Department of Information Engineering, CUHK
MScIE – 2nd Semester, 2015/16

IEMS 5722
Mobile Network Programming and Distributed Server Architecture

Lecture 2
Android Programming

Lecturer: Albert C. M. Au Yeung

21st January, 2016

The Android SDK

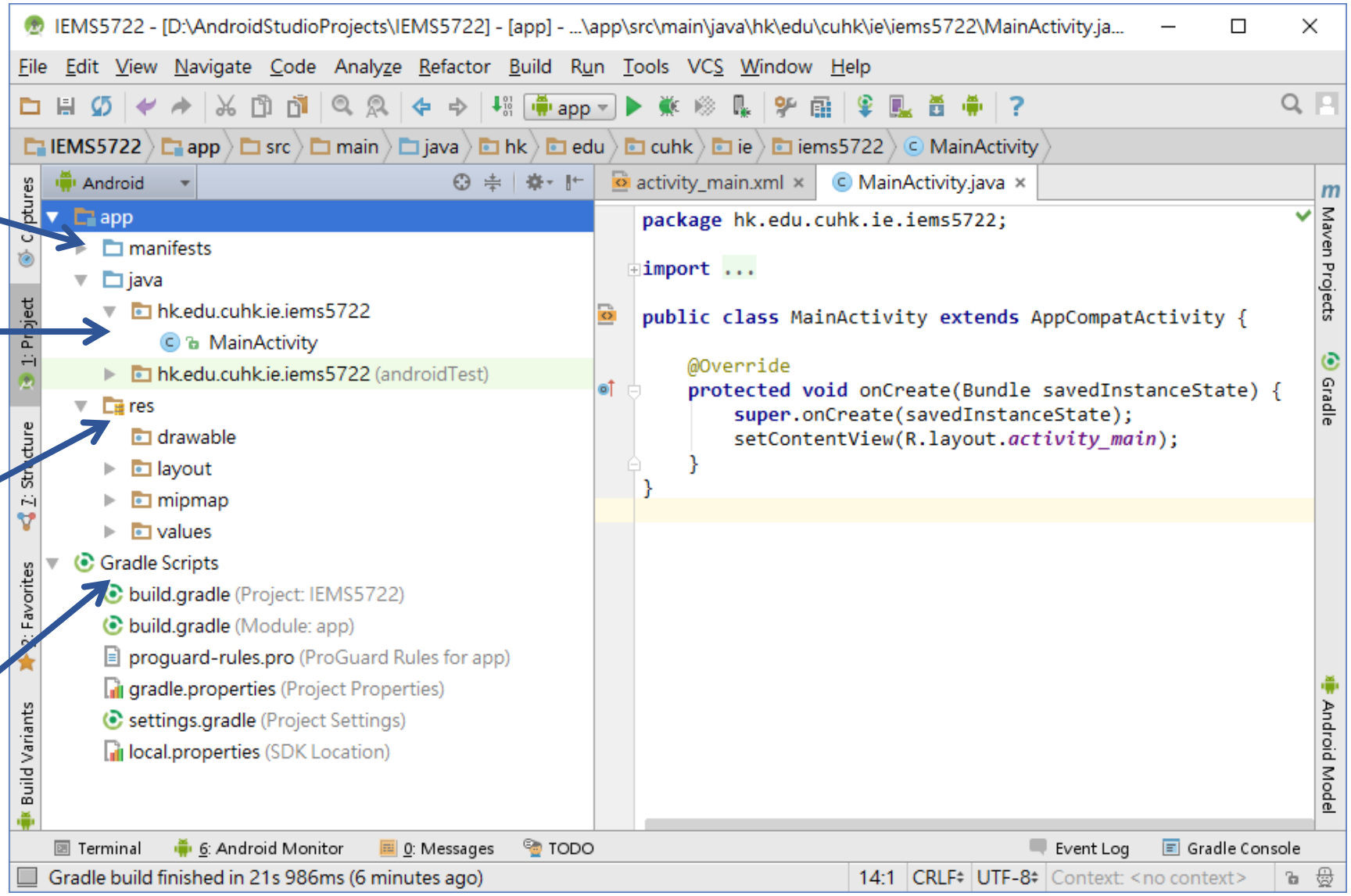
The Android SDK

Android Manifest File

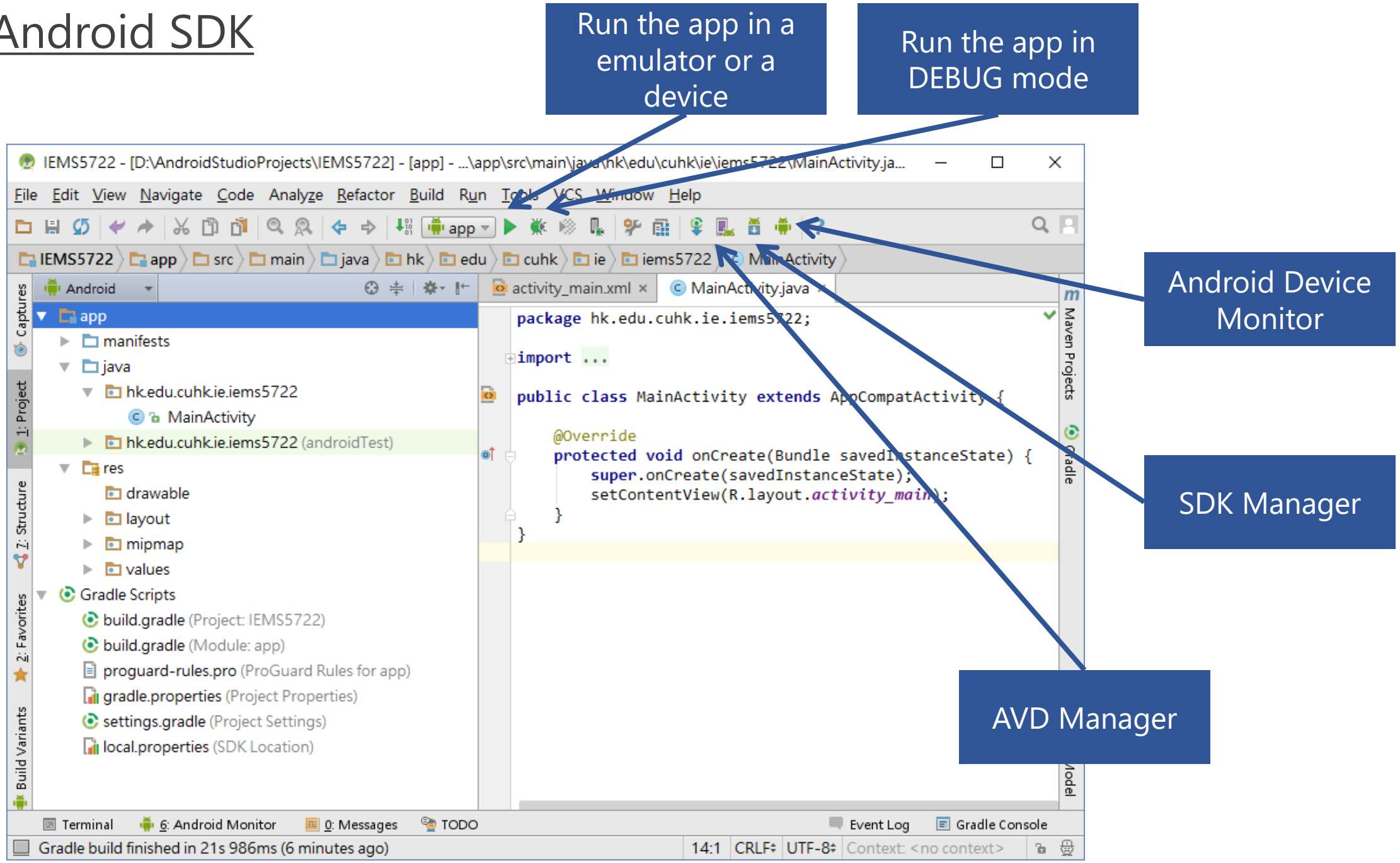
Java source codes

Resources (graphics, layout XML files, strings, colour values, etc.)

App building related settings



The Android SDK

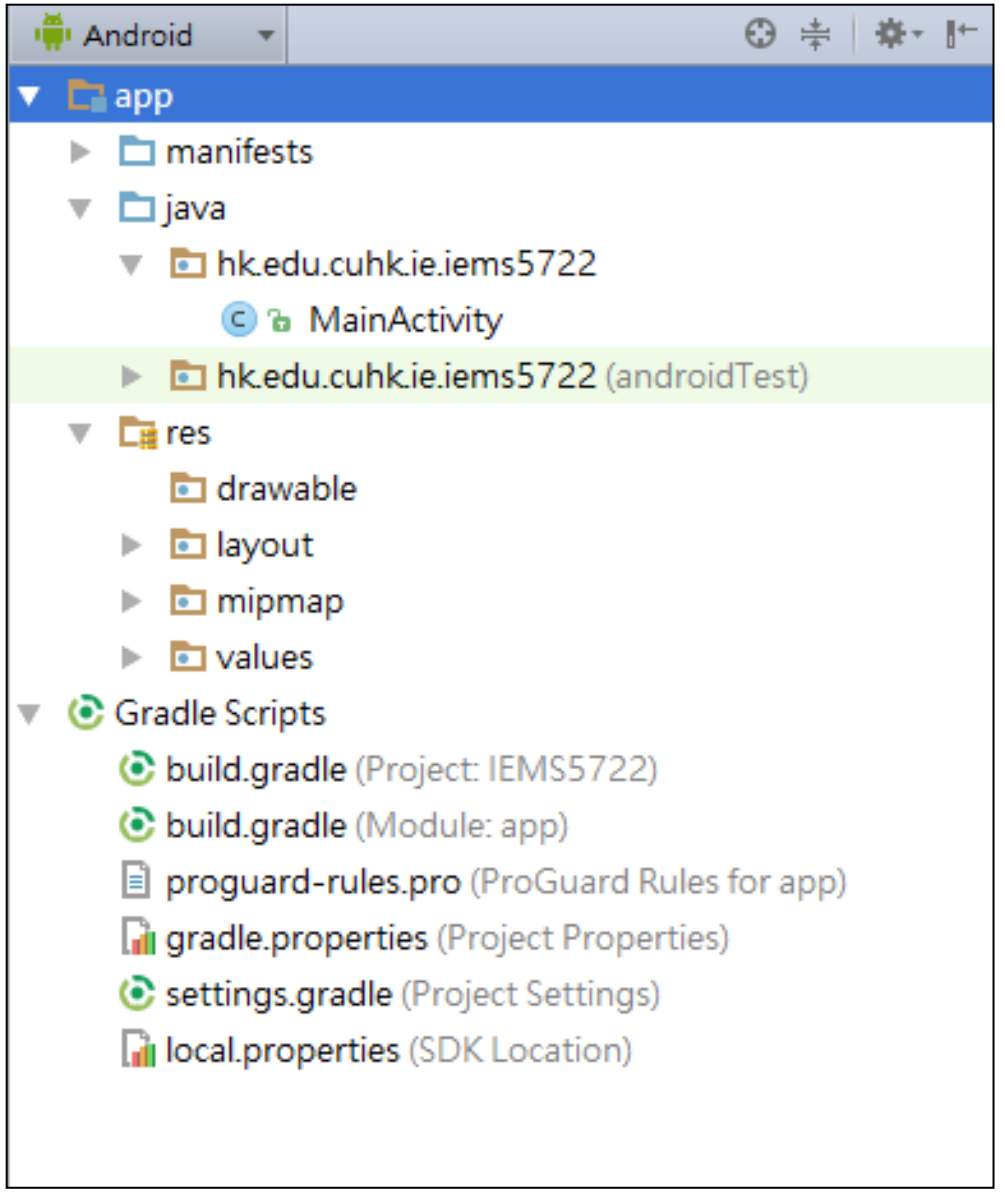


Android App Structure

App Structure

Each Android project consists of several types of files

- Android Manifest file
- Java source code
- Drawables
- Layout and menu files
- Colour list
- String / array resources
- Gradle configuration files



Android Manifest File

The AndroidManifest.xml File

- Must be present in every Android app
- Specify the following essential information about the app:
  - The Java package name
  - The components of the app (e.g. Activities and services in the app)
  - The permissions the app asks for from the user
  - Other information about the libraries the app is using

Reference: https://developer.android.com/guide/topics/manifest/manifest-intro.html

Android Manifest File (Example)

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="hk.edu.cuhk.ie.iems5722">

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

    <application
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/android:Theme.Holo" >
        <activity
            android:name=".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>
```

The package name of your Android app (a unique identifier in the Android app universe!)

Permissions that the user will grant your app for the app to work properly

You should specify here every activity that appears in your app before you can use them.

App Resources

Layouts – XML files that define the architecture for the UI of an activity or other components of a UI

```
<?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" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

App Resources

Drawables - A general concept for a graphic that can be drawn to the screen or apply to another XML resource with attributes such as `android:drawable` and `android:icon`

- Bitmap Files (PNG, JPG or GIF) (PNG are recommended)
- Layer List (An array of drawables)
- State List (Describes the different states of a drawable)
- Level List
- Shapes, Transitions, Scales, etc.

Common uses: Bitmap files or definitions of background shapes

App Resources

Colours – a file called ‘colors.xml’ stored under res/values, it defines the colour values that are used in the mobile app

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

    <color name="light_grey">#DDDDDD</color>
    <color name="dark_grey">#333333</color>
    <color name="button_highlight">#6633B5E5</color>

    <color name="tab_indicator_colour">#2C96DD</color>
    <color name="tab_background">#FFFFFF</color>

    ...
</resources>
```

App Resources

Strings – a file called 'strings.xml' stored under res/values, it defines the strings are used in the mobile app

- If you want your app to serve users using different languages, you can create different string files for different languages
- E.g. strings.xml (default), strings-fr.xml (French strings), strings-zh.xml (Chinese strings), …

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My App</string>
    <string name="yes">Yes</string>
</resources>
```

strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">My App</string>
    <string name="yes">是</string>
</resources>
```

strings-zh.xml

Assets

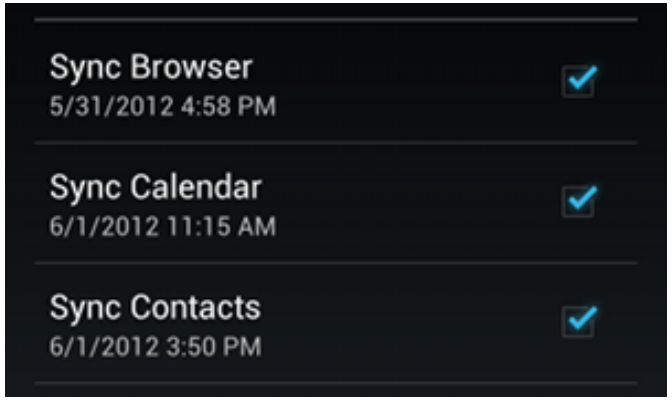
- Sometimes you might want to include some data files in your app so that you can use the data inside your app
- Create a folder called '**assets**' under app/src/main
- Examples of data files:
  - Text files
  - HTML files (e.g. for display in WebViews)
  - CSV files (e.g. for initialising a local database)

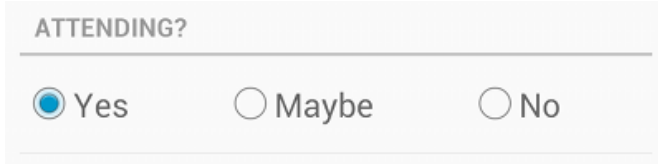
User Interface

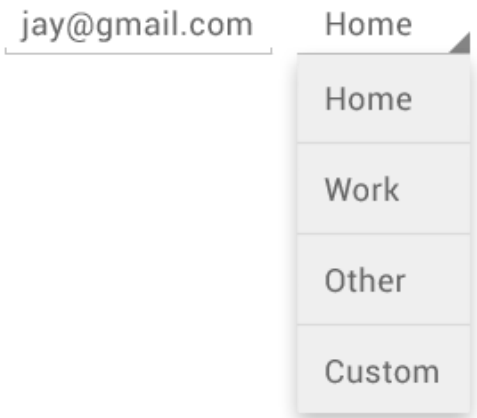
UI Components

Android offers many pre-defined UI components that you can use

- TextView
- EditText
- Button
- ImageView / ImageButton
- Checkbox
- Radio Button
- Toggle Button
- Spinner
- Picker







Layouts

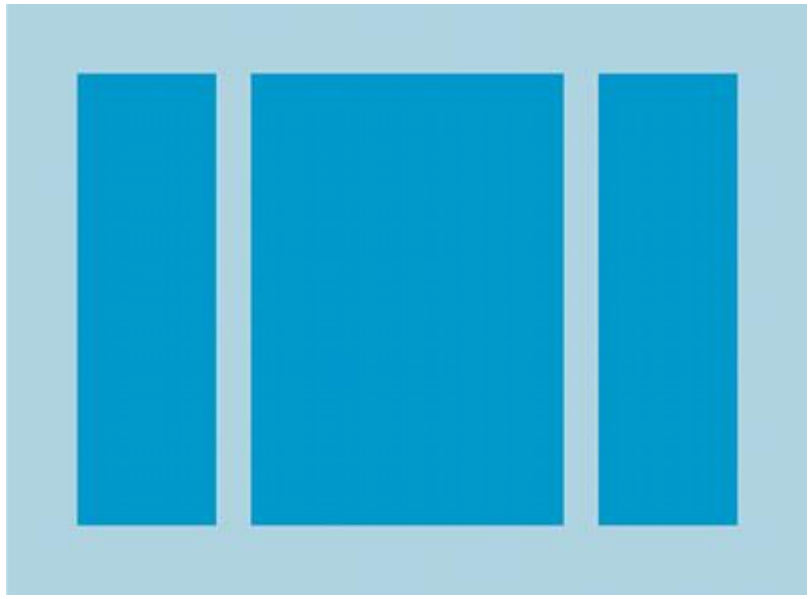
- Layouts defines the **visual structure** of the app
- Layouts can be declared in two different ways:
  1. Declare UI elements in an **XML file**
  2. Instantiate layout elements in the **Java code** (runtime)

- Two major layouts in Android:
  1. **Linear Layout**
  2. **Relative Layout**

Reference: http://developer.android.com/guide/topics/ui/declaring-layout.html

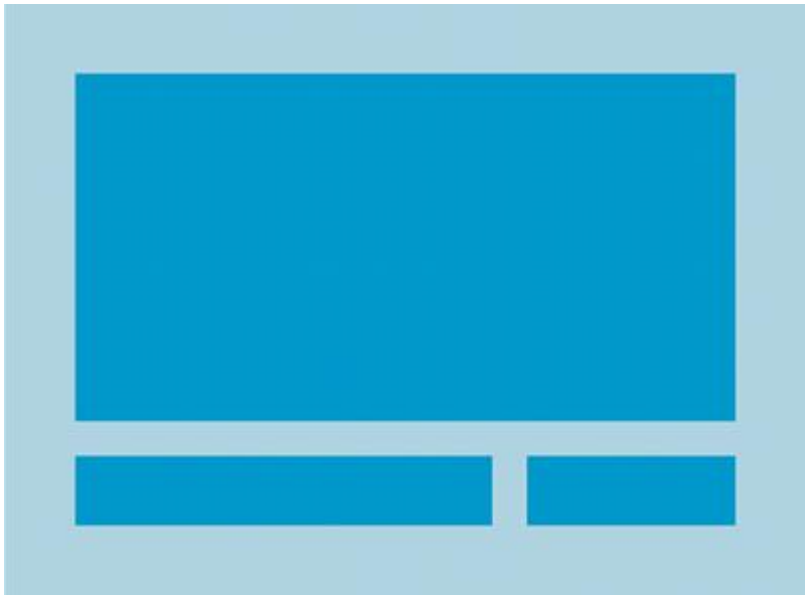
Layouts

Linear Layout



Linear Layout presents UI components one after another, either **vertically** or **horizontally**

Relative Layout



In Relative Layout, UI components are placed relative to the other components (e.g. Center in parent component, left to another component)

Reference: https://developer.android.com/guide/topics/ui/declaring-layout.html

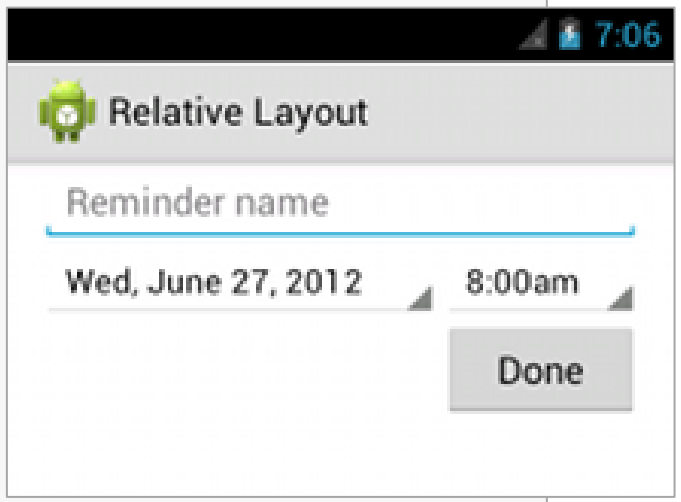
Layouts

```
<?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" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Reference: https://developer.android.com/guide/topics/ui/layout/linear.html

Layouts

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
...
```

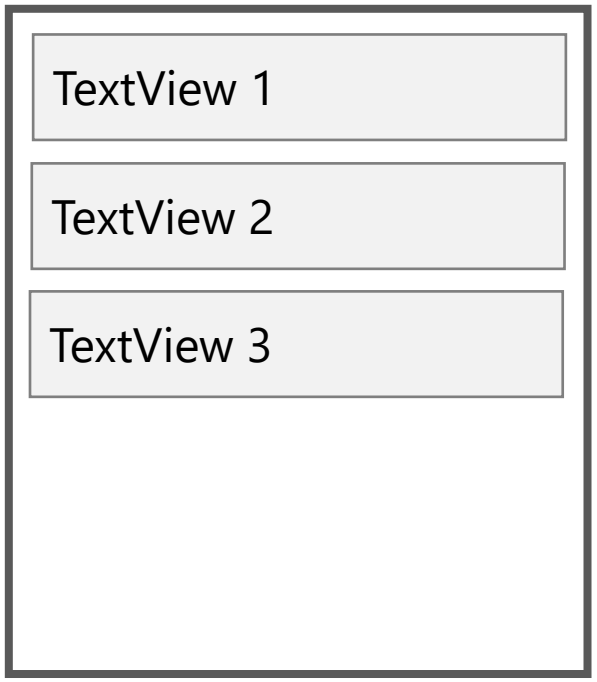


Reference: https://developer.android.com/guide/topics/ui/layout/relative.html

Layouts

Let's see some examples

- What if you want to put three Text Views vertically one after another inside a Linear Layout?

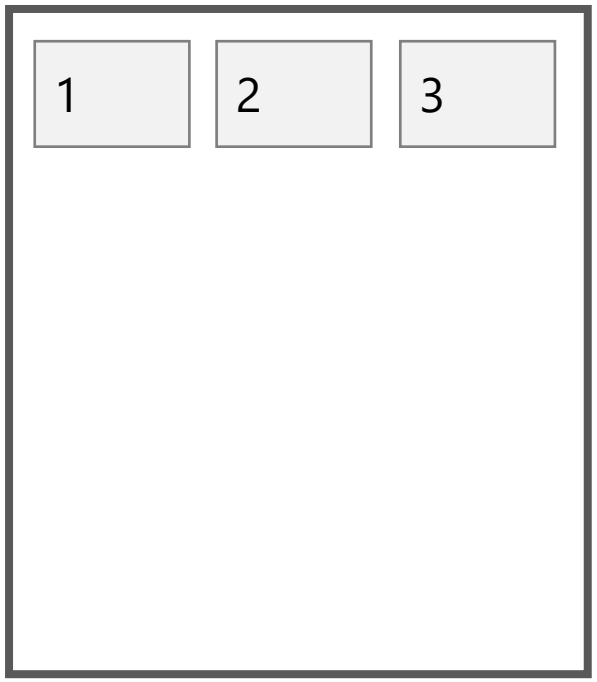


LinearLayout

```
<?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" >
    <TextView ... />
    <TextView ... />
    <TextView ... />
</LinearLayout>
```

Layouts

- What if you want to put three Text Views horizontally with equal width?

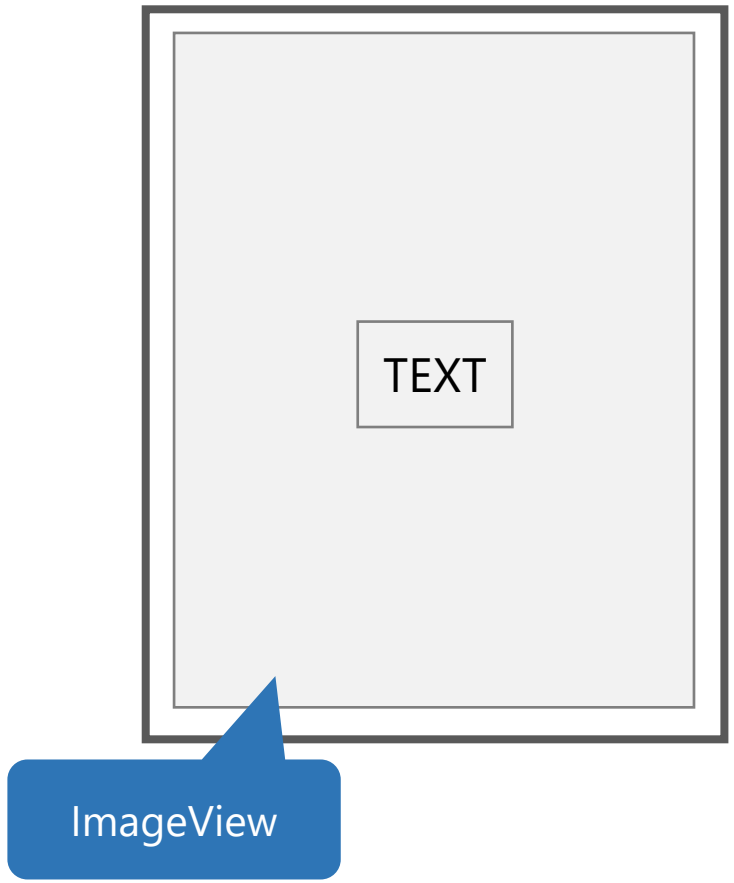


LinearLayout

```
<?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" >
    <TextView
        android:layout_width="0dip"
        android:layout_height="wrap_content"
        android:layout_weight="1"/>
    <TextView ... />
    <TextView ... />
</LinearLayout>
```

Layouts

- What if you want to put a TextView in the centre of the screen on top of a background image held by an ImageView?



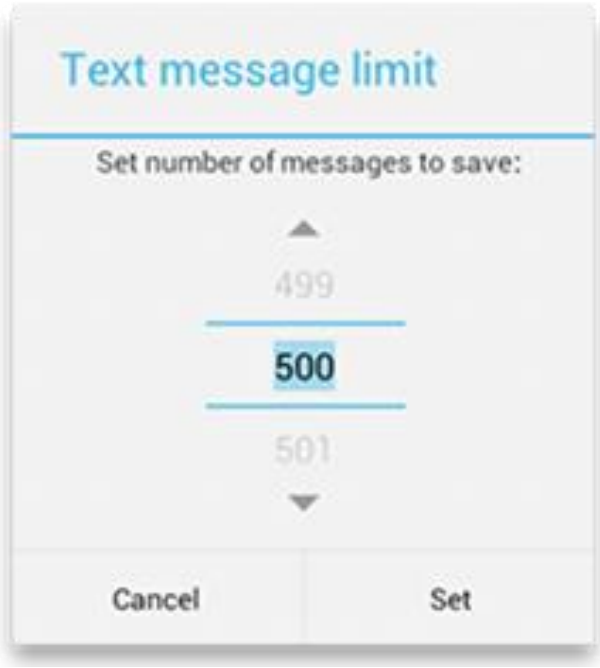

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView .../>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerInParent="true"/>
</RelativeLayout>
```

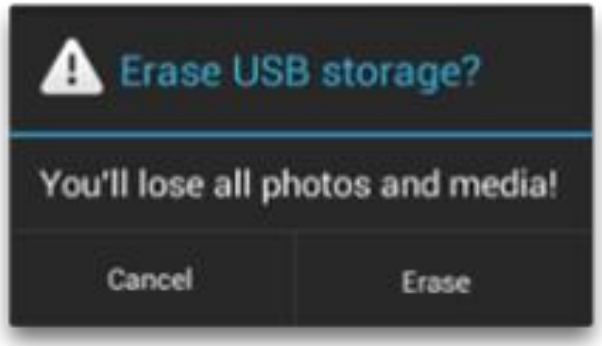
Dialogs

Dialogs are small windows that pop up in an activity to give the user alerts, or prompt the user for input

Types of dialogs:

- AlertDialog
- DatePickerDialog
- TimePickerDialog





http://developer.android.com/guide/topics/ui/dialogs.html
https://www.google.com/design/spec/components/dialogs.html

Dialogs

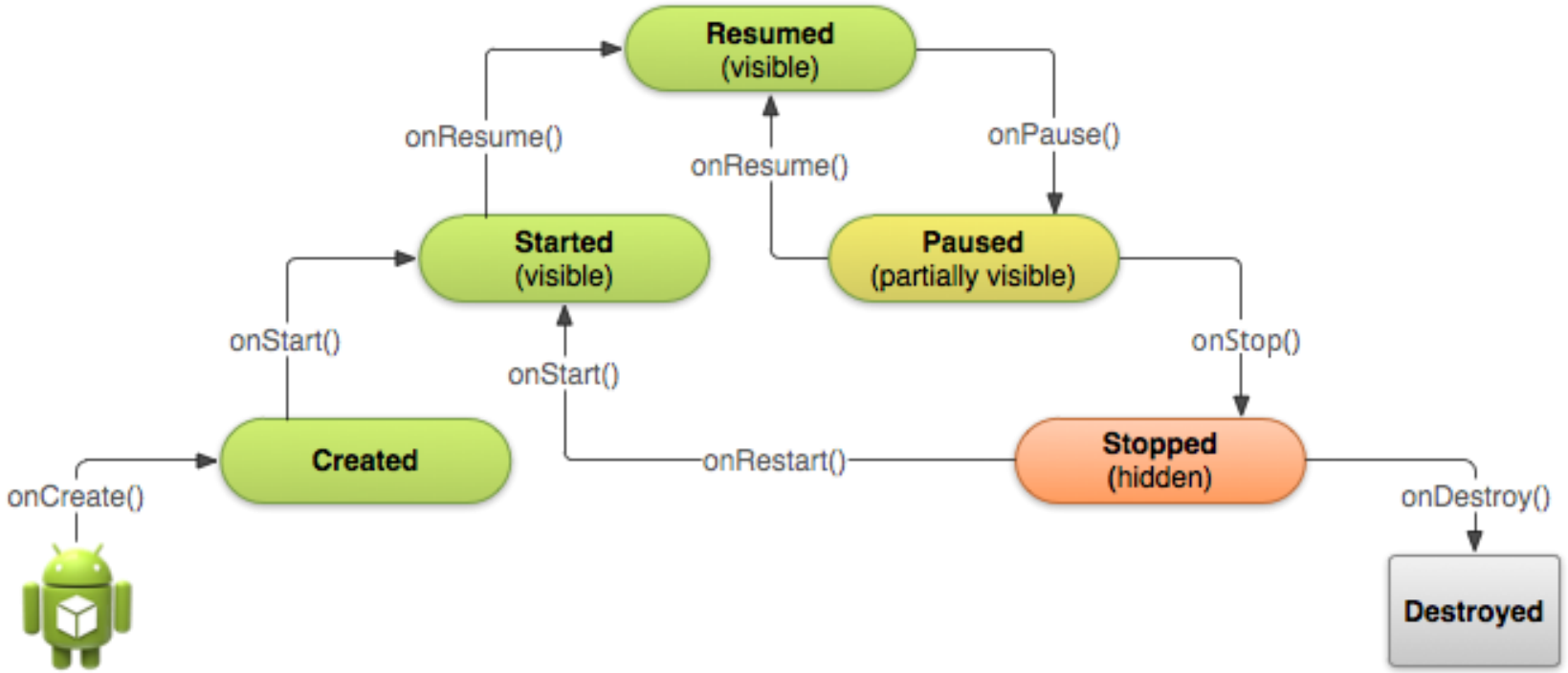
Creating an AlertDialog using the AlertDialog.Builder

```
// Use the Builder class for convenient dialog construction
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
builder.setMessage(R.string.dialog_fire_missiles)
       .setPositiveButton(R.string.fire, new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
               // FIRE ZE MISSILES!
           }
       })
       .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
               // User cancelled the dialog
           }
       });
// Create the AlertDialog object and show the dialog;
AlertDialog dialog = builder.create();
dialog.show();
```

Activity

Activity

- Activity is a fundamental class in Android
- Each page in Android is an Activity
- Each Activity has its own 'life cycle'



Activity

Let's take a look at an Activity class

- You should at least override the onCreate method when you are creating a new Activity
- You should specify the actions and logics to be performed when the activity is created
- Other methods can be overridden if necessary

```
public class MainActivity extends AppCompatActivity {

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

    }

    @Override
    protected void onPause() {
        super.onPause();
    }

    @Override
    protected void onResume() {
        super.onResume();
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
    }
}
```

Activity

In an Activity, how can we refer to the UI components defined in the XML file?

- Let's assume you have a TextView and a Button defined

Inside the XML file

```
<TextView
    android:id="@+id/text_field"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>

<Button
    android:id="@+id/button_1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

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

    text_field = (TextView)findViewById(R.id.text_field);
    button_1 = (Button)findViewById(R.id.button_1);

    ...
}
```

Inside the Java code

Intents and Intent Filters

Android Programming - Intents

Intents

- To request an action to be performed:
  - Start an activity (Either of your app or another app in Android)
  - Start a service (background running process)
  - Deliver a broadcast
- Two types of Intents:
  - **Explicit**: You specify the component to be started
  - **Implicit**: You declare the action to be performed, let Android or the user decide which app or component to invoke

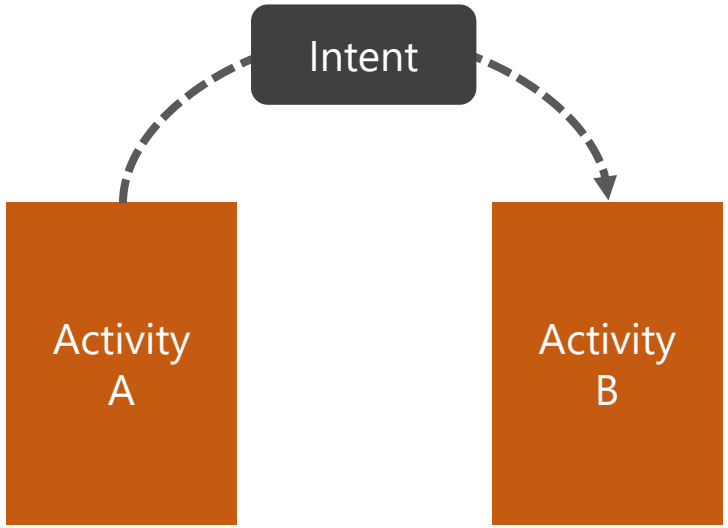
Reference:
https://developer.android.com/guide/components/intents-filters.html

Android Intents

Explicit Intents

Specify the component to be started (e.g. an activity)

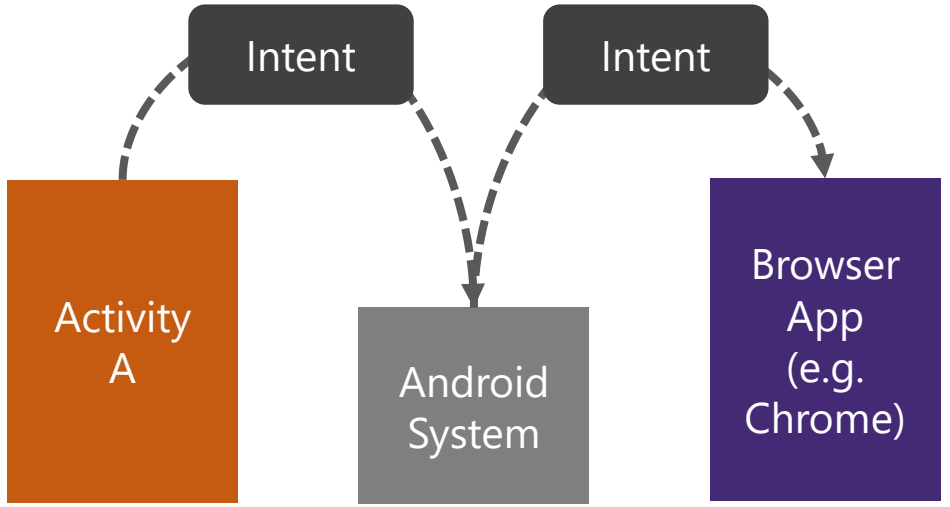
Example:
In Activity A, when user clicks the button, start Activity B



Implicit Intents

Specify the action to take, let Android or the user to decide what to invoke

Example:
In Activity A, when the user clicks the button, open an URL (e.g. http://www.cuhk.edu.hk)



Reference: https://developer.android.com/guide/components/intents-filters.html

Android Intents

Intents are messaging objects for requesting an action. Three fundamental use-cases:

1. To start an **activity**
    - An activity is a screen in an app, can be invoked using **startActivity()** or **startActivityForResult()**
2. To start a **service**
    - A service is a component for performing background operations without a user interface, can be invoked using **startService()**
3. To deliver a **broadcast**
    - A broadcast is a message that any app in the system can receive, can be invoked by **sendBroadcast()**

Components of Intents

An intent object contains information that the Android system uses to determine which app and component it should start

In general, an intent would contain 4 major pieces of information

1. **Component name**
    - Refers to the component to be started, e.g. **com.iems5722.asgn3.MainActivity**
    - Optional, but required if you want to create an explicit intent
    - When this is omitted, the Android system will determine which app and component it should invoke, based on the action parameter you provided

Components of Intents

2. Action

- A **string** that specific the action to perform
- You can specify your own actions to be used in intents
- The Intent class provides a set of standard actions, for example:
  - **ACTION_VIEW**
    For displaying some information to the user
  - **ACTION_SEND**
    For sending or sharing the data or information through another app
- If it is an explicit intent, the action is optional

Components of Intents

3. Data

- The data component contains the **URI** referring to the data and/or the **MIME type** of that data
- The content is usually dependent on the action of the intent
  (e.g. ACTION_EDIT should be accompanied by a URI to the file to be edited)
- To set the URI, call **Intent.setData()**
- To set the MIME type, call **Intent.setType()**
- To set BOTH the URI and the MIME type, call **Intent.setDataAndType()**

Components of Intents

4. Extras

- Extras are key-value pairs that can be used to pass parameters to the activity or service to be started
- Use **Intent.putExtra(key, value)** to set the parameters to be passed
- The Intent class has defined some standard keys for passing parameters, e.g.:
  - EXTRA_SUBJECT
  - EXTRA_EMAIL
  - EXTRA_TITLE

Constructing an Explicit Intent

An explicit intent is used to launch a specific app component

For example, we construct an intent like this:

```
...
Intent intent = new Intent(MyActivity.this, NextActivity.class);
intent.putExtras(“PARAM_1”, “value_1”);
intent.putExtras(“PARAM_2”, “value_2”);
startActivity(intent);
```

And in the NextActivity’s onCreate() method, we extract the extras like this:

```
...
Bundle extras = getIntent().getExtras();
String p1 = extras.getString(“PARAM_1”);
String p2 = extras.getString(“PARAM_2”);
```

Constructing an Implicit Intent

If you want to perform an action, but let the Android system to decide the most appropriate app to perform the action, you can use an implicit intent

For example, if you would like to let the user share some text:

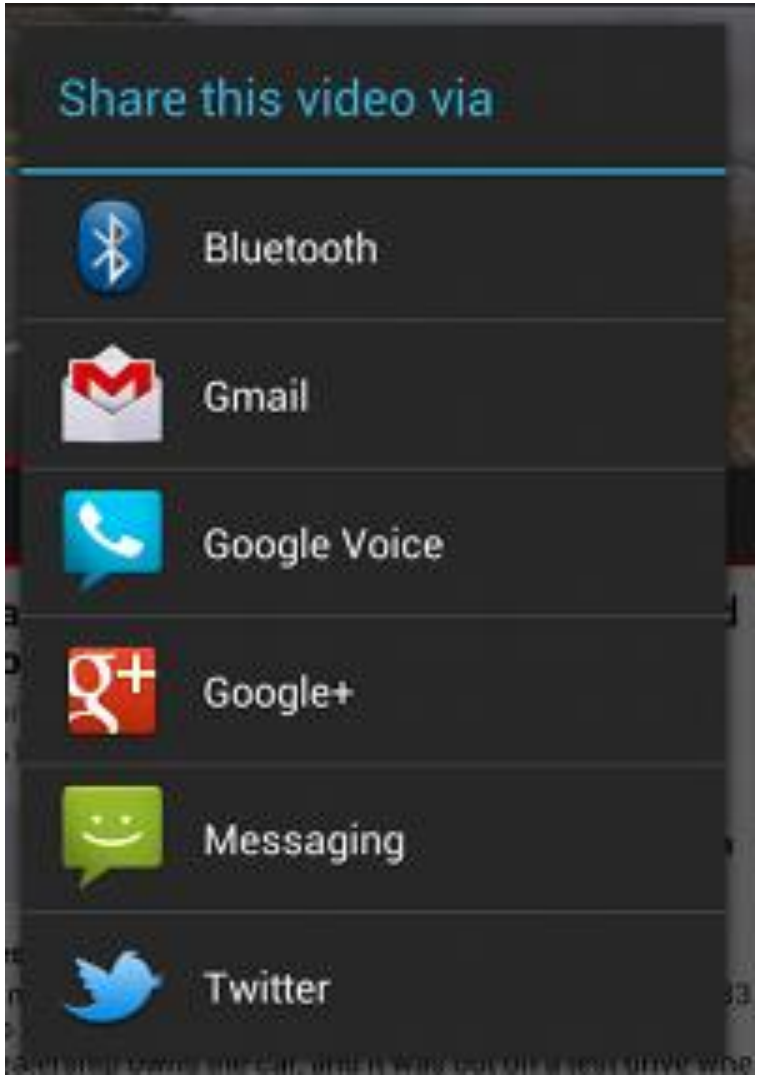
```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType(HTTP.PLAIN_TEXT_TYPE); // "text/plain" MIME type

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

Check whether there is an app that can handle this intent

The App Chooser

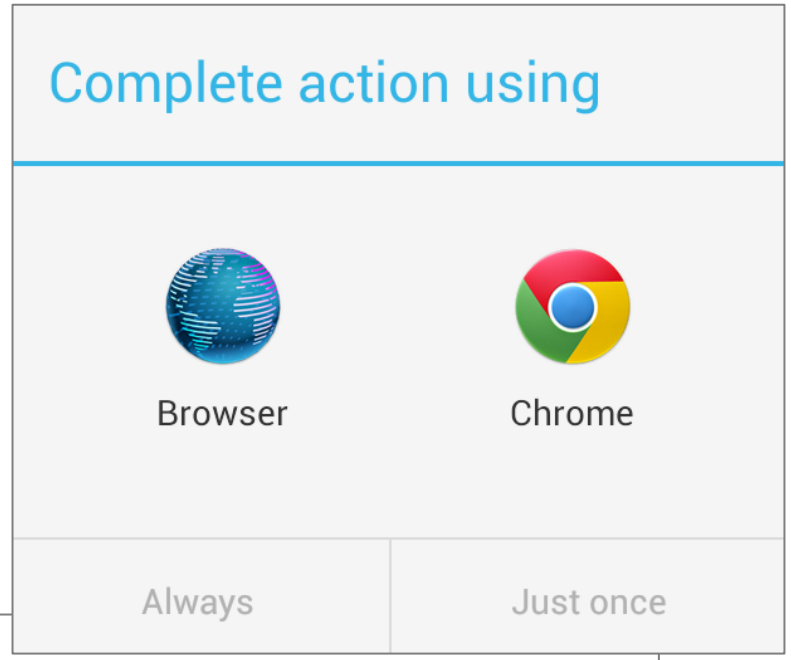
When you use an implicit intent:

- If there is only one app that can handle the intent, that app will be launched immediately
- If there are more than one app that can handle the intent, Android will present the user an "App Chooser"



Forcing the App Chooser

- Note that the user can choose one app to be the default app to handle a specific intent
- If you want to force the app chooser to appear every time when you use such intent, you should do something like this:




```
Intent sendIntent = new Intent(Intent.ACTION_SEND);
...
String title = getResources().getString(R.string.chooser_title);
Intent chooser = Intent.createChooser(sendIntent, title);

// Verify the original intent will resolve to at least one activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(chooser);
}
```

Intent Filters

You can define which implicit intents your app can receive by declaring intent filters in the AndroidManifest.xml file

Each **`<intent-filter>`** element should contain one or more of these three sub-elements:

- **`<action>`**
  - The action of the intent to be filtered
- **`<data>`**
  - The URI scheme or the MIME type
- **`<category>`**
  - You must at least include the CATEGORY_DEFAULT category in the intent filter

Intent Filters

Examples:

```
<intent-filter>
    <action android:name="android.intent.action.SEND"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <data android:mimeType="text/plain"/>
</intent-filter>
```

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <data android:scheme="http" android:host="www.example.com" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.BROWSABLE" />
</intent-filter>
```

References: https://developer.android.com/guide/components/intents-common.html

Broadcast Receivers

Broadcast Receivers

Broadcast receivers can be used to receive intents that are broadcasted to the whole Android system

Main usage scenarios

1. Implement a broadcast receiver to receive broadcast messages from Android APIs (e.g. Bluetooth connection changes, Google cloud messaging)

2. Handle broadcast messages within your app

Local Broadcast Receivers

Generate and send a broadcast message

```
Intent intent = new Intent(“com.iems5722.example.action001”);
intent.putExtra(“result”, “testing”);
LocalBroadcastManager.getInstance(MyActivity.this).sendBroadcast(intent);
```

Note

- When you only want to broadcast a message within your app, use the LocalBroadcastManager whenever possible
- This will be more efficient and secure (other apps will not be able to intercept and receive this intent)

Local Broadcast Receivers

The corresponding broadcast receiver should be defined as something like this:

```
BroadcastReceiver receiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        Bundle extras = intent.getExtras();
        String result = extras.get(“result”);
        ...
    }
};
```

Local Broadcast Receivers

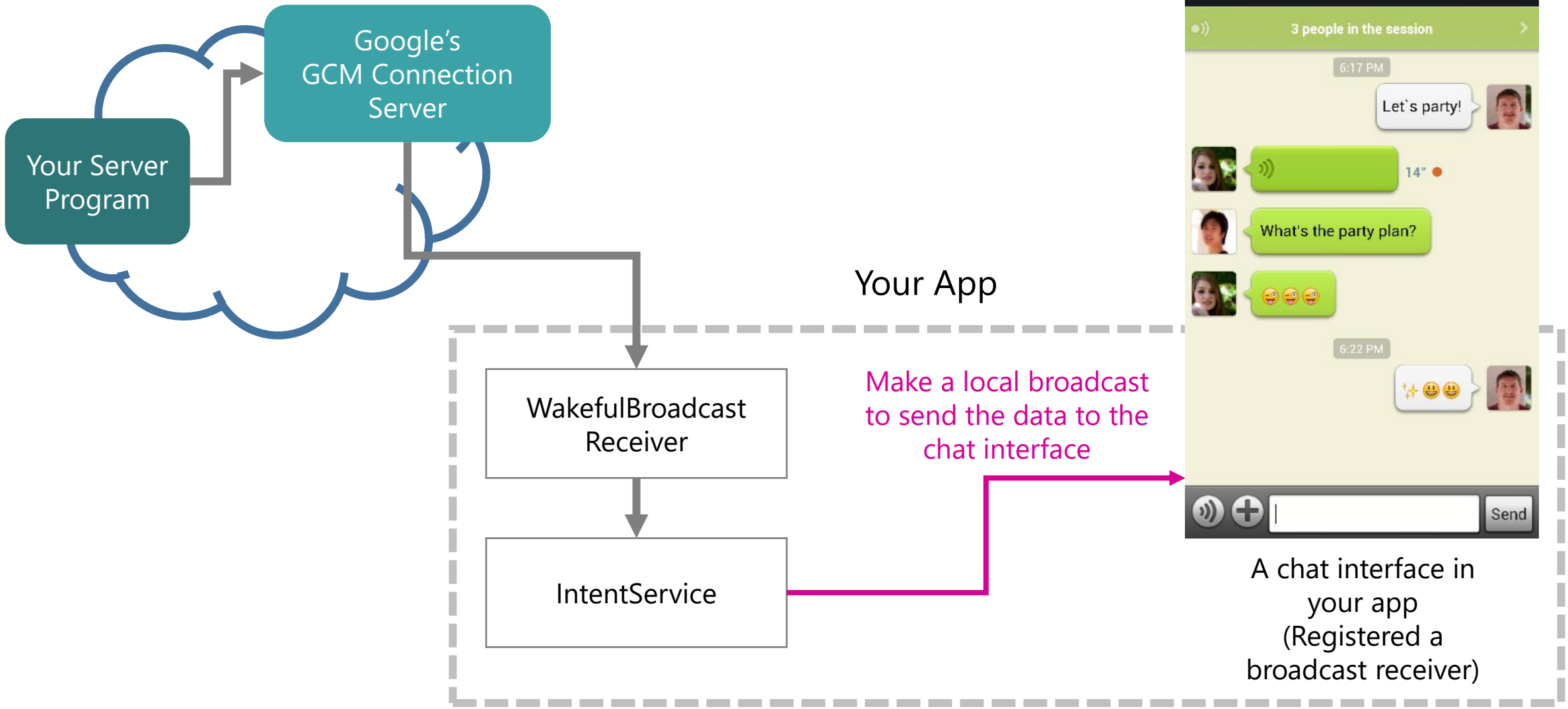
Registering and unregistering the broadcast receiver in an activity

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    IntentFilter filter = new IntentFilter("com.iems5722.example.action001");
    LocalBroadcastManager.getInstance(this).registerReceiver(receiver, filter);
    ...
}
```

```
@Override
protected void onDestroy() {
    if (receiver != null) {
        LocalBroadcastManager.getInstance(this).unregisterReceiver(receiver);
    }
    super.onDestroy();
}
```

Local Broadcast Receivers

Example:



Local Broadcast Receivers

Local broadcast receivers can be used for **cross-thread communication**

- One thread broadcasts a message with a specific intent
- Another thread (e.g. the UI thread) registers a broadcast receiver to act on that message

PROS

- The two threads do not need to share any components or variables

CONS

- No message event queue is available
- You cannot schedule an action to be performed at a specific time

Local Storage

Local Storage

Even if you app is supported by the Internet and a server, you may need to store data inside the device, such as

- User Preferences
- Cache of data from the Internet

Android has several built-in mechanism for you to store data in the device

- Shared Preferences (Key / Value pairs)
- Internal Storage / External Storage (Files)
- SQLite Databases (Structured data)

Local Storage – Shared Preferences

The simplest way of storing data in an Android device is to use Shared Preferences

- For saving and retrieving persistent key-value pairs
- Support primitive data types such as integer, double, string or boolean

```
SharedPreferences settings =
    getSharedPreferences("PREF_NAME", 0);

SharedPreferences.Editor editor = settings.edit();
editor.putInt("USER_ID", 54321);

// Remember to commit
editor.commit();
```

Writing or updating the value of a key

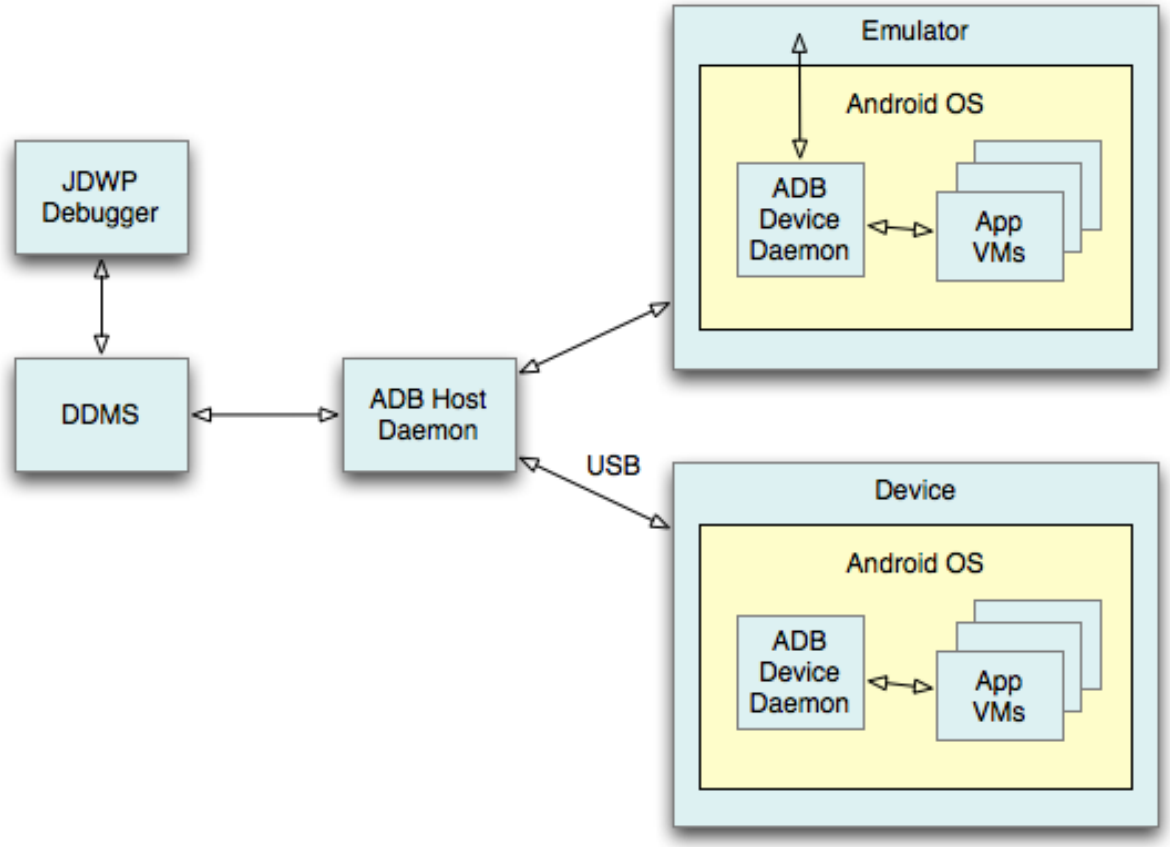
```
SharedPreferences settings =
    getSharedPreferences("PREF_NAME", 0);

// Default value is 0 if the key does not exist
settings.getInt("USER_ID", 0);
```

Reading the value of a key

Debugging

Debugging Android Apps

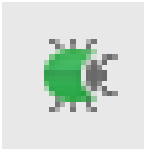
- In developing Android Apps, it is common that you encounter problems when the app is executed (e.g. freeze, crash, unexpected behaviour)
- Debugging tools allow you to identify and trace the problem, and help you debug your app
- The major tool you will use is the **DDMS (Dalvik Debug Monitor Server)**

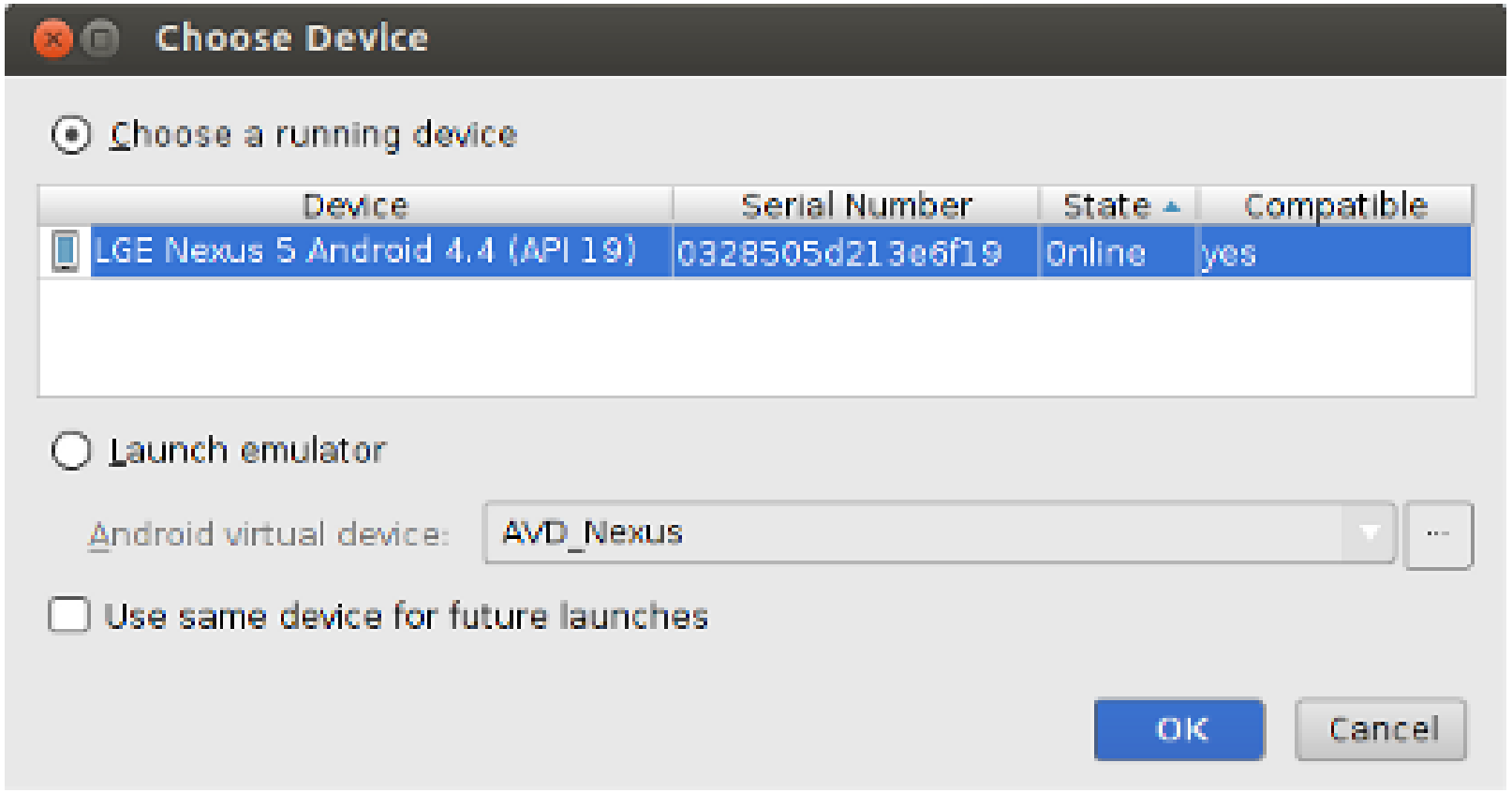


http://developer.android.com/tools/debugging/index.html

Debugging Android Apps

Run the app in DEBUG mode



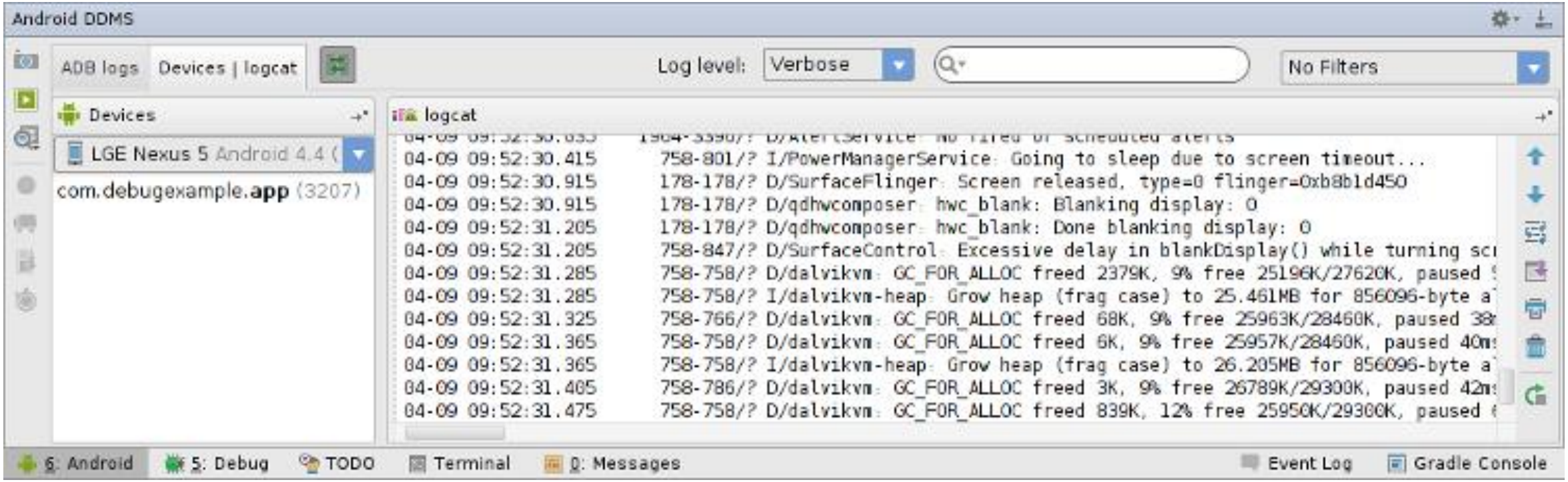


Reference: http://developer.android.com/tools/debugging/debugging-log.html

Debugging Android Apps

Common Methods to Debug Your App (1)

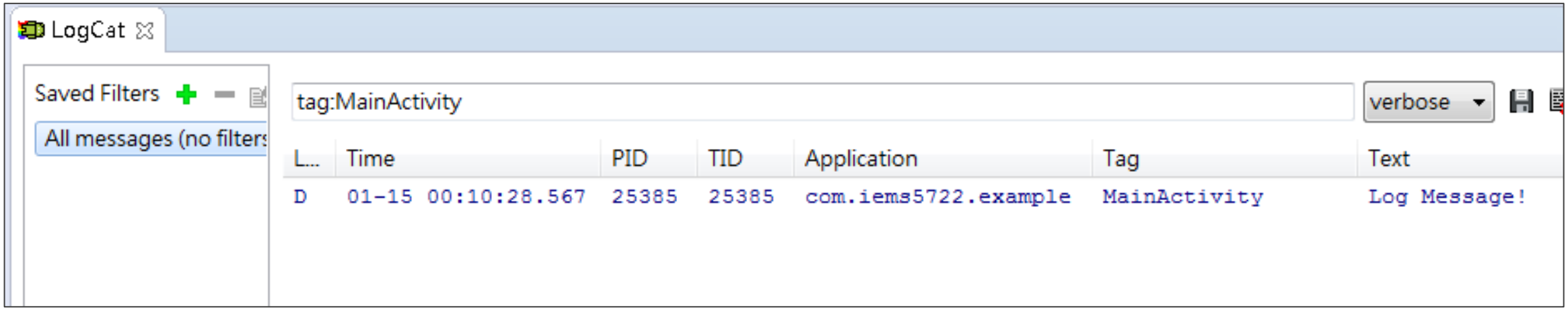
- Using the **Log** class to write logs
- View the logs in the "**logcat**" tool



Reference: http://developer.android.com/tools/debugging/debugging-log.html

Debugging Android Apps

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

    Log.d("MainActivity", "Log Message!");

}
```

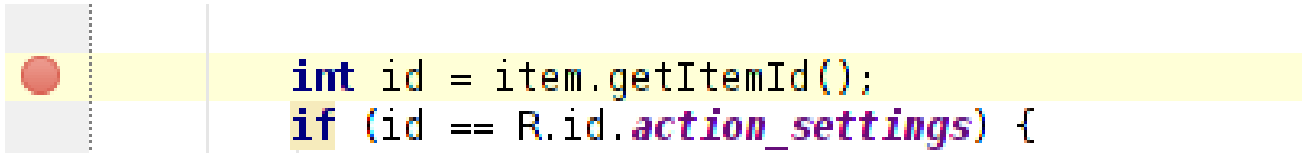


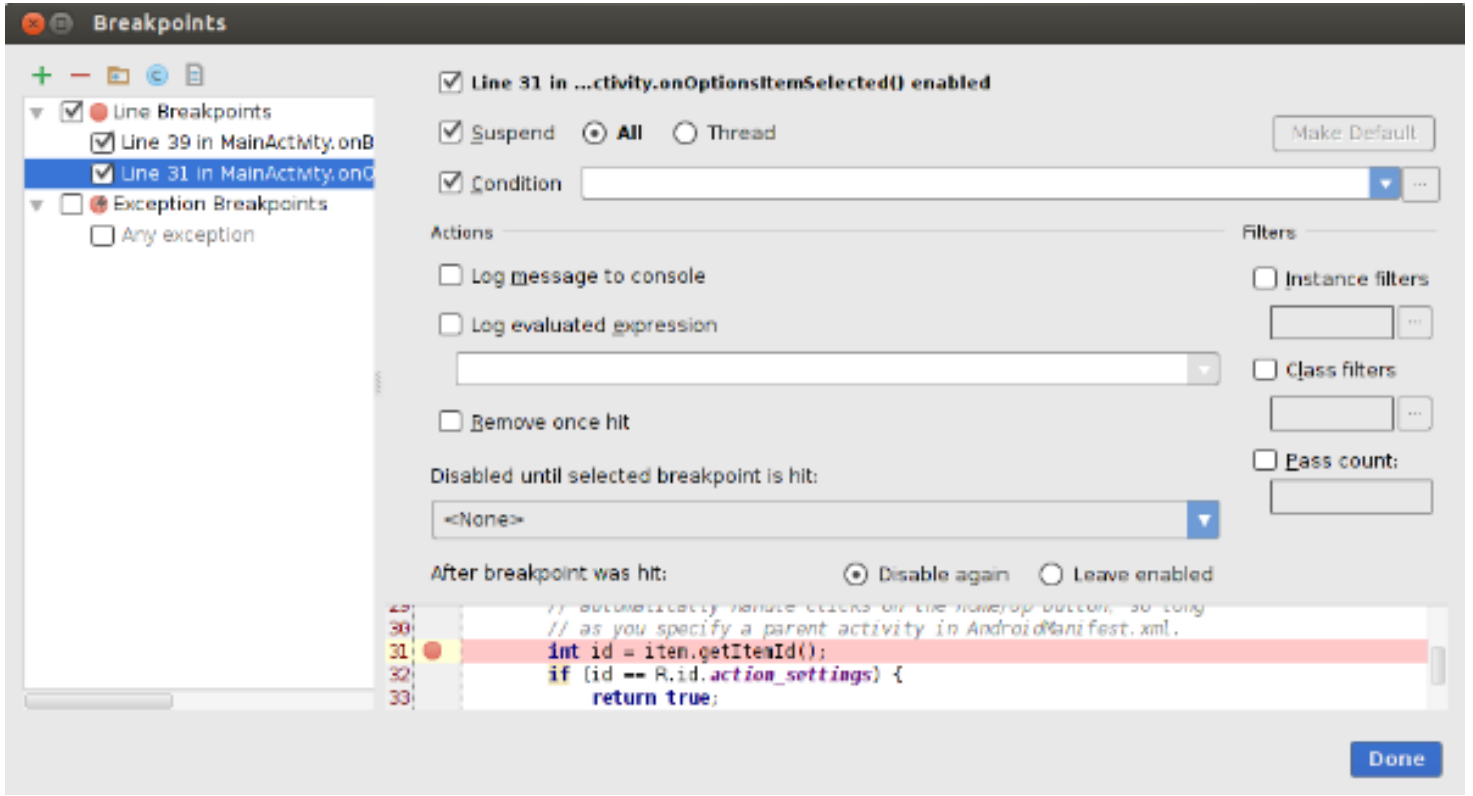
http://developer.android.com/tools/debugging/debugging-log.html

Debugging Android Apps

Common Methods to Debug Your App (2)

- Set **break points** in your code
- User the debugger and walk through your code line-by-line
- Watch the values of the variables in the program





Reference: http://developer.android.com/tools/debugging/debugging-projects.html

Learning Resources

Java Programming

- The Java Tutorial: http://docs.oracle.com/javase/tutorial/
- Java HashMap Tutorial: http://www.tutorialspoint.com/java/java_hashmap_class.htm

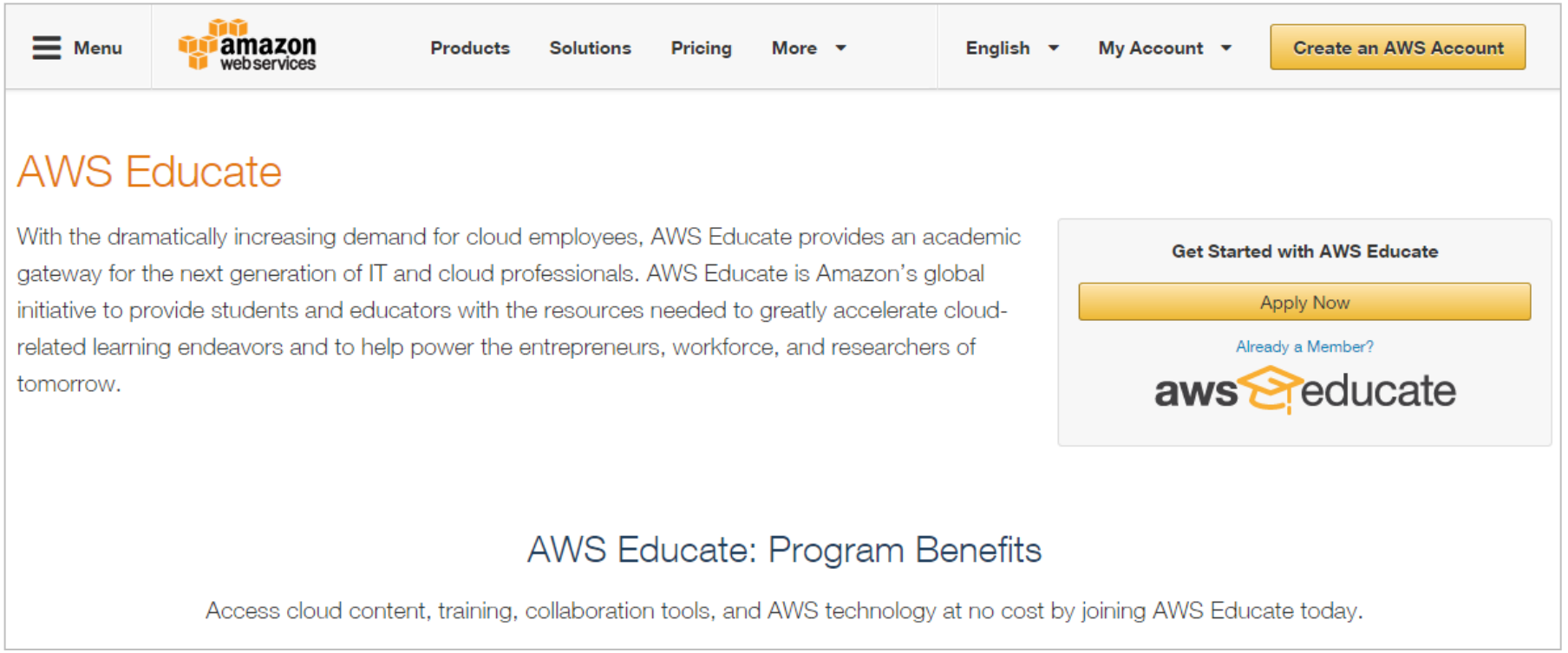
Android Programming

- Layouts: http://developer.android.com/guide/topics/ui/declaring-layout.html
- Input Controls: http://developer.android.com/guide/topics/ui/controls.html
- Input Events: http://developer.android.com/guide/topics/ui/ui-events.html
- Toast Messages: http://developer.android.com/guide/topics/ui/notifiers/toasts.html
- Dialog Messages: http://developer.android.com/guide/topics/ui/dialogs.html
- Intent and Sharing: http://developer.android.com/training/sharing/send.html

Amazon AWS

Amazon AWS

- For your assignments and project, you will need to build a server. You can use Amazon AWS's free tier service.
- Create an account at AWS Educate to get USD 35 credits
- https://www.awseducate.com/



Next Lecture:
Data Communications and Client-Server Architecture

End of Lecture 2