



MONASH University

Information Technology

FIT5183: Mobile and Distributed Computing Systems (MDCS)

Lecture 7 - Android Application Development

Outline

- ☐ Overview of Android OS
- ☐ Review of Android Components and File Structure
- ☐ Activities, Services, Broadcast Receivers and Content Providers (SQLite)
- ☐ Intents and Intent Filters
- ☐ Bundles. Serializable and Parcelable Data
- ☐ Fragments and SharedFragments
- ☐ Threads and AsyncTask
- ☐ HttpURLConnection

Source material for these slides was derived from relevant APIs and official documentation at <https://developer.android.com>

Some additional Slides used in this lecture were also sourced from Android Development eBook available at http://www.tutorialspoint.com/android/android_tutorial.pdf

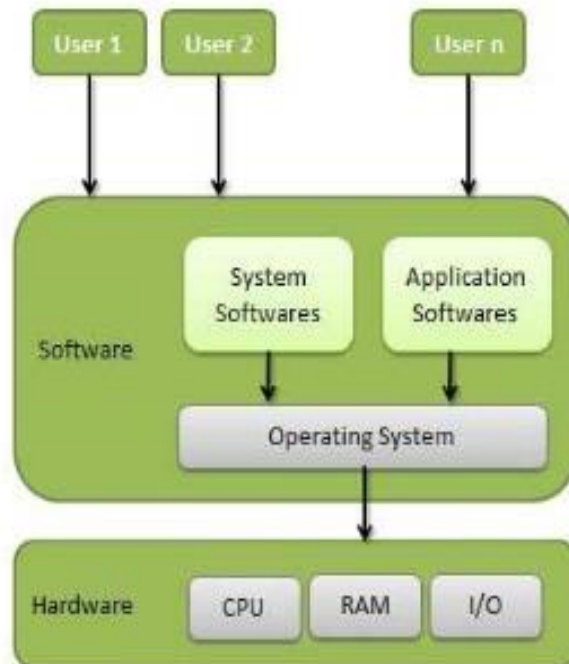


Overview of Android Operating System

Operating Systems – Back to Basics!

Definition

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



Following are some of important functions of an operating System.

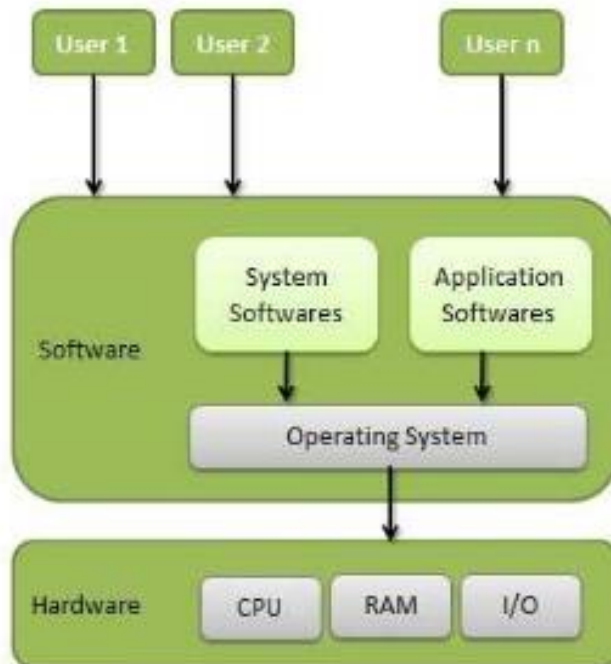
- ▣ Memory Management
- ▣ Processor Management
- ▣ Device Management
- ▣ File Management
- ▣ Security
- ▣ Control over system performance
- ▣ Job accounting
- ▣ Error detecting aids
- ▣ Coordination between other software and users

http://www.tutorialspoint.com/android/android_tutorial.pdf

Operating Systems – Back to Basics!

Definition

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

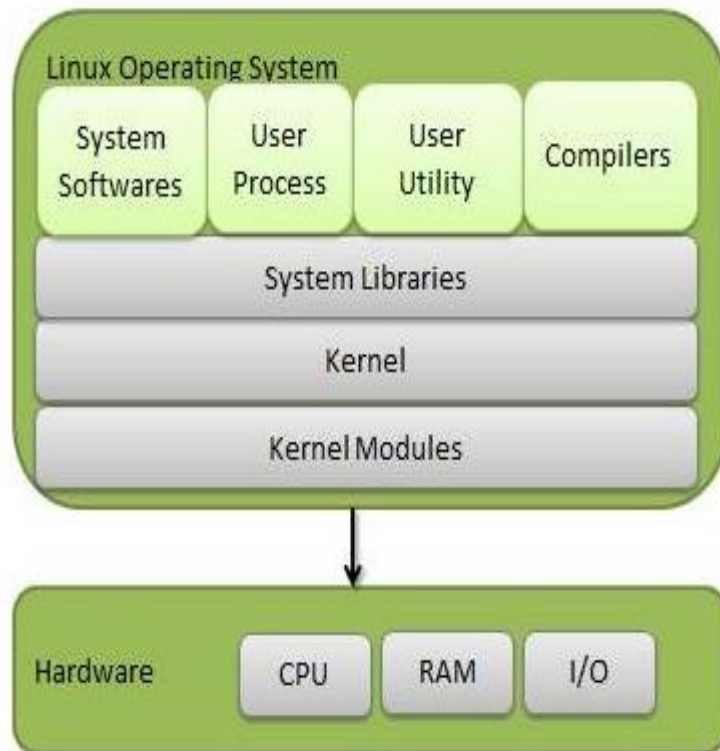


Following are some of important functions of an operating System.

- ▣ Memory Management
- ▣ Processor Management
- ▣ Device Management
- ▣ File Management
- ▣ Security
- ▣ Control over system performance
- ▣ Job accounting
- ▣ Error detecting aids
- ▣ Coordination between other software and users

http://www.tutorialspoint.com/android/android_tutorial.pdf

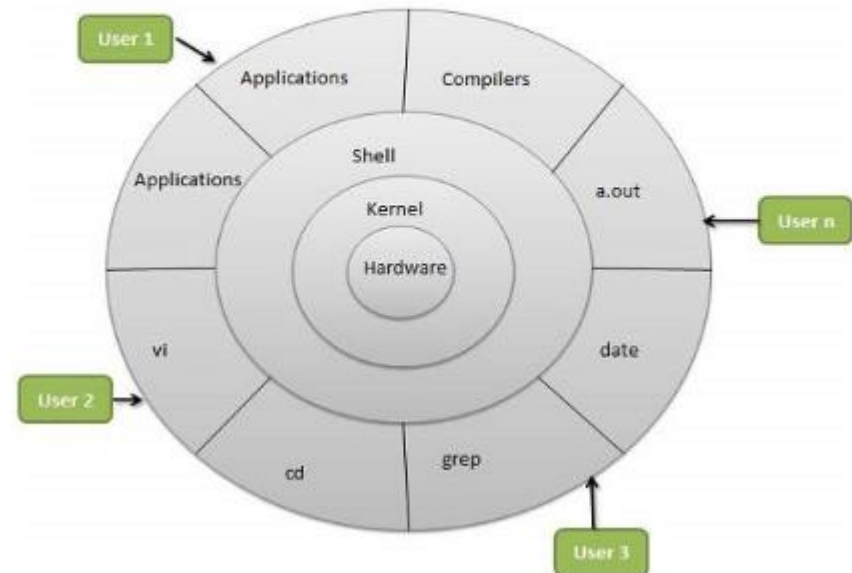
Linux OS



http://www.tutorialspoint.com/android/android_tutorial.pdf

Architecture

The following illustration shows the architecture of a Linux system –

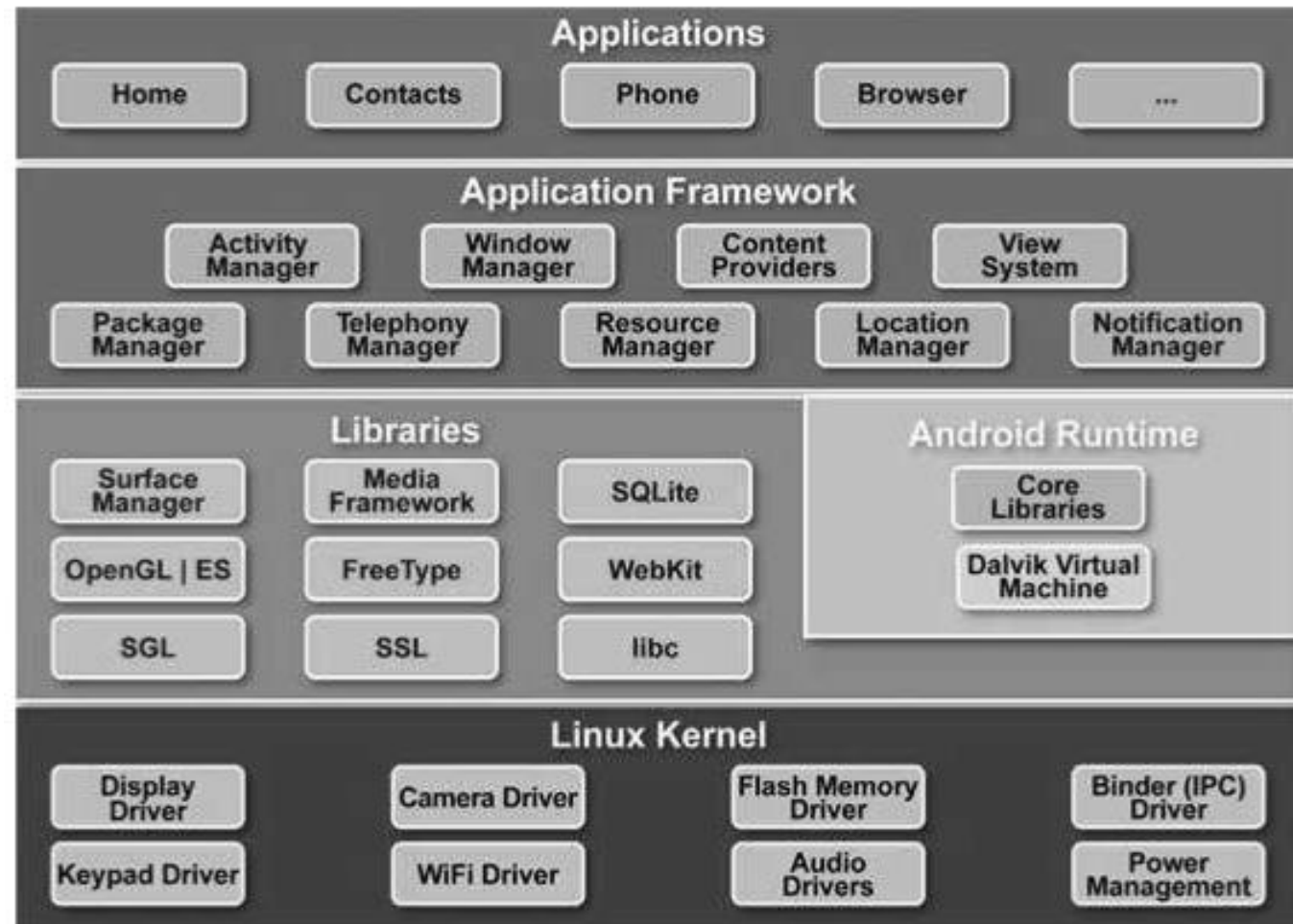


The architecture of a Linux System consists of the following layers –

- **Hardware layer** – Hardware consists of all peripheral devices (RAM/ HDD/ CPU etc).
- **Kernel** – It is the core component of Operating System, interacts directly with hardware, provides low level services to upper layer components.
- **Shell** – An interface to kernel, hiding complexity of kernel's functions from users. The shell takes commands from the user and executes kernel's functions.
- **Utilities** – Utility programs that provide the user most of the functionalities of an operating systems.

Android OS Stack

- ❑ The Android Operating system is a stack of software components which is roughly divided into 5 sections and four main layers.



http://www.tutorialspoint.com/android/android_tutorial.pdf

Main Android Software Components

Linux kernel At the bottom of the layers is Linux 2.6 with approximately 115 patches. This provides basic system functionality like process management, memory management, device management like camera, keypad, display etc.

Libraries On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

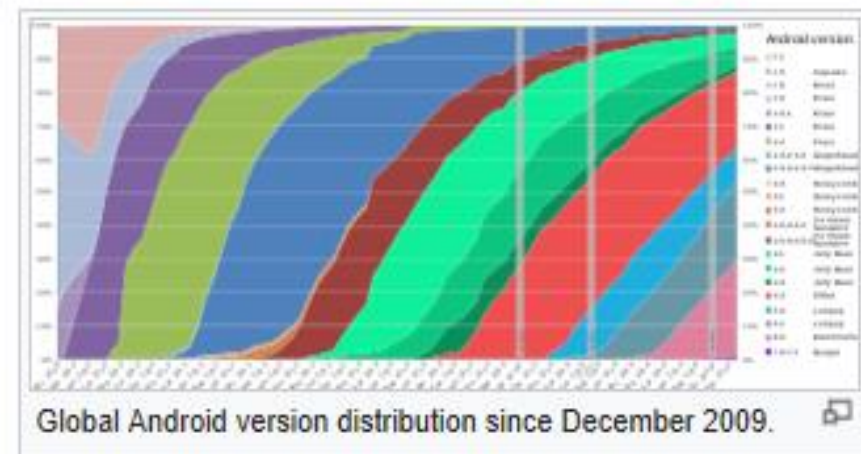
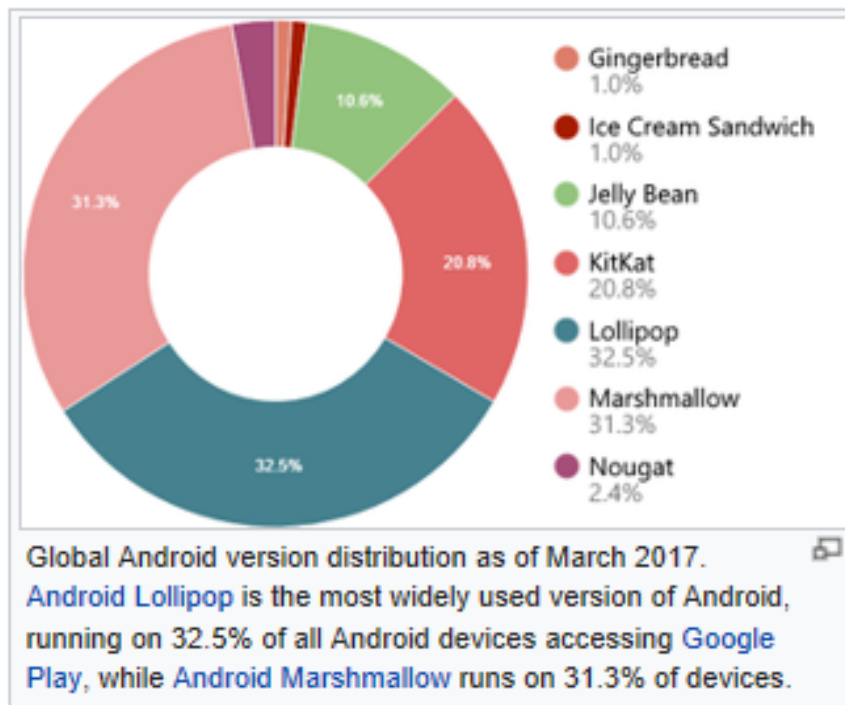
Android Runtime This is the third section of the architecture available on the second layer from the bottom. This section provides a key component called *Dalvik Virtual Machine* which is a kind of Java Virtual Machine specially designed and optimized for Android. The Dalvik VM makes use of Linux core features like memory management and multi-threading which are intrinsic in Java.

Application Framework The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.

Applications You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games, etc.

Android Evolution (repeat)

- ❑ At latest count Android 5 (Lollipop) and 6 (Marshmallow) accounted for almost 64% of all Android devices (measured as access to Google Play).
- ❑ The distribution of Android 7 (Nougat) was measured as 2.4% and is expected to grow.



Source: wikipedia



Review of Main Android Application Components and File Structure

Android Application Components

- ❑ Application components are the essential building blocks of an Android application. These components are *loosely coupled* by the application manifest file **AndroidManifest.xml** that describes each component of the application and how they interact.
- ❑ There are following four main components that can be used within an Android application:
 - **Activities** They dictate the UI and handle the user interaction to the smartphone screen
 - **Services** They handle background processing associated with an application.
 - **Broadcast Receivers** They handle communication between Android OS and applications.
 - **Content Providers** They handle data and database management issues.

Android Application Components (cont'd)

- ❑ Additional components are used in the construction of above mentioned entities, their logic, and wiring between them:
- **Fragments** Represent a behaviour or a portion of user interface in an Activity.
- **Views** UI elements that are drawn onscreen including buttons, lists forms etc.
- **Layouts** View hierarchies that control screen format and appearance of the views.
- **Intents** Messages wiring components together.
- **Resources** External elements, such as strings, constants and drawable pictures.
- **Manifest** Configuration file for the application.

Anatomy of a Basic Android Application

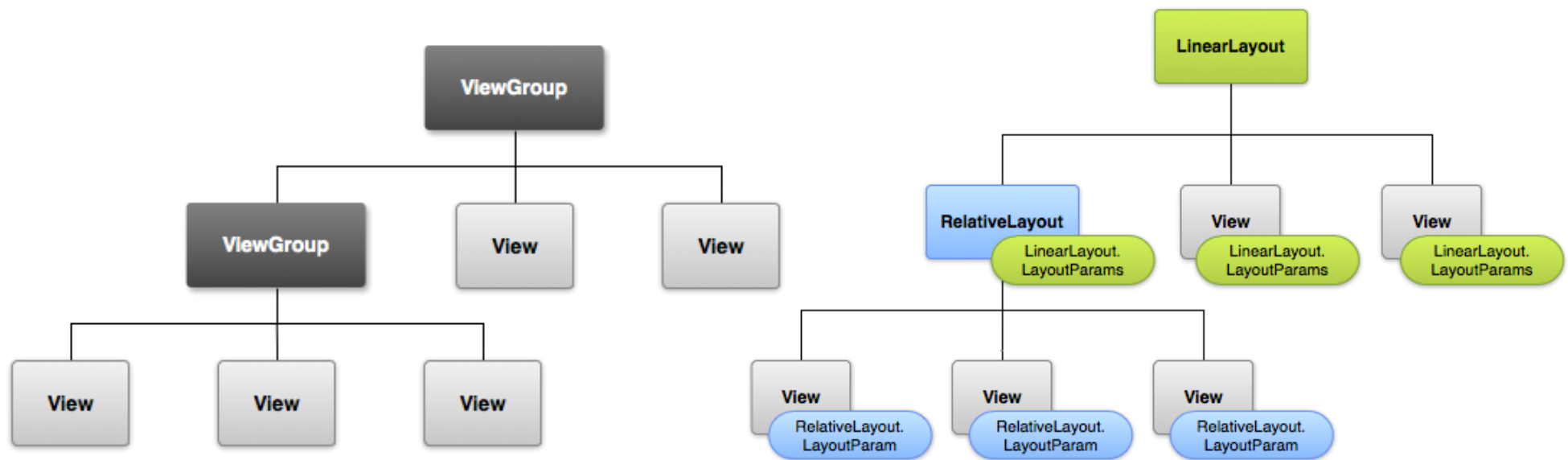
- ❑ **src** This contains the .java source files for your project. By default, it includes a MainActivity.java source file having an activity class that runs when your app is launched using the app icon.
- ❑ **gen** This contains the .R file, a **compiler-generated** file that references all the **resources** found in your project.
- ❑ **bin** This folder contains the **Android package files .apk** built by the Android Studio during the build process and everything else needed to run an Android application.
- ❑ **res/drawable-*** This is a directory for drawable objects that are designed for screens of different pixel densities (PD).
- ❑ **res/layout** This is a directory for files that define your app's user interface.
- ❑ **res/values** This is a directory for other various XML files that contain a collection of resources, such as strings and colors definitions.
- ❑ **AndroidManifest.xml** This is the manifest file which describes the fundamental characteristics of the app and defines each of its components.

Summary of Android Project Files:

- ❑ **The Main Activity File** The main activity code is a Java file `MainActivity.java`. This is the actual application file which ultimately gets converted to a Dalvik executable and runs your application
- ❑ **The Manifest File** Whatever component you develop as a part of your application, you must declare all its components in a manifest file called `AndroidManifest.xml` which resides at the root of the application project directory. This file works as an interface between Android OS and your application
- ❑ **The Strings File** The `strings.xml` file is located in the `res/values` folder and it contains all the text that your application uses. For example, the names of buttons, labels, default text, and similar types of strings go into this file. This file is responsible for their textual content.
- ❑ **The Layout File** The `activity_main.xml` is a layout file available in `res/layout` directory that is referenced by your application when building its interface. You will modify this file very frequently to change the layout of your application.

Views

- ❑ A View occupies a rectangular area on the screen and is responsible for drawing and event handling
- ❑ View is the base class for *widgets*, which are used to create interactive UI components (buttons, text fields, etc.)



Views

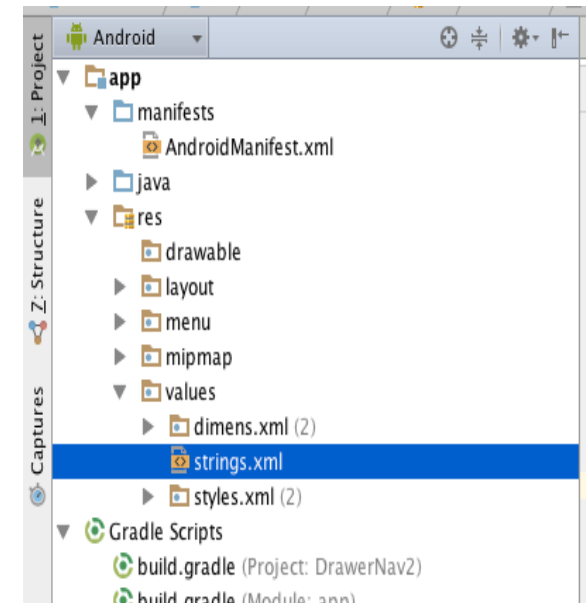
- ❑ Views may have an integer id associated with them to uniquely identify the View
- ❑ When the application is compiled, this ID is referenced as an integer
 - `android:id="@+id/my_button"`
 - The at-symbol (@) for XML parsing purposes
 - The plus-symbol (+) means that this is a new resource name that must be created and added to our resources (in the R.java file)

Views

- ❑ After creating views, you can reference them from the application:

- ❑ In xml file:

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



- ❑ **wrap_content**: it sizes the view to the dimensions required by its content
- ❑ **match_parent**: it sizes the view as big as its parent view group will allow
- ❑ In Activity class (usually in the onCreate()method) :

```
Button exitButton = (Button) findViewById(R.id.exitButton);
```

- ❑ **List Adapter** (Extended Adapter class). The List Adapter is the bridge between a ListView and the data associated with the list. The ListView can display any data provided that is wrapped in a ListAdapter.

Two Ways to Handle Events

1. Applying an OnClickListener to the button in your activity

```
exitButton.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        // Perform action on click  
    }  
});
```


2. Or assign a method to your button in the XML layout, using the android:onClick attribute

```
<Button  
    android:id="@+id/exitButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="@string/exitButton_text"/>  
    android:onClick="doSomething" />
```

```
public void doSomething(View view)  
{  
    // perform some action  
}
```

<http://developer.android.com/reference/android/widget/Button.htm>

!



Activities, Services, Receivers and Providers

Activities

- ❑ Main component of applications that displays a screen which users' interaction takes place.
 - We can view this component as a “**Controller**” in the MVC Architecture.
 - Activities facilitate communication between Models and Views.
 - Will be the main component we work with when developing Android applications.
- ❑ Every application comprises **of one main activity** which is opened when the application is launched.
- ❑ When an activity is started, the previous activity is stopped.
 - Android keeps a “**back stack**” of activities which we can use for temporal navigation.

Activity Lifecycle

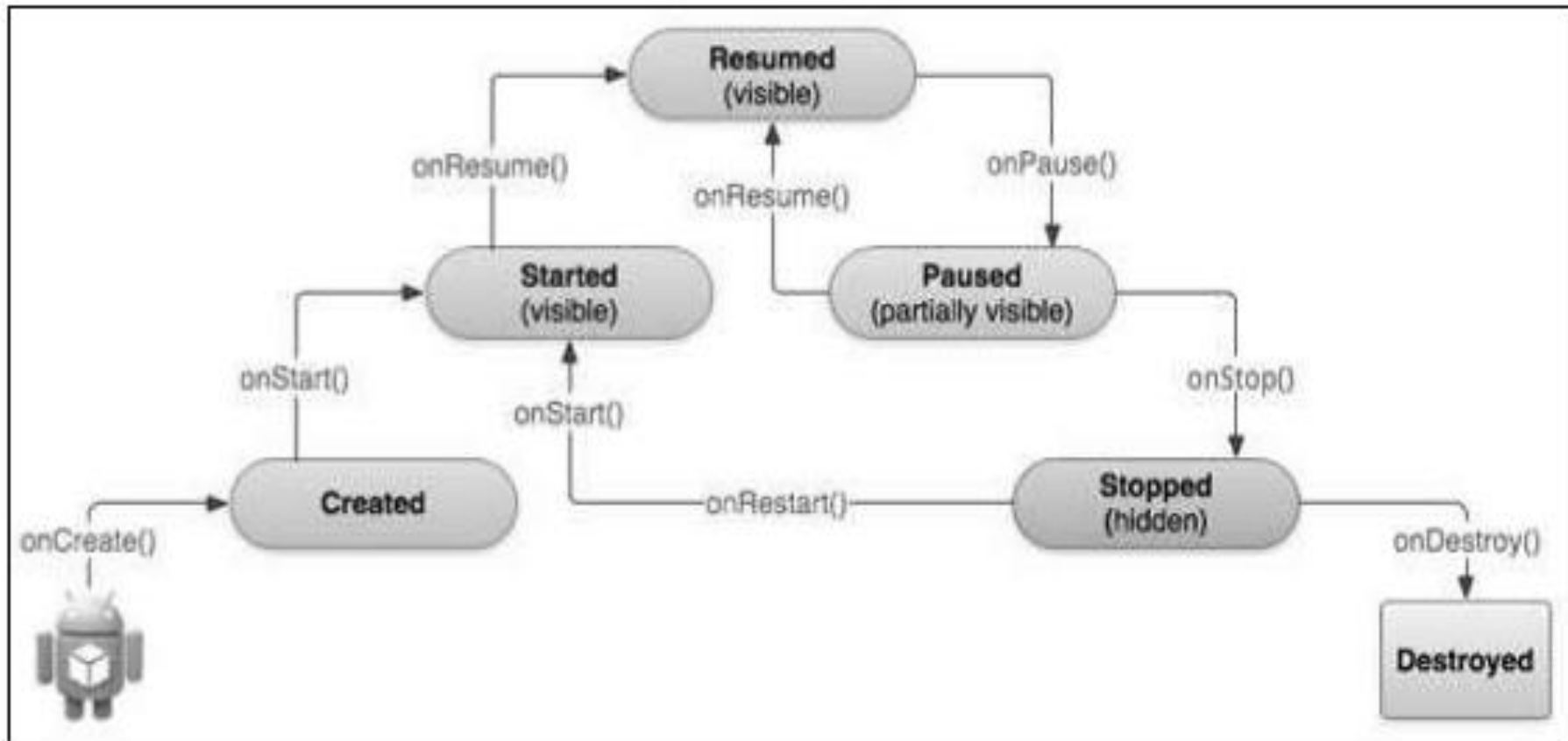
- ❑ Many applications will have several embedded activities that provide functionality to the user. However, we cannot have all activities active at the same time!
- ❑ An activity can be assigned the following **three states**:
 - Paused
 - Resumed
 - Stopped
- ❑ When an activity has been paused or stopped, the operating system **may remove it from memory**.
- ❑ The **finish()** method declared in the activity is called when this happens.
- ❑ The life of an Activity is managed by implementing **callback** methods.

Activity Lifecycle: Callback Methods

□ Activity Method Description:

- **onCreate()** Called when activity is launched for first time.
- **onRestart()** Called after activity has been stopped.
- **onStart()** Called before activity is displayed to user.
- **onResume()** Called before user interaction takes place.
- **onPause()** Called before Android starts another activity.
- **onStop()** Called when activity is no longer shown on screen.
- **onDestroy()** Final call before activity is removed.

Activity Lifecycle: Callback Methods



Original source <http://www.android.com>

Reprinted in http://www.tutorialspoint.com/android/android_tutorial.pdf

A simple program to log callback events

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.util.Log;

public class MainActivity extends Activity {
    String msg = "Android : ";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d(msg, "The onCreate() event");
    }

    /** Called when activity is about to become visible. */
    @Override
    protected void onStart() {
        super.onStart();
        Log.d(msg, "The onStart() event");
    }

    /** Called when another activity is taking focus. */
    @Override
    protected void onPause() {
        super.onPause();
        Log.d(msg, "The onPause() event");
    }

    /** Called when the activity is no longer visible. */
    @Override
    protected void onStop() {
        super.onStop();
        Log.d(msg, "The onStop() event");
    }

    /** Called just before the activity is destroyed. */
    @Override
    public void onDestroy() {
        super.onDestroy();
        Log.d(msg, "The onDestroy() event");
    }
}
```


Multiple Activities or Fragments (more later)

- ❑ Simple applications with a single activity
- ❑ Complex applications with multiple activities
- ❑ Complex applications with a single activity and a group of fragments
 - Use multiple activities if really needed
 - Fragments can be used in other activities (reuse code), so more efficient

Passing Data between activities

- ❑ When you have multiple activities or fragments you most likely need to pass data between them using :
 - Intent
 - Bundle
 - SharedPreferences
- ❑ We will be considering each of these in turn
- ❑ While we have been discussing **activities**, however, let's first briefly consider the other three main components of Android applications namely, **services**, **broadcast providers** and **content providers**.

Services

- ❑ A service is a component that **runs in the background to perform long-running operations without needing to interact with the user.**
- ❑ For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.
- ❑ A service can essentially take two states

Started A service is started when an application component, such as an activity, starts it by calling `startService()`. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.

Bound A service is bound when an application component binds to it by calling `bindService()`. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with inter-process communication (IPC)

Services (cont'd)

- ❑ To create a service, first create a Java class extending the Service base class or a subclass. The Service base class defines various callback methods, including the following:

onStartCommand() The system calls this method when another component, e.g. an activity, requests that the service be started, by calling `startService()`. To stop the service when its work is done, you must call `stopSelf()` or `stopService()` methods.

onBind() The system calls this method when another component wants to bind with the service by calling `bindService()`. If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an `IBinder` object

onUnbind() The system calls this method when all clients have disconnected from a particular interface published by the service.

Services (cont'd)

- ❑ To create a service, first create a Java class extending the Service base class or a subclass. The Service base class defines various callback methods, including the following:

onRebind() The system calls this method when new clients connect to the service.

onCreate() The system calls this method when the service is first created using onStartCommand() or onBind(). This call is required to perform one-time setup.

onDestroy() The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners and receivers.

Broadcast Providers

- ❑ Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called **events** or **intents** (we will discuss intents and intent filters further.)
- ❑ For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

There are following two important steps to make BroadcastReceiver work for the system broadcasted **intents**:

1. Creating the Broadcast Receiver.
2. Registering Broadcast Receiver

Broadcast Providers (cont'd)

There are several system generated events defined as final static fields in the Intent class. The following table lists a few important **system events**:

android.intent.action.**BATTERY_CHANGED** Sticky broadcast containing the charging state, level, and other information about the battery.

android.intent.action.**BATTERY_LOW** Indicates low battery condition on the device.

android.intent.action.**BATTERY_OKAY** Indicates the battery is now okay after being low.

android.intent.action.**BOOT_COMPLETED** This is broadcast once, after the system has finished booting.

android.intent.action.**BUG_REPORT** Show activity for reporting a bug.

android.intent.action.**CALL** Perform a call to someone specified

android.intent.action.**CALL_BUTTON** The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.

android.intent.action.**DATE_CHANGED** The date has changed.

android.intent.action.**REBOOT** Have the device reboot.

Content Providers

- ❑ A content provider component **supplies data from one application to other on request**. Such requests are handled by the methods of the ContentResolver class.
- ❑ A content provider can use different ways to store its data such as in a **database, files**, or even **over a network**. Each Android application runs in its own process with its own permissions which keeps an application data hidden from other applications. Content providers become very useful when it is required to share data across applications.
- ❑ Content providers **let you centralize content in one place** and have many different applications access it as needed.
- ❑ A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using **insert()**, **update()**, **delete()**, and **query()** methods.
- ❑ In most cases this data is stored in an **SQLite database**.

SQLite

- ❑ SQLite is an open source SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation. SQLite supports all the relational database features. To access this database, you don't need to establish any kind of connections for it like JDBC, ODBC etc.
- ❑ Database - Package The main package is `android.database.sqlite` that contains the classes to manage your own databases.
- ❑ Database - Creation In order to create a database you need to call the method `openOrCreateDatabase` with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object. Its syntax is given below:
- ❑ `SQLiteDatabse mydatabase = openOrCreateDatabase("your database name", MODE_PRIVATE, null);`

Database - Insertion

- ❑ We can create or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below:

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS  
TutorialsPoint (Username VARCHAR, Password VARCHAR);");  
mydatabase.execSQL("INSERT INTO TutorialsPoint  
VALUES ('admin', 'admin');");
```

- ❑ This will insert some values into our table in our database. Another method that also does the same job but take some additional parameter is given below

```
execSQL(String sql, Object[] bindArgs)
```

This method not only insert data, but also used to update or modify already existing data in database using bind arguments.

Database - Fetching

- ❑ We can retrieve anything from database using an object of the Cursor class.
- ❑ We will call a method of this class called `rawQuery` and it will return a `resultset` with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatabase.rawQuery  
("Select * from TutorialsPoint",null);  
resultSet.moveToFirst();  
String username = resultSet.getString(1);  
String password = resultSet.getString(2);
```

Other SQLite functions

- ❑ **getColumnCount()** This method returns the total number of columns of the table.
- ❑ **getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column.
- ❑ **getColumnName(int columnIndex)** This method returns the name of the column by specifying the index of the column.
- ❑ **getColumnNames()** This method returns the array of all the column names of the table.
- ❑ **getCount()** This method returns the total number of rows in the cursor.
- ❑ **getPosition()** This method returns the current position of the cursor in the table.
- ❑ **isClosed()** This method returns true if the cursor is closed and returns false otherwise.



Intents and Intent Filters

Intents

- ❑ An Intent is a **messaging object** you can use to request an action from another app component. Intents facilitate communication between components.
- ❑ **Asynchronous messages** which allow applications to **signal when particular events have occurred**.
- ❑ Intents are sent to Android using method calls such as *startActivity()* when creating a new screen of content.

```
Intent i = new Intent(this, SecondActivity.class);  
startActivity(i);
```

- ❑ When sending intents, they can contain data:

```
i.putExtra("name", "Matthew");
```

- ❑ Then we can retrieve that data in the activity that was launched:

```
Intent i = getIntent();  
String name = i.getStringExtra("name");
```

Intents (cont'd)

- ❑ The intent can also be used to start a service
 - A Service is a component that performs operations in the background without a user interface (e.g. downloading a file)
- ❑ To start a broadcast (a message that any app can receive)
- ❑ Intents can be assigned *Actions* which applications can respond to upon receiving them.
- ❑ Examples (Many more exist!):
 - ACTION_MAIN
 - ACTION_VIEW
 - ACTION_EDIT
 - Events can be handled appropriately for each type of action (and any custom actions defined).

Intent Objects

An Intent object is a bundle of information which is used by the component that receives the intent plus information used by the Android system. An Intent object can contain the following components based on what it is communicating or going to perform:

Action This is mandatory part of the Intent object and is a string naming the action to be performed or, in the case of broadcast intents, the action that took place and is being reported.

Data The URI of the data to be acted on and the MIME type of that data. For example, if the action field is ACTION_EDIT, the data field would contain the URI of the document to be displayed for editing.

Intent message objects may also contain the following optional components:

Category, Extras, Component Name and Flags (see next page)

Additional Intent Attributes

In addition to the primary (required) action attribute there are a number of secondary attributes that you can include with an intent:

Category -- Gives additional information about the action to execute. For example, `CATEGORY_LAUNCHER` means it should appear in the Launcher as a top-level application

Data/type -- Specifies an explicit type (a MIME type) of the intent data. Normally the type is inferred from the data itself. By setting this attribute, you disable that evaluation and force an explicit type.

Component -- Specifies an explicit name of a component class to use for the intent. Normally this is determined by looking at the other information in the intent (the action, data/type, and categories) and matching that with a component that can handle it. If this attribute is set then none of the evaluation is performed, and this component is used exactly as is

Extras -- This is a **Bundle** of any additional information. This can be used to provide extended information to the component.

Intent Types

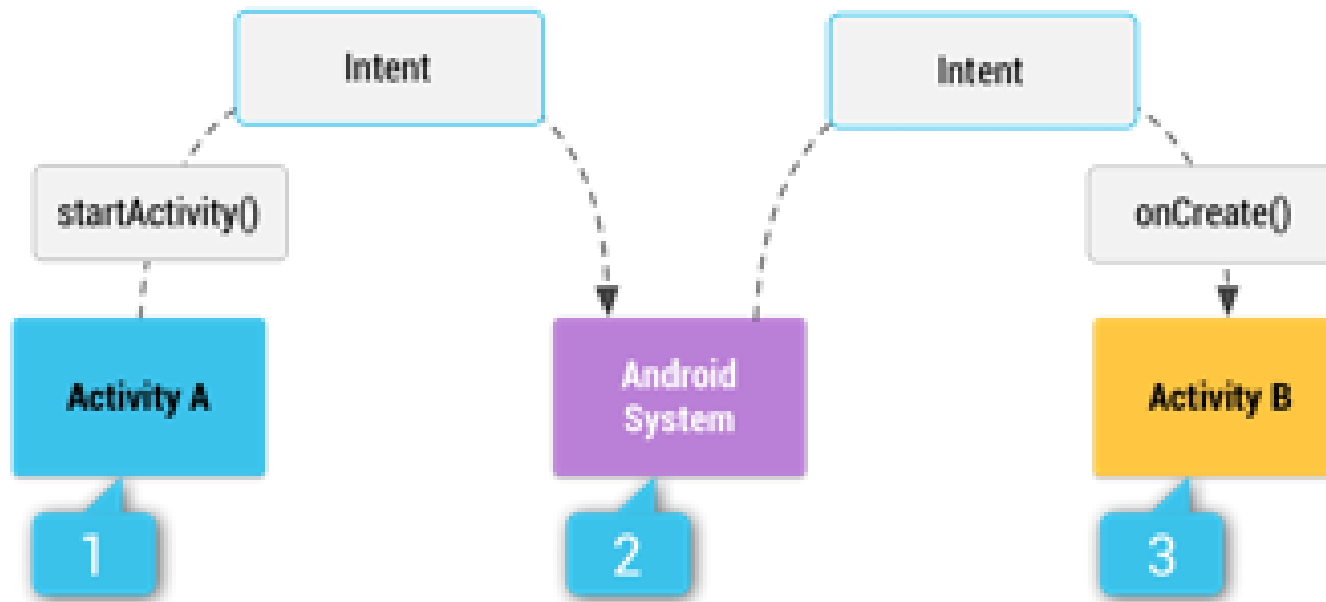
Explicit intents specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start.

- For example, you can start a new activity in response to a user action or start a service to download a file in the background.

Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.

- For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Implicit intents



How an implicit intent is delivered through the system to start another activity: **[1]** *Activity A* creates an [Intent](#) with an action description and passes it to [startActivity\(\)](#). **[2]** The Android System searches all apps for an intent filter that matches the intent. When a match is found, **[3]** the system starts the matching activity (*Activity B*) by invoking its [onCreate\(\)](#) method and passing it the [Intent](#).

<http://developer.android.com/reference/android/content/Intent.html>

Intent Filters

- ❑ Activities can respond based on the actions passed by Intents. These are defined by *intent filters*.
- ❑ Intent filters are used to inform Android the capabilities of the application.

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

- ❑ While intents are very useful for passing and moving between activities, they can be used outside of the application.

Intents - Facilitate communications between components

☐ To start an **activity**

- You can start a new instance of an Activity by passing an Intent to startActivity()

☐ To start a **service**

- A Service is a component that performs operations in the background without a user interface (e.g. download a file)

☐ To start a **broadcast**

- A broadcast is a message that any app can receive

Navigating Mobile Applications

- ❑ Most useful applications require more than one screen's worth of content to prevent information overload.
- ❑ So how do we move from one screen to another?
 - In Android movement between screens is often controlled by the **ActionBar** or via **Intents**

Action Bar

- ❑ Interface element that was introduced in Android 3.0+ to provide another element of navigation in the application.
- ❑ Features:
 - Makes important or common actions visible and accessible to the user in a consistent manner.
 - Enables easy **switching between views**.
 - Can be easily customised to give your application a unique look.
- ❑ ActionBar has an XML declaration for each Activity used.
 - Located under **menu** in the resource directory.
- ❑ Action Bar is a highly recommended addition to Android applications and is supported by the Android Support Library for older applications.

Action Bar Structure



1. **App Icon:** Identify of your application. Also provides “up” level navigation back to the main activity.
2. **View Control:** Enables uses to switch between different view screens for viewing data. Example: List or Grid display of items.
3. **Action Buttons:** Provide easy access to important functions of your application.
4. **Action overflow:** Access to less used actions such as an About screen or application settings.

Source: <http://developer.android.com/design/patterns/actionbar.html>

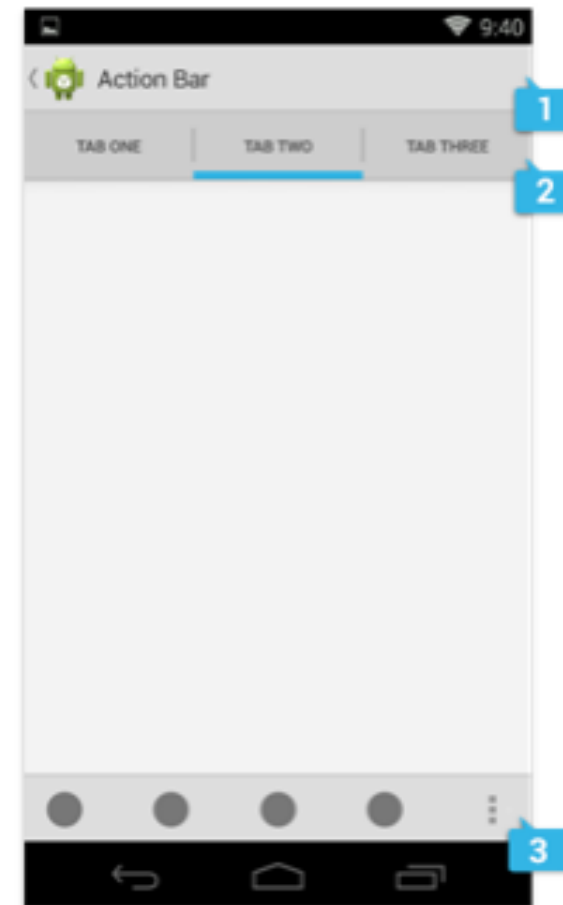
Split Action Bars

When splitting up content across multiple action bars, you generally have three possible locations for action bar content:

1. Main action bar If the user can navigate up the hierarchy from a given screen, the main action bar contains the up caret, at a minimum.

2. Top bar To allow the user to quickly switch between the views your app provides, use tabs or a spinner in the top bar.

3. Bottom bar To display actions and, if necessary, the action overflow, use the bottom bar.



Action Buttons



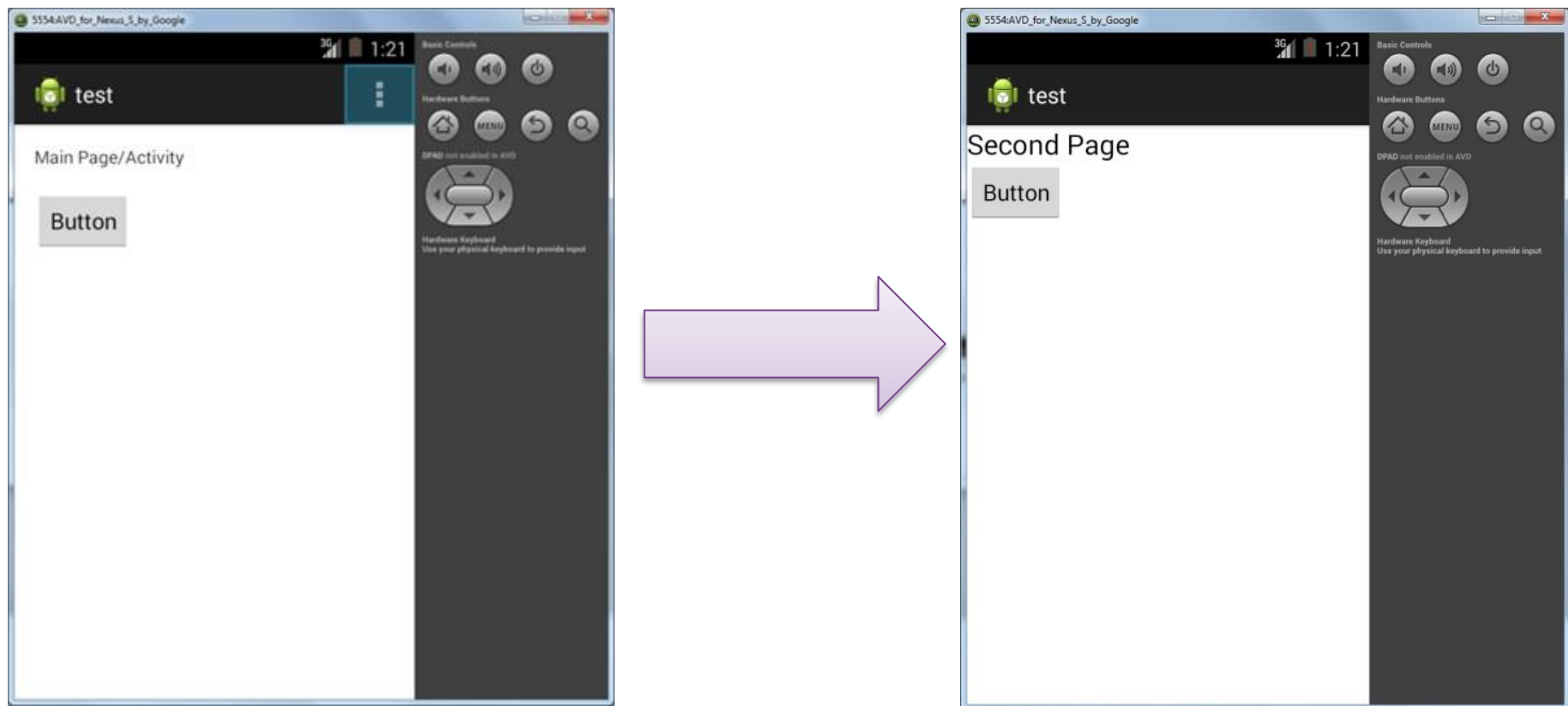
- ❑ *Action buttons* on the action bar surface your app's most important activities. Think about which buttons will get used most often, and order them accordingly.
- ❑ Depending on available screen real estate, the system shows your most important actions as action buttons and moves the rest to the action overflow.
- ❑ The action bar should show only those actions that are available to the user. If an action is unavailable in the current context, hide it. Do not show it as disabled
- ❑ For guidance on prioritizing actions and action buttons, use the **FIT** scheme:

(F)requent: Will people use this action frequently

(I)mportant: Do you want people to discover this action because it's important or a good selling point?

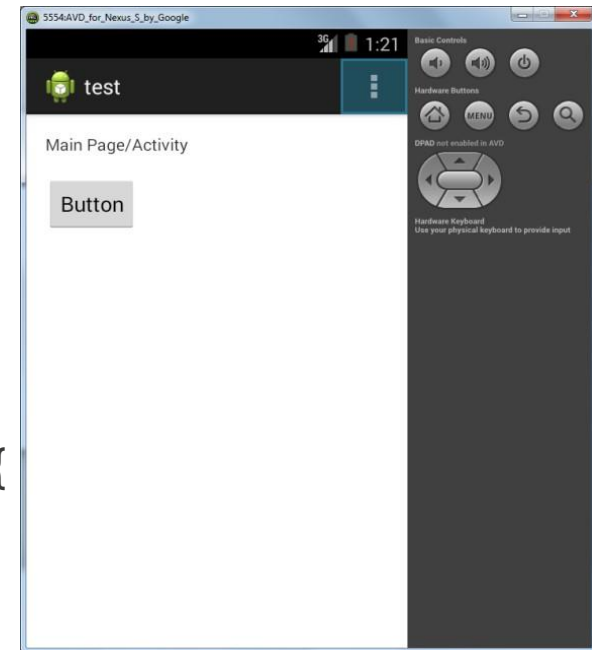
(T)ypical: Is the button typically presented as a first-class action in similar apps?

Application with Multiple Screens using Intent



To Start an Activity – Main Activity

```
public class MainActivity extends Activity {  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
        Button btn = (Button) findViewById(R.id.button1);  
        btn.setOnClickListener(new View.OnClickListener() {  
            @Override  
            public void onClick(View v) {  
                Intent intent = new Intent(MainActivity.this,  
                    SecondActivity.class);  
                startActivity(intent);  
            }  
        });  
    }  
}
```



public Intent (Context, Class)

- Used to create an intent for a specific component (explicit). It creates an intent that is intended to execute a hard-coded class name

To add Activities – AndroidManifest.xml

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    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>
    <activity
        android:name="com.example.test.SecondActivity" >
        </activity>
</application>
```

Using Intent To Pass Primitive Data

- ❑ An intent not only allows you to start another activity, but it enables sending data to the activity
 - There is a list of different types of data that can be carried by an intent, passed as the second parameter (check the API):
 - `intent.putExtra(String name, String value)`
 - `intent.putExtra(String name, double value)`
 - `intent.putExtra(String name, int[] value)`
- ❑ Example:
 - In the first activity
 - `Intent intent = new Intent(MainActivity.this, SecondActivity.class);`
 - `Intent.putExtra("message", msg);` //msg has String data type
 - `startActivity(intent);`
 - In the second activity

Get Data from Intent

- ❑ One way to get data from the intent is to use `getIntent()`; that will return the intent that started the second activity
- ❑ From this returned intent, we can retrieve the data it is carrying by using a right method depending on the type of data
 - E.g. `Intent intent=getIntent();`
 - `String msg = intent.getStringExtra("message");`



Serializable data, Parcels and Bundles

Pass Data as Bundle

- ❑ You can also add a bundle to an intent: *putExtra(String name, Bundle value)*
- ❑ Bundle is a “mapping from String keys to various Parcelable values”
- ❑ Bundle enables passing data between activities and fragments
- ❑ You can use put and get methods for different data types, even arraylists
 - Bundle bundle=new bundle();
 - Bundle.putString(name, value);
 - intent.putExtra(String name, Bundle value); // in the sender activity
 - getIntent().getBundleExtra (String name); // in the receiver activity

OR without a key: *intent. putExtras(bundle); // in the sender activity*
getIntent().getExtras(); // in the receiver activity

Passing Objects Using Bundle - Serializable

- ❑ Bundle can be also used to pass objects (serializable or parcelable) rather than primitive data
- ❑ To pass data as serializable
- ❑ First, you need to make sure the class of your object implements Serializable
- ❑ Then you can call the bundle method of
`putSerializable(String key, Serializable value)`

E.g. In MainActivity:

```
Bundle bundle = new Bundle();  
bundle.putSerializable("student", student); // refers to a  
serializable student
```

- ❑ In the other activity or fragment:
`getSerializable ("student");`

Passing Objects Using Bundle - Parcelable

- ❑ You can also use **Parcelable** to pass objects if Student implements Parcelable

```
bundle.putParcelable ("student", student);
```

- ❑ In the other activity or fragment

```
getParcelable ("student");
```

- ❑ If you want to use Parcelable, you need to:

1. Create a class that implements Parcelable
2. Implement all get and set methods for its attributes
3. Implement writeToParcel to write values to parcel in a specific order
4. Implement the constructor for reading in values in the same order

Parcelable: Example

- ❑ An Example: A student class with two attributes of id and name

The class implements Parcelable

```
public class Student implements Parcelable{
```

Implement all get and set methods for its attributes

Implement writeToParcel

```
public void writeToParcel(Parcel parcel, int flags){
```

```
    parcel.writeInt(id);
```


```
    parcel.writeString(name); }
```

Implement the constructor for reading in values in the same order you wrote them

```
public Student(Parcel in) {
```

```
    this.id= in.readInt();
```

```
    this.name = in.readString(); }
```



Fragments and Shared Fragments

(To be continued in Lecture 8)

Fragments

“A Fragment represents a behavior or a portion of user interface in an Activity”

- ❑ A fragments is like sub-activity that lives inside an activity
- ❑ Fragments can be used to provide reuse and modularity
- ❑ Fragments can be used for building dynamic and multi-pane user interfaces

E.g. when working with different screen sizes, a small screen can show one fragment at a time, and a large screen can show all two or three of them.



Fragments (cont'd)

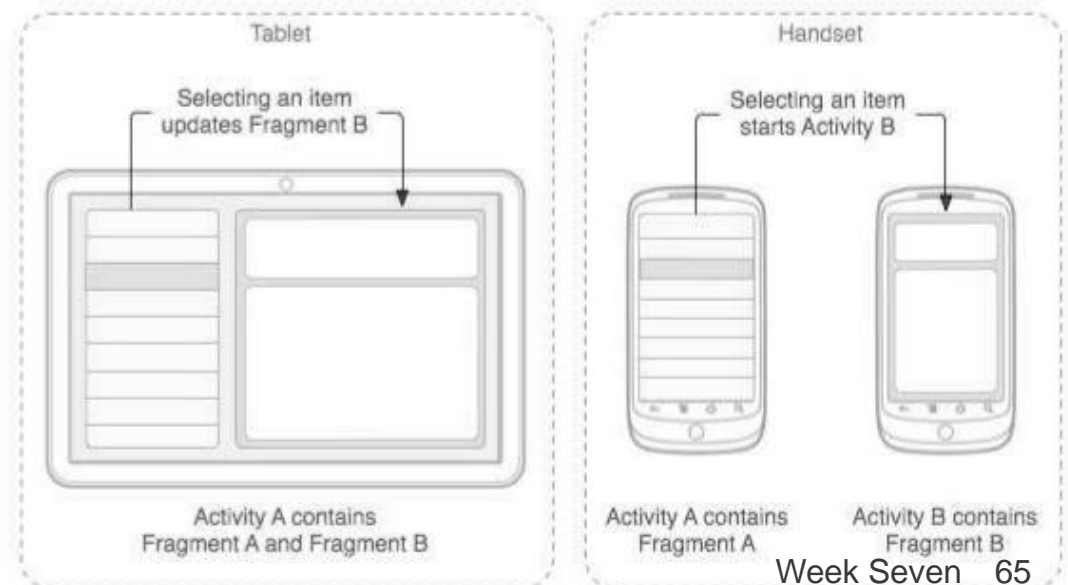
A Fragment is a piece of an application's user interface or behaviour that can be placed in an Activity which enable more modular activity design. Fragments have the following features:

- ☐ A fragment has its own layout, behaviour and lifecycle callbacks.
- ☐ Fragments can be added or removed while the activity is running.
- ☐ Multiple fragments can be combined in an activity to build a multipane UI.
- ☐ A fragment can be used in multiple activities.
- ☐ Fragment life cycle is closely related to the lifecycle of its host activity. If the activity is paused, all associated fragments available will also be stopped.
- ☐ A fragment can implement a behaviour that has no user UI component.
- ☐ Fragments were added to the Android API in Honeycomb version (API 11)

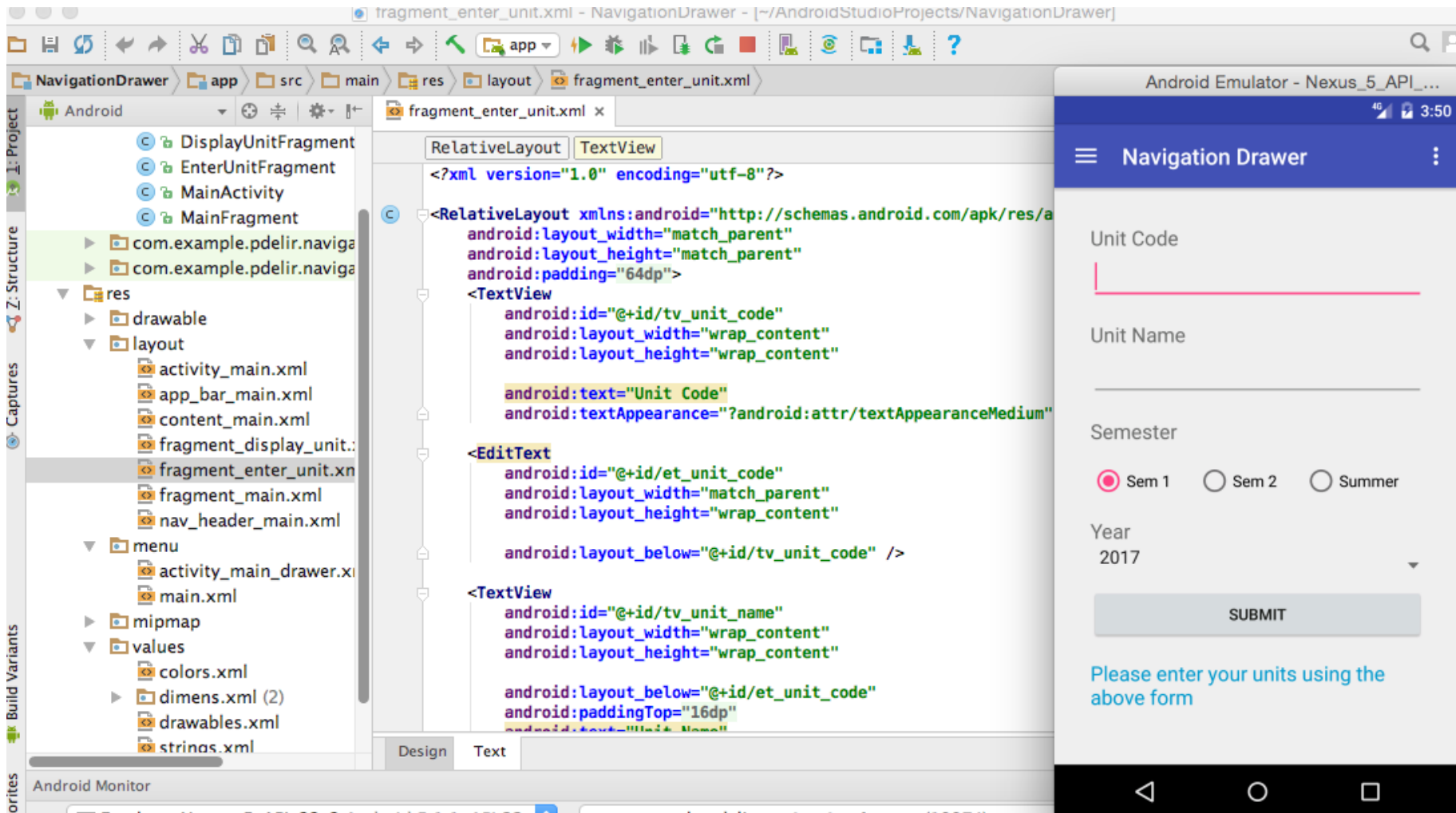
You create fragments by extending Fragment class and you can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a <fragment> element

Fragments and Navigation Drawer

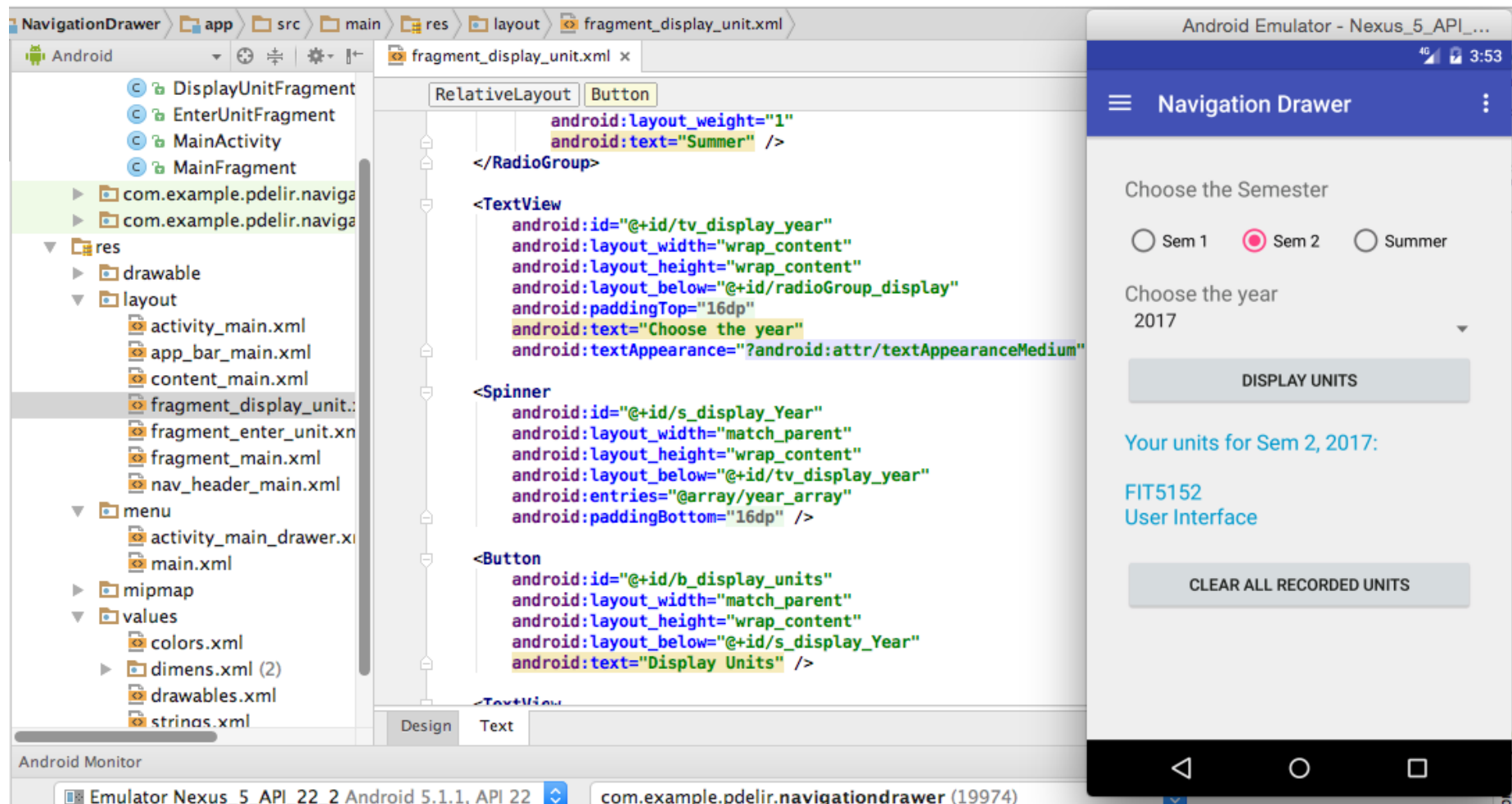
- ❑ You can have multiple fragments for one single MainActivity
- ❑ Navigation Drawer is a good example
 - It helps to avoid code duplication and overheads
- ❑ You create a fragment class by extending Fragment
E.g. public class DisplayUnitFragment extends **Fragment** {
- ❑ You can then have a group of fragment layouts where each is associated with a fragment
- ❑ E.g. the fragment layout for DisplayUnitFragment.java:
fragment_display_unit.xml



Enter Unit Fragment



Display Unit Fragment



Fragments (cont'd)

- ❑ To associate a fragment with a layout, you implement the `onCreateView()` which will return a `View`

```
public class DisplayUnitFragment extends Fragment {  
    ...  
    public View onCreateView(LayoutInflater inflater,  
        ViewGroup container, Bundle savedInstanceState) {  
        vDisplayUnit =  
            inflater.inflate(R.layout.fragment_display_unit,  
                container, false);  
        ...  
        return vDisplayUnit;  
    }  
}
```

Inflate

- ❑ In a simple application, we have an `activity_main.xml` and `MainActivity.java`, and the xml layout is inflated and rendered by `setContentView(R.layout.activity_main);`
- ❑ Working with fragments, you need to explicitly inflate views to instantiate a layout XML file into its corresponding View objects and create new views and viewgroups:

```
vDisplayUnit =  
    inflater.inflate(R.layout.fragment_display_unit,  
        container, false);
```

- ❑ You can use the created view to retrieve other views/widgets

```
Button bClearUnits = (Button)  
    vDisplayUnit.findViewById(R.id.b_clear_units);
```

Fragments (cont'd)

- ❑ To add, remove or replace fragments in your activity, you need to use `FragmentManager` and `FragmentTransaction`
- ❑ Example code from Navigator Drawer Tutorial:

```
FragmentManager  
fragmentManager=getFragmentManager();  
fragmentManager.beginTransaction()  
    .replace(R.id.content_frame, nextFragment)  
    .commit();
```

- ❑ `Replace()` method replaces whatever is in the `fragment_container` view with the selected fragment

There are different ways to work with fragments. Android Fragment Tutorial:

<https://developer.android.com/training/basics/fragments/creating.html>

<https://developer.android.com/guide/components/fragments.html>

<http://developer.android.com/reference/android/app/FragmentManager.html>

Passing Data Using SharedPreferences

- ❑ **SharedPreferences** enables to save and retrieve persistent key-value pairs of primitive data types between multiple fragments and activities
- ❑ A SharedPreferences object points to a file containing key-value pairs
- ❑ To get a SharedPreferences object for your application, you call `getSharedPreferences (String file, int operatingMode):`

E.g.: `SharedPreferences spMyUnits = getActivity().
getSharedPreferences ("myUnits", Context.MODE_PRIVATE);`

`//Use 0 or MODE_PRIVATE for the default operation`

To write to a shared preferences file, you need to create a `SharedPreferences.Editor` by calling `edit()`

`SharedPreferences.Editor eMyUnits = spMyUnits.edit();`

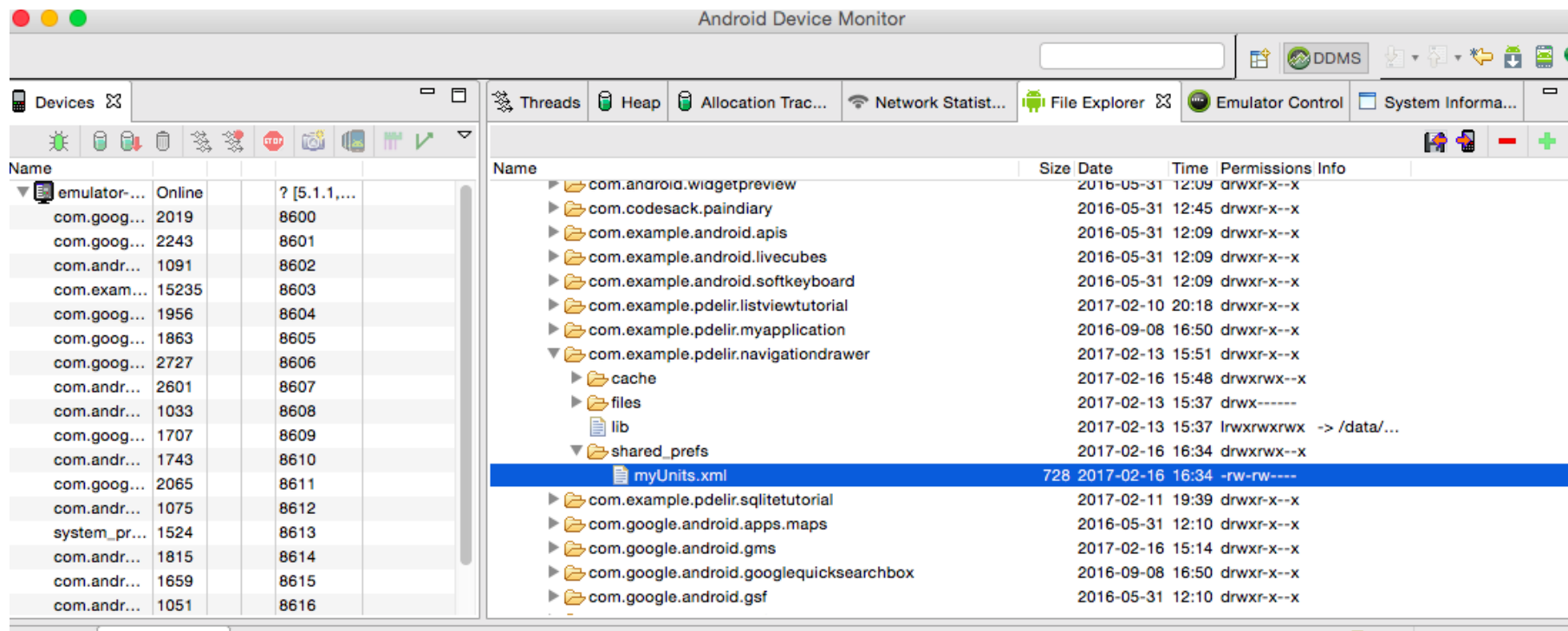
SharedPreferences (cont'd)

- ❑ You will pass the keys and values to the Editor with available methods such as `putLong()`, `putInt()`, `putString()`:
 - `eMyUnits.putString("myUnits", jaMyUnits.toString());`
- ❑ You will finally call `Apply` or `commit` to save changes (*commit returns a Boolean value true if successful*)
 - `eMyUnits.apply();`
- ❑ To read values, use `SharedPreferences` methods like `getString()` and pass the key
 - `String sMyUnits = spMyUnits.getString("myUnits", null);`

//the second parameter is the value to return if this key does not exist

Where Are SharedPreferences Stored?

- ❑ SharedPreferences are stored as an xml file under `data/data/yourapplication/shared_prefs`
- ❑ Use ADM to view and export to a different location to open and browse





Threads and AsyncTask

Threads

- ❑ When an Android application is launched, the system creates a thread of execution for the application, called **"main."**
- ❑ **The main thread aka the UI thread**
- ❑ The UI Thread is the main thread of execution for your application
 - It is the thread in which the application **interacts with components from the Android UI toolkit**
 - It dispatches events to the appropriate user interface widgets

Single Thread Model

- ❑ When an Android application is launched, the system creates a "main." thread, aka the UI thread
- ❑ The UI Thread is the single thread of execution for your application
- ❑ It is the thread in which the application interacts with components from the Android UI toolkit
- ❑ Two rules to Android's single thread model:
 1. Do not access the Android UI toolkit from outside the UI thread
 2. Do not block the UI thread
- ❑ When performing intensive work such as network access or database queries
- ❑ If you have such operations, do them in background or worker threads

UI Thread

- ❑ When performing intensive work (such as network access or database queries), it can block the UI thread
 - From the user's perspective, the application appears to hang:
The infamous "application not responding" (ANR) dialog
- ❑ Two rules to Android's single thread model:
 1. **Do not block the UI thread**
 - If you have operations to perform that are not occurring instantly, you should make sure to do them in separate threads ("background" or "worker" threads)
 2. **Do not access the Android UI toolkit from outside the UI thread**

An example

- ❑ This code creates a new thread to handle the network operation
- ❑ It modifies the ImageView from another thread instead of the UI thread
- ❑ It violates the second rule: accesses the Android UI toolkit from outside the UI thread

```
public void onClick(View v) {  
    new Thread(new Runnable() {  
        public void run() {  
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");  
            mImageView.setImageBitmap(b);  
        }  
    }).start();  
}
```

Source: <https://developer.android.com/guide/components/processes-and-threads.html>

UI Thread (cont'd)

- ❑ Android offers several ways to access the UI thread from other threads:

`Activity.runOnUiThread(Runnable)`

`View.post(Runnable)` //The runnable will be in UI thread

`View.postDelayed(Runnable, long)`

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            final Bitmap bitmap = loadImageFromNetwork("http://example.com/image.png");
            mImageView.post(new Runnable() {
                public void run() {
                    mImageView.setImageBitmap(bitmap);
                }
            });
        }
    }).start();
}
```

Source: <https://developer.android.com/guide/components/processes-and-threads.html>

- ❑ In more complex interactions, you can use a Handler and attach it to the UI thread

For more details refer to <https://developer.android.com/training/multiple-threads/communicate-ui.html>

Other Options

- ❑ A better solution is to extend the **AsyncTask class**, which simplifies the execution of worker thread tasks that need to interact with the UI
- ❑ It enables tasks to be performed asynchronously

AsyncTask

- ❑ AsyncTask allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers
- ❑ AsyncTask must be subclassed to be used
private class Factorial **extends** AsyncTask<String, Void, String> {
- ❑ The subclass will override at least one method (doInBackground(Params...))
protected String doInBackground(String... params) {
- ❑ Most often will override a second method (onPostExecute(Result))
protected void onPostExecute(String result) {
- ❑ Other methods: onPreExecute() and onProgressUpdate(Progress...)

Source: <http://developer.android.com/reference/android/os/AsyncTask.html>

Sample Code

```
private class Factorial extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... params) {
        int from = Integer.parseInt(params[0]);
        int to = Integer.parseInt(params[1]);
        String result = "";
        for(int i = from; i <= to; i++) {
            result = i + "! = " + calculatingFactorial(i) + "\n ----- \n" + result;
        }
        return result;
    }
    @Override
    protected void onPostExecute(String result) {
        tv.setText(result);
    }
}
```

Executed as below:

```
public void calculate() {
    Factorial f = new Factorial();
    f.execute(new String[] {numberfrom, numberto});
}
```

Types

❑ The three types used by an asynchronous task are the following:

❑ `AsyncTask<Params, Progress, Result>`

Params: the type of the parameters sent to the task (`doInBackground()`)

It can be an array of objects

Progress: the type of the progress units published during the background computation (optional)

Result: the type of the result of the background computation (`onPostExecute()`)

❑ To mark a type as unused, use `Void`

E.g.

```
private class Factorial extends AsyncTask<String, Void, String>
{..}
```

```
private class DownloadImageTask extends AsyncTask<String, Void,
Bitmap> {..}
```

Source: <http://developer.android.com/reference/android/os/AsyncTask.html>

AsyncTask Steps

❑ **onPreExecute()**

- Invoked on the UI thread before the task is executed.
- E.g. showing a progress bar in the user interface.

❑ **doInBackground(Params...)**

- Invoked on the background thread immediately after `onPreExecute()` finishes executing.
- The parameters of the asynchronous task are passed to this step
- The result of the computation must be returned by this step and will be passed back to the last step

Source: <http://developer.android.com/reference/android/os/AsyncTask.html>

Steps

❑ **onProgressUpdate(Progress...)**

This method is used to display any form of progress in the user interface while the background computation is still executing. It can be used to animate a progress bar or show logs in a text field.

```
protected void onProgressUpdate(Integer... progress) {  
    setProgressPercent(progress[0]);  
}
```

❑ **onPostExecute(Result)**

Invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step.

```
protected void onPostExecute(String result) {  
    tv.setText(result);  
}
```

Source: <http://developer.android.com/reference/android/os/AsyncTask.html>

Another Example

```
private class LoadText extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... txt) {
        int max = Integer.parseInt(txt[0]);
        String text = "";
        for(int i = 0; i < max; i++){
            text += " " + i;
        }
    }
    @Override
    protected void onPostExecute(String result) {
        tv.setText(result);
    }
}

public void readText(View view) {
    tv.setText("");
    LoadText task = new LoadText();
    task.execute(new String[] {"10000"});
}
```





URLConnection

URLConnection and AsyncTask

```
private class GetRESTResponse extends AsyncTask<String, Void, String> {
    @Override
    protected String doInBackground(String... args) {
        ...
        try {
            url = new URL\("http://<ip\_address>:8080/FriendsDB/webresources/friend.friends"\);
            conn = (URLConnection) url.openConnection();

            conn.setRequestMethod("GET");
            conn.setRequestProperty("Content-Type", "application/json");
            conn.setRequestProperty("Accept", "application/json");
            InputStream instream = new BufferedInputStream(conn.getInputStream());
            BufferedReader buffer = new BufferedReader(new InputStreamReader(instream));

            while ((s = buffer.readLine()) != null) { resmsg += s;}

            Log.i("Responses is ", resmsg);
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        finally { conn.disconnect(); } return resmsg;
    }
}
```

HttpURLConnection

- ❑ Android app can have network functionality
- ❑ To use http to send and receive data, HttpURLConnection can be used
- ❑ HttpURLConnection enables your Android app to interact with REST web services as well
- ❑ You can use http methods like GET, POST, PUT or DELETE in HttpURLConnection to access and modify data by invoking REST methods
- ❑ Before using HttpURLConnection, you make sure that the manifest file includes the internet permissions
- ❑ `<uses-permission
android:name="android.permission.INTERNET" />`

URLConnection (cont'd)

1. Get a new `URLConnection` by calling `URL.openConnection()`
 - `URLConnection conn = (URLConnection) url.openConnection();`
2. You will prepare the http request
 - The URI: `url = new URL("http://ip address:8080/FriendsDB/webresources/friend.friends");`
 - Use `setRequestMethod` for http methods (default is GET)
`conn.setRequestMethod("POST");`
 - Request headers
`conn.setRequestProperty("Content-Type", "application/json");`
`conn.setRequestProperty("Accept", "application/json");`
3. Read the request, e.g. from the stream returned by `getInputStream()`
`Scanner inStream = new Scanner(conn.getInputStream());`
4. Finally you disconnect it
`conn.disconnect();`

HttpURLConnection and URL

- ❑ It is very important that you use the correct URL to make http calls to the REST web service and its methods
- ❑ Check the path in the REST method that you are calling and make sure you have the right path
- ❑ Check: Your IP address, the port, the bucket (the root path, the method name, parameters (how many and what order))

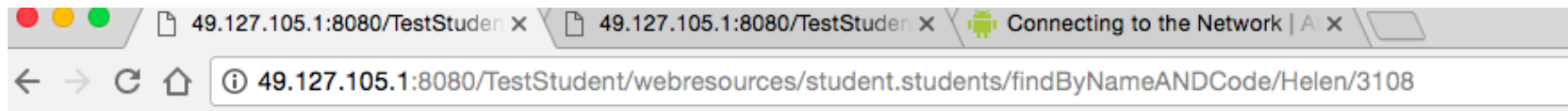
@GET

@Path("findByNameANDCode/{studname}/{studpostcode}")

@Produces({"application/json"})

- ❑ You can always check this in a browser before including in the Android app

<http://49.127.105.1:8080/TestStudent/webresources/student.students/findByNameANDCode/Helen/3108>



```
[{"courseid":{"courseid":5046,"coursename":"Mobile"},"studid":1,"studname":"Helen","studpostcode":"3108"}]
```

References

- ❑ <http://developer.android.com/design/patterns/actionbar.html>
- ❑ <http://developer.android.com/guide/components/intents.html>
- ❑ <http://developer.android.com/guide/components/intents-filters.html>
- ❑ <http://developer.android.com/reference/android/os/Bundle.html>
- ❑ <https://developer.android.com/training/basics/fragments/creating.html>
- ❑ <https://developer.android.com/guide/components/fragments.html>
- ❑ <http://developer.android.com/reference/android/app/FragmentManager.html>
- ❑ <http://developer.android.com/guide/components/processes-and-threads.html>
- ❑ <https://developer.android.com/training/multiple-threads/communicate-ui.html>
- ❑ <http://developer.android.com/reference/android/os/AsyncTask.html>
- ❑ <https://developer.android.com/reference/java/net/URLConnection.html>