iCreate 2013
Android Training

By: Uday Athreya (citusa@nus.edu.sg)

Prerequisite

- A Basic Understanding of **Java**
- Experience in working with **Eclipse**
- The training nature of this session will be

  - 80% Practical

  - 20% Theory

  Hence require laptops with Eclipse and Android SDK installed.

Agenda

- Android Basics (Getting Started)
- Application Fundamentals
- Activity Life Cycle
- Debugging
- Basic UI Elements (Layouts, Views, Buttons, Dialogs)
- Using Google Maps
- Publishing Apps
- Using LAPI

In this session there are **7 practice exercises** to get you familiar with the SDK. To download solutions for the exercises, click this link.

Lets Start!!

Android Basics

Installing the Android SDK:

- For windows:
  http://www.cit.nus.edu.sg/icreate/Setting%20up%20Android%20Development%20Environment.pdf
  Alternatively download the entire bundle (SDK and Eclipse) using this link

- For Detailed instructions:
  http://developer.android.com/sdk/installing.html

- Things to be installed to get started:
  1. Eclipse IDE for Java Developers
  2. Java Platform *(JDK 7)*
  3. Android SDK Tools

Why Android?

- Android is open source.
- Popularity in smart phone market
- Coding language used is Java.
- Apk singing procedure is simple.
- Market subscription fee is only $25
- Alternate markets available.
- Tools and Documentation

Why Android?

	Android	Apple	Windows
Devices	Hundreds	11	20
No. of Apps	800,000+	775,000+	90,000+
No. of developers	71%	62%	27%
Developer Subscription	Free SDK. Only $25 for publishing apps.	Free IOS SDK. $99 annual subscription	Free VS Express SDK. $99 annual subscription
Coding Language	Java	Objective C	C# and VB
Platforms Supported	Windows, Mac and Linux	Mac	Windows

Features in Android

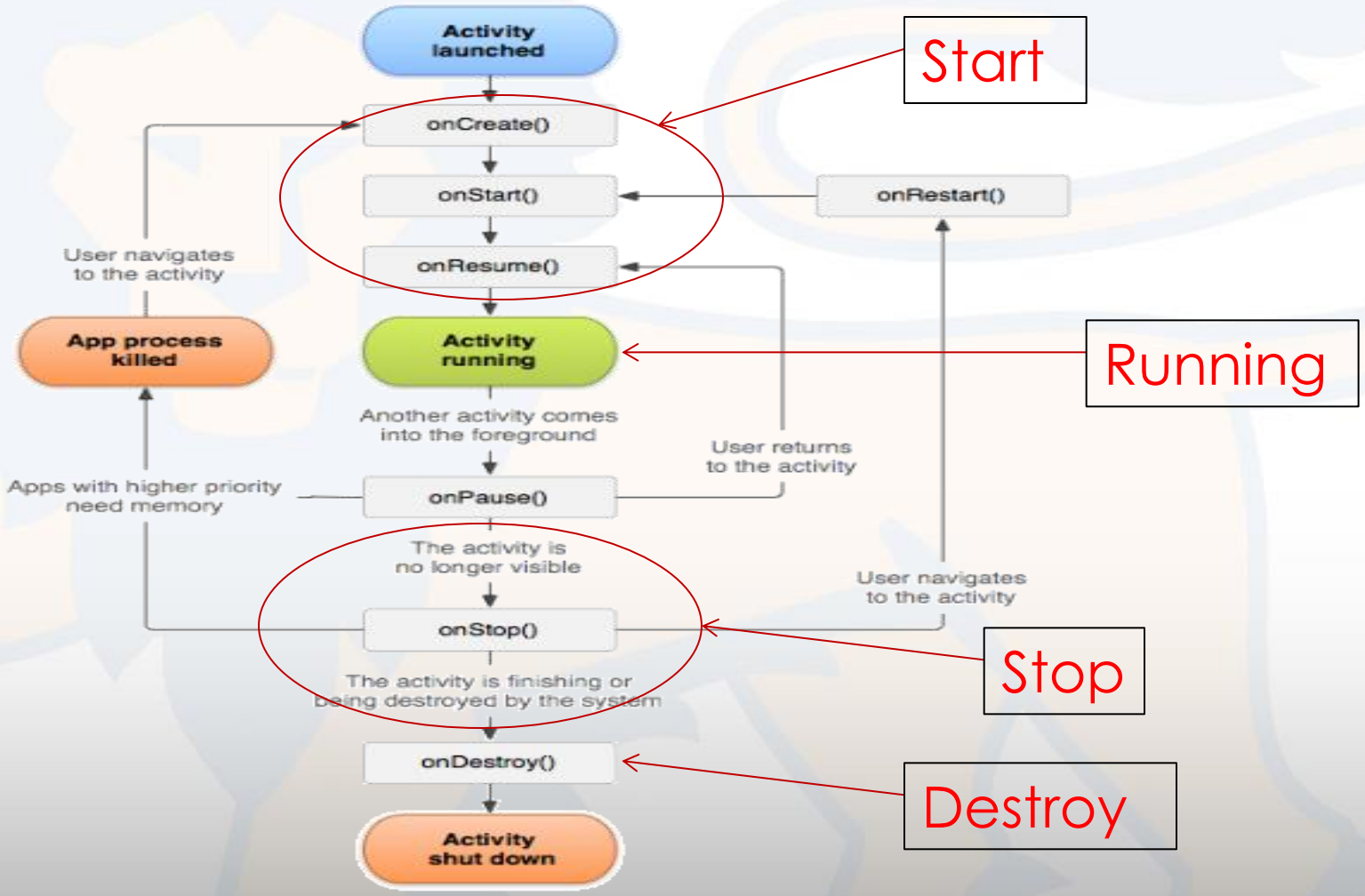
- User Interface: IO Widgets (lists, textboxes, buttons, images)
- Google Maps
- Hardware APIs: GPS/Geo-location, Calls, Accelerometer, Compass, Bluetooth, Camera
- Multiple processes
  - Managed by Android Dalvik VM
  - Background Services
  - Interprocess communications (e.g. Intents)
- Databases
- Application Life Cycle

Application Fundamentals

- Basic Definitions:

1. **Activity**: An activity is a screen or form that is presented to the user.
   For example: an email application might have one activity that shows a list of new emails, another activity to compose an email.
2. **Application**: is a collection of activities.
3. **Services**: A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. **A service does not provide a user interface.**
4. **Content providers**: A content provider manages a shared set of application data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access.
5. **Broadcast receivers**: A broadcast receiver is a component that responds to system-wide broadcast announcements.
6. **Intent**: An intent is an abstract description of an operation to be performed. It's an "intention" to do an action. **Usually used with startActivity**.

Activity Life Cycle



Activity Life Cycle

- The activity class has the following method callbacks to manage the app:
  - onCreate()
  - onStart()
  - onResume()
  - onPause()
  - onStop()
  - onRestart()
  - onDestroy()

- To use a callback just override it in your activity java file.

- For more info on how to use callback refer:
  http://developer.android.com/reference/android/app/Activity.html

- The lifecycle is explained very well here:
  http://developer.android.com/videos/index.html#v=fL6gSd4ugSI

Debugging

- Dalvik Debug Monitor Server (DDMS): Android ships with a debugging tool which provides port-forwarding services, screen capture on the device, thread and heap information on the device, **logcat**, process and so on.
- To run DDMS in Eclipse click Window > Open Perspective > Other... > DDMS.
- To add log information in your code, use the Log Api to send log output. Generally, use the Log.v() Log.d() Log.i() Log.w() and Log.e() methods.

  Eg: Log.d("Testing variable a: ", a.toString());
- To view the log output, open DDMS and check the logcat.

Exercise -1: HelloWorld

Build a sample HelloWorld application:

- Task 1: Create a new project
  1. Open Eclipse. Click on File -> New -> Android Project
  2. Enter project name as HelloWorld, click Next
  3. Select the checkbox next to Android 2.1, click Next
  4. Enter the package name as com.hello.world, click Finish

- Start the emulator
  1. Open the Android Virtual Device Manager
  2. Select one of the virtual devices listed, click Start.
  3. Check “Scale Display to real size” and enter Screen Size 7. Click Launch.
  4. Wait for the device to start up.

- Run the application
  1. Expand the src folder of your app and open the HelloWorldActivity.java file.
  2. Click the Run button or Ctrl – F11 keyboard shortcut.
  3. Check if the app is installed in the emulator and is running.
  4. Examine the folder structure of the app and various files in each of the folders.

Exercise -2: Toast Widget

Add a toast message to the HelloWorld application:

- Task 1: Open the HelloWorld project
  1. Open Eclipse. Click on File -> Import -> Android Project
  2. Browse to your folder containing the project , click Next

- Task 2: add the Toast Widget
  1. Open the HelloWorldActivity.java file
  2. Add the following lines after setContentView();

```
Context context = getApplicationContext();
CharSequence text = "Hello World ☺";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, text, duration);
toast.show();

or

Toast.makeText(context, text, duration).show();
```

- Run the application

Exercise -2: Toast Widget

- Additional Tasks:

1. Override the 8 Activity call back methods. Add Log information in each of these methods and experiment with different scenarios when these methods are called.

2. Notice that by default the toast message is displayed at the bottom of the activity. Your task is to make the Toast message to be displayed at a selected location of your choice. (Hint: Use the Gravity class and toast.setGravity(..))

Basic UI Elements

- UI of an Android application is typically defined in an XML file located in /res/layout folder.
- Each element in the XML file is compiled into its equivalent Android GUI class during the compile time.
- To load the XML UI to an activity class, use the setContentView() method. This is when the android system creates the UI.

```
@Override
  public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
  }
```

- Every activity has a **view**. One or more views grouped together are known as ViewGroups.
- ViewGroups provides a layout where views can be sequenced. We call ViewGroups as **layouts**.

Basic UI Elements - Layouts

- Android supports the following layouts:

- **LinearLayout**: arranges views in a single column or single row

- **AbsoluteLayout**: specify the exact location of the view

- **TableLayout**: group views in row and column

- **RelativeLayout**: position views relative to each other

- **FrameLayout**: placeholder on screen that you can use to display a single view

- **ScrollView**: special view which enables scroll through a list of views.

Basic UI Elements - Layouts

Example layout with a view:

```
<?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" >

   <TextView
    android:id="@+id/textHello"
    android:layout_width= "wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/hello" />

</LinearLayout>
```

- From personal experience, use **Relativelayouts** to keep the UI consistent across various phone resolutions.

Basic UI Elements - Views

- **Form Widgets**: **TextView**, **Button**, Checkbox, Radio Button, Spinner, Progress Bar etc.
- **Text Fields**: **Plain Text**, Email, Phone, **Password**, Address etc.
- **Composite**: **ListView**, Expandable List, GridView, ScrollView, WebView etc.
- **Images and Media**: **Image View, Image Button**, Gallery, Media Controller, Video View
- **Time & Date**: Time Picker, Date Picker, Chronometer, Analog Clock.
- **Others**: Zoom Control, SurfaceView, View, requestFocus

Exercise -3: Login App

Build a sample application to simulate Login screen:

- Task 1: Create a new project
  1. Open Eclipse. Click on File -> New -> Android Project
  2. Enter project name as Login, click Next
  3. Select the checkbox next to Android 2.1, click Next
  4. Enter the package name as com.my.login, click Finish

- Task 2: Create the UI
  1. Open the main.xml under /res/layout.
  2. Change the layout in the xml to RelativeLayout.
  3. From the pallete, drag a **TextView** and drop it on the screen at a suitable position.
  4. Change the `android:text` attribute of the Textview to `android:text="User id: "`
  5. Drag **PlainText** widget listed under TextFields to the right of the above textview.
  6. Open the xml and add an attribute to the Plaintext as `android:hint="User id".`
  7. Similarly add another **TextView** and **Password** widget below the user id.
  8. Drag a Button widget from the palette and place it below Password. Change button text to `android:text="Login".`
  9. Save the xml file.

Exercise -3: Login App

- Task 3: Add code to LoginActivity.java
  1. Open LoginActivity.java under /src/com.my.login/
  2. You can already see an onCreate() method with setContentView().
  3. Declare the following variables for the LoginActivity class:

```
private String username = "john";
private String pass = "123";
private EditText userid, password;
private Button login;
```

Note: You will have to import EditText and Button widget from android.widget, otherwise Eclipse will show you an error. Highlight the error for quick fixes and import the necessary packages.

  4. Assign variables to their respective views. Add this code below setContentView().

```
userid = (EditText) findViewById(R.id.editText1);
password = (EditText) findViewById(R.id.editText2);
login = (Button) findViewById(R.id.button1);
```

  5. Add an onClickListener to the login button and check if the user entered login id and password match with the variables username and pass.

Exercise -3: Login App

```
login.setOnClickListener(new OnClickListener() {

    public void onClick(View arg0) {
        // TODO Auto-generated method stub
        if(userid.getText().toString().equals(username) &&
    password.getText().toString().equals(pass)){
        startActivity(new Intent("com.my.login.NewActivity"));
        }
    }
});
```

6. If the above condition is met launch a new activity. Before that, create another activity. Steps to create an activity:
   - In the package explorer right click on the package /Login/src/com.my.login, click New and select Class. Enter the class name as “NewActivity” (without quotes) and click Finish.
   - Create a new layout for the “NewActivity”. Right click the layout folder (/Login/layout), click New and select Android XML file. Enter name as “newlayout” and click Finish. Add a textView with text “Logged in”.
   - Add the onCreate and setContentView methods to NewActivity as follows:

Exercise -3: Login App

```
public class NewActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.newlayout);
    }
}
```

7. Let Android know about this activity. In the AndroidManifest.xml, add these lines inside the application tag:

```
<activity android:name="NewActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="com.my.login.NewActivity" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
</activity>
```

8. Open any of the java files and run the project.

Exercise -3: Login App

- Additional Task:

1. Add a TextView below the Login button with text as “Login not successful”. The Textview should be shown only upon click of Login button and when the user login credentials are wrong. (Hint: setVisibility attribute).

2. Add a Reset button next to Login button. Upon clicking this button, it should reset the user id and password fields to “”.

WebView

- Is an extension of android's view class.
- Allows you to build an hybrid app (native UI with web components).
- Allows to display a web page as part of an activity.
- Webview component renders views using **Webkit.**
- A useful feature of Webview is you can create interfaces from your JavaScript to your client-side Android code.
- Two things to note before starting to use Webviews:
  - WebViews do not support JavaScript by default. To enable javascript add the following piece of code:

```
webview.getSettings().setJavaScriptEnabled(true);
```

  - Permission to use internet needs to be added in the Android.manifest xml as `<uses-permission android:name="android.permission.INTERNET" />` to avoid "Webpage not available error".

Exercise -4: WebView

Build a sample application to display Web pages using Webviews:

Part1: Display any webpage

- Task 1: Create a new project
  1. Open Eclipse. Click on File -> New -> Android Project
  2. Enter project name as Webview, click Next
  3. Select the checkbox next to Android 2.1, click Next
  4. Enter the package name as com.my.webview, click Finish

- Task 2: Create the UI
  1. Open the main.xml under /res/layout.
  2. Change the layout in the xml to RelativeLayout.
  3. From the pallete, add two Buttons. Change button texts to `android:text="Load Webpage"` and `"Load Local Web page"`
  4. Save the xml file.

- Task 3: Add code to WebviewActivity
  1. Declare a private variables to access Buttons as `private Button button1, button2;`
  2. Assign variables to their respective views. Add this code below setContentView().

```
button1 = (Button) findViewById(R.id.button1);
button2 = (Button) findViewById(R.id.button2);
```

Exercise -4: WebView

3. Add an onclickListener to both the button1

```
button1.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
            // TODO Auto-generated method stub
            Web_flag =1;
            startActivity(new Intent("com.my.webview.Webview"));
    }
});
```

4. Create a new activity called as Webview.
   - In the package explorer right click on the package /Login/src/com.my.webview, click New and select Class. Enter the class name as “Webview” (without quotes) and click Finish.
   - Create a new layout for “Webview”. Right click the layout folder (/Login/layout), click New and select Android XML file. Enter name as “webview” and click Finish.
   - Modify the xml code to:

```
<?xml version="1.0" encoding="utf-8"?>
<WebView  xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webView1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

Exercise -4: WebView

- Add the onCreate and setContentView methods to Webview activity as follows:

```
public class Webview extends Activity {
    /** Called when the activity is first created. */
          private WebView webView;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.webview);
        webView = (WebView) findViewById(R.id.webView1);
        if(WebviewActivity.Web_flag == 1){
          webView.loadUrl("http://www.nus.edu.sg");
        }
    }
}
```

Note: Remember to add the Webview activity in the AndroidManifest.xml.

5. Open the java file and run the app.

Exercise -4: WebView

Part 2: Display a local webpage

- Task 1: Create a local html webpage (name it as default.html) and copy it in the assets folder of the Webview project.

- Task 2: Add an onclickListener to button2 in WebViewActivity.

```
button2.setOnClickListener(new OnClickListener() {

      @Override
      public void onClick(View v) {
                // TODO Auto-generated method stub
                        Web_flag =2;
                        startActivity(new Intent("com.my.webview.Webview"));

      }
   });
```

- Task 3: Add code to Webview activity to open the local webpage. Add an else condition to the if statement in the Webview activity as follows:

```
else{
      webView.loadUrl("file:///android_asset/default.html");
}
```

- Open any java file and run the app.

Exercise -4: WebView

Part 3: Bind Javascript to Android Code

- Task 1: Create a local html webpage (name it as default.html) and copy it in the assets folder of the Webview project. (Copy the default.html from the solutions).

- Task 2: In the Webview activity enable the Javascript for the webview:

```
webView.getSettings().setJavaScriptEnabled(true);
```

- Task 3: Add a Javascript Interface to the webview:

```
webView.addJavascriptInterface(new JSInterface(this), "myjsinterface");
```

- Task 4: Create a class called as JSInterface with a simple function to display a Toast.

```
public class JSInterface{
 Context mContext;
    /** Instantiate the interface and set the context */
    JSInterface(Context c) {
        mContext = c;
    }

  public void showToast() {
        Toast.makeText(mContext, "Local page loaded", Toast.LENGTH_SHORT).show();
    }

 }
```

Exercise -4: WebView

- Task 5: Accessing the showToast method from default.html
  - Add a button to the html:

```
<input type="button" value="Say hello‟ onClick="showAndroidToast()" />
```

  - The Android showToast() method can be accessed from the html page via a javascript as follows:

```
<script type="text/javascript">
    function showAndroidToast() {
        myjsinterface.showToast();
    }
</script>
```

- Task 6: Open any java file and run the app. Clicking the button in local page should display a toast message.

Exercise -4: WebView

- Additional Task:

1. Add an image tag on your local html page. The source of the image should be from your phone. Write another method (public String img_path()) in the JSInterface class. This method should return the path of a image on the phone. Call this method in the javascript of the html page and display the image on page load.

2. Display the links inside your webview. Suppose you display a webpage in your webview and that page has weblinks, clicking on the web link should not open the browser, but display in your webview. (Hint: Refer this link).

ListView

- A view that shows items in a vertically scrolling list. The items come from the **ListAdapter** is associated with this view.
- This is similar to Swing's Jlist.
- The difference between list and table view is that each row of listview is selectable.
- A ListView's adapter, holds all the list's underlying data as well as the Views necessary to render each row of the ListView.
- **If you want rows to have a different look than the stock rows, supply your own layout XML to be used for each row and over ride the getView() method. (Refer this link).**

Exercise -5: ListView

Build a sample application to display List of items in a listview:

- Task 1: Create a new project
  1. Open Eclipse. Click on File -> New -> Android Project
  2. Enter project name as Listview, click Next
  3. Select the checkbox next to Android 2.1, click Next
  4. Enter the package name as com.my.listview, click Finish

- Task 2: Create the UI
  1. Open the main.xml under /res/layout.
  2. Change the layout in the xml to RelativeLayout.
  3. From the pallete, add a ListView (under Composite).
  4. Save the xml file.

- Task 3: Add code to ListviewActivity
  1. Declare a private variables to access ListView as

```
private ListView list;
```

  2. Assign variables to their respective views. Add this code below setContentView().

```
list = (ListView) findViewById(R.id.listView1);
```

Exercise -5: ListView

3. Setting the listview adapter.
   - Declare a private String[] array.

```
String[] items = {"India", "Singapore", "China", "SriLanka", "Japan",
                  "Malaysia", "Bangladesh", "Indonesia", "Bhutan", "Hong Kong"};
```

   - Set the listview adapter using ArrayAdapter<T>

```
list.setAdapter(new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1,
items));
```

   - Add an onItemclick event on listview to do something when user clicks a row.

```
list.setOnItemClickListener(new OnItemClickListener() {

            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,
                                    long arg3) {
                // TODO Auto-generated method stub
                Toast.makeText(ListViewActivity.this, "Clicked on: "+items[arg2],
Toast.LENGTH_SHORT).show();
            }
        });
```

- Task 4: Save the java file and run the app.

Exercise -5: ListView

- Additional Tasks:

Create a Listview which has a custom layout for each row. Each row should have an image and text. (Hint: Refer the tutorial link shown earlier, write your own custom adapter, xml layout for each row and use the LayoutInflater to inflate the view).

Your output should should similar to the screenshot shown below:

Dialogs and Menus

- A dialog is usually a small window that appears in front of the current Activity.
- The underlying Activity loses focus and the dialog accepts all user interaction.
- Dialog class is the base class for creating any dialogs. However most of the times we use one of the sub classes. In this session we will play around with AlertDialog and ProgressDialog.

  1. AlertDialog: As the name suggests, when used pops up as an alert to the user.

  2. Progress Dialog: Usually used as a message to tell the user to wait as the system is performing some calculations (Eg: Login Screens, loading images etc).

- To show a dialog on screen use the .show() method and to close the dialog use the .dismiss() method.

Dialogs and Menus

- Menu is another UI component which is widely used.
- Android phones with versions lesser than 3.0 have a dedicated menu button from the **options menu** can be launched. Options menu appears at the bottom of the screen and it can **hold a maximum of 6 menu items**.
- Another type of menu which is mostly used is **Context menu**. This is a floating menu which appears upon a **long click** of an element.
- For any menu type, Android provides a standard XML where the menu and its items are defined in a menu resource.

Exercise -6: Dialogs and Menus

Build a sample application with dialogs and menu:

- Task 1: Create a new project
  1. Open Eclipse. Click on File -> New -> Android Project
  2. Enter project name as DialogsMenu, click Next
  3. Select the checkbox next to Android 2.1, click Next
  4. Enter the package name as com.my.dialogsmenu, click Finish

- Task 2: Create the UI
  1. Open the main.xml under /res/layout.
  2. Change the layout in the xml to RelativeLayout.
  3. From the pallete, add 3 buttons. Name them as “Alert Dialog”, “Progress Dialog”, “Context Menu”
  4. Save the xml file.

- Task 3: Add code to DialogsMenuActivity
  1. Declare a private variables to access the 3 Buttons, assign variables to their respective views and add onClickListener on each of them as done in the previous exercises.

Exercise -6: Dialogs and Menus

- Task 4: Display an Alert Dialog
  Add the following code inside the onclickListener of alert dialog button created in Task - 2.

```
AlertDialog.Builder builder = new AlertDialog.Builder(DialogsMenuActivity.this);
builder.setMessage("Are you sure you want to exit?")
   .setCancelable(false)
   .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
       public void onClick(DialogInterface dialog, int id) {
            DialogsMenuActivity.this.finish();
       }
   })
   .setNegativeButton("No", new DialogInterface.OnClickListener() {
       public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
       }
   });
AlertDialog alert = builder.create();
alert.show(); //dont forget to call this
```

Exercise -6: Dialogs and Menus

- Task 5: Display a Progress Dialog
  - To do this task you need to know basics of **Threading.**

  - Copy the Pi.java from the solutions folder into the source folder of the project. This class computes value of Pi upto 800 digits.

  - Add the following code inside the onclickListener of progress dialog button created in Task - 2 to start a new activity.

```
startActivity(new Intent("com.my.dialogsmenu.ProgressDialogActivity"));
```

  - Create a new activity called as ProgressDialogActivity and also a UI (progress_dialog.xml) with a textview.

  - Here we create the ProgressDialog using the static ProgressDialog.show() function, and as a result the pd variable is initialized.

```
pd = ProgressDialog.show(ProgressDialogActivity.this, "Working..", "Calculating Pi", true,
false);
```

Exercise -6: Dialogs and Menus

- Create a new Thread. In the run() of the thread, compute Pi, assign it to the variable pi_string and then send an empty message to our handler object.

```
Thread thread = new Thread(new Runnable (){
        @Override
        public void run() {
                // TODO Auto-generated method stub
                pi_string = Pi.computePi(800).toString();
                handler.sendEmptyMessage(0);
        }
});

thread.start();
```

- In the handler, **dismiss the progress dialog** and then display the value of pi in the textview.

```
private Handler handler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
            pd.dismiss();
            tv.setText("Value of Pi is: \n"+pi_string);
    }
};
```

-

Exercise -6: Dialogs and Menus

- Task 6: Display a Context Menu to delete an item from a list.
  - To do this task you will need to copy the ListviewActivity created in Exercise 5 to the source folder of this project.
  - Add the following code inside the onclickListener of context menu button created in Task – 2 to start the ListView activity.

```
startActivity(new Intent("com.my.dialogsmenu.ListViewActivity"));
```

  - Create a new layout (list.xml) with a listview for the ListViewActivity. Declare a global integer variable (pos_clicked) to hold the position of item clicked in the list. Add a long click listener on each item of the list

```
list.setOnItemLongClickListener(new OnItemLongClickListener() {
      @Override
      public boolean onItemLongClick(AdapterView<?> arg0, View arg1,
      int arg2, long arg3) {
                // TODO Auto-generated method stub
                pos_clicked = arg2;
                registerForContextMenu(list);
                return false;
      }
});
```

Exercise -6: Dialogs and Menus

- Create a context Menu by over-riding the onCreateContextMenu(…) method.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.setHeaderTitle("Actions");
    menu.add(0, v.getId(), 0, "Delete This Entry");
}
```

- To handle the menu item is clicked, override the onContextItemSelected(…) method

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    if(item.getTitle()=="Delete This Entry"){
            String new_arr[] = function1(item.getItemId());
            list.setAdapter(new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, new_arr));
    }else {
            return false;
    }
return true;
}
```

Exercise -6: Dialogs and Menus

- The function1(…) is used to remove item from the list.

```
@Override
public String[] function1(int id){
     if(items != null){
               List<String> list = new ArrayList<String>(Arrays.asList(items));
               list.remove(pos_clicked);
               return list.toArray(new String[0]);
     }
     else{
               return new String[0];
     }
}
```

Exercise -6: Dialogs and Menus

- Task 7: Display a Options Menu.
  - To do this task you will need to create a new folder named as “menu” in /res/ and an xml for the menu (menu.xml).
  - Menu root node consists of an item leaf node. Item consists of an id, icon and title.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/exit_icon"
    android:title="Exit"
    android:icon="@drawable/ic_launcher" />
</menu>
```

  - In the DialogsMenuActivity, create a optionsMenu by over-riding the onCreateOptionsMenu(…) method.

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
            MenuInflater inflater = getMenuInflater();
            inflater.inflate(R.menu.menu, menu);
            return true;
    }
```

Exercise -6: Dialogs and Menus

- To handle the menu item is clicked, use the onOptionsItemSelected(…) method

```
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
     case R.id.exit_icon:
     AlertDialog.Builder builder = new AlertDialog.Builder(DialogsMenuActivity.this);
     builder.setMessage("Are you sure you want to exit?")
     .setCancelable(false)
     .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int id) {
           DialogsMenuActivity.this.finish();
      }
     })
   .setNegativeButton("No", new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int id) {
           dialog.cancel();
      }
   });
    AlertDialog alert = builder.create();
    alert.show();
    break;
   }
   return true;
}
```

Using Google Maps API

- Android provides easy and tight integration of Google Maps to Android apps.
- In order to be able to use Google Maps, the Google APIs have to be present in your SDK. In case the Google APIs are not already installed, you will have to manually install them.
- Before we start writing our apps, we need to obtain Google Maps API key from the certificate's SHA1 finger print.

  - Locate the SDK debug certificate located in the default folder of "C:\Documents and Settings\<username>\Local Settings\Application Data\Android". The filename of the debug keystore is debug.keystore.
  - Extract the SHA1 fingerprint of the keystore using the keytool.exe. The keytool can be found in "C:\Program Files\Java\<JDK_version_number>\bin".
  - Open the DOS prompt and issue the command

```
keytool -list -v -keystore C:\debug.keystore"
```

  - Copy the SHA1 certificate fingerprint and navigate your web browser to: this link. Follow the instructions on the page to complete the application and obtain the Google Maps key.

Exercise -7: Google Maps

Build a sample application using Google Maps to
1. Display a location using Google Maps.
2. Navigate from one location to another.

- Task 1: Create a new project
  1. Open Eclipse. Click on File -> New -> Android Project
  2. Enter project name as GoogleMaps, click Next
  3. Select the checkbox next to Google APIs 2.1, click Next
  4. Enter the package name as com.my.googlemaps, click Finish

- Task 2: Create the UI
  1. Open the main.xml under /res/layout.
  2. Change the layout in the xml to RelativeLayout.
  3. From the pallete, add two Buttons. Change button texts to `android:text="Display Location"` and `"Navigate"`
  4. Save the xml file.

Exercise -7: Google Maps

- Task 3: Add code to GoogleMapsActivity
  1. Declare a private variables to access Buttons as `private Button button1, button2;`
  2. Assign variables to their respective views. Add this code below setContentView().

```
button1 = (Button) findViewById(R.id.button1);
button2 = (Button) findViewById(R.id.button2);
```

  3. Add an onclickListener to button1 and start an activity called Maps.

```
startActivity(new Intent("com.my.googlemaps.Maps"));
```

- Task 4: Adding Google Maps to Maps activity.
  1. Create a UI (maylayout.xml) and add a MapView to it

```
<com.google.android.maps.MapView
    android:id="@+id/myMap"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:clickable="true"
    android:apiKey="//your mapApiKey here" />
```

  2. Configure the Android.Manifest xml

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

Exercise -7: Google Maps

3. Add code to Maps activity

- Extend the activity to MapActivity and override the oncreate method. Set the contentView to `R.layout.maplayout`

- Declare a MapView object and assign it to the MapView. Also add a simple zoom feature

```
MapView mapView = (MapView) findViewById(R.id.mapview);
mapView.setBuiltInZoomControls(true);
```

4. Add markers and overlays.

- Create a new Java class named HelloItemizedOverlay that implements ItemizedOverlay.

- Right-click the package name in the Eclipse Package Explorer, and select New > Class. Fill-in the Name field as HelloItemizedOverlay. For the Superclass, enter "com.google.android.maps.ItemizedOverlay". Click the checkbox for Constructors from superclass. Click Finish.

Exercise -7: Google Maps

- Declare a Overlay Arraylist to hold all the OverlayItem objects

```
private ArrayList<OverlayItem> mOverlays = new ArrayList<OverlayItem>();
```

- The constructor calls default marker drawable for each overlayitem

```
public HelloItemizedOverlay(Drawable defaultMarker, Context context)
{
    super(boundCenterBottom(defaultMarker));
    mContext = context;
}
```

- To add new OverlayItems to the ArrayList use the addOverlay(..)

```
public void addOverlay(OverlayItem overlay)
{
    mOverlays.add(overlay);
    populate();
}
```

- populate() executes each time a new OverlayItem is added. This in turn calls createitem(..) method which retrieve each OverlayItem.

Exercise -7: Google Maps

- populate() executes each time a new OverlayItem is added. This in turn calls createitem(..) method which retrieves each OverlayItem.

```
@Override
protected OverlayItem createItem(int i)
{
     return mOverlays.get(i);
}
@Override
public int size()
{
     return mOverlays.size();
}
```

- To handle the tap or touch events and pop up a dialog override the onTap(..)

```
@Override
protected boolean onTap(int index)
{
     OverlayItem item = mOverlays.get(index);
     AlertDialog.Builder dialog = new AlertDialog.Builder(mContext);
     dialog.setTitle(item.getTitle());
     dialog.setMessage(item.getSnippet());
     dialog.show();
     return true;
}
```

Exercise -7: Google Maps

5. At the end of your existing onCreate() method do the following:
   - Instantiate :

```
List<Overlay> mapOverlays = mapView.getOverlays();
Drawable drawable = this.getResources().getDrawable(R.drawable.ic_launcher);
HelloItemizedOverlay itemizedoverlay = new HelloItemizedOverlay(drawable, this);
```

- Create a GeoPoint that defines the map coordinates for the first overlay item, and pass it to a new OverlayItem.

```
GeoPoint point = new GeoPoint(30443769,-91158458);
OverlayItem overlayitem = new OverlayItem(point, "Laissez les bon temps rouler!", "I'm in Louisiana!");
```

- Lastly add this overlay item to your collection.

```
itemizedoverlay.addOverlay(overlayitem);
mapOverlays.add(itemizedoverlay);
```

Exercise -7: Google Maps

6. To navigate from one location to another using Google Maps.
   - In the GoogleMapsActivity add an onclickListener to button2 and add the following code.

```
button2.setOnClickListener(new OnClickListener() {
   @Override
   public void onClick(View v) {
            // TODO Auto-generated method stub
            String start_latitude = "1.294887", start_longitude = "103.77369"; //SOC
            String end_latitude = "1.296394", end_longitude = "103.772832"; //Library
            String uri = String
            .format("http://maps.google.com/maps?saddr="
            + start_latitude + "," + start_longitude
            + "&daddr=" + end_latitude + ","
            + end_longitude);
            Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(uri));
            startActivity(intent);
   }
   });
```

Publishing your apps

- To publish (release) your app, you need to do the following:
  1. Obtain a suitable private key
  2. Compile the application in release mode
  3. Sign your application with your private key
  4. Align the final APK package

- To publish an app without Google Maps:
  1. Right click the project folder in Eclipse Package Explorer > Export > Export Android Application (under Android). Click Next.
  2. Create a new keystore. Enter the name as **"something".keystore** and password for the keystore and click Next.
  3. Enter Alias Name as alias_name, enter the password, validity as 25, your firstname. Click Next.
  4. Choose your location for apk file.
  5. Click Finish.

Publishing your apps

- To publish an app with Google Maps:
  1. We need to get the Maps key for our release keystore.
  2. Follow the same procedure as in slide 42, but enter the following command in the DOS prompt.

     ```
     keytool -list -alias alias_name -keystore C:\Users\citusa\Desktop\test.keystore
     ```

  3. 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.
  4. Replace this key in your mapslayout xml. Save it.
  5. Right click the project folder in Eclipse Package Explorer > Export > Export Android Application (under Android). Click Next.
  6. Use Existing keystore. Browse to the above keystore which you signed and enter password for the keystore and click Next.
  7. Enter Alias Name as alias_name, enter the password, Click Next.
  8. Choose your location for apk file.
  9. Click Finish

LAPI

- LAPI is API developed by the IVLE team. These API's can be used by the developers to integrate to tools on IVLE. For more details click here.
- The LAPI provides JSON and XML functionality.
- JSON is much simpler and user friendly than XML.
- All you need to do is to write a JSON parser to retrieve data. A simple JSON parser is shown here .

Some Useful Links

- Databases: http://www.vogella.com/articles/AndroidSQLite/article.html
- Google Maps: http://www.javacodegeeks.com/2011/02/android-google-maps-tutorial.html
- Camera: http://www.vogella.com/articles/AndroidCamera/article.html
- GPS: http://www.vogella.com/articles/AndroidLocationAPI/article.html , http://www.slideshare.net/androidstream/android-gps-tutorial
- Developer Guide: http://developer.android.com/guide/index.html
- Stack Overflow: http://stackoverflow.com/questions/tagged/android
- Few of my sample apps in the Google Play Store can be found here.

Thank you