SLIIT
Discover Your Future

Lecture 5

Main Components of Android Application

- Mobile Mindset
- Mobile Platforms and Application Development fundamentals
- Introduction to Android Operating System
- Mobile Interface Design Concepts and UI/UX Design Fundamentals
- Main Components of Android Application
- Data Handling in Mobile Platforms
- Handling Media in Android Applications
- Sensors in Android
- Security Aspects of Mobile Application development
- Android Services

SLIIT
Discover Your Future

Android Core Building Blocks

Android Core Building Blocks

- Make it easy to break them down into conceptual units.
- So that you can work on them independently, and then easily put them back together into a complete package.

Eg: Basic Features

- Login screen
- Main screen
- Play screen

Background tasks

- Network connection
- Caching data

Activity Manager

- ActivityManager class gives information about, and interacts with activities, services, and the containing processes.

- Responsible for creating, destroying and managing activities.

Activity Manager cont..

Eg:

- When the user starts an application for the first time, the Activity Manager will create its activity and put it onto the screen.
- Later, when the user switches screens, it will move that previous activity to a holding place.
- This way, if the user wants to go back to an older activity, it can be started more quickly.
- Older activities that the user hasn't used in a while will be destroyed in order to free more space for the currently active one.
- This mechanism is designed to help improve the speed of the user interface and thus improve the overall user experience.

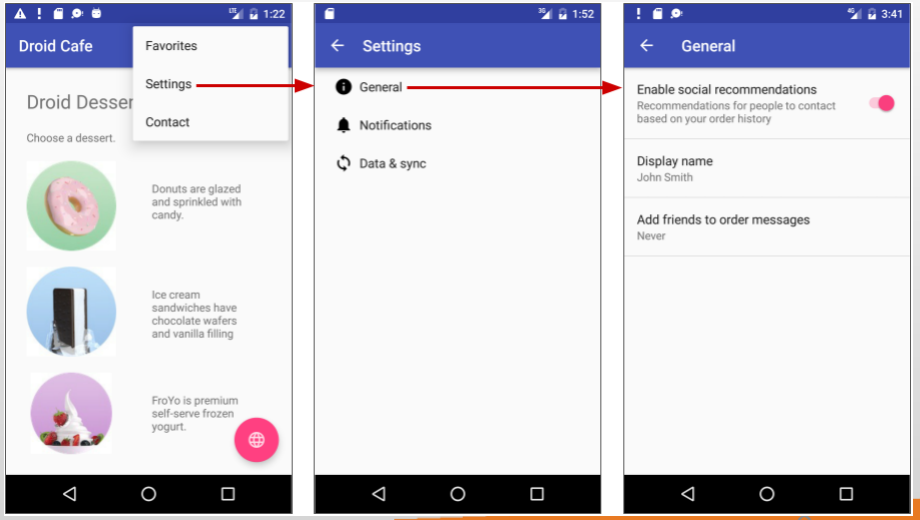
Activities

"A single screen that the user sees on the device at one time. "

- Most visible part of your application
- Just like a website has to provide some sort of navigation among various pages, an Android app should do the same.

Activities cont..

- Activities are Expensive!!!
- Launching single activity in Android involves:
  - Creating new Linux process.
  - Allocating memory for all the UI objects.
  - Retrieving the XML Layouts.
  - Setting up the whole screen.



How to create an Activity

```
package com.example.mad.helloapp;

import android.app.Activity;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends Activity {

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

Activity class loads it's UI component main.xml file

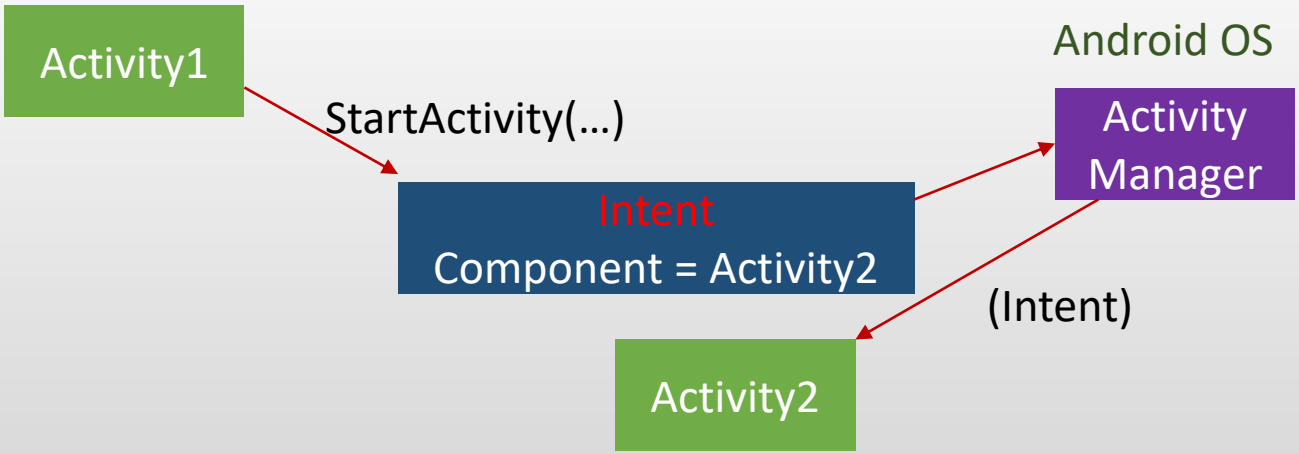
SLIIT
Discover Your Future

Activity in Manifest file

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="net.learn2develop.Activities"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon"
        android:label="@string/app_name">
        <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>
    <uses-sdk android:minSdkVersion="9" />
</manifest>
```

Intents

- The message that is passed between activities.
- Intents are asynchronous (the code that sends them doesn't have to wait for them to be completed).
- Allowing us to pass data between activities.



Intents

```
Intent myIntent = new Intent(this, MainMenu.class);
startActivity(myIntent);
```

Note:

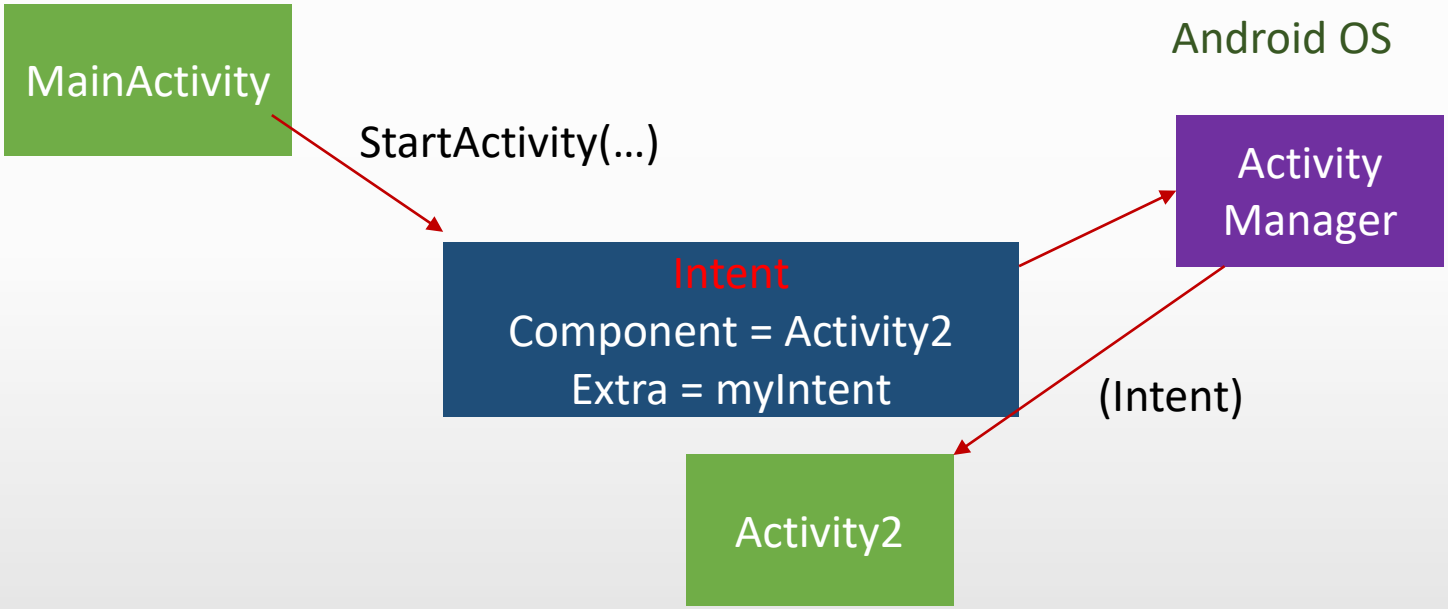
Intent class has a constructor that has two parameters.

1 – reference to the current activity (this)

2 – name of the activity we want to open

Intent Extras

- Passing data between Activities.



Intent Extras

```
public class MainActivity extends Activity {

    public String myExtra = "text";
    Button button1;
     @Override
     protected void onCreate(Bundle savedInstanceState) {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.activity_main);
         button1 = (Button)findViewById(R.id.button1);
         button1.setOnClickListener(new View.OnClickListener() {
             @Override
             public void onClick(View view) {
                 Intent intent = new Intent(MainActivity.this, Activity2.class);
                 intent.putExtra("MAIN_EXTRA", myExtra);
                 startActivity(intent);
             }
         });
     }
}
```

```
public class Activity2 extends AppCompatActivity {

    String takeExtra;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_2);
        Intent myIntent =  getIntent();
        takeExtra = myIntent.getStringExtra("MAIN_EXTRA");
    }
}
```

Explicit Intents

- Specifies the component to be invoked from the current activity.
- Can also pass the information from one activity to another.

Implicit Intents

- Doesn't specify the component.
- The sender specifies the type of receiver.

Eg:

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.

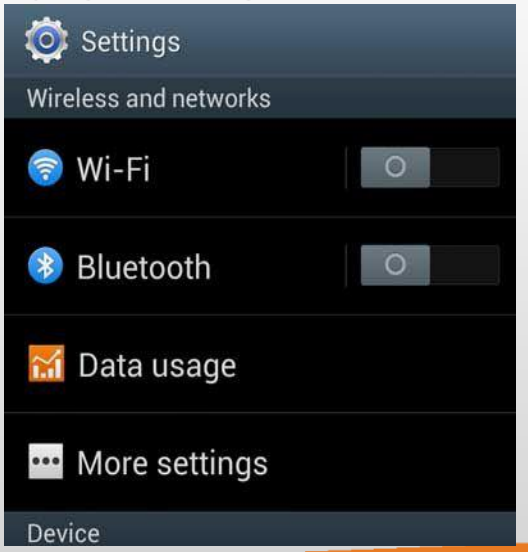
```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.google.com"));
startActivity(intent);
```

Android Services

- Services run in the background and don't have any user interface components.
- They can perform the same actions as activities, but without any user interface.
- Services are useful for actions that we want to perform for a while, regardless of what is on the screen.

Eg: - playing music in music player even as you are flipping between other applications

- Handle network transactions



Declare the services in Manifest file

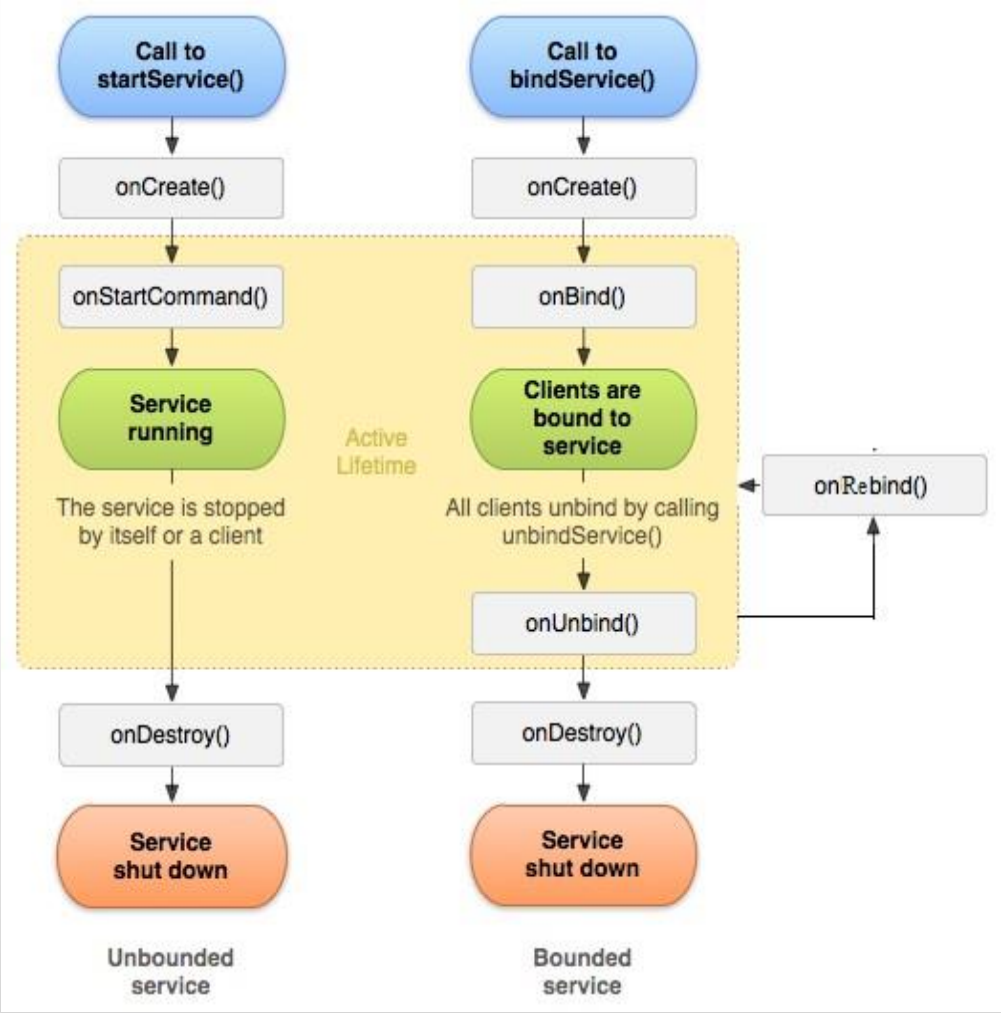
- Declare all services in your application's manifest file.

Note: Add a <service> element as a child of the <application> element.

```
<manifest ... >
  ...
  <application ... >
      <service android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

Android Service Life Cycle

- Two forms of a service
  1. Started
  2. Bound



Android Service Life Cycle cont..

Started Service

- Tells the system *about* something it wants to be doing in the background.

- By calling 'Context.startService()', it asks the system to schedule work for the service, to be run until the service or someone else explicitly stop it.

Android Service Life Cycle cont..

Bound Service

- Application can be exposed some of its functionality to other applications.
- By calling 'Context.bindService()', allows a long-standing connection to be made to the service in order to interact with it.

Broadcast Receivers

- Allows you to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens.

- Simply, Broadcast messages from other application or from the system itself.
  - Eg:
    - User can see when it reduces brightness after the battery runs under 20%.
    - Let other applications know that some data has been downloaded to the device and available for them to use.

Broadcast Receivers cont..

- Two important steps:

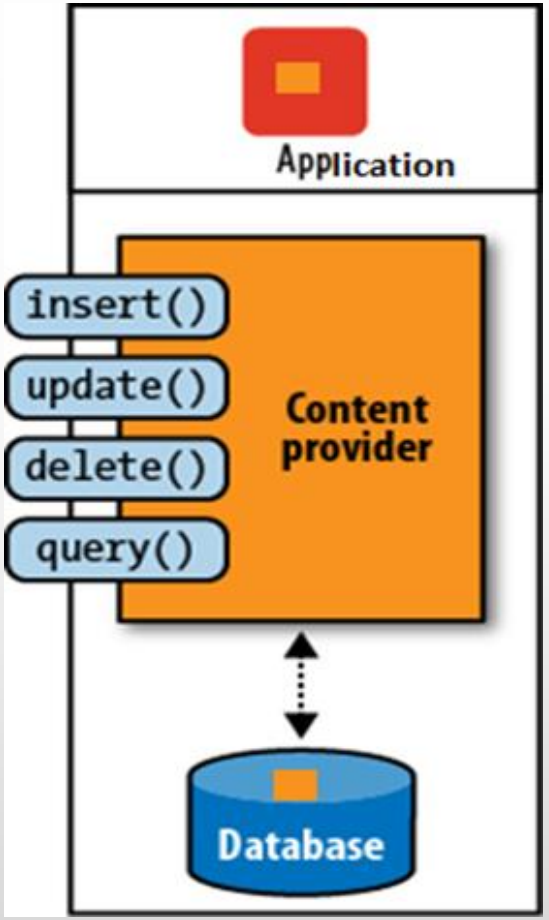
1. Creating the Broadcast Receiver

Extend the BroadcastReceiver class and do the implementation in onReceive() method.

2. Registering Broadcast Receiver

Content Providers

- Interfaces for sharing data between applications
- Behaves like a database.
- Data can be stored in a database, in files, or over a network.
- Let you centralize content in one place and have main different applications access it as needed.
- Implemented as a subclass of ContentProvider class.



Content Providers

- Manages several kinds of data such as audio, video, images, and personal contact information.

- Can be used to manage both structured data(Eg: SQLite relational databases) and unstructured data (Eg: image files).

Content Providers -Advantages

- Control the permission for accessing data.

- Abstract away the details for accessing different data sources in your application.

  Eg:

  your application might store structured records in a SQLite database, as well as video and audio files. You can use a content provider to access all of this data, if you implement this development pattern in your application

Create Content Provider

1. Create a Content Provider class that extends the *ContentProviderbaseclass.*

2. Define your content provider URI address which will be used to access the content.

3. Create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override *onCreate()* method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the *onCreate()* handler of each of its Content Providers is called on the main application thread.

Create Content Provider cont..

4. Implement Content Provider queries to perform different database specific operations.

5. Register your Content Provider in your activity file using <provider> tag.

Overridden methods for Content Providers

- **onCreate()** - Called when the provider is started.
- **query()** - Receives a request from a client. The result is returned as a Cursor object.
- **insert()** - Inserts a new record into the content provider.
- **delete()** - Deletes an existing record from the content provider.
- **update()** - Updates an existing record from the content provider.
- **getType()** - Returns the MIME type of the data at the given URI.