the idea of Keys, Certificates and how they work, you can read more information about it on Wikipedia.

The Eclipse ADT plugin provides a simple Export Wizard that automates the entire process for you. Follow the instructions on the Android Developer Site on how to [Compile and sign with Eclipse ADT](#). Make sure to read the short section following this one on [Securing Your Private Key](#). While it's not that necessary for this lab, it will be important when you release an application to the public. Make sure to keep track of your ".apk" file. You will be handing it in at the end of the lab as proof that you completed the Lab.

3) Stage v (verify)

Home Activities:

Activity 1:

Create an android application to print the relevant student name and reg number with table of 2 on TextView. (Printing sequence is like $2 \times 1 = 2$

ali FA13—
 $2 \times 2 = 4$
 ali FA13—)

Create an android application that divides your roll number by 20 and print the table of resultant remainder (only first ten multiples). For example if your roll number is 39, the remainder will be 19 and the output should be

$19 \times 1 = 19$
 $19 \times 2 = 38$
 $19 \times 3 = 57$
 $19 \times 4 = 76$
 $19 \times 5 = 95$
 $19 \times 6 = 114$
 $19 \times 7 = 133$
 $19 \times 8 = 152$
 $19 \times 9 = 171$
 $19 \times 10 = 190$

4) Stage a2 (assess)

Assignment:

To complete this lab you will be required to:

Submit your signed ".apk" file to the Digital Dropbox on Blackboard. This effectively provides time-stamped evidence that you submitted the lab on time should there be any discrepancy later on in the course. The name of your application should be **lab1<cal-poly-username>.apk**. So if your username is **ali**, then your file would be named **lab1ali.apk**.

For this student will submit Lab Assignment before the deadline.

Statement Purpose:

In this lab we will be learning how to use and extend the Android user interface library. In a number of ways it is very similar to the Java Swing library, and in perhaps just as many ways it is different.

Activity Outcomes:

At the end of this lab students will be expected to know:

- What Views, View Groups, Layouts, and Widgets are and how they relate to each other.
- How to declare layouts dynamically at runtime.
- How to reference resources in code and from other resource layout files.
- How to use Events and Event Listeners.

Instructor Note:

ANDROID APPLICATIONS BASICS

A single screen of UI that appears in

your app –the fundamental units of GUI in an Android app


 view: items that appear onscreen in an activity


–widget: GUI control such as a button or text field


–layout: invisible container that manages positions/sizes of widgets

●event: action that occurs when user interacts with widgets –e.g. clicks, typing, scrolling.

Designing a user interface/ Layout:

 Open XML file for your layout (e.g. activity_main.xml)

 Drag widgets from left Palette to the preview image

 Set their properties in lower-right Properties panel

Android-widgets/views:

Views are also referred to as widgets.

Typical examples include standard items such as the Button, CheckBox, ProgressBar and TextView classes.

Views have an integer id associated with them. These ids are assigned in the layout XML files.

Define a Button in the layout file and assign it a unique ID.

```
<Button
    android:id="@+id/my_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/my_button_text"/>
```

Layout

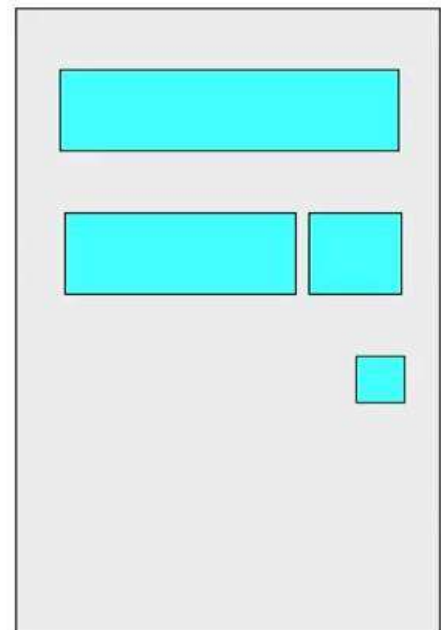
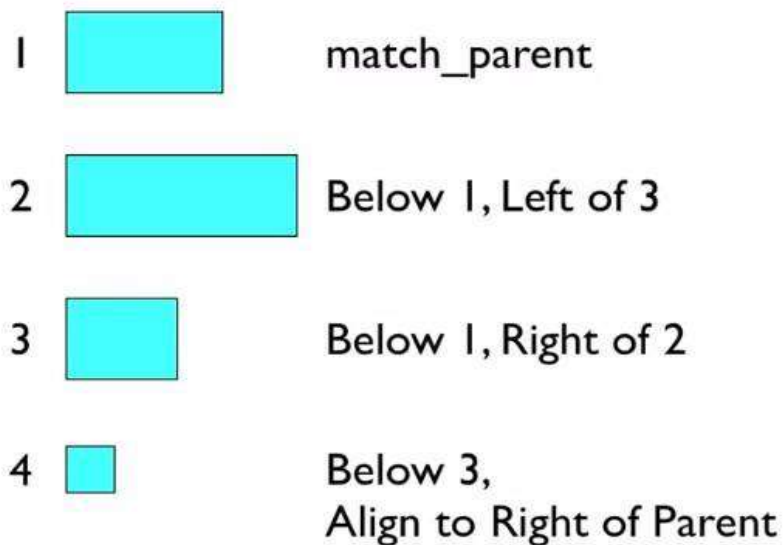
layouts are described in XML and mirrored in Java code

- Android provides several pre-existing layout managers; you can define your own custom layouts if needed
- layouts can be nested to achieve combinations of features

Layout -RelativeLayout

each widget's position and size are relative to other views

- relative to "parent" (the activity itself)
- relative to other widgets/views
- x-position of reference: left, right, center
- y-position of reference: top, bottom, center



Relative anchor points

properties for x/y relative to another widget:

layout_below, above, toLeftOf, toRightOf

set these to the ID of another widget in the format "@id/theID"

properties for x/y relative to layout container (the activity):

–layout_alignParentTop, Bottom, Left, Right

set these flags to a boolean value of "true" to enable them

–layout_centerHorizontal, Vertical, InParent

set these flags to "true" to center the control within its parent in a dimension

Each widget has an associated Java object you can access they are subclasses of parent class View

```
<RelativeLayout...>
```

```
<Button...android:id="@+id/b1"android:text="B1"
```

```
android:layout_alignParentTop="true" android:layout_centerHorizontal="true"/>
```

```
<Button...android:id="@+id/b2"android:text="B2"
```

```
android:layout_alignParentLeft="true" android:layout_below="@+id/b1" />
```

```
<Button...android:id="@+id/b3"android:text="B3"
```

```
android:layout_centerHorizontal="true" android:layout_below="@+id/b2" />
```

```

<Button...android:id="@+id/b4"android:text="B4"
android:layout_alignParentRight="true" android:layout_below="@+id/b2" />
<TextView... android:id="@+id/tv1" android:text="I'm a TextView!"
android:layout_centerInParent="true" />
<Button...android:id="@+id/b5"android:text="B5" android:padding="50dp"
android:layout_centerHorizontal="true" android:layout_alignParentBottom="true"
android:layout_marginBottom="50dp"/>
</RelativeLayout>

```



Accessing View Objects in Java

Each widget has an associated Java object you can access they are subclasses of parent class View

–examples: Button, TextView, EditText, ...

View objects have many get and set methods that correspond to the properties in the Design view:

–background, ID, margin, padding, text, textSize, visibility, ...

–example: for a Button's text property, there will be methods:

```
public String getText()
```

```
public void setText(String text)
```

From the onCreate method of an Activity, the button's object is created by following code:

```
Button myButton= (Button) findViewById(R.id.my_button);
```

Find-View-by-ID

This function is used to retrieve the widgets in the UI that you need to interact with programmatically.

- `findViewById(int id)` is a method of the `View`.
- This method will take a resource Id usually in the form of `R.id.mView` and will return to you a `View` object that is a reference to that `View`.
- The returned object needs to be type casted to the correct type of `View` before you can start interacting with it.

```
TextView answerLabel= (TextView) findViewById(R.id.textView1);
```

```
Button getAnswerButton= (Button) findViewById(R.id.button1);
```

Input Events and Event Handling

Events are a useful way to collect data about user's interaction with interactive components of applications like button press, screen touch etc. The android framework maintains an event queue as first-in, first-out (FIFO) basis. We can capture these events in our program and take appropriate action as per requirements.

Events are actions performed by users. It can be in any form. These events are the inputs to the applications. When application interacts with user it receives input in the form events.

For example an event may be a button click, key press etc.

- ✓ **Event Listeners** –An event listener is an interface in the `View` class that contains a `setOnClickListener()` method. These methods will be called by the Android framework when the `View` to which the listener has been registered is triggered by user interaction with the item in the UI.
- ✓ **Event Listeners Registration** –Event Registration is the process by which an `Event Handler` gets registered with an `Event Listener` so that the handler is called when the `Event Listener` fires the event.
- ✓ **Event Handlers** –When an event happens and we have registered an event listener for the event, the event listener calls the `Event Handlers`, which is the method that actually handles the event. □

Event Registration is the process by which an `Event Handler` gets registered with an `Event Listener` so that the handler is called when the `Event Listener` fires the event. Though there are several tricky ways to register our event listener for any event, but we are going to discuss only 3 ways, out of which we can use based on the requirement or scenario.

- □ Using an Anonymous Inner Class
- □ Activity class implements the Listener interface.
- □ Using Layout file `activity_main.xml` to specify event handler directly. □

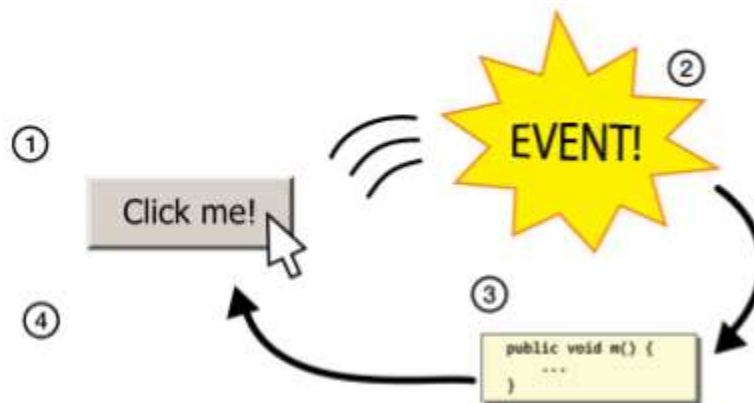
The event handling approach is to capture the events and take appropriate response based on the event type. Event handling is accomplished with the help of event listeners and event handlers.

Event listeners listen to events and event handlers execute the appropriate code as response to the events. When a control is created it must be registered with event listener in order to, notified by

the listeners. When an event is triggered by users, the listeners are notified, and the listeners inform the event handlers that there is an event triggered. Then the event handler is executed, inside the event handler the code is written which is specifying what should happen for the event which was triggered.

'OnClick' Event

- Create 'OnClick' Event on Button Widget in Android Studio



OnClickListener

Interface definition for a callback to be invoked when a view is clicked.

onClick(View v):

Called when a view has been clicked.

setOnClickListener():

Register a callback to be invoked when this view is clicked.

OnClickListener for Button

```
public class MainActivity extends ActionBarActivity implements OnClickListener{
```

```
    Button myButton1 = null;
```

```
    TextView myTextView = null;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        myTextView = (TextView) findViewById(R.id.textView1);
```

```
        myButton1 = (Button) findViewById(R.id.button1);
```

```
        myButton1.setOnClickListener(this);
```

```
    }
```

```
    @Override
```

```
    public void onClick(View v) {
```

```
        if(v.getId() == myButton1.getId()){
```

```
            myTextView.setText("Button 1 Clicked");
```

```
        }
```

```
    }
```

```
Image Button
```


Clickable widget with an image label

to setup an image resource:

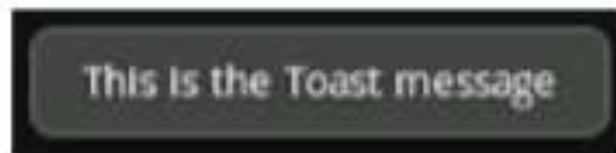
– put image file in project folder app/src/main/res/drawable

– use @drawable/footorefer to foo.png

use simple filenames with only letters and numbers

EditText

An editable text input box



Radio Button



A toggle able on/off switch; part of a group need to be nested inside a Radio Group tag in XML so that only one can be selected at a time

RadioGroup

<LinearLayout...

android:orientation="vertical" android:gravity="center|top">

<RadioGroup...

android:orientation="horizontal">

<RadioButton ...android:id="@+id/lions"

android:text="Lions"

/>

<RadioButton ...android:id="@+id/tigers"

android:text="Tigers" android:checked="true"

/>

<RadioButton ...android:id="@+id/bears"

android:text="Bears,ohmy!"

/>

</RadioGroup>

</LinearLayout>

On Click Listener for Radio Button

```
public class MainActivity extends ActionBarActivity implements OnClickListener{
```

```
    RadioButton myRadioBtn1 = null;
```

```
    RadioButton myRadioBtn2 = null;
```

```
    TextView myTextView = null;
```

```
    @Override
```

```

protected void onCreate(Bundle savedInstanceState) {
    ....
    myTextView= (TextView) findViewById(R.id.textView1);
    myRadioBtn1 = (RadioButton)findViewById(R.id.radioButton1);
    myRadioBtn2 = (RadioButton)findViewById(R.id.radioButton2);
    myRadioBtn1.setOnClickListener(this);
    myRadioBtn2.setOnClickListener(this);
}
@Override
public void onClick(View v) {
    if(v.getId() == myRadioBtn1.getId()){
        myTextView.setText("Radio Button 1 Clicked"); }
    else if(v.getId() == myRadioBtn2.getId()){
        myTextView.setText("Radio Button 2 Clicked"); }
}

```

Spinner

A drop-down menu of select able choices.

also need to handle events in Java code

–must get the Spinner object using find View By Id

–then call its set On Item Selected Listener method

String resources

Declare constant strings and arrays in res/values/strings.xml:

```

<resources>
<string name="name">value</string>
<string name="name">value</string>
<string-array name="arrayname">
<item>value</item>
<item>value</item>
<!--must escape ' as \' in values-->
<item>value</item>
...
<item>value</item>
</string-array>
</resources>

```

Spinner example

Like combo-box in java swing (a dropdown will be displayed having the <items>.)

```

<LinearLayout ...>
<Spinner... android:id="@+id/tmnt" android:entries="@array/turtles"
/>
<TextView ... android:id="@+id/result" />

```

```

</LinearLayout>
inres/values/strings.xml:
<resources>
<string-array name="turtles">
<item>Leonardo</item>
<item>Michelangelo</item>
<item>Donatello</item>
<item>Raphael</item>
</string-array>
</resources>

```

Scroll View

A container with scrollbars around another widget or container

```

<LinearLayout...>
...
<ScrollView
android:layout_width="wrap_content"
android:layout_height="wrap_content">
<TextView ...android:id="@+id/turtle_info"/>
</ScrollView>
</LinearLayout>

```

1) Stage J (Journey)

Introduction

In this lab we will be learning how to use and extend the Android user interface library. In a number of ways it is very similar to the Java Swing library, and in perhaps just as many ways it is different. While being familiar with Swing may help in some situations, it is not necessary. It is important to note that this lab is meant to be done in order, from start to finish. Each activity builds on the previous one, so skipping over earlier activities in the lab may cause you to miss an important lesson that you should be using in later activities.

2) Stage a1 (apply)

Lab Activities:

ACTIVITY: Number game

- New let's build that "Bigger Number" game! :
- user is shown two numbers
- must choose which one is bigger by clicking on the appropriate button
- game pops up brief "correct" / "incorrect" message after each guess
- get points for each correct answer (lose points for incorrect answers)



Displaying Toasts

`Toast.makeText(this, "message", duration).show();`

–where duration is `Toast.LENGTH_SHORT` or `LENGTH_LONG`

- A "Toast" is a pop-up message that appears for a short time.
- Useful for displaying short updates in response to events.

3) Stage v (verify)

Home Activities:

1. Make an app that contains two Buttons (with labels "Push Me" and "Click Me") and a TextView (with text "This is a Test"). Use the XML-based approach, and you can hardcode the label of the buttons and the text of the TextView inside main.xml (i.e., you do not need to use strings.xml at all yet). Use the `android:text` attribute in both cases, but it is easiest to use the visual editor first, then edit main.xml later. Nothing needs to happen when you press the buttons.
2. Test your app on the emulator.
3. If you have an Android phone or tablet, test your app on it.
4. Update your app so that the Button labels and TextView text are taken from strings.xml.
5. Give your buttons some behavior. Here are some options for the button behaviors:
 - Make them pop up Toasts (copy the code from one of the SayHello apps from the lecture, which is in your Eclipse projects)
 - Have them change the foreground color of the Button that was clicked. Choose at random among `Color.RED`, `Color.BLUE`, `Color.YELLOW`, etc. To change the color of the Button, call `setTextColor` on the Button that is passed to the event handler. However, note that although Button has a `setTextColor` method, View (the parent class of Button) does not. So, you have to cast the View to Button before calling `setTextColor`.

- Have them change the text of the Button to “I was clicked n times”. Use two instance variables for the counts. Use either the pure-XML way or the hybrid way of assigning the event handler. Test on the emulator and, if you have one, a real phone or tablet.

6. Make a new app that is similar to the old one, but this time, when you press a button, it should change the color of the TextView instead of the color of the button that was pressed. You can assign the event handler either in XML or in Java, but you’ll need to at least partially use the hybrid approach because you need an explicit Java reference to the TextView that was defined in the XML file.

Statement Purpose:

Activity Outcomes:

After completing this chapter students will be able to understand the following topics.

- Activity
- Life Cycle of an Activity
- Creating an Activity
- Intents
- Types of Intents
- GUI Components
- Fonts and Colors

Instructor Note:

Activity

Activity is the basic building block of any android application. Android applications contain one or more Activities. The introduction of activity has been given in the previous chapter 'Ingredients of an android application'. As we have seen earlier, android activity always has a user interface.

When user launches an application, a window will be displayed. The whole window provides the user interface to the user, this screen makes an activity. . The controls placed in the UI allow the user to do a certain action. The controls in an activity can be created in two different ways. They can be created either by java code or by adding XML code to define the UI. The latter method is always preferred.

Activity stack

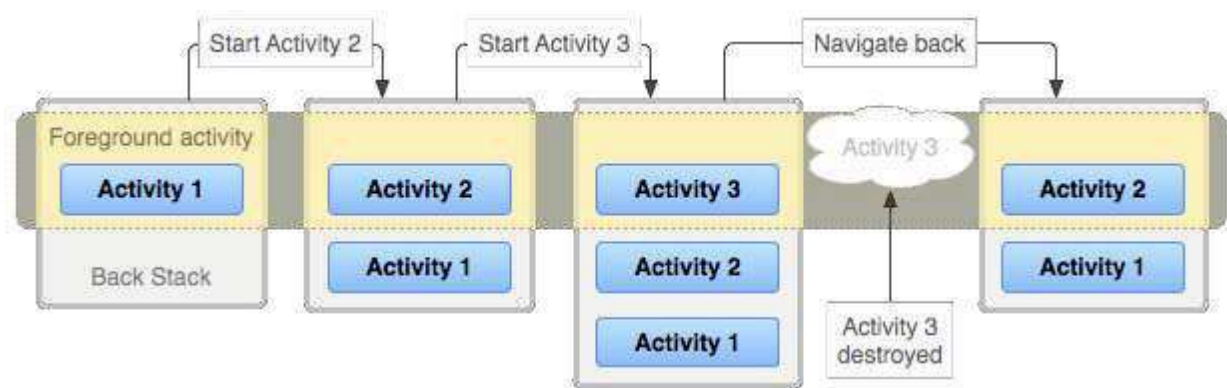
An android application usually contains lots of activities. We can navigate over these activities. When a user starts an activity, android OS pushes that into a stack. If user starts another activity then first activity goes down and newly started activity is added to the top of the stack. When user pushes back button, then top activity is destroyed and the activity which is below the top activity is resumed.

ACTIVITY

For example, a user receives a message on android phone. Now consider the actions performed by the user. By tracking those actions, we can understand how activity stack works. The user views the list of messages in inbox is activity 1. The user opens a message for reading is Activity

2. If the user replies to that message that becomes activity 3. If the user presses back button, then he gets activity 2 again. When he gets activity 2 again, the activity 3 is destroyed. Together these groups of activities form a task.

Task is a collection of activities that users interact with when performing a certain job. The activities are arranged in a stack (the *back stack*), in the order in which each activity is opened.



A representation of how each new activity in a task adds an item to the back stack. When the user presses the *Back* button, the current activity is destroyed and the previous activity resumes.

Life Cycle of an Activity

An android activity has a lifecycle during performing a few things. Why is it required? Let's understand it with an example.

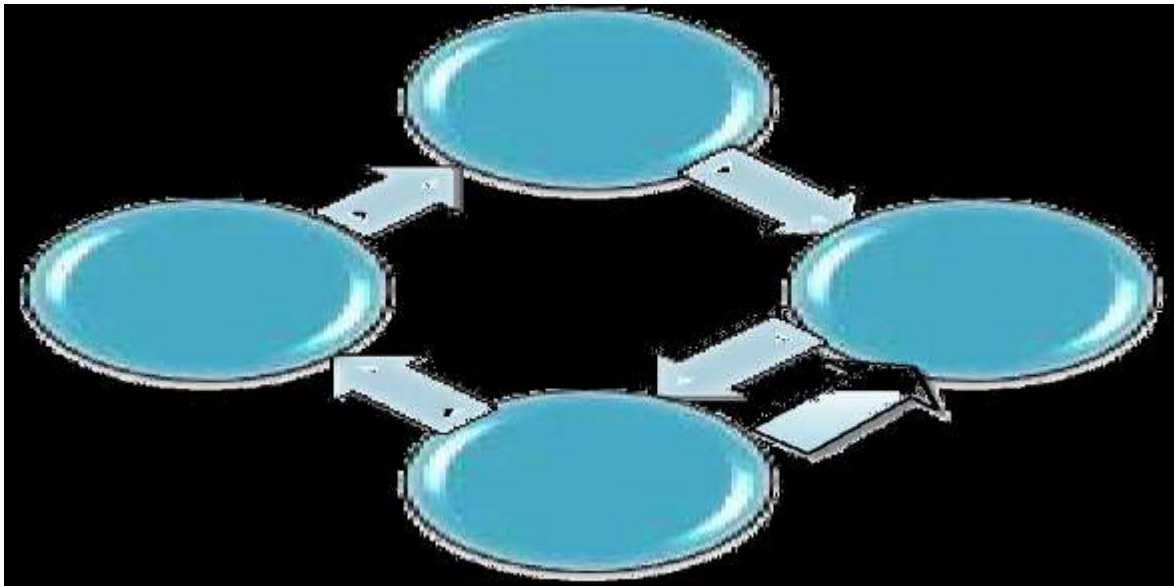
For example, a user is playing game in his mobile. During that time, he receives a call from someone. When playing game, if the user receives a call the game is paused and the calling window appears on top of the game screen. When the call ends, the user can resume the game at the point where he left. Here game is an application. When that application is interfered by any other activity or application, the state of the game application is saved.

How the game state is saved and resumed? There must be some technique to save the game state. To handle such situations android provides activity lifecycle. The life cycle of an activity is handled by a set of functions and they are invoked when something happens to the activity.



An activity life cycle starts from creating an activity. A created activity, in a running application can be in one of the following three states:

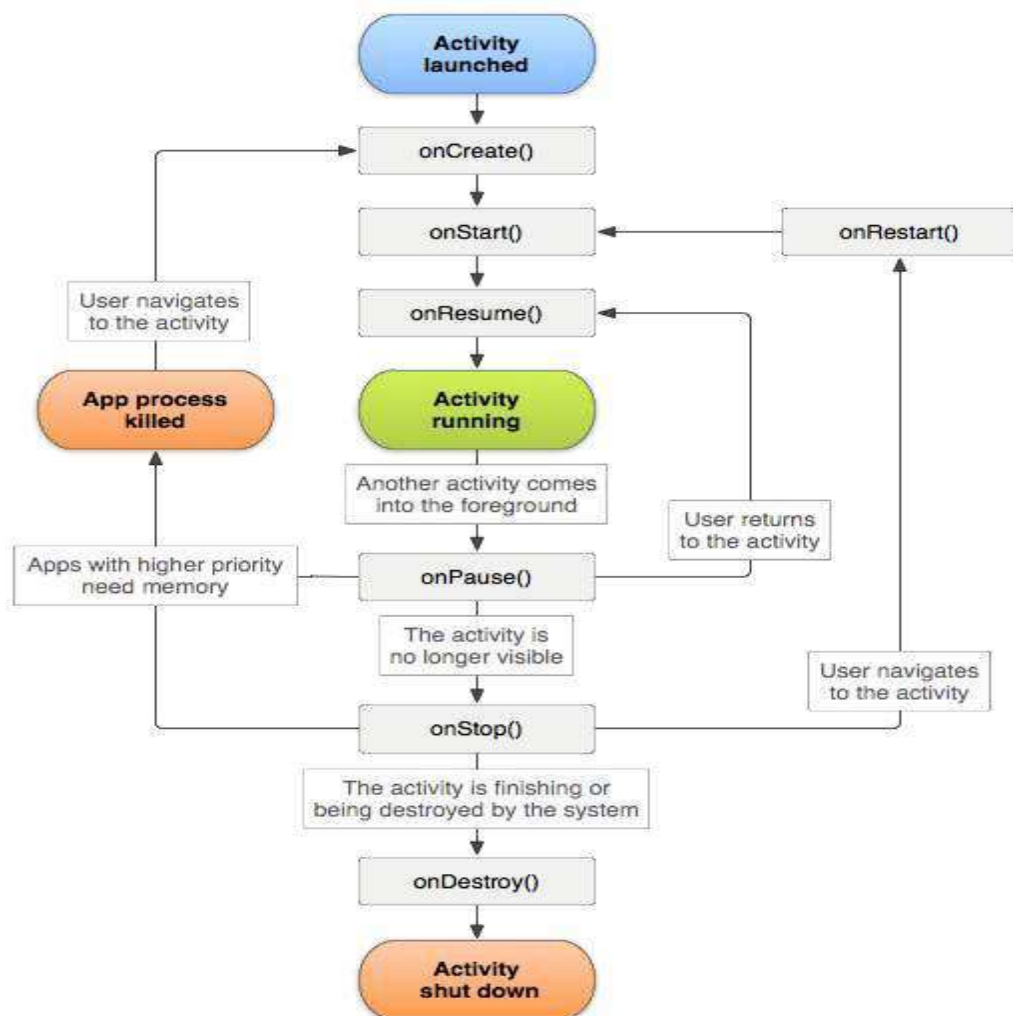
- Resumed – Activity is running or in active state, it is focused and the user can interact with it.
- Paused – Activity is not running or focused, it might be covered by another activity, or is transparent)
- Stopped – Activity is stopped or not



When an activity goes from one state to another, the different life cycle methods are invoked where we can fill in with our code.

Methods for Activity Life Cycle

When an activity transitions from one state to another, set of callback methods are invoked for each state. How a set of call back methods for each state of an activity is invoked in its life cycle is given in the below picture.



Following are the available callback methods

- onCreate()
- onStart()
- onPause()
- onResume()
- onStop()
- onRestart()
- onDestroy()

onCreate The onCreate() method is called when an activity is created for the first time. This method is invoked only once during the entire lifecycle of an activity. To create an activity, we must override the onCreate() method which is available in the class named Activity. Since this method is called only once at the beginning of an activity it can be used for initialization. As the onCreate() method is called while creating an activity, the method for setting the activity layout setContentView() is always invoked inside of this method. **onStart:**

The `onStart()` method is called after the creation of an activity and just before the activity becomes visible to the user. This `onStart()` method can be called from two places - after `onRestart()` and `onCreate()`. i.e., after creating an activity to start it for a first time, or to restart an activity. This `onStart()` method can also be used to reset any data of an activity.

onResume: The `onResume()` method is called when our activity comes into the foreground, from the paused state. We already discussed an example that receiving a call while playing game in a phone. In that example, when the received call is ended the game is resumed.

During that time (when game is resumed), the game activity is on top of the activity stack, so that the activity is ready to interact with user. `onResume()` is a good place to update the screen with new results.

onPause: The `onPause()` method is invoked, when a new activity comes on top of the existing activity. Typically anything that steals the user away from an activity will result in calling `onPause()` method. The `onPause()` method can have the code for releasing resources, saving the application data, stopping background threads etc. It is always guaranteed that whenever the activity is becoming invisible or partially invisible, `onPause()` will be called. But once `onPause()` is called, android reserves the right to kill our activity at any point. Hence we should not be relying on receiving any further events.

onStop:

The `onStop()` method gets called when an activity finishes its work or stopped by the user. We can use this method to shut down when we need to create time intensive or CPU intensive operations.

OnRestart: The `onRestart()` method is similar to `onCreate()`, but `onRestart()` is called only after `onStop()`. Through this method in an application, we can understand that whether the application is starting a fresh or getting restarted. When an activity invokes `onRestart()`, the activity state is stored and variables are reinitialised.

onDestroy: This method is invoked at the last stage of an activity life cycle. When an activity is killed, the `onDestroy()` method is invoked. For example consider that when the user presses back button on any activity, the foreground activity gets destroyed and control will be returned to the previous activity. Even though, we use always `onPause()` and `onStop()` to clean up resources, the `onDestroy()` allows our application to have another chance to do that before it exits. But there is no guarantee that `onDestroy()` will be called. It will be called only when the system is low on resources or user press the back button or if we use `finish()` explicitly in our code. **Process**

Lifecycle The Android system attempts to keep application process around for as long as possible, but eventually will need to remove old processes when memory runs low. As described in Activity Lifecycle, the decision about which process to remove is intimately tied to the state of the user's interaction with it. In general, there are four states a process can be in based on the activities running in it, listed here in order of importance. The system will kill less important processes (the last ones) before it resorts to killing more important processes (the first ones).

- The foreground activity (the activity at the top of the screen that the user is currently

interacting with) is considered the most important. Its process will only be killed as a last resort, if it uses more memory than is available on the device. Generally at this point the device has reached a memory paging state, so this is required in order to keep the user interface responsive.

- A visible activity (an activity that is visible to the user but not in the foreground, such as one sitting behind a foreground dialog) is considered extremely important and will not be killed unless that is required to keep the foreground activity running.
- A background activity (an activity that is not visible to the user and has been paused) is no longer critical, so the system may safely kill its process to reclaim memory for other foreground or visible processes. If its process needs to be killed, when the user navigates back to the activity (making it visible on the screen again), its `onCreate(Bundle)` method will be called with the `savedInstanceState` it had previously supplied in `onSaveInstanceState(Bundle)` so that it can restart itself in the same state as the user last left it.
- An empty process is one hosting no activities or other application components (such as Service or BroadcastReceiver classes). These are killed very quickly by the system as memory becomes low. For this reason, any background operation you do outside of an activity must be executed in the context of an activity BroadcastReceiver or Service to ensure that the system knows it needs to keep your process around.

Sometimes an Activity may need to do a long-running operation that exists independently of the activity lifecycle itself. An example may be a camera application that allows you to upload a picture to a web site. The upload may take a long time, and the application should allow the user to leave the application while it is executing. To accomplish this, your Activity should start a Service in which the upload takes place. This allows the system to properly prioritize your process (considering it to be more important than other non-visible applications) for the duration of the upload, independent of whether the original activity is paused, stopped, or finished.

3. **Creating an Activity** Creating an activity class is as simple as creating a class in java. It simply involves creating a class and overriding a method. To create an activity, we need to extend the Activity class which is available in the `android.app` package. The `onCreate()` method must be overridden in order to create a new activity. The example code gives an understanding of creating an activity class and event handling in android. In an android project,

- Create an xml file under layout folder which is available in `app/src/main/res`
- Create an activity class under `app/src/main/java` folder

Layout XML file By default, an xml file named `activity_main.xml` is available under `res/layout` folder. We can create user interface through two ways either by xml file or by java code. But the previous technique is preferred i.e., through xml file. Using the palette available in

android studio, design the user interface. When controls are created, automatically code is placed in the xml file as shown below.

For each control an id is given which will be used as a reference to that control in code. Here the attribute android:id contains the id for the control EditText. The code for the controls is placed inside opening and closing layout tag. The layout tag looks like as follows.

activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin" tools:context=".MainActivity">
<EditText android:layout_width="wrap_content" android:layout_height="wrap_content"
android:id="@+id/editText" android:layout_marginTop="37dp"
android:layout_centerHorizontal="true" android:height="30dp" android:width="50dp"
android:background="#ff76ffa6" /> <TextView android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content" android:text="Enter the number1"
android:id="@+id/textView2" android:layout_alignTop="@+id/editText"
```

cseitquestions.blogspot.in cseitquestions.blogspot.in

```
android:layout_alignParentLeft="true" android:layout_alignParentStart="true"
android:layout_alignBottom="@+id/editText" /> <TextView
android:layout_width="wrap_content" android:layout_height="wrap_content"
android:text="Enter the number2" android:id="@+id/textView3"
android:layout_below="@+id/textView2" android:layout_alignParentLeft="true"
android:layout_alignParentStart="true" android:layout_marginTop="31dp" /> <EditText
android:layout_width="wrap_content" android:layout_height="wrap_content"
android:id="@+id/editText2" android:layout_alignTop="@+id/textView3"
android:layout_alignLeft="@+id/editText" android:layout_alignStart="@+id/editText"
android:width="50dp" android:height="30dp" android:background="#ff6fffab" /> <Button
android:layout_width="wrap_content" android:layout_height="wrap_content"
android:text="Add" android:id="@+id/button" android:layout_below="@+id/editText2"
android:layout_alignRight="@+id/editText2" android:layout_alignEnd="@+id/editText2"
android:layout_marginTop="45dp" /> <TextView android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceMedium" android:text="Result"
android:id="@+id/textView4" android:layout_below="@+id/button"
android:layout_alignRight="@+id/editText2" android:layout_alignEnd="@+id/editText2"
android:layout_marginTop="85dp" />
```

</RelativeLayout> cseitquestions.blogspot.in cseitquestions.blogspot.in

MainActivity.java public class MainActivity extends Activity implements OnClickListener

```
{ Button button; EditText editText; EditText editText2; TextView textView4; @Override
protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main); button = (Button) findViewById(R.id.button);
editText = (EditText) findViewById(R.id.editText); editText2 = (EditText)
findViewById(R.id.editText2); textView4 = (TextView) findViewById(R.id.textView4);
button.setOnClickListener(this); } public void onClick(View view) { int a,b,c;
a=Integer.parseInt(editText.getText().toString());
b=Integer.parseInt(editText2.getText().toString()); c=a+b; textView4.setText("The result is
"+c); } }
```

Output



Intents

Actually intents are not one of android application components; but used for activating components in Android. It is the part of core message system in android. It defines a message to activate the target component. For example, if we want to invoke a new activity from our current activity, then we need to fire an intent specifying the new activity. Further if we want to start another application from our activity, then also we require intent.

Intent is a messaging object that can be used to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use-cases:

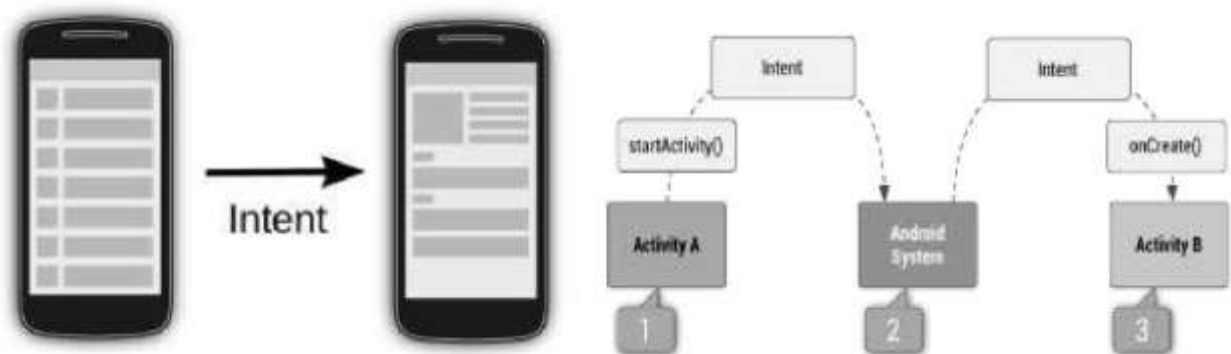
- ✓ Activity
- ✓ Service
- ✓ Broadcast

Starting activities We may want to start another activity from the existing one either as a new activity on top of the activity stack or as a result back from the previous one. For example, if we start an activity which allows the user to pick a person from a list of

contacts, it returns the person that was selected when selection completed i.e end of previous activity.

- `startActivity(Intent)`
- `startActivityForResult(Intent, int)`

The `startActivity(Intent)` method is used to start a new activity, which will be placed at the top of the activity stack. It takes a single argument, an `Intent`, which describes the activity to be executed. Sometimes we may want to get a result back from an activity when it ends.



Types of Intents: Intents have been classified into two types. They are

- Explicit Intents
- Implicit Intents

1) Explicit Intents:

Explicit intent is called for internal communication of an application. It is being invoked by mentioning the target component name. The component or activity can be specified which should be active on receiving the intent. For this reason mostly it is used for intra application communication. The following code describes the way of creating an explicit intent. While creating explicit intent, the target activity is specified so that the android system will invoke the activity.

Now the addition of two numbers program has been modified in order to use intent. In the first activity, inside the `onClick()` method intent object is created. While creating intent object the target component is specified that should be get activated. Here the `MainActivity2` is the target activity that will be invoked by this intent. Since intents are part of the core message system of android, data is passed to the second activity `_MainActivity2'`. i.e. input numbers are read in the first activity, when the button is clicked intent object is created and the result is passed through the intent object. It will activate the target activity. We will get the result in the second activity. The code for second activity is given below.

The second activity is activated by the intent object. Also the second activity receives data or message from the first activity through intent object. The Bundle class is used to save the activity state. So data from the intent object is read with the help of Bundle. The getIntent() method will retrieve data from the intent object. Intent can have the following information.

a) **Component name** - The name of the component to start. This is optional, but it's the critical piece of information that makes an intent explicit, meaning that the intent should be delivered

only to the app component defined by the component name. Without a component name, the intent is implicit and the system decides which component should receive the intent based on the other intent information (such as the action, data, and category).

b) **Action** - A string that specifies the generic action to perform (such as view or pick). Some common actions for starting an activity are:

- **ACTION_VIEW** - This action is used in an intent with startActivity() when it has

some information that an activity can show to the user, such as a photo to view in a gallery app, or an address to view in a map app.

- **ACTION_SEND** - Also known as the "share" intent, this intent is used with startActivity() when some data that the user can share through another app, such as an email app or social sharing app.

c) **Data** - The URI (a Uri object) that references the data to be acted on and/or the MIME type of that data. The type of data supplied is generally dictated by the intent's action. For example, if the action is ACTION_EDIT, the data should contain the URI of the document to edit.

d) **Categories** - A string containing additional information about the kind of component that should handle the intent.

- **Extras** - Key-value pairs that carry additional information required to accomplish the requested action.

- **Flags** - Flags defined in the Intent class that function as metadata for the intent. The flags may instruct the Android system how to launch an activity.

2) Implicit Intents:

When implicit intents are used, a message is sent to the android system to find an appropriate activity to respond to the intent. For example, to share a video, we can use intent. The video can be shared through any type of external application. To do this we can use intent. When intent is received, android system will invoke an activity which is capable

of sending video. If there is more than one activity is capable of receiving the intent, the android system will present a chooser so that the user can select which activity or application should handle it. The following sample code shows the use of implicit intents.

GUI Components Basics of User Interface

App's user interface is everything that the user can see and interact with. Android provides a variety of pre-build UI components such as structured layout objects and UI controls that allow us to build the graphical user interface for our app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus. All user interface elements in an android app are built using View and ViewGroup objects. A View is an object that draws something on the screen that the user can interact with. A ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface. All UI controls have attributes and id. Attributes are used to specify values for the property of a UI control. The id for a UI control is used for referring the controls. **Attributes** Every View and ViewGroup object supports their own variety of XML attributes. Some attributes are specific to a View object (for example, TextView supports the textSize attribute), but these attributes are also inherited by any View objects that may extend this class. Some are common to all View objects, because they are inherited from the root View class (like the id attribute). And, other attributes are considered "layout parameters" which are attributes that describe certain layout orientations of the View object, as defined by that object's parent ViewGroup object. **ID** Any View object may have an integer ID associated with it, to uniquely identify the View within the tree. When the application is compiled, this ID is referenced as an integer, but the ID is typically assigned in the layout XML file as a string, in the id attribute. This is an XML attribute common to all View objects (defined by the View class) and you will use it very often. The syntax for an ID, inside an XML tag is:

```
android:id="@+id/my_button"
```

Input Controls

Input controls are the interactive components in our app's user interface. Android provides a wide variety of controls we can use in our UI, such as buttons, text fields, seek bars, checkboxes, zoom buttons, toggle buttons etc.

Common Controls

Android provides more controls but only few are listed here. To explore other controls browse the android.widget package. If our app requires a specific kind of input control, we can build our own custom components.

1) Stage J (Journey)

Introduction

The commonly used controls have been discussed. There are various controls available under android.widget package. Few controls have been covered here. **EditText** We can add a text box control to our app by adding EditText control called EditTextName in the layout file. This EditText control called EditTextEmail acts as a form field for the user's email address. The following code shows the edit text control added to the layout xml file. The edit text control has a property called android:inputType . It provides advantage for users to set the type of input we want to receive in the text box. The value for the input type property can be any of the valid property values such as text, number, password, email etc.

2) Stage a1 (apply)

Lab Activities:

Running the GUI Components, Fonts, Colors App

GUI Components and colours

The following program illustrates the GUI Components and Colours for Android App

Step 1: Design

1. Open the actual Project folder(app) in Android Studio IDE
2. Click res directory -> layout -> activity_main.xml -> Design
3. Insert the GUI components to Design view in activity_main.xml
4. Enter the id for each component



Step 2: Open res directory -> layout ->

 **activity_main.xml** and add following code

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

```

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="@dimen/
    activity_horizontal_margin" android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context=".MainActivity">
    <TextView android:layout_width="match_parent" android:layout_height="wrap_content"
    android:text="GUI Components" android:id="@+id/t1"
    android:layout_alignParentTop="true" android:layout_alignParentEnd="false"
    android:layout_alignParentStart="false" android:autoText="false"
    android:minHeight="40dp" android:textStyle="bold" android:textSize="30dp"
    android:textIsSelectable="false" android:textAlignment="center"
    android:textColor="@color/accent_material_dark" />
    <Button android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:text="Green" android:id="@+id/b1" android:layout_alignTop="@+id/b2"
    android:layout_alignParentStart="true" />
    <Button android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:text="Red" android:id="@+id/b2" android:layout_centerVertical="true"
    android:layout_centerHorizontal="true" />

```



3) Stage v (verify)

Home Activities:

Activity 1:

In Android, just use “[android.widget.Button](#)” class to display a normal button.

In this tutorial, we show you how to display a normal button, add a click listener, when user click on the button, open an URL in your Android’s internet browser.

P.S This project is developed in Eclipse 3.7, and tested with Android 2.3.3.

Note

For more advance function, like image, please refer to this [ImageButton example](#) and also this [ImageButton selector example](#).

Statement Purpose:

Activity Outcomes:

- ☐ Layouts
- ☐ Event Listeners
- ☐ Menus

Instructor Note:

Layout

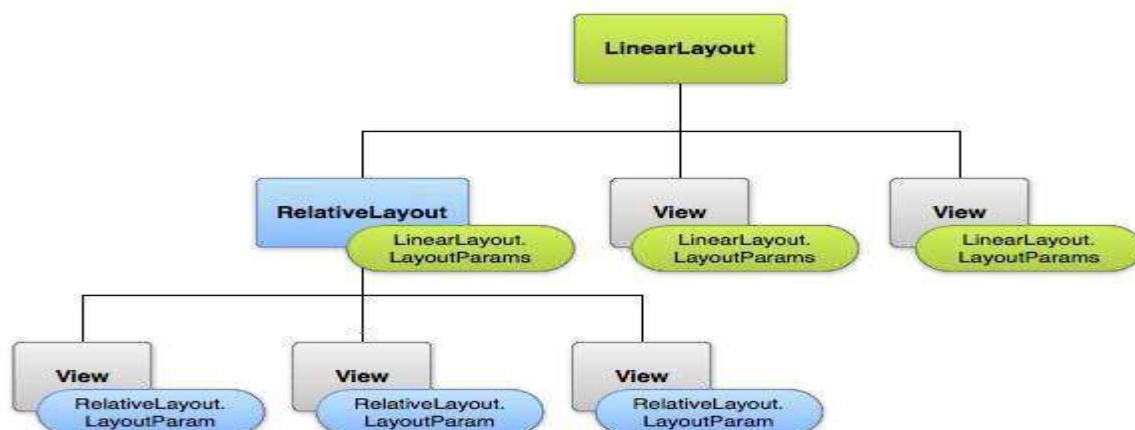
A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. Layout can be declared in two ways:

- Declare UI elements in XML

Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.

- Instantiate layout elements at runtime

Our application can create View and ViewGroup objects (and manipulate their properties) programmatically. XML layout attributes named layout_something define layout parameters for the View that are appropriate for the ViewGroup in which it resides. Every ViewGroup class implements a nested class that extends ViewGroup.LayoutParams. This subclass contains property types that define the size and position for each child view, as appropriate for the view group. As given in the below picture, the parent view group defines layout parameters for each child view (including the child view group).



Android Layout Types There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

ViewGroup or Layout



- There are six types of Layouts:

1. **LinearLayout** (the box model)
2. **RelativeLayout** (a rule-based model)
3. **TableLayout** (the grid model)
4. **Frame Layout** (it provides space in layout)
5. **Absolute Layout** (Non flexible model) – *Deprecated Now*
6. **Grid Layout** (the grid model) – *Introduced in ice cream sandwich*

www.sisoft.in

5

Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and there are other attributes which are specific to that layout. Few attributes of layouts are given below.

Each View or ViewGroup can have its own set of attributes...but, some are very common

Attribute	Description
layout_width	specifies width of View or ViewGroup
layout_height	specifies height
layout_marginTop	extra space on top
layout_marginBottom	extra space on bottom side
layout_marginLeft	extra space on left side
layout_marginRight	extra space on right side
layout_gravity	how child views are positioned
layout_weight	how much extra space in layout should be allocated to View (only when in LinearLayout or TableView)
layout_x	x-coordinate
layout_y	y-coordinate

View Group as layout

View Groups superclass represents containers of widgets/views

–Layout classes are also View Groups

–layouts are described in XML and mirrored in Java code




- Android provides several pre-existing layout managers; you can define your own custom layouts if needed
- layouts can be nested to achieve combinations of features
- widgets can be added to a view group, which will then manage that widget's position/size behavior.



```
1 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
3   android:layout_height="match_parent" android:paddingLeft="16dp"
4   android:paddingRight="16dp"
5   android:paddingTop="16dp"
6   android:paddingBottom="16dp" tools:context=".MainActivity">
7
8 </LinearLayout>
9
```


Menus

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, we should use the Menu APIs. There are three types of application menus:

-  ☐ Options Menu - The primary collection of menu items for an activity, which appears when the user touches the MENU button.
-  ☐ Context Menu - A floating list of menu items that appears when the user touches and holds a view that's registered to provide a context menu.
-  ☐ Submenu - A floating list of menu items that appears when the user touches a menu item that contains a nested menu.

Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs. We will discuss how to create fundamental types of menus or action presentations on different versions of Android.

Options menu The options menu is the primary collection of menu items for an activity. It's where we should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings." If we are developing for Android 2.3 or lower, users can reveal the options menu panel by pressing the Menu button. On Android 3.0 and higher, items from the options menu are presented by the action bar as a combination of on-screen action items and overflow options. Beginning with Android 3.0, the Menu button is deprecated (some devices don't have one), so we should migrate toward using the action bar to provide access to actions and other options.

-  The action bar is a window feature that identifies the user location, and provides user actions and navigation modes. Use of action bar in an app, offers the users a familiar

interface across applications that the system gracefully adapts for different screen configurations.



The action bar provides several key functions:

- a. Provides a dedicated space for giving our app an identity and indicating the user's location in the app.
- b. Makes important actions prominent and accessible in a predictable way (such as *Search*).
- c. Supports consistent navigation and view switching within apps (with tabs or drop-down lists).

Context menu

A context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.

When developing for Android 3.0 and higher, we should instead use the contextual action mode to enable actions on selected content. This mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

Popup menu

A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command. Actions in a popup menu should not directly affect the corresponding content— that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in our activity.

The easiest way to create a menu is to define the menu in XML layout and then inflate a menu resource during the `onOptionsItemSelected()` callback method.

When the user selects a menu item from the Options Menu the system calls the activity's `onOptionsItemSelected()` method. This method passes the `MenuItem` that the user selected. We can identify the menu item by calling `getItemId()`, which returns the unique ID for the menu item (defined by the `android:id` attribute in the menu resource or with an integer given to the `add()` method). We can match this ID against known menu items and perform the appropriate action. A menu group is a collection of menu items that share certain traits. With a group, you can:



Show or hide all items with `setGroupVisible()`



Enable or disable all items with `setGroupEnabled()`



Specify whether all items are checkable with `setGroupCheckable()`



Dynamically Changing menu items at runtime

Once the activity is created, the `onCreateOptionsMenu()` method is called only once, as described above. The system keeps and re-uses the Menu defined in this method until the activity is destroyed. If we want to change the Options Menu any time after it's first created, we must override the `onPrepareOptionsMenu()` method. This passes the Menu object as it currently exists.

This example only deals with Options menu and Submenu only. This sample code helps to understand the Menu life cycle and how to create menus using XML layout and programmatically using code for adding and removing menus dynamically.

1) Stage J (Journey)

Introduction

We create native calculator application in android. This application will perform mathematical operation into the android device. Let us design for UI in activity_main.xml

2) Stage a1 (apply)

Lab Activities:

ACTIVITY: Step 1: Design for UI in layout file (activity_main.xml)



Activity_main.xml

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
android:layout_height="match_parent" android:orientation="vertical"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin" tools:context=".Home"
android:background="#fff" android:weightSum="1">
<LinearLayout
android:orientation="horizontal" android:layout_width="match_parent"
android:layout_height="72dp" android:id="@+id/onoff">
<Switch android:layout_width="match_parent" android:layout_height="match_parent"
android:id="@+id/switch1" android:checked="false" />
```

```

</LinearLayout>
<LinearLayout
    android:orientation="horizontal" android:layout_width="match_parent"
    android:layout_height="55dp" android:layout_gravity="center_horizontal"
    android:id="@+id/l1"> cseitquestions.blogspot.in cseitquestions.blogspot.in
    <EditText android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:id="@+id/display" android:hint="Enter the Values" android:layout_weight="1" />
</LinearLayout>
<LinearLayout
    android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:gravity="center"
    android:paddingTop="20dp" android:id="@+id/l5">
    <Button android:layout_width="55dp" android:layout_height="wrap_content" android:id =
        "@+id/one" android:text="1" />
    <Button android:layout_width="55dp" android:layout_height="wrap_content" android:id =
        "@+id/two" android:text="2" />
    <Button android:layout_width="55dp" android:layout_height="wrap_content" android:id =
        "@+id/three" android:text="3" />
    <Button android:layout_width="55dp" android:layout_height="wrap_content" android:id =
        "@+id/div" android:text="/" />
</LinearLayout> cseitquestions.blogspot.in cseitquestions.blogspot.in
<LinearLayout android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:layout_gravity="center"
    android:gravity="center" android:paddingTop="20dp" android:id="@+id/l2"> <Button
    android:layout_width="55dp" android:layout_height="wrap_content" android:id =
        "@+id/four" android:text="4" /> <Button android:layout_width="55dp"
    android:layout_height="wrap_content" android:id = "@+id/five" android:text="5" />
    <Button android:layout_width="55dp" android:layout_height="wrap_content" android:id =
        "@+id/six" android:text="6" /> <Button android:layout_width="55dp"
    android:layout_height="wrap_content" android:id = "@+id/mul" android:text="*" />
</LinearLayout> <LinearLayout
    android:orientation="horizontal" android:layout_width="fill_parent"
    android:layout_height="wrap_content" android:gravity="center"
    android:paddingTop="20dp" android:id="@+id/l3"> cseitquestions.blogspot.in
    cseitquestions.blogspot.in
    <Button android:layout_width="55dp" android:layout_height="wrap_content" android:id =
        "@+id/seven" android:text="7" /> <Button android:layout_width="55dp"
    android:layout_height="wrap_content" android:id = "@+id/eight" android:text="8" />
    <Button android:layout_width="55dp" android:layout_height="wrap_content" android:id =
        "@+id/nine" android:text="9" /> <Button android:layout_width="55dp"
    android:layout_height="wrap_content" android:id = "@+id/sub" android:text="-" />
</LinearLayout> <LinearLayout android:orientation="horizontal"

```

```

android:layout_width="fill_parent" android:layout_height="wrap_content"
android:gravity="center" android:paddingTop="20dp" android:id="@+id/l4"> <Button
android:layout_width="55dp" android:layout_height="wrap_content" android:id =
"@+id/cancel" android:text="C" /> <Button android:layout_width="55dp"
android:layout_height="wrap_content" android:id = "@+id/zero" android:text="0" />
<Button android:layout_width="55dp"
android:layout_height="wrap_content" android:id = "@+id/equal" android:text="=" />
cseitquestions.blogspot.in cseitquestions.blogspot.in
<Button android:layout_width="55dp" android:layout_height="wrap_content" android:id =
"@+id/add" android:text="+" /> </LinearLayout> </LinearLayout> Step 2: Open
MainActivity.java and add following code package com.example.kamarajios33.calc; import
android.os.Bundle; import android.app.Activity; import
android.support.annotation.RequiresPermission; import android.text.Editable; import
android.view.Menu; import android.view.View; import android.widget.Button; import
android.widget.CompoundButton; import android.widget.EditText; import
android.widget.Switch; public class MainActivity extends Activity implements
View.OnClickListener { Button one, two, three, four, five, six, seven, eight, nine, zero, add,
sub, mul, div, cancel, equal; EditText disp; int op1; int op2; String optr; Switch onoff;
@Override protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState); setContentView(R.layout.activity_main); onoff =
(Switch)findViewById(R.id.switch1); onoff.setChecked(true);
one = (Button) findViewById(R.id.one); two = (Button) findViewById(R.id.two);
cseitquestions.blogspot.in cseitquestions.blogspot.in
three = (Button) findViewById(R.id.three); four = (Button) findViewById(R.id.four); five =
(Button) findViewById(R.id.five); six = (Button) findViewById(R.id.six); seven = (Button)
findViewById(R.id.seven); eight = (Button) findViewById(R.id.eight); nine = (Button)
findViewById(R.id.nine); zero = (Button) findViewById(R.id.zero); add = (Button)
findViewById(R.id.add); sub = (Button) findViewById(R.id.sub); mul = (Button)
findViewById(R.id.mul); div = (Button) findViewById(R.id.div);
cancel = (Button) findViewById(R.id.cancel); equal = (Button) findViewById(R.id.equal); disp
= (EditText) findViewById(R.id.display); onoff.setOnCheckedChangeListener(new
CompoundButton.OnCheckedChangeListener() { @Override
public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
if(isChecked)
{ one.setEnabled(true);
two.setEnabled(true);
three.setEnabled(true);
four.setEnabled(true);
five.setEnabled(true);
six.setEnabled(true);
seven.setEnabled(true);
eight.setEnabled(true);

```

```

nine.setEnabled(true);
zero.setEnabled(true);
add.setEnabled(true);
sub.setEnabled(true);
mul.setEnabled(true);
sub.setEnabled(true); cseitquestions.blogspot.in cseitquestions.blogspot.in
mul.setEnabled(true); div.setEnabled(true); cancel.setEnabled(true); equal.setEnabled(true);
disp.setEnabled(true); } else { one.setEnabled(false); two.setEnabled(false);
three.setEnabled(false); four.setEnabled(false); five.setEnabled(false); six.setEnabled(false);
seven.setEnabled(false); eight.setEnabled(false); nine.setEnabled(false);
zero.setEnabled(false); add.setEnabled(false); sub.setEnabled(false); mul.setEnabled(false);
sub.setEnabled(false); mul.setEnabled(false); div.setEnabled(false); cancel.setEnabled(false);
equal.setEnabled(false); disp.setEnabled(false); } } }; try { one.setOnClickListener(this);
two.setOnClickListener(this); three.setOnClickListener(this); four.setOnClickListener(this);
five.setOnClickListener(this); cseitquestions.blogspot.in cseitquestions.blogspot.in
six.setOnClickListener(this);
seven.setOnClickListener(this);
eight.setOnClickListener(this);
nine.setOnClickListener(this);
zero.setOnClickListener(this);
cancel.setOnClickListener(this);
add.setOnClickListener(this);
sub.setOnClickListener(this);
mul.setOnClickListener(this);
div.setOnClickListener(this);
equal.setOnClickListener(this); } catch (Exception e) {
} }
public void operation() { if (optr.equals("+")) {
op2 = Integer.parseInt(disp.getText().toString()); disp.setText("");
op1 = op1 + op2; disp.setText(Integer.toString(op1));
} else if (optr.equals("-")) {
op2 = Integer.parseInt(disp.getText().toString()); disp.setText("");
op1 = op1 - op2; disp.setText(Integer.toString(op1));
} else if (optr.equals("*")) {
op2 = Integer.parseInt(disp.getText().toString()); disp.setText("");
op1 = op1 * op2; disp.setText(Integer.toString(op1));
} else if (optr.equals("/")) {
op2 = Integer.parseInt(disp.getText().toString()); disp.setText("");
op1 = op1 / op2; disp.setText(Integer.toString(op1));
} } cseitquestions.blogspot.in cseitquestions.blogspot.in
@Override
public void onClick(View arg0)

```

```

{
Editable str = disp.getText(); switch (arg0.getId()) {
case R.id.one:
if (op2 != 0) { op2 = 0; disp.setText("");
}
str = str.append(two.getText()); disp.setText(str);
break;
case R.id.two: if (op2 != 0) {
op2 = 0; disp.setText("");
}
str = str.append(two.getText()); disp.setText(str);
break;
case R.id.three: if (op2 != 0) {
op2 = 0; disp.setText("");
}
str = str.append(three.getText()); disp.setText(str);
break;
case R.id.four: if (op2 != 0) {
op2 = 0; disp.setText("");
}
str = str.append(four.getText()); disp.setText(str);
break; cseitquestions.blogspot.in cseitquestions.blogspot.in
case R.id.five: if (op2 != 0) {
op2 = 0; disp.setText("");
}
str = str.append(five.getText()); disp.setText(str);
break; case R.id.six:
if (op2 != 0) { op2 = 0; disp.setText("");
}
str = str.append(six.getText()); disp.setText(str);
break;
case R.id.seven: if (op2 != 0) {
op2 = 0; disp.setText("");
}
str = str.append(seven.getText()); disp.setText(str);
break;
case R.id.eight: if (op2 != 0) {
op2 = 0; disp.setText("");
}
str = str.append(eight.getText()); disp.setText(str);
break; cseitquestions.blogspot.in cseitquestions.blogspot.in

```

```

case R.id.nine: if (op2 != 0) { op2 = 0; disp.setText(""); } str = str.append(nine.getText());
disp.setText(str); break; case R.id.zero: if (op2 != 0) { op2 = 0; disp.setText(""); } str =
str.append(zero.getText()); disp.setText(str); case R.id.cancel: op1 = 0; op2 = 0;
disp.setText(""); disp.setHint("Perform Operation"); break; case R.id.add: optr = "+"; if (op1
== 0) { op1 = Integer.parseInt(disp.getText().toString()); disp.setText(""); } else if (op2 != 0) {
op2 = 0; disp.setText(""); } else { op2 = Integer.parseInt(disp.getText().toString());
disp.setText(""); op1 = op1 + op2; disp.setText(Integer.toString(op1)); }
break; cseitquestions.blogspot.in cseitquestions.blogspot.in
case R.id.sub: optr = "-"; if (op1 == 0) { op1 = Integer.parseInt(disp.getText().toString());
disp.setText(""); } else if (op2 != 0) { op2 = 0; disp.setText(""); } else { op2 =
Integer.parseInt(disp.getText().toString()); disp.setText(""); op1 = op1 - op2;
disp.setText(Integer.toString(op1)); } break; case R.id.mul: optr = "*"; if (op1 == 0) { op1 =
Integer.parseInt(disp.getText().toString()); disp.setText(""); } else if (op2 != 0) { op2 = 0;
disp.setText(""); } else { op2 = Integer.parseInt(disp.getText().toString()); disp.setText("");
op1 = op1 * op2; disp.setText(Integer.toString(op1)); }
break; cseitquestions.blogspot.in cseitquestions.blogspot.in
case R.id.div: optr = "/";
if (op1 == 0)
{
op1 = Integer.parseInt(disp.getText().toString()); disp.setText("");
}
else if (op2 != 0) { op2 = 0; disp.setText("");
}
else
{
op2 = Integer.parseInt(disp.getText().toString()); disp.setText("");
op1 = op1 / op2; disp.setText(Integer.toString(op1));
}
break;
case R.id.equal:
if (!optr.equals(null))
{
if (op2 != 0)
{
if (optr.equals("+"))
{
disp.setText(""); op1 = op1 + op2;
disp.setText(Integer.toString(op1));
}
else if (optr.equals("-"))
{
disp.setText(""); op1 = op1 - op2;

```

```

disp.setText(Integer.toString(op1));
} cseitquestions.blogspot.in cseitquestions.blogspot.in
else if (optr.equals("*"))
{
disp.setText(""); op1 = op1 * op2;
disp.setText(Integer.toString(op1));
}
else if (optr.equals("/"))
{
disp.setText(""); op1 = op1 / op2;
disp.setText(Integer.toString(op1));
}
}
else
{
operation();
}
}break; }
}}

```

ACTIVITY: Running the Layout Managers and Event Click Listener App

Step 1: Create an android project using android studio

Step 2: After creating the project, open the res directory -> layout -> resource file named activity_main.xml

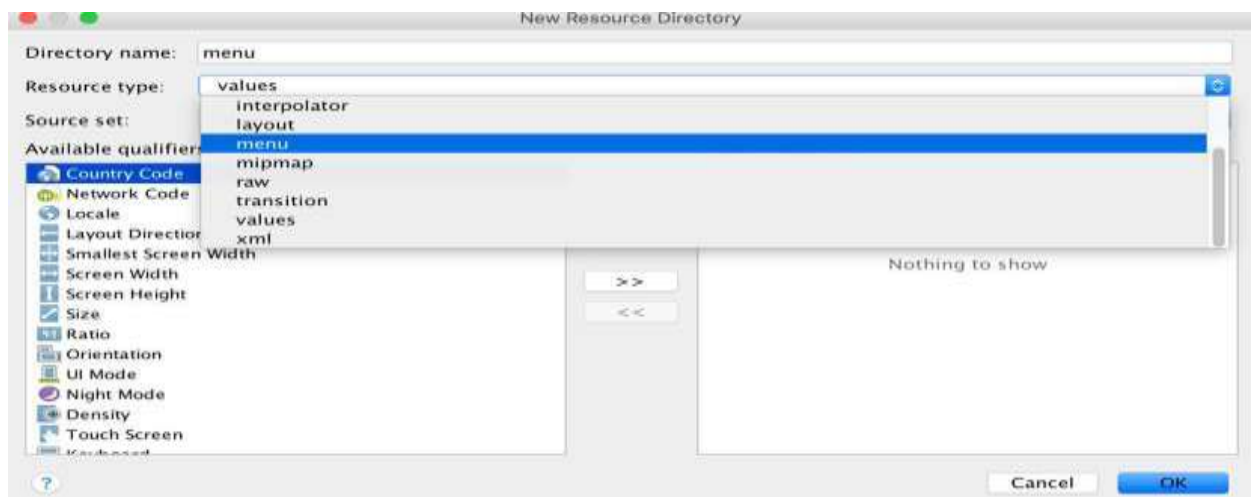
Step 3: Create two resource (*.xml) file named activity_second.xml, activity_third.xml & two activity (*.java) file named second.java and third.java file.

Right click res directory -> New -> Activity -> Empty Activity

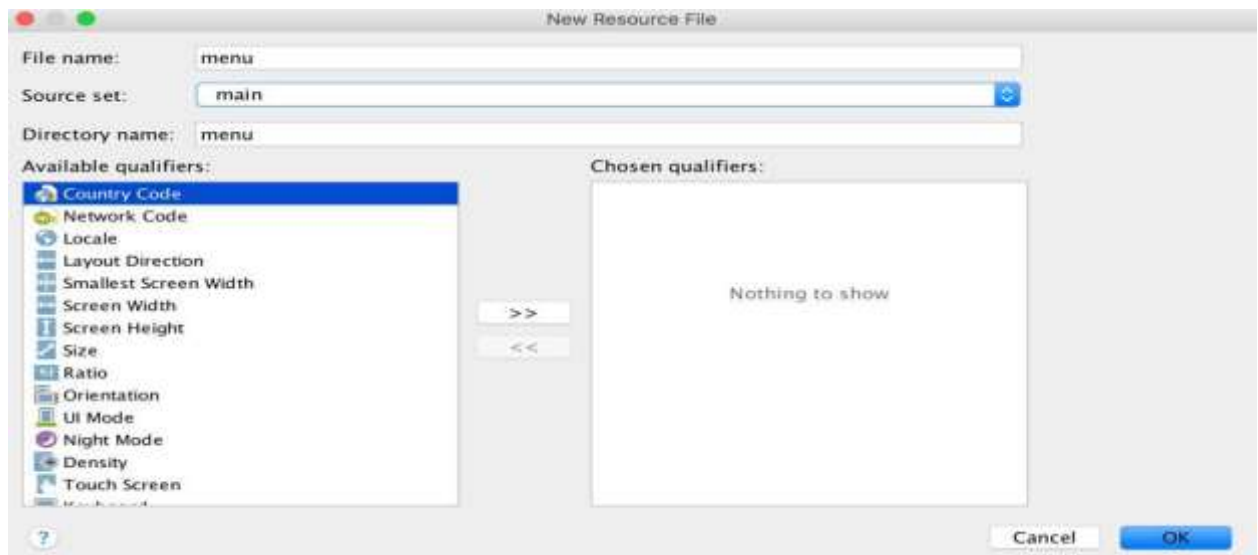


Step 5: Create new Resource directory named menu and new resource file named menu

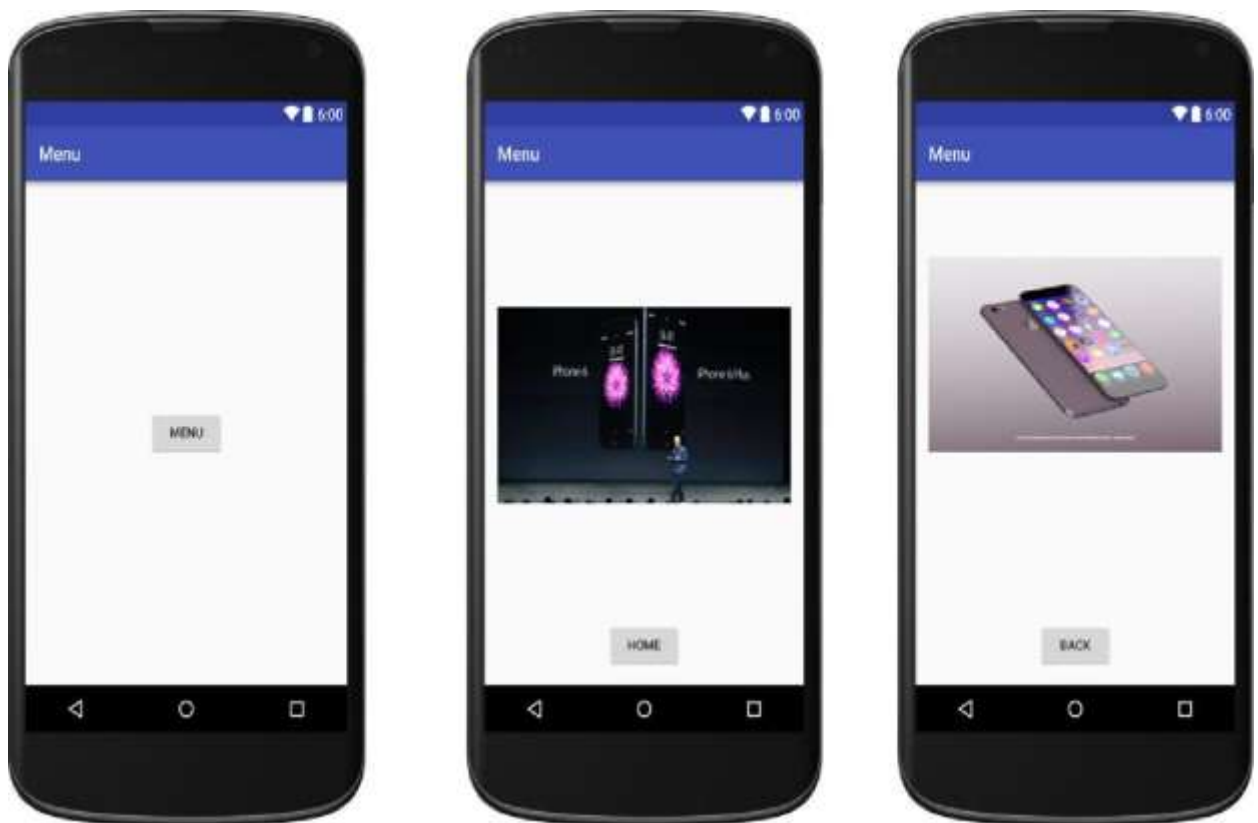
Right click res directory -> New -> Android Resource Directory -> resource type -> select menu -> finish.



Right click menu directory -> New -> new menu resource file -> enter file name -> Ok



Step 4: Design (After the design, the xml code will be generated automatically in the layout file)



Open menu.xml and add the following code

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
tools:context="com.example.kamarajios33.menu.MainActivity"><item
android:id="@+id/one"
android:title="One"/>
<item
```

```

android:id="@+id/two"
android:title="Two"/>
</menu>

```

Step 6: Open **MainActivity.java**, **second.java** & **third.java** and add the following code

MainActivity.java

```

public class MainActivity extends AppCompatActivity { Button b1;
@Override
protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main); b1=(Button)findViewById(R.id.b1);
b1.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
PopupMenu pm = new PopupMenu(MainActivity.this,b1);
pm.getMenuInflater().inflate(R.menu.menu,pm.getMenu());
pm.setOnMenuItemClickListener(new PopupMenu.OnMenuItemClickListener() {
@Override
public boolean onMenuItemClick(MenuItem item)
{ switch (item.getItemId()) {
case R.id.one:
Intent o = new Intent(getApplicationContext(), second.class); startActivity(o);
System.exit(0);
case R.id.two: Intent in = new Intent(getApplicationContext(),third.class); startActivity(in);
System.exit(0); break; } return false; } }); pm.show(); } }); } }

```

The output of the above code is as follows.



3) Stage v (verify)

Home Activities:

1. Write XML code for generating the following layout.

TEXT	
TEXT	EDIT TEXT
TEXT	EDIT TEXT
BUTTON	

Statement Purpose:

Student will learn about Graphical Primitives.

Activity Outcomes:

After completing this chapter students will be able to develop and complete hands on android designing.

- Introduction to SQLite
- Android Database API
- Using Content Provider
- Connecting with Database Server

Instructor Note:

1) Stage J (Journey)

Introduction

Students must read and have best practice of past four labs.

Introduction to SQLite

Android SQLite database is an integral part —built-in component. SQLite is an Open Source database. SQLite supports standard relational database features like SQL syntax, transactions and prepared statements. It is a self-contained, transactional database engine that requires no separate server process. To use this database the knowledge of SQL alone is sufficient. We need not to learn any new query language.

SQLite supports the data types such as TEXT (similar to String in Java), INTEGER (similar to long in Java) and REAL (similar to double in Java). All other types must be converted into one of these fields before getting saved in the database. SQLite does not validate, whether the data stored in the columns are actually of the defined type, e.g. we can write an integer into a string column and vice versa.

SQLite is embedded into every android device. Since the database requires limited memory at runtime (approx. 250 KByte) it is widely chosen for embedding in devices. Android platform includes the SQLite embedded database and provides APIs to access the database. To use the SQLite database in android, we do not require a setup procedure.

We need to only have to define the SQL statements for creating and updating the database. Then the database will be managed by the android system. Access to the SQLite database

involves accessing the file system. This can be slow. Therefore it is recommended to perform database operations asynchronously. If our application creates a database, this database is stored by default in the following directory
DATA/data/APP_NAME/databases/FILENAME.

The parts of the above directory have been explained below..

- DATA is the path which the Environment.getDataDirectory() method returns.
- APP_NAME is our application name.
- FILENAME is the name we specify in our code for the database.

Android Database API

The android.database package contains all necessary classes for working with databases. The android.database.sqlite package contains the SQLite specific classes. The following classes will be helpful while developing app that uses database.

- SQLiteDatabase
- Cursor
- SQLiteOpenHelper

1. SQLiteDatabase


SQLiteDatabase is the base class for working with a SQLite database in android and provides methods to open, query, update and close the database. In addition it provides the execSQL() method, which allows to execute an SQL statement directly.

To perform the following database operations, SQLiteDatabase class provides various methods.

a) Insert operation – For inserting values to the database, we need to save data into ContentValues. ContentValues allow defining key/value pairs. The key represents the table column name and the value represents the content corresponding to that column.

The object ContentValues allows to define key/values. The key represents the table column identifier and the value represents the content for the table record in this column.

ContentValues can be used for inserts and updates of database entries.

 **Update Operation:** To update the existing record in your Android sqlite database table, you need to execute the following method.

```
db.update(TABLE_NAME,content_values,WHERE_CLAUSE,WHERE_ARGUMENTS);
```

1.1 The first argument takes the name of the table.

1.2 The second argument takes a new record. Here, we pass the contentvalue object

which contains data corresponding to each column in a key/value pair. So, all the columns which need to get updated are provided in the ContentValues object as a key/ value pair.

- The third argument specifies the where clause (it's a condition). For example, update the employee name where employee ID is —123. Here, the where clause is the —id=1.
- The fourth argument takes the value corresponding to the where clause. For the above example it is —123.

a) Delete Operation: Similarly, you can perform the delete operation on an Android SQLite database table. It takes three arguments. The first is the table name, the second is the where clause, and the third is the value corresponding to that where clause.

```
db.delete(TABLE_NAME,WHERE_CLAUSE,WHERE_ARGUMENTS);
```

b) Read Operation: To read values from an Android SQLite database table, we need to learn about cursors. We execute the select operation on the database and we get multiple rows as a result. We assign those rows to a cursor. A Cursor points to one row at a time. This is how we get the required data.

Raw Query: This query directly accepts SQL as an input. You can pass SQL statement(s) and SQLiteDatabase will execute that query for you.

```
db.rawQuery(sql_statement,null);
```

Cursor

A query returns a Cursor object. A Cursor represents the result of a query and basically points to one row of the query result. This way Android can buffer the query results efficiently; as it does not have to load all data into memory.

To get the number of elements of the resulting query use the getCount() method. To move between individual data rows, you can use the moveToFirst() and moveToNext() methods. The isAfterLast() method allows to check if the end of the query result has been reached.

Cursor provides typed get*() methods, e.g. getLong(columnIndex), getString(columnIndex) to access the column data for the current position of the result. The "columnIndex" is the number of the column you are accessing. Cursor also provides the getColumnIndexOrThrow(String) method which allows to get the column index for a column name of the table. A Cursor needs to be closed with the close() method call.

SQLiteOpenHelper

To create and upgrade a database in our Android application we can use the SQLiteOpenHelper class. • onCreate()
- is called by the framework, if the database is accessed but not yet created.

- onUpgrade() - called, if the database version is increased in your application code. This method allows you to update an existing database schema or to drop the existing database and recreate it via the onCreate() method.

- Both methods receive an SQLiteDatabase object as parameter which is the Java representation of the database.

- The SQLiteOpenHelper class provides the getReadableDatabase() and getWritableDatabase() methods to get access to an SQLiteDatabase object; either in read or write mode. The database tables should use the identifier _id for the primary key of the table. Several Android functions rely on this standard. Enhancing the display of Records After retrieving the records, we need to display them to user. While displaying records, we need to choose the UI control. The following UI controls will be helpful for displaying more number of records from database to the user.

- ListView are Views which allow to display a list of elements.

- ListActivities are specialized activities which make the usage of ListView easier.

- The SimpleCursorAdapter class will map the columns to the Views based on the Cursor passed to it. The SimpleCursorAdapter allows to set a layout for each row of the ListView.

You also define an array which contains the column names and another array which contains the IDs of Views which should be filled with the data Using Content Provider

The security model of android does not allow an application to access data from another application. Still we may want to share data from our app to another app or our app might need data which can be accessed from another app. Content provider is the best way to share data across applications. Content provider is a set of data wrapped up in a custom API to read and write. Applications/Processes have to register themselves as a provider of data. Other applications can request android system to read/write that data through a predefined API. Content provider API adheres to CRUD principle.

Content Provider Workflow :

Examples for content provider are Contactsthat exposes user information to other applications, Media store which allows other applications to access and store media files etc.

Content providers are simple interfaces which uses standard insert(), query(), update(), delete() methods to access app data. A special URI starting with content:// will be assigned to each content providers and that will be recognized across applications or apps.

Above diagram shows how content provider works. App 1 stores its data in its own database and provides a content provider. App 2 communicates access App 1's data through the content provider.

Writing a Content Provider

To the following are the steps for creating a content provider.

- 1) Create sub class for ContentProvider.
- 2) Define content URI
- 3) Implement or override the required methods. insert(), update(), query(), delete(), getType().
- 4) Declare the content provider in AndroidManifest.xml

The steps are explained below with description.

1) **Create a subclass for ContentProvider class.**

Create a class and extend the ContentProvider class available in the android.content package.

2) **Defining URI:**

Content URI has a special path similar to HTTP. Content provider URI consists of four parts.

content://authority/path/id

- content:// All the content provider URIs should start with this value
- 'authority' is Java namespace of the content provider implementation. (fully qualified Java package name)
- 'path' is the virtual directory within the provider that identifies the kind of data being requested.
- 'id' is optional part that specifies the primary key of a record being requested. We can omit this part to request all records.

Overriding the required methods

a) Adding new records:

We need to override insert() method of the ContentProvider to insert new record to the database via content provider. The caller method will have to specify the content provider URI and the values to be inserted without the ID. Successful insert operation will return the URI value with the newly inserted ID.

b) Updating records

To update one or more records via content provider we need to specify the content provider URI. update() method of the ContentProvider is used to update records. We have to specify the ID of the record to be updated. To update multiple records, we have to specify 'selection' parameter to indicate which rows are to be updated. This method will return the number of rows updated.

c) Deleting records

Deleting one or more records is similar to update operation. We need to specify either ID or 'selection' to delete records. The delete() method of the ContentProvider will return the number of records deleted.

d) Querying the content provider

To query the data via content provider we override query() method of ContentProvider. This method has many parameters. We can specify list of columns that will be placed in the result cursor using 'projection' parameter. We can specify 'selection' criteria, 'sortOrder' etc. If we do not specify '_projection' parameter, all the columns will be included in the result cursor. If we do not specify sortOrder the provider will choose its own sort order.

e) getType() method

be the fully qualified name of the content provider class.

WebView

Android WebView component used in Android web app to render the android apps GUI. Webview component is a browser supported view and it allows us implementing web pages in our android application as we like. WebView gives most of the screen space in our android application.

□ Android Web App or Android Hybrid App

Android web app is actually a mix of web app and native android app. The app is developed using support of Android components and web technologies inside a WebView. It contain web technologies like HTML, CSS, JavaScript, SVG, Json, Ruby, Rails etc., Another common term for an Android web app is Android hybrid app. □

□ WebView is Based on Chrome □

The WebView component is fully based on same code as Chrome for android. It indicates that web app which supported in Chrome browser that can be converted into Android App. From Android 4.4 Chrome replacing the old android browser as default/ builtin browser.

□ WebView Needs Internet Permission

In the Android application Internet permission is mandatory if app needs to load a webpage. While installing an app the user should accepts the internet access permission. To get the internet permission in the android app, internet permission element is to add in manifest file.

2) Stage a1 (apply)

Lab Activities:

Activity 1:

Step 1: Create an android project using android studio.

Step 2: After creating the project, open the java file named MainActivity.xml. **MainActivity.java**

```
package com.example.kamarajios33.graphics; import android.content.Intent; import
android.support.v7.app.AppCompatActivity; import android.os.Bundle; import
android.content.Context; import android.graphics.Canvas; import android.graphics.Color; import
android.graphics.Paint; import android.graphics.RectF; import android.view.View; public class
MainActivity extends AppCompatActivity { DemoView dv; @Override protected void
onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main); dv = new DemoView(this); setContentView(dv); } private
class DemoView extends View { public DemoView(Context context) { super(context); }
@Override protected void onDraw(Canvas canvas) { super.onDraw(canvas); Paint ob=new Paint();
```

```
ob.setStyle(Paint.Style.FILL);
```

```
ob.setColor(Color.WHITE);
```

```
canvas.drawPaint(ob);
```

```
ob.setColor(Color.GRAY); canvas.drawCircle(100, 100, 60, ob); ob.setColor(Color.CYAN);
canvas.drawCircle(200, 50, 30, ob); ob.setColor(Color.MAGENTA); canvas.drawRect(200,
200, 400, 400, ob); ob.setColor(Color.RED); canvas.drawLine(250,50,350,400,ob);
canvas.rotate(-45); }}
```

Step 3: The output of the above code is as follows.



Activity 2:

Running the SQLite App

Step 1: Create an android project using android studio

Step 2: Create two resource files (*.xml) and two activity files (*.java)
named activity_main.xml & activity_main2.xml and MainActivity.java & Main2Activity.java.

Step 3: Open res directory -> layout -> activity_main.xml -> Click -> Design button at bottom of the Android Studio. Put the necessary components for both resource files (activity_main.xml, activity_main2.xml)

Step 4: Design (After the design part, the xml code will be generated automatically in the layout file)

This method is used to handle requests for the MIME type of data at the given URI. We use either `vnd.android.cursor.item` or `vnd.android.cursor.dir/`. The `vnd.android.cursor.item` is used to represent specific item. Another one is used to specify all items.

Registering the provider in AndroidManifest.xml

Content providers need to be registered in the AndroidManifest.xml. To register the content providers in manifest file, the `<provider>` element is used. The `<application>` element is the parent tag of the `<provider>` element.



Create an activity file

MainActivity.java

```
package com.example.kamarajios33.database;
import android.app.AlertDialog; import android.content.Intent; import
android.database.Cursor;
import android.database.sqlite.SQLiteDatabase; import
android.support.v7.app.AppCompatActivity;
import android.os.Bundle; import android.view.View; import android.widget.EditText;
import android.widget.TableRow; import android.widget.Toast;
public class MainActivity extends AppCompatActivity { SQLiteDatabase db;
String f,l,e; @Override
protected void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState); setContentView(R.layout.activity_main);
db = openOrCreateDatabase("Mydb",MODE_PRIVATE,null);
db.execSQL("CREATE TABLE IF NOT EXISTS student(fname VARCHAR,lname VARCHAR,email
VARCHAR);");
}
public void Adddata(View view)
{
EditText e1 = (EditText)findViewById(R.id.e1);
EditText e2 = (EditText)findViewById(R.id.e2);
EditText e3 = (EditText)findViewById(R.id.e3); f = e1.getText().toString();
l=e2.getText().toString(); e=e3.getText().toString();
db.execSQL("INSERT INTO student VALUES('"+f+"','"+l+"','"+e+"');");
Toast.makeText(getApplicationContext(),"Row
Inserted",Toast.LENGTH_SHORT).show();
public void close(View view)
{
System.exit(0);
}
public void showdata(View view)
{
//Toast.makeText(getApplicationContext(),"Show Data",Toast.LENGTH_LONG).show();
Intent i = new Intent(getApplicationContext(),Main2Activity.class);
startActivity(i);
}
}
```

Main2Activity.java

```
package com.example.kamarajios33.database;
import android.content.Intent; import android.database.Cursor;
```

```

import android.database.sqlite.SQLiteDatabase; import
android.support.v7.app.AppCompatActivity; import android.os.Bundle;
import android.view.View; import android.widget.Button; import android.widget.EditText;
import android.widget.Toast;
public class Main2Activity extends AppCompatActivity { EditText fn,ln,em;
Button next,back,prev; SQLiteDatabase db; Integer j =0; @Override
protected void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
cseitquestions.blogspot.in cseitquestions.blogspot.in
setContentView(R.layout.activity_main2); fn=(EditText)findViewById(R.id.fn);
ln=(EditText)findViewById(R.id.ln); em=(EditText)findViewById(R.id.em); back
=(Button)findViewById(R.id.button2); next=(Button)findViewById(R.id.button);
prev=(Button)findViewById(R.id.b3);
db = openOrCreateDatabase("Mydb",MODE_PRIVATE,null);
final Cursor c = db.rawQuery("select * from student",null); c.moveToFirst();
fn.setText(c.getString(c.getColumnIndex("fname")));
ln.setText(c.getString(c.getColumnIndex("lname")));
em.setText(c.getString(c.getColumnIndex("email")));
back.setOnClickListener(new View.OnClickListener() { @Override
public void onClick(View v) {
Intent il = new Intent(getApplicationContext(), MainActivity.class); startActivity(il);
System.exit(0);
} });
next.setOnClickListener(new View.OnClickListener() { @Override
public void onClick(View v) { try
{ c.moveToNext();
fn.setText(c.getString(c.getColumnIndex("fname")));
ln.setText(c.getString(c.getColumnIndex("lname")));
em.setText(c.getString(c.getColumnIndex("email")));
} catch (Exception e) {
Toast.makeText(getApplicationContext(),"Last Record",Toast.LENGTH_LONG).show();
e.printStackTrace();
} });
prev.setOnClickListener(new View.OnClickListener() { @Override
public void onClick(View v) { try
{ c.moveToPrevious();
fn.setText(c.getString(c.getColumnIndex("fname")));
ln.setText(c.getString(c.getColumnIndex("lname")));
em.setText(c.getString(c.getColumnIndex("email")));
}
catch (Exception e)
{ Toast.makeText(getApplicationContext(),"First Record",Toast.LENGTH_LONG).show();
e.printStackTrace();

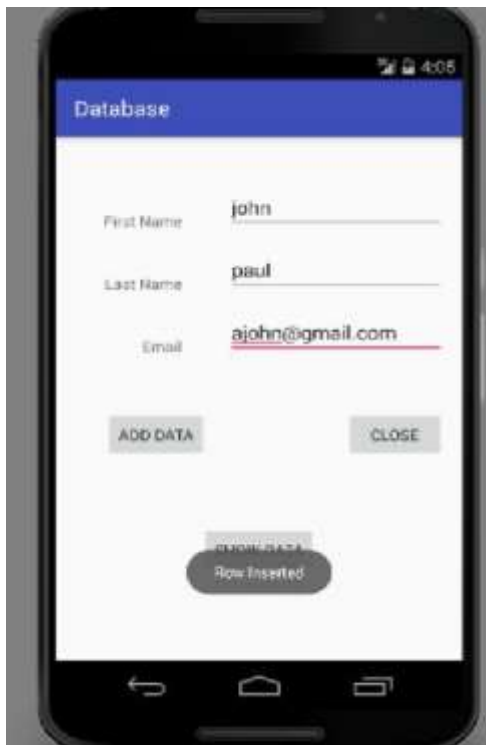
```

```

}}
});
}}

```

Step 6: Run the project. While running, the following output will be shown in the emulator.



3) Stage v (verify)

Home Activities:

1. Write XML code for generating the following layout. And send the information of both edit texts via email, sms and save in database. After saving in database, wait for 5 sec and move to next activity to get a view of database.

TEXT	
TEXT	EDIT TEXT
TEXT	EDIT TEXT
BUTTON	

Statement Purpose:

Today we will create a simple Timer timer app. We will learn the use of handler through this stopwatch app.

Activity Outcomes:

After completing this chapter we will be able to understand the following topics.

- Multithreading
- Multimedia in Android
- Media Player API
- Camera API
- Running the Multithreading App

Instructor Note:

Multithreading

Multi-threading is defined as a feature through which we can run two or more concurrent threads of a process. In this a process, the common data is shared among all these threads also known as sub-processes exclusively.

Multimedia in Android

Multimedia capabilities, or the playing and recording of audio and video, is one such significant task that many users find to be of great value. Take a quick look around us, we will find people using the phone as a means to enjoy a variety of programs as well as share self-recorded media among friends. Android provides APIs to easily access this capability as well as embed multimedia and its manipulation directly within an application. Usually multimedia involves in performing the following basic functions.

- Playing audio and video
- Controlling the camera
- Recording audio
- Recording video

We will discuss the ‘_Stagefright’ architecture from Google. Because the foundation of Android’s multimedia platform is Google’s new media platform Stagefright. Stagefright, as of now, supports the following core media files, services, and features:

- Interfaces for third-party and hardware media codecs, input and output devices, and content policies.
- *Media playback, streaming, downloading, and progressive playback, including third-Generation Partnership Program (3GPP), Moving Picture Experts Group 4 (MPEG-4), Advanced Audio Coding (AAC), and Moving Picture Experts Group (MPEG) Audio Layer 3 (MP3) containers.*
- *Network protocols including RTSP (TRP, SDP), HTTP progressive streaming, and HTTP live streaming.*

- Video and image encoders and decoders, including MPEG-4, International Telecommunication Union H.263 video standard (H.263), Advanced Video Coding (AVC H.264), and the Joint Photographic Experts Group (JPEG).
- Speech codecs, including Adaptive Multi-Rate audio codecs AMR-NB and AMR-WB.
- Audio codecs, including MP3, AAC, and AAC+ etc.
- Media recording, including 3GPP, VP8, MPEG-4, and JPEG.
- Video telephony based on the 3GPP video conferencing standard 324-M.

Media Player API

Android Multimedia Framework – android.media API

The android multimedia framework provides developers a way to easily integrate audio and video playback into applications, and supports most of the common media types. The MediaPlayer class is the key in android multimedia framework. It can be used to play media on the local file system, media files stored in the application's resources, as well as data streaming over a network connection.

Playing Audio

Probably the most basic need for multimedia on a cell phone is the ability to play audio files, whether new ringtones, MP3s, or quick audio notes. Media- Player of android is easy to use. To play an MP3 file follow these steps:

- *Place the MP3 in the res/raw directory in a project (note that we can also use a URI to access files on the network or via the internet).*
- *Create a new instance of the MediaPlayer, and reference the MP3 by calling MediaPlayer.create().*
- *Call the MediaPlayer methods prepare() and start().*

Playing Video

Playing a video is slightly more complicated than playing audio with the MediaPlayer API, because we have to provide a view surface for our video to play on. Android has a VideoView widget that handles this task for us. This widget can be used with any layout manager. Android also provides a number of display options, including scaling and tinting.
cseitquestions.blogspot.in cseitquestions.blogspot.in

Android Camera API

Usually every android device has at least one camera (back camera) or many devices have multiple cameras like front camera and rear camera. There is a default application available

in each android device to access this camera. We can also access this application into our application as intent. Alternatively, we can also have our own camera application using android camera API. There is set of methods and class that support camera in android device.

Camera API in Detail:

We create intent to access camera in android. This intent can created using default camera application or using android camera API. we will create an application using camera API and will create intent to access this. This application will capture an image and will save it to the image directory into the android device. Once an image has been captured, this application will preview this image, while saving. Let us create a new java class that will capture an image as listed.

Running the Multithreading App

Step 1: Select File -> New -> Project -> Android Application Project (or) AndroidProject. Fill the forms and click —Finish|| button.




Step 2: Open res -> layout -> activity_main.xml -> click Design -> Put the necessary components in the layout.

Step 3: Create a layout file for UI (Design for UI after that the code will be generated automatically in activity_main.xml)

1) Stage J (Journey)

Introduction

- Today we will create a simple stopwatch timer app. We will learn the use of handler through this stopwatch app. Following will be the functionality of our stopwatch timer app:

-  Start-pause the stopwatch timer app.
-  Reset the stopwatch timer app.
-  Color change while starting or pausing time

2) Stage a1 (apply)

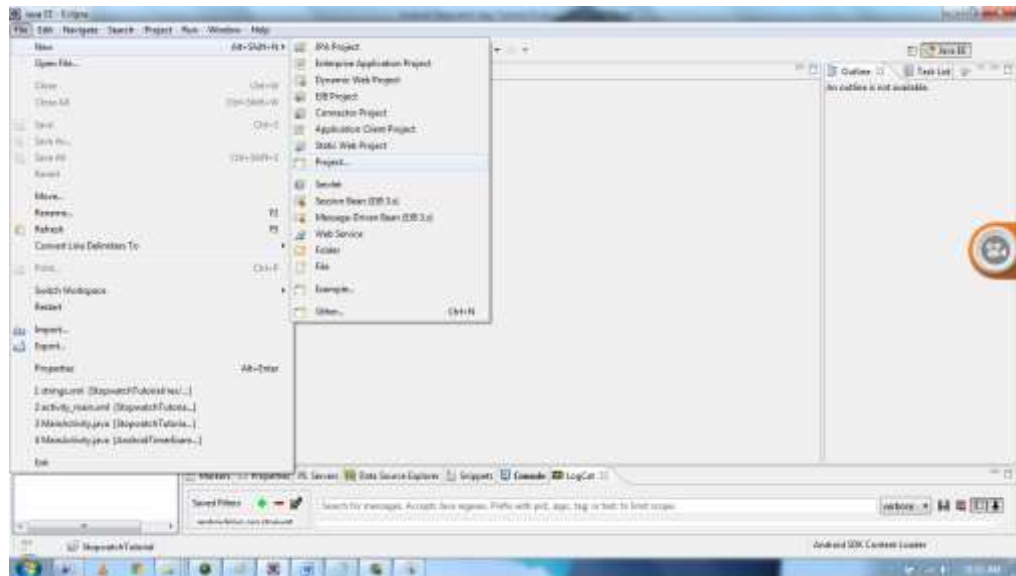
Lab Activities:

Activity 1:

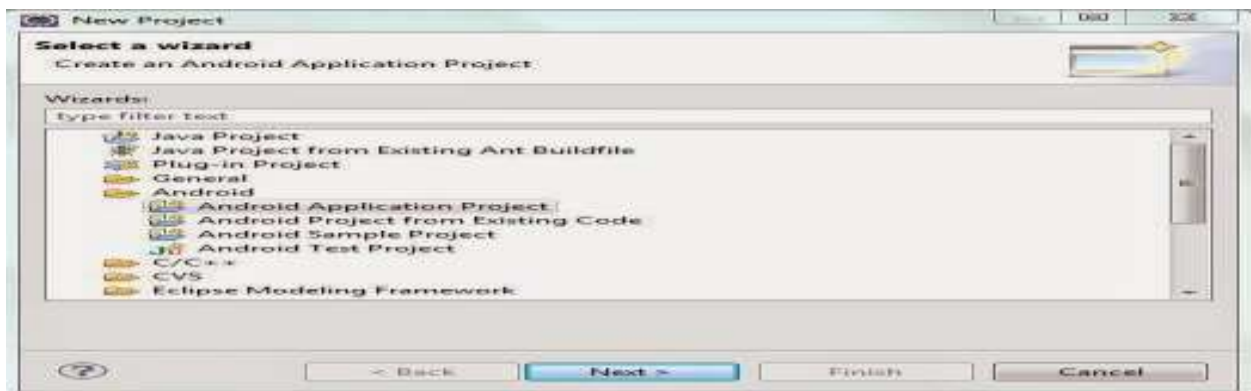
Steps for making App

- Project Setup for Timer App Tutorial:

- Go to **file-->new-->Project**



Select **Android-->Android Application Project**



Xml Files for Timer App Tutorial

- Go to **res-->values-->strings.xml** and add the following code into it.
- `<?xml version="1.0" encoding="utf-8"?>`
- `<resources>`
- `<string name="app_name">StopwatchTutorial</string>`
- `<string name="action_settings">Settings</string>`
- `<string name="hello_world">Hello world!</string>`
- `<string name="start_time">00:00:00</string>`
- `<string name="start">Start</string>`
- `<string name="reset">Reset</string>`
- `<string name="title">Androidplus Timer</string>`
- `</resources>`

```
package androidplus.org. Timertutorial;
import android.os.Bundle;
import android.os.Handler;
```

```

import android.os.SystemClock;
import android.app.Activity;
import android.graphics.Color;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MainActivity extends Activity {
    Button butnstart, butnreset;
    TextView time;
    long starttime = 0L;
    long timeInMilliseconds = 0L;
    long timeSwapBuff = 0L;
    long updatedtime = 0L;
    int t = 1;
    int secs = 0;
    int mins = 0;
    int milliseconds = 0;
    Handler handler = new Handler();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        butnstart = (Button) findViewById(R.id.start);
        butnreset = (Button) findViewById(R.id.reset);
        time = (TextView) findViewById(R.id.timer);
        butnstart.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                if (t == 1) {
                    butnstart.setText("Pause");
                    starttime = SystemClock.uptimeMillis();
                    handler.postDelayed(updateTimer, 0);
                    t = 0;
                } else {
                    butnstart.setText("Start");
                    time.setTextColor(Color.BLUE);
                    timeSwapBuff += timeInMilliseconds;
                    handler.removeCallbacks(updateTimer);
                    t = 1;
                }
            }
        });
    }
}

```

OUTPUT

3) Stage v (verify)

Home Activities:

You have to add a time delay to the taking of the picture. Make SnapShot example so that picture is taken ten seconds after pushing a button.

Statement Purpose:

Today we will create a simple stopwatch timer app. We will learn the use of handler through this stopwatch app.

Activity Outcomes:

After completing this chapter we will be able to understand the following topics.

- Multithreading
- Multimedia in Android
- Media Player API
- Camera API
- Running the Multithreading App

Instructor Note:

Multithreading

Multi-threading is defined as a feature through which we can run two or more concurrent threads of a process. In this a process, the common data is shared among all these threads also known as sub-processes exclusively.

Multimedia in Android

Multimedia capabilities, or the playing and recording of audio and video, is one such significant task that many users find to be of great value. Take a quick look around us, we will find people using the phone as a means to enjoy a variety of programs as well as share self-recorded media among friends. Android provides APIs to easily access this capability as well as embed multimedia and its manipulation directly within an application. Usually multimedia involves in performing the following basic functions.

- Playing audio and video
- Controlling the camera
- Recording audio
- Recording video

We will discuss the ‘_Stagefright’ architecture from Google. Because the foundation of Android’s multimedia platform is Google’s new media platform Stagefright. Stagefright, as of now, supports the following core media files, services, and features:

- Interfaces for third-party and hardware media codecs, input and output devices, and content policies.
- *Media playback, streaming, downloading, and progressive playback, including third-Generation Partnership Program (3GPP), Moving Picture Experts Group 4 (MPEG-4), Advanced Audio Coding (AAC), and Moving Picture Experts Group (MPEG) Audio Layer 3 (MP3) containers.*
- *Network protocols including RTSP (TRP, SDP), HTTP progressive streaming, and HTTP live streaming.*

- Video and image encoders and decoders, including MPEG-4, International Telecommunication Union H.263 video standard (H.263), Advanced Video Coding (AVC H.264), and the Joint Photographic Experts Group (JPEG).
- Speech codecs, including Adaptive Multi-Rate audio codecs AMR-NB and AMR-WB.
- Audio codecs, including MP3, AAC, and AAC+ etc.
- Media recording, including 3GPP, VP8, MPEG-4, and JPEG.
- Video telephony based on the 3GPP video conferencing standard 324-M.

Media Player API

Android Multimedia Framework – android.media API

The android multimedia framework provides developers a way to easily integrate audio and video playback into applications, and supports most of the common media types. The MediaPlayer class is the key in android multimedia framework. It can be used to play media on the local file system, media files stored in the application's resources, as well as data streaming over a network connection.

Playing Audio

Probably the most basic need for multimedia on a cell phone is the ability to play audio files, whether new ringtones, MP3s, or quick audio notes. Media- Player of android is easy to use. To play an MP3 file follow these steps:

- *Place the MP3 in the res/raw directory in a project (note that we can also use a URI to access files on the network or via the internet).*
- *Create a new instance of the MediaPlayer, and reference the MP3 by calling MediaPlayer.create().*
- *Call the MediaPlayer methods prepare() and start().*

Playing Video

Playing a video is slightly more complicated than playing audio with the MediaPlayer API, because we have to provide a view surface for our video to play on. Android has a VideoView widget that handles this task for us. This widget can be used with any layout manager. Android also provides a number of display options, including scaling and tinting.
cseitquestions.blogspot.in cseitquestions.blogspot.in

Android Camera API

Usually every android device has at least one camera (back camera) or many devices have multiple cameras like front camera and rear camera. There is a default application available

in each android device to access this camera. We can also access this application into our application as intent. Alternatively, we can also have our own camera application using android camera API. There is set of methods and class that support camera in android device.

Camera API in Detail:

We create intent to access camera in android. This intent can created using default camera application or using android camera API. we will create an application using camera API and will create intent to access this. This application will capture an image and will save it to the image directory into the android device. Once an image has been captured, this application will preview this image, while saving. Let us create a new java class that will capture an image as listed.

Running the Multithreading App

Step 1: Select File -> New -> Project -> Android Application Project (or) AndroidProject. Fill the forms and click —Finish|| button.




Step 2: Open res -> layout -> activity_main.xml -> click Design -> Put the necessary components in the layout.

Step 3: Create a layout file for UI (Design for UI after that the code will be generated automatically in activity_main.xml)

4) Stage J (Journey)

Introduction

- Today we will create a simple stopwatch timer app. We will learn the use of handler through this stopwatch app. Following will be the functionality of our stopwatch timer app:

-  Start-pause the stopwatch timer app.
-  Reset the stopwatch timer app.
-  Color change while starting or pausing time

5) Stage a1 (apply)

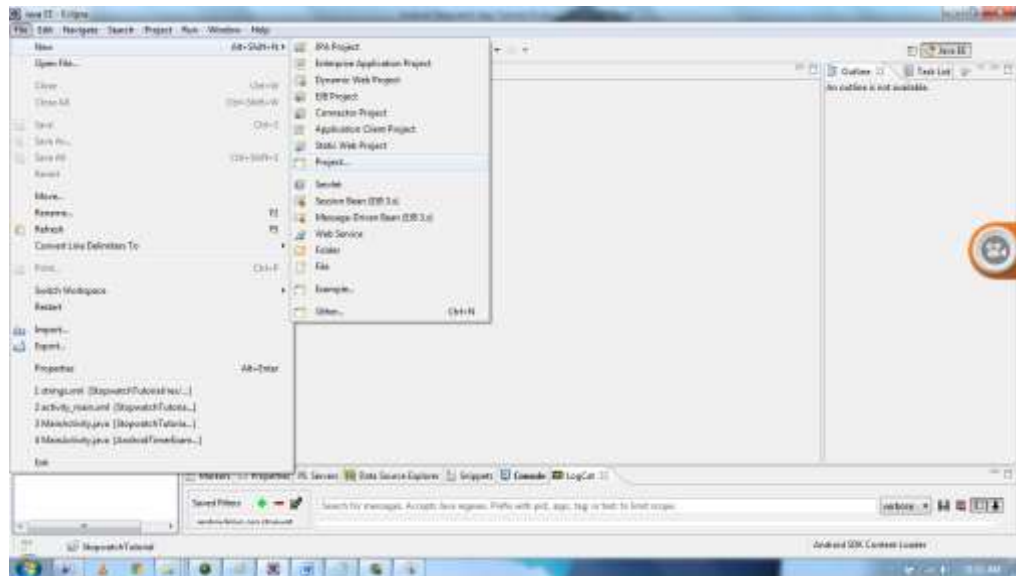
Lab Activities:

Activity 1:

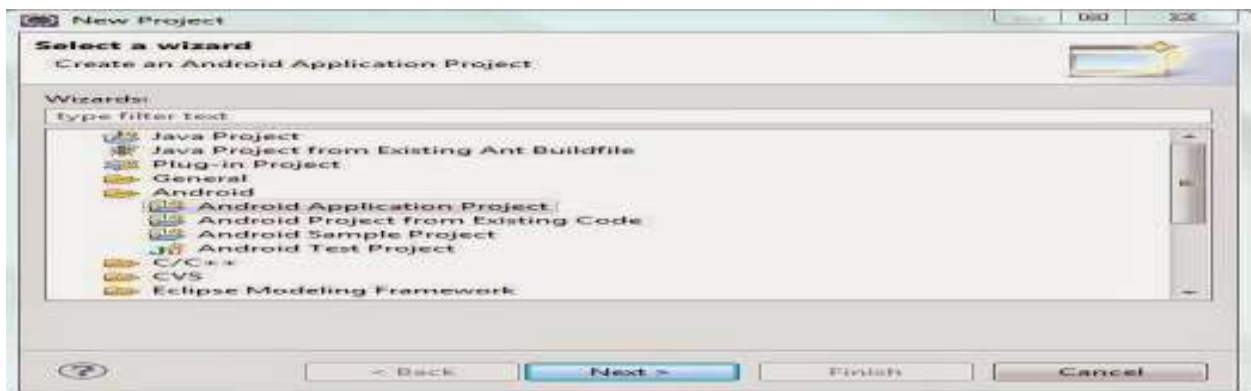
Steps for making App

- Project Setup for Stopwatch App Tutorial:

- Go to **file-->new-->Project**



Select **Android-->Android Application Project**



Xml Files for Stopwatch App Tutorial

- Go to **res-->values-->strings.xml** and add the following code into it.
- `<?xml version="1.0" encoding="utf-8"?>`
- `<resources>`
- `<string name="app_name">StopwatchTutorial</string>`
- `<string name="action_settings">Settings</string>`
- `<string name="hello_world">Hello world!</string>`
- `<string name="start_time">00:00:00</string>`
- `<string name="start">Start</string>`
- `<string name="reset">Reset</string>`
- `<string name="title">Androidplus Timer</string>`
- `</resources>`



```
package androidplus.org.stopwatchtutorial;
import android.os.Bundle;
import android.os.Handler;
import android.os.SystemClock;
import android.app.Activity;
import android.graphics.Color;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
public class MainActivity extends Activity {
    Button butnstart, butnreset;
    TextView time;
    long starttime = 0L;
    long timeInMilliseconds = 0L;
    long timeSwapBuff = 0L;
    long updatedtime = 0L;
    int t = 1;
    int secs = 0;
    int mins = 0;
    int milliseconds = 0;
    Handler handler = new Handler();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        butnstart = (Button) findViewById(R.id.start);
```

```

butnreset = (Button) findViewById(R.id.reset);
time = (TextView) findViewById(R.id.timer);
butnstart.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        // TODO Auto-generated method stub
        if (t == 1) {
            butnstart.setText("Pause");
            starttime = SystemClock.uptimeMillis();
            handler.postDelayed(updateTimer, 0);
            t = 0;
        } else {
            butnstart.setText("Start");
            time.setTextColor(Color.BLUE);
            timeSwapBuff += timeInMilliseconds;
            handler.removeCallbacks(updateTimer);
            t = 1;
        }
    }
});

```

OUTPUT



6) Stage v (verify)

Home Activities:

You have to add a time delay to the taking of the picture. Make Snapshot example so that picture is taken ten seconds after pushing a button.

Statement Purpose:

Students will learn that how to set *An ordered collection of selectable choices*.

Activity Outcomes:

Instructor Note:

List adapters

- Adapter: Helps turn list data into listviewitems.

–common adapters: ArrayAdapter, CursorAdapter

- Syntax for creating an adapter:

ArrayAdapter<String> name = new ArrayAdapter<String>(activity, layout, array); the activity is usually this

- the default *layout* for lists is android.R.layout.simple_list_item_1

- get the *array* by reading your file or data source of choice (it can be an array like String[], or a list like ArrayList<String>)

–Once you have an adapter, you can attach it to your list by calling the setAdapter method of the ListView object in the Java code.

Handling list events

- Unfortunately lists don't use a simple onClick event.

–The event listeners must be attached in the Java code.

List events

- List views respond to the following events:

–setOnItemClickListener(AdapterView.OnItemClickListener)

Listener for when an item in the list has been clicked.

–setOnItemLongClickListener(AdapterView.OnItemLongClickListener)

Listener for when an item in the list has been clicked and held.

–setOnItemSelectedListener(AdapterView.OnItemSelectedListener)

Listener for when an item in the list has been selected.

1) Stage J (Journey)

Introduction

The classic listbox widget in Android is known as ListView. Include one of these in your layout, invoke setAdapter() to supply your data and child views, and attach a listener via setOnItemSelectedListener() to find out when the selection has changed. With that, you have a fully-functioning listbox.

However, if your activity is dominated by a single list, you might well consider creating your activity as a subclass of ListActivity, rather than the regular Activity base class. If your main view is just the list, you do not even need to supply a layout—ListActivity will construct a full-screen list for you. If you do want to customize the layout, you can, so long as you identify your ListView as @android:id/list, so ListActivity knows which widget is the main list for the activity.

2) Stage a1 (apply)

Lab Activities:

Static lists

static list: Content is fixed and known before the app runs.

–Declare the list elements in the **strings.xml** resource file.

```
<!--res/values/strings.xml -->
<resources>
<string-array name="oses">
<item>Android</item>
<item>iPhone</item>
...
<item>Max OS X</item>
</string-array>
</resources>
<!--res/layout/activity_main.xml -->
<ListView... android:id="@+id/mylist"
android:entries="@array/oses" />
```

Dynamic lists

Content is read or generated as the program runs.

–Comes from a array, or a data file, or from the internet, etc.

–Must be set in the Java code.

–Suppose we have the following file and want to make a list from it:

```
// res/raw/oses.txt
Android
iPhone
...
Max OS X
import java.util.*;
public class ArrayListExamples{
public static void main(String args[]) {
// Creating an empty array list
ArrayList<String> list = new ArrayList<String>();
// Adding items to arrayList
list.add("Item1");
list.add("Item2");
list.add(2, "Item3"); // it will add Item3 to the third position of array list
list.add("Item4");
// Getting the size of the list
intsize = list.size();
System.out.println("The size of the list is: " + size);
// Getting the element in a specific position
```

```
String item = list.get(0);
System.out.println("The item is the index 0 is: " + item);
// Retrieve elements from the arraylist
// loop using index and size list
System.out.println("Retrieving items with loop using index and size list");
for (inti= 0; i< list.size(); i++) {
System.out.println("Index: " + i+ " -Item: " + list.get(i));
}}
```

LIST ADAPTER ACTIVITY

```
ArrayList<String> myArray= ...; // load or store data .....
ArrayAdapter<String> myAdapter= new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1,
myArray);
ListViewlist = (ListView) findViewById(R.id.mylist);
list.setAdapter(myAdapter);
```

LIST EVENT LISTENER ACTIVITY

```
OnCreate(...) {
list = (ListView) findViewById(R.id.listView);
list.setAdapter(myAdapter);
list.setOnItemClickListener(this);
}
@Override
public void onItemClick(AdapterView<?> parent, View view, intposition, long id) {
tv1.setText("AAAA == " + position);
}
```

LIST EVENT LISTENER ACTIVITY

```
ListViewlist = (ListView) findViewById(R.id.id);
list.setOnItemClickListener(
new AdapterView.OnItemClickListener() {
@Override
public void onItemClick(AdapterView<?> list,
View row,
intindex,
long rowID) {
// code to run when user clicks that item
...
}
}
);
```

3) Stage v (verify)

Home Activities:

Here are some things you can try beyond those step-by-step instructions:

- See what the activity looks like if you use a `Spinner` instead of a `ListView`.
- Make the address field, presently an `EditText` widget, into an `AutoCompleteTextView`, using the other addresses as values to possibly reuse (e.g., for multiple restaurants in one place, such as a food court or mall).

Statement Purpose:

Activity Outcomes:

After completing this chapter we will be able to understand the following topics.

- Introduction to Services
- Lifecycle of a Service
- Managing Services
- Introduction to Broadcast Receivers
- Types of Broadcast Receivers
- Creating and Registering Broadcast Receivers

Instructor Note:

Students must read and have best practice of past labs.

1) Stage J (Journey)

Introduction

Introduction to Services

Service can be described as long running background app without any user interface (UI). The services will be running as a separate thread, while any other activity and task is running on fore ground. It can be managed by app component such as activity. While developing android app, we should be able to manage different task simultaneously. For example, playing music in background while playing games. Such simultaneous tasks are managed by services. To interact with services we need to create an activity. For example, service might interact with a content provider, music player and so on. All these tasks will be running in the background.

Forms of Services

Services are used for regularly updating the data source in background. The updated data source is used by activities running on foreground. Services are mainly used for triggering notifications such as email or message arrival. Services are controlled (i.e. Started/stopped) from other android app components such as broadcast receivers, activities and other service components. Services are classified into two types. They are

- Unbound Services
- Bound Services

a) UnBound Service

The unbound service is also known as started service. The `startService()` method is used to start the service, once the service is started it will be running in background for indefinite period of time, even if we destroy the started component. Started service performs the intended operation and does not return any result to the caller component.

When we open the game, music starts playing in background automatically till we exit the game.

b) Bound Services

Bound Service can be referred as client–server approach. The bound services allow the app components to interact with each other. It allows the app components to send request and get result using inter process communication (IPC). Bound service uses `bindService()` method to bind with one or more app components. Bound service will be running until another component is bound to it. Multiple components can be binded to the service at a time. The service will be destroyed when the bind components are unbind.

Lifecycle of a Service

The lifecycle of a service can follow two different paths depending on the form of the service, which can be either unbound or bound.

To understand the lifecycle of a service, consider the example like playing games. When we open the game app, an activity is started. During the lifetime of this app, activity may invoke a service.

i.e. service to enable or disable the music by changing the game or exiting the game. If user exits the game app, all service components acquired by the app are released and destroyed.

Lifecycle of UnBound (Started) Service

To invoke the service component `startService()` method need to be called. Once the service is started, it is running in background for indefinite period of time :

- The service stops itself by calling the `stopSelf()` method.
- Another app component stops the service by calling the `stopService()` method. When a service is stopped, when the system destroys service components and its resources.

The figure depicts the Callback method Executed during the Lifecycle of a started service or unbound service. **Lifecycle of Bound Service** We can create bound service from another components by calling `bindService()` method. While calling the `bindService()` method, the service is bind to the components. We can bind multiple components with a service. Bound services communicate through `IBinder` interface. It can be unbind by calling `unbindService()` method. Service need not to be explicitly stopped, it is automatically stopped and destroy all

the resources which are bind to that service component. Stopping services automatically, is a good practice in app development because if a service is not stopped itself, optimization of resources cannot be achieved.

Starting a Service

When the service component call the `startService()` method, it will check whether the service is running or not, if service is not running the android system will call `onCreate()` method, if service is already running in background then android system will call `onStartCommand()` method. When multiple components start a service, they call `onStartCommand()` method then each start request is called individually.

The service will start running after receiving request from the component. It will continue running, until it `stopSelf()` or `stopService()` method is called. We have to note one point that even multiple requests made to the `startService()` method, the call to the `startService()` method will not be nested. No matter how many requests or calls to the `startService()` method made, but the service will be stopped once the `stopService()` or `stopSelf()` method is called. . In bound service, the component that starts the service can create a `PendingIntent` [it is a token that is given to another app (e.g. `NotificationManager`, `AlarmManager`, `Home Screen AppWidgetManager`), which allows that app to use current app's permissions to execute a predefined piece of code] object and pass this intent object to the caller service components. The service can then use this object to send a result back to the calling component.

Stopping a Service

As discussed previously service is independent block of code, it needs to be manage its own lifetime. The service will be running in the background until android platform regain the memory occupied by the service. When service is ideal for long time, it has to be stopped by itself by calling `stopSelf()` callback method. To destroy the service explicitly we need to call the `stopService(Intent Service)` callback method. Unlike the `onStartCommand()` method, only one call to the `stopService` or `stopSelf()` method destroys the service.

Introduction to Broadcast Receivers

In an android device, broadcast message is generated, to inform the currently running app that certain event has occurred.

Broadcasts can be generated in two ways:

❑ The system-level event broadcast:

The broadcast message generated by the system is known as system-level event broadcast, such as the screen is being turned off, the battery is low, or an SMS has arrived.

❑ An app-level event broadcast:

The broadcast message generated by an app is known as app-level event broadcast, For example, an app that is downloading some data may want to inform other apps that the

data has been downloaded and ready to use. By using app-level event broadcast, we can inform other apps that some event has occurred. This can be achieved using a broadcast receiver, which enables the apps to register themselves to receive a notification whenever a new feed is available.

Normal Broadcasts

The broadcasts messenger sends message in undefined order to all the registered receivers at the same time. For example, the broadcasts, such as battery low (`ACTION_BATTERYLOW`) or timezone changed (`ACTION_TIMEZONE_CHANGED`) will be received by all the registered receivers at the same time. This type of broadcasts are sent using the `sendBroadcast()` method of the `android.content.Context` class.

Ordered Broadcasts

The broadcast messenger sends message to all the registered receivers in an ordered manner, which means that a broadcast is delivered to one receiver at a time. When a receiver receives a broadcast, it can either propagate the broadcast to the next receiver or it can completely abort the broadcast. If a broadcast message is aborted by a receiver, it will not be passed to other receivers.

Creating and Registering a Broadcast Receiver

1. Creating the Broadcast Receiver

Each broadcast is delivered in the form of an intent object. Broadcast receivers enable apps to receive intents object that are either broadcasted by the system or by other apps. A broadcast receiver is implemented as a subclass of Broadcast Receiver class and overriding the `onReceive()` method where each message is received as a Intent object parameter.

The `onReceive()` method is called when a broadcast receiver receives a broadcast intent object. This method must include the code that needs to be executed when a broadcast is received. This method is usually called within the main thread of its process. Therefore, we should not perform long-running operations in this method. The system allows a timeout of 10 seconds before considering the receiver to be blocked. The system can kill a receiver that has been blocked. Also, we cannot launch a pop-up dialog in our `onReceive ()` method implementation. The signature of the `onReceive ()` method is:

```
public abstract void onReceive (Context context, Intent intent)
```

Registering a Broadcast Receiver

To register the broadcast receiver, app's broadcast intent has to be registered in `AndroidManifest.xml` file.

To declare a broadcast receiver in the manifest file, you need to add a `<receiver>` element as a child of the `<application>` element by specifying the class name of the broadcast receiver

to register. The <receiver> element also needs to include an <intent-filter> element that specifies the action string being listened for.

Broadcaster receivers can be registered by either one the two methods.

- Statically
- Dynamically

By using the method `registerReceiver()` a broadcast receiver is registered dynamically. If the manifest file is used for registering the broadcast receiver, it is known as static method.

Dynamic Registering

We can register a broadcast receiver using the `registerReceiver()` method. The `registerReceiver()` method is called with a broadcast intent that matches the filter in the main app thread. The signature of the `registerReceiver()` method is :

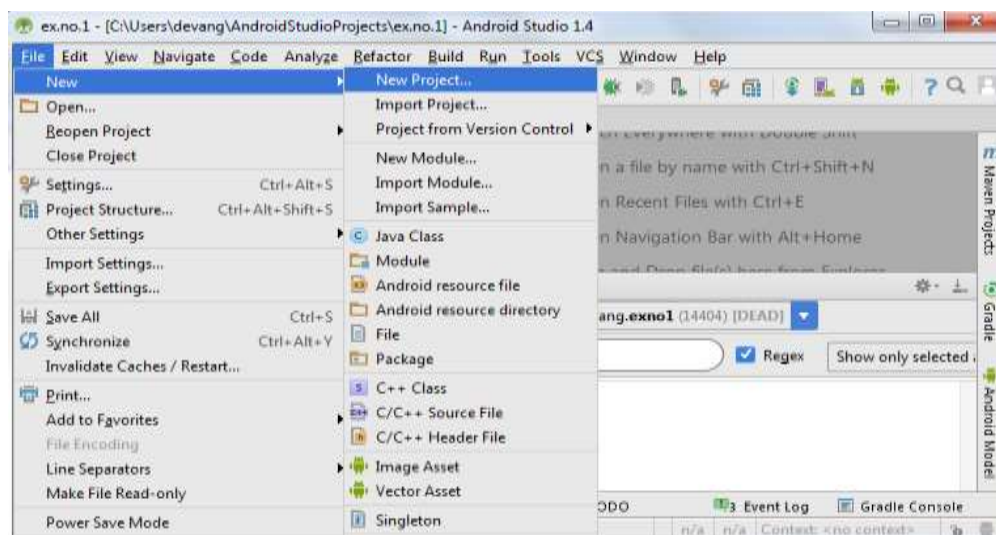
```
public abstract Intent registerReceiver(BroadcastReceiver receiver, IntentFilter filter)
```

2) Stage a1 (apply)

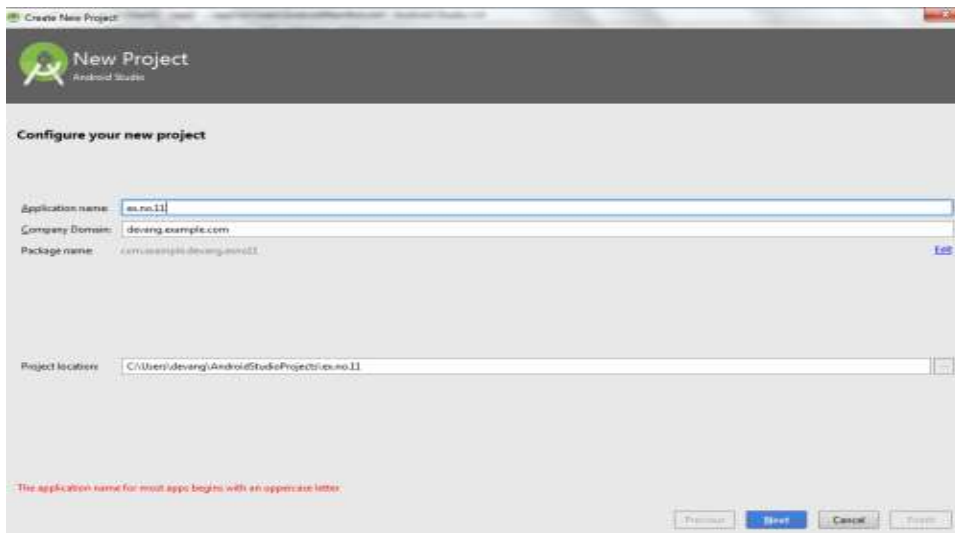
Lab Activities:

Activity 1:Creating a New project:

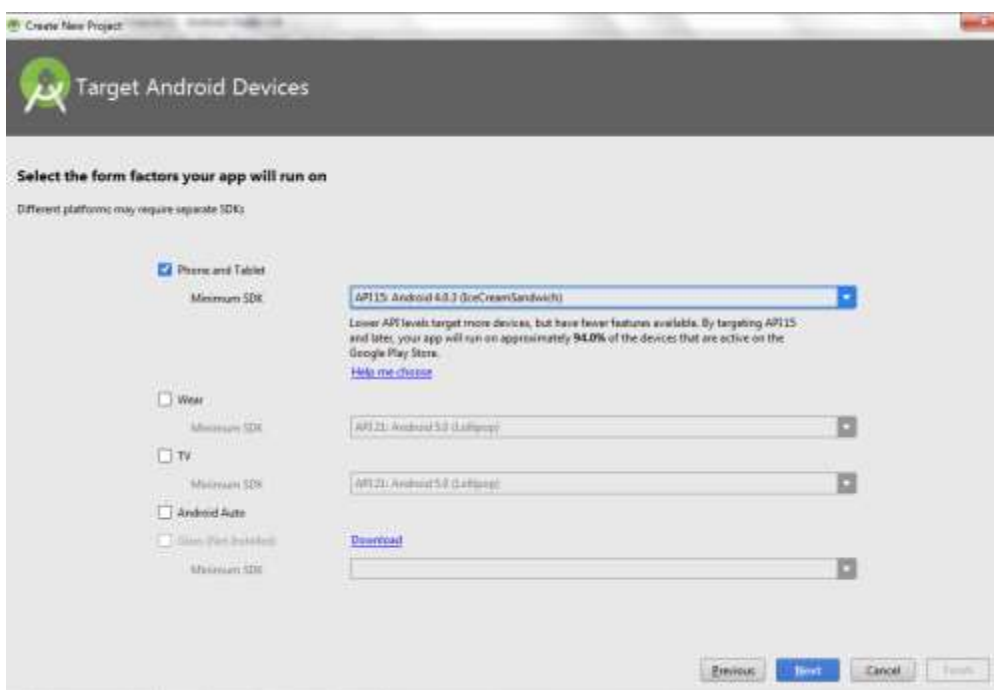
- Open Android Studio and then click on **File -> New -> New project.**



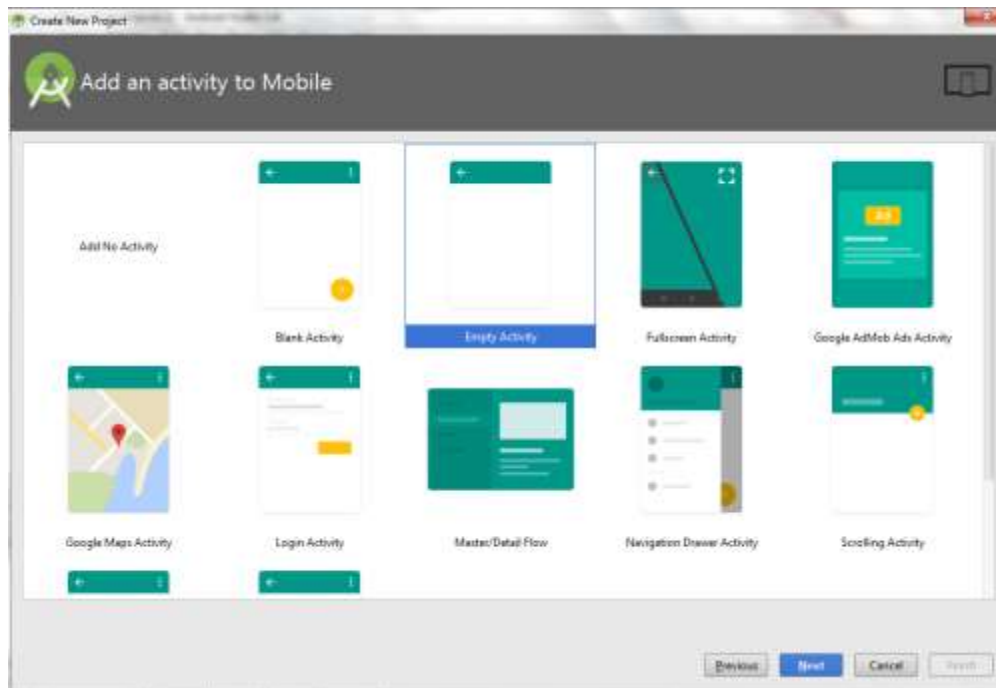
- Then type the Application name as “**ex.no.11**” and click **Next**.



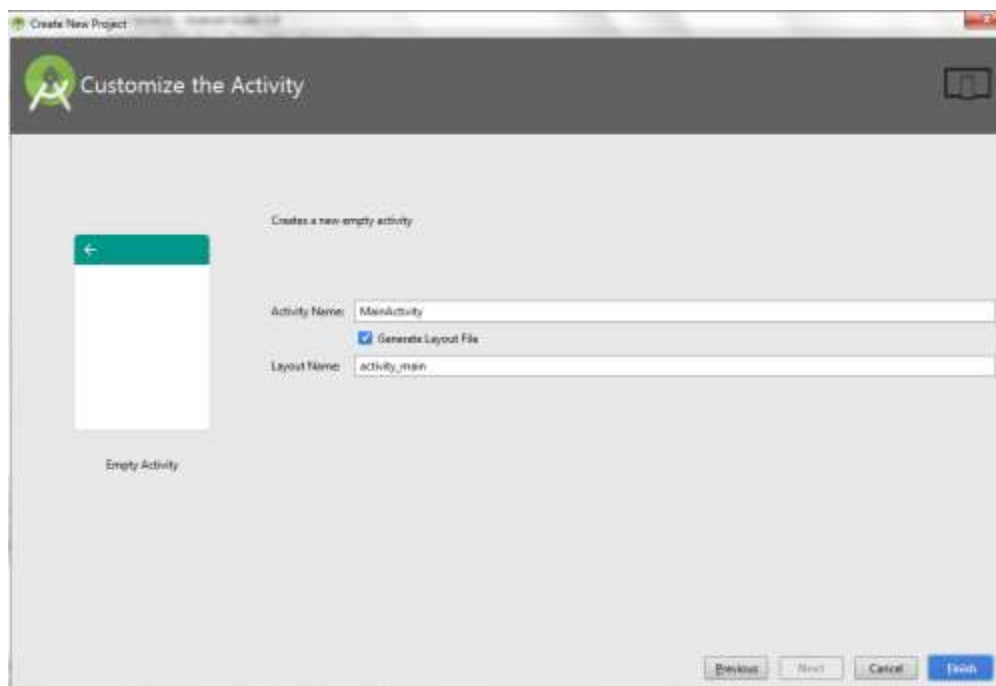
- Then select the **Minimum SDK** as shown below and click **Next**.



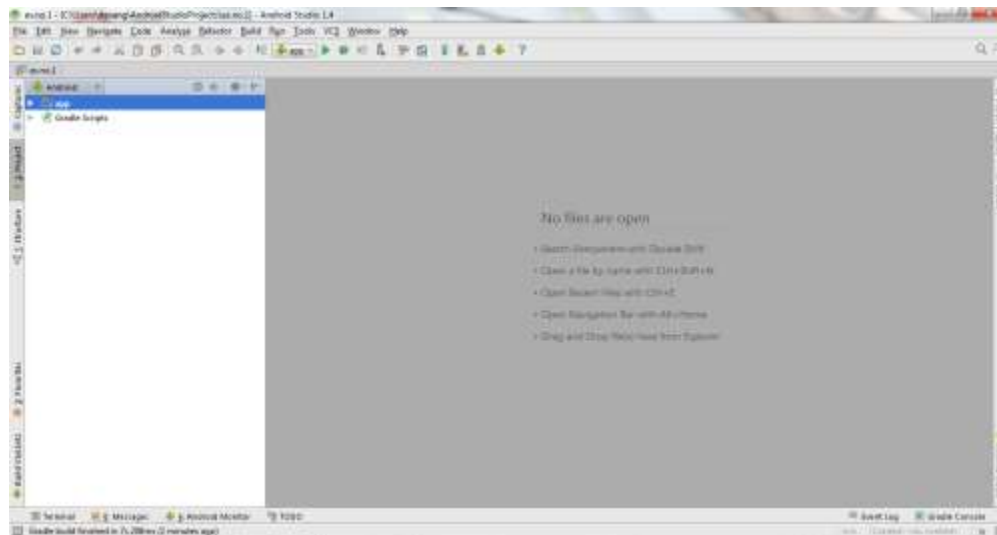
- Then select the **Empty Activity** and click **Next**.



- Finally click **Finish**.

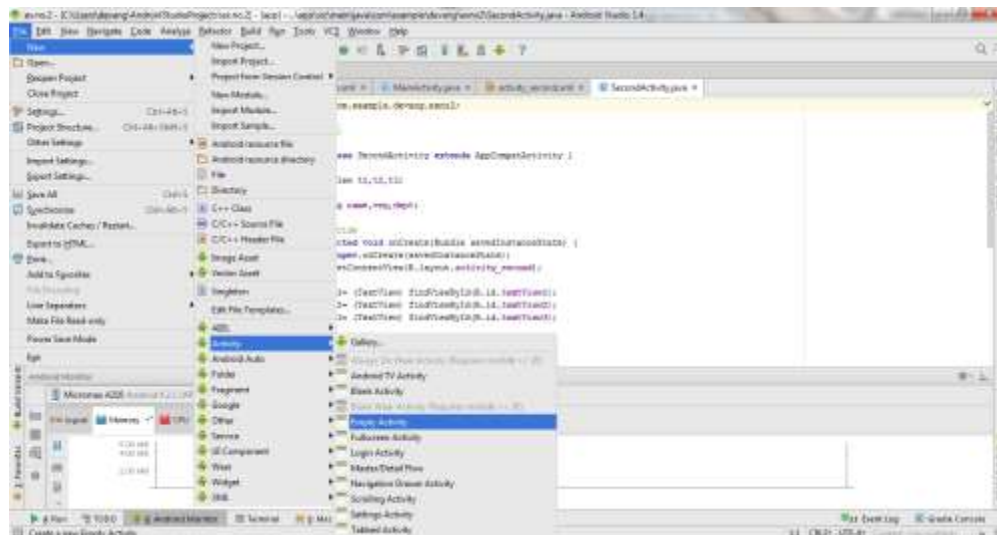


- It will take some time to build and load the project.
- After completion it will look as given below.

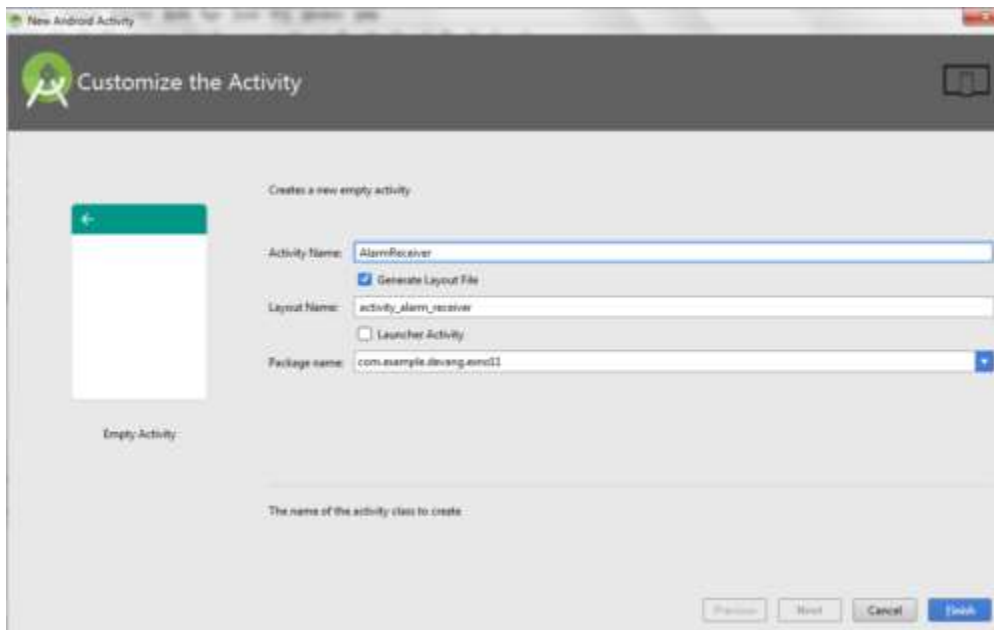


Creating Second Activity for the Android Application:

- Click on **File -> New -> Activity -> Empty Activity**.



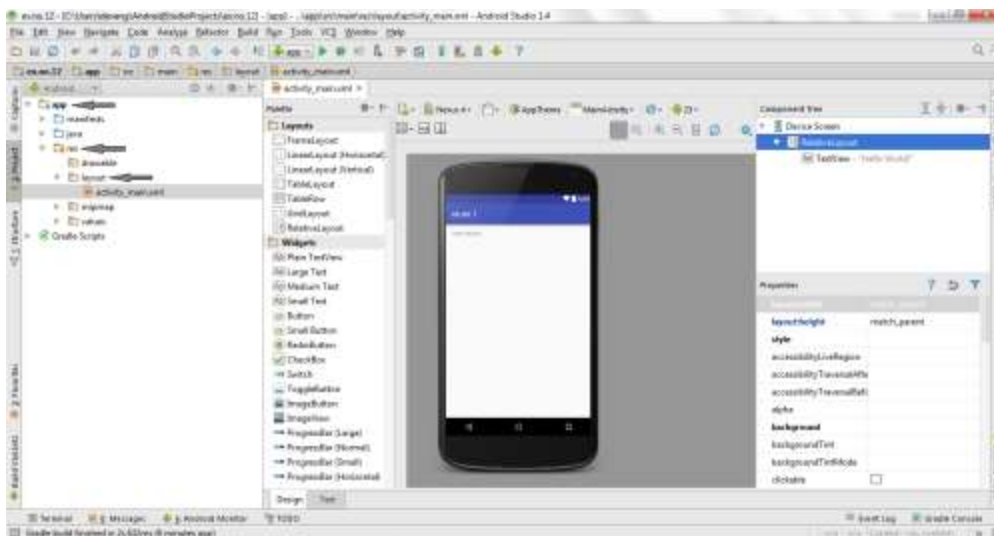
- Type the Activity Name as **AlarmReceiver** and click **Finish** button.



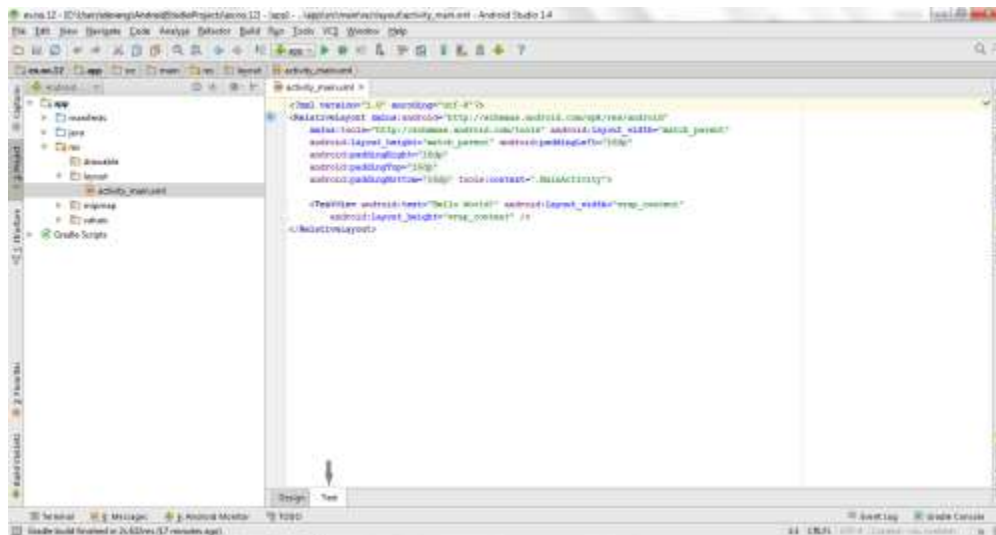
- Thus Second Activity For the application is created.

Designing layout for the Android Application:

- Click on **app** -> **res** -> **layout** -> **activity_main.xml**.



- Now click on **Text** as shown below.



- Then delete the code which is there and type the code as given below.

Code for Activity_main.xml:

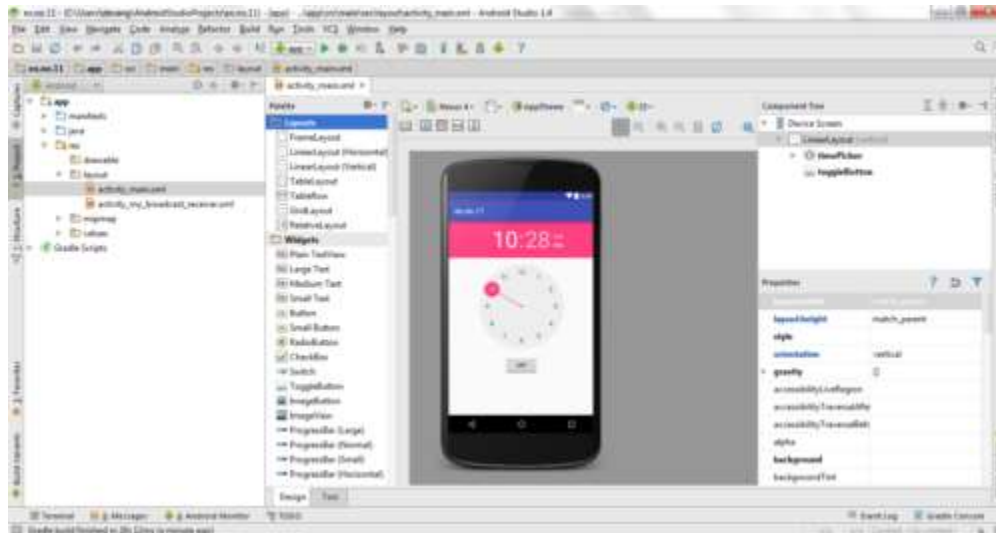
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TimePicker
        android:id="@+id/timePicker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center" />

    <ToggleButton
        android:id="@+id/toggleButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center"
        android:layout_margin="20dp"
        android:checked="false"
        android:onClick="OnToggleClicked" />

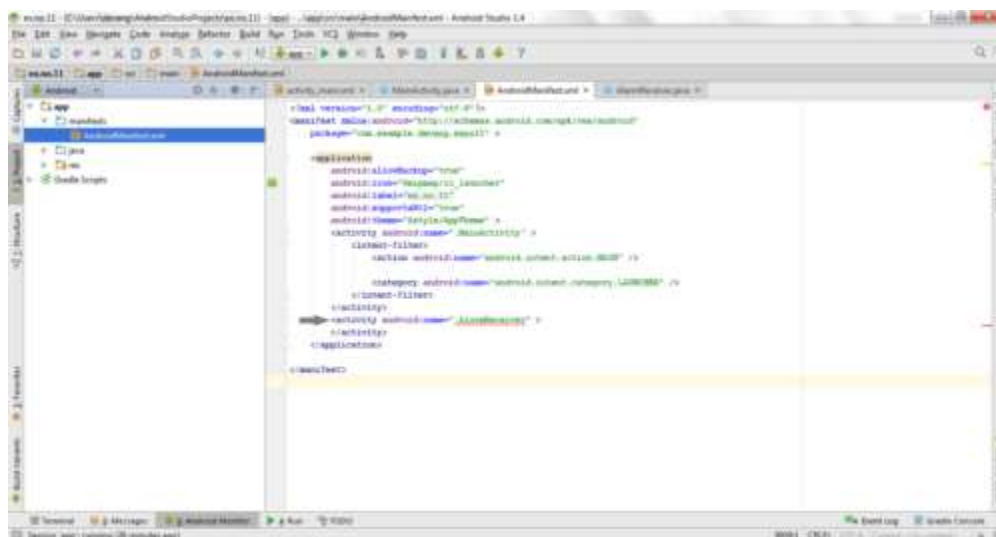
</LinearLayout>
```

- Now click on **Design** and your application will look as given below.

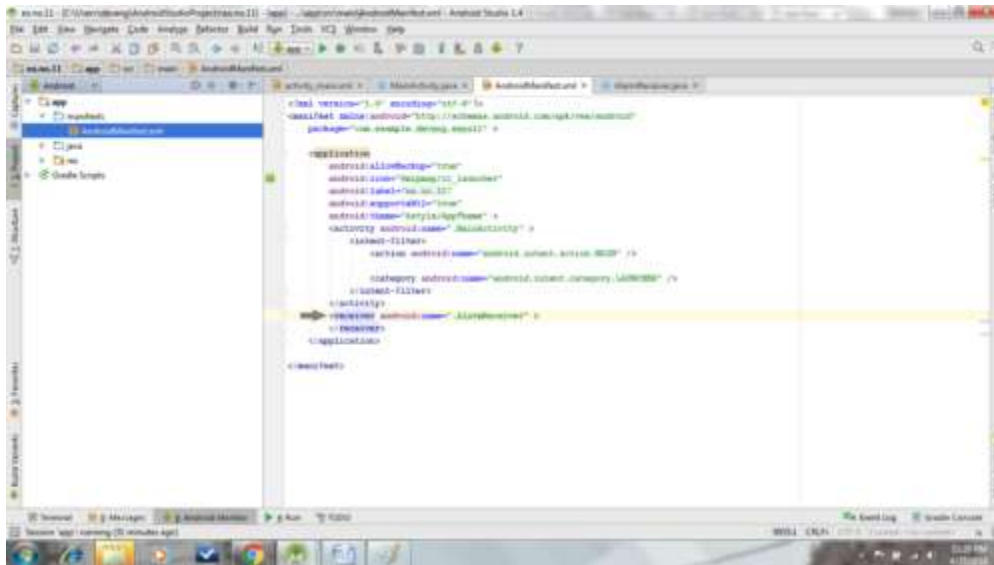


- So now the designing part is completed.

Changes in Manifest for the Android Application: Click on app -> manifests -> AndroidManifest.xml



- Now change the **activity tag** to **receiver tag** in the AndroidManifest.xml file as shown below



Code for AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.exno11" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:name=".AlarmReceiver" >
        </receiver>
    </application>

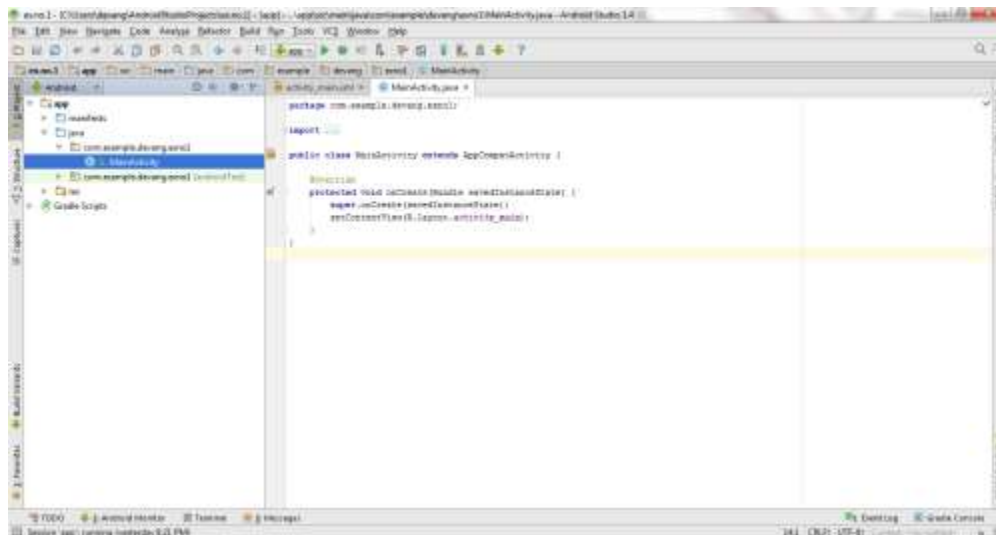
</manifest>
```

- So now the changes are done in the Manifest.

Java Coding for the Android Application:

Java Coding for Main Activity:

- Click on **app** -> **java** -> **com.example.exno11** -> **MainActivity**.



- Then delete the code which is there and type the code as given below.

Code for MainActivity.java:

```
package com.example.exnoll;

import android.app.AlarmManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.TimePicker;
import android.widget.Toast;
import android.widget.ToggleButton;

import java.util.Calendar;

public class MainActivity extends AppCompatActivity
{
    TimePicker alarmTimePicker;
    PendingIntent pendingIntent;
    AlarmManager alarmManager;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        alarmTimePicker = (TimePicker) findViewById(R.id.timePicker);
        alarmManager = (AlarmManager) getSystemService(ALARM_SERVICE);
    }
    public void OnToggleClicked(View view)
    {
        long time;
        if (((ToggleButton) view).isChecked())
        {
            Toast.makeText(MainActivity.this, "ALARM ON",
            Toast.LENGTH_SHORT).show();
            Calendar calendar = Calendar.getInstance();
            calendar.set(Calendar.HOUR_OF_DAY,
            alarmTimePicker.getCurrentHour());
        }
    }
}
```

```

        calendar.set(Calendar.MINUTE,
alarmTimePicker.getCurrentMinute());
        Intent intent = new Intent(this, AlarmReceiver.class);
        pendingIntent = PendingIntent.getBroadcast(this, 0, intent, 0);

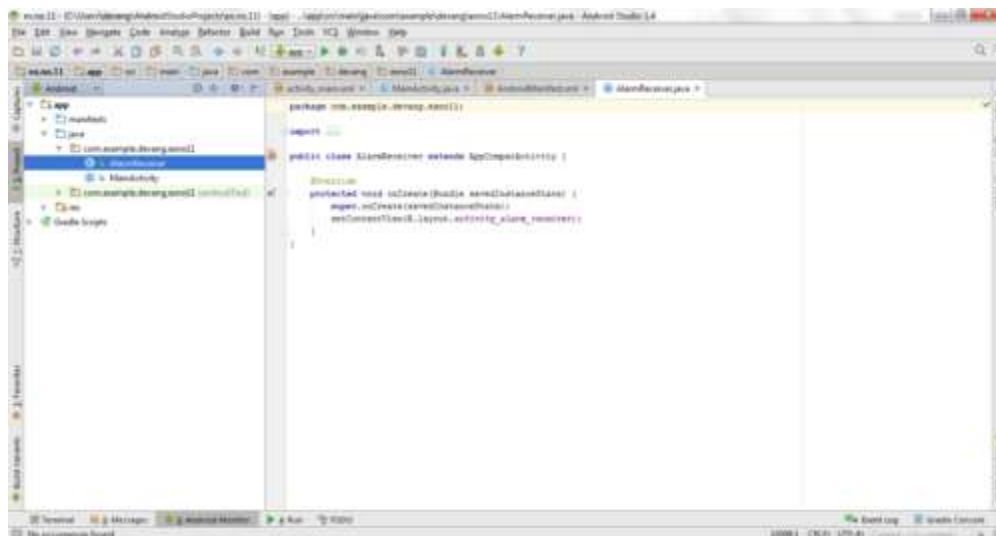
        time=(calendar.getTimeInMillis()-
(calendar.getTimeInMillis()%60000));
        if(System.currentTimeMillis()>time)
        {
            if (calendar.AM_PM == 0)
                time = time + (1000*60*60*12);
            else
                time = time + (1000*60*60*24);
        }
        alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, time, 10000,
pendingIntent);
    }
    else
    {
        alarmManager.cancel(pendingIntent);
        Toast.makeText(MainActivity.this, "ALARM OFF",
Toast.LENGTH_SHORT).show();
    }
}
}
}

```

- So now the Coding part of Main Activity is completed.

Java Coding for Alarm Receiver:

- Click on **app -> java -> com.example.exno11 -> AlarmReceiver**.



- Then delete the code which is there and type the code as given below.

Code for AlarmReceiver.java:

```

package com.example.exno11;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.media.Ringtone;
import android.media.RingtoneManager;
import android.net.Uri;
import android.widget.Toast;

```

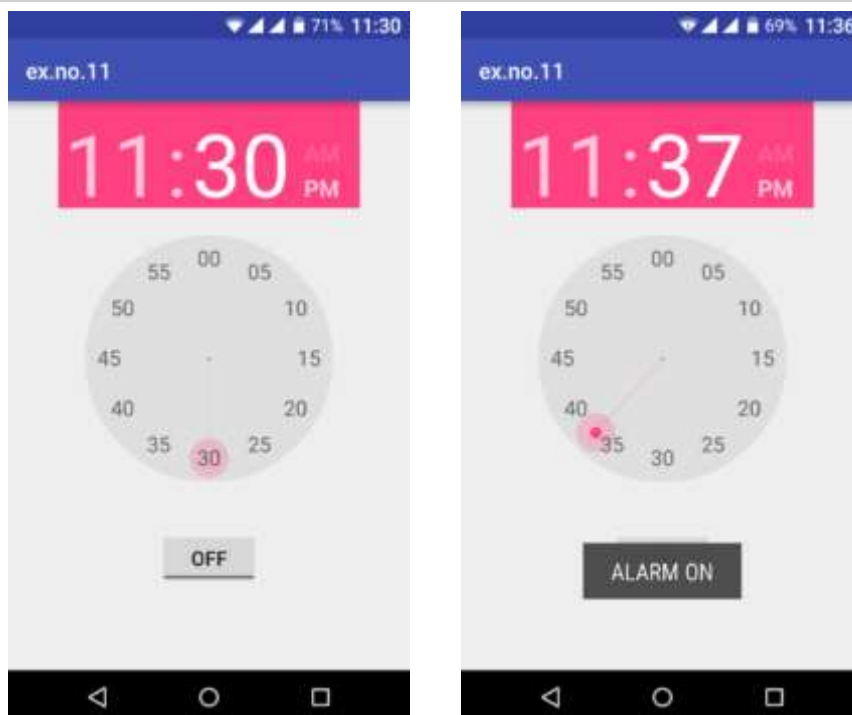
```

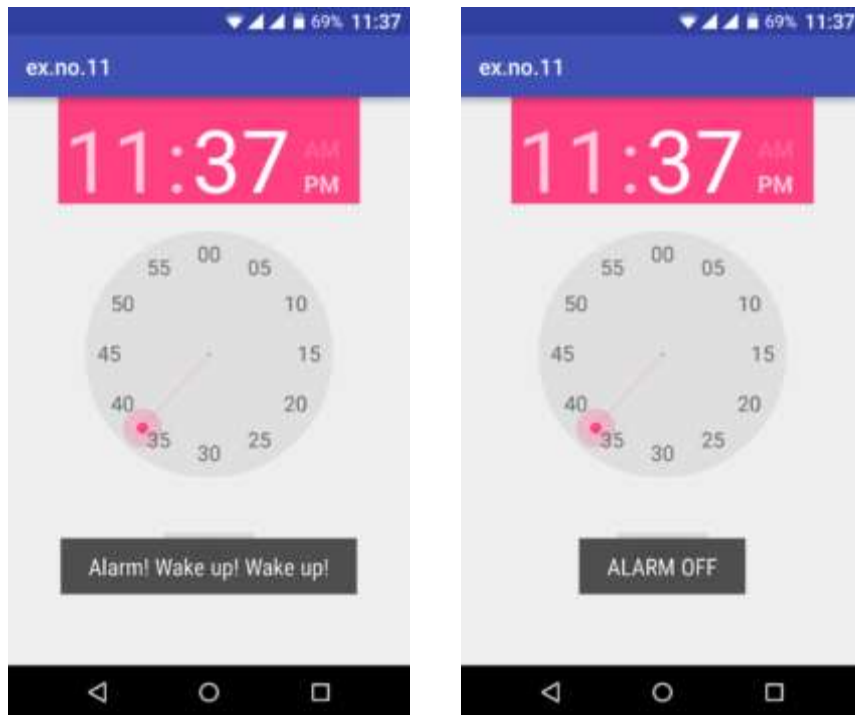
public class AlarmReceiver extends BroadcastReceiver
{
    @Override
    public void onReceive(Context context, Intent intent)
    {
        Toast.makeText(context, "Alarm! Wake up! Wake up!",
        Toast.LENGTH_LONG).show();
        Uri alarmUri =
        RingtoneManager.getDefaultUri(RingtoneManager.TYPE_ALARM);
        if (alarmUri == null)
        {
            alarmUri =
            RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION);
        }
        Ringtone ringtone = RingtoneManager.getRingtone(context, alarmUri);
        ringtone.play();
    }
}

```

- So now the Coding part of Alarm Receiver is also completed.
- Now run the application to see the output.

Output:





Result:

Thus Android Application that creates Alarm Clock is developed and executed successfully.

3) Stage v (verify)

Home Activities:

Activity 1:

Auto alarm generator for next hour.

Statement Purpose:

Here, we are going to see a simple example to play the audio/video file. In the next page, we will see the example to control the audio playback like start, stop, pause etc.

Activity Outcomes:

Students can play and control the audio/video files in android by the help of **MediaPlayer** class.

Instructor Note:

Media Player API

Android Multimedia Framework – android.media API

The android multimedia framework provides developers a way to easily integrate audio and video playback into applications, and supports most of the common media types. The MediaPlayer class is the key in android multimedia framework. It can be used to play media on the local file system, media files stored in the application's resources, as well as data streaming over a network connection.

1) Stage J (Journey)

Introduction

playing Audio

Probably the most basic need for multimedia on a cell phone is the ability to play audio files, whether new ringtones, MP3s, or quick audio notes. Media- Player of android is easy to use. To play an MP3 file follow these steps:

- ✓ Place the MP3 in the res/raw directory in a project (note that we can also use a URI to access files on the network or via the internet).
- ✓ Create a new instance of the MediaPlayer, and reference the MP3 by calling MediaPlayer.create().
- ✓ Call the MediaPlayer methods prepare() and start().

Playing Video

Playing a video is slightly more complicated than playing audio with the MediaPlayer API, because we have to provide a view surface for our video to play on. Android has a Video View widget that handles this task for us. This widget can be used with any layout manager. Android also provides a number of display options, including scaling and tinting.

2) Stage a1 (apply)

Lab Activities:

Activity 1:

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    tools:context="com.example.ait.mediaplayer.MainActivity">
    <TextView android:text="Music Palyer" android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/textview"
        android:textSize="35sp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        tools:ignore="HardcodedText" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Tutorials point"
        android:id="@+id/textView"
        android:layout_below="@+id/textview"
        android:layout_centerHorizontal="true"
        android:textColor="#ff7aff24"
        android:textSize="35sp"
        tools:ignore="HardcodedText" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/forward"
        android:id="@+id/button"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pause"
        android:id="@+id/button2"
        android:layout_alignParentBottom="true"
        android:layout_alignLeft="@+id/imageView"
        android:layout_alignStart="@+id/imageView" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/back"
        android:id="@+id/button3"
        android:layout_alignTop="@+id/button2"
        android:layout_toRightOf="@+id/button2"
        android:layout_toEndOf="@+id/button2" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/rewind"
        android:id="@+id/button4"
        android:layout_alignTop="@+id/button3"
        android:layout_toRightOf="@+id/button3"
        android:layout_toEndOf="@+id/button3" />
```



```

<SeekBar
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/seekBar"
    android:layout_alignLeft="@+id/textview"
    android:layout_alignStart="@+id/textview"
    android:layout_alignRight="@+id/textview"
    android:layout_alignEnd="@+id/textview"
    android:layout_above="@+id/button" />
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:src="@drawable/abc"
    android:layout_below="@+id/textview"
    android:layout_marginTop="16dp"
    android:layout_alignLeft="@+id/textView"
    android:layout_alignStart="@+id/textView"
    android:layout_alignRight="@+id/textView"
    android:layout_alignEnd="@+id/textView"
    android:contentDescription=""
    tools:ignore="ContentDescription" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceMedium"
    android:text="Medium Text"
    android:id="@+id/textView4"
    android:layout_above="@+id/seekBar"
    android:layout_toLeftOf="@+id/button4"
    android:layout_toStartOf="@+id/button4" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:text="Small Text"
    android:id="@+id/textView3"
    android:layout_above="@+id/seekBar"
    android:layout_alignRight="@+id/button4"
    android:layout_alignEnd="@+id/button4" />
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceSmall"
    android:text="Small Text"
    android:id="@+id/textView2"
    android:layout_alignTop="@+id/textView4"
    android:layout_toLeftOf="@+id/seekBar"
    android:layout_toStartOf="@+id/seekBar" />
</RelativeLayout>

```

Manifest

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ait.mediaplayer" >
    <application
        android:allowBackup="true"
        android:icon="@drawable/abc"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.example.ait.mediaplayer.MainActivity"
            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>

```

Strings

```

<resources>
<string name="app_name">Medioplayer</string>
<string name="back"><![CDATA[<]]></string>
<string name="rewind"><![CDATA[<<]]></string>
<string name="forward"><![CDATA[>>]]></string>
<string name="pause">| |</string>
</resources>

```

MainActivity.java

```

package com.example.ait.medioplayer;
import android.app.Activity;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.SeekBar;
import android.widget.TextView;
import android.widget.Toast;
import java.util.concurrent.TimeUnit;
public class MainActivity extends Activity {
private Button b1,b2,b3,b4;
private ImageView iv;
private MediaPlayer mediaPlayer;
private double startTime= 0;
private double finalTime= 0;
private Handler myHandler= new Handler();;
private int forwardTime= 5000;
private int backwardTime= 5000;
private SeekBar seekbar;
private TextView tx1,tx2,tx3;
public static int oneTimeOnly= 0;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
b1 = (Button) findViewById(R.id.button);
b2 = (Button) findViewById(R.id.button2);
b3 = (Button) findViewById(R.id.button3);
b4 = (Button) findViewById(R.id.button4);
iv = (ImageView) findViewById(R.id.imageView);
tx1 = (TextView) findViewById(R.id.textView2);
tx2 = (TextView) findViewById(R.id.textView3);
tx3 = (TextView) findViewById(R.id.textView4);
tx3.setText("Song.mp3");
mediaPlayer= MediaPlayer.create(this, R.raw.sss);
seekbar= (SeekBar) findViewById(R.id.seekBar);
seekbar.setClickable(false);
b2.setEnabled(false);
b3.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Toast.makeText(getApplicationContext(), "Playing sound", Toast.LENGTH_SHORT).show();
mediaPlayer.start();
finalTime= mediaPlayer.getDuration();

```

```

startTime= mediaPlayer.getCurrentPosition();

if (oneTimeOnly== 0) {
seekbar.setMax((int) finalTime);
oneTimeOnly= 1;
}

tx2.setText(String.format("%d min, %d sec",
TimeUnit.MILLISECONDS.toMinutes((long) finalTime),
TimeUnit.MILLISECONDS.toSeconds((long) finalTime) -
TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes((long)
finalTime)))
);

tx1.setText(String.format("%d min, %d sec",
TimeUnit.MILLISECONDS.toMinutes((long) startTime),
TimeUnit.MILLISECONDS.toSeconds((long) startTime) -
TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes((long)
startTime)))
);

seekbar.setProgress((int)startTime);
myHandler.postDelayed(UpdateSongTime,100);
b2.setEnabled(true);
b3.setEnabled(false);
}
});

b2.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Toast.makeText(getApplicationContext(), "Pausing sound",Toast.LENGTH_SHORT).show();
mediaPlayer.pause();
b2.setEnabled(false);
b3.setEnabled(true);
}
});

b1.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
inttemp = (int)startTime;

if((temp+forwardTime)<=finalTime){
startTime= startTime+ forwardTime;
mediaPlayer.seekTo((int) startTime);
Toast.makeText(getApplicationContext(),"You have Jumped forward 5 seconds",Toast.LENGTH_SHORT).show();
}
else{
Toast.makeText(getApplicationContext(),"Cannot jump forward 5 seconds",Toast.LENGTH_SHORT).show();
}
}
});

if((temp-backwardTime)>0){
startTime= startTime- backwardTime;
mediaPlayer.seekTo((int) startTime);
Toast.makeText(getApplicationContext(),"You have Jumped backward 5 seconds",Toast.LENGTH_SHORT).show();
}
else{
Toast.makeText(getApplicationContext(),"Cannot jump backward 5 seconds",Toast.LENGTH_SHORT).show();
}
}
});
}

```

```

private Runnable UpdateSongTime= new Runnable() {

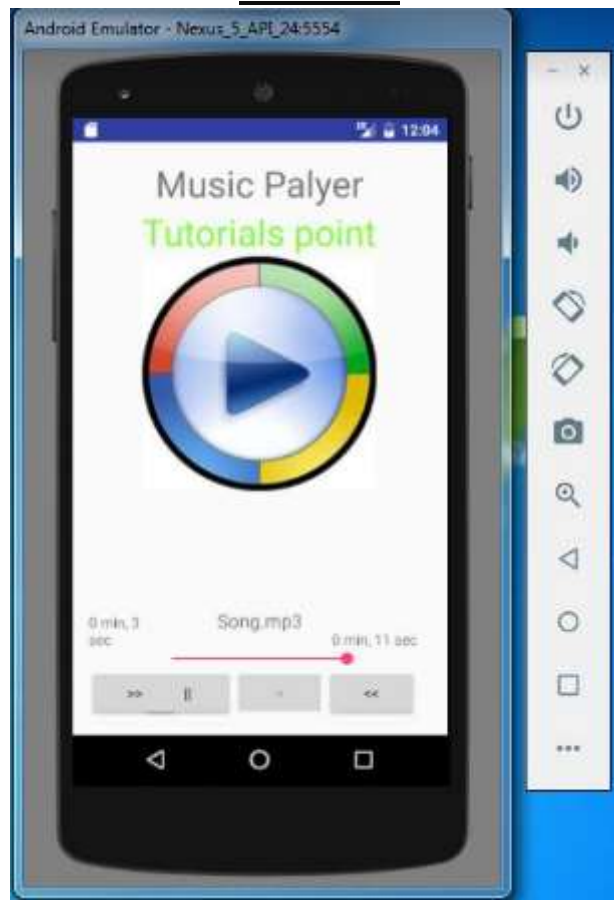
```

```

public void run() {
    startTime= mediaPlayer.getCurrentPosition();
    tx1.setText(String.format("%d min, %d sec", TimeUnit.MILLISECONDS.toMinutes((long) startTime),
        TimeUnit.MILLISECONDS.toSeconds((long) startTime) -
            TimeUnit.MINUTES.toSeconds(TimeUnit.MILLISECONDS.toMinutes((long) startTime)))
        );
    seekbar.setProgress((int)startTime);
    myHandler.postDelayed(this, 100);
    }
};
}

```

OUTPUT:



Playing Video

Manifest

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.ait.videoplayer">

    <uses-permission android:name="android.permission.INTERNET"></uses-permission>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme">

```

```

<activity android:name=".MainActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
</application>

</manifest>

```

Strings

```

<resources>
<string name="app_name">My Application</string>
<string name="hello_world">Hello world!</string>
<string name="action_settings">Settings</string>
</resources>

```

Activity_main.xml

```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context="com.example.ait.videoplayer.MainActivity">

<VideoView
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:id="@+id/videoView"
android:layout_gravity="center" />

```

```

</FrameLayout>

```

MainActivity.java

```

package com.example.ait.videoplayer;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.res.Configuration;
import android.media.MediaPlayer;
import android.media.MediaPlayer.OnPreparedListener;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;

```

```

import android.widget.MediaController;
import android.widget.VideoView;

public class MainActivity extends AppCompatActivity {
    private VideoView videoView;
    private int position = 0;
    private MediaController mediaController;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        videoView = (VideoView) findViewById(R.id.videoView);
        // Set the media controller buttons
        if (mediaController == null) {
            mediaController = new MediaController(MainActivity.this);
            // Set the videoView that acts as the anchor for the MediaController.
            mediaController.setAnchorView(videoView);

            // Set MediaController for VideoView
            videoView.setMediaController(mediaController);
        }
        try {
            // ID of video file.
            int id = this.getResources().getRawResIdByName("wildlife");
            videoView.setVideoURI(Uri.parse("android.resource://" + getPackageName() + "/" + id));

        } catch (Exception e) {
            Log.e("Error", e.getMessage());
            e.printStackTrace();
        }

        videoView.requestFocus();
        videoView.setOnPreparedListener(new OnPreparedListener() {
            public void onPrepared(MediaPlayer mediaPlayer) {
                videoView.seekTo(position);
                if (position == 0) {
                    videoView.start();
                }
                mediaPlayer.setOnVideoSizeChangedListener(new MediaPlayer.OnVideoSizeChangedListener() {
                    @Override
                    public void onVideoSizeChanged(MediaPlayer mp, int width, int height) {
                        mediaController.setAnchorView(videoView);
                    }
                });
            }
        });
    }

    // Find ID corresponding to the name of the resource (in the directory raw).
    public int getRawResIdByName(String resName) {
        String pkgName = this.getPackageName();
        // Return 0 if not found.

```

```

intresID = this.getResources().getIdentifier(resName, "raw", pkgName);
Log.i("AndroidVideoView", "Res Name: " + resName + "==> Res ID = " + resID);
return resID;
}
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    super.onSaveInstanceState(savedInstanceState);
    // Store current position.
    savedInstanceState.putInt("CurrentPosition", videoView.getCurrentPosition());
    videoView.pause();
}
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    super.onRestoreInstanceState(savedInstanceState);
    // Get saved position.
    position = savedInstanceState.getInt("CurrentPosition");
    videoView.seekTo(position);
}

```

🚦 OUTPUT



3) Stage v (verify)

Home Activities:

Activity 1:

Make an application that takes the video from the sdcard and play it and save it with another name in the same file and assets folder.

Statement Purpose:

To develop a Android Application that writes data to the SD Card.

Activity Outcomes:

Students can perform IO operation and can manipulate data within SDcard.

Instructor Note:

Student must familiar with the IO streams in java.io package and to work with.

1) Stage J (Journey)

Introduction

Student must familiar with the IO streams in java.io package and to work with.

2) Stage a1 (apply)

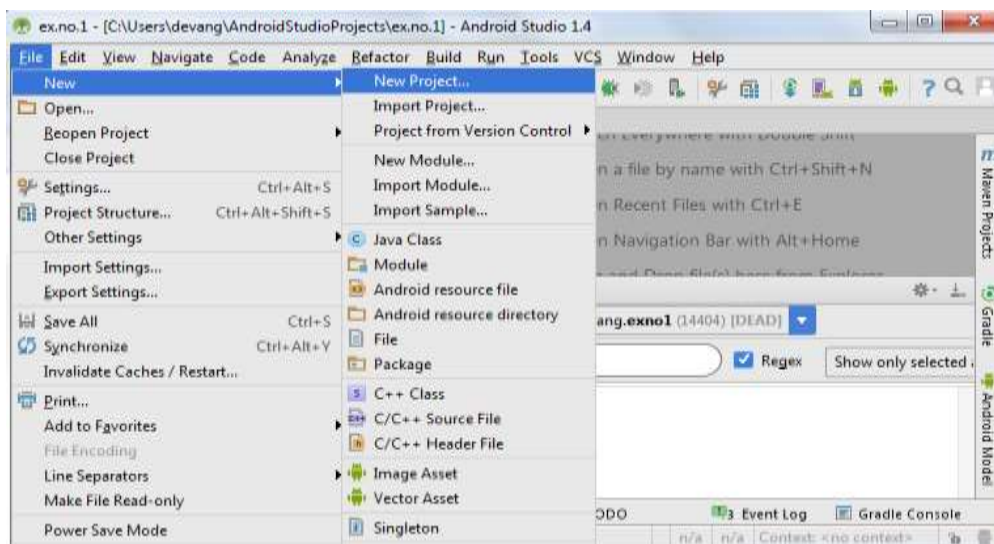
Lab Activities:

Activity 1:

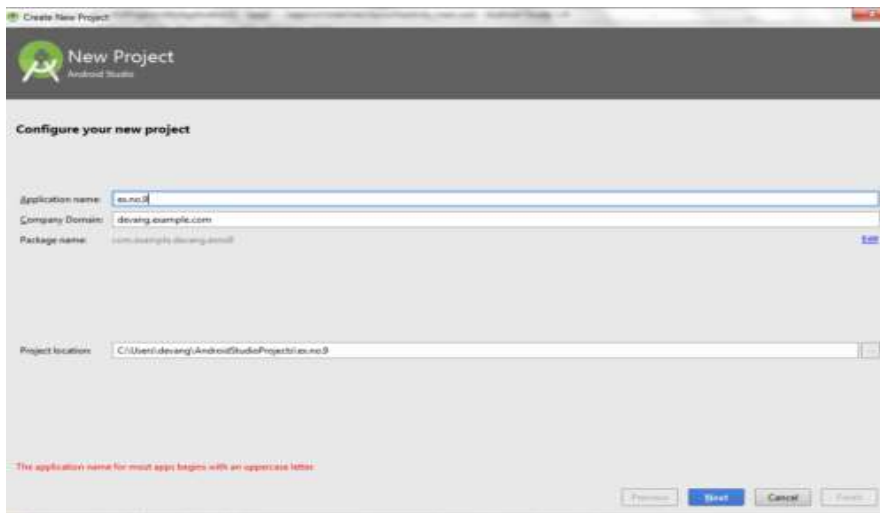
Procedure:

Creating a New project:

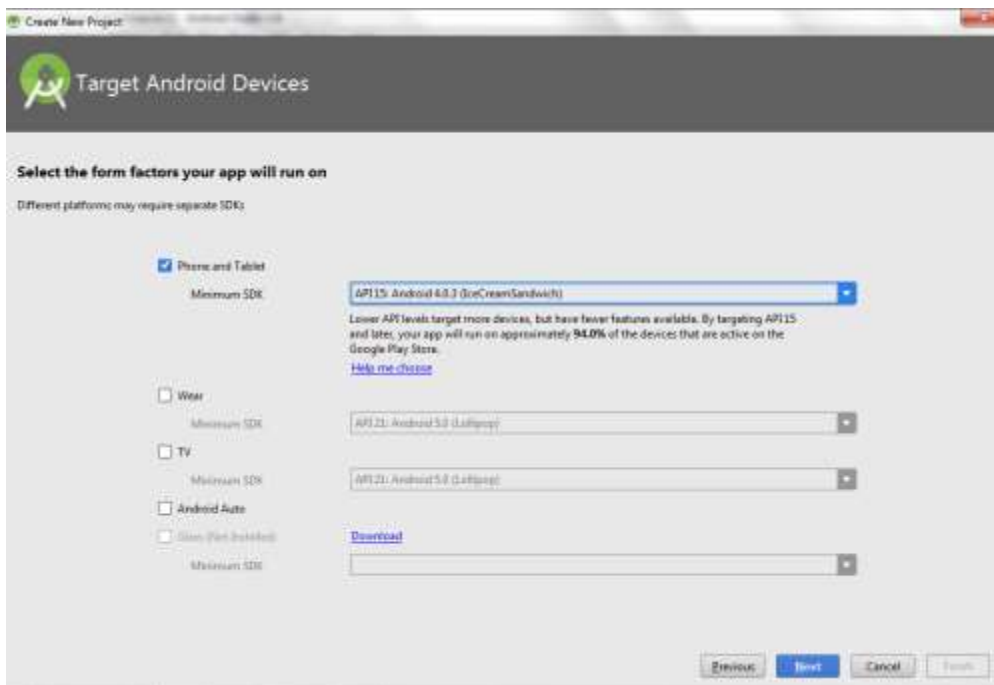
- Open Android Studio and then click on **File -> New -> New project**.



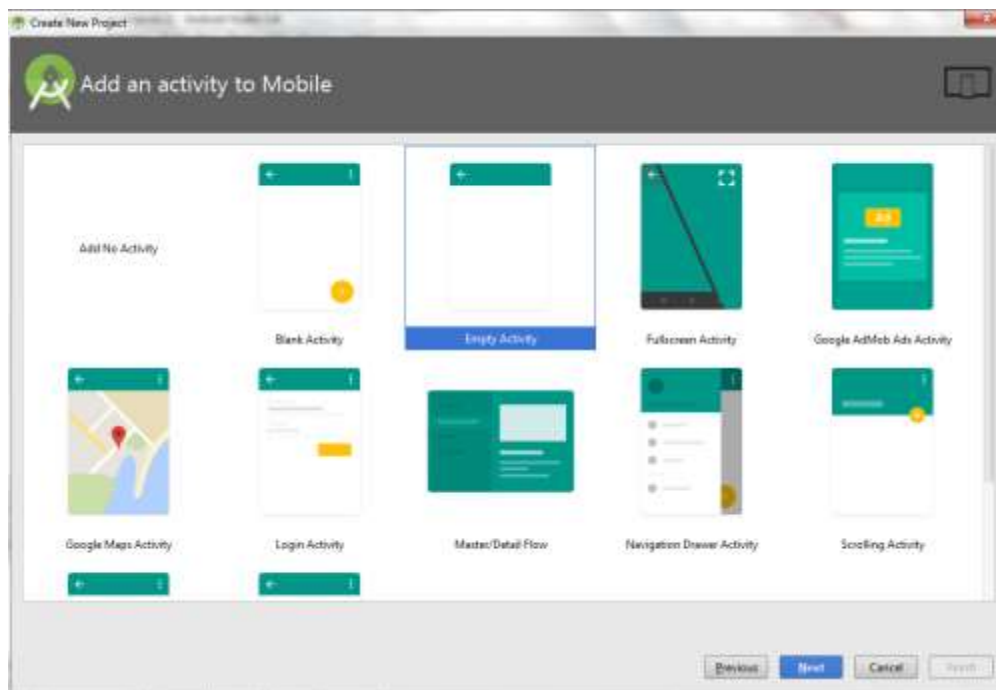
- Then type the Application name as **"ex.no.9"** and click **Next**.



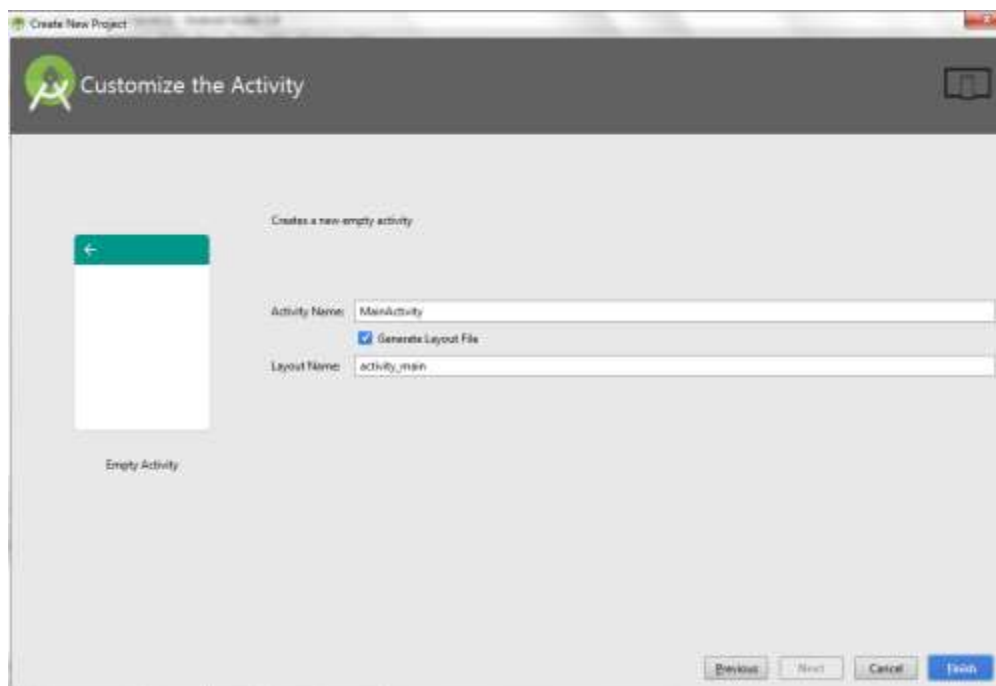
- Then select the **Minimum SDK** as shown below and click **Next**.



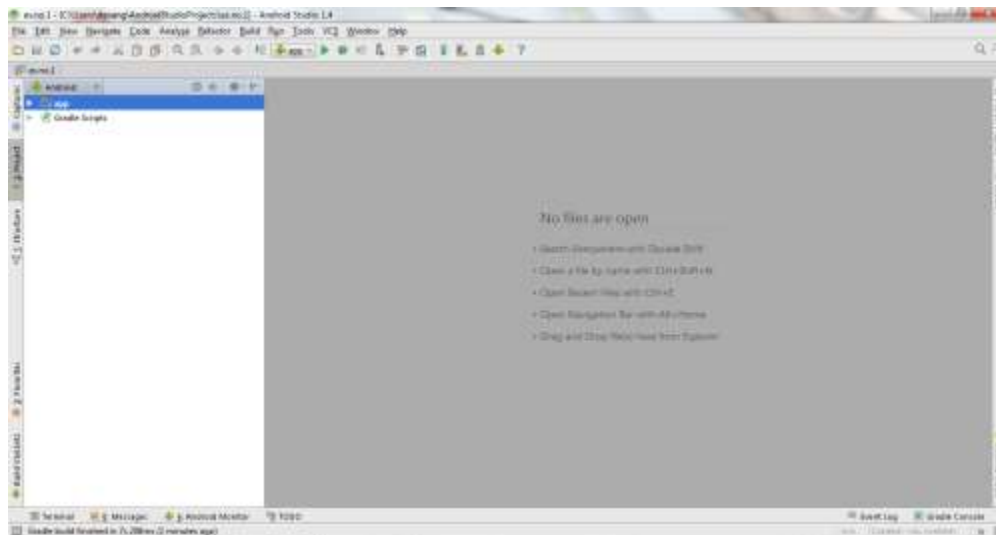
- Then select the **Empty Activity** and click **Next**.



- Finally click **Finish**.

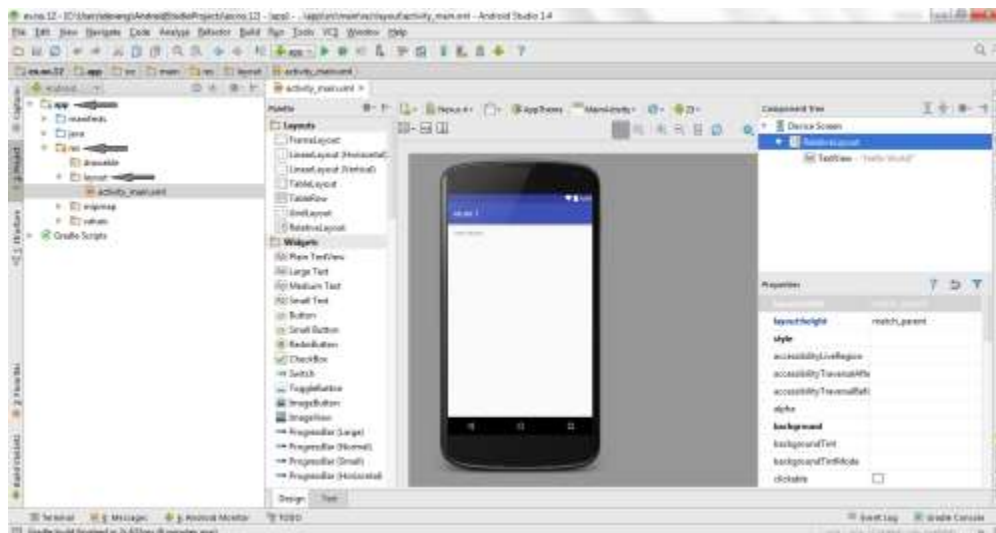


- It will take some time to build and load the project.
- After completion it will look as given below.

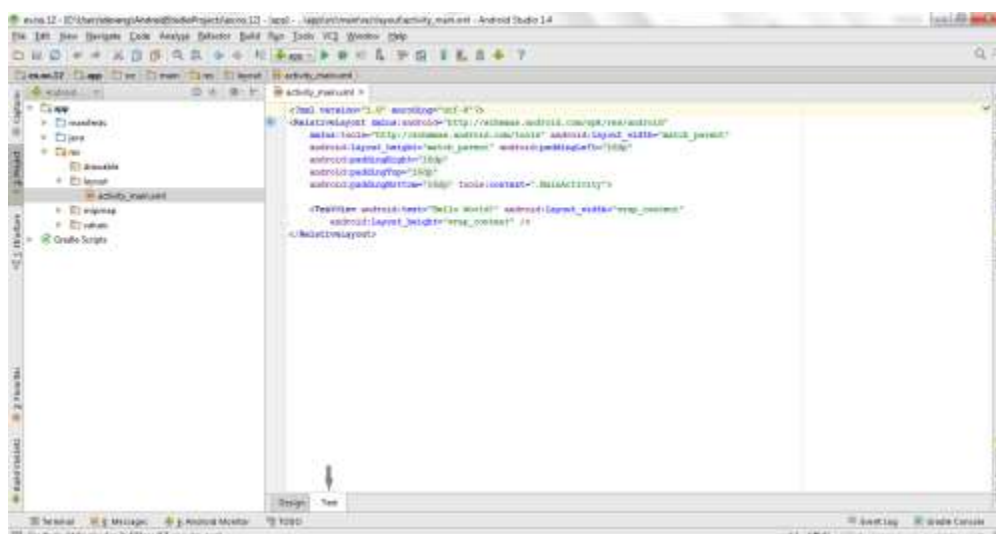


Designing layout for the Android Application:

- Click on **app** -> **res** -> **layout** -> **activity_main.xml**.



- Now click on **Text** as shown below.



- Then delete the code which is there and type the code as given below.

Code for Activity_main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="20dp"
    android:orientation="vertical">

    <EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:singleLine="true"
        android:textSize="30dp" />

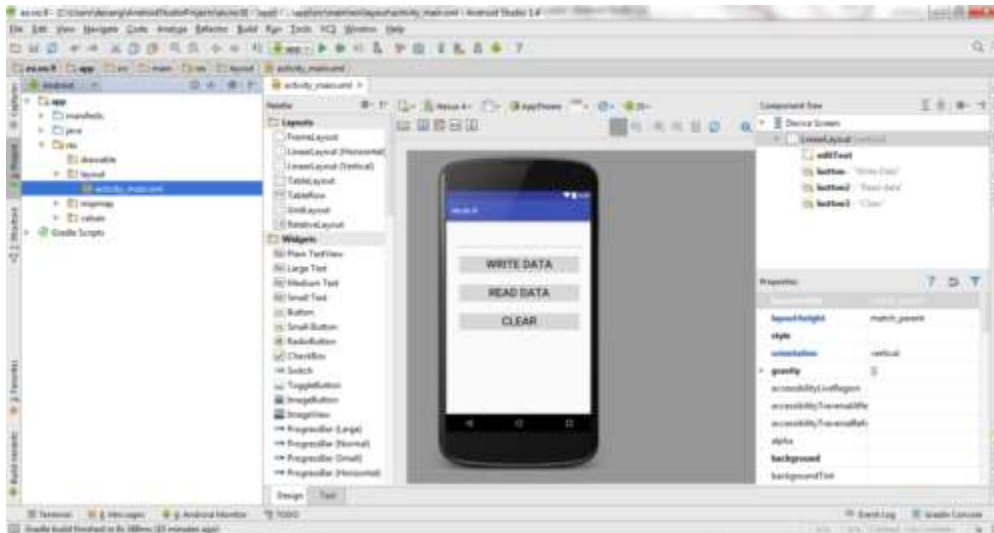
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="Write Data"
        android:textSize="30dp" />

    <Button
        android:id="@+id/button2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="Read data"
        android:textSize="30dp" />

    <Button
        android:id="@+id/button3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="Clear"
        android:textSize="30dp" />

</LinearLayout>
```

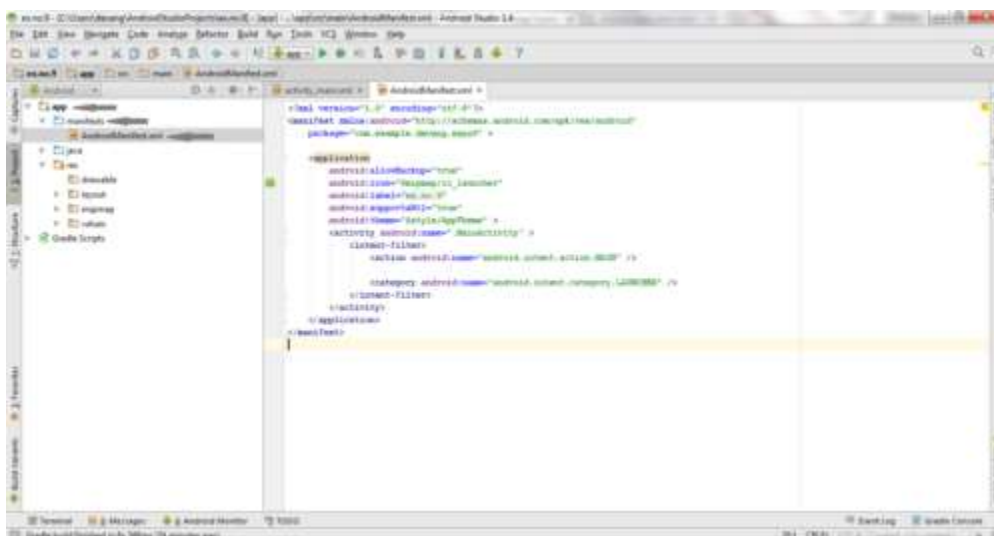
- Now click on **Design** and your application will look as given below.



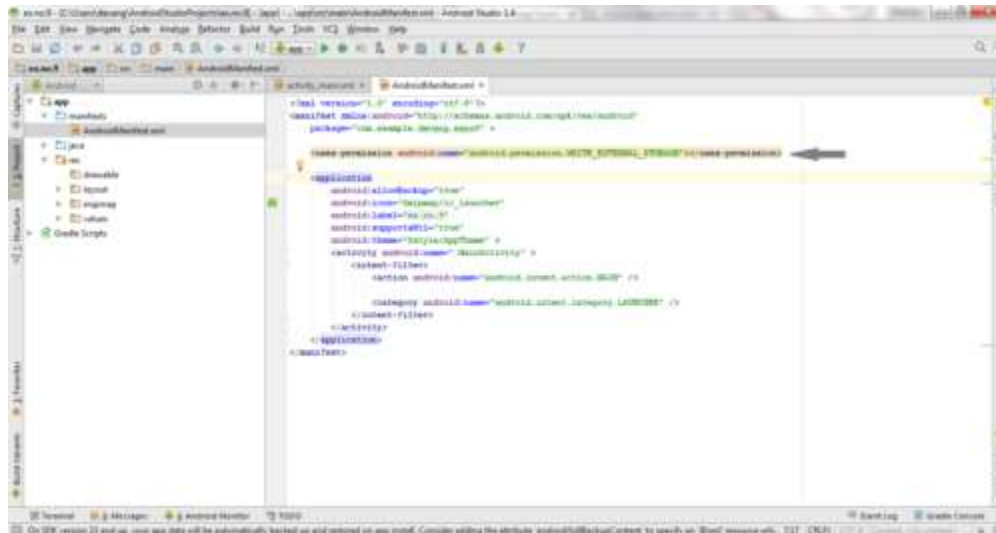
- So now the designing part is completed.

Adding permissions in Manifest for the Android Application:

- Click on **app** -> **manifests** -> **AndroidManifest.xml**



- Now include the **WRITE_EXTERNAL_STORAGE** permissions in the AndroidManifest.xml file as shown below



Code for AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.exno9" >

    <uses-permission
        android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-permission>

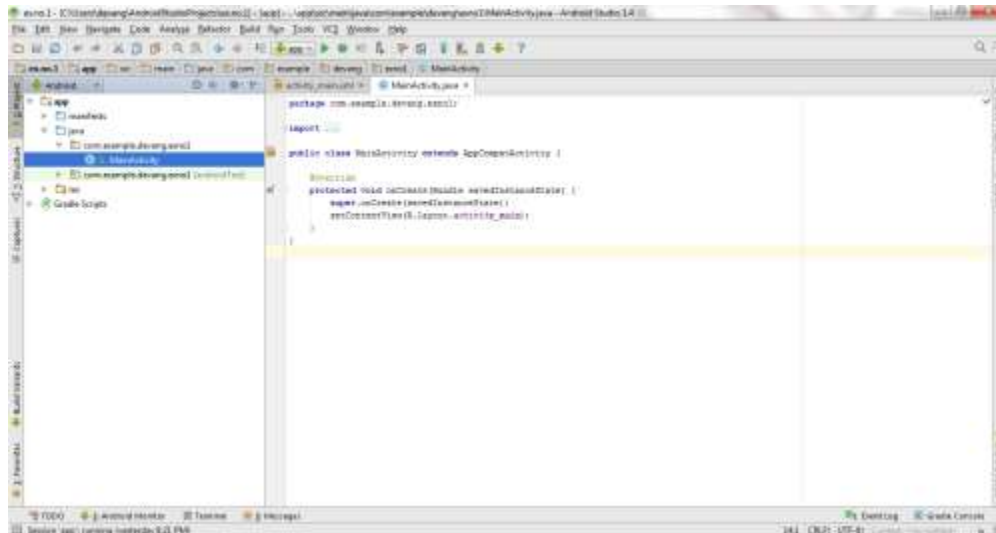
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

- So now the Permissions are added in the Manifest.

Java Coding for the Android Application:

- Click on **app** -> **java** -> **com.example.exno9** -> **MainActivity**.



- Then delete the code which is there and type the code as given below.

Code for MainActivity.java:

```
package com.example.exno9;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStreamReader;

public class MainActivity extends AppCompatActivity
{
    EditText e1;
    Button write, read, clear;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        e1= (EditText) findViewById(R.id.editText);
        write= (Button) findViewById(R.id.button);
        read= (Button) findViewById(R.id.button2);
        clear= (Button) findViewById(R.id.button3);

        write.setOnClickListener(new View.OnClickListener()
        {
            @Override
            public void onClick(View v)
            {
                String message=e1.getText().toString();
                try
                {
                    File f=new File("/sdcard/myfile.txt");
```

```

        f.createNewFile();
        FileOutputStream fout=new FileOutputStream(f);
        fout.write(message.getBytes());
        fout.close();
        Toast.makeText(getApplicationContext(),"Data Written in
SDCARD",Toast.LENGTH_LONG).show();
    }
    catch (Exception e)
    {
        Toast.makeText(getApplicationContext(),e.getMessage(),Toast.LE
NGTH_LONG).show();
    }
}
});

read.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        String message;
        String buf = "";
        try
        {
            File f = new File("/sdcard/myfile.txt");
            FileInputStream fin = new FileInputStream(f);
            BufferedReader br = new BufferedReader(new
InputStreamReader(fin));
            while ((message = br.readLine()) != null)
            {
                buf += message;
            }
            e1.setText(buf);
            br.close();
            fin.close();
            Toast.makeText(getApplicationContext(),"Data Recived from
SDCARD",Toast.LENGTH_LONG).show();
        }
        catch (Exception e)
        {
            Toast.makeText(getApplicationContext(), e.getMessage(),
Toast.LENGTH_LONG).show();
        }
    }
});

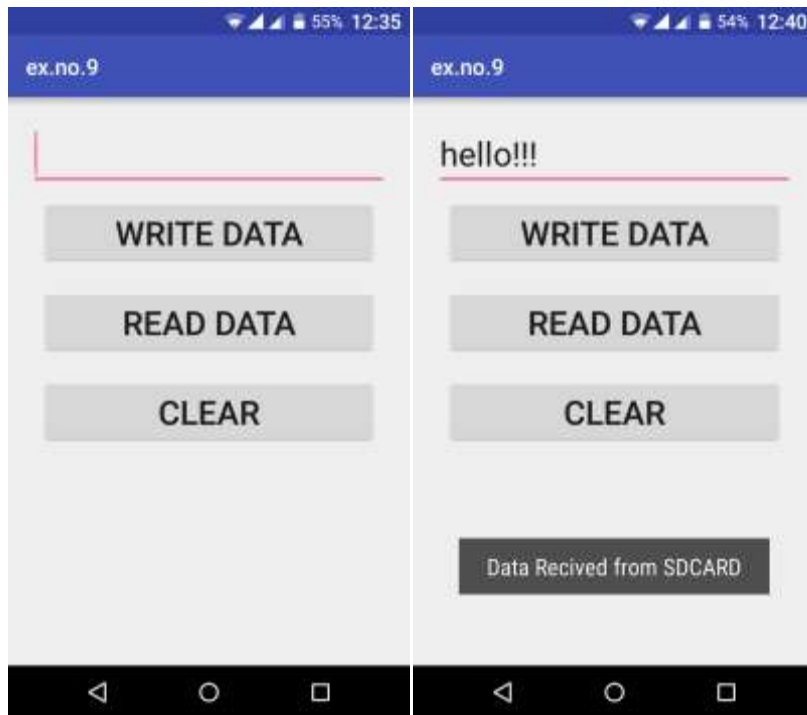
clear.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View v)
    {
        e1.setText("");
    }
});
}
}

```

- So now the Coding part is also completed.
- Now run the application to see the output.

Output:





Result:

Thus Android Application that writes data to the SD Card is developed and executed successfully.

3) Stage v (verify)

Home Activities:

Activity 1:

take 10 file (5 audio and 5 text files) and perform read write operation while place them in the sd card.

Statement Purpose:

Activity Outcomes:

To develop a Android Application that makes use of RSS Feed.

Instructor Note:

Student must read the topic of XML objects and make sure to familiar with the RSS file making and how to read them.

1) Stage J (Journey)

Introduction

The National Weather Service XML format is...curiously structured, relying heavily on sequential position in lists versus the more object-oriented style you find in formats like RSS or Atom. That being said, we can take a few liberties and simplify the parsing somewhat, taking advantage of the fact that the elements we want (start-valid-time for the forecast time, value for the temperature, and icon-link for the icon URL) are all unique within the document. The HTML comes in as an InputStream and is fed into the DOM parser. From there, we scan for the start-valid-time elements and populate a set of Forecast models using those start times. Then, we find the temperature value elements and icon-link URLs and fill those in to the Forecast objects.

2) Stage a1 (apply)

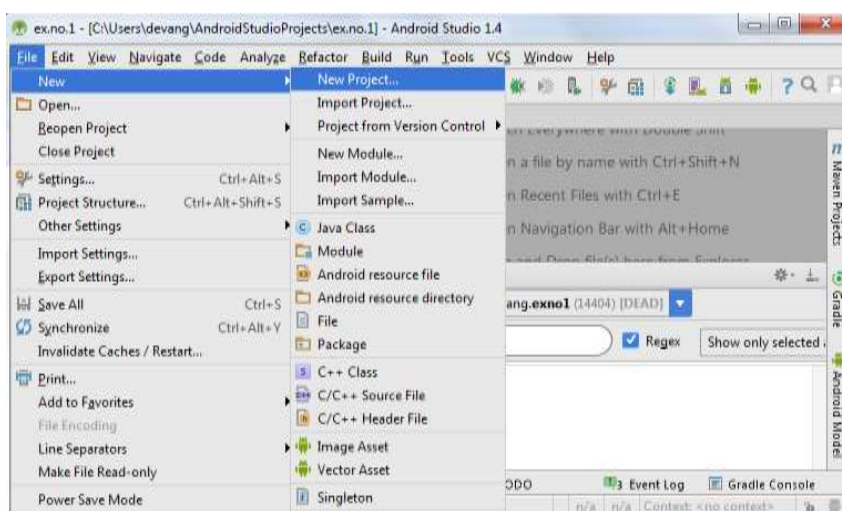
Lab Activities:

Activity 1:

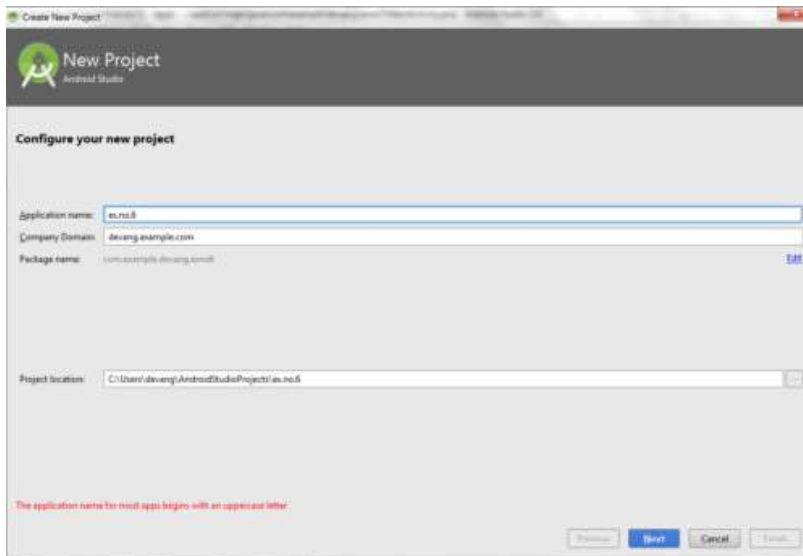
Procedure:

Creating a New project:

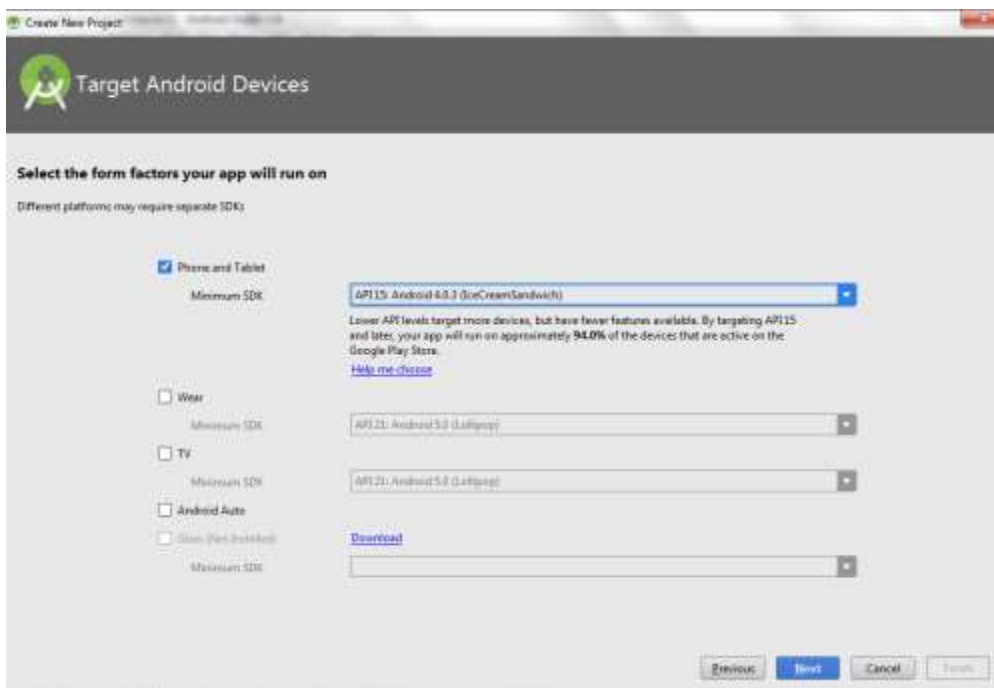
- Open Android Studio and then click on **File -> New -> New project.**



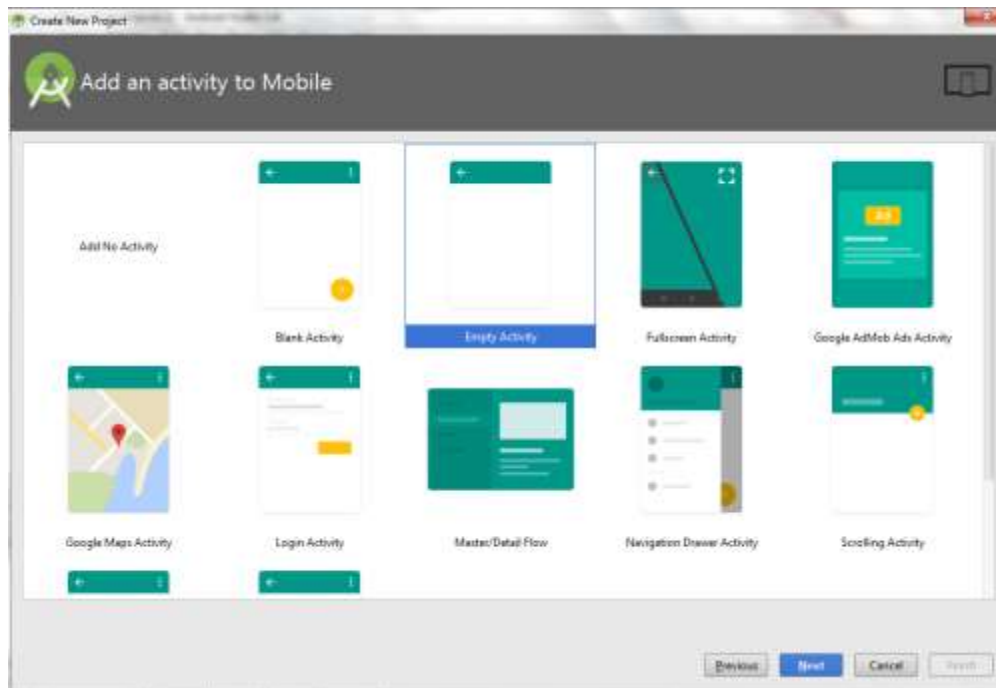
- Then type the Application name as “**ex.no.6**” and click **Next**.



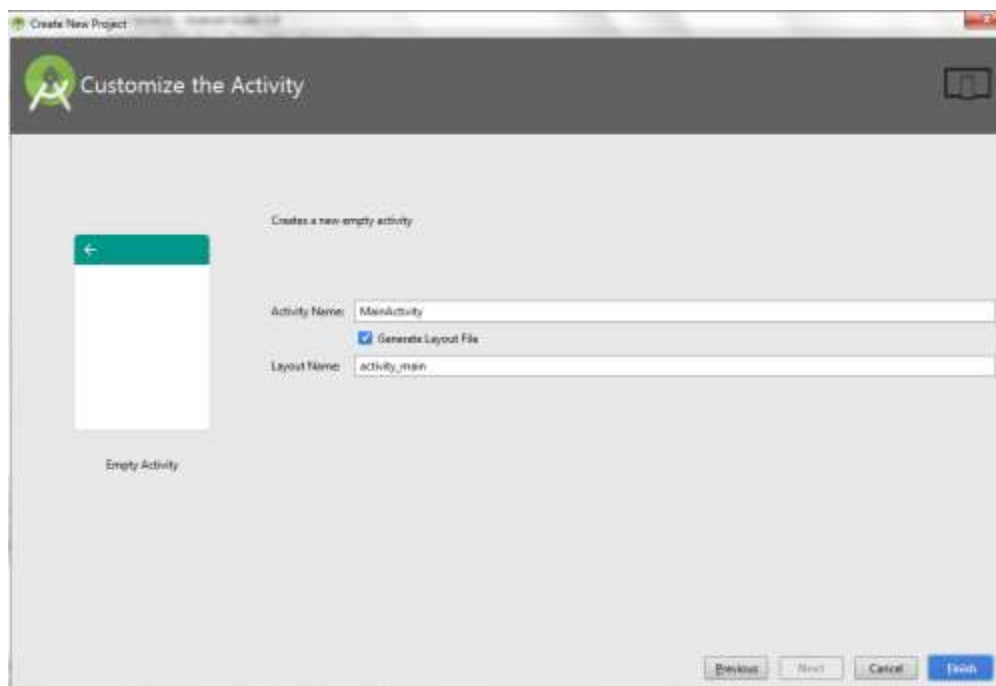
- Then select the **Minimum SDK** as shown below and click **Next**.



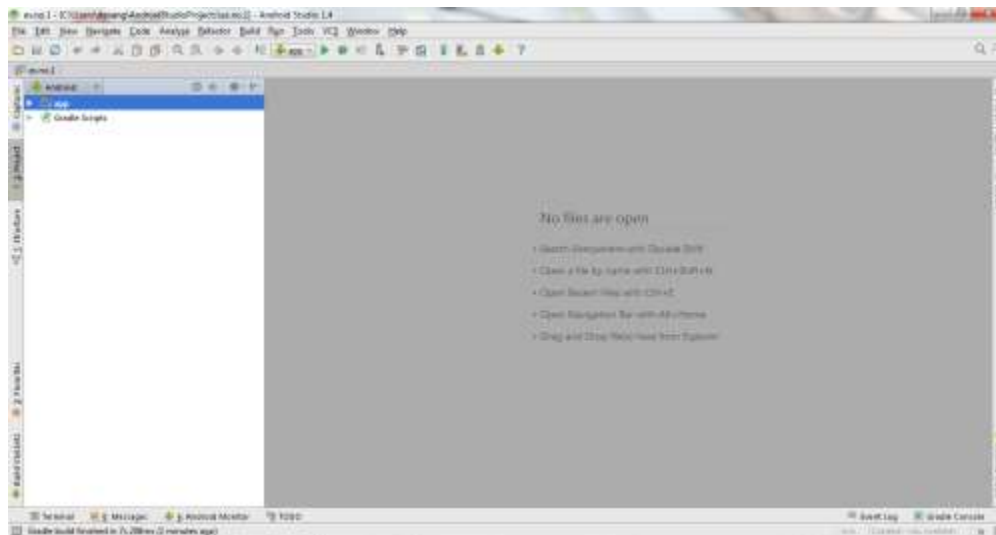
- Then select the **Empty Activity** and click **Next**.



- Finally click **Finish**.

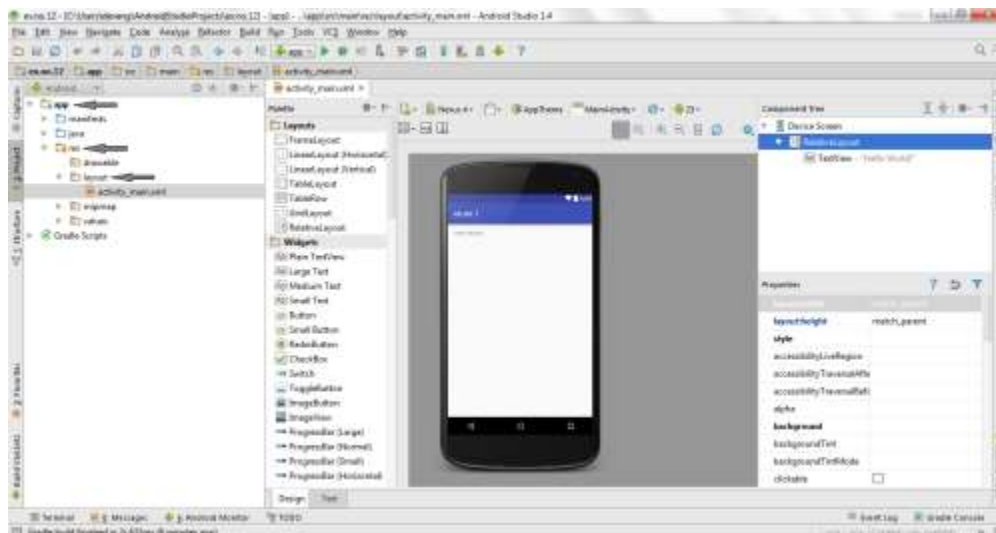


- It will take some time to build and load the project.
- After completion it will look as given below.

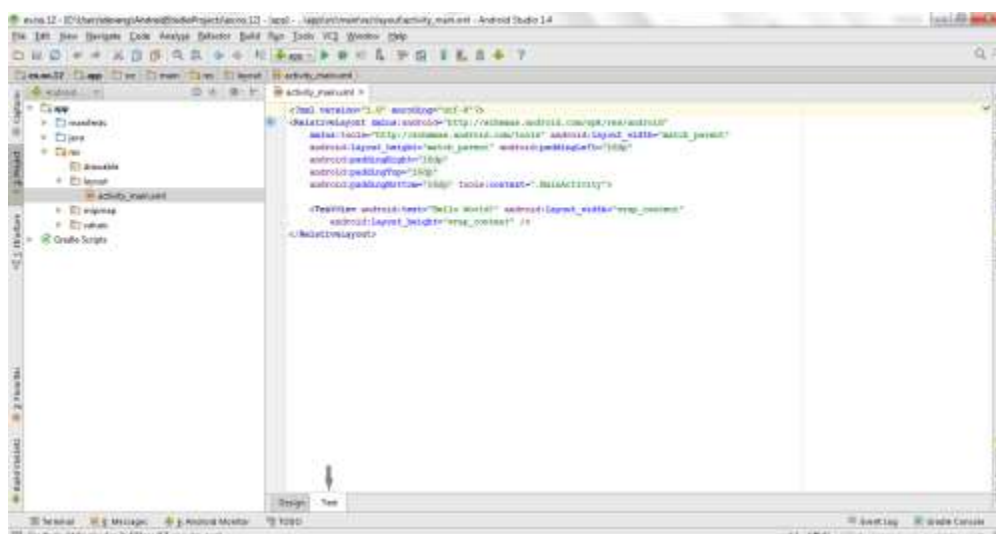


Designing layout for the Android Application:

- Click on **app** -> **res** -> **layout** -> **activity_main.xml**



- Now click on **Text** as shown below.



- Then delete the code which is there and type the code as given below.

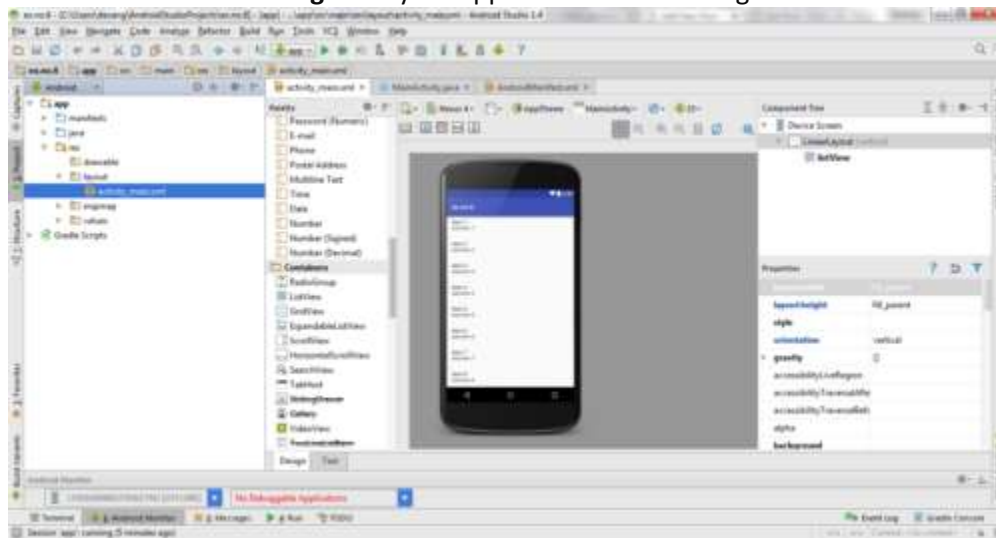
Code for Activity_main.xml:

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

    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

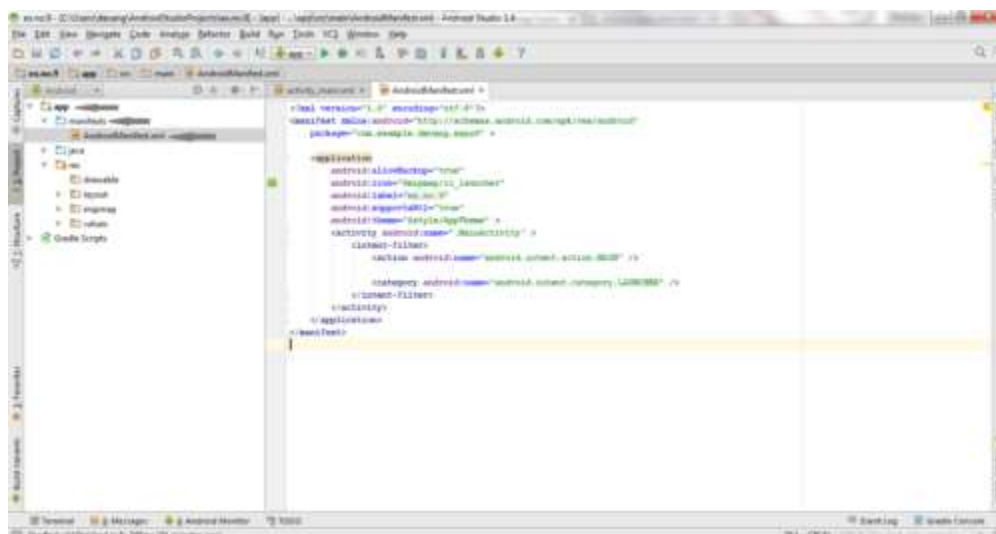
- Now click on **Design** and your application will look as given below.



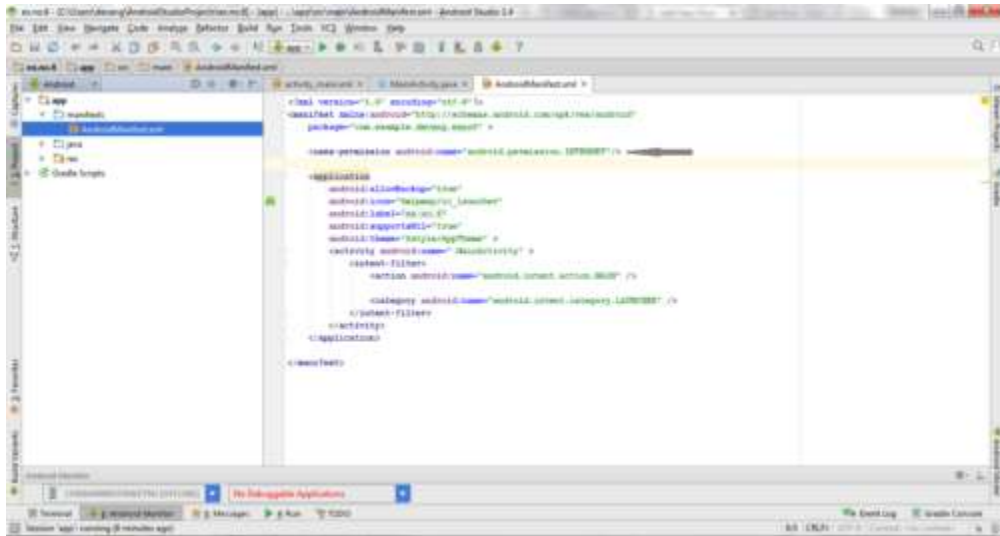
- So now the designing part is completed.

Adding permissions in Manifest for the Android Application:

- Click on **app -> manifests -> AndroidManifest.xml**



- Now include the **INTERNET** permissions in the AndroidManifest.xml file as shown below



Code for AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.exno6" >

    <uses-permission android:name="android.permission.INTERNET"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportRtl="true"
        android:theme="@style/AppTheme" >
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

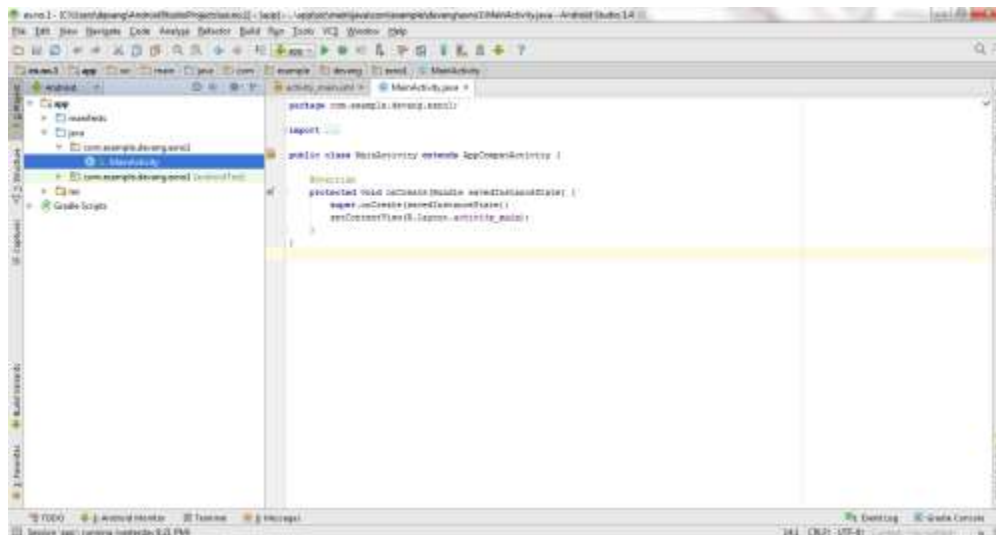
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

- So now the Permissions are added in the Manifest.

Java Coding for the Android Application:

- Click on **app** -> **java** -> **com.example.exno6** -> **MainActivity**.



- Then delete the code which is there and type the code as given below.

Code for MainActivity.java:

```
package com.example.exno6;

import android.app.ListActivity;
import android.content.Intent;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListView;
import org.xmlpull.v1.XmlPullParser;
import org.xmlpull.v1.XmlPullParserException;
import org.xmlpull.v1.XmlPullParserFactory;
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;

public class MainActivity extends ListActivity
{
    List headlines;
    List links;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        new MyAsyncTask().execute();
    }

    class MyAsyncTask extends AsyncTask<Object, Void, ArrayAdapter>
    {
        @Override
        protected ArrayAdapter doInBackground(Object[] params)
        {
            headlines = new ArrayList();
```

```

        links = new ArrayList();
        try
        {
            URL url = new URL("http://www.codingconnect.net/feed");
            XmlPullParserFactory factory =
XmlPullParserFactory.newInstance();
            factory.setNamespaceAware(false);
            XmlPullParser xpp = factory.newPullParser();

            // We will get the XML from an input stream
            xpp.setInput(getInputStream(url), "UTF_8");
            boolean insideItem = false;

            // Returns the type of current event: START_TAG, END_TAG,
etc..

            int eventType = xpp.getEventType();
            while (eventType != XmlPullParser.END_DOCUMENT)
            {
                if (eventType == XmlPullParser.START_TAG)
                {
                    if (xpp.getName().equalsIgnoreCase("item"))
                    {
                        insideItem = true;
                    }
                    else if (xpp.getName().equalsIgnoreCase("title"))
                    {
                        if (insideItem)
                            headlines.add(xpp.nextText()); //extract
the headline
                    }
                    else if (xpp.getName().equalsIgnoreCase("link"))
                    {
                        if (insideItem)
                            links.add(xpp.nextText()); //extract the
link of article
                    }
                }
                else if (eventType == XmlPullParser.END_TAG &&
xpp.getName().equalsIgnoreCase("item"))
                {
                    insideItem = false;
                }
                eventType = xpp.next(); //move to next element
            }

        }
        catch (MalformedURLException e)
        {
            e.printStackTrace();
        }
        catch (XmlPullParserException e)
        {
            e.printStackTrace();
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
        return null;
    }
    protected void onPostExecute(ArrayAdapter adapter)
    {

```

```

        adapter = new ArrayAdapter(MainActivity.this,
android.R.layout.simple_list_item_1, headlines);
        setListAdapter(adapter);
    }

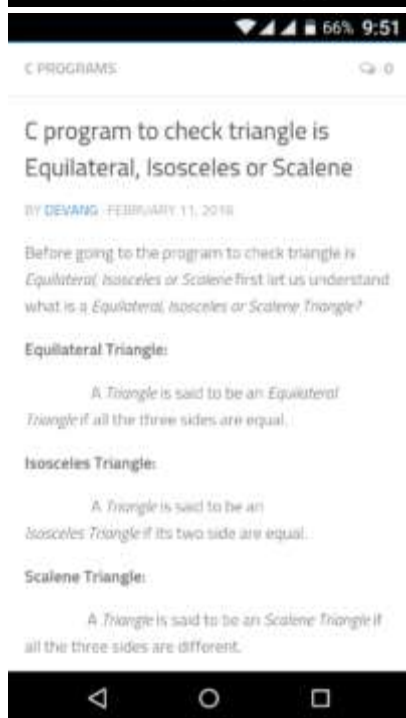
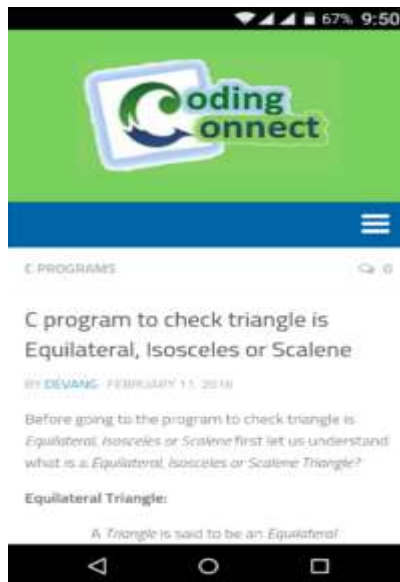
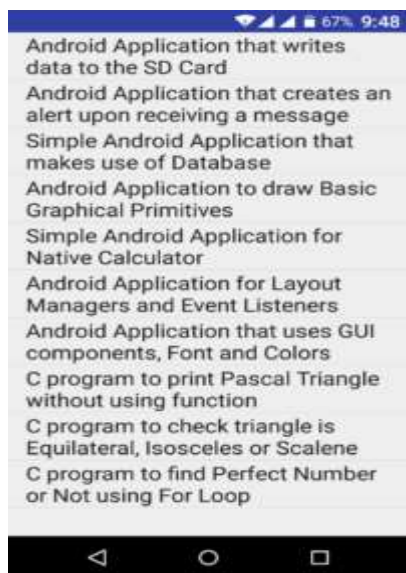
    @Override
    protected void onItemClick(ListView l, View v, int position, long id)
    {
        Uri uri = Uri.parse((links.get(position)).toString());
        Intent intent = new Intent(Intent.ACTION_VIEW, uri);
        startActivity(intent);
    }

    public InputStream getInputStream(URL url)
    {
        try
        {
            return url.openConnection().getInputStream();
        }
        catch (IOException e)
        {
            return null;
        }
    }
}

```

- So now the Coding part is also completed.
- Now run the application to see the output.

Output:



Result:

Thus Android Application that makes use of RSS Feed is developed and executed successfully.

3) Stage v (verify)

Home Activities:

Activity 1:

Read RSS news from the BBC news page and print them on the text view/listview and also save it in the database.

Statement Purpose:

To learn the GPS and MAPS systems using android to get familiar with the tracking apps making.

Activity Outcomes:

At the end of this lab you will be expected to know:

- How to incorporate Google Maps into an application.
- How to register for and receive GPS location information.
- How to draw graphics on the screen using the Canvas class.
- How to create Google Maps Overlays.
- How to use the Camera.
- How to write data to the SD card.
- How to create and delete private application files.
- How to launch and receive results from Activities.

Instructor Note:

Activities

For this lab you will be working with a brand new application, completely independent of the previous labs. Over the course of the lab, you will be iteratively refining and adding functionality to the WalkAbout app. With each iteration you will be improving upon the previous iteration's functionality. You'll start by setting up and familiarizing yourself with the Eclipse project. You will then register for a Google Maps API key and begin incrementally developing the main map-viewing Activity. These first few exercises will have you display a map and the user's current position. Next, you will add functionality to record the user's GPS location by registering for and receiving data from what is known as the GPS Location Provider. After that, you will implement a Camera activity for taking pictures and saving them to the SD-Card. In the final section, you will save a user's GPS path to a file that is private to the application, and allow the application to restore itself from the file as well.

IMPORTANT:

You will be given a Skeleton Project to work with. This project contains all of the java and resource files you will need to complete the lab. Some method stubs, member variables, and resource values and ids have been added as well. It is important that you not change the names of these methods, variables, and resource values and ids. These are given to you because there are unit tests included in this project as well that depend on these items being declared exactly as they are. These unit tests will be used to evaluate the correctness of your lab. You have complete access to these test cases during development, which gives you the ability to run these tests yourself. In fact, you are encouraged to run these tests to ensure that your application is functioning properly.

Setting Up...

Creating the Project

To begin, you will need to download and extract the skeleton project for the WalkAbout application.

[Click Here](#) to download the skeleton project.

Extract the project, making sure to preserve the directory structure.
Take note of the path to the root folder of the skeleton project.

Next you will need to setup an Android project for this app. Since the skeleton project was created in Eclipse, the easiest thing is to import this project into Eclipse

- Select **File -> Import**.
- In the Import Wizard, expand **General** and select **Existing Projects into Workspace**. Click **Next**.
- In the **Import Project** wizard, click **select root directory** and click **Browse**. Select the root directory of the skeleton project that you extracted. Click **Open** and then **Finish**.
- Click on the project name in the Package Explorer. Select **File -> Rename** and change the name of your project to lab5<userid> where <userid> is your user id (e.g. jsmith).

Next you will need to set the correct build target. In the **Android SDK and AVD Manager** (available from **Windows** menu in Eclipse), ensure that you have Google APIs for API 4. If you don't click on Available Packages (see the lecture notes or [here](#) for further information).

Once you have the Google APIs add-on install, you need to use it as the target for your project. Right-click on the lab5<userid> project, then select **Properties**. Select **Android** and select Google APIs for 1.6 (API 4) as the Project Build Target.

Familiarize Yourself with the Project

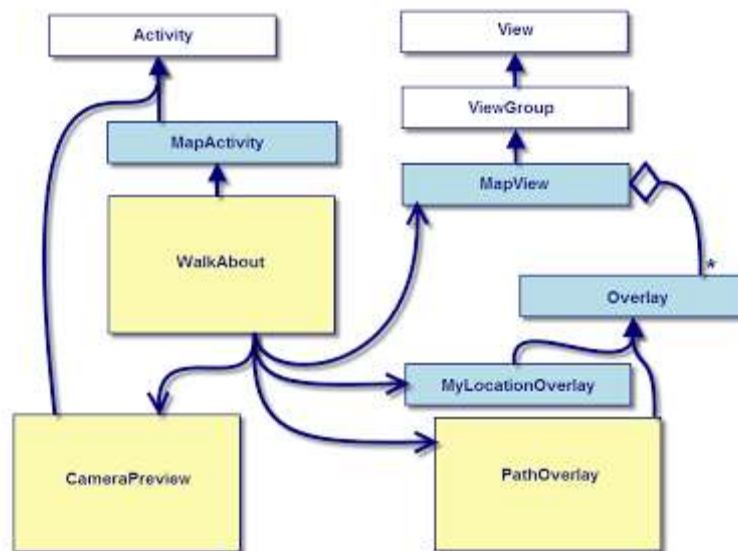
The project contains three java class files and a single XML layout file which you will have to implement. **WalkAbout.java** will contain the definition for the main WalkAbout Activity class. This is the class that will display the map and the user's recorded path. The WalkAbout class makes use of a very simple XML layout file called **map_layout.xml** which you will have to fill in.

From the WalkAbout Activity you will be able to launch the CameraPreview Activity, which will be defined in the **CameraPreview.java** file. The CameraPreview Activity class displays a live camera preview on the screen and will allow the user to capture and save a picture to the SD-Card. This file was borrowed from the Google API's Demo

application and the logic to display the camera preview is already implemented for you. You will have to fill in the rest.

The WalkAbout Activity class will rely on the PathOverlay class to draw the user's traveled path on its map. You will implement the logic needed for drawing the path in the **PathOverlay.java** file.

A general class diagram for the project is depicted below. Classes you will be implementing are colored yellow, the Google Maps API classes that you will be working with are colored in blue, and the standard Android classes you should be familiar with are colored in white.



Using Google Maps

In this section of the lab you will be working extensively with the Google Maps package. The Google Maps package allows you to include and manipulate Maps in your Android Applications. The general strategy for displaying a map in your application is to have an entire Activity dedicated to viewing a map. This activity must *extend* the `com.google.android.maps.MapActivity` class, which takes care of all the intricacies involved in setting up and tearing down the services required to support displaying a map.

The actual map that gets displayed is an instance of the `com.google.android.maps.MapView` class, which extends the standard Android `ViewGroup` class. This class encapsulates all the gesture logic necessary for handling panning, zooming, and touching objects on the map. The map is just a Google map and can be displayed in three different modes: satellite, street, and traffic. In order to use this class you will need a Google Maps API Key. We will go over how to do this later.

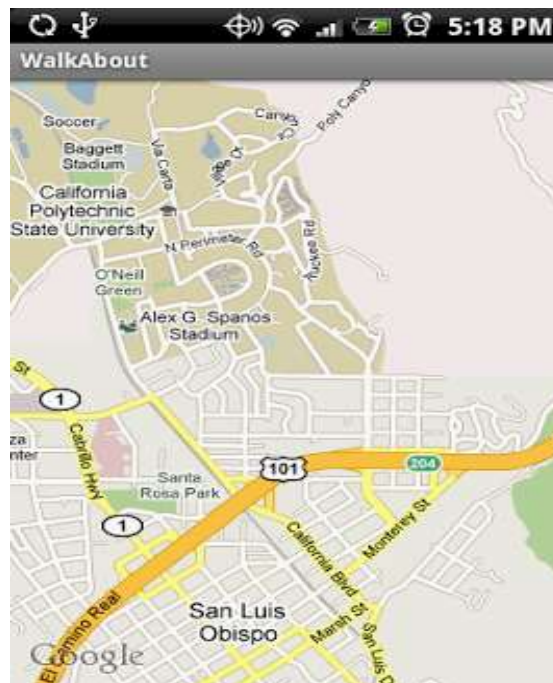
By making use of the `com.google.android.maps.Overlay` class you can *overlay* interesting data on top of your map. By either extending this base class or using other overlay

classes you can add another interaction layer onto your map.

The `com.google.android.maps.MyLocationOverlay` class is one such example; it can draw a beacon on the map to display the device's current position, and it can draw a compass on the map to display the device's current heading.

Displaying a Map

You will begin by implementing the functionality necessary to display a full-screen map in the `WalkAbout` Activity class. The layout for the `WalkAbout` Activity should be specified in the `res/layout/map_layout.xml` file, which you will have to fill in. This will require you to use the `com.google.android.maps.MapView` class. In order to do this, you will have to register the debug keystore (that Eclipse uses to run your applications) with Google in order to receive a Maps API Key. When finished, your application should appear as depicted in the figure below:



For more information and examples on working with the Google Maps Package see the documentation site and/or visit the Android Developer Tutorial on the Google Maps Package.

Get a Google Maps API Key.

In order to use the Google Maps API's and classes you will have to register the keystore you use to sign your application with the Google Maps Service. Once your keystore is registered, you will be provided with a Google Maps API key that can be used with any application signed by your keystore.

Every distribution of the Android Development Toolkit comes with a debug keystore that is used to sign your application when it is launched and run from Eclipse. This is how you are able to run your application on a device or emulator without signing it yourself. For the purposes of this lab, you need only to register your debug keystore. If

you end up releasing an application to the market you will need to register an actual keystore.

- Retrieve the MD5 Fingerprint of your debug.keystore file:
 - Follow the instructions [here](#)
- Register the Fingerprint with Google Maps Service:
 - Follow the instructions [here](#)
- Save the API Key as a resource string in your Application:
 - Copy and Paste the API Key into the res/values/strings.xml file in between the **mapApiKey** string tags:

```
<string  
name="mapApiKey">INSERT_YOUR_API_KEY_HERE</st  
ring>
```

- *Storing the API Key as a resource allows you to more easily change it in the future.*
- Your application must declare that it uses the Google Maps Library in its manifest. Do so by adding the the following **<uses-library ... />** line to your AndroidManifest.xml file. This line should be nested inside of the **<application></application>** tags:

```
<application ...>  
    <uses-library android:name="com.google.android.maps" />  
</application>
```

- Your application must use an internet connection to retrieve map data so it must also declare that it uses the Internet in its manifest. Do so by adding the following **<uses-permission ... />** line to your AndroidManifest.xml file. This line should be nested inside of the **<manifest></manifest>** tags:

```
<manifest ...>  
    <uses-permission  
android:name="android.permission.INTERNET"/>  
</manifest>
```

Fill in the MapView XML Layout File

The main WalkAbout MapActivity class uses a very simple layout consisting of a root ViewGroup that contains only a single MapView element. Implement this layout by filling in the map_layout.xml file:

- Use whatever root ViewGroup you like to contain the MapView element.
- When adding the MapView element, you need to use the fully qualified class name:

```
<com.google.android.maps.MapView ... />
```

- IMPORTANT: You must give the MapView element an id of "**m_vwMap**".
- The MapView should fill the entire contents of the screen.
- When using a MapView element you must give it your API Key. You do this by inserting a special **android:apiKey** attribute into the MapView element declaration.
 - Set the attribute equal to the string resource that contains the API Key.

```
android:apiKey="@string/mapApiKey"
```

- *This is how you reference resources from within a resource file.*
- Set the MapView to clickable by adding an **android:clickable** attribute with a value of **true**.

Display the Map

The WalkAbout MapActivity will display the MapView. All initialization relating to the layout should be done in WalkAbout.initLayout(), which gets called by onCreate():

- Inflate your map_layout.xml file.
- Initialize the MapView **m_vwMap** member variable by retrieving a reference to the MapView element in the XML layout file.
- Force the Zoom Controls to be displayed in by making a call to MapView.setBuiltInZoomControls(...) (hint: you don't actually type MapView).

You should be able to run your application and see a Google Map. When you touch the map, the Zoom Controls should be displayed centered along the bottom of the screen.

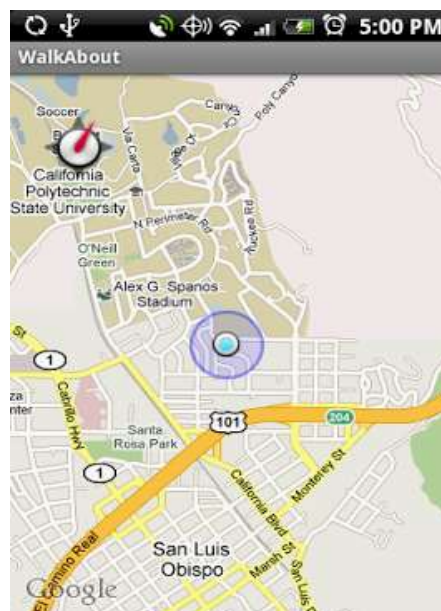
Adding Map Overlays

As stated before, Overlays are used when you want to draw objects on top of a MapView. Such objects could represent pin points for restaurant locations, a route for displaying

directions, or a simple logo. Each MapView instance is responsible for drawing the overlays that get displayed on top of it. As such, each MapView instance has a list of all Overlays that it should draw. When you want to add an Overlay to a MapView, you simply retrieve the MapView's list of Overlays and add to it.

The Overlay objects are different from regular View objects in that they are not requested to be drawn by the system. Instead they are requested to be drawn by the MapView when the system requests that the MapView draw itself. As such, when you make changes to an Overlay, you cannot explicitly request that the overlay be re-drawn by calling a method like View.invalidate() or View.postInvalidate(). In order to get the Overlay objects to be redrawn you must call View.invalidate() or View.postInvalidate() on the MapView that owns the Overlay objects.

You will now add an Overlay to your current MapView which will display the device's current location and heading on top of the MapView. When finished, your application should appear as depicted in the figure below:



Display Your Location

In order for your application to display your location, it must access Positioning data from a Location Provider. In order to do this, it must declare that it uses the ACCESS_FINE_LOCATION permission in its manifest.

- Do so by adding the following <uses-permission ... /> line to your AndroidManifest.xml file. This line should be nested inside of the <manifest></manifest> tags:

```
<manifest ...>
    <uses-permission
        android:name="android.permission.ACCESS_FINE_LOCATION"/>
</manifest>
```

Enable the map to display your current location and heading by initializing and adding a `com.google.android.maps.MyLocationOverlay` to your `MapView`. This should be done in the `WalkAbout.initLayout()` method:

- Initialize `m_locationOverlay` with a new `MyLocationOverlay` object.
- Enable the `MyLocationOverlay` object's Compass and `MyLocation` beacon by using the `MyLocationOverlay.enableCompass()` and `MyLocationOverlay.enableMyLocation()` methods.
 - It is important to note that the `MyLocationOverlay` depends on the GPS or Network Location settings being enabled. These methods will do nothing if you do not have either the Network or GPS Location setting enabled.
- Retrieve `m_vwMap`'s list of attached Overlay objects by calling its `getOverlays()` method.
- Add the `m_locationOverlay` to the list of Overlay objects.

You should be able to run your application and see a Google Map containing a beacon pointing out your current location and a compass displaying your current heading.

Initialize the WalkAbout Options Menu

The `WalkAbout` Activity has an Options Menu that will display five different `MenuItem`s. Create the Options Menu as follows: Do not worry about setting `OnMenuItemClickListeners` yet, you will do that later.



- "Start/Stop": This MenuItem acts as a toggle for either starting or stopping GPS Location recording. This recording is a record of the path that you have traveled. While recording, the MenuItem should display "Stop." While stopped, the MenuItem should display "Start."
 - Initialize the text for the MenuItem with R.string.startRecording resource string.
 - Later on, you will dynamically update the text to reflect the current recording state.
 - Use WalkAbout.STARTSTOP_MENU_ITEM constant as the Id for the MenuItem.
- "Save": This MenuItem will save the current recorded path.
 - Initialize the text for the MenuItem with R.string.save.
 - Use the WalkAbout.SAVE_MENU_ITEM constant as the Id for the MenuItem.
- "Load": This MenuItem loads the last saved path.
 - Initialize the text for the MenuItem with R.string.load.
 - Use the WalkAbout.LOAD_MENU_ITEM constant as the Id for the MenuItem.
- "Take Picture": This MenuItem launches a Camera Activity that allows you to take pictures.
 - Initialize the text for the MenuItem with R.string.takePicture.
 - Use the WalkAbout.PICTURE_MENU_ITEM constant as the Id for the MenuItem.
- "Enable GPS": This MenuItem launches the Device Settings Activity that allows you to enable the GPS Provider.
 - Initialize the text for the MenuItem with R.string.enableGPS.
 - Use the WalkAbout.ENABLEGPS_MENU_ITEM constant as the Id for the MenuItem.

Using Location Services

The Android System provides services for determining the current location of the device. The framework for working with these location based services lives under the android.location package. A number of useful classes live inside this package:

- **LocationManager**: Provides an interface for the location based services. You will be interacting with this class most of the time when trying to obtain location information.
- **LocationProvider**: LocationProviders are classes that provide updates on the current location of the device. There exists a LocationProvider for each different technology that determines location. There exists a GPS LocationProvider and a Network LocationProvider. Each LocationProvider specifies criteria that must be satisfied in order for it to be used. For example, the GPS LocationProvider requires that the device have GPS hardware and that it be enabled.

- **Criteria:** This class allows you to programmatically outline criteria you would like from a LocationProvider. Such criteria may include accuracy, power consumption, altitude reporting, speed reporting, monetary cost, etc. You can then setup and give an instance of this class to the LocationManager and allow it to choose the LocationProvider that best matches your criteria.
- **LocationListener:** This is an interface that defines call backs for different events that are generated by LocationProviders. These can be registered with a particular LocationProvider to receive updates when the location changes or the state of the LocationProvider changes.
- **Location:** A data object containing location based information. Generally contains latitude and longitude, as well as a date and timestamp containing the time at which the location was determined. Might also contain elevation, speed, and heading information.
- **Location based utility classes:** A utility class for Geo-Coding which transforms addresses to GPS Coordinates and back, a data object class for addresses, and some classes for monitoring the status of the GPS satellites the device is locked onto.

In general, you usually query the LocationManager for the information you are seeking. You can check the current status of a LocationProvider, you can check the last known location reported by a LocationProvider, and you can register to receive updates from a LocationProvider all through the LocationManager (*just to name a few*). To receive updates on location information directly from a LocationProvider, you need to implement the LocationListener interface and register yourself with the LocationProvider.

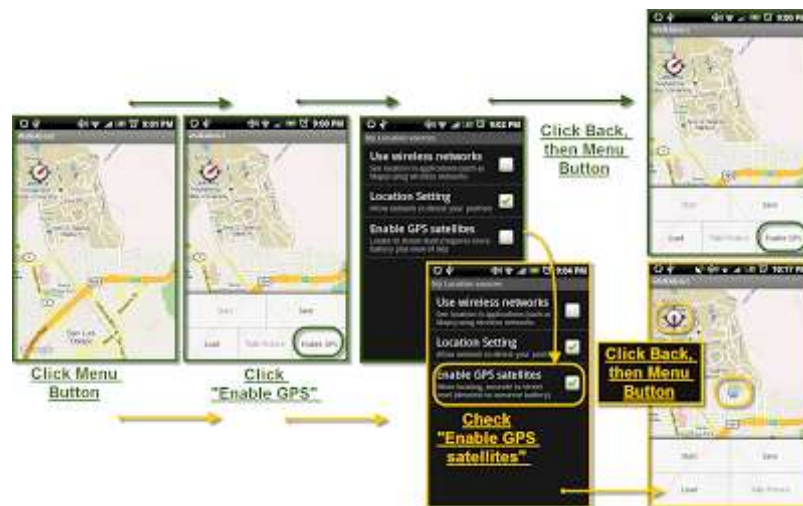
In the subsections that follow, you add functionality to monitor and enable the GPS LocationProvider in the WalkAbout Activity class. You will record changes in location as the user's path. You will then implement an Overlay class that will draw this path onto the WalkAbout Activity's MapView object.

Testing & Enabling the GPS LocationProvider

It is entirely possible that the user has disabled the GPS hardware. Before you can monitor and record data from the GPS LocationProvider, the GPS hardware must be enabled. In this particular subsection, you will begin by querying the LocationManager for whether the GPS hardware is enabled on the device. If GPS is not enabled, you will allow the user to launch the Location Settings Activity to enable it from the "Enable GPS" Options MenuItem. After the user is done editing the Location Settings, they can return to the WalkAbout Activity by hitting the back button. When the GPS hardware is enabled the "Enable GPS" MenuItem will no longer be visible in the Options Menu.

The following diagram depicts two use cases (*Click on the image to view it at full size*). Both use cases have the application starting up without GPS hardware enabled and

without the Network LocationProvider enabled. The first use case follows the dark green arrows along the top of the figure. The User clicks the menu button, then clicks the "Enable GPS" MenuItem. The Location Settings Activity launches, the user then **decides not to enable the GPS**. The user clicks the back button and is returned to the Walkabout Activity. The user clicks the Menu button, and the "Enable GPS" MenuItem is still visible. Notice that the Network and GPS Location Providers are disabled, that the MyLocationOverlay's Current Location Beacon is not present, and Compass heading location is inaccurate. This is because the MyLocation Overlay relies on the LocationProvider for displaying this information.



The second use case follows the golden arrows along the bottom of the figure. The User clicks the menu button, then clicks the "Enable GPS" MenuItem. The Location Settings Activity launches, the user then **decides to enable the GPS**. The user checks the "Enable GPS satellites" check box (*Note that this option may be labeled different in different version of Android*). The user clicks the back button and is returned to the Walkabout Activity. The user clicks the Menu button, and the "Enable GPS" MenuItem is no longer visible. Notice that when the user returns to the WalkAbout Activity after enabling the GPS hardware, that the MyLocationOverlay's current Location Beacon is now present, and Compass heading location is now accurate. It is also important to note that if the GPS Hardware is enabled when the application starts, then the "Enable GPS" MenuItem should not be visible.

Initialize LocationManager

In various parts of the WalkAbout class, you will request location information from the LocationManager attached to this application context. The WalkAbout class will store a reference to this LocationManager object in the m_locManager member variable for convenience. Initialize this member variable in the WalkAbout.initLocationData() method:

- Retrieve the LocationManager by calling the Activity.getSystemService(...) method, passing in the Context.LOCATION_SERVICE constant.

- Remember that **this** instance of the *WalkAbout* class **is** an *Activity* and therefore all public and protected *Activity* methods are members of **this** object.
- set `m_locManager` to the object returned to you. You will have to cast the return value, because `getSystemService(...)` returns an object of type `Object`.

Dynamically Update Options Menu

The Options menu should display the "Start/Stop" MenuItem as disabled if the GPS Location Provider *is disabled*. Additionally, the Options menu should NOT display the "Enable GPS" MenuItem at all if the GPS Location Provider *is enabled*.

Each time the Options menu is displayed, you should check to see if the GPS Location Provider is enabled and update the "Start/Stop" and "Enable GPS" MenuItems accordingly.

- You must do this from the `Activity.onPrepareOptionsMenu(...)`. If this is done from the `Activity.onCreateOptionsMenu(...)` it will only be done the first time the menu is shown. See the Android Documentation on this method for more details.
- You can determine whether a Location Provider is enabled by calling the `LocationManager.isProviderEnabled(...)` method and passing in the name of the Provider. See the Android Documentation on this method for more details.
- The names of the different Providers are stored as string constants in the `LocationManager` class. Check the `LocationManager` Constants Documentation to determine which string to pass in.

You should be able to run your application and test that the "Start/Stop" and "Enable GPS" MenuItems are enabled and invisible respectively when the GPS Provider *is enabled*. They should also be disabled and visible respectively when the GPS Provider *is disabled*.

Implement "Enable GPS" MenuItem

When the GPS Location Provider is disabled, the user will be able to launch the settings activity to enable the GPS Location Provider from the "Enable GPS" Options MenuItem. Set and fill in the `OnMenuItemClickListener` for the "Enable GPS" MenuItem:

- Instantiate a new `Intent` object, passing in the `Settings.ACTION_LOCATION_SOURCE_SETTINGS` *Action* String constant.

- Call the `Activity.startActivityForResult()` method passing in the intent you just created and the `WalkAbout.ENABLE_GPS_REQUEST_CODE` static constant.
 - We pass in the Request Code so that when we handle the result, we know which request generated the result. The result of any Activity that we launch gets processed by the same method and the Request Code mechanism lets us know which result is coming from which Activity.

You should be able to run your application and test that you can launch the settings activity to enable the GPS provider from the "Enable GPS" MenuItem. When you hit the Back button you should return to the WalkAbout Activity. However, you should note that the MyLocationOverlay still won't be working.

Handle the Location Settings Activity Result

After the user is done with the Location Settings Activity and returns to the WalkAbout Activity, you need to try re-enabling the MyLocationOverlay. Since you called the `Activity.startActivityForResult(...)` method to launch the Location Settings Activity, the `WalkAbout.onActivityResult(...)` method will be called. The `requestCode` parameter in this method will contain the value that you passed into the `startActivityForResult(...)` method. You should test this `requestCode` parameter to see if it matches the `WalkAbout.ENABLE_GPS_REQUEST_CODE` constant. You test the `requestCode` because this same method will be called when any Activity that you launch returns a result (*You must test this value because you will be launching another activity later and you want to be able to identify which Activity is returning a result*).

Do this by overriding and filling in the `Activity.onActivityResult(...)` method:

- Make sure to call `super.onActivityResult(...)`.
- Test the `requestCode` argument to ensure that it matches the request code that you used when you launched the CameraPreview activity.
 - If the request codes don't match, then don't do anything and just return. If they do match then continue on.
- Attempt to re-enable the Compass and MyLocation beacon as you did in section 2.2.1.

You should now run your application with all Location Providers disabled and test that the MyLocation Overlay is displayed properly once you enable the GPS LocationProvider.

Retrieving GPS Location Information

In this subsection, you will implement the functionality necessary to monitor and record GPS location information. You will monitor GPS location information by making the WalkAbout Activity register itself with the GPS LocationProvider. By doing this, the WalkAbout Activity will be notified by the GPS LocationProvider when the location changes. However, before the WalkAbout Activity class can register itself with the GPS LocationProvider, it must implement the `com.google.android.maps.LocationListener` interface. You will start by making the WalkAbout Activity implement the `LocationListener` interface.

You will then implement a "Start/Stop" MenuItem that will toggle the WalkAbout Activity between actively-recording and recording-stopped states. While in the actively-recording state, the WalkAbout Activity will be registered to receive updates from the GPS LocationProvider. As the WalkAbout Activity receives notices about location changes, it will re-center its MapView about the new location. It will also store changes in location as latitude-longitude coordinates in an ArrayList. Additionally, while in the actively-recording state the "Start/Stop" MenuItem will display the word "Stop" to indicate that clicking on the MenuItem will cause the WalkAbout Activity to switch to the recording-stopped state.

While in the recording-stopped state, the WalkAbout Activity will no longer be registered to receive updates from the GPS LocationProvider. Additionally, while in the actively-recording state the "Start/Stop" MenuItem will display the word "Start" to indicate that clicking on the MenuItem will cause the WalkAbout Activity to switch to the recording-stopped state. Note that when the Application starts up, it is by default in the recording-stopped state. When the user switches from the recording-stopped state to the actively-recording state, the ArrayList used to record changes in location should be cleared. See the Figure below for more details on the use case:



Implement LocationListener Interface

The WalkAbout Activity class will receive updates from the GPS Location Provider by registering itself with the Provider. In order to register with the Provider, the WalkAbout Activity must implement the `LocationListener` interface. This interface provides a set of callback methods that the Provider will call when the location changes, the Provider is enabled, the Provider is disabled, or when the status of the Provider changes.

In particular, you will monitor and record location changes, and stop recording in the event that the Provider is disabled for some reason. The WalkAbout class records location changes in an `ArrayList<GeoPoint>` member variable named `m_arrPathPoints`.

It also maintains the current recording state (*whether it is currently recording or not*) in a boolean member variable named `m_bRecording`. Perform the following initializations in the `initLocationData()` method.

- Initialize `m_arrPathPoints` to a new `ArrayList<GeoPoint>`.
- Initialize `m_bRecording` to false.

Make the `WalkAbout MapActivity` class implement the `LocationListener` interface:

- There is a total of four methods that you must implement. See the Android Documentation on `LocationListener` for a complete listing and description of each.
 - *Of the four methods, you only need to fill in two of them, `onLocationChanged(...)` and `onProviderDisabled(...)`.*
 - *You still need to declare the other two methods but you can leave them as empty stubs.*
- Record location changes in the `onLocationChanged(...)` method.
 - *The `Location` argument passed in has both degrees latitude and longitude values, represented as type `double`. Some of the Google Maps classes that you will be using later on make use of the `GeoPoint` class, which represents latitude and longitude as `int` values of microdegrees. This is why you are storing an `ArrayList` of `GeoPoints` and not `Locations`. You will have to convert from `Location` to `GeoPoint`.*
 - *1 degree is equal to 1E6 microdegrees. The `WalkAbout Activity` class has a constant for this value named `GEO_CONVERSION`*
 - *99.123456 degrees = 99123456 microdegrees*
 - Get the degrees latitude value from the `Location` argument by calling the `Location.getLatitude()` method and convert it to microdegrees.
 - Convert by multiplying the returned `double` by the `WalkAbout.GEO_CONVERSION` integer constant.
 - Get the degrees longitude value from the `Location` argument by calling the `Location.getLongitude()` method and convert it to microdegrees.
 - Instantiate a new `GeoPoint` object, passing into the constructor the latitude and longitude values you just converted into microdegrees.
 - You will have to cast these to type **int** as they are currently of type **double**.
 - Add the `GeoPoint` to `m_arrPathPoints`.
- In the `onProviderDisabled(...)` method, instruct the `WalkAbout Activity` to stop recording location changes.

- Do this by making a call to `WalkAbout.setRecordingState(false)`. You will implement this method in the sections that follow.
-

Dynamically Update the "Start/Stop" MenuItem Text

When the Options Menu is displayed, the text of the "Start/Stop" MenuItem should be set to the **R.string.startRecording** if the activity is **not recording** and **R.string.stopRecording** if the activity is **recording**.

3.2.3 Implement "Start/Stop" MenuItem

When the "Start/Stop" MenuItem is clicked, the recording state should be toggled. If the activity is currently recording, then the activity should stop recording. Conversely, if the activity is not recording, then the activity should start recording.

The functionality for setting the recording state will be encapsulated in the `WalkAbout.setRecordingState(boolean recording)` method. Passing in a value of true indicates that the activity should start recording and passing in a value of false indicates that the activity should stop recording.

Set and fill in the "Start/Stop" OnMenuItemClickListener so that it toggles the recording state by calling `WalkAbout.setRecordingState(...)` with the proper value.

- This is all this method should do.
- Make use of the `m_bRecordingState` member variable to determine the current recording state.

Fill in the `setRecordingState(...)` method:

- You should update the `m_bRecording` member variable with the new state.
- If the new state is **not recording** you should tell the GPS Provider to stop sending you Location Updates.
 - You can do this by calling the `LocationManager.removeUpdates(...)` method and pass in the `LocationListener` that you want to un-register.
- Otherwise, if the new state is **recording**, you should re-initialize your recording data.
 - All previously recorded GeoPoints should be erased. Do not instantiate a new list of GeoPoints, just clear your current one.
 - *You should get used to trying to re-use objects when possible. It is expensive to create new ones and it may be some time before the old ones get garbage collected.*
 - Initialize your list of GeoPoints with the last known location from the GPS Provider.

- You can do this by calling the `LocationManager.getLastKnownLocation(...)` method.
- You can then manually call your `WalkAbout.onLocationChanged(...)` method with the Location you just retrieved. This will update your list of GeoPoints for you.
- Tell the GPS Provider to start sending you Location Updates.
 - You can do this by calling the `LocationManager.requestLocationUpdates(...)` method.
 - Pass in the String name of the Location Provider you want to receive Location Updates from
 - Pass in `WalkAbout.MIN_TIME_CHANGE`, to specify the minimum amount of time between updates.
 - Pass in `WalkAbout.MIN_DISTANCE_CHANGE`, to specify the minimum amount of distance between updates.
 - Pass in the `LocationListener` which should receive the updates.

If you run your application, it should now record GPS Location changes. You can test this by adding a Toast notification to the `WalkAbout.onLocationChanged(...)` method that prints out the new location. Don't forget to remove the Toast after you've finished testing.

If you are running this on the emulator, you can simulate location changes on the emulator two different ways. You can do this from the DDMS perspective in Eclipse, or from the Console. For instructions on how to open up a console, see the Android reference on Using the Console.

- **DDMS:** see the DDMS documentation on Emulator Controls. Feel free to use the `testGPS.gpx` file included in the root folder of the skeleton project as a test path.
- **Console:** see the documentation on Location Services

Center MapView on Current Location

As the application stands now, when you change location, the MapView does not move. It is therefore possible for your MyLocation beacon to move off of the MapView. You will now change this so that when your position changes, the MapView will center itself about the MyLocation beacon. This is done using the MapView's MapController object. The MapController can be used for a number of different things including changing the zoom level.

- In the `WalkAbout.onLocationChanged(...)` method, retrieve the `MapController` from your `MapView` by calling the `MapView.getController()` method.
- Use the `MapController.animateTo(Point point)` method to force the map to scroll so that the current location is now in the center.

You should now be able to run your application to ensure that the Map is always centered about the device's current location.

Path Overlay

For this subsection, you will implement your own `Overlay` class that draws a path on top of a `MapView`. You will begin by filling in the `PathOverlay` class so that it draws a green starting circle at the first point in the path, and then draws a red line to connect each subsequent point in the path. The `PathOverlay` class will contain a reference to a `List` of `GeoPoints`. You will then add the `PathOverlay` to the `WalkAbout` Activity's `MapView` object, setting it up so that the `PathOverlay` uses the `WalkAbout` Activity's recorded list of `GeoPoints` as the path that it should draw.

Fill in the PathOverlay Class

Begin by filling in the constructor:

- Set `m_arrPathPoints` to the `pathPoints` constructor argument. This is the list of `GeoPoints` that represents the path that this overlay class will draw. Notice that it is declared as `final`; from the perspective of the `PathOverlay` class this object is immutable/Read-Only. You should not attempt to modify the contents of this `List`, you should only read from it.



- Initialize each of the other four member variables with new default constructed objects. You will use these objects to help you draw the path. Instead of recreating them every time the draw method is called, you will simply re-use them to improve performance.

The only thing that this object is supposed to do is draw the path overlay, and thus it has only one method, `draw(...)`. The `PathOverlay` should draw the first point as a green circle and connect each subsequent point with red a line. If the list of `GeoPoints` is empty or null, the method should do nothing.

The `Canvas` argument that is passed into the `draw(...)` method is what performs the drawing for you. You can get something to show up on the screen by asking the `Canvas` object to draw something for you. You do that by telling it what to draw, where to draw it, and what color it should use. The `MapView` argument is a reference to the `Map` on which this overlay will be drawn. Don't worry about the `shadow` argument for now.

Start by drawing the starting circle:

- Set the color to green by calling the `Paint.setARGB(...)` method on `m_paint`, passing in the Red, Green, Blue, and Alpha values.
- Specify where to draw the circle by calling the `RectF.set(...)` method on `m_rect`. This outlines a `Rectangle` in which to draw the circle. The `RectF.set(...)` method takes in the x values of the left & right edges and the y values of the top & bottom edges.
 - Before you can call the `RectF.set(...)` method, you need to translate your first `GeoPoint` coordinate into screen coordinates.
 - Call the `MapView.getProjection()` method to retrieve a projection object that will perform the translation for you.
 - You can now call the `Projection.toPixels(...)` method, passing in your first `GeoPoint` and the `m_point` member variable.
 - After the call, `m_point` will contain the x-y screen coordinates.
 - Use the following line to set the bounding rectangle:

```
m_rect.set(m_point.x-START_RADIUS, m_point.y-
START_RADIUS, m_point.x+START_RADIUS,
m_point.y+START_RADIUS);
```

- Tell the canvas to draw the circle by calling `Canvas.drawOval(...)`, passing in your bounding rectangle and the color it should use.

You should now start drawing your path. For each point in the list of `GeoPoints`, draw a red line starting from the current point to the point following it. For example, if your list has **3** points in it, you should draw a line connecting point-1 to point-2 then you should draw a line connecting point-2 to point-3.

- Use `m_point2` to store the second set of screen coordinates.
- The line should have a thickness equal to `PathOverlay.PATH_WIDTH`. You can set the width of the line by calling the `Paint.setStrokeWidth(...)` method.
- You can use the `Canvas.drawLine(...)` method to draw the line.
 - *Note: You won't use the `m_rect` to draw the path, you only use it to draw the starting circle.*

Add a PathOverlay to Your Map

In the `WalkAbout` activity class

- Initialize and add a `PathOverlay` to your `MapView` in the `WalkAbout.initLayout()` method.
- Since the `PathOverlay` only gets redrawn when the `MapView` gets redrawn, you need to notify the `MapView` to redraw itself any time you update your path/recorded-list-of-GeoPoints.
 - You tell the `MapView` to redraw itself by calling `MapView.postInvalidate();`
 - When you add `GeoPoints` to your recording-list/path, you need to redraw the `MapView`.
 - When you clear your recording-list/path, you need to redraw the `MapView`

You should now be able to run your application to ensure that when recording, your path gets continuously updated and displayed on the `MapView`.

Using The Camera

Camera Preview

Android allows applications direct access to the Camera Hardware. In the section that follows you will implement the `WalkAbout` Activity "Take Picture" `MenuItem`. The `MenuItem` should only be enabled while the Application is in the actively-recording state. When the `MenuItem` is clicked, it will launch the `CameraPreview` Activity. It is your job to extend this Activity so that it can capture a picture and save it to the SD-Card. The `CameraPreview` Activity was borrowed from the Google API Demos application, and currently is only able to display a full screen preview of what the camera is looking at.

The `CameraPreview` Activity will take a picture once the user touches anywhere on the screen. Once the picture is taken, it will be saved on the SD-Card and the file-name will be returned to the `WalkAbout` Activity as a result. When the user returns to the `WalkAbout` Activity, a short Toast notification containing a success message and the file-name the picture was saved to will be displayed to the user. Alternatively, the user can hit the back button without taking a picture to return to the `WalkAbout` Activity, effectively canceling the `CameraPreview` Activity. Should the `CameraPreview` Activity be canceled, the user will see a short Toast notification containing a failure message.



Dynamically Update Options Menu

When the WalkAbout Options Menu is displayed, the "Take Picture" MenuItem should only be enabled if the Activity is recording GPS location changes. If the Activity is not recording, the MenuItem should be disabled.

Launch the CameraPreview Activity

When the "Take Picture" MenuItem is clicked, the CameraPreview Activity should be launched. Currently, the CameraPreview Activity class does not have the ability to take a picture, but rather only provides a preview image. You will add the functionality to this class to take the picture in the next section. Before you can do that, you need to update your the AndroidManifest.xml:

- Add the following `<uses-permission ... />` and `<uses-feature ... />` lines to your AndroidManifest.xml file. These lines should be nested inside of the `<manifest></manifest>` tags:

```
<manifest ...>
    <uses-feature android:name="android.hardware.camera" />
    <uses-permission
        android:name="android.permission.CAMERA"/>
</manifest>
```

In the WalkAbout Activity, set and fill in the "Take Picture" OnMenuItemClickListener so that it launches the CameraPreview Activity and receives a result back from the activity.

- Create a new Intent, passing in a reference to this application Context and the edu.calpoly.android.walkabout.CameraPreview Class object.
- Call `Activity.startActivityForResult(...)`, passing in the Intent object and the `WalkAbout.PICTURE_REQUEST_CODE` integer constant.

Take a Picture from the CameraPreview Activity

The CameraPreview Activity class comes from the Google Android API Demos application which provides sample applications for a number of different Android API's.

In order to display a Camera Preview you need to use some advanced Android features. Instead of having you implement this yourself, it's provided for you so that you can see how its done.

- Notice the inner Preview class that extends the SurfaceView class. This is the View that renders the Camera Preview Image and displays it to the user.
- The SurfaceView class is used when rendering is too computationally expensive to be done from the GUI thread or the View needs to be refreshed very quickly.
- The Preview class uses a Camera object and its Camera.Parameters inner class to retrieve the camera image data and manipulate its settings.

A single getter method was added to the Preview class for the purposes of retrieving a reference to the Camera object. This was done so that you can add functionality to the CameraPreview Activity class to take a picture. When a user clicks anywhere on the CameraPreview's mPreview object (which occupies the entire screen) the Preview object's camera should take a picture.

- Set an OnClickListener for the CameraPreview's Preview object and fill in its onClick(...) method
 - Retrieve the Preview object's Camera and call its takePicture(...) method.
 - This method takes 3 separate listeners, each of which implement a different callback that will be called during different stages of the photo taking process. All of these can be null if we don't want to respond to any of these stages.
 - The First is a ShutterCallback that will be called once the image has been captured.
 - The Second is a PictureCallback that will be called once the raw image data has been generated.
 - The Third is another PictureCallback that will be called once the jpeg data has been generated. This is the only one we are interested in.
 - Pass in null for the first two parameters and pass in a reference to *this* CameraPreview instance for the third. CameraPreview implements the PictureCallback interface. This will make CameraPreview's onPictureTaken(...) method be called once the jpeg data has been generated.

You should be able to run your application and verify that you can launch the CameraPreview activity. You should be able to see a preview of what the camera is seeing. If you add a Toast notification to the CameraPreview.onPictureTaken(...) method, you should see the notification when you click anywhere on the screen.

Save the Picture to SD-Card

Now that the picture has been taken, we need to save the jpeg data into a file on the SD-Card when the CameraPreview.onPictureTake(...) method is called. In order for your

application to write data to the SD-Card, it must declare that it uses the `WRITE_EXTERNAL_STORAGE` permission in its manifest.

- Do so by adding the following `<uses-permission ... />` line to your `AndroidManifest.xml` file. This line should be nested inside of the `<manifest></manifest>` tags:

```
<manifest ...>
  <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
</manifest>
```

Now you can implement the `File-Output` functionality by filling in the `CameraPreview.onPictureTaken(...)` method:

- Instantiate a new `java.io.File` object, passing in the path to the SD-Card directory and the name of the File you want to create.
 - You can retrieve the path to the SD-Card directory by calling the static `Environment.getExternalStorageDirectory()` method.
 - To guarantee that all of your file names are unique, use the Current System Time appended with the ".jpg" extension as your file name.
 - *For example: If you call `System.currentTimeMillis()` your file will look something like `"111111111111.jpg"`*
- Instantiate a new `java.io.FileOutputStream` object passing your new `File` object into the constructor.
- Write the **byte[] data** argument passed into the `onPictureTaken(...)` method out to the `FileOutputStream`.
- Flush the stream and close it.
- You must catch any `IOExceptions` that occur.
 - Should an `IOException` occur, set the result of the Activity by calling the `Activity.setResult(...)` method.
 - Pass in `Activity.RESULT_CANCELED` as the result code and null as the Intent data to indicate that the Picture was not taken successfully.
 - call the `Activity.finish()` method
- If the file was written successfully:
 - Default instantiate a new Intent object. You can use this Intent object to pass the path to the image file back to the `WalkAbout` Activity.
 - Use the `Intent.putExtra(...)` method to store the absolute path to the image file. You can use `CameraPreview.IMAGE_FILE_NAME` as the key.
 - Set the result for the Activity with `Activity.RESULT_OK` and the Intent data, then finish the Activity.

Handle the Result

After the `CameraPreview` Activity finishes, the `WalkAbout` Activity needs to handle the result of the Intent. The `WalkAbout` Activity should make a Toast Notification

displaying either success or failure as returned by the CameraPreview Activity. If the pictures was taken successfully, the path to the created image file should be displayed in the notification. Do this by overriding and filling in the Activity.onActivityResult(...) method:

- Make sure to call super.onActivityResult(...).
- Test the **requestCode** argument to ensure that it matches the request code that you used when you launched the CameraPreview activity.
 - If the request codes don't match, then don't do anything and just return. If they do match then continue on.
- Test the **resultCode** argument.
 - If it equals Activity.RESULT_CANCELED then print a Toast Notification displaying the R.string.pictureFail string resource.
 - If it equals Activity.RESULT_OK then print a Toast Notification displaying the R.string.pictureSuccess string resource and the full image file path.
 - You can retrieve the image file path by calling getExtras().getString(CameraPreview.IMAGE_FILE_NAME) on the Intent **data** argument.

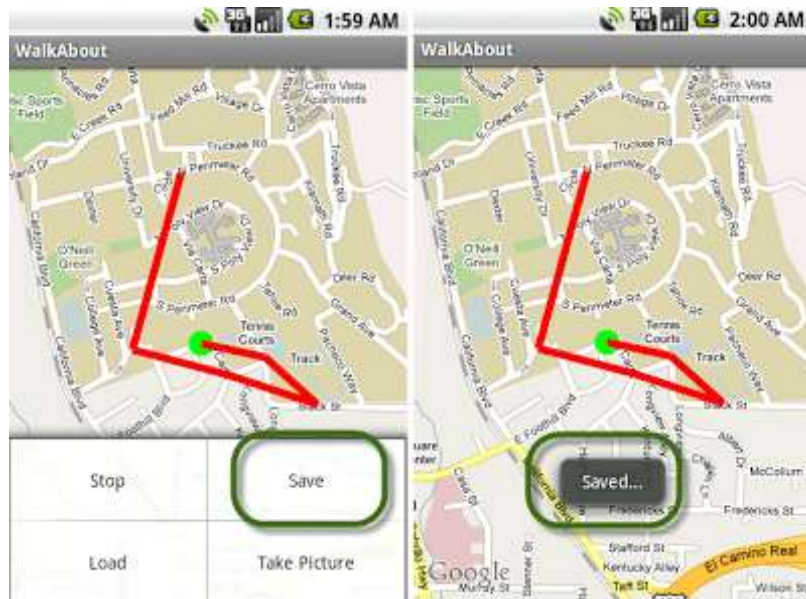
You should be able to run your application and verify that the Toast Notification displays the proper text when the CameraPreview Activity finishes.

Private Application Files

Android allows you to Create, Write, Read and Delete local files. These files become private to the application that created them by default. You have the option of overriding this privacy mechanism, allowing them to be shared with other applications.

Creating, Writing, & Deleting Files

You will now implement WalkAbout Activity's "Save" MenuItem. This MenuItem's OnMenuItemClickListener should only make a single call to the WalkAbout.saveRecording() method. The functionality necessary to save the current recorded path should be composed in the WalkAbout.saveRecording() method. You should fill in the saveRecording method so that it properly writes out only the contents of m_arrPathPoints to a private application file. You should only ever create one file and it should be truncated/cleared each time you write to it.



The format of the file is quite simple. The contents of `m_arrPathPoints` will be written into a single continuous line. The latitude and longitude of each `GeoPoint` in `m_arrPathPoints` should be written out in that order, separated by a comma, and no spaces between them. Each `GeoPoint` should be separated by a semi-colon, and no spaces between them. So for example:

Given:

```
m_arrPathPoints = [(lat1,long1),(lat2,long2),(lat3,long3)]
```

The file should look like:

```
lat1,long1;lat2,long2;lat3,long3;
```

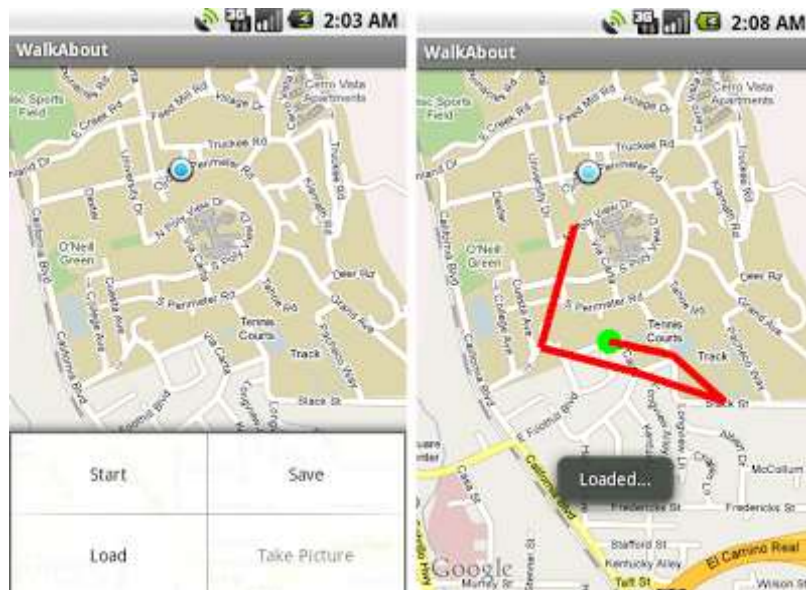
If the Save was performed successfully, then a Toast notification should be displayed containing the `R.string.saveSuccess` resource string. If an exception is thrown you should display a Toast notification containing the `R.string.saveFailed` resource string. If there is no data to save, as is the case on the initial loading of the application, then a Toast notification should be displayed containing the `R.string.saveNoData` resource string. You are tasked to do this on your own, however, you should make use of the following hints:

- Use the `R.string.geoPathFileName` string resource as the filename
- You will have to make use of the `Context.openFileOutput(String name, int mode)` and `Context.deleteFile(String name)` methods.
- You will have to use the `Context.MODE_PRIVATE` constant.
- Documentation on the `java.io.PrintWriter` class (*This is merely a suggestion, you are free to use whatever Java I/O classes you like to write to the file*).

Reading Files

Your last task is to implement the WalkAbout Activity's "Load" MenuItem. This MenuItem's `OnMenuItemClickListener` should only set the recording state to false and make a call to the `WalkAbout.loadRecording()` method. The rest of the File Loading

functionality should be implemented in the `loadRecording()` method. You should fill in the `loadRecording` method so that it properly initializes `m_arrPathPoints` to contain only the data in the file that `saveRecording()` writes out. Once finished, the path loaded from the file should be displayed exactly as it was when it was first recorded.



If the Load was performed successfully, then a Toast notification should be displayed containing the `R.string.loadSuccess` resource string. If an exception is thrown, you should display a Toast notification containing the `R.string.loadFailed` resource string. You are to do this task on your own, however, you should make use of the following hints:

- You will have to make use of the `Context.openFileInput(String name)` method in order to open the file.
- Documentation on the `java.util.Scanner` class (*This is merely a suggestion, you are free to use whatever Java I/O classes you like to read from the file*).

1) Stage J (Journey)

Introduction

For this lab you will be developing a new GPS recording application called WalkAbout. The purpose of the application is to allow users to record their GPS location information as they travel. While the application records the user's GPS data, it displays it back to the user in the form of a path drawn on top of a Google Map. While recording data, the user can launch a Camera activity that will capture and store pictures on an SD-Card. When finished recording, the application gives the user the option of storing the current GPS data as a private application file to be loaded and displayed at a later time.

2) Stage a1 (apply)

Lab Activities:

Activity 1:

- 1) Open eclipse or android studio and select new android project
- 2) Give project name and select next
- 3) Choose the android version. Choose the lowest android version (Android 2.2) and select next
- 4) Enter the package name. package name must be two word separated by comma and click finish
- 5) Go to package explorer in the left hand side. select our project.
- 6) Go to res folder and select layout. Double click the main.xml file. Add the code below

Code

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/relativeLayout1"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <Button
        android:id="@+id/show_Location"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Show_Location"
        android:layout_centerVertical="true"
        android:layout_centerHorizontal="true"
    />
</RelativeLayout>
```

- 7) Now select mainactivity.java file and type the following code. In my coding mainactivity name

is GPSlocationActivity.

```
package gps.location;
//import android.R;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;
public class GPSlocationActivity extends Activity {
    /** Called when the activity is first created. */
    Button btnShowLocation;
    GPSTrace gps;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        btnShowLocation=(Button)findViewById(R.id.show_Location);
        btnShowLocation.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
```

```

gps=new GPSTrace(GPSlocationActivity.this);
if(gps.canGetLocation()){
double latitude=gps.getLatitude(); double
longitude=gps.getLongitude();
Toast.makeText(getApplicationContext(),"Your Location is
\nLat:"+latitude+"\nLong:"+longitude, Toast.LENGTH_LONG).show();
}
else
{
gps.showSettingAlert();
}}});}}

```

8)Go to src folder and Right Click on your package folder and choose new class and give the class name as GPSTrace

9)Select the GPSTrace.java file and paste the following code.

```

package gps.location;
import android.app.AlertDialog;
import android.app.Service;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.provider.Settings;
public class GPSTrace extends Service implements LocationListener{
private final Context context;
boolean isGPSEnabled=false;
boolean canGetLocation=false;
boolean isNetworkEnabled=false;
Location location;
double latitude;
double longitude;
private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES=10;
private static final long MIN_TIME_BW_UPDATES=1000*60*1;
protected LocationManager locationManager;
public GPSTrace(Context context)
{
this.context=context;
getLocation();
}
public Location getLocation()
{
try{
locationManager=(LocationManager) context.getSystemService(LOCATION_SERVICE);
isGPSEnabled=locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);
isNetworkEnabled=locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);
if(!isGPSEnabled && !isNetworkEnabled){

```



```

    }else{
    this.canGetLocation=true;
    if(isNetworkEnabled){
    locationManager.requestLocationUpdates(
    LocationManager.NETWORK_PROVIDER,
    MIN_TIME_BW_UPDATES,
    MIN_DISTANCE_CHANGE_FOR_UPDATES,this);
    }
    if(locationManager!=null){
    location=locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVID
    ER)
    ;
    if(location !=null){
    latitude=location.getLatitude();
    longitude=location.getLongitude();
    }
    }
    }
    if(isGPSEnabled){
    if(location==null){
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,MIN_TIME_
    B
    W_UPDATES, MIN_DISTANCE_CHANGE_FOR_UPDATES, this);
    if(locationManager!=null){
    location=locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
    if(location!=null){
    latitude=location.getLatitude();
    longitude=location.getLongitude();
    }}}}
    catch(Exception e)
    {
    e.printStackTrace();
    }
    return location;
    }
    public void stopUsingGPS(){
    if(locationManager!=null){
    locationManager.removeUpdates(GPSTrace.this);
    }}
    public double getLatitude(){
    if(location!=null){
    latitude=location.getLatitude();
    }
    return latitude;
    }
    public double getLongtiude(){
    if(location!=null){
    longitude=location.getLatitude();
    }
    return longitude;
    }

```

```

public boolean canGetLocation(){
return this.canGetLocation;
}
public void showSettingAlert(){
AlertDialog.Builder alertDialog=new AlertDialog.Builder(context);
alertDialog.setTitle("GPS is settings");
alertDialog.setMessage("GPS is not enabled.Do you want to go to setting menu?");
alertDialog.setPositiveButton("settings", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog,int which){
Intent intent=new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
context.startActivity(intent);
}
});
alertDialog.setNegativeButton("cancel", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
// TODO Auto-generated method stub
dialog.cancel();
}
});
alertDialog.show();
}
@Override
public void onLocationChanged(Location location) {
// TODO Auto-generated method stub
}
@Override
public void onProviderDisabled(String provider) {
// TODO Auto-generated method stub
}
@Override
public void onProviderEnabled(String provider) {
// TODO Auto-generated method stub
}
@Override
public void onStatusChanged(String provider, int status, Bundle extras) {
// TODO Auto-generated method stub
}
@Override
public IBinder onBind(Intent intent) {
// TODO Auto-generated method stub
return null;
}
}
}
10)Go to manifest.xml file and add the code below
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
android:name="android.permission.INTERNET"/>
11)Now go to main.xml and right click .select run as option and select run configuration

```

12) Android output is present in the android emulator as shown in below.

3) Stage v (verify)

Home Activities:

Activity 1:

Description :

- a. Setup the Android development environment. We highly recommend that you develop the Android project in Eclipse. You also need to install Google APIs package using the android SDK. In this assignment, we recommend that you finish the assignment using the Android Platform 2.3.3, API level 10.
- b. This assignment has three parts: **download a contact file, add data to Phone Contacts using code (not by Contact software), show the people's positions with their names on Google Map(you can only query the Contacts to get the data in this step).** (Details below)
- c. The Project should be named "MapOfContacts".

Detail:

- a. Fetch the contact file "contact.txt" from the url:

<http://www.cs.columbia.edu/~coms6998-8/assignments/homework2/contacts/contacts.txt>

The data will look like

Dan dan@columbia.edu 40010787 116257324

John john@gmail.com 23079732 79145508

Daniel daniel@gmail.com 37985339 23716735

Johnny johnny@gmail.com 40774042 -73959961

Makiyo makiyo@gmail.com 36155618 139746094

Add first column to Name in the Contacts. Add second column to Email in the Contacts.

Add third column to Mobile Number in the Contacts. (the number will be used as latitude in the Map) Add fourth column to Home Number in the Contacts. (the number will be used as longitude in the Map).

b. You have to set up API keys in order to use Google Map API. Here is a tutorial, you can set up your API keys following it:

<http://remwebdevelopment.com/dev/a35/Android-How-To-Set-Up-an-API-Key-for-Google-Maps.html>

c. Just showing markers on GoogleMap is not enough to show which person in what place, so I recommend you to use AlertDialog to show the Name of the person when you tap it.

d. This assignment has no restriction on what features you must use and also no restriction on what user interface you design. As long as you can implement those three functions in code, and demo to me that the results is correct. You are fine.

Statement Purpose:

In this section students learn that how to make their application public.

Activity Outcomes:

After completing this chapter we will be able to understand the following topics.

- Debugging
- Android App Development Life Cycle
- APK Conversion Process
- App Publishing Guidance
- Tips for Launching an App

Instructor Note:

Debugging

The Android SDK provides most of the tools that you need to debug your applications. You need a JDWP-compliant debugger if you want to be able to do things such as step through code, view variable values, and pause execution of an application. If you are using Android Studio, a JDWP-compliant debugger is already included and there is no setup required. If you are using another IDE, you can use the debugger that comes with it and attach the debugger to a special port so it can communicate with the application VMs on your devices. The main components that comprise adb acts as a middleman between a device and your development system. It provides various device management capabilities, including moving and syncing files to the emulator, running a UNIX shell on the device or emulator, and providing a general means to communicate with connected emulators and devices.

Dalvik Debug Monitor Server

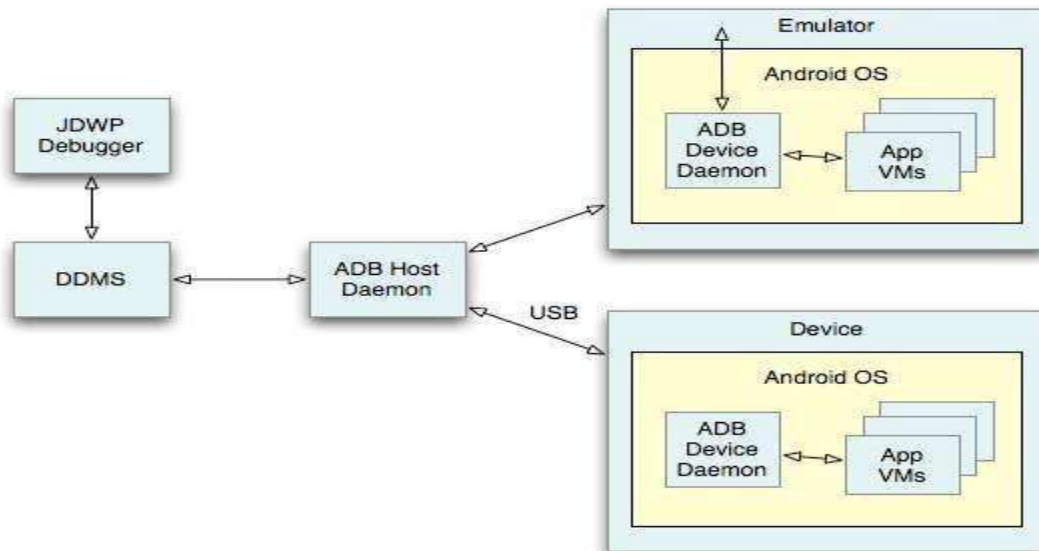
DDMS is a graphical program that communicates with your devices through adb. DDMS can capture screenshots, gather thread and stack information, spoof incoming calls and SMS messages, and has many other features.

Device or Android Virtual Device

Your application must run in a device or in an AVD so that it can be debugged. An adb device daemon runs on the device or emulator and provides a means for the adb host daemon to communicate with the device or emulator. cseitquestions.blogspot.in

cseitquestions.blogspot.in

JDWP debugger The Dalvik VM (Virtual Machine) supports the JDWP protocol to allow debuggers to attach to a VM. Each application runs in a VM and exposes a unique port that you can attach a debugger to via DDMS. If you want to debug multiple applications, attaching to each port might become tedious, so DDMS provides a port forwarding feature that can forward a specific VM's debugging port to port 8700. You can switch freely from application to application by highlighting it in the Devices tab of DDMS. DDMS forwards the appropriate port to port 8700. Most modern Java IDEs include a JDWP debugger, or you can use a command line debugger such as jdb.



DDMS Android ships with a debugging tool called the Dalvik Debug Monitor Server (DDMS), which provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more. This page provides a modest discussion of DDMS features; it is not an exhaustive exploration of all the features and capabilities. The following operation done using DDMS

- Viewing heap usage for a process
- Tracking memory allocation of objects
- Working with an emulator or device's file system
- Examining thread information
- Starting method profiling
- Network Traffic tool
- LogCat
- Emulating phone operations and location

Android App Development Life Cycle

Android application publishing is a process that makes your android applications available to users. Infact, publishing is the last phase of the android application development process.

1) Stage J (Journey)

Introduction

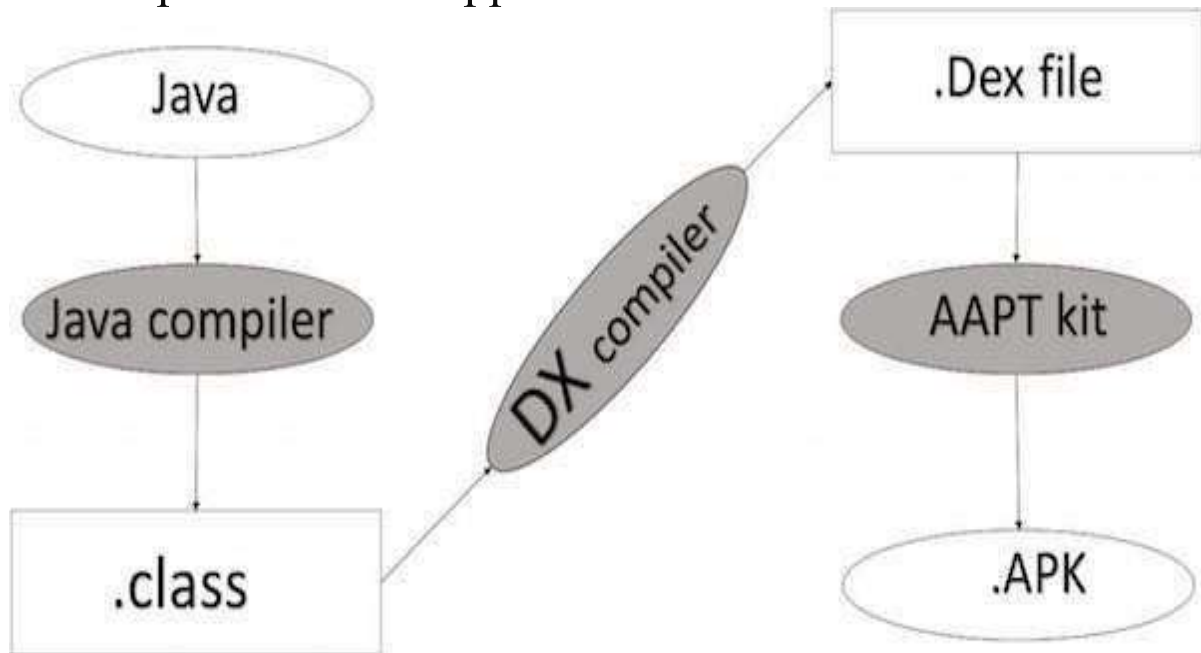
APK Conversion Process

Before exporting the apps, you must be aware of some of the tools like

- Dxtools(Dalvik executable tools): It going to convert .class file to .dex file. it hasuseful for memory optimization and reduce the boot-up speed time
- AAPT(Android assistance packaging tool):it has useful to convert .Dex file to.Apk
- APK(Android packaging kit): The final stage of deployment process is called as .apk.

You will need to export your application as an APK (Android Package) file before you upload it Google Play marketplace.

Export Android Application Process



To export an application, just open that application project in Android studio and select Build → Generate Signed APK from your Android studio and follow the simple steps to export your application.

Exporting (generating signed APK) your application from Android Studio for installing on mobile devices. Application is exported with apk extension.

App Publishing Guidance

To release your application to users you need to create a release-ready package that users can install and run on their Android-powered devices. The release-ready package contains the same components as the debug .apk file — compiled source code, resources, manifest file, and so on — and it is built using the same build tools. However, unlike the debug.apk file, the release-ready .apk file is signed with your own certificate and it is optimized with the zipalign tool.

You perform five main tasks to prepare your application for release. The signing and optimization tasks are usually seamless if you are building your application with Android Studio. For example, you can use Android Studio with the Gradle build files to compile, sign, and optimize your application all at once. You can also configure the Gradle build files to do the same when you build from the command line. For more details about using the Gradle build files, see the Build System guide.

To prepare your application for release you typically perform five main tasks (see figure 2). Each main task may include one or more smaller tasks depending on how you are releasing your application. For example, if you are releasing your application through Google Play you may want while you are configuring your application for release. Similarly, to meet Google Play publishing guidelines you may have to prepare screenshots and create promotional text while you are gathering materials for release.

You usually perform the tasks listed in figure 2 after you have thoroughly debugged and tested your application. The Android SDK contains several tools to help you test and debug your Android applications.

Step 1 Gathering Materials and Resources

To begin preparing your application for release you need to gather several supporting items. At a minimum this includes cryptographic keys for signing your application and an application icon. You might also want to include an end-user license agreement.

Cryptographic keys

The Android system requires that each installed application be digitally signed with a certificate that is owned by the application's developer (that is, a certificate for which the developer holds the private key). The Android system uses the certificate as a means of identifying the author of an application and establishing trust relationships between applications. The certificate that you use for signing does not need to be signed by a certificate authority; the Android system allows you to sign your applications with a self-signed certificate.

Important: Your application must be signed with a cryptographic key whose validity period ends after 22 October 2033.

Application Icon Be sure you have an application icon and that it meets the recommended icon guidelines. Your application's icon helps users identify your application on a device's Home screen and in the Launcher window. It also appears in Manage Applications, My Downloads, and elsewhere. In addition, publishing services such as Google Play display your icon to users. Note: If you are releasing your application on Google Play, you need to create a high resolution version of your icon. See [Graphic Assets for your Application](#) for more information. **End-user License Agreement** Consider preparing an End User License Agreement (EULA) for your application. A EULA can help protect your person, organization, and intellectual property, and we recommend that you provide one with your application. **Miscellaneous Materials** You might also have to prepare promotional and marketing materials to publicize your application. For example, if you are releasing your application on Google Play you will need to prepare some promotional text and you will need to create screenshots of your application.

Step 2 Configuring Your Application for Release

After you gather all of your supporting materials you can start configuring your application for release. This section provides a summary of the configuration changes we recommend that you make to your source code, resource files, and application manifest prior to releasing your application. Although most of the configuration changes listed in this section are optional, they are considered good coding practices and we encourage you to implement them. In some cases, you may have already made these configuration changes as part of your development process. **Choose a good package name** Make sure you choose a package name that is suitable over the life of your application. You cannot change the package name after you distribute your application to users. You can set the package name in application's manifest file. For more information, see the [package attribute documentation](#).

Also, you should remove all Debug tracing calls that you added to your code, such as `startMethodTracing()` and `stopMethodTracing()` method calls.

Important: Ensure that you disable debugging for your app if using WebView to display paid for content or if using JavaScript interfaces, since debugging allows users to inject scripts

and extract content using Chrome DevTools. To disable debugging, use the `WebView.setWebContentsDebuggingEnabled()` method.

Clean up your project directories

Clean up your project and make sure it conforms to the directory structure described in Android Projects. Leaving stray or orphaned files in your project can prevent your application from compiling and cause your application to behave unpredictably. At a minimum you should do the following cleanup tasks:

- Review the contents of your `jni/`, `lib/`, and `src/` directories. The `jni/` directory should contain only source files associated with the Android NDK, such as `.c`, `.cpp`, `.h`, and `.mk` files. The `lib/` directory should contain only third-party library files or private library files, including prebuilt shared and static libraries (for example, `.so` files). The `src/` directory should contain only the source files for your application (`.java` and `.aidl` files). The `src/` directory should not contain any `.jar` files.
- Check your project for private or proprietary data files that your application does not use and remove them. For example, look in your project's `res/` directory for old drawable files, layout files, and values files that you are no longer using and delete them.
- Check your `lib/` directory for test libraries and remove them if they are no longer being used by your application.
- Review the contents of your `assets/` directory and your `res/raw/` directory for raw asset files and static files that you need to update or remove prior to release.

Correct App Code

Caution: Modifying code in the debugger window impacts future app behavior in the emulator; however it does not modify actual source code, so make your source code modifications in the DEVELOP tab.

1. Close the Developer Tools floating window.
2. If necessary, click the DEVELOP tab.
3. In the file tree, open the `www/js` folder and choose the `debug.js` file.
4. Scroll to near line 46 and comment out `var breakDebugger = invalidVariable;>`
5. Choose File>Save.

Retest App Functionality

Now try clicking the Bad button in the Intel XDK Emulator floating window. Better? If not, click the the Reload App icon on the toolbar and try again.

More Handy Information

☐ What you are actually debugging is a simulation of a real mobile device using the Chromium desktop engine augmented with Cordova* and Intel XDK APIs. This simulation is designed to provide an idea of how your app will render on various devices and form factors. Some visual aspects of your app may render differently on real devices, especially if the real devices have an old OS version. ☐

□ Using the built-in version of the CDT debugger, you can set breakpoints, single-step, display variables in your JavaScript* code, do full DOM debugging, and see the effects of CSS on the DOM. You also have access to the CDT JavaScript* console, where you can view your app console.log messages and interact with your app JavaScript* context by manually inspecting properties and executing methods.

2) Stage a1 (apply)

Lab Activities:

Activity 1:

Run Your App on a Real Mobile Device

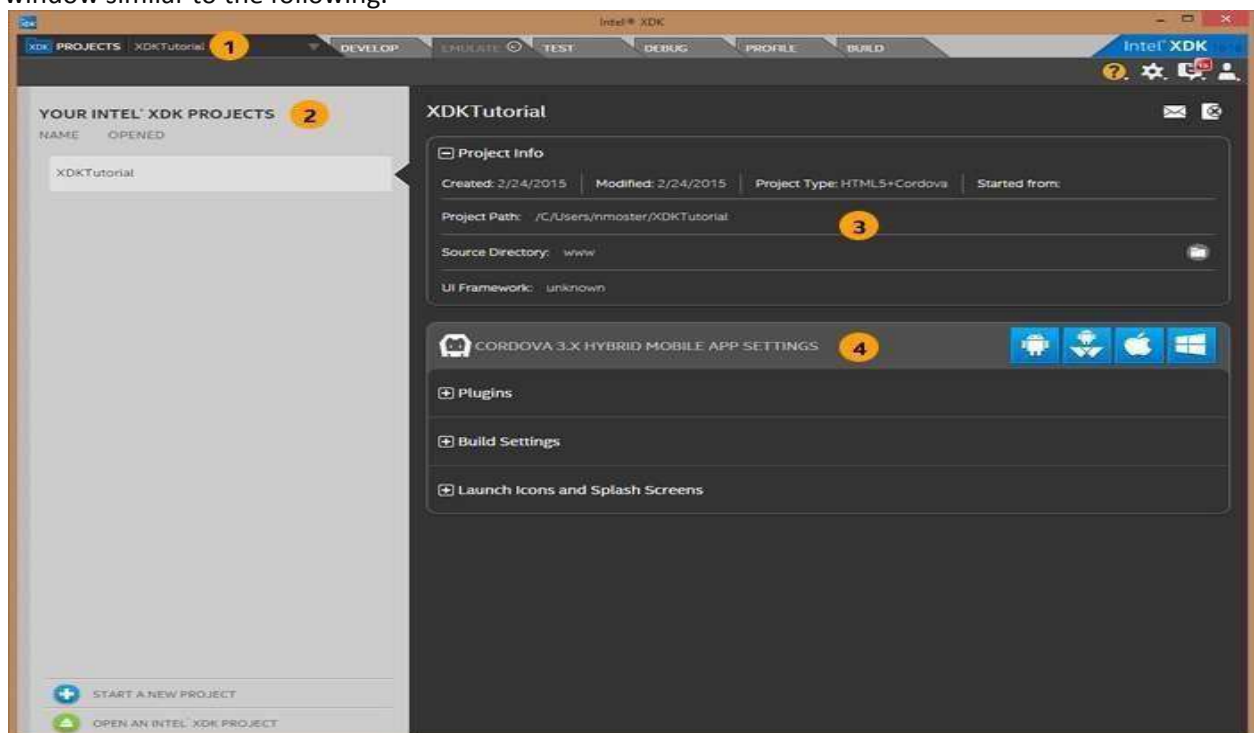
Now that you are confident your *XDKTutorial* app works properly on virtual devices, it is time to run it on a real mobile device.

1. Click the TEST tab.
2. Click the I HAVE INSTALLED APP PREVIEW button - because you already did this, right? ;-)
3. Your development machine displays a Please sync with our testing server message the first time you click the TEST tab for a project. Click the DISMISS button so we can first explore the TEST tab.

Package Options

Check Your App Package Options

Now that you are confident the *XDKTutorial* app works properly on a real mobile device, it is time to build your app. But let's check your app package options first. Click the PROJECTS tab to display a window similar to the following.



If the build fails, click the link to review a more detailed build log.

The Intel XDK does not provide actual store submission services for your app; however, it does explain a variety of possible next steps (which we obviously will not perform for this *XDKTutorial*app). Click the **Close Build Page** button.

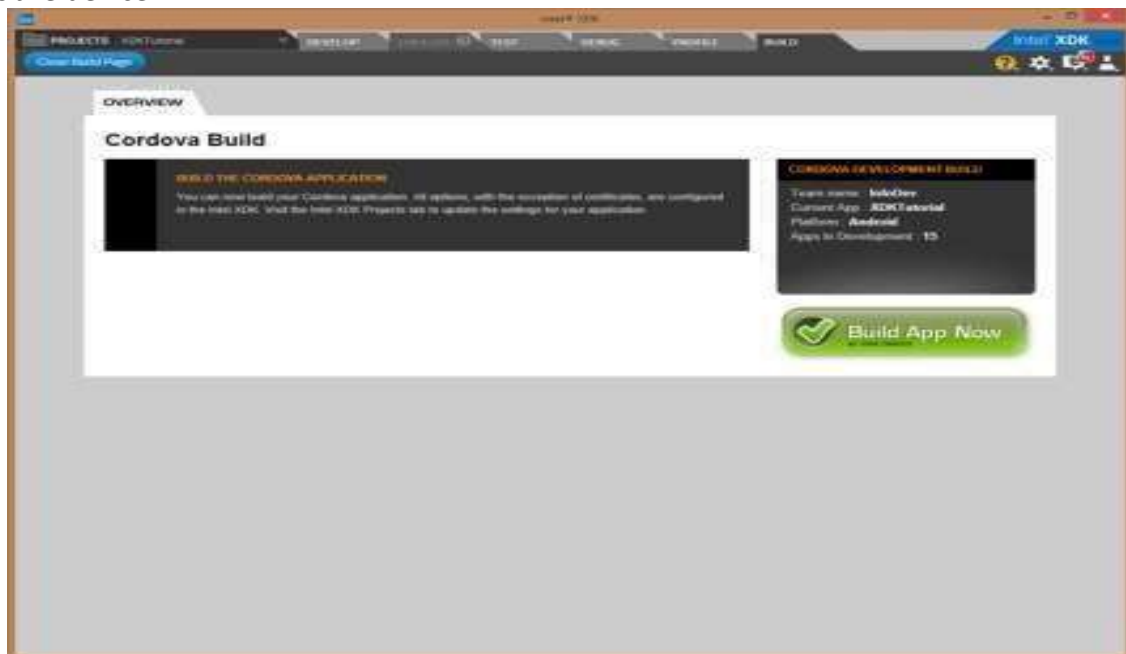
More Handy Information

☐ Builds are performed in the cloud, so you do not need to install and configure an SDK or native development system for each target platform (as you must do if you are building a Cordova*/ PhoneGap* app). ☐

☐ You must obtain the proper developer credentials to submit apps to most app stores. ☐

☐ With the exception of Crosswalk for Android* packages, all packages use the built-in webview (embedded browser) that is part of the target mobile device firmware to execute (render) your app. ☐

For example, Android* packages use the Android* browser webview built into the Android* mobile device, and iOS* packages use the Apple Safari* browser webview built into the iOS* mobile device.



3) Stage v (verify)

Home Activities:

Activity 1:

Upload your semester project (error free) as an app for free download.