IT332: Mobile Application Development

Lecture # 04: Activities & Intents

Muhammad Imran

Outline

- Declare Permissions for Activities
- The Life Cycle of an Activity
- Intents and Intent Types
- Linking Activities Using Intents
- Returning Results from an Intent
- Passing Data using Intent

Creating Android Activity

- To create an activity, you create a Java class that extends the Activity base class

```
public class MainActivity extends AppCompatActivity {

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

- Your activity class loads its user interface (UI) component using the XML file defined in your res/layout folder.
- Every activity of application must be declared in AndroidManifest.xml file

Declare Permissions for Activities

- You can use the manifest's <activity> tag to control which apps can start a particular activity.
- A parent activity cannot launch a child activity unless both activities have the same permissions in their manifest.
- If you declare a <uses-permission> element for a parent activity, each child activity must have a matching <uses-permission> element.

Declare Permissions for Activities

- For example, if your app wants to use a hypothetical app named SocialApp to share a post on social media, SocialApp itself must define the permission that an app calling it must have:

```
<manifest>
<activity android:name="...."
   android:permission="com.google.socialapp.permission.SHARE_POST"

/>
```

- Then, to be allowed to call SocialApp, your app must match the permission by declaring them like:

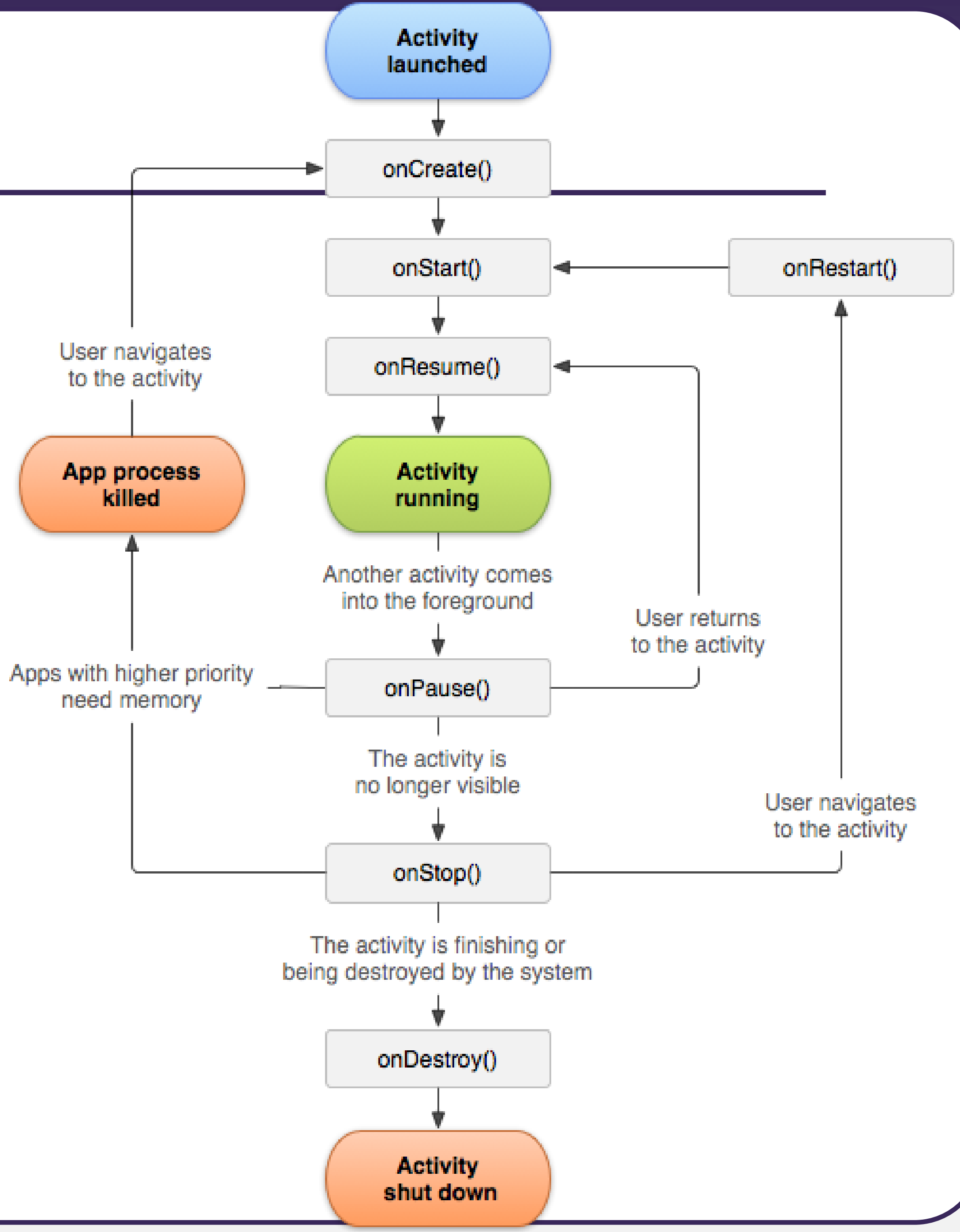
```
<manifest>
   <uses-permission android:name="com.google.socialapp.permission.SHARE_POST" />
</manifest>
```

Android Activities

- An Android application can have zero or more activities and typically, applications have one or more activities.
- The main purpose of an activity is to interact with the user.
- From the moment an activity appears on the screen to the moment it is hidden, it goes through several stages known as an activity's life cycle.
- The Activity class provides several callbacks that allow the activity to know that a state has changed: that the system is creating, stopping, or resuming an activity, or destroying the process in which the activity resides.

Activity Life Cycle

- The Activity base class defines a series of events that govern the life cycle of an activity



Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity Life Cycle

- The Activity class defines the following events:
  - onCreate()—Called when the activity is first created
  - onStart()—Called when the activity becomes visible to the user
  - onResume()—Called when the activity starts interacting with the user
  - onPause()—Called when the current activity is being paused and the previous activity is being resumed
  - onStop()—Called when the activity is no longer visible to the user
  - onDestroy()—Called before the activity is destroyed by the system (either manually or by the system to conserve memory)
  - onRestart()—Called when the activity has been stopped and is restarting again

Observe Activity Life Cycle

- Using Android Studio, create a new Android project and name it Activity101
- In the Activity101Activity.java file, add the following highlighted statements

```
import android.util.Log;
public class MainActivity extends AppCompatActivity
{
     String tag = "Lifecycle Step";
     @Override
     protected void onCreate(Bundle savedInstanceState)
    {
         super.onCreate(savedInstanceState);
         setContentView(R.layout.activity_main);
         Log.d(tag, "In the onCreate() event");
    }
    public void onStart()
    {
         super.onStart();
         Log.d(tag, "In the onStart() event");
     }
    public void onRestart()
    {
         super.onRestart();
         Log.d(tag, "In the onRestart() event");
     }
```

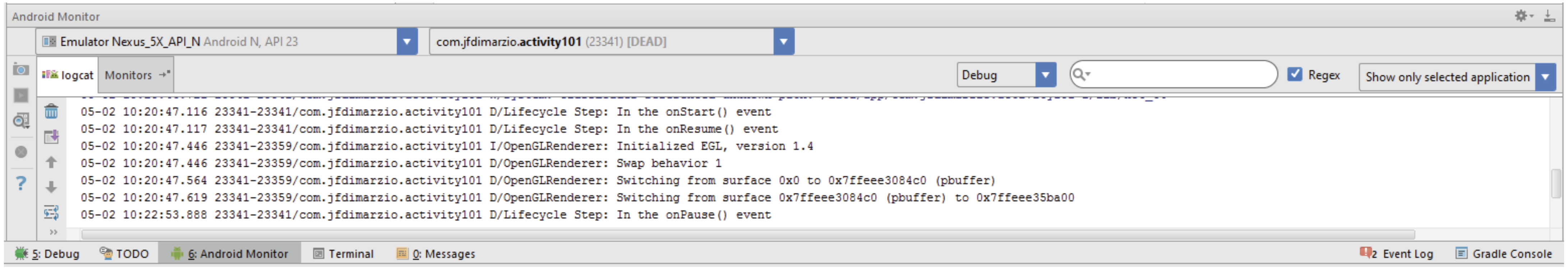
Observe Activity Life Cycle

- Run => Debug, or Shift + F9 to debug the app

```
public void onResume()
   {
        super.onResume();
        Log.d(tag, "In the onResume() event");
   }

   public void onPause()
   {
        super.onPause();
        Log.d(tag, "In the onPause() event");
    }

   public void onStop()
   {
        super.onStop();
        Log.d(tag, "In the onStop() event");
    }

   public void onDestroy()
   {
         super.onDestroy();
         Log.d(tag, "In the onDestroy() event");
   }
}
```

Observe Activity Life Cycle

- When the activity is first loaded, you should see something very similar to the following in the logcat console


Android Monitor

Emulator Nexus_5X_API_N Android N, API 23 | com.jfdimarzio.activity101 (23341) [DEAD]

logcat | Monitors | Debug | Regex ✓ | Show only selected application

```
05-02 10:20:47.116 23341-23341/com.jfdimarzio.activity101 D/Lifecycle Step: In the onStart() event
05-02 10:20:47.117 23341-23341/com.jfdimarzio.activity101 D/Lifecycle Step: In the onResume() event
05-02 10:20:47.446 23341-23359/com.jfdimarzio.activity101 I/OpenGLRenderer: Initialized EGL, version 1.4
05-02 10:20:47.446 23341-23359/com.jfdimarzio.activity101 D/OpenGLRenderer: Swap behavior 1
05-02 10:20:47.564 23341-23359/com.jfdimarzio.activity101 D/OpenGLRenderer: Switching from surface 0x0 to 0x7ffeee3084c0 (pbuffer)
05-02 10:20:47.619 23341-23359/com.jfdimarzio.activity101 D/OpenGLRenderer: Switching from surface 0x7ffeee3084c0 (pbuffer) to 0x7ffeee35ba00
05-02 10:22:53.888 23341-23341/com.jfdimarzio.activity101 D/Lifecycle Step: In the onPause() event
```

- If you click the Back button on the Android emulator, you will see:

```
11-16 06:29:26.665: D/Lifecycle Step(559): In the onPause() event
11-16 06:29:28.465: D/Lifecycle Step(559): In the onStop() event
11-16 06:29:28.465: D/Lifecycle Step(559): In the onDestroy() event
```

Observe Activity Life Cycle

- Click the Home button, click the Overview icon, select the Activity101 app, you will see:

```
11-16 06:31:08.905: D/Lifecycle Step(559): In the onCreate() event
11-16 06:31:08.905: D/Lifecycle Step(559): In the onStart() event
11-16 06:31:08.925: D/Lifecycle Step(559): In the onResume() event
```

- Click the Home button and then click the Phone button on the Android emulator so that the activity is pushed to the background

```
11-16 06:32:00.585: D/Lifecycle Step(559): In the onPause() event
11-16 06:32:05.015: D/Lifecycle Step(559): In the onStop() event
```

- Exit the phone dialer by clicking the Back button, the activity is now visible again:

```
11-16 06:32:50.515: D/Lifecycle(559): In the onRestart() event
11-16 06:32:50.515: D/Lifecycle(559): In the onStart() event
11-16 06:32:50.515: D/Lifecycle(559): In the onResume() event
```

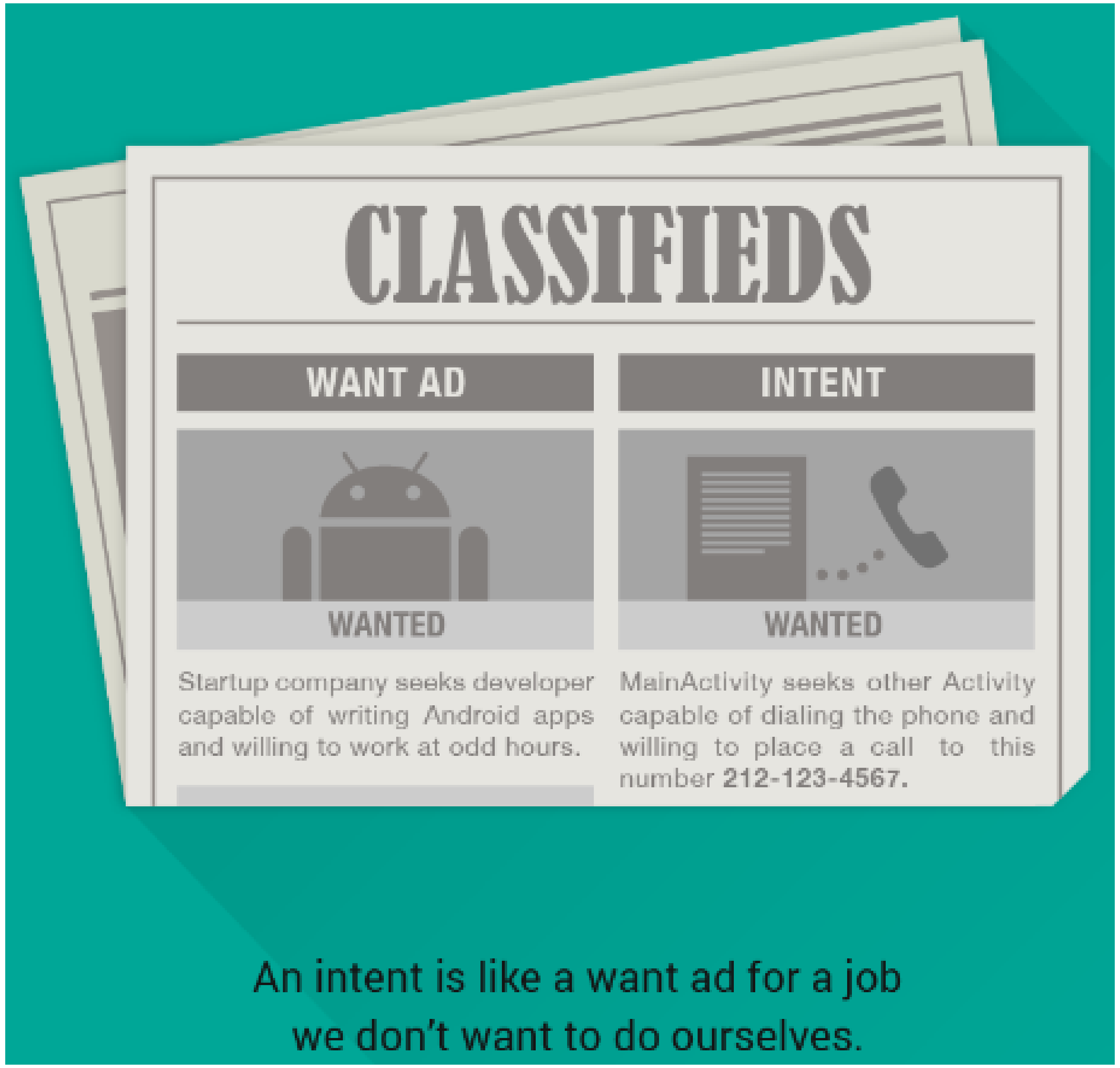
How Activity Lifecycle Works

- An activity is destroyed when you click the Back button.
- This is crucial to understand because state of the activity will be lost after destruction.
- This means you need to write additional code in your activity to preserve its state when the activity is destroyed
- The onPause() method is called in both scenarios
  - When an activity is sent to the background
  - When a user kills an activity by tapping the Back button
- When activity is started, the onStart() and onResume() methods are always called, regardless of whether the activity is restored from the background or newly created.
- When an activity is created for the first time, the onCreate() method is called.

Observe Activity Life Cycle : Summary

- Use the onCreate() method to create and instantiate the objects that you will be using in your application
- Use the onResume() method to start any services or code that needs to run while your activity is in the foreground
- Use the onPause() method to stop any services or code that does not need to run when your activity is not in the foreground
- Use the onDestroy() method to free up resources before your activity is destroyed

Intents



Startup company seeks developer capable of writing Android apps and willing to work at odd hours.

MainActivity seeks other Activity capable of dialing the phone and willing to place a call to this number **212-123-4567.**

An intent is like a want ad for a job we don't want to do ourselves.

Intents (From Android Vocabulary Glossary)

Read This Out

Each app contains at least one object of class Activity. An object of this class has methods that tell the device to display a user interface on the screen: images, buttons, and pieces of text.

In addition to displaying the interface, each class of Activity object does one useful job. It could dial the phone, send email, or play a video.

Intents (From Android Vocabulary Glossary)

Read This Out (contd…)

An Activity object in one app can ask for help from another Activity object in the same app, or even in another app.

The first Activity might describe what it needs by saying "I am looking for an Activity that is capable of displaying a Google map" or "I am looking for an Activity that can scan a barcode".

It loads this description into an object of type Intent, and then sends the Intent out to search the Android device.

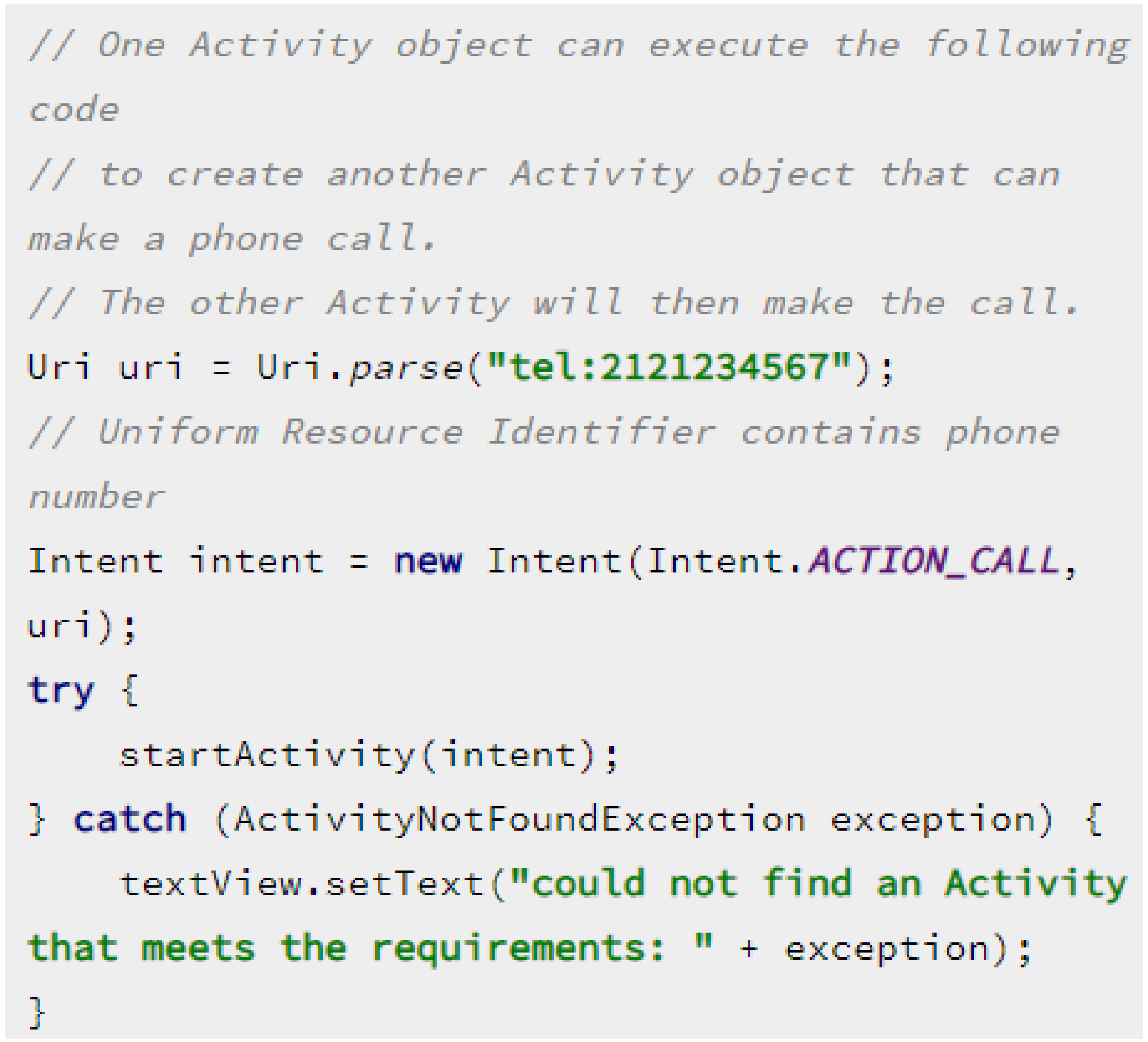
Intents (From Android Vocabulary Glossary)

Read This Out **(contd…)**

Each app on the device contains a manifest file that lists the names of the Activity classes in that app. The manifest also contains filters which describe the capabilities of each Activity.

The Intent object examines these manifests. If it finds an Activity class that meets the requirements, it will create an object of that class and execute its onCreate method. If not, it will return the disappointing news to the original Activity.

Intents (From Android Vocabulary Glossary)



```
// One Activity object can execute the following
code
// to create another Activity object that can
make a phone call.
// The other Activity will then make the call.
Uri uri = Uri.parse("tel:2121234567");
// Uniform Resource Identifier contains phone
number
Intent intent = new Intent(Intent.ACTION_CALL,
uri);
try {
    startActivity(intent);
} catch (ActivityNotFoundException exception) {
    textView.setText("could not find an Activity
that meets the requirements: " + exception);
}
```

Intents – For Navigating Between Activities

- When your application has more than one activity, you often need to navigate from one to another.
- In Android, you navigate between activities through what is known as an intent
- An Intent is a messaging object you can use to request an action from another app component.
- Although intents facilitate communication between components in several ways, there are three fundamental use cases:
  - Starting an activity
  - Starting a service
  - Delivering a broadcast

Starting an Activity with Intent

- An Activity represents a single screen in an app.
- You can start a new instance of an Activity by passing an Intent to startActivity().
- The Intent describes the activity to start and carries any necessary data.
- If you want to receive a result from the activity when it finishes, call startActivityForResult()
- Your activity receives the result as a separate Intent object in your activity's onActivityResult() callback

- There are two types of intents:
  - Explicit intents
  - Implicit intents

Intents Filters

- An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive.
- For instance, by declaring an intent filter for an activity, you make it possible for other apps to directly start your activity with a certain kind of intent. (implicit intent)
- Likewise, if you do not declare any intent filters for an activity, then it can be started only with an explicit intent.
- For example, an explicit request might tell the system to “Start the Send Email activity in the Gmail app". By contrast, an implicit request tells the system to “Start a Send Email screen in any activity that can do the job."

Intents Filters

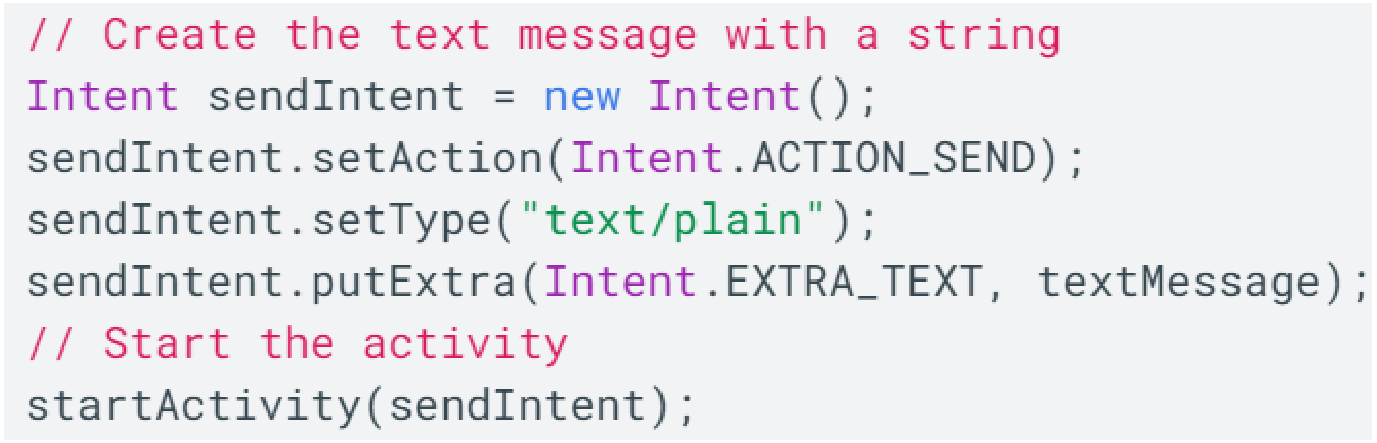
- The definition of <intent-filter> includes action, category and data
- For example, the following code snippet shows how to configure an activity that sends text data, and receives requests from other activities to do so:

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

- The <action> element specifies that this activity sends data.
- Declaring the <category> element as DEFAULT enables the activity to receive launch requests.
- The <data> element specifies the type of data that this activity can send.

Intents Filters

- The following code snippet shows how to call the activity with and intent



```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.setType("text/plain");
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
// Start the activity
startActivity(sendIntent);
```

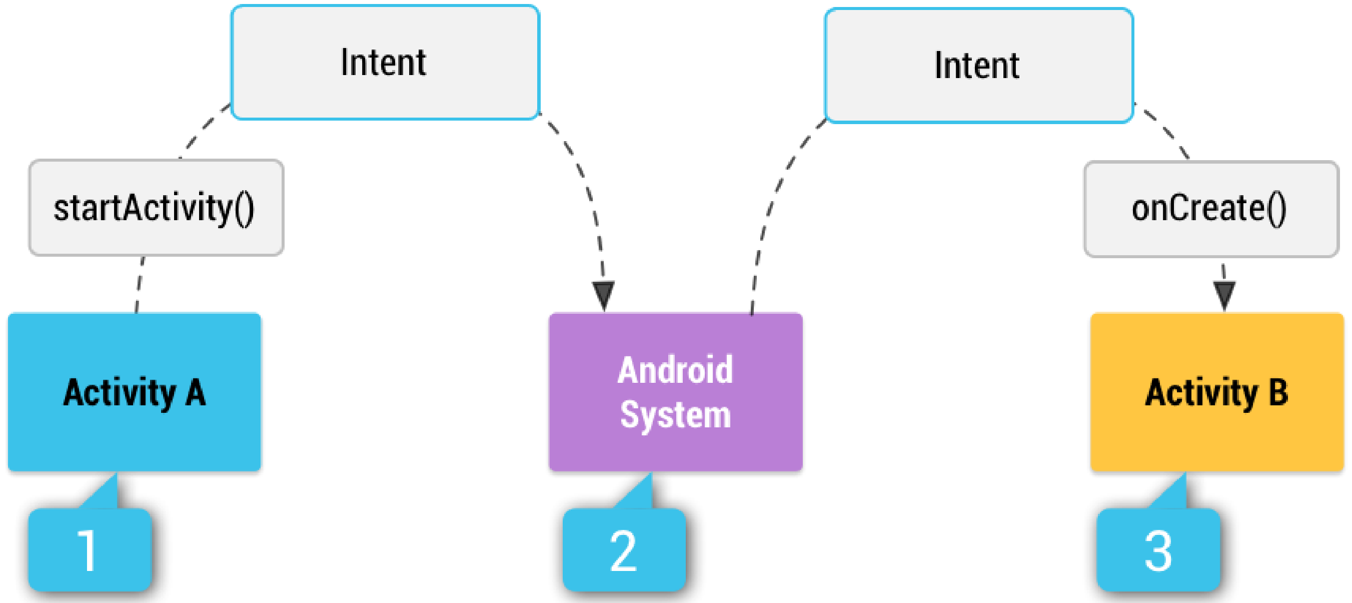
- If you intend for your app to be self-contained and not allow other apps to activate its activities, you don't need any intent filters.

- Activities that you don't want to make available to other applications should have no intent filters, and you can start them yourself using explicit intents.

Explicit Intents

- Explicit intents specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component 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 might start a new activity within your app in response to a user action, or start a service to download a file in the background.

Explicit 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.

Implicit Intents

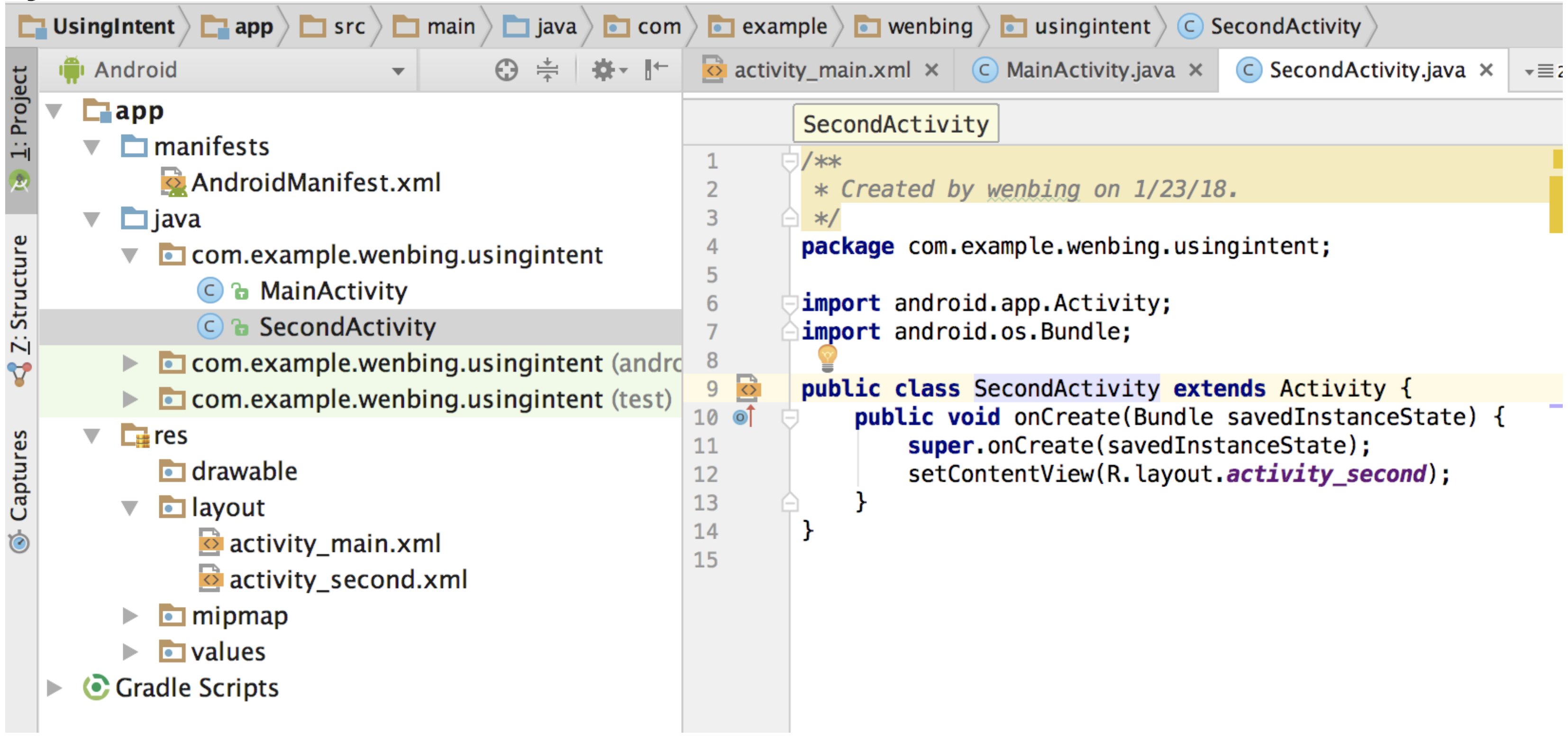
- Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.
- For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Implicit Intents

- When you use an implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the intent filters declared in the manifest file of other apps on the device.
- If the intent matches an intent filter, the system starts that component and delivers it the Intent object.
- If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

Linking Activities with Intents: Example

- Create a new Android project with an empty Activity named MainActivity and then add a SecondActivity



UsingIntent > app > src > main > java > com > example > wenbing > usingintent > SecondActivity

activity_main.xml × | MainActivity.java × | SecondActivity.java ×

```
app
  manifests
    AndroidManifest.xml
  java
    com.example.wenbing.usingintent
      MainActivity
      SecondActivity
    com.example.wenbing.usingintent (andrc
    com.example.wenbing.usingintent (test)
  res
    drawable
    layout
      activity_main.xml
      activity_second.xml
    mipmap
    values
Gradle Scripts
```

```java
/**
 * Created by wenbing on 1/23/18.
 */
package com.example.wenbing.usingintent;

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

public class SecondActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_second);
    }
}
```

- Now add required code for providing the Intent Filter in AndroidManifest.xml file

Linking Activities with Intents: Example

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.nomadlearner.usingintent">
    <application
         android:allowBackup="true”
         ...
         <activity android:name=".MainActivity">
             .....
         </activity>
         <activity android:name=".SecondActivity" >
             <intent-filter >
                 <action android:name="com.nomadlearner.usingintent.SecondActivity"/>
                 <category android:name="android.intent.category.DEFAULT" />
             </intent-filter>
          </activity>
      </application>
</manifest>
```

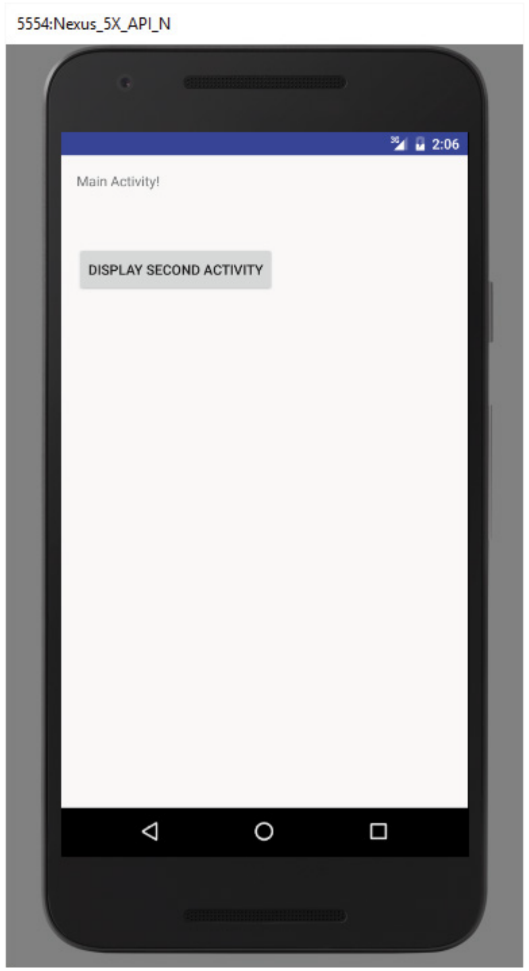
Linking Activities with Intents: Example

- Modify the MainActivity.java file as shown in the bolded lines in the following code:

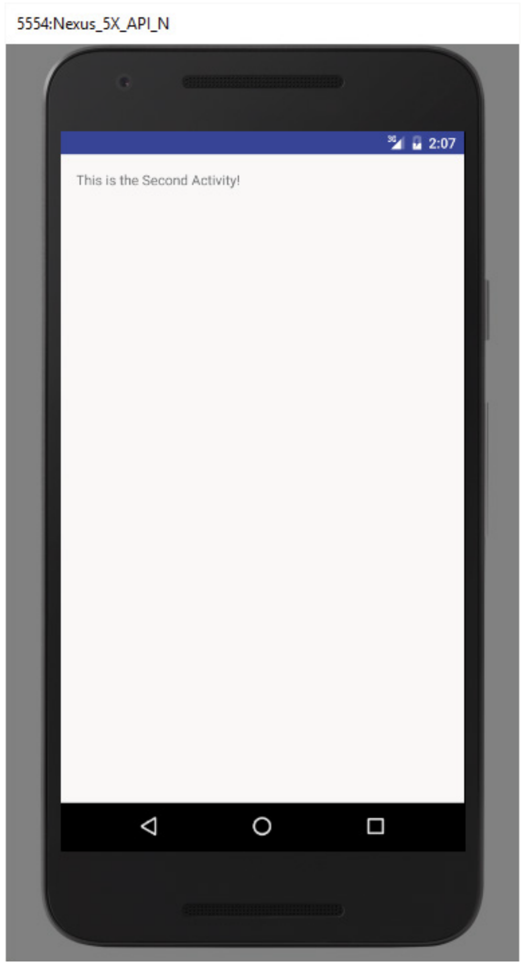
```
import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.view.View;

public class MainActivity extends Activity {
     @Override
     protected void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
           setContentView(R.layout.activity_main);
      }
      public void onClick(View view) {
            startActivity(new Intent("com.username.usingintent.SecondActivity"));
      }
}
```

Linking Activities with Intents: Example

- When the first activity is loaded, click the button and the second activity also loads



Main Activity!



This is the Second Activity!

Toast

```
public class Toast extends Object
java.lang.Object
          ↳android.widget.Toast
```

- A toast is a view containing a quick little message for the user
- When the view is shown to the user, appears as a floating view over the application
- The idea is to be as unobtrusive as possible, while still showing the user the information you want them to see
- Two examples are the volume control, and the brief message saying that your settings have been saved

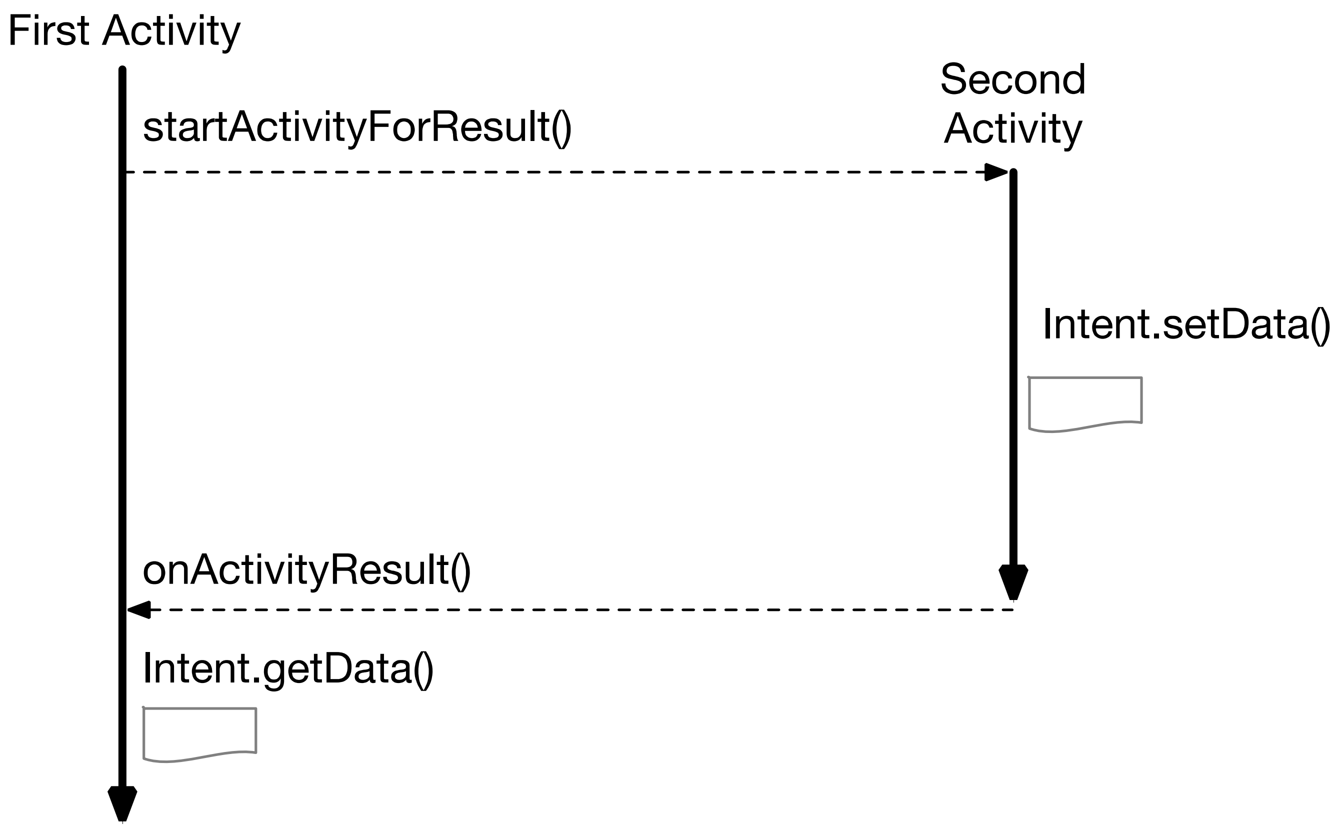
int	LENGTH_LONG Show the view or text notification for a long period of time.
int	LENGTH_SHORT Show the view or text notification for a short period of time.

Returning Results from an Activity

- Sometimes you want to get a result back from an activity when it ends.
- For example, you may start an activity that lets the user pick a person in a list of contacts; when it ends, it returns the person that was selected.
- The startActivity() method invokes another activity but does not return a result.

- If you need to pass data back from an activity, use startActivityForResult(Intent, int) method.
- The result comes back through your onActivityResult(int, int, Intent) method
- When a child activity exits, it can call setResult(int) to return data to its parent.

Returning Results from an Activity



Returning Results from an Activity

- To call an activity and wait for a result to be returned from it, you need to use the startActivityForResult() method, like this:

```
startActivityForResult(new Intent("com.jfdimarzio.usingintent.
  SecondActivity"),request_Code);
```

- In addition to passing an Intent object, you need to pass a request code as well.
- The request code is simply an integer value that identifies an activity you are calling.
- This is needed because when an activity returns a value, you must have a way to identify it.
- For example, you might be calling multiple activities at the same time, though some activities might not return immediately (for example, waiting for a reply from a server).
- When an activity returns, you need this request code to determine which activity is actually returned.

Returning Results from an Activity

- In order for an activity to return a value to the calling activity, you use an Intent object to send data back via the setData() method

```
Intent data = new Intent();
//---get the EditText view---
EditText txt_username =
    (EditText) findViewById(R.id.txt_username);
//---set the data to pass back---
data.setData(Uri.parse(
    txt_username.getText().toString()));
setResult(RESULT_OK, data);
//---closes the activity---
finish();
```

- The setResult() method sets a result code (either RESULT_OK or RESULT_CANCELLED or any custom values starting at RESULT_FIRST_USER) and the data (an Intent object) to be returned back to the calling activity.
  - RESULT_CANCELED if a child activity fails for any reason, such as crashing
- The finish() method closes the activity and returns control to the calling activity.

Returning Results from an Activity

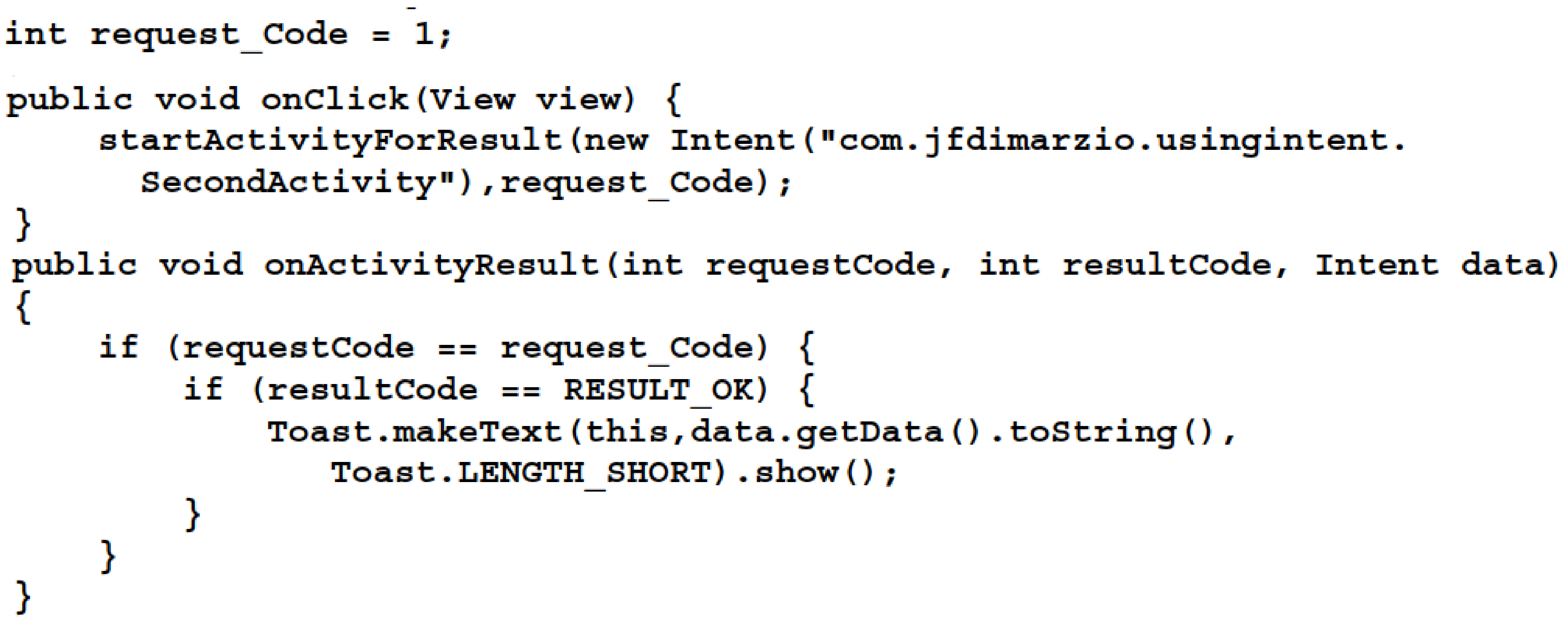
- In the calling activity, you need to implement the onActivityResult() method, which is called whenever an activity returns:

```
public void onActivityResult(int requestCode, int resultCode,
Intent data)
{
    if (requestCode == request_Code) {
        if (resultCode == RESULT_OK) {
            Toast.makeText(this,data.getData().toString(),
                Toast.LENGTH_SHORT).show();
        }
    }
}
```

- Here, you check for the appropriate request and result codes and display the result that is returned.
- The returned result is passed in via the data argument; and you obtain its details through the getData() method.

Returning Results from an Activity

MainActivity.java



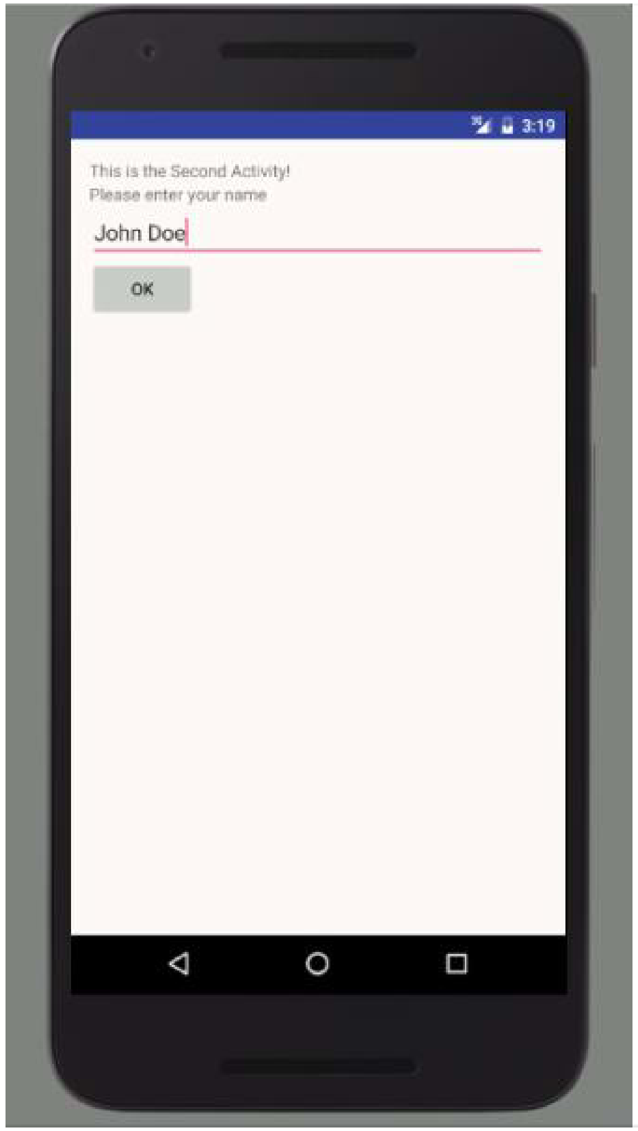
```
int request_Code = 1;

public void onClick(View view) {
    startActivityForResult(new Intent("com.jfdimarzio.usingintent.
      SecondActivity"),request_Code);
}
public void onActivityResult(int requestCode, int resultCode, Intent data)
{
    if (requestCode == request_Code) {
        if (resultCode == RESULT_OK) {
            Toast.makeText(this,data.getData().toString(),
               Toast.LENGTH_SHORT).show();
        }
    }
}
```

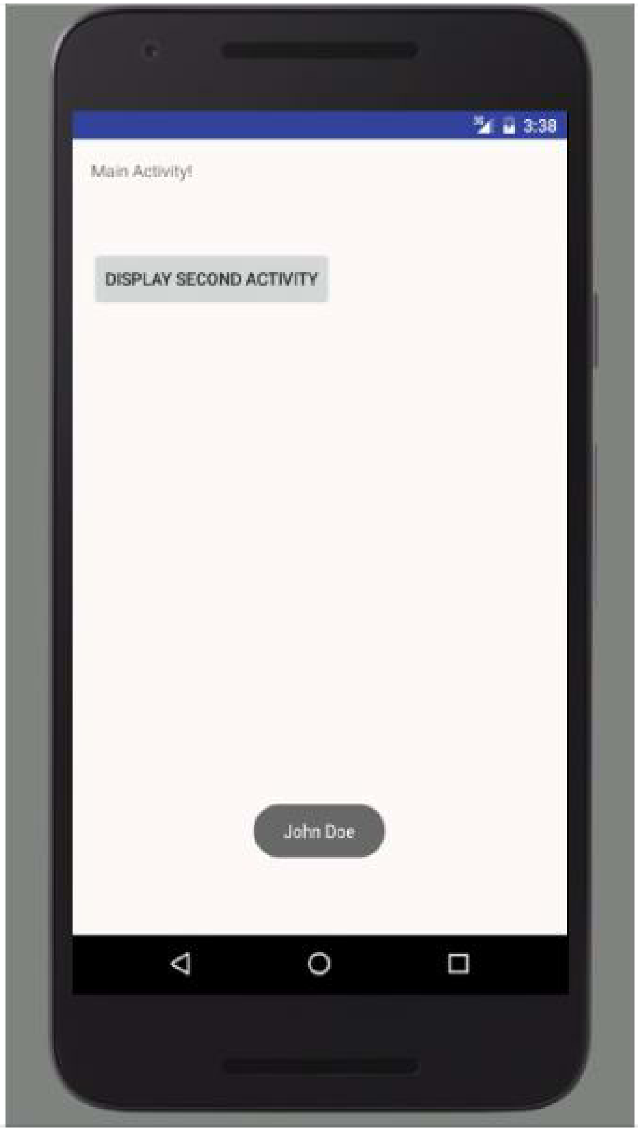
Returning Results from an Activity

SecondActivity.java

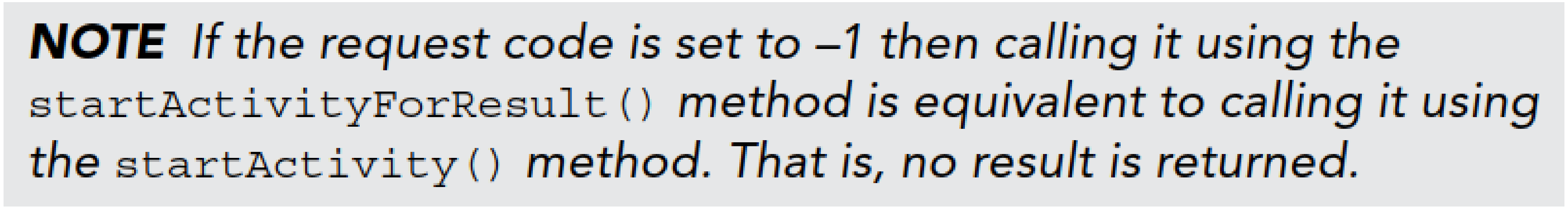
```
public void onClick(View view) {
     Intent data = new Intent();
     //---get the EditText view---
     EditText txt_username = (EditText)findViewById(R.id.txtUsername);
     //---set the data to pass back---
     data.setData(Uri.parse( txt_username.getText().toString()));
     setResult(RESULT_OK, data);
     //---closes the activity---
     finish();
}
```

Returning Results from an Activity





Returning Results from an Activity



NOTE *If the request code is set to –1 then calling it using the* `startActivityForResult()` *method is equivalent to calling it using the* `startActivity()` *method. That is, no result is returned.*

Passing Data using an Intent Object

- Besides returning data from an activity, it is also common to pass data to an activity.
- For example, you might want to set some default text in the EditText view of next activity before the activity is displayed.
- In this case, you can use the Intent object to pass the data to the target activity.

Passing Data using an Intent Object

- First, you can use the putExtra() method of an Intent object to add a name/value pair:

```
//---use putExtra() to add new name/value pairs---
i.putExtra("str1", "This is a string");
i.putExtra("age1", 25);
```

- The preceding statements add two name/value pairs to the Intent object: one of type string and one of type integer.

Passing Data using an Intent Object

- Besides using the putExtra() method, you can also create a Bundle object and then attach it using the putExtras() method.
- Think of a Bundle object as a dictionary object—it contains a set of name/value pairs.
- The following statements create a Bundle object and then add two name/value pairs to it and the Bundle is then attached to the Intent object:

```
//---use a Bundle object to add new name/values pairs---
Bundle extras = new Bundle();
extras.putString("str2", "This is another string");
extras.putInt("age2", 35);
//---attach the Bundle object to the Intent object---
i.putExtras(extras);
```

Passing Data using an Intent Object

- To obtain the data sent using the Intent object, you first obtain the Intent object using the getIntent() method.
- Then, call its getStringExtra() method to get the string value set using the putExtra() method:

```
//---get the data passed in using getStringExtra()---
Toast.makeText(this,getIntent().getStringExtra("str1"),
        Toast.LENGTH_SHORT).show();
```

Passing Data using an Intent Object

- In this case, you have to call the appropriate method to extract the name/value pair based on the type of data set.
- For the integer value, use the getIntExtra() method (the second argument is the default value in case no value is stored in the specified name):

```
//---get the data passed in using getIntExtra()---
Toast.makeText(this,Integer.toString(
        getIntent().getIntExtra("age1", 0)),
        Toast.LENGTH_SHORT).show();
```

- To retrieve the Bundle object, use the getExtras() method:

```
//---get the Bundle object passed in---
Bundle bundle = getIntent().getExtras();
```

Passing Data using an Intent Object

- To get the individual name/value pairs, use the appropriate method. For the string value, use the getString() method:

```
//---get the data using the getString()---
Toast.makeText(this, bundle.getString("str2"),
        Toast.LENGTH_SHORT).show();
```

- Likewise, use the getInt() method to retrieve an integer value:

```
//---get the data using the getInt() method---
Toast.makeText(this,Integer.toString(bundle.getInt("age2")),
        Toast.LENGTH_SHORT).show();
```

- Another way to pass data to an activity is to use the setData() method (as used in the previous section), like this:

```
//---use the setData() method to return some value---
i.setData(Uri.parse(
        "Something passed back to main activity"));
```

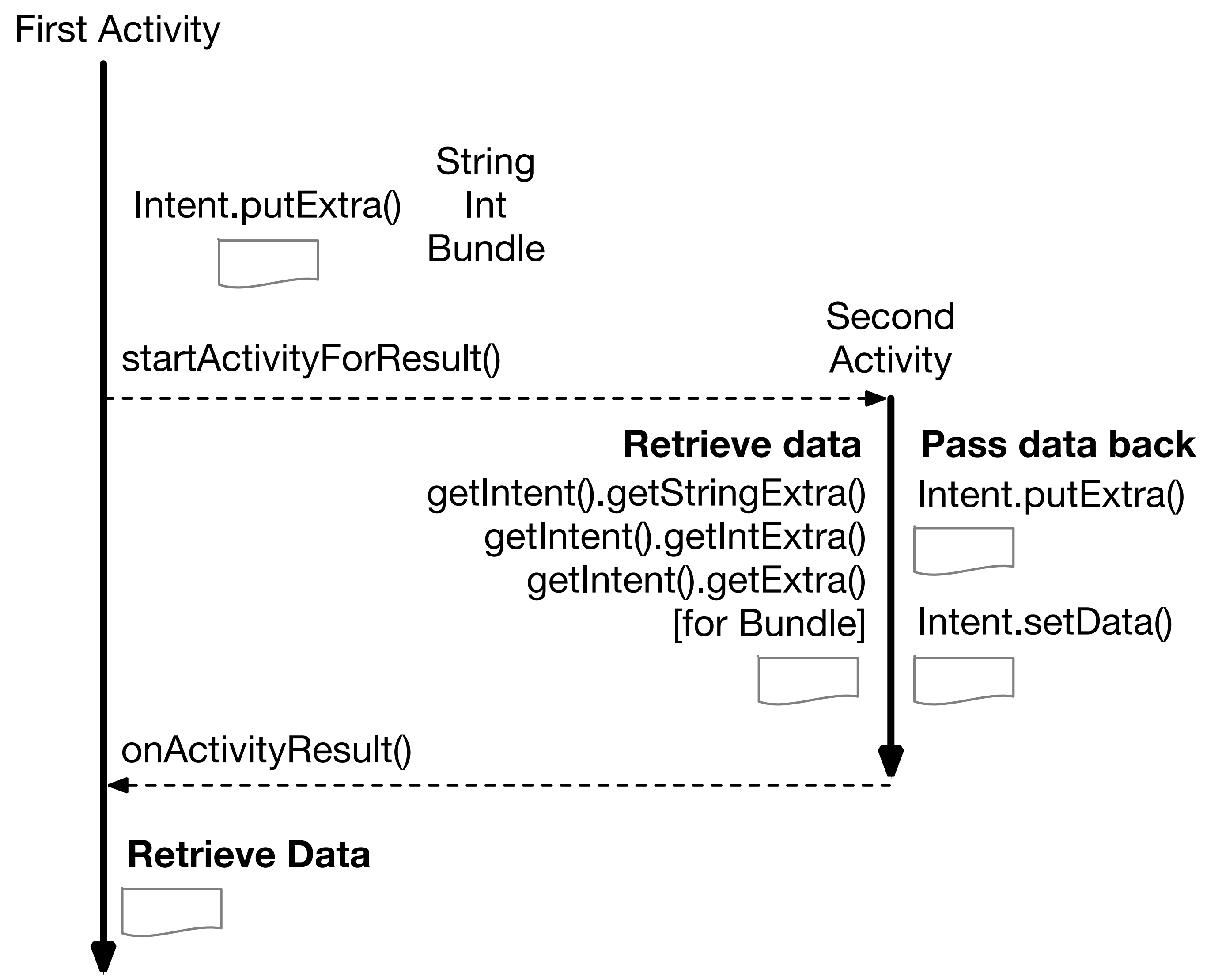
Passing Data using an Intent Object

- Usually, you use the setData() method to set the data on which an Intent object is going to operate, such as passing a URL to an Intent object so that it can invoke a web browser to view a web page.

- To retrieve the data set using the setData() method, use the getData() method (in this example data is an Intent object):

```
//---get the result using getData()---
Toast.makeText(this, data.getData().toString(),
    Toast.LENGTH_SHORT).show();
```

Passing Data using an Intent Object



Passing Data using an Intent Object

Button on the First Activity

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click to go to Second Activity"
    android:id="@+id/button"
    android:onClick="onClick"/>
```

Button on the Second Activity

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click to go to Main Activity"
    android:id="@+id/button"
    android:onClick="onClick"/>
```

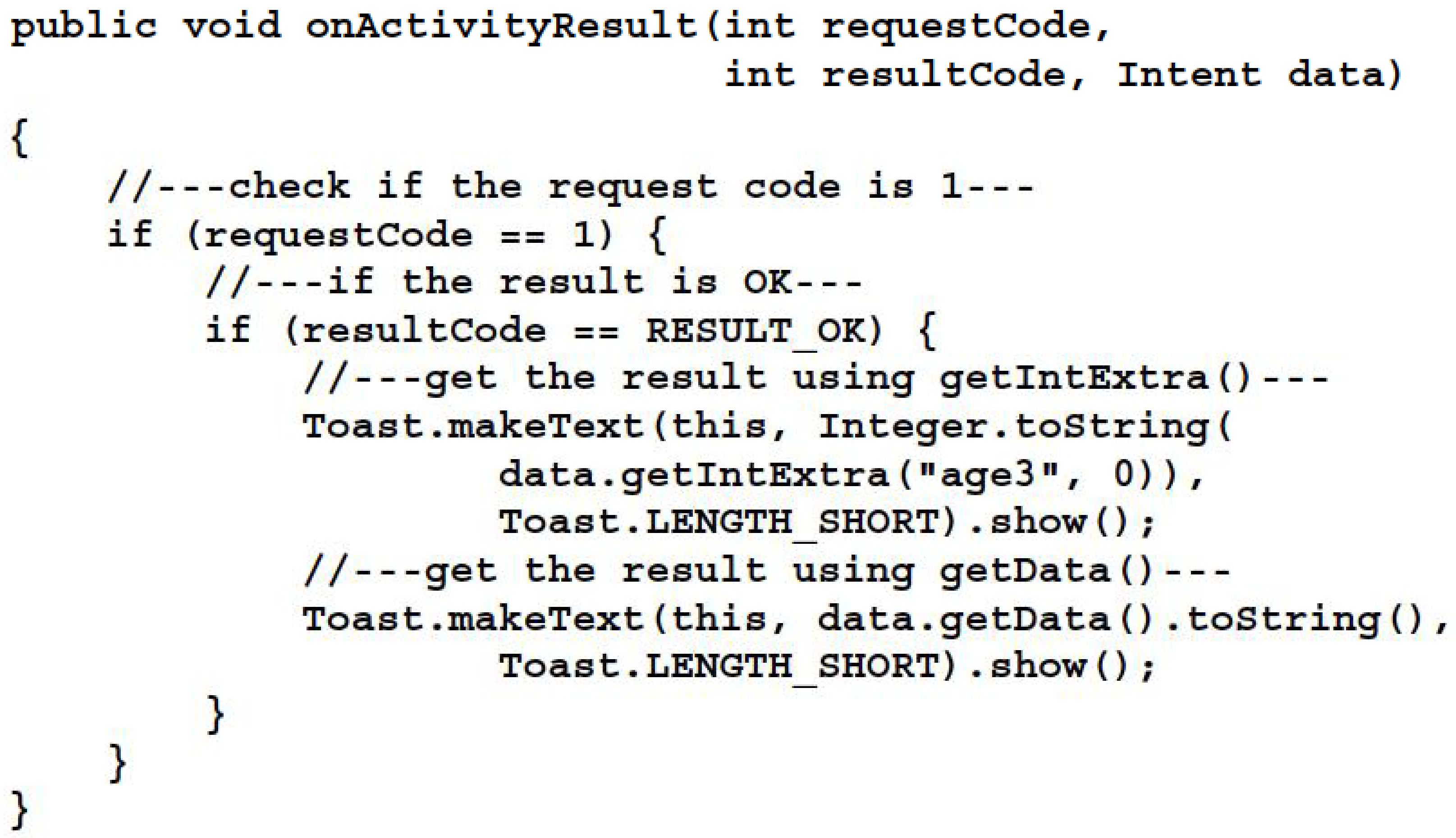
Passing Data using an Intent Object

onClick() in MainActivity.java

```
public void onClick(View view) {
    Intent i = new
            Intent("com.jfdimarzio.passingdata.SecondActivity");
    //---use putExtra() to add new name/value pairs---
    i.putExtra("str1", "This is a string");
    i.putExtra("age1", 25);
    //---use a Bundle object to add new name/values
    // pairs---
    Bundle extras = new Bundle();
    extras.putString("str2", "This is another string");
    extras.putInt("age2", 35);
    //---attach the Bundle object to the Intent object---
    i.putExtras(extras);
    //---start the activity to get a result back---
    startActivityForResult(i, 1);
}
```

Passing Data using an Intent Object

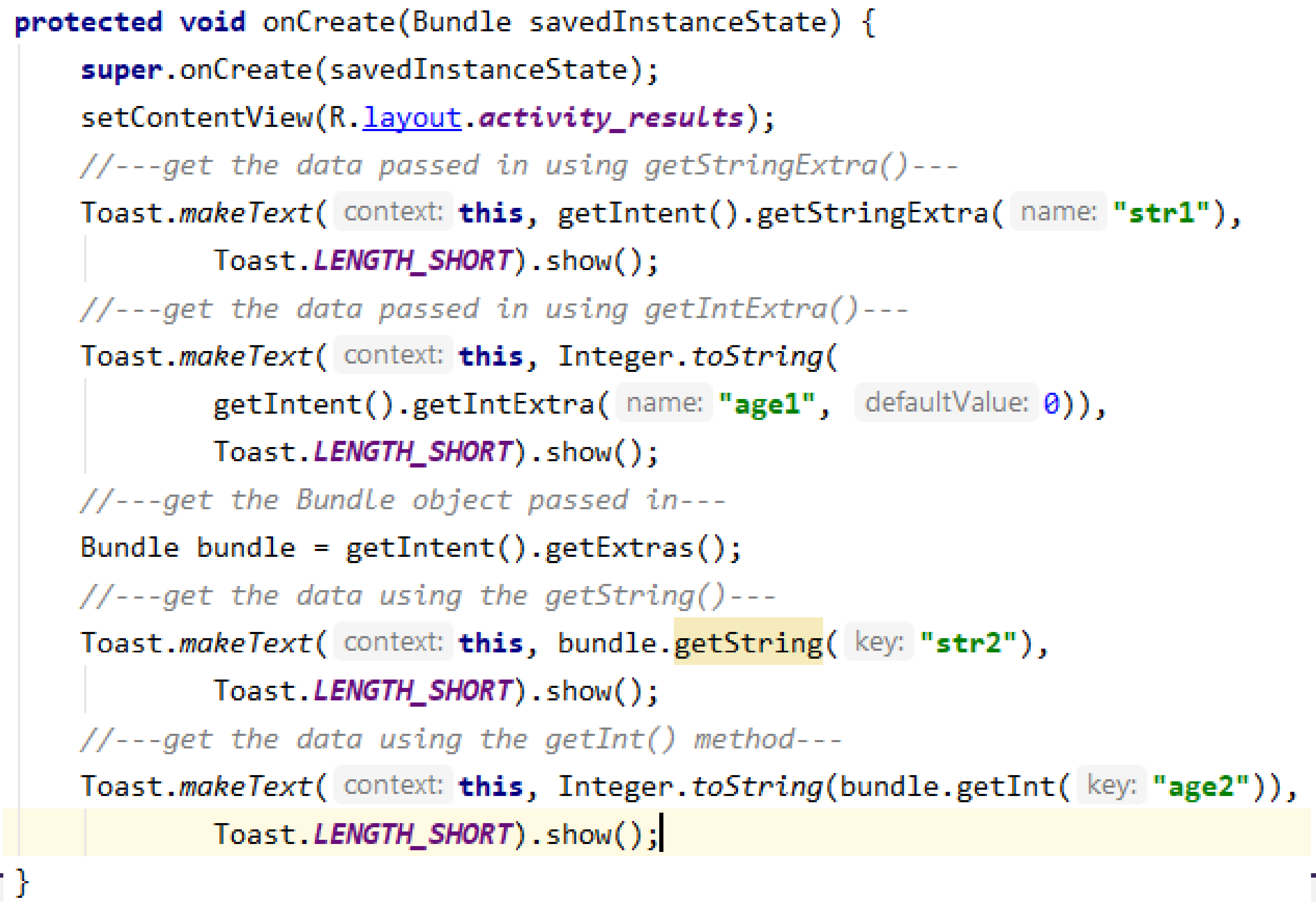
onActivityResult() in MainActivity.java



```
public void onActivityResult(int requestCode,
                             int resultCode, Intent data)
{
    //---check if the request code is 1---
    if (requestCode == 1) {
        //---if the result is OK---
        if (resultCode == RESULT_OK) {
            //---get the result using getIntExtra()---
            Toast.makeText(this, Integer.toString(
                    data.getIntExtra("age3", 0)),
                    Toast.LENGTH_SHORT).show();
            //---get the result using getData()---
            Toast.makeText(this, data.getData().toString(),
                    Toast.LENGTH_SHORT).show();
        }
    }
}
```

Passing Data using an Intent Object

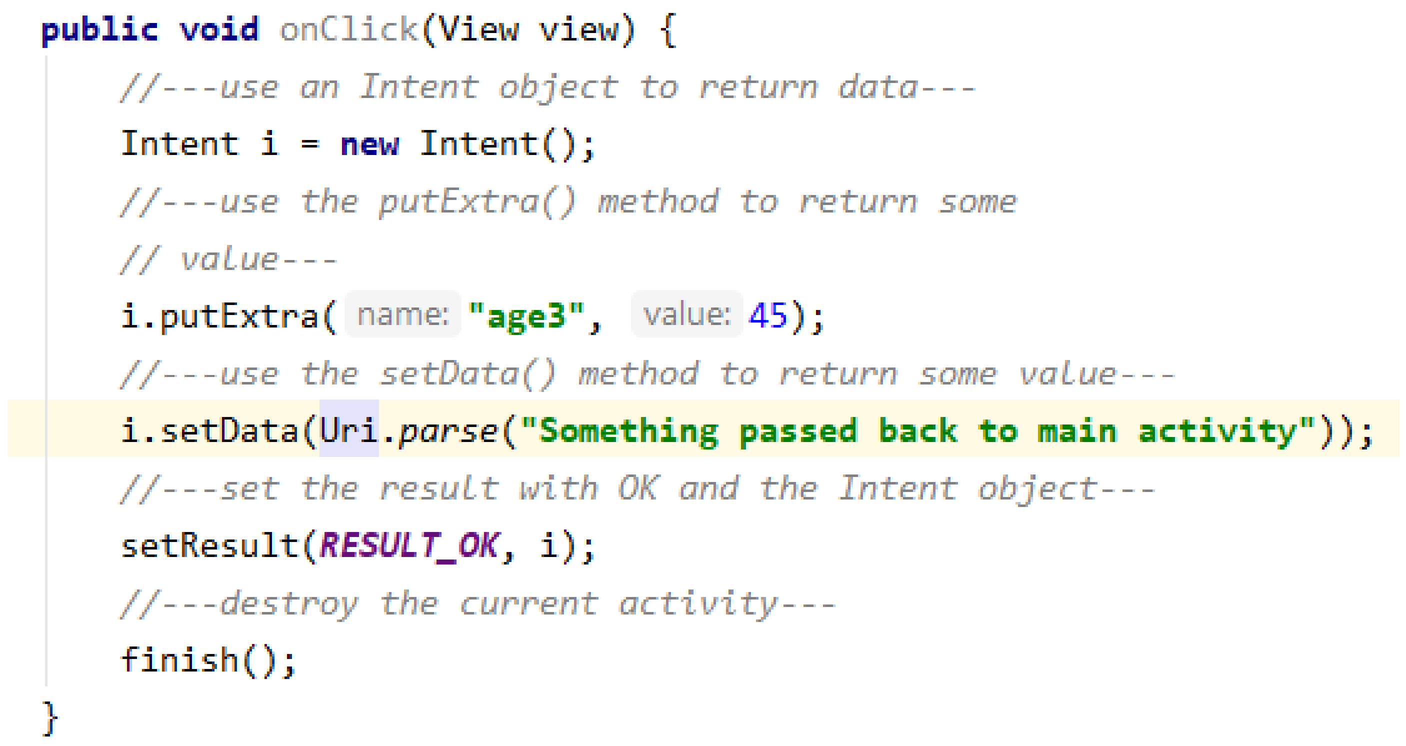
SecondActivity.java



```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_results);
    //---get the data passed in using getStringExtra()---
    Toast.makeText( context: this, getIntent().getStringExtra( name: "str1"),
            Toast.LENGTH_SHORT).show();
    //---get the data passed in using getIntExtra()---
    Toast.makeText( context: this, Integer.toString(
            getIntent().getIntExtra( name: "age1",  defaultValue: 0)),
            Toast.LENGTH_SHORT).show();
    //---get the Bundle object passed in---
    Bundle bundle = getIntent().getExtras();
    //---get the data using the getString()---
    Toast.makeText( context: this, bundle.getString( key: "str2"),
            Toast.LENGTH_SHORT).show();
    //---get the data using the getInt() method---
    Toast.makeText( context: this, Integer.toString(bundle.getInt( key: "age2")),
            Toast.LENGTH_SHORT).show();
}
```

Passing Data using an Intent Object

onClick() in SecondActivity.java



```java
public void onClick(View view) {
    //---use an Intent object to return data---
    Intent i = new Intent();
    //---use the putExtra() method to return some
    // value---
    i.putExtra( name: "age3",  value: 45);
    //---use the setData() method to return some value---
    i.setData(Uri.parse("Something passed back to main activity"));
    //---set the result with OK and the Intent object---
    setResult(RESULT_OK, i);
    //---destroy the current activity---
    finish();
}
```

Passing Data using an Intent Object

Manifest entry for SecondActivity

```
    <activity android:name=".SecondActvity" >
     <intent-filter >
         <action android:name="com.jfdimarzio.passingdata.SecondActivity" />
          <category android:name="android.intent.category.DEFAULT" />
     </intent-filter>

</activity>
```

In Class Task 03

Add a UI control on the screen of the second activity so that you can go back to the first activity (i.e., the main activity). In addition, on the main activity, display an iteration count on the number of times the main activity is displayed.

Recommended Readings

- Page # 47 to 53, Chapter # 03: Activities, Fragments, and Intents from Beginning Android Programming with Android Studio, 4th Edition by J. F. DiMarzio, Wrox, 2017
- Page # 61 to 75, Chapter # 03: Activities, Fragments, and Intents from Beginning Android Programming with Android Studio, 4th Edition by J. F. DiMarzio, Wrox, 2017

- User Guide: developer.android.com/studio/intro
- Android Vocabulary Glossary https://developers.google.com/android/for-all/vocab-words