ITW202: Mobile Application

Unit IV: Developing for Android

Ms. Sonam Wangmo

Gyalpozhing College of Information Technology
Royal University of Bhutan

March 15, 2021

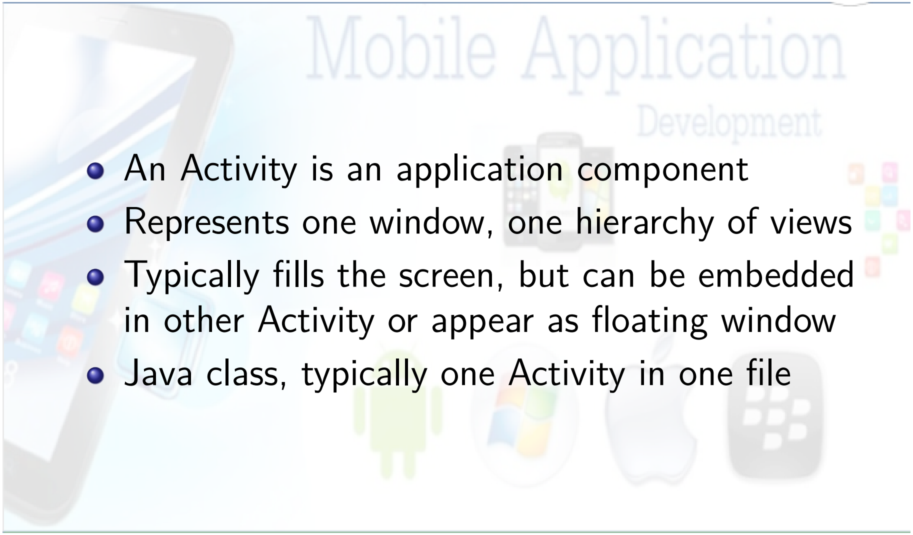
Activities and Intents

Contents

- Activities
- Defining an Activity
- Starting a new Activity with an Intent
- Passing data between activities with extras
- Navigating between activities

Activities

What is an Activity?

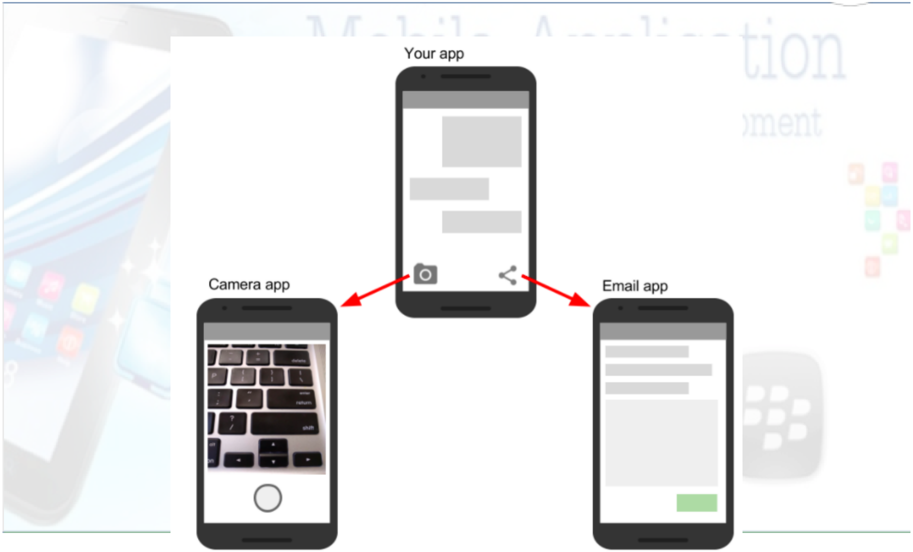


- An Activity is an application component
- Represents one window, one hierarchy of views
- Typically fills the screen, but can be embedded in other Activity or appear as floating window
- Java class, typically one Activity in one file

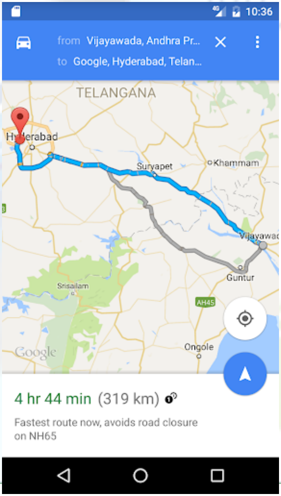
What does an Activity do?

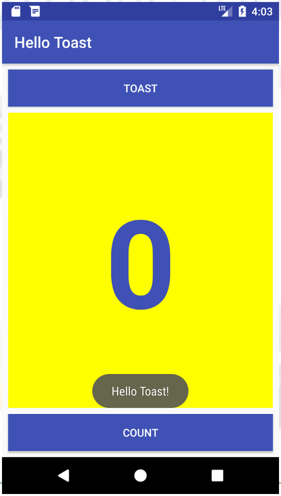
- Represents an activity, such as ordering groceries, sending email, or getting directions
- Handles user interactions, such as button clicks, text entry, or login verification
- Can start other activities in the same or other apps
- Has a life cycle—is created, started, runs, is paused, resumed, stopped, and destroyed

Examples of activities



Examples of activities


10:36
from Vijayawada, Andhra Pr...
to Google, Hyderabad, Telan...
TELANGANA
Hyderabad
Suryapet
Khammam
Vijayawada
Guntur
Srisailam
AH45
Ongole
Google
4 hr 44 min (319 km)
Fastest route now, avoids road closure on NH65



Apps and activities

- Activities are loosely tied together to make up an app
- First Activity user sees is typically called "main activity"
- Activities can be organized in parent-child relationships in the Android manifest to aid navigation

Layouts and Activities

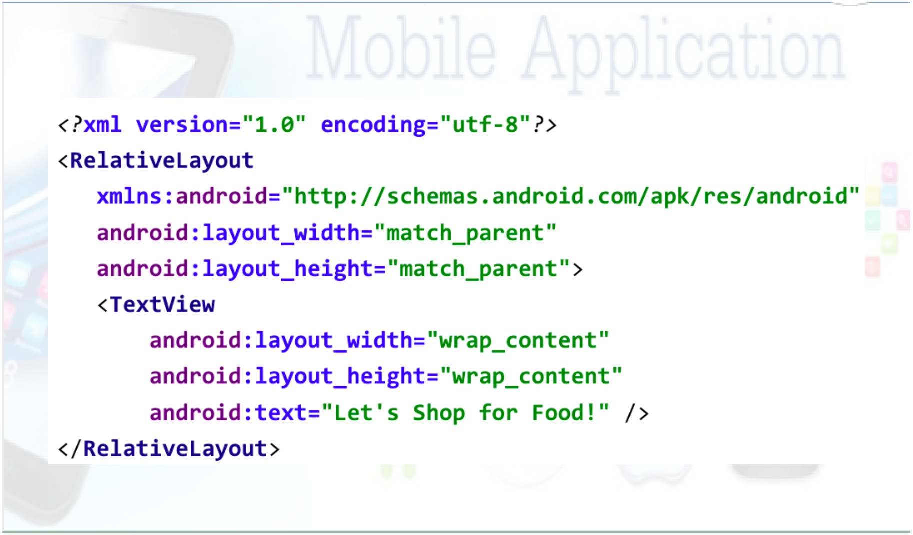
- An Activity typically has a UI layout
- Layout is usually defined in one or more XML files
- Activity "inflates" layout as part of being created

Implementing Activities

Implement new activities

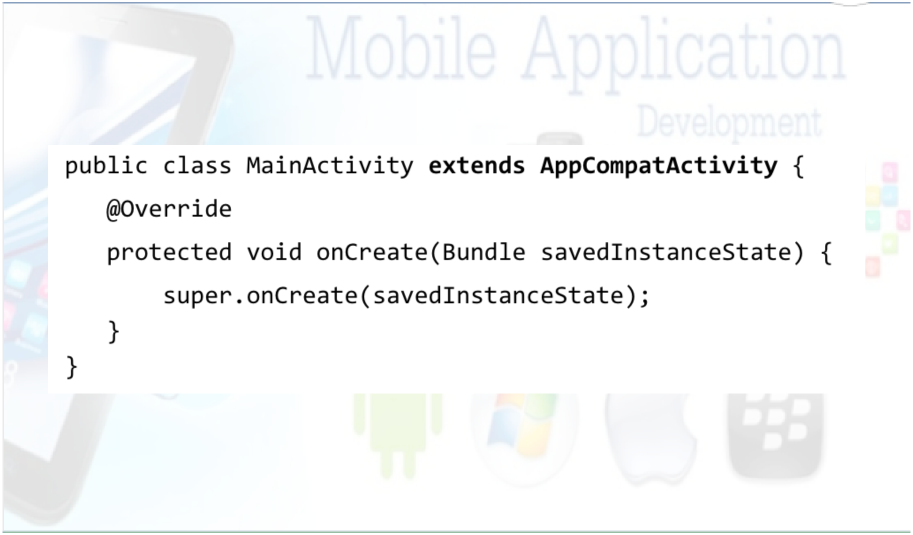
1. Define layout in XML
2. Define Activity Java class
   extends AppCompatActivity
3. Connect Activity with Layout
   Set content view in onCreate()
4. Declare Activity in the Android manifest

1. Define layout in XML



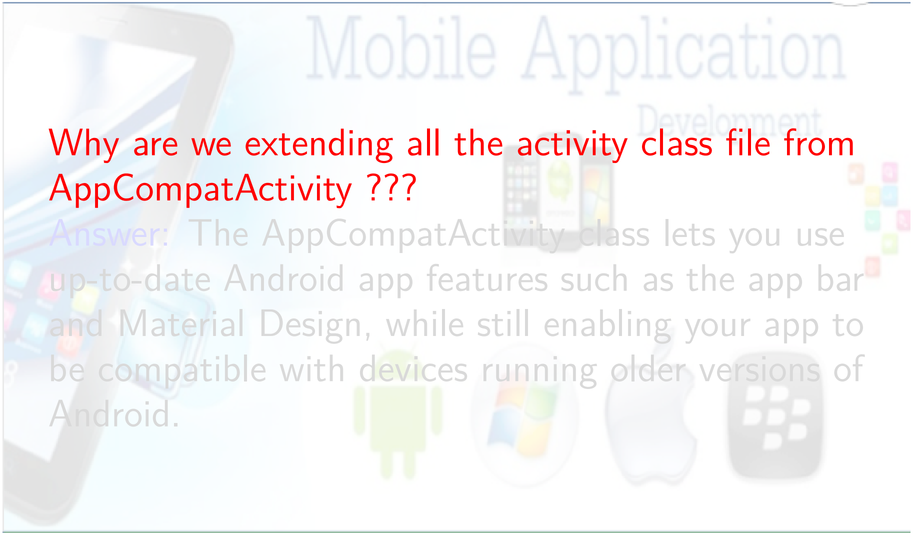
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:text="Let's Shop for Food!" />
</RelativeLayout>
```

2. Define Activity Java class



```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
   }
}
```

2. Define Activity Java class



Why are we extending all the activity class file from AppCompatActivity ???

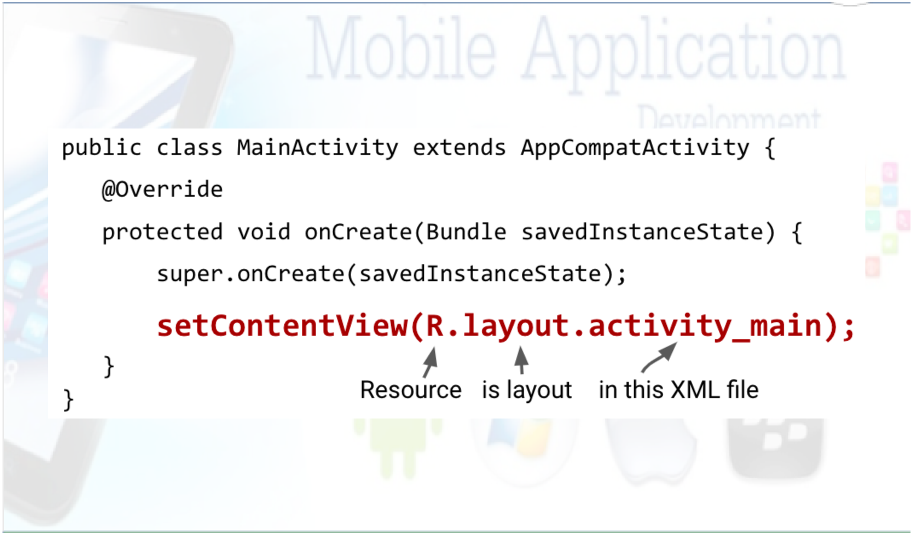
Answer: The AppCompatActivity class lets you use up-to-date Android app features such as the app bar and Material Design, while still enabling your app to be compatible with devices running older versions of Android.

2. Define Activity Java class

Why are we extending all the activity class file from AppCompatActivity ???

Answer: The AppCompatActivity class lets you use up-to-date Android app features such as the app bar and Material Design, while still enabling your app to be compatible with devices running older versions of Android.

3. Connect activity with layout

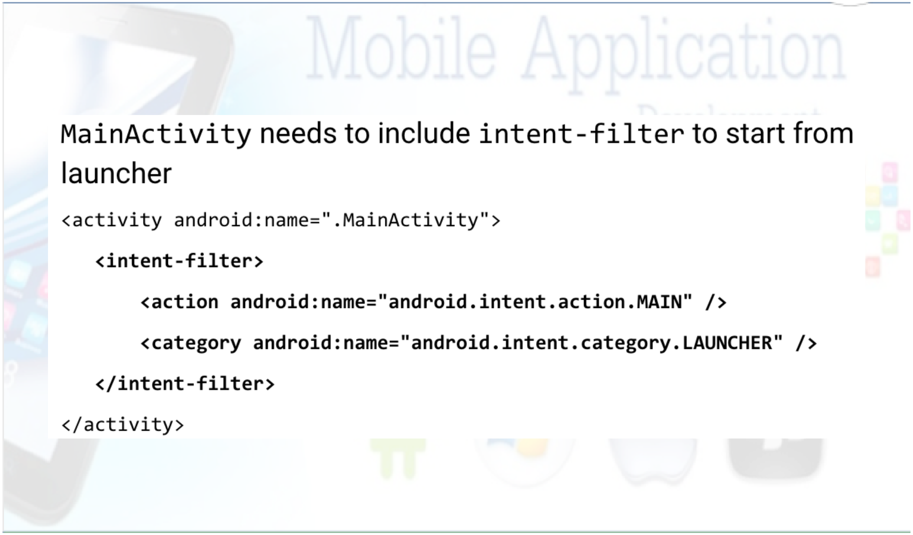


```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

4. Declare activity in Android manifest

```
<activity android:name=".MainActivity">
```

4. Declare activity in Android manifest


Mobile Application

MainActivity needs to include intent-filter to start from launcher

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

4. Declare activity in Android manifest

- Intent filters must include at least one <action> element, and can also include a <category> and optional <data>
- The <action> element specifies that this is the "main" entry point to the app.
- The <category> element specifies that this Activity should be listed in the system's app launcher (to allow users to launch this Activity).

4. Declare activity in Android manifest

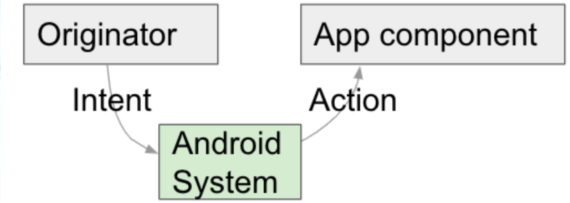
Each Activity in your app can also declare Intent filters, but only your MainActivity should include the "main" action.

Add another Activity to your project

- You can add a new Activity to your project by choosing File -> New -> Activity
- Choose the Activity template you want to use, or open the Gallery to see all the available templates.
- When you choose an Activity template, you see the same set of screens for creating the new activity that you did when you created the project.

What is an intent?

- An Intent is a description of an operation to be performed.
- An Intent is an object used to request an action from another app component via the Android system.



What can intents do?

- Start an Activity

  A button click starts a new Activity for text entry
  Clicking Share opens an app that allows you to post a photo

- Start an Service

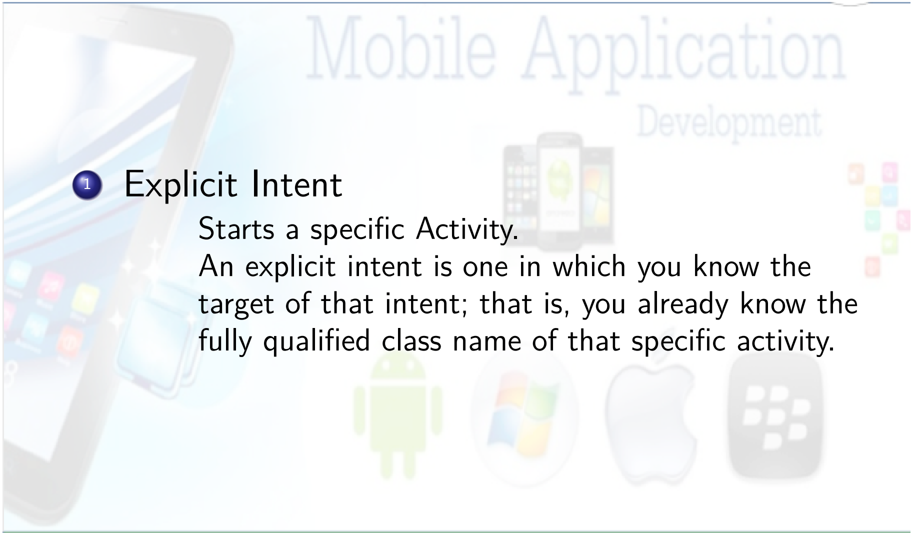
  Initiate downloading a file in the background

- Deliver Broadcast

  The system informs everybody that the phone is now charging

Intent Types

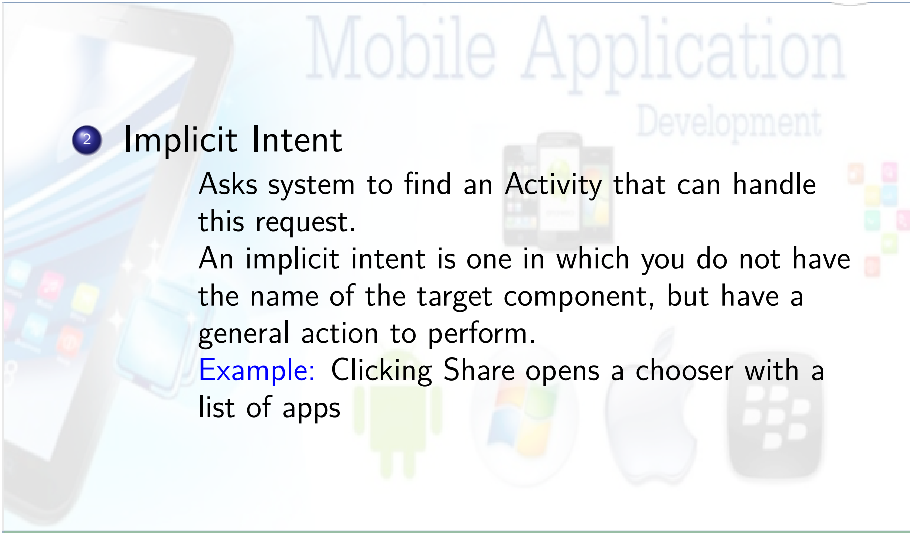
1. Explicit intents



1. Explicit Intent

   Starts a specific Activity.
   An explicit intent is one in which you know the target of that intent; that is, you already know the fully qualified class name of that specific activity.

2. Implicit intents



2. Implicit Intent

   Asks system to find an Activity that can handle this request.
   An implicit intent is one in which you do not have the name of the target component, but have a general action to perform.
   Example: Clicking Share opens a chooser with a list of apps

Start an Activity with an explicit intent

To start a specific Activity, use an explicit Intent.

1. Create an Intent

   Intent intent = new Intent(this, ActivityName.class);

2. Use the Intent to start the Activity

   startActivity(intent);

Start an Activity with an explicit intent

The intent constructor takes two arguments for an explicit Intent:

1. An application context. In this example, the Activity class provides the context (this).
2. The specific component to start (ShowMessageActivity.class).

Behind the scene of explicit intent

The startActivity() method sends the Intent to the Android system, which launches the ShowMessageActivity class on behalf of your app. The new Activity appears on the screen, and the originating Activity is paused.

Implicit Intents - Examples

Show a web page

```
Uri uri = Uri.parse("http://www.google.com");
Intent it = new
Intent(Intent.ACTION_VIEW,uri);
startActivity(it);
```

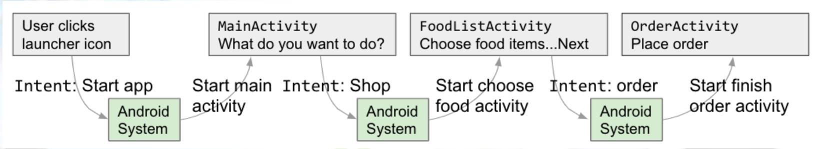
Implicit Intents - Examples

Dial a phone number

```
Uri uri = Uri.parse("tel:8005551234");
Intent it = new Intent(Intent.ACTION_DIAL,
uri);
startActivity(it);
```

How Activities Run

- All Activity instances are managed by the Android runtime
- Started by an "Intent", a message to the Android runtime to run an activity


User clicks launcher icon
Intent: Start app
Android System
Start main activity
MainActivity What do you want to do?
Intent: Shop
Android System
Start choose food activity
FoodListActivity Choose food items...Next
Intent: order
Android System
Start finish order activity
OrderActivity Place order

Sending and Receiving Data

Sending and retrieving data

In the first (sending) Activity:

1. Create the Intent object
2. Put data or extras into that Intent
3. Start the new Activity with startActivity()

Sending and retrieving data

In the second (receiving) Activity:

1. Get the Intent object, the Activity was started with
2. Retrieve the data or extras from the Intent object

Two types of sending data with intents

1. Data—one piece of information whose data location can be represented by an URI
2. Extras—one or more pieces of information as a collection of key-value pairs in a Bundle

When to use Intent data or Intent extras

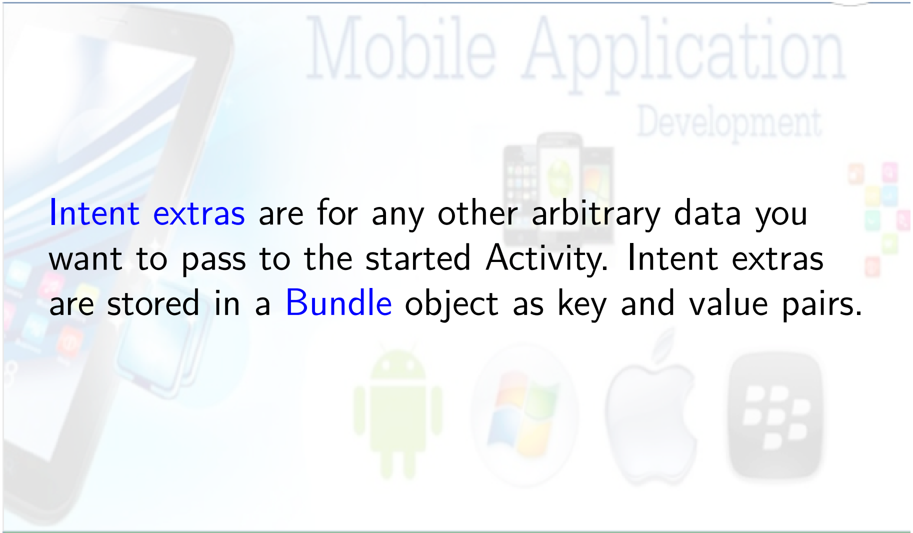
The Intent data can hold only one piece of information: a URI representing the location of the data you want to operate on. That URI could be a web page URL (http://), a telephone number (tel://), a geographic location (geo://) or any other custom URI you define.

When to use Intent data or Intent extras

Use the Intent data field:

- When you only have one piece of information you need to send to the started Activity.
- When that information is a data location that can be represented by a URI.

When to use Intent data or Intent extras

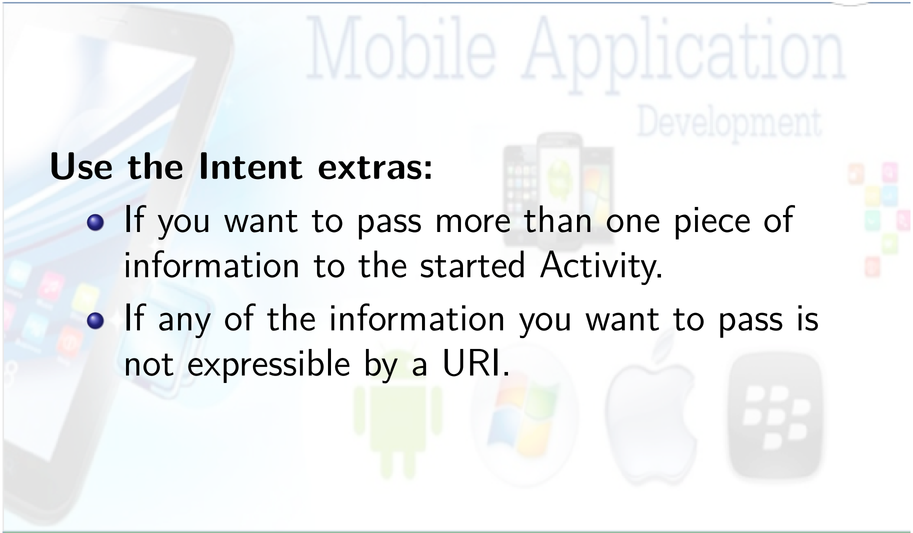


Intent extras are for any other arbitrary data you want to pass to the started Activity. Intent extras are stored in a Bundle object as key and value pairs.

When to use Intent data or Intent extras

A Bundle is a map, optimized for Android, in which a key is a string, and a value can be any primitive or object type.

When to use Intent data or Intent extras



Use the Intent extras:

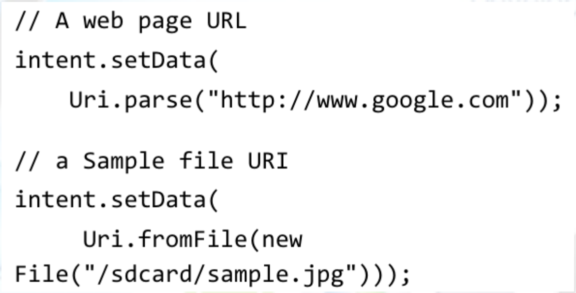
- If you want to pass more than one piece of information to the started Activity.
- If any of the information you want to pass is not expressible by a URI.

Add data to the Intent

- Create the Intent object
  Intent messageIntent = new Intent(this, ShowMessageActivity.class);
- Use the setData() method with a Uri object to add that URI to the Intent.

Add data to the Intent

Some examples of using setData() with URIs:



```
// A web page URL
intent.setData(
    Uri.parse("http://www.google.com"));

// a Sample file URI
intent.setData(
     Uri.fromFile(new
File("/sdcard/sample.jpg")));
```

Keep in mind that the data field can only contain a single URI;

Add data to the Intent

After you've added the data, you can start the Activity with the Intent as usual:

```
startActivity(messageIntent);
```

Add extras to the Intent

To add Intent extras to an explicit Intent from the originating Activity:

- Determine the keys to use for the information you want to put into the extras, or define your own. Each piece of information needs its own unique key.
- Use the putExtra() methods to add your key/value pairs to the Intent extras. Optionally you can create a Bundle object, add your data to the Bundle, and then add the Bundle to the Intent.

Add extras to the Intent

The Intent class includes extra keys you can use, defined as constants that begin with the word EXTRA_

Example: You could use Intent.EXTRA_EMAIL to indicate an array of email addresses

Add extras to the Intent

You can also define your own Intent extra keys.

```
public final static String EXTRA_MESSAGE =
                                        "com.example.mysampleapp.MESSAGE";
public final static String EXTRA_POSITION_X = "com.example.mysampleapp.X";
public final static String EXTRA_POSITION_Y = "com.example.mysampleapp.Y";
```

Add extras to the Intent

Create an Intent object (if one does not already exist):

```
Intent messageIntent = new Intent(this,
ShowMessageActivity.class);
```

Add extras to the Intent

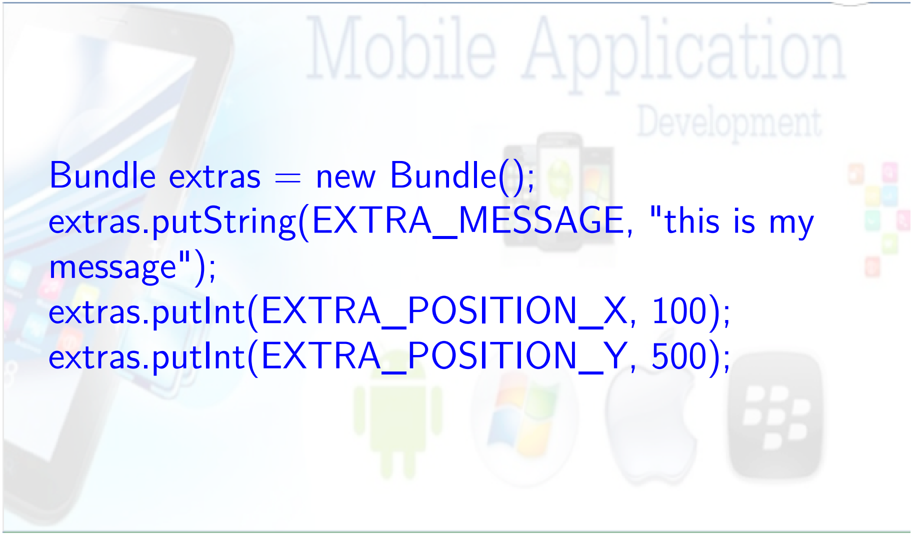
Use a putExtra() method with a key to put data into the Intent extras

```
messageIntent.putExtra(EXTRA_MESSAGE, "this is my message");
messageIntent.putExtra(EXTRA_POSITION_X, 100);
messageIntent.putExtra(EXTRA_POSITION_Y, 500);
```

Add extras to the Intent

Alternately, you can create a new Bundle and populate that Bundle with your Intent extras. Bundle defines many "put" methods for different kinds of primitive data as well as objects

Add extras to the Intent



```
Bundle extras = new Bundle();
extras.putString(EXTRA_MESSAGE, "this is my
message");
extras.putInt(EXTRA_POSITION_X, 100);
extras.putInt(EXTRA_POSITION_Y, 500);
```

Add extras to the Intent

After you've populated the Bundle, add it to the Intent with the putExtras() method (note the "s" in Extras):

```
messageIntent.putExtras(extras);
```

Start the Activity with the Intent as usual:

```
startActivity(messageIntent);
```

Retrieve the data from the Intent in the started Activity

To retrieve the Intent the Activity (or other component) was started with, use the getIntent() method:

```
Intent intent = getIntent();
```

Retrieve the data from the Intent in the started Activity

Use getData() to get the URI from that Intent:

```
Uri locationUri = intent.getData();
```

Retrieve the data from the Intent in the started Activity

To get the extras out of the Intent:
You can use the keys you defined in the originating Activity (if they were defined as public.)

Retrieve the data from the Intent in the started Activity

Use one of the getExtra() methods to extract extra data out of the Intent object:

String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
int positionX = intent.getIntExtra(MainActivity.EXTRA_POSITION_X);
int positionY = intent.getIntExtra(MainActivity.EXTRA_POSITION_Y);)

Retrieve the data from the Intent in the started Activity

Or you can get the entire extras Bundle from the Intent and extract the values with the various Bundle methods:

```
Bundle extras = intent.getExtras();
String message =
extras.getString(MainActivity.EXTRA_MESSAGE);
```

Getting data back from an Activity

To launch a new Activity and get a result back, do the following steps in your originating Activity:

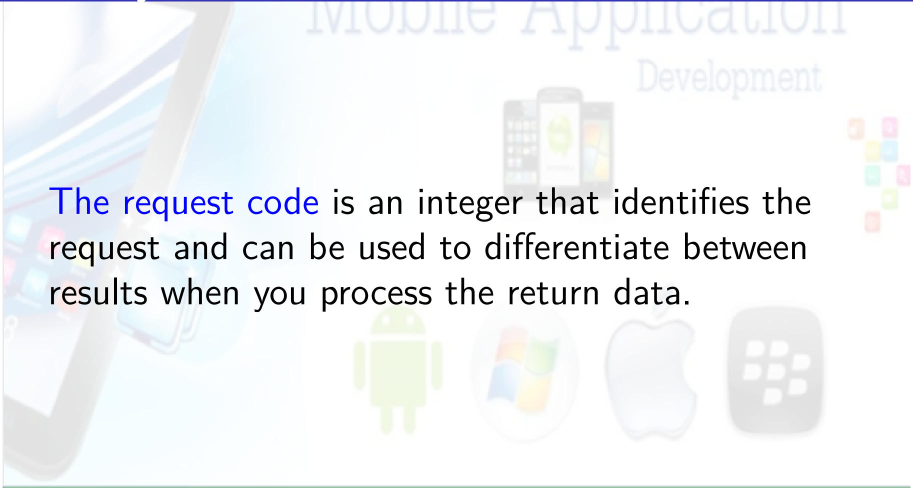
1. Instead of launching the Activity with startActivity(), call startActivityForResult() with the Intent and a request code.
2. Create a new Intent in the launched Activity and add the return data to that Intent.
3. Implement onActivityResult() in the originating Activity to process the returned data.

Use startActivityForResult() to launch the Activity

```
startActivityForResult(messageIntent,
TEXT_REQUEST);
```

StartActivityForResult takes an Intent argument that contains information about the Activity to be launched and any data to send to that Activity. The startActivityForResult() method, however, also needs a request code.

Use startActivityForResult() to launch the Activity



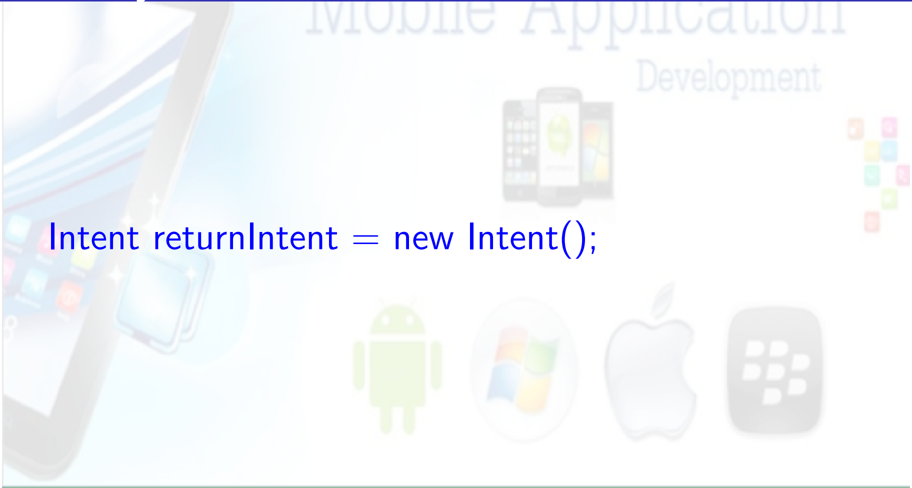
The request code is an integer that identifies the request and can be used to differentiate between results when you process the return data.

Use startActivityForResult() to launch the Activity

Conventionally you define request codes as static integer variables with names that include REQUEST. Use a different integer for each code. For example:

```
public static final int PHOTO_REQUEST = 1;
public static final int PHOTO_PICK_REQUEST =
2;
public static final int TEXT_REQUEST = 3;
```

Return a response from the launched Activity



```
Intent returnIntent = new Intent();
```

Return a response from the launched Activity

Define keys for the return Intent extras at the start of your class

```
public final static String
EXTRA_RETURN_MESSAGE =
"com.example.mysampleapp.RETURN_MESSAGE";
```

Return a response from the launched Activity

Then put your return data into the Intent as usual.

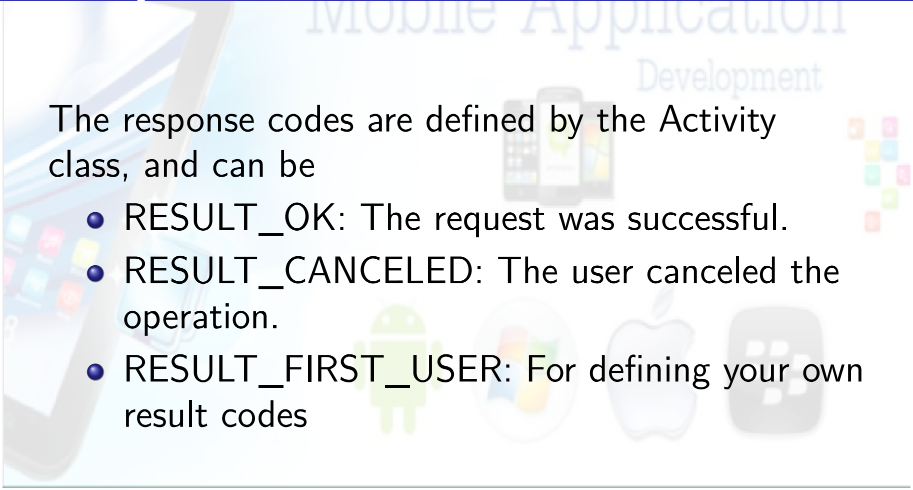
```
messageIntent.putExtra(EXTRA_RETURN_MESSAGE,
mMessage);
```

Return a response from the launched Activity

Use the setResult() method with a response code and the Intent with the response data:

```
setResult(RESULT_OK,replyIntent);
```

Return a response from the launched Activity



The response codes are defined by the Activity class, and can be

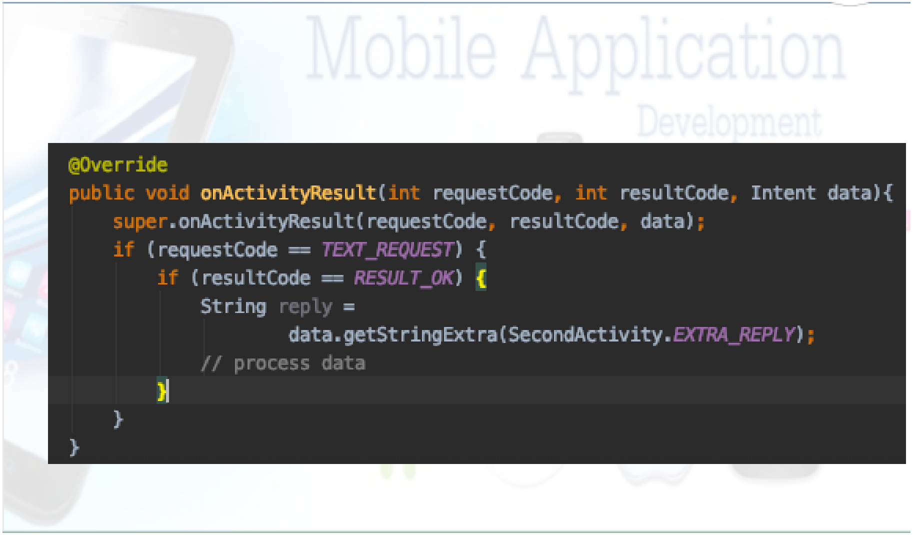
- RESULT_OK: The request was successful.
- RESULT_CANCELED: The user canceled the operation.
- RESULT_FIRST_USER: For defining your own result codes

Return a response from the launched Activity

Finally, call finish() to close the Activity and resume the originating Activity:

```
finish();
```

Read response data in onActivityResult()



```
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data){
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == TEXT_REQUEST) {
        if (resultCode == RESULT_OK) {
            String reply =
                    data.getStringExtra(SecondActivity.EXTRA_REPLY);
            // process data
        }
    }
}
```

Read response data in onActivityResult()

The three arguments to onActivityResult() contain all the information you need to handle the return data.

- Request code: The request code you set when you launched the Activity with startActivityForResult().

Read response data in onActivityResult()

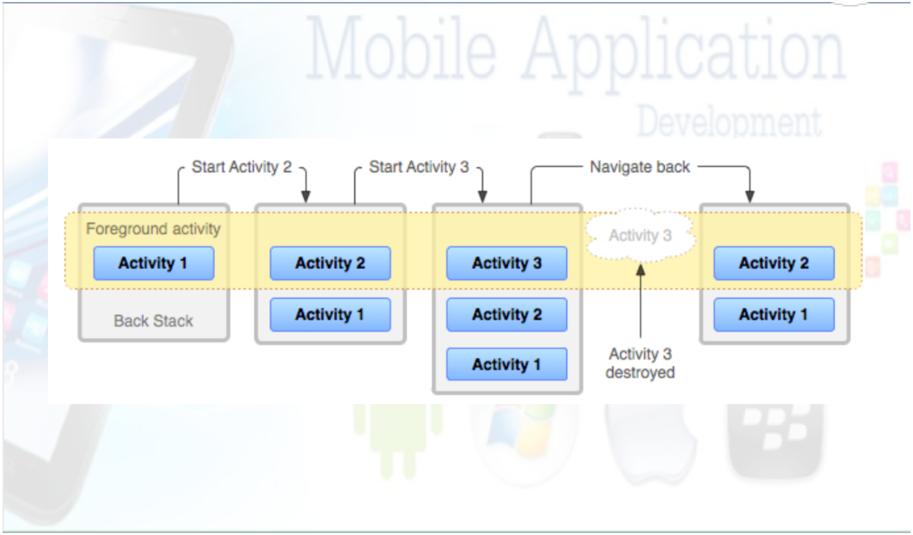
- Result code: the result code set in the launched Activity, usually one of RESULT_OK or RESULT_CANCELED.
- Intent data: the Intent that contains the data returned from the launch Activity.

Navigation

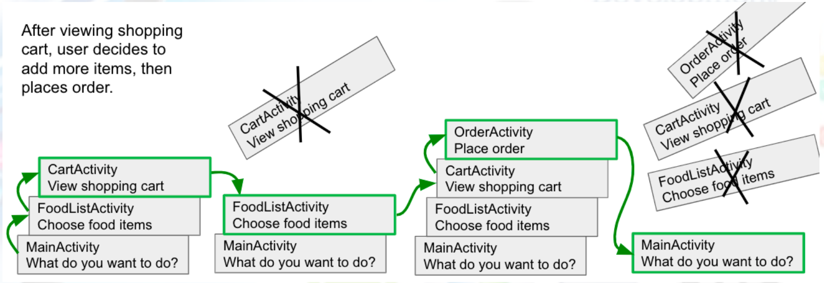
Activity stack

- When a new Activity is started, the previous Activity is stopped and pushed on the Activity back stack
- Last-in-first-out-stack—when the current Activity ends, or the user presses the Back button, it is popped from the stack and the previous Activity resumes

Activity stack

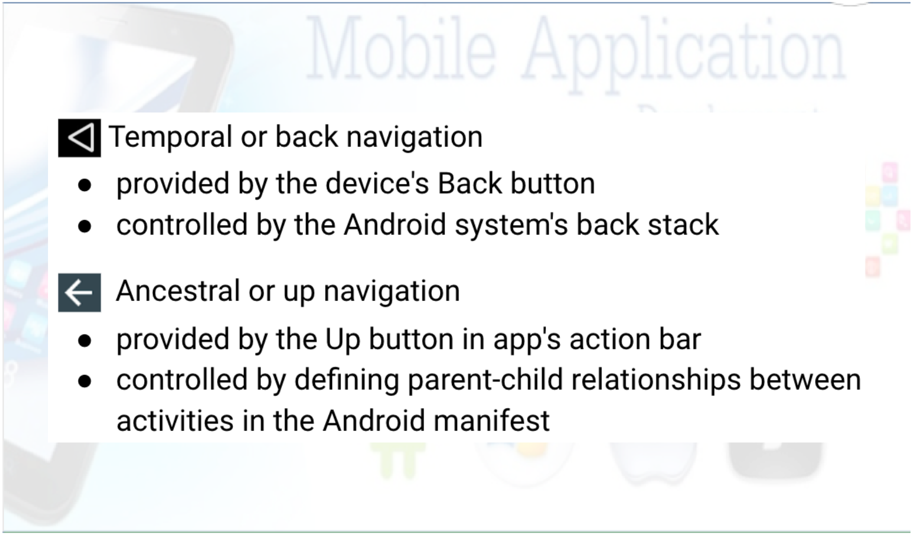


Activity stack


After viewing shopping cart, user decides to add more items, then places order.

CartActivity View shopping cart
FoodListActivity Choose food items
MainActivity What do you want to do?
CartActivity View shopping cart
FoodListActivity Choose food items
MainActivity What do you want to do?
OrderActivity Place order
CartActivity View shopping cart
FoodListActivity Choose food items
MainActivity What do you want to do?
OrderActivity Place order
CartActivity View shopping cart
FoodListActivity Choose food items
MainActivity What do you want to do?

Two forms of navigation


Mobile Application

◁ Temporal or back navigation

- provided by the device's Back button
- controlled by the Android system's back stack

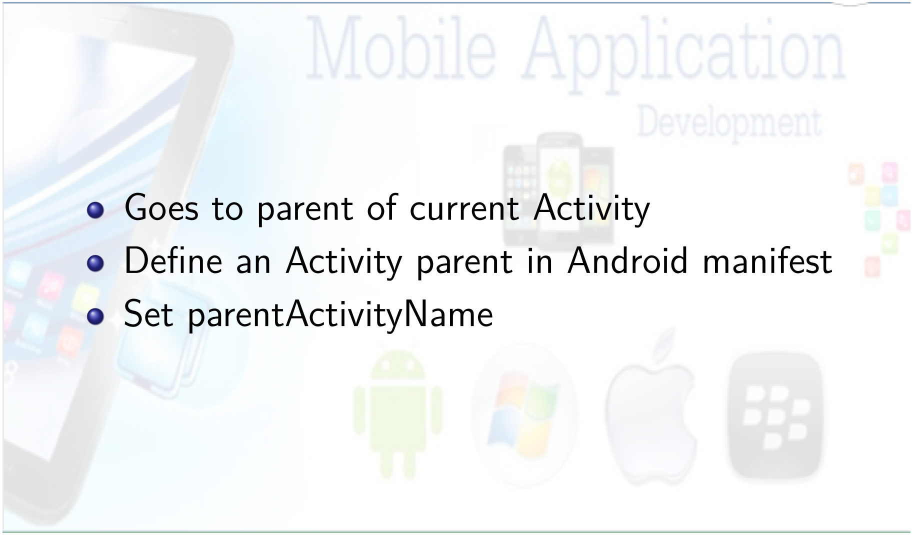
← Ancestral or up navigation

- provided by the Up button in app's action bar
- controlled by defining parent-child relationships between activities in the Android manifest

Back navigation

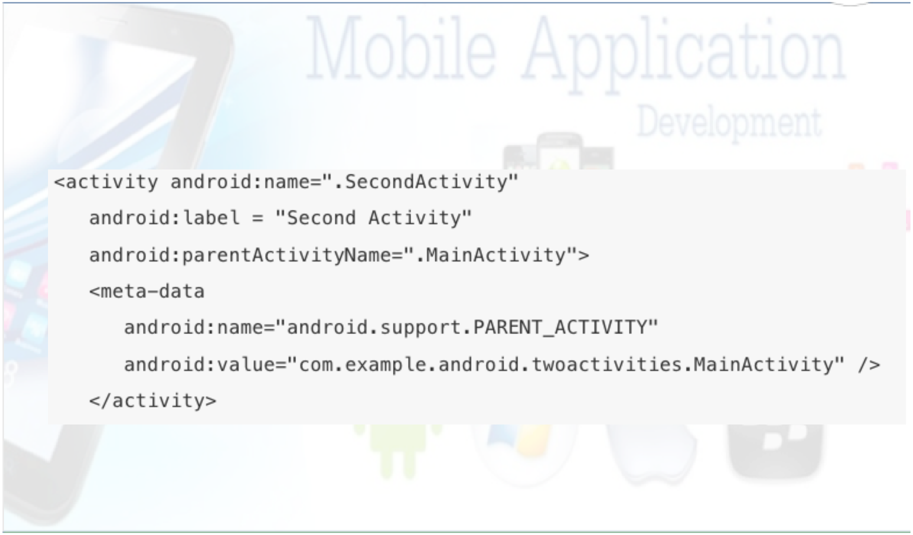
- Back stack preserves history of recently viewed screens
- Back stack contains all the Activity instances that have been launched by the user in reverse order for the current task

Up navigation



- Goes to parent of current Activity
- Define an Activity parent in Android manifest
- Set parentActivityName

Up navigation



```
<activity android:name=".SecondActivity"
   android:label = "Second Activity"
   android:parentActivityName=".MainActivity">
   <meta-data
      android:name="android.support.PARENT_ACTIVITY"
      android:value="com.example.android.twoactivities.MainActivity" />
   </activity>
```

Up navigation

Note

Beginning in Android 4.1 (API level 16), declare the logical parent of each Activity by specifying the android:parentActivityName attribute in the <activity> element. To support older versions of Android, include <meta-data> information to define the parent Activity explicitly. Use both methods to be backwards-compatible with all versions of Android.

THANK YOU