Practical 4: Activities and intents

An Activity represents a single screen in your app with which your user can perform a single, focused task such as taking a photo, sending an email, or viewing a map. An activity is usually presented to the user as a full-screen window.

An app usually consists of multiple screens that are loosely bound to each other. Each screen is an activity. Typically, one activity in an app is specified as the "main" activity (MainActivity.java), which is presented to the user when the app is launched. The main activity can then start other activities to perform different actions.

Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack"). When a new activity starts, that new activity is pushed onto the back stack and takes user focus. The back stack follows basic "last in, first out" stack logic. When the user is done with the current activity and presses the Back button, that activity is popped from the stack and destroyed, and the previous activity resumes.

An activity is started or activated with an *intent*. An Intent is an asynchronous message that you can use in your activity to request an action from another activity, or from some other app component. You use an intent to start one activity from another activity, and to pass data between activities.

An Intent can be *explicit* or *implicit*:

- 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.
- An ***implicit intent*** is one in which you do not have the name of the target component, but you have a general action to perform.

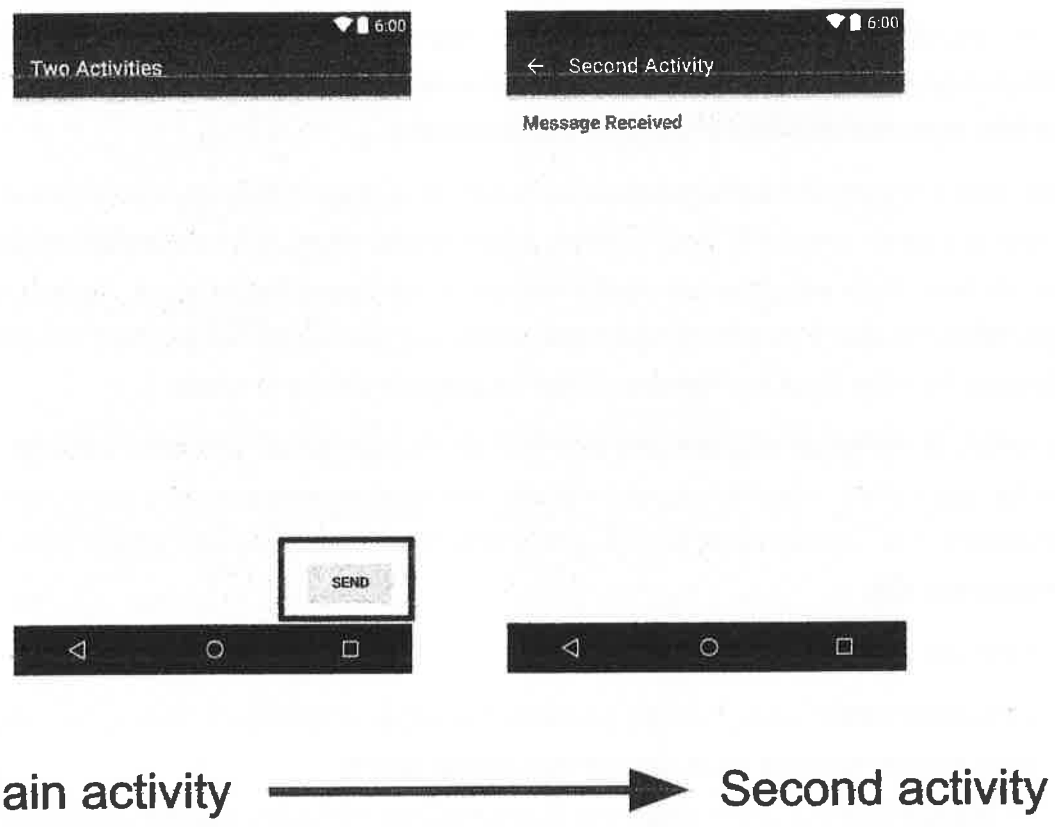
In this practical you create explicit intents. You find out how to use implicit intents in a later practical.

- Create a new Android app with a main Activity and a second Activity.
- Pass some data (a string) from the main Activity to the second using an Intent, and display that data in the second Activity.
- Send a second different bit of data back to the main Activity, also using an Intent.

App overview

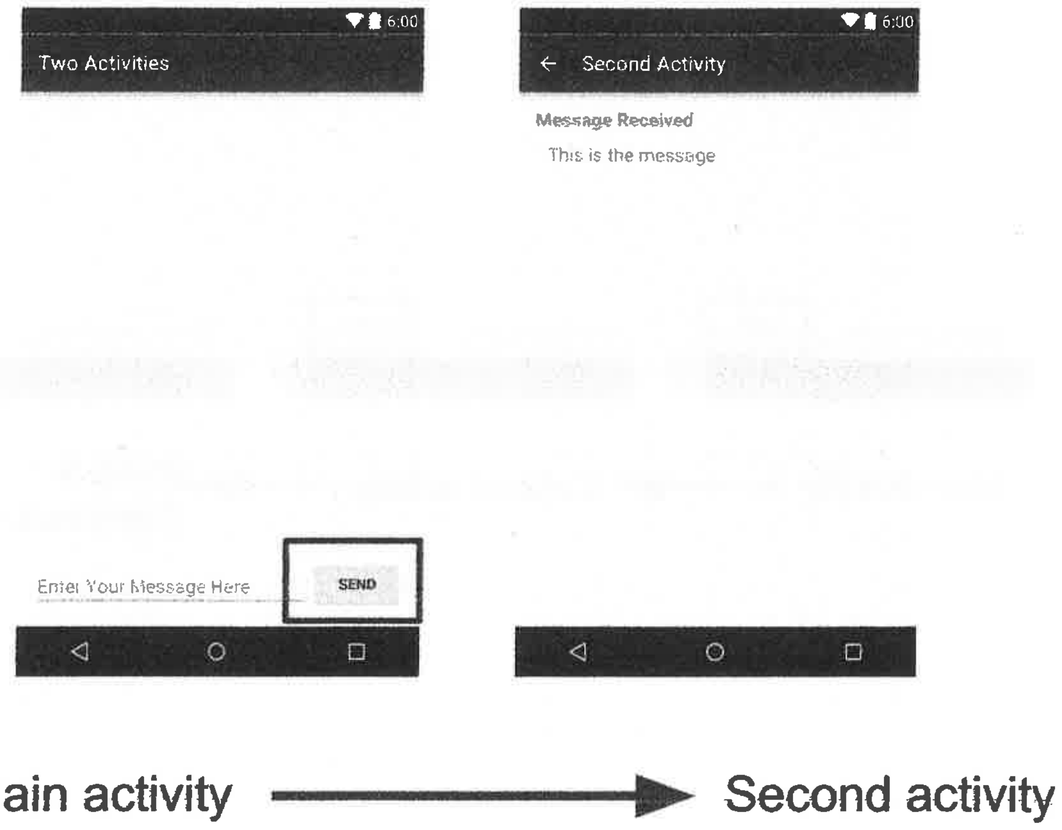
In this chapter you create and build an app called Two Activities that, unsurprisingly, contains two Activity implementations. You build the app in three stages.

In the first stage, you create an app whose main activity contains one button, **Send**. When the user clicks this button, your main activity uses an intent to start the second activity.



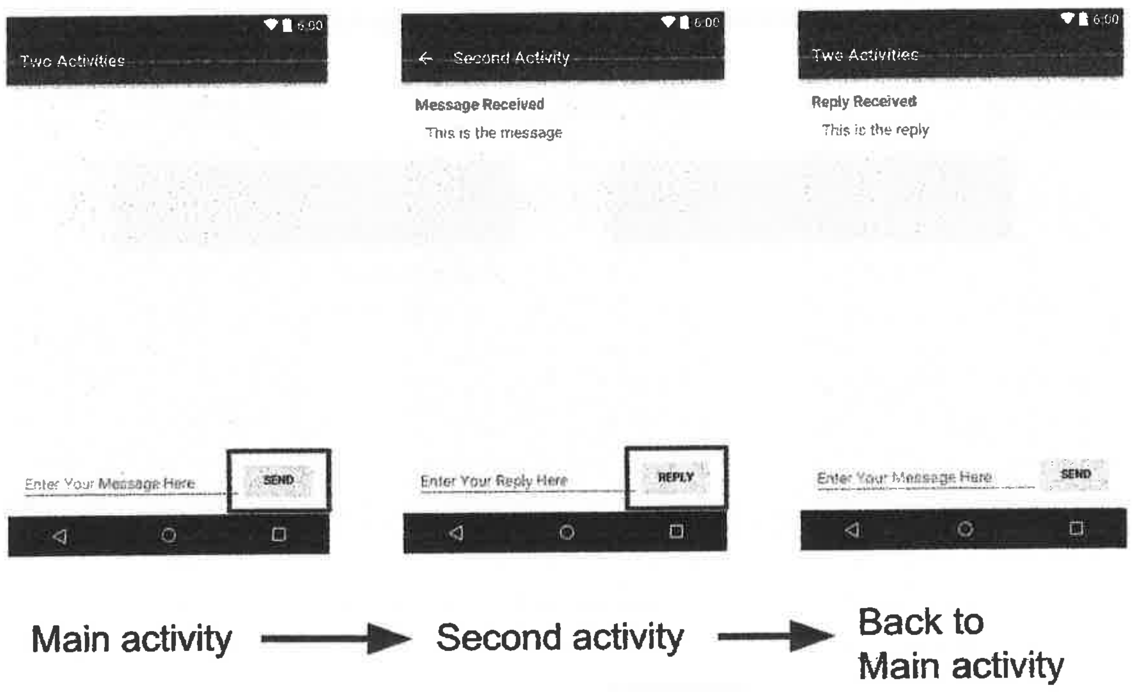
In the second stage, you add an EditText view to the main activity. The user enters a message and clicks **Send**.

The main activity uses an intent to start the second activity and send the user's message to the second activity. The second activity displays the message it received.



In the final stage of creating the Two Activities app, you add an EditText and a **Reply** button to the second activity.

The user can now type a reply message and tap **Reply**, and the reply is displayed on the main activity. At this point, you use an intent to pass the reply back from the second activity to the main activity.



Task 1: Create the TwoActivities project

In this task you set up the initial project with a main Activity, define the layout, and define a skeleton method for the onClick button event.

1.1 Create the TwoActivities project

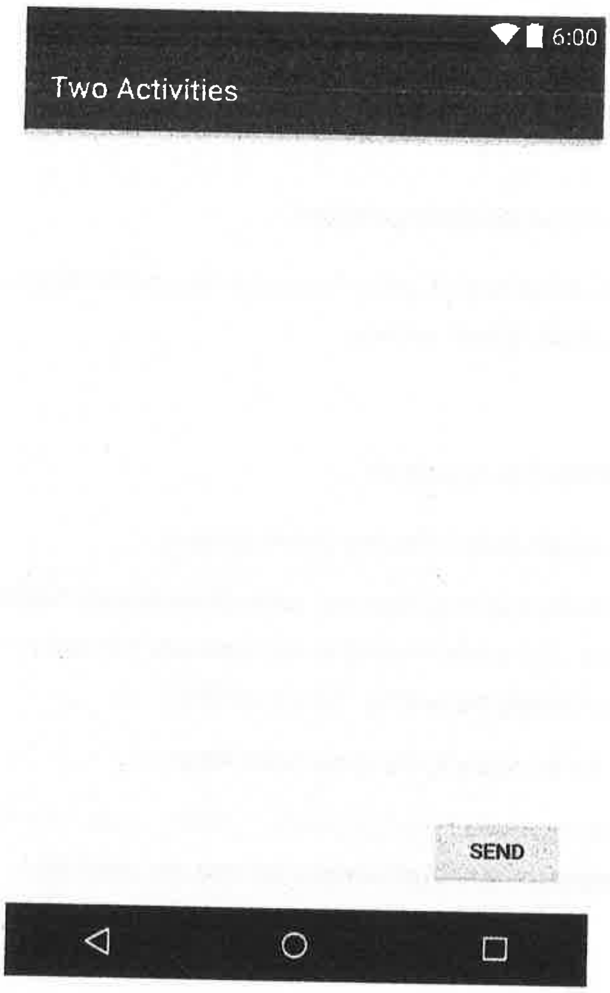
1. Start Android Studio and create a new Android Studio project.

   Name your app **Two Activities** and choose the same **Phone and Tablet** settings that you used in previous practicals. The project folder is automatically named TwoActivities, and the app name that appears in the app bar will be "Two Activities".

2. Choose **Empty Activity** for the Activity template. Click **Next.**
3. Accept the default Activity name (MainActivity). Make sure the **Generate Layout file** and **Backwards Compatibility (AppCompat)** options are checked.
4. Click **Finish**.

1.2 Define the layout for the main Activity

1. Open **res > layout > activity_main.xml** in the **Project > Android** pane. The layout editor appears.
2. Click the **Design** tab if it is not already selected, and delete the TextView (the one that says "Hello World") in the **Component Tree** pane.
3. With **Autoconnect turned on** (the default setting), drag a Button from the **Palette** pane to the lower right corner of the layout. Autoconnect creates constraints for the Button.
4. In the **Attributes** pane, set the **ID** to **button_main**, the **layout_width** and **layout_height** to **wrap_content**, and enter **Send** for the Text field. The layout should now look like this:



5. Click the **Text** tab to edit the XML code. Add the following attribute to the Button:

```
android:onClick="launchSecondActivity"
```

The attribute value is underlined in red because the **launchSecondActivity() method** has not yet been created. Ignore this error for now; you fix it in the next task.

6. **Extract the string resource**, as described in a previous practical, for "Send" and use the name button_main for the resource.

The XML code for the Button should look like the following:

```
<Button
    android:id="@+id/button_main"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="16dp"
    android:layout_marginRight="16dp"
```

```
android:text="@string/button_main"
android:onClick="launchSecondActivity"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintRight_toRightOf="parent" />
```

1.3 Define the Button action

In this task you implement the launchSecondActivity() method you referred to in the layout for the android:onClick attribute.

1. Click on **"launchSecondActivity"** in the **activity_main.xml** XML code.
2. Press **Alt+Enter** (Option+Enter on a Mac) and select **Create 'launchSecondActivity(View)' in 'MainActivity.**

   The **MainActivity file** opens, and **Android Studio generates a skeleton method for the launchSecondActivity() handler**.

3. Inside **launchSecondActivity(), add a Log statement** that says **"Button Clicked!"**

   ```
   Log.d(LOG_TAG, "Button clicked!");
   ```

   LOG_TAG will show as red. You add the definition for that variable in a later step.

4. At the **top** of the **MainActivity** class, **add a constant** for the LOG_TAG variable:

   ```
   private static final String LOG_TAG =  MainActivity.class.getSimpleName();
   ```

   This constant uses the name of the class itself as the tag.

5. Run your app. When you click the **Send** button you see the "Button Clicked!" message in the **Logcat** pane. If there's too much output in the monitor, type **MainActivity** into the search box, and the **Logcat** pane will only show lines that match that tag.

The code for **MainActivity** should look as follows:

```
package com.example.android.twoactivities;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;

public class MainActivity extends AppCompatActivity {
  private static final String LOG_TAG = MainActivity.class.getSimpleName();

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

  public void launchSecondActivity(View view) {
     Log.d(LOG_TAG, "Button clicked!");
  }
}
```

Task 2: Create and launch the second Activity

Each new activity you add to your project has its own layout and Java files, separate from those of the main activity. They also have their own <activity> elements in the AndroidManifest.xml file. As with the main activity, new activity implementations that you create in Android Studio also extend from the AppCompatActivity class.

Each activity in your app is only loosely connected with other activities. However, you can define an activity as a parent of another activity in the AndroidManifest.xml file. This parent-child relationship enables Android to add navigation hints such as left-facing arrows in the title bar for each activity.

An activity communicates with other activities (in the same app and across different apps) with an intent. An Intent can be *explicit* or *implicit*:

- 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.
- An *implicit intent* is one in which you do not have the name of the target component, but have a general action to perform.

In this task you add a second activity to our app, with its own layout. You modify the AndroidManifest.xml file to define the main activity as the parent of the second activity. Then you modify the launchSecondActivity() method in MainActivity to include an intent that launches the second activity when you click the button.

2.1 Create the second Activity

1. Click the **app** folder for your project and choose **File > New > Activity > Empty Activity**.
2. Name the new Activity **SecondActivity**. Make sure **Generate Layout File** and **Backwards Compatibility (AppCompat)** are checked. The layout name is filled in as activity_second. Do *not* check the **Launcher Activity** option.
3. Click **Finish**. Android Studio adds both a new Activity layout (activity_second.xml) and a new Java file (SecondActivity.java) to your project for the new Activity. It also updates the AndroidManifest.xml file to include the new Activity.

2.2 Modify the AndroidManifest.xml file

1. Open **manifests > AndroidManifest.xml**.
2. Find the <activity> element that Android Studio created for the second Activity.

```
<activity android:name=".SecondActivity"></activity>
```

3. Replace the entire <activity> element with the following:

```
<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>
```

The label attribute adds the title of the Activity to the app bar.

With the parentActivityName attribute, you indicate that the main activity is the parent of the second activity. This relationship is used for Up navigation in your app: the app bar for the second activity will have a left-facing arrow so the user can navigate "upward" to the main activity.

With the <meta-data> element, you provide additional arbitrary information about the activity in the form of key-value pairs. In this case the metadata attributes do the same thing as the android:parentActivityName attribute—they define a relationship between two activities for upward navigation. These metadata attributes are required for older versions of Android, because the android:parentActivityName attribute is only available for API levels 16 and higher.

4. Extract a string resource for "Second Activity" in the code above, and use activity2_name as the resource name.

2.3 Define the layout for the second Activity

1. Open **activity_second.xml** and click the **Design** tab if it is not already selected.
2. Drag a **TextView** from the **Palette** pane to the top left corner of the layout, and add constraints to the top and left sides of the layout. Set its attributes in the **Attributes** pane as follows:

3. Click the **Text** tab to edit the XML code, and extract the "Message Received" string into a resource named text_header.
4. Add the android:layout_marginLeft="8dp" attribute to the TextView to complement the layout_marginStart attribute for older versions of Android.

   The XML code for activity_second.xml should be as follows:

```
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="com.example.android.twoactivities.SecondActivity">

  <TextView
    android:id="@+id/text_header"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="16dp"
    android:text="@string/text_header"
    android:textAppearance=
              "@style/TextAppearance.AppCompat.Medium"
    android:textStyle="bold"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

2.4 Add an Intent to the main Activity

In this task you add an explicit Intent to the main Activity. This Intent is used to activate the second Activity when the **Send** button is clicked.

1. Open **MainActivity**.
2. Create a new Intent in the launchSecondActivity() method.

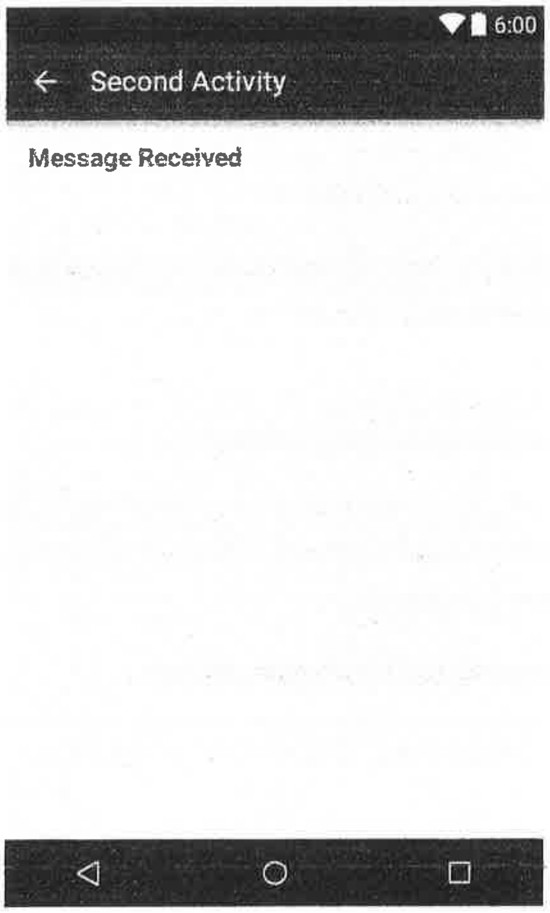
   The Intent constructor takes two arguments for an explicit Intent: an application Context and the specific component that will receive that Intent. Here you should use this as the Context, and SecondActivity.class as the specific class:

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

Attribute	Value
id	text_header
Top margin	16
Left margin	8
layout_width	wrap_content
layout_height	wrap_content
text	Message Received
textAppearance	AppCompat.Medium
textStyle	B (bold)

The value of **textAppearance** is a special Android theme attribute that defines basic font styles. You learn more about themes in a later lesson.

The layout should now look like this:



3. Call the startActivity() method with the new Intent as the argument.

```
startActivity(intent);
```

4. Run the app.

   When you click the **Send** button, MainActivity sends the Intent and the Android system launches SecondActivity, which appears on the screen. To return to MainActivity, click the **Up** button (the left arrow in the app bar) or the Back button at the bottom of the screen.

Task 3: Send data from the main Activity to the second Activity

In the last task, you added an explicit intent to MainActivity that launched SecondActivity. You can also use an intent to *send data* from one activity to another while launching it.

Your intent object can pass data to the target activity in two ways: in the data field, or in the intent *extras*. The intent data is a URI indicating the specific data to be acted on. If the information you want to pass to an activity through an intent is not a URI, or you have more than one piece of information you want to send, you can put that additional information into the *extras* instead.

The intent *extras* are key/value pairs in a Bundle. A Bundle is a collection of data, stored as key/value pairs. To pass information from one activity to another, you put keys and values into the intent extra Bundle from the sending activity, and then get them back out again in the receiving activity.

In this task, you modify the explicit intent in MainActivity to include additional data (in this case, a user-entered string) in the intent extra Bundle. You then modify SecondActivity to get that data back out of the intent extra Bundle and display it on the screen.

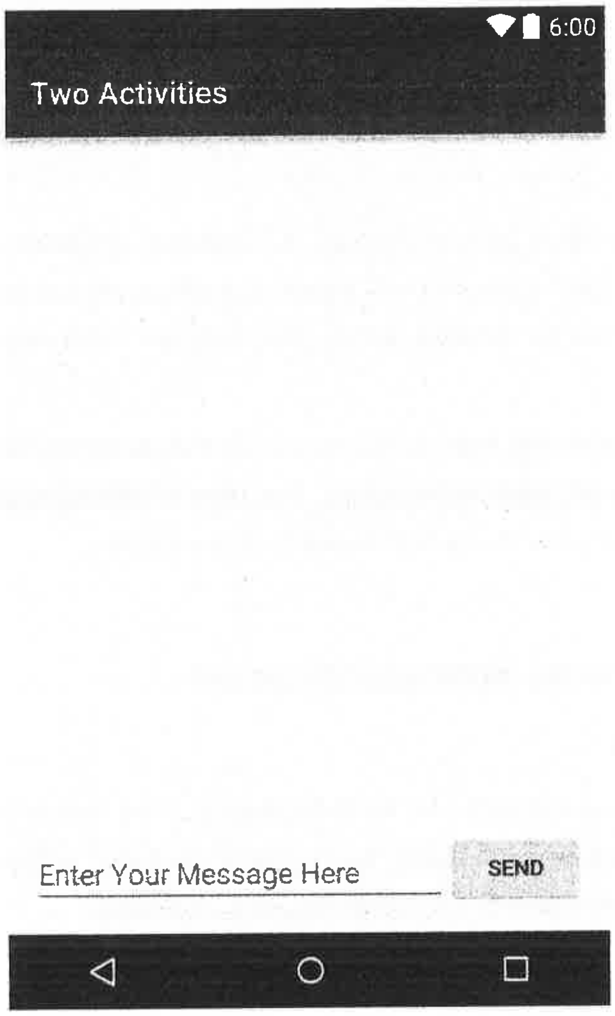
3.1 Add an EditText to the MainActivity layout

1. Open **activity_main.xml**.
2. Drag a **Plain Text** (EditText) element from the **Palette** pane to the bottom of the layout, and add constraints to the left side of the layout, the bottom of the layout, and the left side of the **Send** Button. Set its attributes in the **Attributes** pane as follows:

Attribute	Value
id	editText_main
Right margin	8
Left margin	8
Bottom margin	16
layout_width	match_constraint
layout_height	wrap_content

inputType	textLongMessage
hint	Enter Your Message Here
text	(Delete any text in this field)

The new layout in **activity_main.xml** looks like this:



3. Click the **Text** tab to edit the XML code, and extract the "Enter Your Message Here" string into a resource named editText_main.

The **XML code** for the layout should look something like the following.

```
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="com.example.android.twoactivities.MainActivity">

  <Button
    android:id="@+id/button_main"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginBottom="16dp"
    android:layout_marginRight="16dp"
    android:text="@string/button_main"
    android:onClick="launchSecondActivity"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintRight_toRightOf="parent" />

  <EditText
    android:id="@+id/editText_main"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginBottom="16dp"
    android:layout_marginEnd="8dp"
    android:layout_marginStart="8dp"
    android:ems="10"
    android:hint="@string/editText_main"
    android:inputType="textLongMessage"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toStartOf="@+id/button_main"
    app:layout_constraintStart_toStartOf="parent" />
</android.support.constraint.ConstraintLayout>
```

3.2 Add a string to the Intent extras

The **Intent** ***extras*** are **key/value pairs in a Bundle**. A Bundle is a **collection of data, stored as key/value pairs**.

To **pass information** from one Activity to another, you **put keys and values** into the **Intent extra Bundle** from the sending Activity, and then **get them back out again** in the **receiving Activity**.

1. Open **MainActivity**.
2. **Add a public constant** at the top of the class to define the key for the Intent extra:

```
public static final String EXTRA_MESSAGE =
        "com.example.android.twoactivities.extra.MESSAGE";
```

3. **Add a private variable** at the top of the class to hold the EditText:

```
private EditText mMessageEditText;
```

4. In the **onCreate() method**, use findViewById() to get a **reference to the EditText** and assign it to that private variable:

```
mMessageEditText = findViewById(R.id.editText_main);
```

5. In the **launchSecondActivity() method**, just under the new Intent, **get the text from the EditText as a string**:

```
String message = mMessageEditText.getText().toString();
```

6. **Add** that string to the Intent as an extra with the EXTRA_MESSAGE constant as the key and the string as the value:

```
intent.putExtra(EXTRA_MESSAGE, message);
```

The **onCreate() method** in **MainActivity** should now look like the following:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mMessageEditText = findViewById(R.id.editText_main);
}
```

The **launchSecondActivity() method** in **MainActivity** should now look like the following:

```
public void launchSecondActivity(View view) {
    Log.d(LOG_TAG, "Button clicked!");
    Intent intent = new Intent(this, SecondActivity.class);
    String message = mMessageEditText.getText().toString();
    intent.putExtra(EXTRA_MESSAGE, message);
    startActivity(intent);
}
```

3.3 Add a TextView to SecondActivity for the message

1. Open **activity_second.xml**.
2. **Drag** another **TextView** to the layout underneath the text_header TextView, and add constraints to the left side of the layout and to the bottom of text_header.
3. **Set the new TextView attributes** in the **Attributes** pane as follows:

Attribute	Value
id	text_message
Top margin	8
Left margin	8
layout_width	wrap_content
layout_height	wrap_content
text	(Delete any text in this field)
textAppearance	AppCompat.Medium

The new layout looks the same as it did in the previous task, because the new TextView does not (yet) contain any text, and thus does not appear on the screen.

The XML code for the **activity_second.xml** layout should look something like the following:

```
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="com.example.android.twoactivities.SecondActivity">

   <TextView
      android:id="@+id/text_header"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginStart="8dp"
      android:layout_marginTop="16dp"
      android:text="@string/text_header"
```

```
        android:textAppearance=
                "@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/text_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/text_header" />
</android.support.constraint.ConstraintLayout>
```

3.4 Modify SecondActivity to get the extras and display the message

1. Open **SecondActivity** to add code to the onCreate() method.
2. Get the Intent that activated this Activity:

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

3. Get the string containing the message from the Intent extras using the MainActivity.EXTRA_MESSAGE static variable as the key:

   ```
   String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
   ```

4. Use findViewByID() to get a reference to the TextView for the message from the layout:

   ```
   TextView textView = findViewById(R.id.text_message);
   ```

5. Set the text of the TextView to the string from the Intent extra:

   ```
   textView.setText(message);
   ```

6. Run the app. When you type a message in MainActivity and click **Send**, SecondActivity launches and displays the message.

The SecondActivity onCreate() **method** should look as follows:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_second);
   Intent intent = getIntent();
   String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
   TextView textView = findViewById(R.id.text_message);
   textView.setText(message);
}
```

Task 4 Return data back to the main Activity

Now that you have an app that launches a new activity and sends data to it, the final step is to return data from the second activity back to the main activity. You also use an intent and intent *extras* for this task.

4.1 Add an EditText and a Button to the SecondActivity layout

1. Open **strings.xml** and add string resources for the Button text and the hint for the EditText that you will add to SecondActivity:

```
<string name="button_second">Reply</string>
<string name="editText_second">Enter Your Reply Here</string>
```

2. Open **activity_main.xml** and **activity_second.xml**.
3. **Copy** the EditText and Button from the **activity_main.xml** layout file and **Paste** them into the **activity_second.xml** layout.
4. In **activity_second.xml**, **modify the attribute values** for the Button as follows:

Old attribute value	**New attribute value**
android:id="@+id/button_main"	android:id="@+id/button_second"
android:onClick= "launchSecondActivity"	android:onClick="returnReply"
android:text= "@string/button_main"	android:text= "@string/button_second"

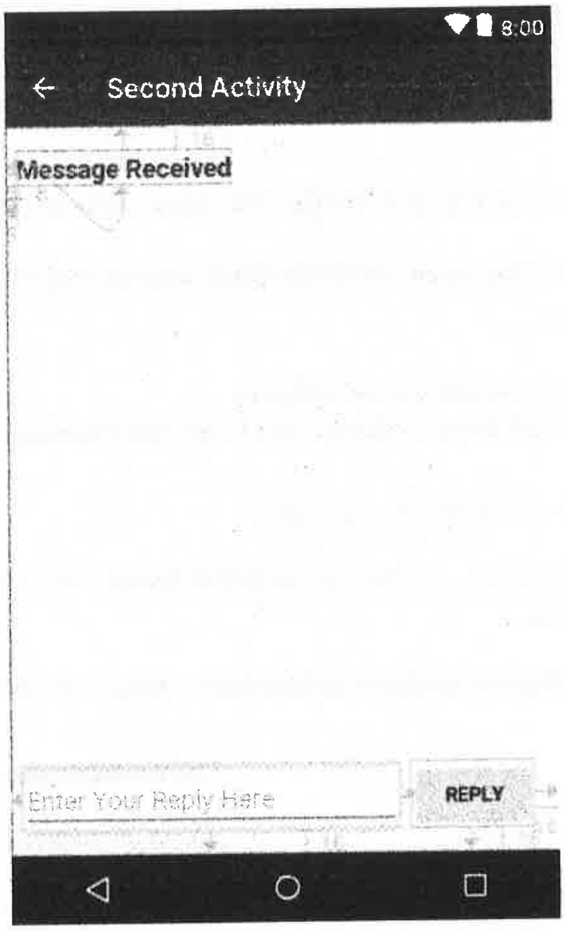
5. In activity_second.xml, modify the attribute values for the EditText as follows:

Old attribute value	**New attribute value**
android:id="@+id/editText_main"	android:id="@+id/editText_second"
app:layout_constraintEnd_toStartOf="@+id/button"	app:layout_constraintEnd_toStartOf="@+id/button_second"
android:hint= "@string/editText_main"	android:hint= "@string/editText_second"

6. In the XML layout editor, click on **returnReply**, press **Alt+Enter** (Option+Return on a Mac), and select **Create 'returnReply(View)' in 'SecondActivity'.**

   Android Studio generates a skeleton method for the **returnReply() handler**. You implement this method in the next task.

   The new layout for **activity_second.xml** looks like this:



   The XML code for the **activity_second.xml** layout file is as follows:

```
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="com.example.android.twoactivities.SecondActivity">

   <TextView
      android:id="@+id/text_header"
      android:layout_width="wrap_content"
```

4.2 Create a response Intent in the second Activity

The response data from the second Activity back to the main Activity is sent in an Intent extra. You construct this return Intent and put the data into it in much the same way you do for the sending Intent.

1. Open **SecondActivity**.
2. At the top of the class, **add a public constant** to define the key for the Intent extra:

```
public static final String EXTRA_REPLY =
                    "com.example.android.twoactivities.extra.REPLY";
```

3. **Add a private variable** at the top of the class to hold the EditText.

```
private EditText mReply;
```

4. In the **onCreate() method**, before the Intent code, use **findViewByID()** to **get a reference to the EditText** and **assign it to that private variable**:

```
mReply = findViewById(R.id.editText_second);
```

5. In the **returnReply() method**, **get the text** of the EditText as a **string**:

```
String reply = mReply.getText().toString();
```

6. In the **returnReply() method**, **create a new intent** for the response—*don't* reuse the Intent object that you received from the original request.

```
Intent replyIntent = new Intent();
```

7. **Add the reply string from the EditText to the new intent** as an **Intent *extra***. Because *extras* are **key/value pairs**, here the key is EXTRA_REPLY, and the value is the reply:

```
replyIntent.putExtra(EXTRA_REPLY, reply);
```

8. **Set the result to RESULT_OK** to indicate that the response was successful. The Activity class defines the result codes, including RESULT_OK and RESULT_CANCELLED.

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

```
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="16dp"
        android:text="@string/text_header"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/text_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/text_header" />

    <Button
        android:id="@+id/button_second"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginRight="16dp"
        android:text="@string/button_second"
        android:onClick="returnReply"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

    <EditText
        android:id="@+id/editText_second"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:ems="10"
        android:hint="@string/editText_second"
        android:inputType="textLongMessage"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/button_second"
        app:layout_constraintStart_toStartOf="parent" />
</android.support.constraint.ConstraintLayout>
```

9. **Call finish() to close** the Activity and return to MainActivity.

```
finish();
```

The code for SecondActivity should now be as follows:

```
public class SecondActivity extends AppCompatActivity {
  public static final String EXTRA_REPLY =
                "com.example.android.twoactivities.extra.REPLY";
  private EditText mReply;

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_second);
    mReply = findViewById(R.id.editText_second);
    Intent intent = getIntent();
    String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
    TextView textView = findViewById(R.id.text_message);
    textView.setText(message);
  }

  public void returnReply(View view) {
    String reply = mReply.getText().toString();
    Intent replyIntent = new Intent();
    replyIntent.putExtra(EXTRA_REPLY, reply);
    setResult(RESULT_OK, replyIntent);
    finish();
  }
}
```

4.3 Add TextView elements to display the reply

MainActivity needs a way to display the reply that SecondActivity sends. In this task you add TextView elements to the activity_main.xml layout to display the reply in MainActivity.

To make this task easier, you copy the TextView elements you used in SecondActivity.

1. Open **strings.xml** and add a string resource for the reply header:

```
<string name="text_header_reply">Reply Received</string>
```

2. Open **activity_main.xml** and **activity_second.xml**.

3. **Copy** the two TextView elements from the **activity_second.xml** layout file and paste them into the **activity_main.xml** layout above the Button.

4. In **activity_main.xml**, **modify the attribute values** for the first TextView as follows:

Old attribute value	New attribute value
android:id="@+id/text_header"	android:id="@+id/text_header_reply"
android:text="@string/text_header"	android:text="@string/text_header_reply"

5. In **activity_main.xml**, **modify the attribute values** for the second TextView a follows:

Old attribute value	New attribute value
android:id="@+id/text_message"	android:id= "@+id/text_message_reply"
app:layout_constraintTop_toBottomOf="@+id/text_header"	app:layout_constraintTop_toBottomOf="@+id/text_header_reply"

6. **Add the android:visibility attribute** to each TextView to make them initially invisible. (Having them visible on the screen, but without any content, can be confusing to the user.)

   android:visibility="invisible"

   You will make these TextView elements visible after the response data is passed back from the second Activity.

   The **activity_main.xml** layout looks the same as it did in the previous task—although you have added two new TextView elements to the layout. Because you set these elements to invisible, they do not appear on the screen.

   The following is the **XML code** for the **activity_main.xml** file:

```
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  tools:context="com.example.android.twoactivities.MainActivity">

  <TextView
    android:id="@+id/text_header_reply"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginStart="8dp"
```

4.4 Get the reply from the Intent extra and display it

When you use an explicit Intent to start another Activity, you may not expect to get any data back—you're just activating that Activity. In that case, you use startActivity() to start the new Activity, as you did earlier in this practical. If you want to get data back from the activated Activity, however, you need to start it with startActivityForResult().

In this task you modify the app to start SecondActivity expecting a result, to extract that return data from the Intent, and to display that data in the TextView elements you created in the last task.

1. Open **MainActivity**.
2. **Add a public constant** at the top of the class to define the key for a particular type of response you're interested in:

```
public static final int TEXT_REQUEST = 1;
```

3. **Add two private variables** to hold the reply header and reply TextView elements:

```
private TextView mReplyHeadTextView;
private TextView mReplyTextView;
```

4. In the **onCreate() method**, use **findViewByID()** to **get references from the layout to the reply header and reply TextView elements**. Assign those view instances to the private variables:

```
mReplyHeadTextView = findViewById(R.id.text_header_reply);
mReplyTextView = findViewById(R.id.text_message_reply);
```

The full **onCreate() method** should now look like this:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    mMessageEditText = findViewById(R.id.editText_main);
    mReplyHeadTextView = findViewById(R.id.text_header_reply);
    mReplyTextView = findViewById(R.id.text_message_reply);
}
```

```
        android:layout_marginLeft="8dp"
        android:layout_marginTop="16dp"
        android:text="@string/text_header_reply"
        android:textAppearance="@style/TextAppearance.AppCompat.Medium"
        android:textStyle="bold"
        android:visibility="invisible"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/text_message_reply"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="8dp"
        android:visibility="invisible"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/text_header_reply" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginRight="16dp"
        android:text="@string/button_main"
        android:onClick="launchSecondActivity"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintRight_toRightOf="parent" />

    <EditText
        android:id="@+id/editText_main"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginBottom="16dp"
        android:layout_marginEnd="8dp"
        android:layout_marginStart="8dp"
        android:ems="10"
        android:hint="@string/editText_main"
        android:inputType="textLongMessage"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/button2"
        app:layout_constraintStart_toStartOf="parent" />
</android.support.constraint.ConstraintLayout>
```

5. In the **launchSecondActivity() method**, **change** the call to **startActivity()** to be **startActivityForResult()**, and include the TEXT_REQUEST key as an argument:

```
startActivityForResult(intent, TEXT_REQUEST);
```

6. **Override** the **onActivityResult()** callback method with this signature:

```
@Override
public void onActivityResult(int requestCode,
                        int resultCode, Intent data) {
}
```

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

- the **requestCode** you set when you launched the Activity with startActivityForResult(),
- the **resultCode** set in the launched Activity (usually one of RESULT_OK or RESULT_CANCELED), and
- the **Intent data** that contains the data returned from the launch Activity.

7. Inside **onActivityResult()**, call **super.onActivityResult()**:

```
super.onActivityResult(requestCode, resultCode, data);
```

8. **Add** code to test for TEXT_REQUEST to make sure you process the right Intent result, in case there are several. Also test for RESULT_OK, to make sure that the request was successful:

```
if (requestCode == TEXT_REQUEST) {
  if (resultCode == RESULT_OK) {
  }
}
```

The Activity class defines the result codes. The code can be RESULT_OK (the request was successful), RESULT_CANCELED (the user cancelled the operation), or RESULT_FIRST_USER (for defining your own result codes).

9. Inside the inner if block, **get the Intent extra from the response Intent (data).** Here the key for the extra is the EXTRA_REPLY constant from SecondActivity:

```
String reply = data.getStringExtra(SecondActivity.EXTRA_REPLY);
```

10. **Set** the **visibility** of the reply header to **true**:

```
mReplyHeadTextView.setVisibility(View.VISIBLE);
```

11. **Set** the reply TextView text to the reply, and set its **visibility** to **true**:

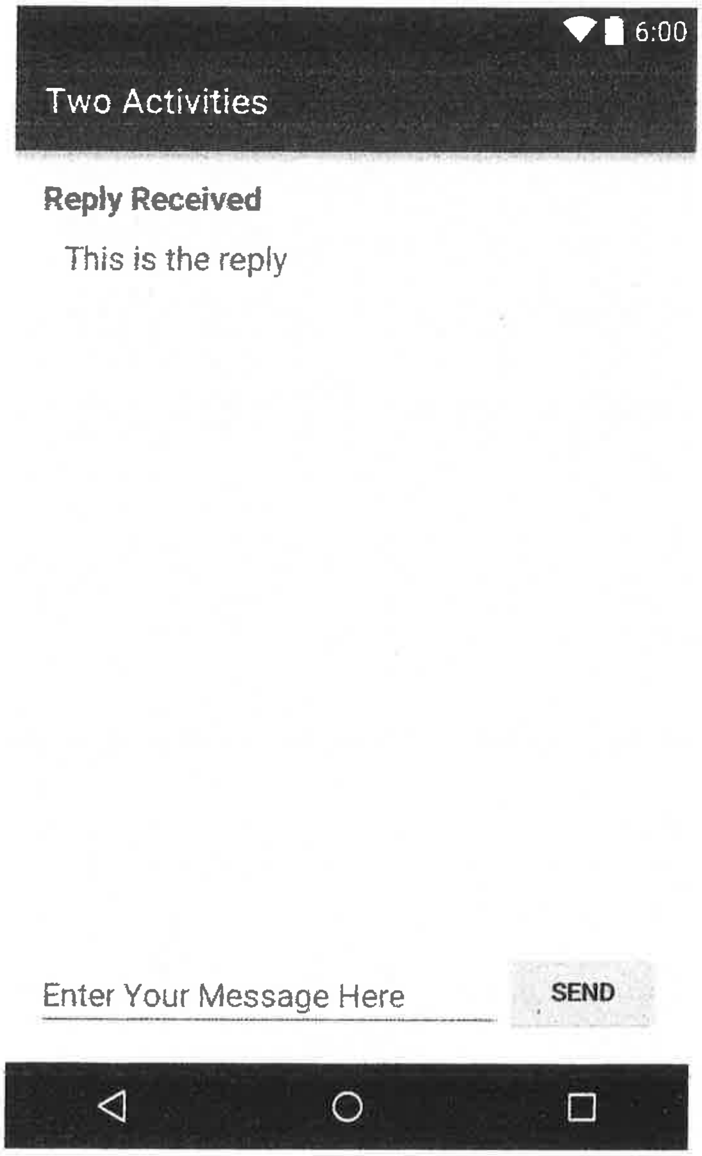
```
mReplyTextView.setText(reply);
mReplyTextView.setVisibility(View.VISIBLE);
```

The full onActivityResult() **method** should now look like this:

```
@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);
         mReplyHeadTextView.setVisibility(View.VISIBLE);
         mReplyTextView.setText(reply);
         mReplyTextView.setVisibility(View.VISIBLE);
      }
   }
}
```

12. **Run** the app.

Now, when you send a message to the second Activity and get a reply, the main Activity updates to display the reply.


6:00

Two Activities

Reply Received

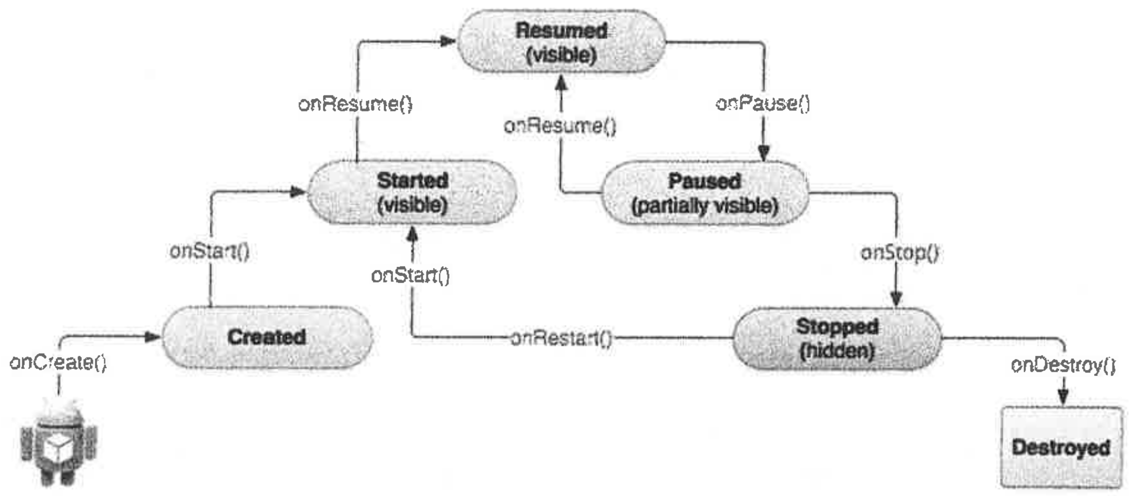
This is the reply

Enter Your Message Here

SEND

Activity lifecycle and state

In this practical you learn more about the *activity lifecycle*. The lifecycle is the set of states an activity can be in during its entire lifetime, from when it's created to when it's destroyed and the system reclaims its resources. As a user navigates between activities in your app (as well as into and out of your app), activities transition between different states in their lifecycles.



Each stage in an activity's lifecycle has a corresponding callback method: onCreate(), onStart(), onPause(), and so on. When an activity changes state, the associated callback method is invoked. You've already seen one of these methods: onCreate(). By overriding any of the lifecycle callback methods in your Activity classes, you can change the activity's default behavior in response to user or system actions.

The activity state can also change in response to device-configuration changes, for example when the user rotates the device from portrait to landscape. When these configuration changes happen, the activity is destroyed and recreated in its default state, and the user might lose information that they've entered in the activity. To avoid confusing your users, it's important that you develop your app to prevent unexpected data loss. Later in this practical you experiment with configuration changes and learn how to preserve an activity's state in response to device configuration changes and other activity lifecycle events.