PES INSTITUTE OF TECHNOLOGY (BSC)
V MCA, Second IA Test, September 2019
Mobile Applications (17MCA53)
Solution Set

Faculty: Jeny Jijo

1) Explain any 4 View groups/Layouts supported by Android. ***[8]***

The AbsoluteLayout enables you to specify the exact location of its children

```
<?xml version=”1.0” encoding=”utf-8”?>
    <AbsoluteLayout
            android:layout_width=”fill_parent”
            android:layout_height=”fill_parent”
    xmlns:android=http://schemas.android.com/apk/res/android
    <Button
            android:layout_width=”188dp”
            android:layout_height=”wrap_content”
            android:text=”Button”
            android:layout_x=”126px”
            android:layout_y=”361px”
    />
<Button
    android:layout_width=”113dp”
    android:layout_height=”wrap_content”
    android:text=”Button”
    android:layout_x=”12px”
    android:layout_y=”361px”
/>
    </AbsoluteLayout>
<?xml version=”1.0” encoding=”utf-8”?>
<RelativeLayout
    android:id=”@+id/RLayout”
    android:layout_width=”fill_parent”
    android:layout_height=”fill_parent”
xmlns:android=http://schemas.android.com/apk/res/android>
<TextView
    android:id=”@+id/lblComments”
    android:layout_width=”wrap_content”
    android:layout_height=”wrap_content”
    android:text=”Comments”
    android:layout_alignParentTop=”true”
    android:layout_alignParentLeft=”true”
<EditText
    android:id=”@+id/txtComments”
    android:layout_width=”fill_parent”
    android:layout_height=”170px”
    android:textSize=”18sp”
    android:layout_alignLeft=”@+id/lblComments”
    android:layout_below=”@+id/lblComments”
    android:layout_centerHorizontal=”true”
/>
<Button
```

```
        android:id=”@+id/btnSave”
        android:layout_width=”125px”
        android:layout_height=”wrap_content”
        android:text=”Save”
        android:layout_below=”@+id/txtComments”
        android:layout_alignRight=”@+id/txtComments”
    <Button
        android:id=”@+id/btnCancel”
        android:layout_width=”124px”
        android:layout_height=”wrap_content”
        android:text=”Cancel”
        android:layout_below=”@+id/txtComments”
        android:layout_alignLeft=”@+id/txtComments”
    />
    </RelativeLayout>
```

Notice that each view embedded within the RelativeLayout has attributes that enable it to align with another view. These attributes are as follows:

layout_alignParentTop
layout_alignParentLeft
layout_alignLeft
layout_alignRight
layout_below
layout_centerHorizontal

2) What are the units of measurements to manage the components of screen? [8]

When specifying the size of an element on an Android UI, you should be aware of the following units of measurement:
dp — Density-independent pixel. 1 dp is equivalent to one pixel on a 160 dpi screen. This is the recommended unit of measurement when specifying the dimension of views in your layout. The 160 dpi screen is the baseline density assumed by Android. You can specify either “dp” or “dip” when referring to a density-independent pixel.
sp — Scale-independent pixel. This is similar to dp and is recommended for specifying font sizes.
pt — Point. A point is defi ned to be 1/72 of an inch, based on the physical screen size.
px — Pixel. Corresponds to actual pixels on the screen. Using this unit is not recommended, as your UI may not render correctly on devices with a different screen resolution.

Android defines and recognizes four screen densities:

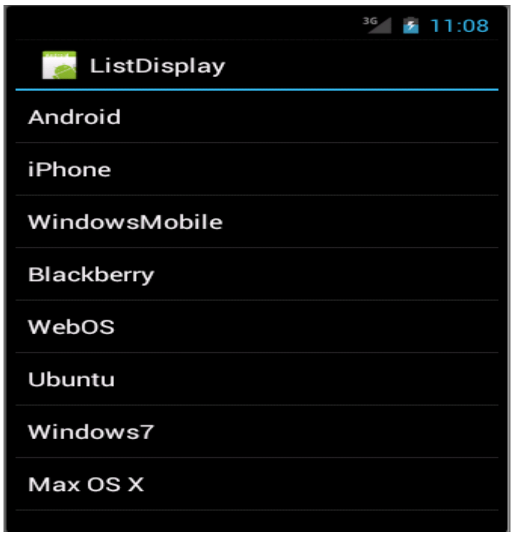
- Low density (*ldpi*) — 120 dpi
- Medium density (*mdpi*) — 160 dpi
- High density (*hdpi*) — 240 dpi
- Extra High density (*xhdpi*) — 320 dpi

3) Explain the difference List view and Spinner view with suitable example [8]

ListView View

The ListView displays a list of items in a vertically scrolling list. Android **List View** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.

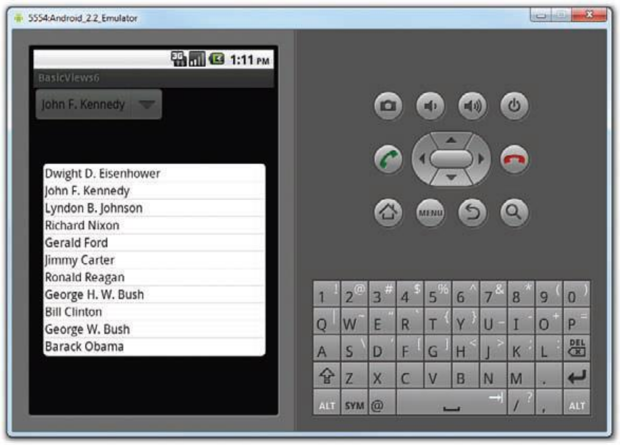
Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc. The **List View** and **Grid View** are subclasses of **Adapter View** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.


11:08
ListDisplay
Android
iPhone
WindowsMobile
Blackberry
WebOS
Ubuntu
Windows7
Max OS X

```
<ListView
                    android:id=”@+id/lstWords”
                    android:layout_width=”fill_parent”
                    android:divider=”#ddd”
                    android:dividerHeight=”1px”
                    android:paddingBottom=”67dp”
                    android:layout_height=”fill_parent” />
static final String[] words = new String[]{ “Hello”, “World” };
          ArrayAdapter<String>  itemsAdapter  =  new  ArrayAdapter<String>(this,
R.id.list_content,words));
          ListView  lstWords = (ListView)findViewById(R.id.lstWords);
          lstWords.setAdapter(itemsAdapter);
```

SpinnerView

The ListView displays a long list of items in an activity, but sometimes you may want your user interface to display other views, and hence you do not have the additional space for a full-screen view like the ListView. In such cases, you should use the SpinnerView. The SpinnerView displays one item at a time from a list and enables users to choose among them.

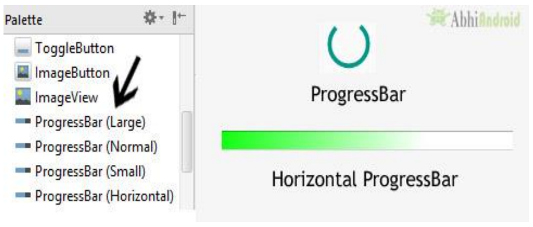


Dwight D. Eisenhower
John F. Kennedy
Lyndon B. Johnson
Richard Nixon
Gerald Ford
Jimmy Carter
Ronald Reagan
George H. W. Bush
Bill Clinton
George W. Bush
Barack Obama

4) Explain Progress Bar view in Android with suitable code. ***[8]***

The ProgressBar view provides visual feedback of some ongoing tasks, such as when you are performing a task in the background.

The default mode of the ProgressBar view is indeterminate that is, it shows a cyclic animation. This mode is useful for tasks that do not have a clear indication of when they will be completed, such as when you are sending some data to a web service and waiting for the server to respond



A Thread class together with a Runnable Object is used to simulate performing some long-running tasks .

The run() method starts the execution of the thread, which in this case calls the doSomeWork() method to simulate doing some work. When the simulated work is done (after about five seconds), you use a Handler object to send a message to the thread to dismiss the ProgressBar

The style of progressbar can be made horizontal by the attribute style in xml file

style="?android:attr/progressBarStyleHorizontal"

5 Design an application that contains Phone Contacts in vertical linear manner. Selected contact appears at the top of the list with a large italicized font and a blue background. [8]

Refer lab pgm1

6) Mention and describe the steps involved for managing changes to screen orientation. [8]

Android supports two screen orientations: *portrait* and *landscape*. By default, when you change the display orientation of your Android device, the current activity that is displayed automatically redraws its content in the new orientation. This is because the onCreate() method of the activity is fi red whenever there is a change in display orientation.

In general, you can employ two techniques to handle changes in screen orientation:
➤ **Anchoring** — The easiest way is to "anchor" your views to the four edges of the screen. When the screen orientation changes, the views can anchor neatly to the edges
➤ **Resizing and repositioning** — Whereas anchoring and centralizing are simple techniques to ensure that views can handle changes in screen orientation, the ultimate technique is resizing each and every view according to the current screen orientation.

Resizing and Repositioning

Apart from anchoring your views to the four edges of the screen, an easier way to customize the UI based on screen orientation is to create a separate res/layout folder containing the XML fi les for the UI of each orientation.
To support landscape mode, you can create a new folder in the res folder and name it as layout-land (representing landscape). Figure 3-23 shows the new folder containing the fi le main.xml. Basically,

the main.xml fi le contained within the layout folder defines the UI for the activity in portrait mode, whereas the main.xml fi le in the layout-land folder defi nes the UI in landscape mode.

7) Describe the three specialized fragments you can use in your Android application. [8]

Fragments are really "mini-activities" that have their own life cycles. Using fragments, we can customize the user interface of our Android application by dynamically rearranging fragments to fit within an activity. This enables us to build applications that run on devices with different screen sizes.

To create a fragment, a class is required that extends the Fragment base class. Besides the Fragment base class, we can also extend from some other subclasses of the Fragment base class to create more specialized fragments.
Three subclasses of Fragment:

- ListFragment,
- DialogFragment,
- PreferenceFragment.

ListFragment

- A list fragment is a fragment that contains a ListView, displaying a list of items from a data source such as an array or a Cursor.
- A list fragment is very useful, as you may often have one fragment that contains a list of items (such as a list of RSS postings), and another fragment that displays details about the selected posting. To create a list fragment, you need to extend the ListFragment base class.

DialogFragment

A dialog fragment floats on top of an activity and is displayed modally. Dialog fragments are useful for cases in which you need to obtain the user's response before continuing with execution. To create a dialog fragment, you need to extend the DialogFragment base class.

```
public class Fragment1 extends DialogFragment
{
static Fragment1 newInstance(String title)
        {
        Fragment1 fragment = new Fragment1();
        Bundle args = new Bundle();
        args.putString("title", title);
        fragment.setArguments(args);
        return fragment
        }
```

The newInstance() method allows a new instance of the fragment to be created, and at the same time it accepts an argument specifying the string (title) to display in the alert dialog. The title is then stored in a Bundle object for use later.

PreferenceFragment

To create a list of preferences in your Android application, we first need to create the preferences .xml file and populate it with the various XML elements. This XML file defines the various items that we want to persist in our application.
To create the preference fragment, we need to extend the PreferenceFragment base class:

```
public class Fragment1 extends PreferenceFragment
{
        }
```

To load the preferences file in the preference fragment, the addPreferencesFromResource() method is used

```
@Override
public void onCreate(Bundle savedInstanceState)
{
        super.onCreate(savedInstanceState);
//---load the preferences from an XML file---
        addPreferencesFromResource(R.xml.preferences);
}
```

All the operation that are made by the **FragmentManager** happens inside a "transaction" (for that Android fragment transaction) like in a database operation. First, we can get the **FragmentManger** using the Activity method **getFragmentManager()**. Once we have the reference to this component we have to start the transaction in this way

To display the preference fragment in our activity, we can make use of the FragmentManager and the FragmentTransaction classes:

```
        FragmentManager fragmentManager = getFragmentManager();
        FragmentTransaction fragmentTransaction =
        fragmentManager.beginTransaction();
        Fragment1 fragment1 = new Fragment1();
        fragmentTransaction.replace(android.R.id.content, fragment1);
        fragmentTransaction.addToBackStack(null);
        fragmentTransaction.commit();
```

The preference fragment should be added back to the stack using the addToBackStack() method so that the user can dismiss the fragment by clicking the back button.

8) Write in detail how to display Google maps in your own Android Application [8]

- ***Google Maps*** is one of the many applications bundled with the Android platform. We need to apply for a free Google Maps API key before you can integrate Google Maps into your Android application
- You can verify the existence of the debug certificate by going to Eclipse and selecting Window ➪ Preferences. Expand the Android item and select Build
- The filename of the debug keystore is debug.keystore. This is the certificate that Eclipse uses to sign your application so that it may be run on the Android Emulator or devices. need to extract its MD5 fingerprint using the Keytool.exe application included with your JDK installation. This fingerprint is needed to apply for the free Google Maps key.

Issue the following command to extract the MD5 fingerprint:
keytool.exe -list -alias androiddebugkey -keystore "C:\Users\<username>\.android\debug.keystore" - storepass android -keypass android

- MD5 fingerprint is EF:7A:61:EA:AF:E0:B4:2D:FD:43:5E:1D:26:04:34:BA. Copy the MD5 certificate fingerprint and navigate your web browser to: http://code.google.com/android/maps-api-signup.html. Follow the instructions on the page to complete the application and obtain the Google Maps key.
- Modify your AndroidManifest.xml file by adding both the <uses-library> element and the INTERNET permission.
- Add the MapView element to your UI.

```
<com.google.android.maps.MapView
        android:id="@+id/mapView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:enabled="true"
        android:clickable="true"
        android:apiKey="<YOUR KEY>" />
    <uses-library android:name="com.google.android.maps" />
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
```

- In order to display Google Maps in your application, you first need to have the INTERNET permission in your manifest file. You then add the <com.google.android.maps.MapView> element to your UI file to embed the map within your activity. Very importantly, your activity must now extend the MapActivity class, which itself is an extension of the Activity class. For the MapActivity class, you need to implement one method: **isRouteDisplayed()**. This method is used for Google's accounting purposes, and is returned true for this method if routing information is displayed on the map.

```
import com.google.android.maps.MapActivity;
public class MainActivity extends MapActivity {
/** Called when the activity is first created. */
@Override
    public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
  }
@Override
    protected boolean isRouteDisplayed() {
// TODO Auto-generated method stub
    return false;
    }
}
```

9) Explain the steps in signing your application to connect to Google Play. [8]

Publishing Android Applications

In order to get our application running on users' devices, we need a way to deploy it and distribute it.

Preparing For Publishing

The steps to publishing your Android application generally involve the following:

1. Export your application as an APK (Android Package) file.
2. Generate your own self-signed certificate and digitally sign your application with it.
3. Deploy the signed application.
4. Use the Android Market for hosting and selling your application.

Versioning Your Application

The AndroidManifest.xml file of every Android application includes the android:versionCode and android:versionName attributes:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android= "http://schemas.android.com/apk/res/android"
                                                    package="net.learn2develop.LBS"
  android:versionCode="1"
  android:versionName="1.0" >
<uses-sdk android:minSdkVersion="14" />
```

The android:versionCode attribute represents the version number of your application.To differentiate between the newest version and old one, value 1 can be incremented. This attribute is used by Android Market to determine whether a newer version of your application is available.

The value of android:versionCode attribute can be retrieved by using the **getPackageInfo()** method from the **PackageManager** class programmatically

```
private void checkVersion() {
PackageManager pm = getPackageManager();
try {
        //---get the package info---
PackageInfo pi = pm.getPackageInfo("net.learn2develop.LBS", 0);
        //---display the versioncode---
Toast.makeText(getBaseContext(),"VersionCode: " +Integer.toString(pi.versionCode),
                                        Toast.LENGTH_SHORT).show();
}
catch (NameNotFoundException e)
{
// TODO Auto-generated catch block
e.printStackTrace();
} }
```

The **android:versionName** attribute contains versioning information that is visible to users. It should contain values in the format *<major>.<minor>.<point>*.

If your application undergoes a major upgrade, you should increase the *<major>* by 1. For small incremental updates, you can increase either the *<minor>* or *<point>* by 1.

All Android applications must be digitally signed before they are allowed to be deployed onto a device (or emulator). A self-signed certificate can be generated and use it to sign our Android applications.

We can verify this by going to Windows ➪Preferences in Eclipse, expanding the Android item, and selecting Build .Eclipse uses a default debug keystore (appropriately named "debug.keystore") to sign your application.

A keystore is commonly known as a ***digital certificate***.

If you are publishing an Android application, you must sign it with your own certificate.

Applications signed with the debug certificate cannot be published Using the keytool.exe utility provided by the Java SDK, Eclipse makes it easy for you by including a wizard that walks you through the steps to generate a certificate. It will also sign your application with the generated certificate

10) Explain deployment of APK files in Android. ***[8]***

After you have signed your APK files, you need a way to get them onto your users' devices.

Three methods are covered:

➤ Deploying manually using the adb.exe tool

➤ Hosting the application on a web server

➤ Publishing through the Android Market

Besides these methods, you can install your applications on users' devices using e-mail, an SD card, and so on. As long as you can transfer the APK file onto the user's device, the application can be installed.

Using the adb.exe Tool

Once our Android application is signed, we can deploy it to emulators and devices using the adb.exe (Android Debug Bridge) tool (located in the platform-tools folder of the Android SDK).

Using the command prompt in Windows, navigate to the

<Android_SDK>\platform-tools folder.

To install the application to an emulator/device (assuming the emulator is currently up and running or a device is currently connected), give the following command:

```
adb install "C:\Users\Wei-Meng Lee\Desktop\LBS.apk"
```

Another way to deploy an application is to use the DDMS tool in Eclipse. With an emulator (or device) selected, use the File Explorer in DDMS to go to the /data/app folder and use the "Push a file onto the device" button to copy the APK file onto the device.

If you are planning to publish your application on the Android Market (www.android.com/ market/), the AndroidManifest.xml file must have the following attributes:

➤ android:versionCode (within the <manifest> element)

➤ android:versionName (within the <manifest> element)

➤ android:icon (within the <application> element)

➤ android:label (within the <application> element)

The android:label attribute specifies the name of our application. This name is displayed in the Settings ➪ Apps section of our Android device.

****