Mulitple Activities and Intent

Session 2

Learning Objectives

- At the end of this meeting is expected that students will be able to:
  - Describe the main features of Android Programming and Android Software Development
  - Produce simple Mobile Application using the main features of Android

Contents

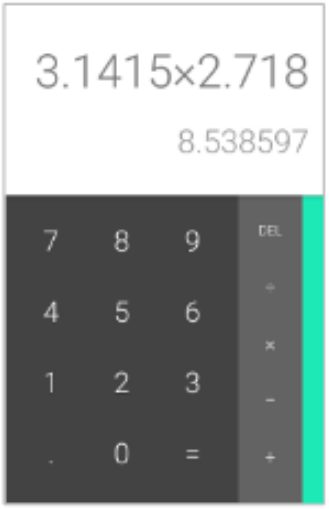
- Apps can Contain More Than One Activity
- Apps Structure
- How Android App works?
- Specify Action

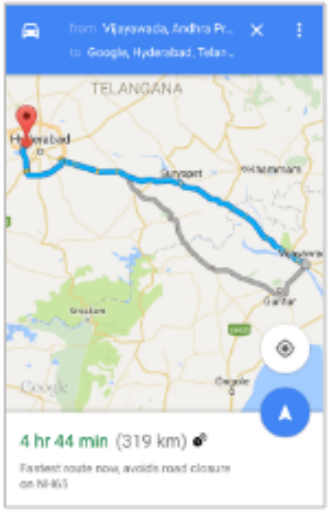
What is an Activity? And What Does an Activity do?

- 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 a appear as floating window
- Java class, typically one Activity in one file
- 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 (will discussed on session 3)

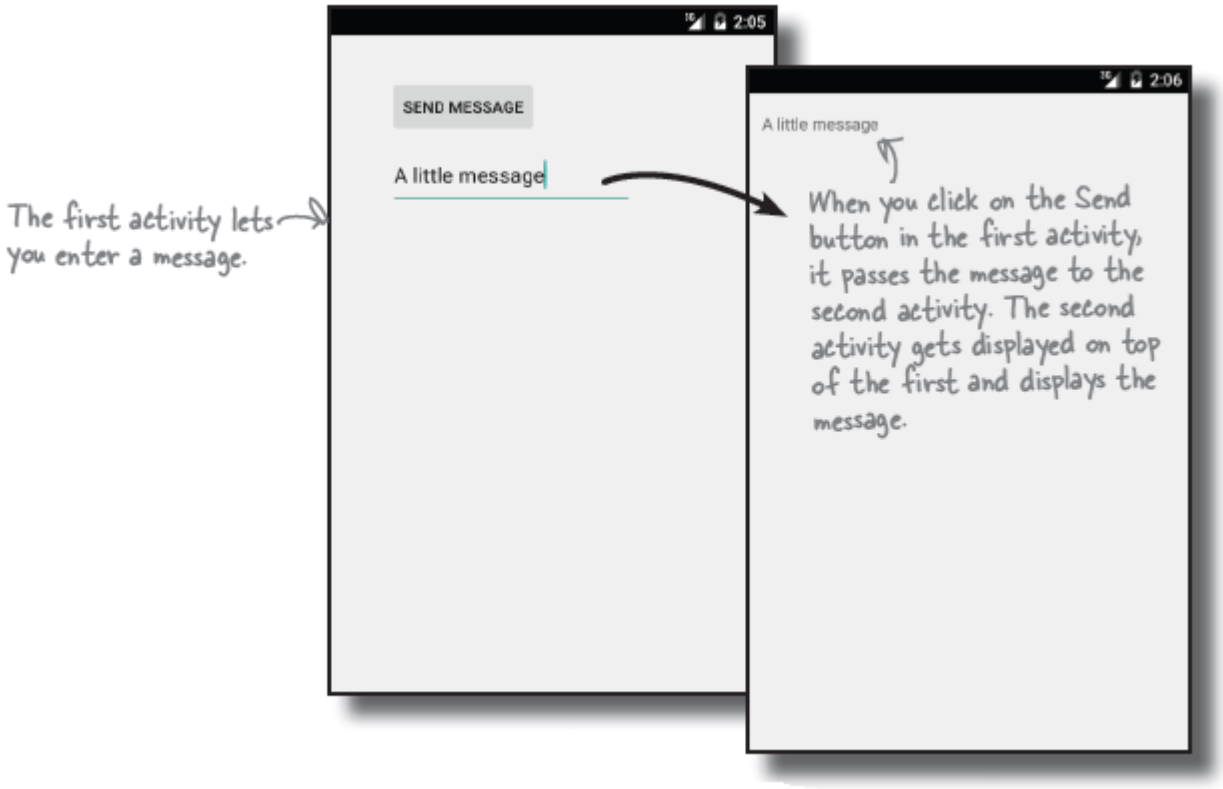
Apps can Contain More Than One Activity

- An activity is a single focused thing your user can do. If you chain multiple activities together to do something more complex, It's called task.
- A lot of the time, you'll want users to do more than just one thing— for example, adding recipes as well as displaying a list of them. If this is the case, you'll need to use multiple activities: one for displaying the list of recipes and another for adding a single recipe.



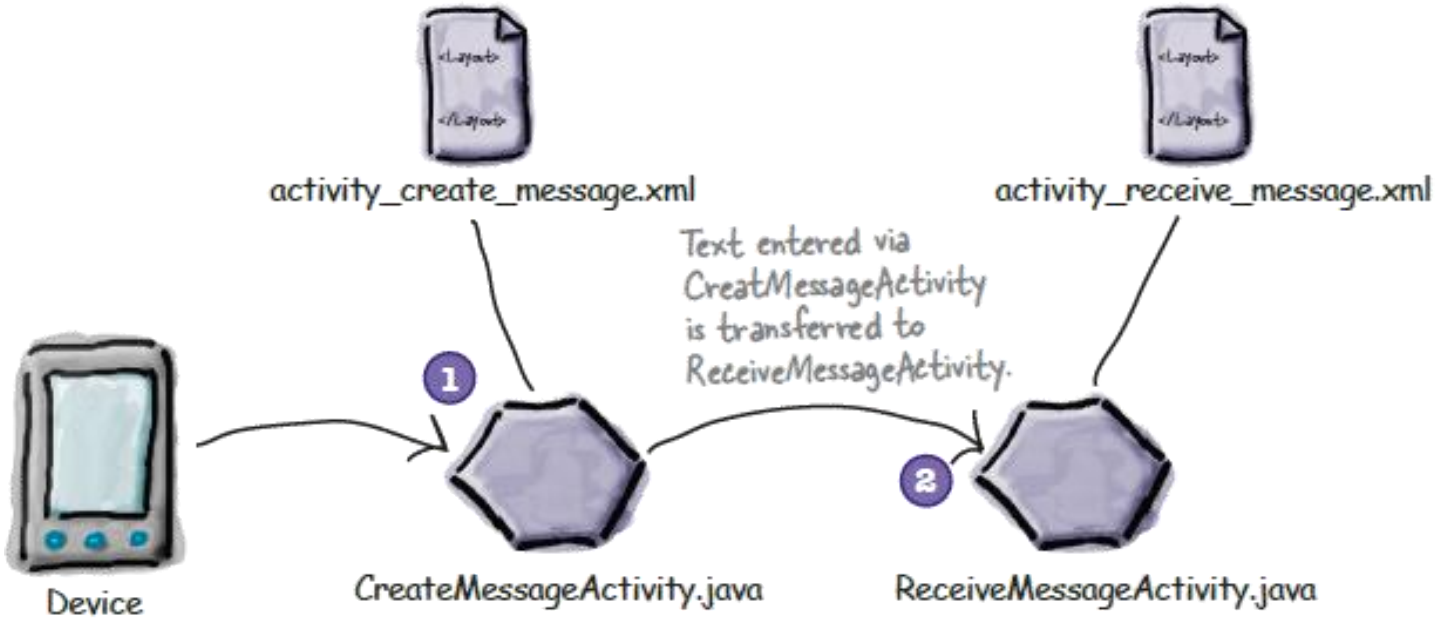


Apps can Contain More Than One Activity (Cont.)



1. Create a basic app with a single activity and layout.
2. Add a second activity and layout.
3. Get the first activity to call the second activity.
4. Get the first activity to pass data to the second activity.

App Structure



1 **When the app gets launched, it starts activity CreateMessageActivity.**
This activity uses the layout *activity_create_message.xml*.

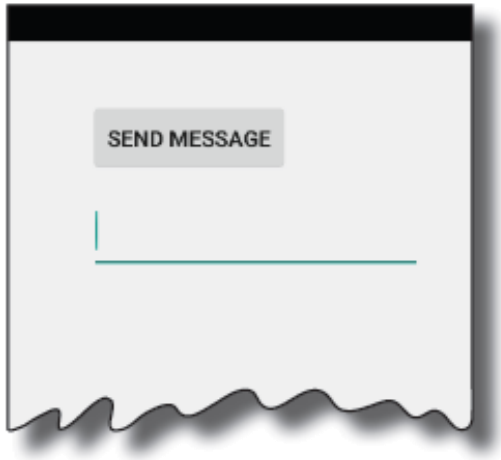
2 **The user clicks on a button in CreateMessageActivity.**
This launches activity `ReceiveMessageActivity`, which uses layout *activity_receive_message.xml*.

Example – Create the 1st activity

- Create a new Android Studio project for an application named "Messenger" with a package name of com.hfad.messenger.
- The minimum SDK should be API 15 so that it will work on most devices.
- You'll need a blank activity called "CreateMessageActivity" with a layout called "activity_create_message" so that your code matches ours.

Example – Create the 1st activity

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context=".CreateMessageActivity" >

    <Button
        android:id="@+id/send"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="36dp"
        android:layout_marginTop="21dp"
        android:onClick="onSendMessage"
        android:text="@string/send" />

    <EditText
        android:id="@+id/message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/send"
        android:layout_below="@+id/send"
        android:layout_marginTop="18dp"
        android:ems="10" />

</RelativeLayout>
```

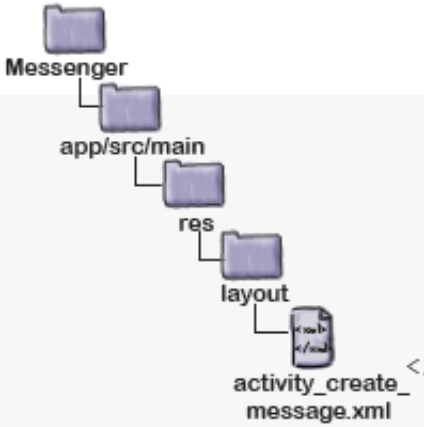


Replace the <TextView> Android Studio gives you with the <Button> and <EditText>.

Clicking on the button runs the onSendMessage() method in the activity.

This is a String resource.

This describes how wide the <EditText> should be. It should be wide enough to accommodate 10 letter M's.

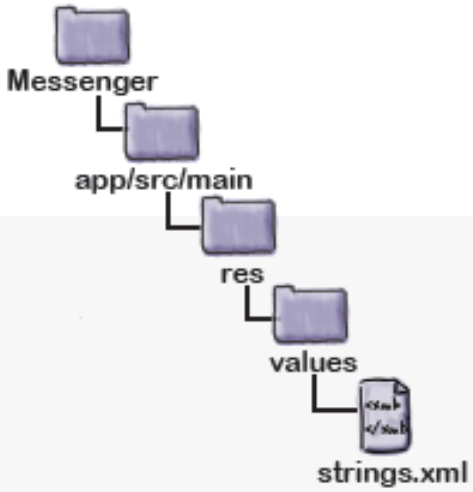


Example – Create the 1st activity

- Update string.xml
  - The button we added has a text value of @string/send. This means we need to add a string called “send” to *strings.xml and give it a value.*

```
...
    <string name="send">Send Message</string>
...
```

Add a new String called send. We gave ours a value of Send Message so that the text "Send Message" appears on the button.



Example – Create the 1st activity

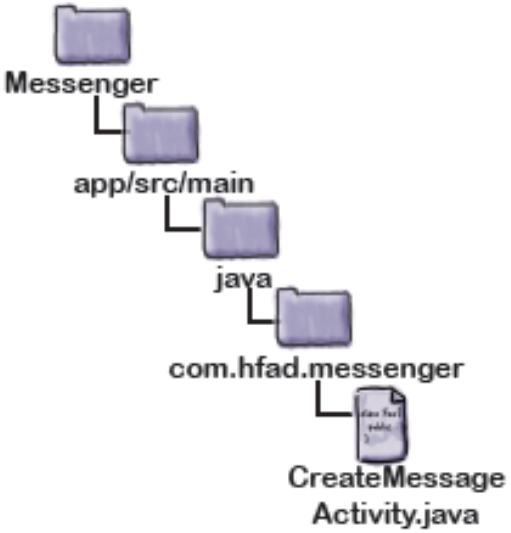
- And add method to the activity

```
package com.hfad.messenger;

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

public class CreateMessageActivity extends Activity {

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

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
    }
}
```

We're replacing the code that Android Studio created for us, as most of the code it creates isn't required.

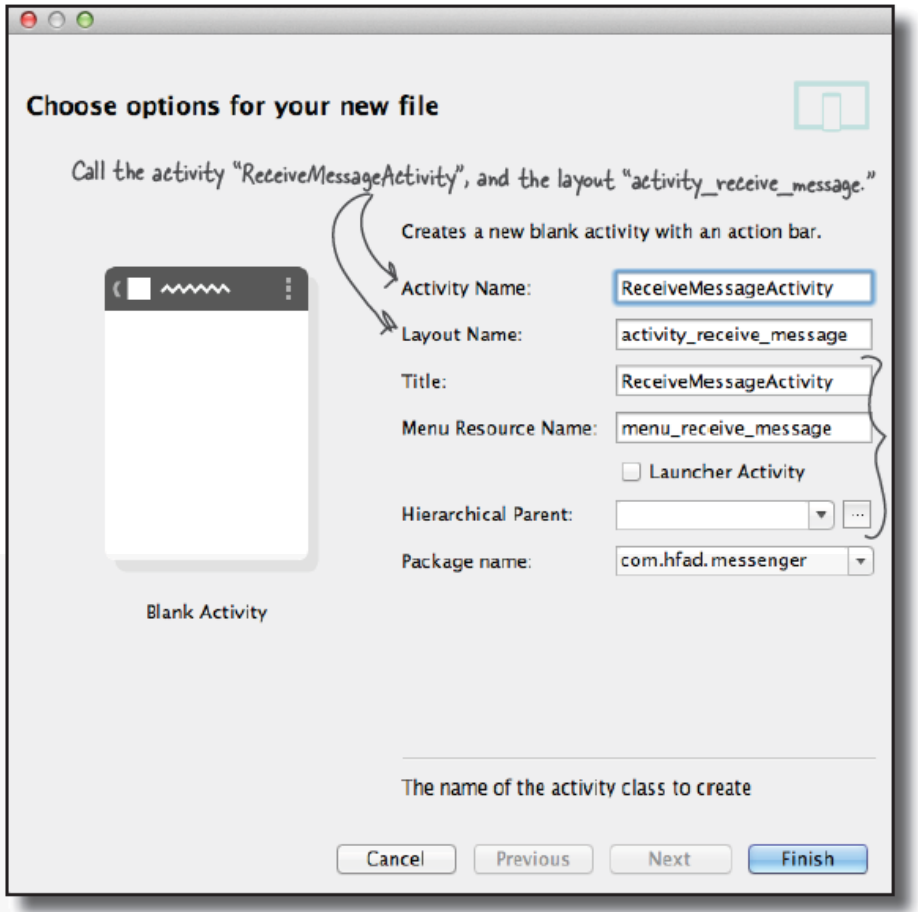
The onCreate() method gets called when the activity is created.

This method will get called when the button's clicked. We'll complete the method body as we work our way through the rest of the chapter.



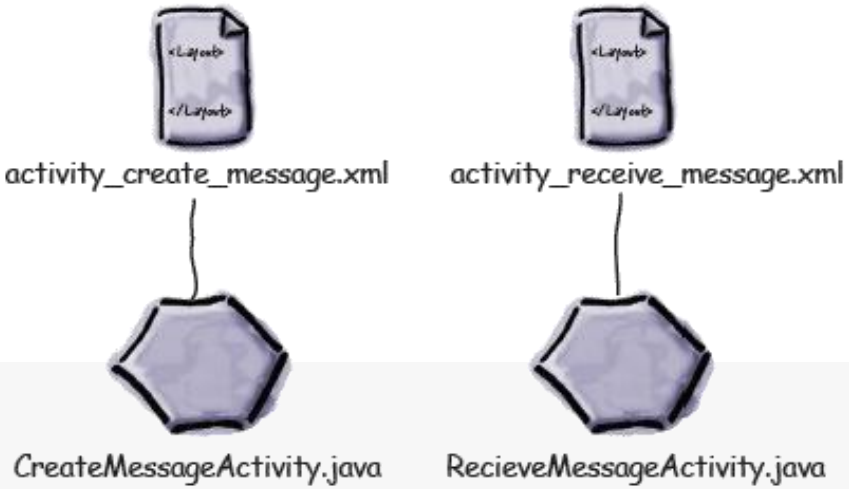
Example – Create 2nd Activity

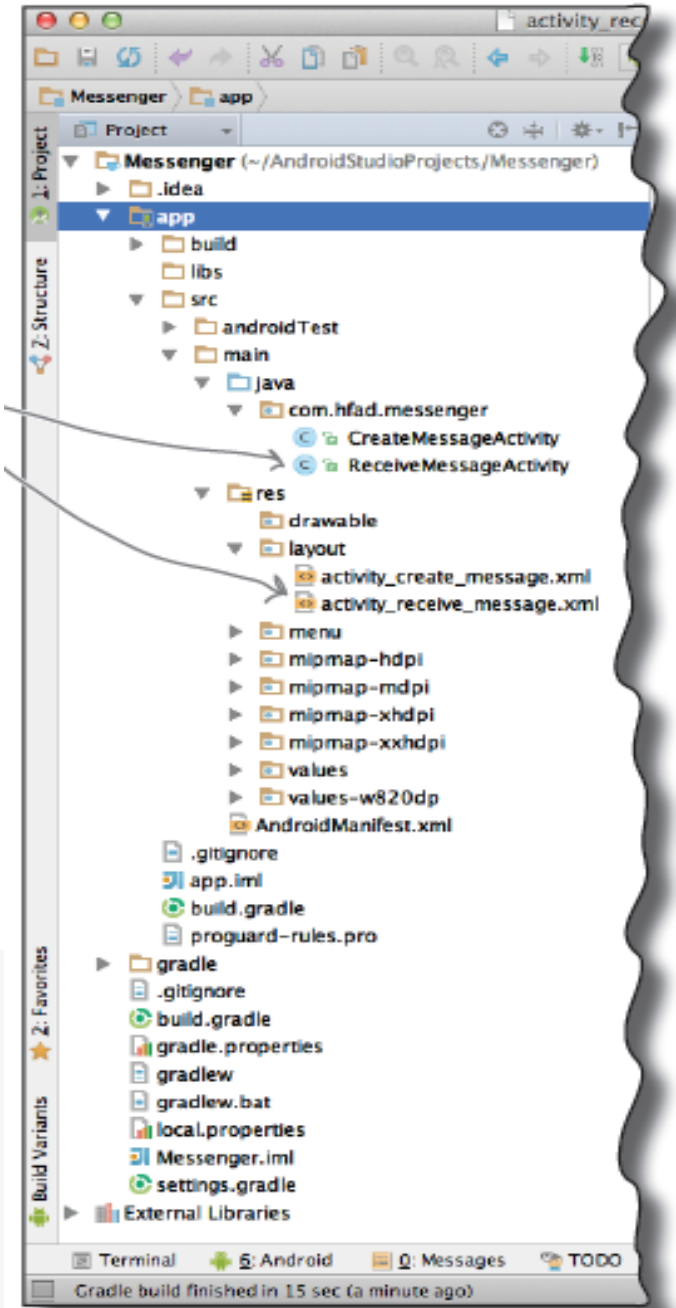
- To create the new activity, choose File → New → Activity, and choose the option for Blank Activity.
- Give the new activity a name of "ReceiveMessageActivity" and the layout a name of "activity_receive_message"



Example – Create 2nd Activity

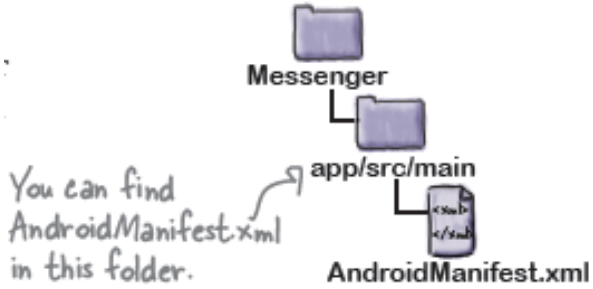
- Each activity uses a different layout. CreateMessageActivity uses the layout *activity_ create_message.xml*, *and ReceiveMessageActivity uses the layout activity_receive_message.xml*.





Example – Create 2nd Activity

- AndroidManifest.xml




```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hfad.messenger" >

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >

        <activity
            android:name=".CreateMessageActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <activity
            android:name=".ReceiveMessageActivity"
            android:label="@string/title_activity_receive_message" >
        </activity>

    </application>
</manifest>
```

This is package name we specified.

Android Studio gave our app a default icon. We'll look at this later in the book.

The theme affects the appearance of the app. We'll look at this later.

This is the first activity, Create Message Activity.

This bit specifies that it's the main activity of the app.

This says the activity can be used to launch the app.

This is the second activity, Receive Message Activity.

Android Studio added these lines for us when we added the second activity.

Every activity needs to be declared

- All activities need to be declared in AndroidManifest.xml. If an activity isn't declared in the file, the system won't know it exists.
- You declare an activity in the manifest by including an <activity> element inside the <application> element.

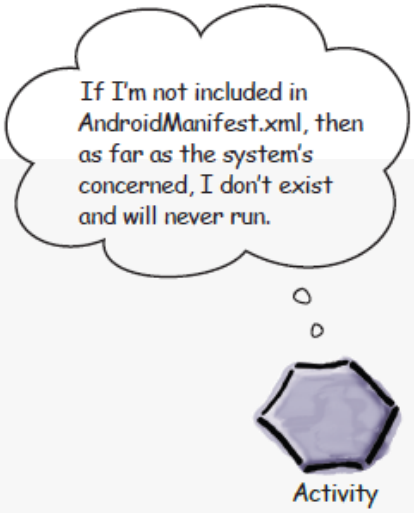
```
<application
    ...
    ...>
    <activity
        android:name="activity_class_name"
        android:label="@string/activity_label"
        ...
        ...>
    ...
    </activity>
    ...
</application>
```

Each activity needs to be declared inside the <application> element

This line is mandatory.

This line is optional, but Android Studio completes it for us.

The activity may have other properties too.



Every activity needs to be declared

- The following line is mandatory and is used to specify the class name of the activity::

```
android:name="activity_class_name"
```

- activity_class_name is the name of the class, prefixed with a “.”. In this case, it’s .ReceiveMessageActivity, because Android combines the class name with the name of the package to derive the *fully qualified class name.*
- This line is optional and is used to specify a user-friendly label for the activity:

```
android:label="@string/activity_label"
```

Example – Call 2nd Activity

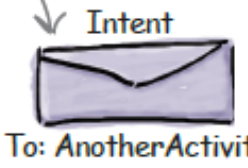
- An Intent is a type of message.
- When the app is launched, our first activity, CreateMessageActivity, will run. What we need to do next is get CreateMessageActivity to call ReceiveMessageActivity when the user clicks the Send Message button.
- Whenever you want an activity to start a second activity, you use an **intent.** You can think of an intent as an "**intent to do something**".
- It's a type of message that allows you to bind separate objects (such as activities) together at runtime. If one activity wants to start a second activity, it does it by sending an intent to Android. Android will start the second activity and pass it the intent.
- You can start an activity by creating an intent and using it in the **startActivity()** method.

Example – Call 2nd Activity

- You start by creating the intent like this:

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

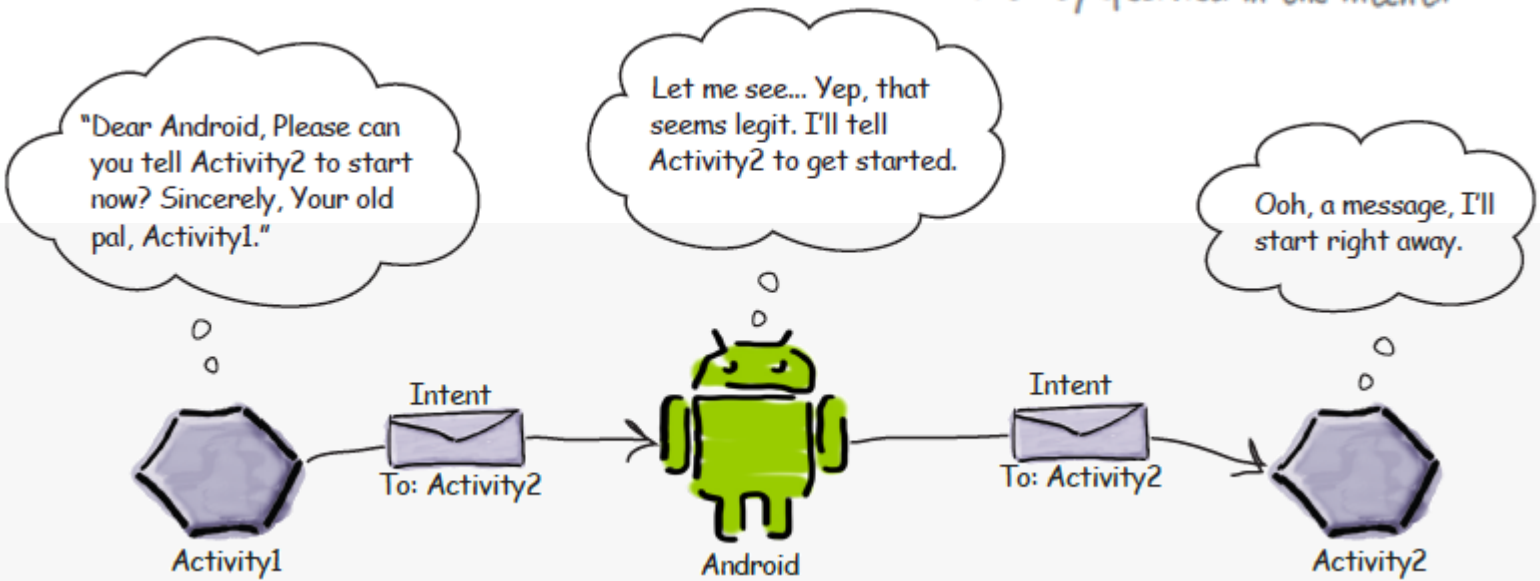
The intent specifies the activity you want to receive it. It's like putting an address on an envelope.



- The first parameter tells Android which object the intent is from, and you can use the word this to refer to the current activity. The second parameter is the class name of the activity that needs to receive the intent.
- Once you've created the intent, you pass it to Android like this:

```
startActivity(intent);
```

startActivity() starts the activity specified in the intent.



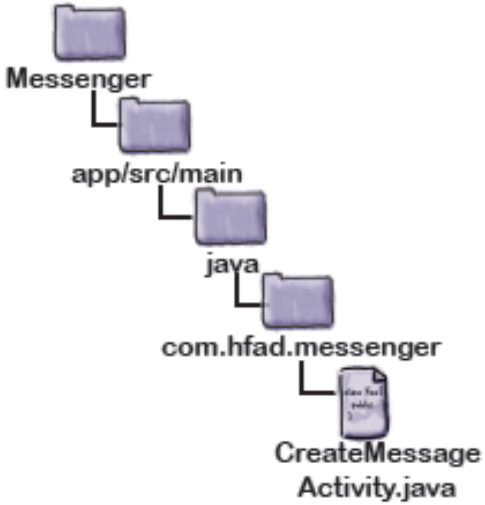
Example – Call 2nd Activity

```
package com.hfad.messenger;

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

public class CreateMessageActivity extends Activity {

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

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        startActivity(intent);
    }

}
```

We need to import the Intent class android.content.Intent as we're using it in onSendMessage().

We've not changed this method.

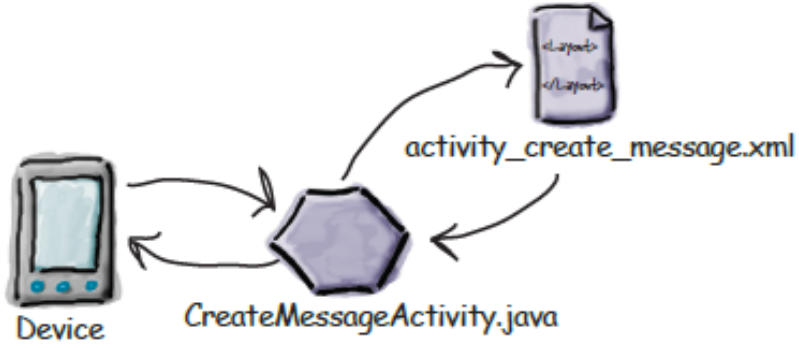
Start activity ReceiveMessageActivity.



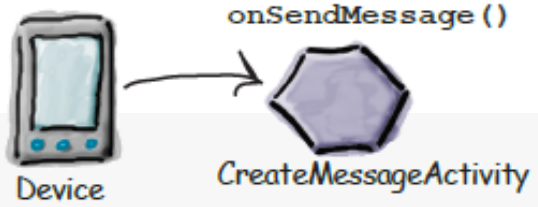
Example – Call 2nd Activity

- What happens when you run the app

1 **When the app gets launched, the main activity, CreateMessageActivity starts.** When it starts, the activity specifies that it uses layout *activity_create_message.xml*. This gets displayed in a new window.



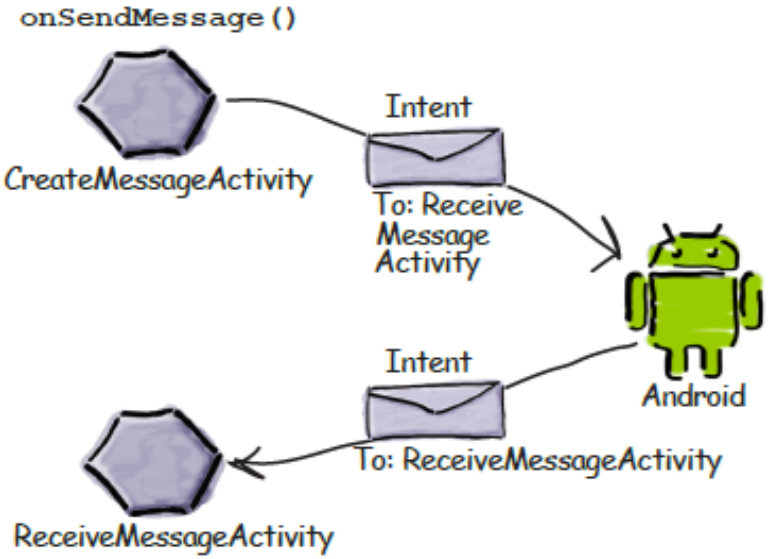
2 **The user clicks on a button.** The `onSendMessage()` method in `CreateMessageActivity` responds to the click.



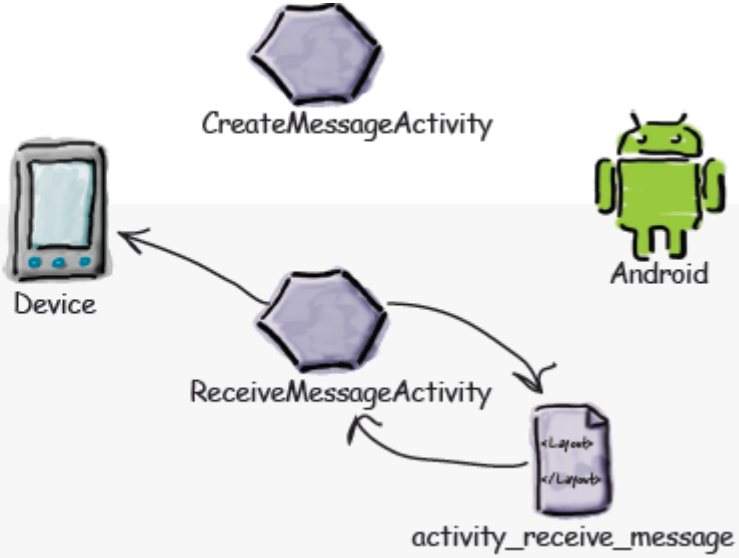
Example – Call 2nd Activity

- What happens when you run the app (Cont.)

3 **The onSendMessage() method tells Android to start activity ReceiveMessageActivity using an intent.**
Android checks that the intent is OK, and then it tells `ReceiveMessageActivity` to start.



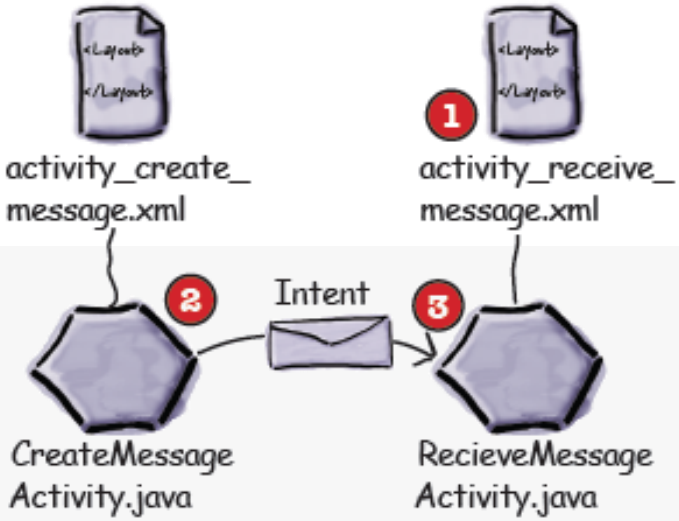
4 **When ReceiveMessageActivity starts, it specifies that it uses layout *activity_receive_message.xml* and this gets displayed in a new window.**



Example – Pass Data

- Pass text to a second activity

1. Tweak the layout *activity_receive_message.xml* so that we can display the text. At the moment it's the default layout the wizard gave us.
2. Update *CreateMessageActivity.xml* so that it gets the text the user inputs. It then needs to add the text to the intent before it sends it.
3. Update *ReceiveMessageActivity.java* so that it displays the text sent in the intent.

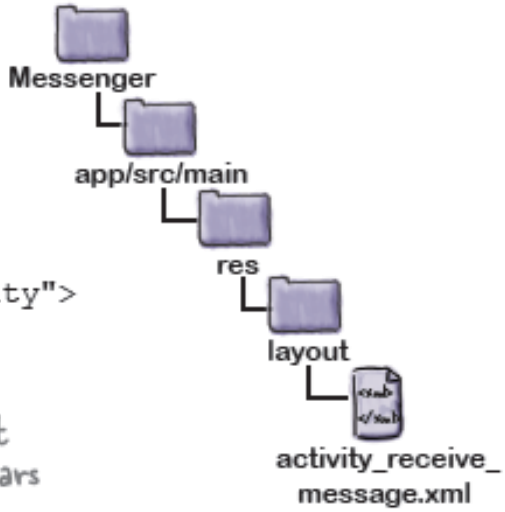


Example – Pass Data

- *activity_receive_message.xml*

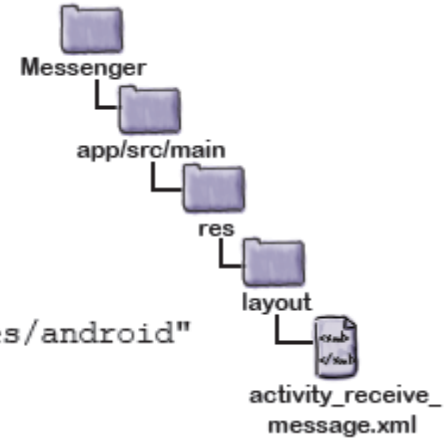
```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.hfad.messenger.ReceiveMessageActivity">

    <TextView
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

Here's the text view that currently appears in the layout.



Example – Pass Data

- Update the text view properties




```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context="com.hfad.messenger.ReceiveMessageActivity">

    <TextView
        android:id="@+id/message"
        android:text="@string/hello_world"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</RelativeLayout>
```

This line gives the <TextView> an ID of message.

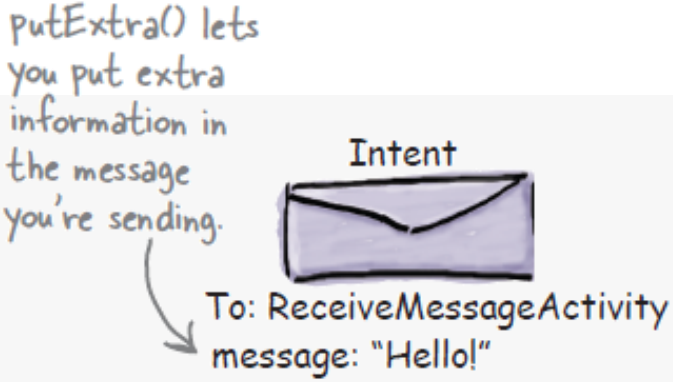
Remove the line that sets the text to @string/hello_world.

Example – Pass Data

- You can add extra information to this intent that can be picked up by the activity you're targeting so it can react in some way. To do this, you use the putExtra() method.

```
intent.putExtra("message", value);
```

- where message is a String name for the value you're passing in, and value is the value. The putExtra() method is overloaded so value has many possible types.



Example – Pass Data

- Use getIntent() to retrieve extra information from an intent.
- getIntent() returns the intent that started the activity, and you can use this to retrieve any extra information that was sent along with it.


```
Intent intent = getIntent();
String string = intent.getStringExtra("message");
```

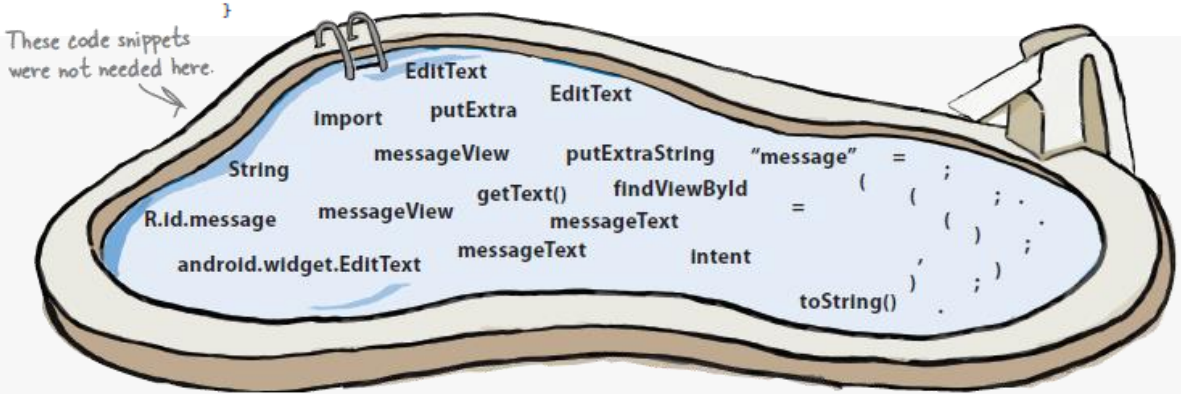
Get the intent.

Get the string passed along with the intent that has a name of "message".

Pool Puzzle

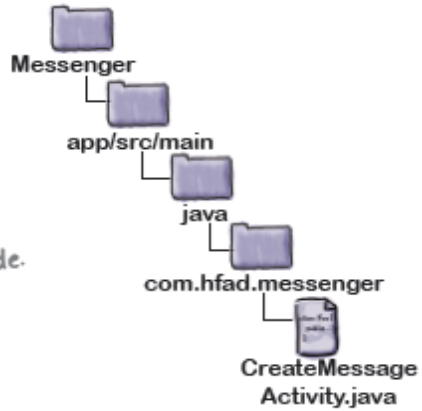
```
package com.hfad.messenger;

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

public class CreateMessageActivity extends Activity {

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

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        .....
        .....
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        .....
        startActivity(intent);
    }
}
```

These code snippets were not needed here.



Example – Pass Data

- Update the CreateMessageActivity code




```
package com.hfad.messenger;

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

public class CreateMessageActivity extends Activity {

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

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        EditText messageView = (EditText)findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        intent.putExtra(ReceiveMessageActivity.EXTRA_MESSAGE, messageText);
        startActivity(intent);
    }
}
```

You need to import the EditText class android.widget.EditText as you're using it in your activity code.

Get the text that's in the EditText.

Create an intent, then add the text to the intent. We're using a constant for the name of the extra information so that we know CreateMessageActivity and ReceiveMessageActivity are using the same String. We'll add this to ReceiveMessageActivity on the next page.

Start ReceiveMessageActivity with the intent.

Example – Pass Data

- Get ReceiveMessageActivity to use the information in the intent

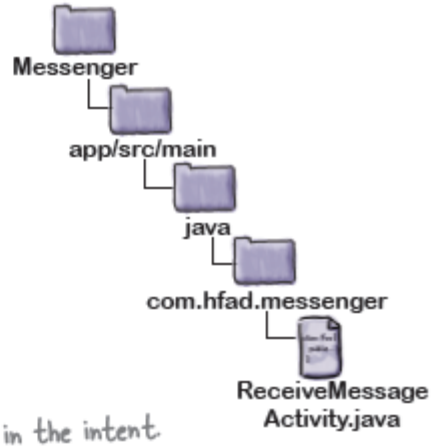
```
package com.hfad.messenger;

import android.os.Bundle;
import android.app.Activity;
import android.content.Intent;
import android.widget.TextView;

public class ReceiveMessageActivity extends Activity {

    public static final String EXTRA_MESSAGE = "message";

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

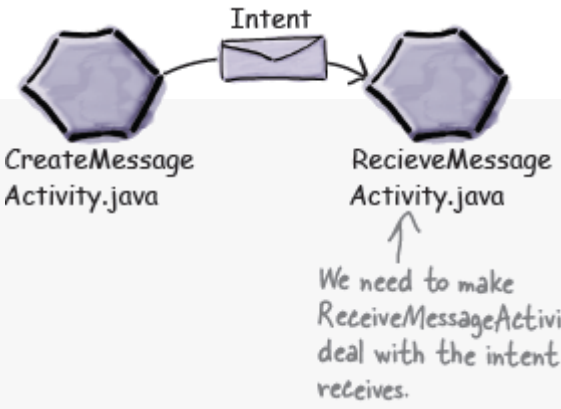
We need to import the Intent and TextView classes

This is the name of the extra value we're passing in the intent.

Get the intent, and get the message from it using getStringExtra().

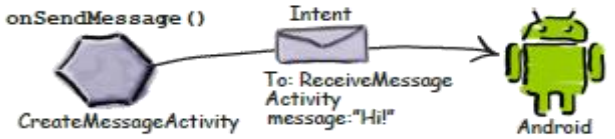
Add the text to the message text view.



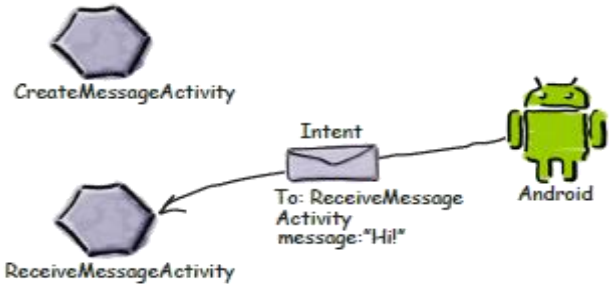


What happens when user clicks the Send Message button

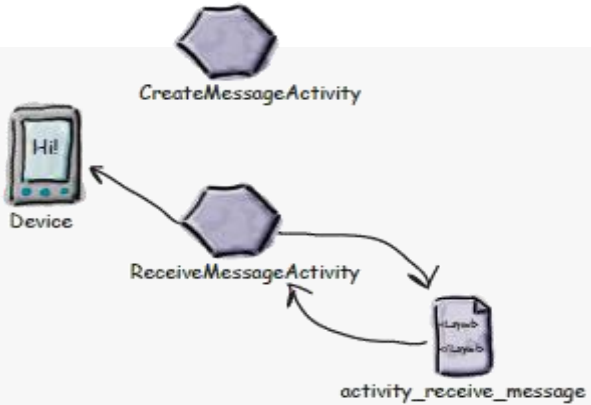
1 **When the user clicks on the button, the onSendMessage() method is called.**
Code within the `onSendMessage()` method creates an intent to start activity `ReceiveMessageActivity`, adds a message to the intent, and passes it to Android with an instruction to start the activity.



2 **Android checks that the intent is OK, and then tells ReceiveMessageActivity to start.**

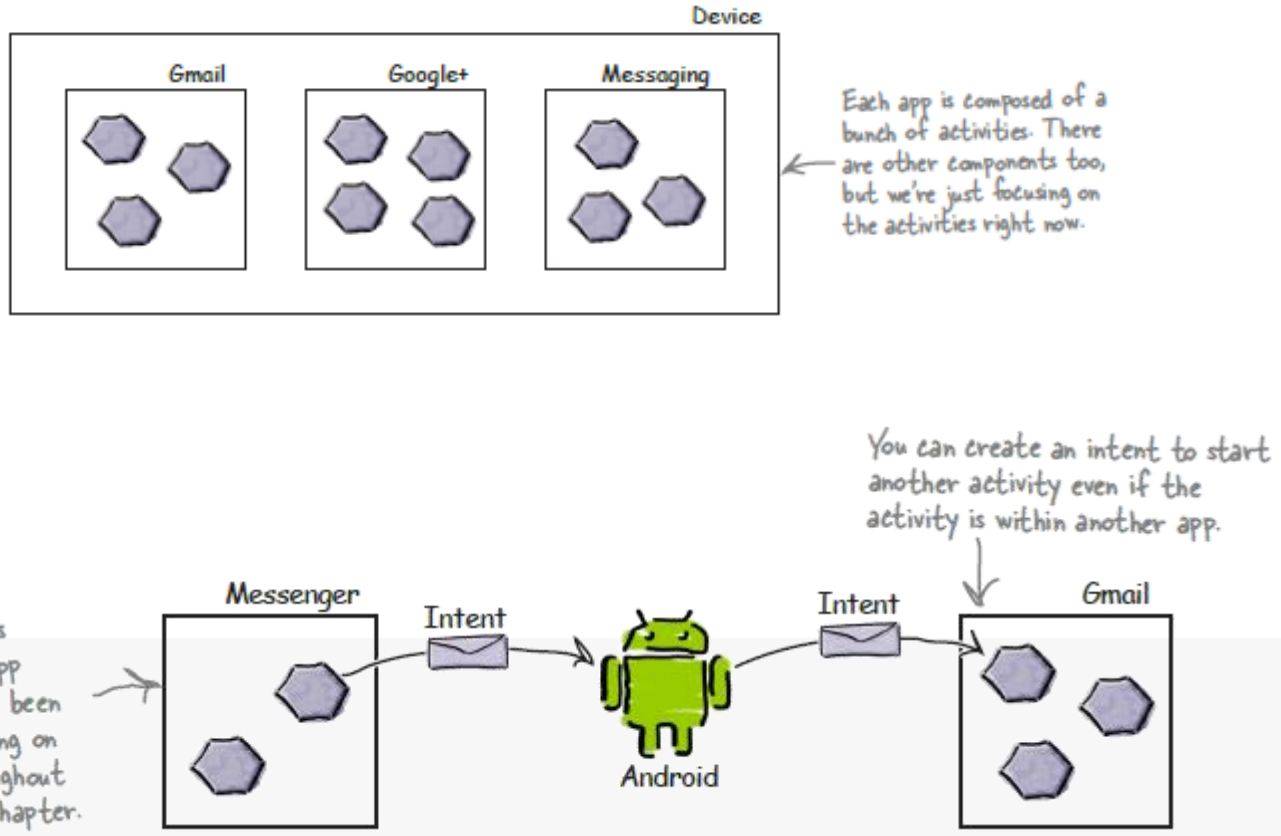


3 **When ReceiveMessageActivity starts, it specifies that it uses layout *activity_receive_message.xml*, and this gets displayed on the device.**
The activity updates the layout so that it displays the extra text included in the intent.



How Android Apps Work

- Intents can start activities in other apps



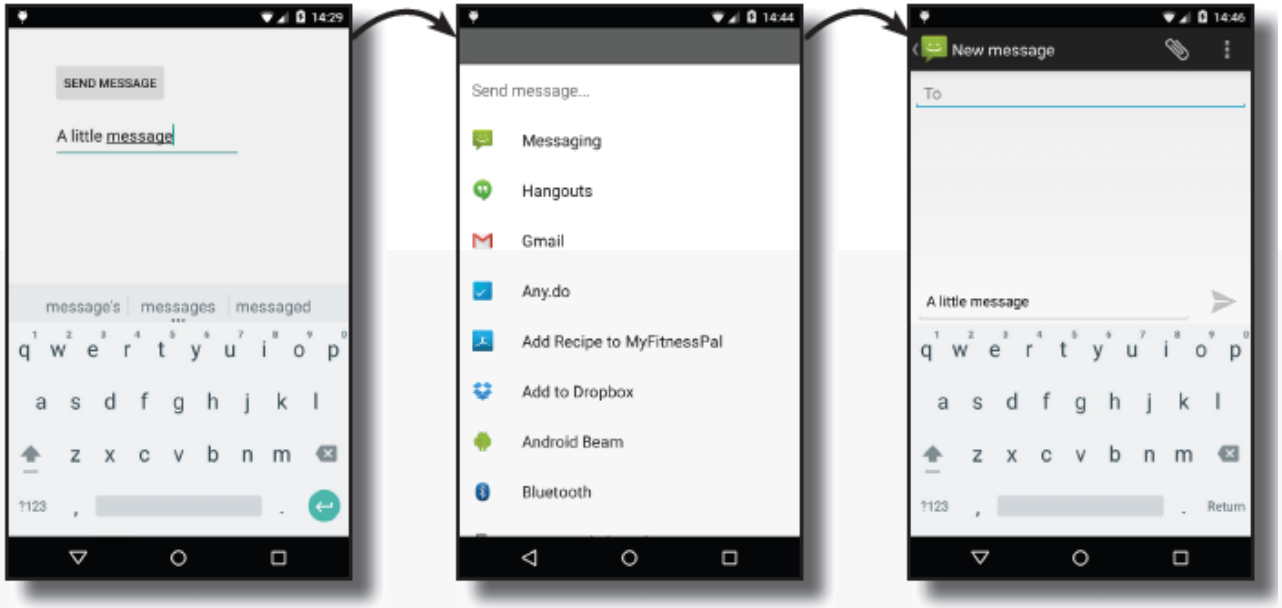
What you are going to do

1. **Create an intent that specifies an action.**

   The intent will tell Android you want to use an activity that can send a message. The intent will include the text of the message.

2. **Allow the user to choose which app to use.**

   The chances are there'll be more than one on the device capable of sending messages, so the user will need to pick one. We want the user to be able to choose one every time they click on the Send Message button.



Specify Action

- **Create an intent that specifies an action**

- Create an intent

```
Intent intent = new Intent(Intent.ACTION_SEND);
```

- Adding extra information

```
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, messageText);
```

These attributes relate to IntentACTION_SEND. They're not relevant for all actions.

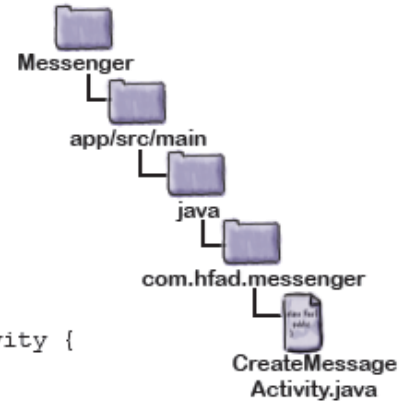
- You can make extra calls to the putExtra() method if there's additional information you want to add. As an example, if you want to specify the subject of the message, you can also use

```
intent.putExtra(Intent.EXTRA_SUBJECT, subject);
```

If subject isn't relevant to a particular app, it will just ignore this information. Any apps that know how to use it will do so.

Specify Action

- **Change the intent that specifies an action**



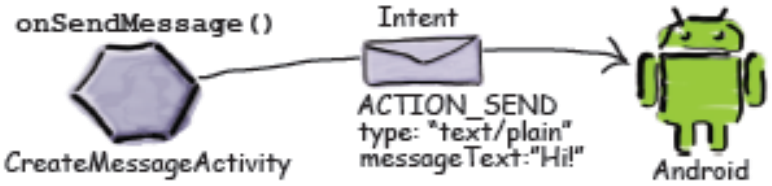

```
package com.hfad.messenger;

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

public class CreateMessageActivity extends Activity {

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

    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        EditText messageView = (EditText)findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(this, ReceiveMessageActivity.class);
        intent.putExtra(ReceiveMessageActivity.EXTRA_MESSAGE, messageText);
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, messageText);
        startActivity(intent);
    }
}
```

Remove these two lines.

Instead of creating an intent that's explicitly for ReceiveMessageActivity, we're creating an intent that uses a send action.

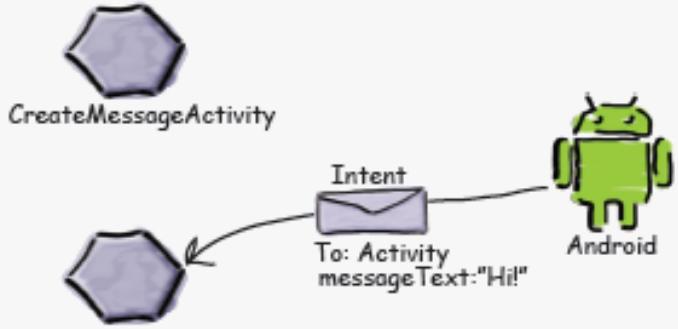
What happens when the code runs

1. **When the onSendMessage() method is called, an intent gets created. The startActivity() method passes the intent to Android.**
The intent specifies an action of `ACTION_SEND`, and a MIME type of text/plain.



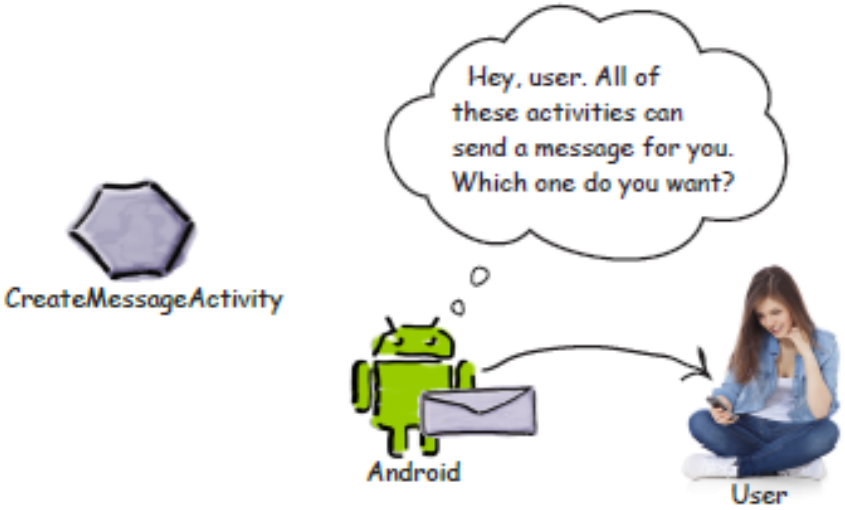
2. **Android sees that the intent can only be passed to activities able to handle ACTION_SEND and text/plain data. Android checks all the activities, looking for ones that are able to receive the intent.**
If no actions are able to handle the intent, an `ActivityNotFoundException` is thrown.

3. **If just one activity is able to receive the intent, Android tells the activity to start and passes it the intent.**

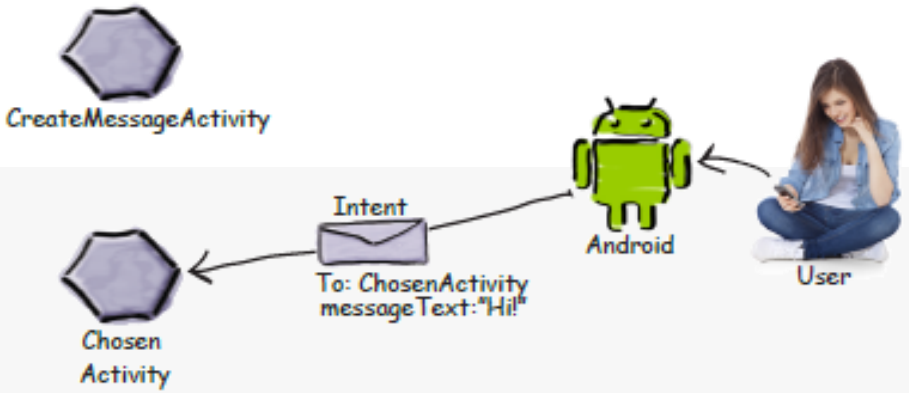


What happens when the code runs (cont.)

4 **If more than one activity is able to receive the intent, Android displays an activity chooser dialog and asks the user which one to use.**



5 **When the user chooses the activity she wants to use, Android tells the activity to start and passes it the intent.** The activity displays the extra text contained in the intent in the body of a new message.



The intent filter tells Android which activities can handle which actions

- When Android is given an intent, it has to figure out which activity, or activities, are able to handle it. This process is known as **intent resolution.**
- When you use an explicit intent,the following code explicitly tells Android to start ReceiveMessageActivity:

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

- When you use an implicit intent

```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
        <data android:mimeType="image/*"/>
    </intent-filter>
</activity>
```

This tells Android the activity can handle ACTION_SEND.

The intent filter must include a category of DEFAULT or it won't be able to receive implicit intents.

These are the types of data the activity can handle.

How Android Uses the Intent Filter

- Android first considers intent filters that include a category of android.intent.category.DEFAULT:

```
<intent-filter>
    <category android:name="android.intent.category.DEFAULT"/>
    ...
</intent-filter>
```

- Similarly, if the intent MIME type is set to “text/plain” using

```
intent.setType("text/plain");
```

- Android will only consider activities that can accommodate this type of data:

```
<intent-filter>
    <data android:mimeType="text/plain"/>
    ...
</intent-filter>
```

Here's the intent

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.setType("text/plain");
intent.putExtra(Intent.EXTRA_TEXT, "Hello");
```

BE the Intent

Your job is to play like you're the intent on the right and say which of the activities described below are compatible with your action and data. Say why, or why not, for each one.

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="*/*"/>
    </intent-filter>
</activity>
```

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.MAIN"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

```
<activity android:name="SendActivity">
    <intent-filter>
        <action android:name="android.intent.action.SENDTO"/>
        <category android:name="android.intent.category.MAIN"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

You need to run your app on a REAL device

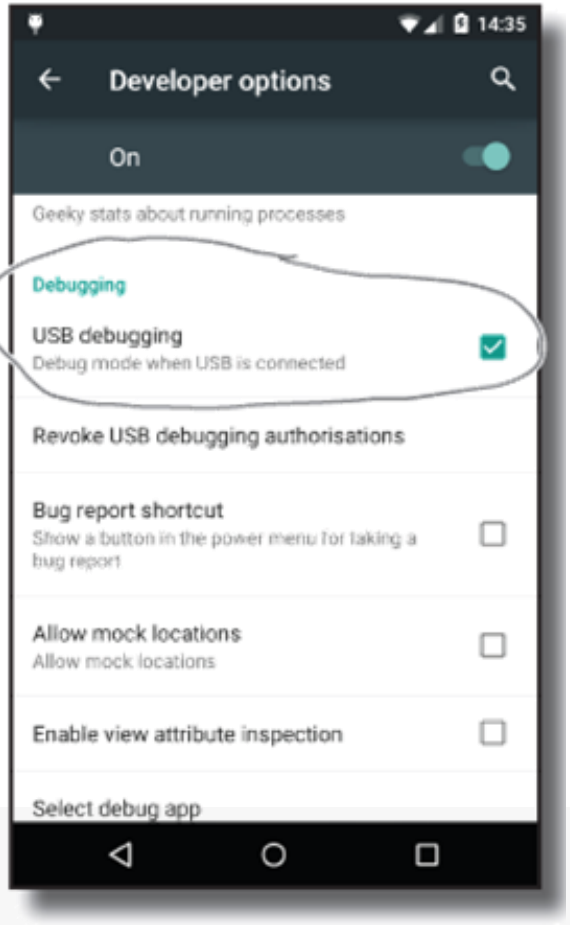
1. Enable USB debugging on your device

On your device, open "Developer options" (in Android 4.0 onward, this is hidden by default). To enable it, go to Settings → About Phone and tap the build number seven times. When you return to the previous screen, you should be able to see "Developer options."

Yep, seriously.

Within "Developer options," tick the box to enable USB debugging

You need to enable USB debugging.



2. Set up your system to detect your device

If you're using a Mac, you can skip this step.

If you're using Windows, you need to install a USB driver. You can find the latest instructions here:

http://developer.android.com/tools/extras/oem-usb.html

If you're using Ubuntu Linux, you need to create a `udev` rules file. You can find the latest instructions on how to do this here:

http://developer.android.com/tools/device.html#setting-up

3. Plug your device into your computer with a USB cable

Your device may ask you if you want to accept an RSA key that allows USB debugging with your computer. If it does, you can tick the "Always allow from this computer" option and choose OK to enable this.

You'll get this message if your device is running Android 4.2.2 or higher.

Running your app on a real device (continued)

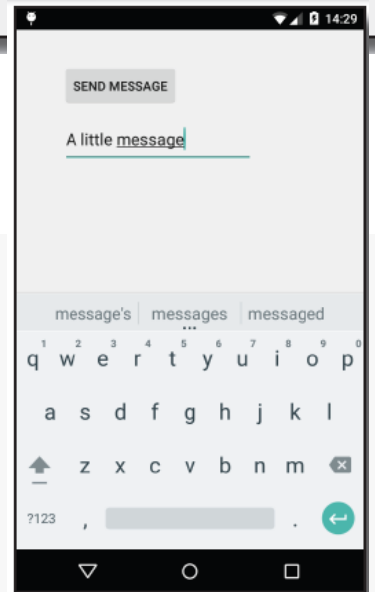
4. Run your app in Android Studio as normal

Android Studio will install the app on your device and launch it. You may be asked to choose which device you want to run your app on. If so, select your device from the list available and click OK.

And here's the app running on the physical device

You should find that your app looks about the same as when you ran it through the emulator. You'll probably also find that your app installs and runs quicker too.

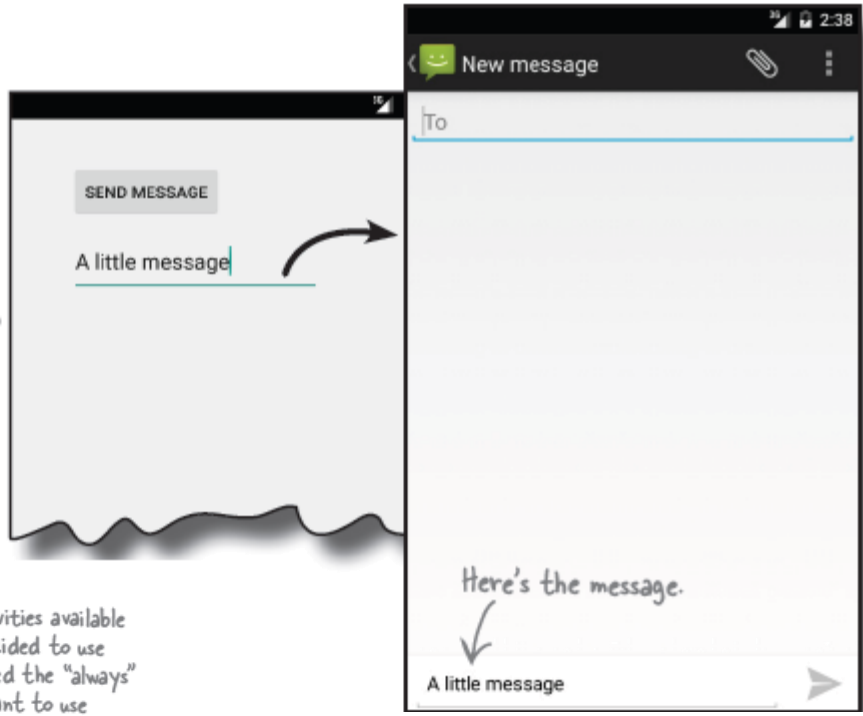
Now that you know how to run the apps you create on your own device, you're all set to test the latest changes to your app.



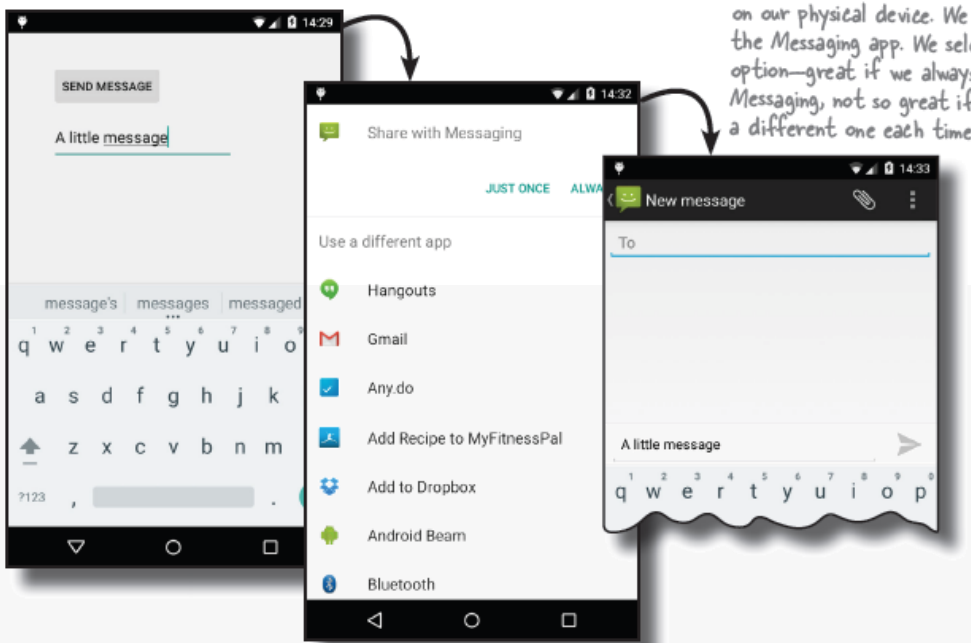
Test Drive

If you have one activity

We only have one activity available on the emulator that can send messages with text data, so when we click on the Send Message button, Android starts the activity.



We have lots of suitable activities available on our physical device. We decided to use the Messaging app. We selected the "always" option—great if we always want to use Messaging, not so great if we want to use a different one each time.



If you have more than one activity

What if you ALWAYS want your users to choose an activity?

- createChooser() allows you to specify a title for the chooser dialog, and doesn't give the user the option of selecting an activity to use by default. It also lets the user know if there are no matching activities by displaying a message.
- Intent.createChooser() displays a chooser dialog

```
Intent chosenIntent = Intent.createChooser(intent, "Send message...");
```

This is the intent you created earlier.

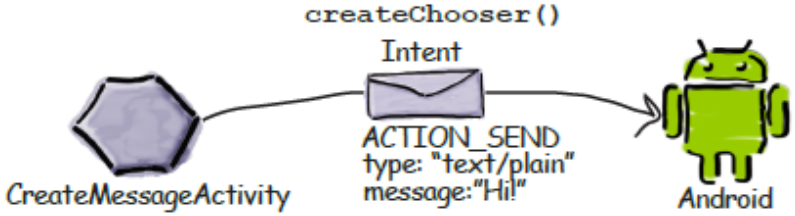
You can pass in a title for the chooser that gets displayed at the top of the screen.

- To start activity:

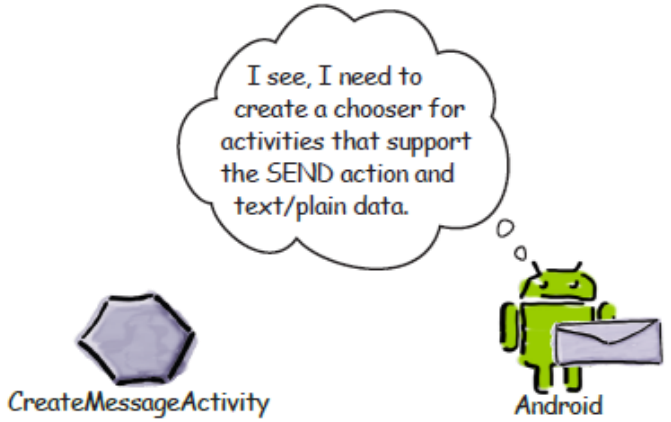
```
startActivity(chosenIntent);
```

What happens when you call createChooser()

1 **The createChooser() method gets called.**
The method includes an intent that specifies the action and MIME type that's required.



2 **Android checks which activities are able to receive the intent by looking at their intent filters.**
It matches on the actions, type of data, and categories they can support.

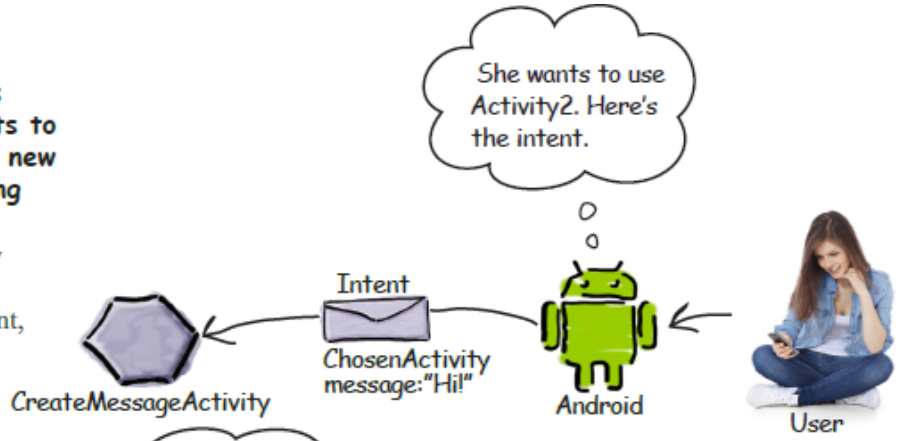


3 **If more than one activity is able to receive the intent, Android displays an activity chooser dialog and asks the user which one to use.**
This time it doesn't give the user the option of always using a particular activity, and it displays "Send message..." in the title.

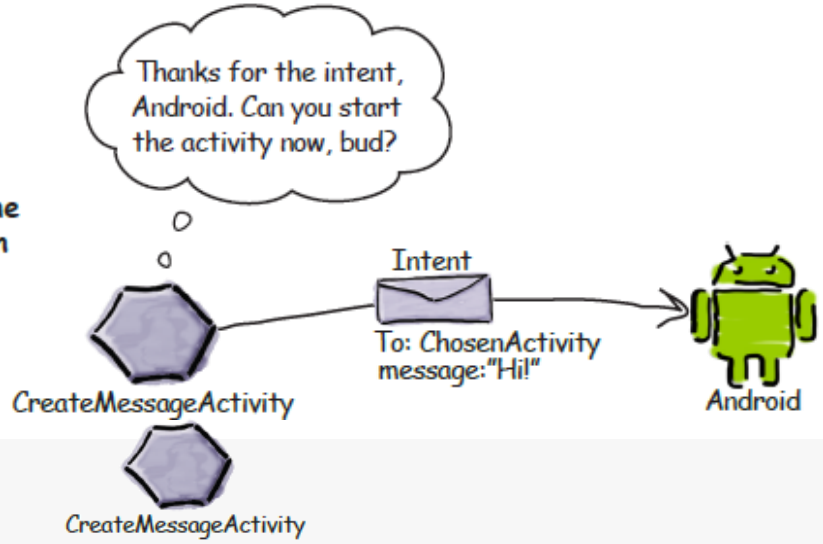
If no activities are found, Android still displays the chooser but shows a message to the user telling her there are no apps that can perform the action.

What happens when you call createChooser() (Cont.)

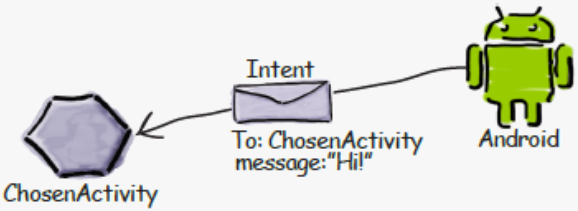
4 **When the user chooses which activity she wants to use, Android returns a new explicit intent describing the chosen activity.**
The new intent includes any extra information that was included in the original intent, such as any extra text.



5 **The activity asks Android to start the activity specified in the intent.**



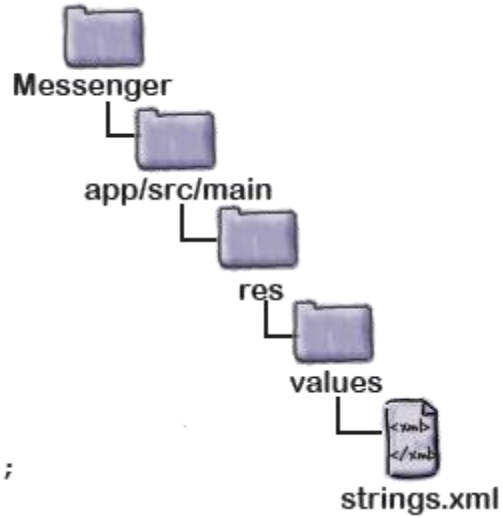
6 **Android asks the activity specified by the intent to start, and then passes it the intent.**



Change the code to create a chooser

- Update strings.xml...

```
...
<string name="chooser">Send message...</string>
...
```



- and update the onSendMessage() method

```
...
    //Call onSendMessage() when the button is clicked
    public void onSendMessage(View view) {
        EditText messageView = (EditText)findViewById(R.id.message);
        String messageText = messageView.getText().toString();
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_TEXT, messageText);
        String chooserTitle = getString(R.string.chooser);
        Intent chosenIntent = Intent.createChooser(intent, chooserTitle);
        startActivity(intent);
        startActivity(chosenIntent);
    }
...
```

Get the chooser title.

Display the chooser dialog.

Start the activity that the user selected.

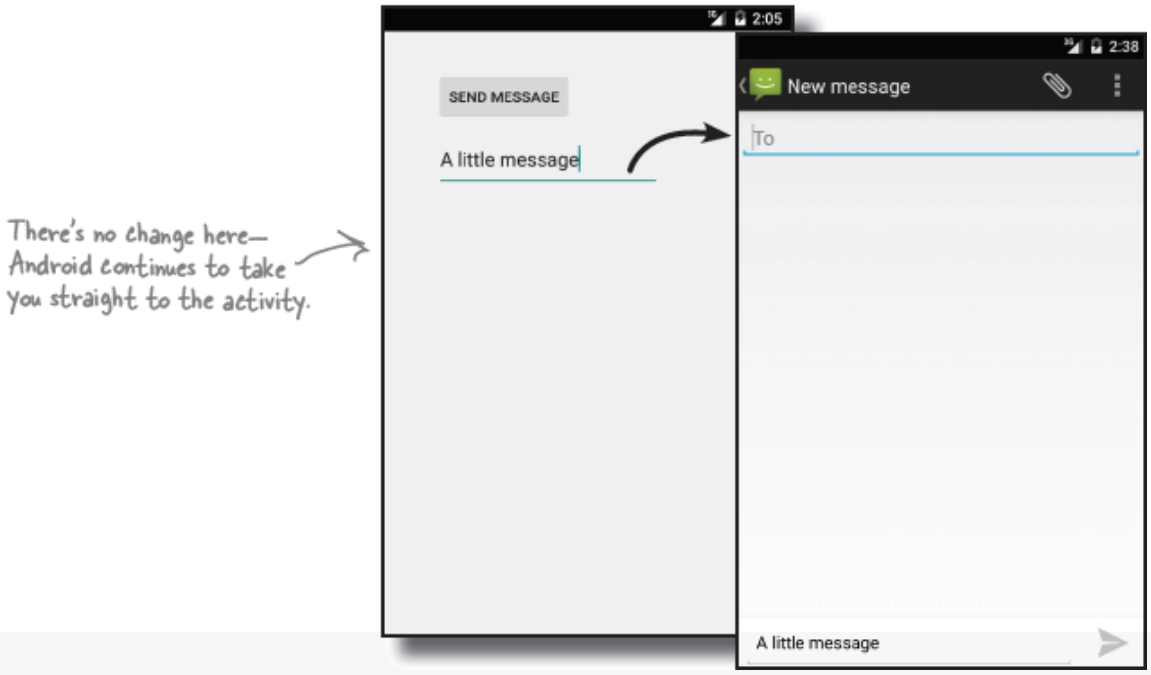
- The getString() method is used to get the value of a string resource. It takes one parameter, the ID of the resource (in our case, this is R.string.chooser):

```
getString(R.string.chooser);
```

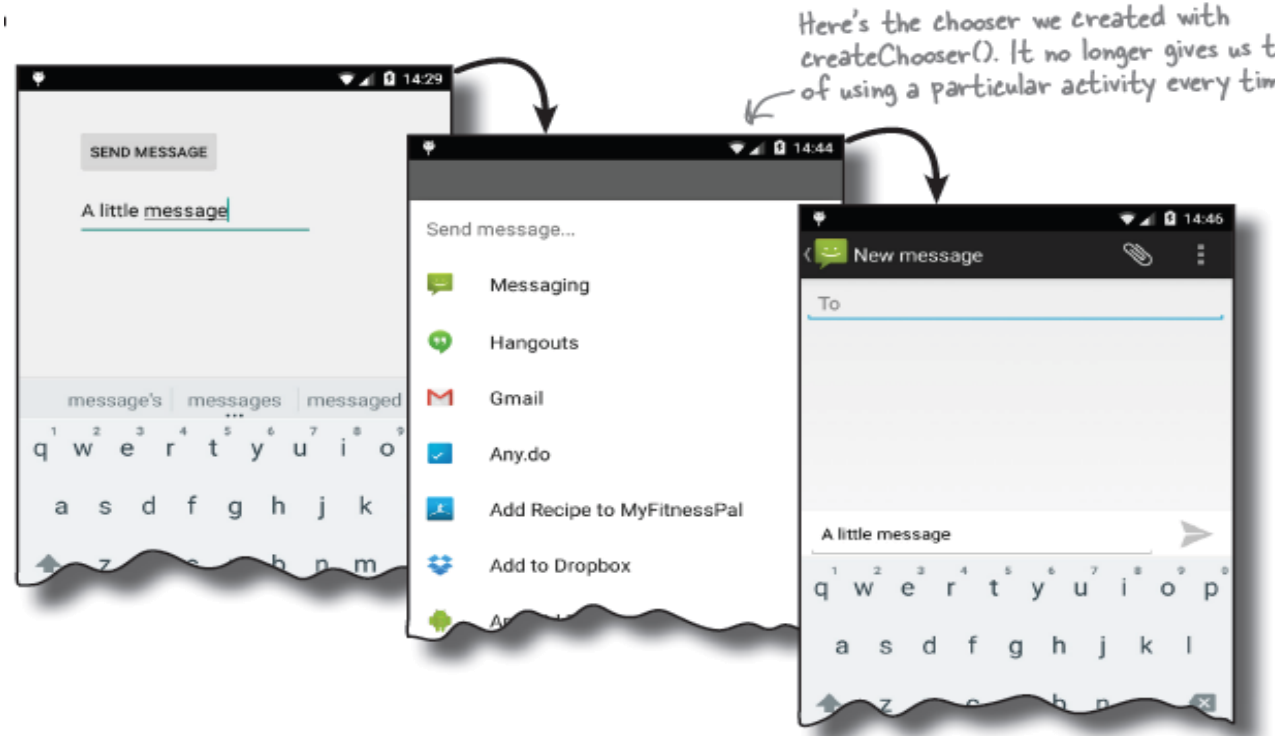
If you look in R.java, you'll find chooser in the inner class called string.

Test drive the App

If you have one activity

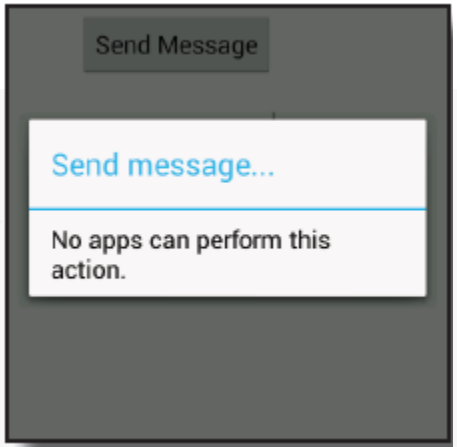


Test drive the App



If you have more than one activity

If you have NO matching activities



If you want to replicate this for yourself, try running the app in the emulator, and disable the Messaging app that's on there.

Summary

BULLET POINTS

- A task is two or more activities chained together.
- The `<EditText>` element defines an editable text field for entering text. It inherits from the Android `View` class.
- You can add a new activity in Android Studio by choosing File → New... → Activity.
- Each activity you create must have an entry in *AndroidManifest.xml*.
- An **intent** is a type of message that Android components use to communicate with one another.
- An explicit intent explicitly specifies the component the intent is targeted at. You create an explicit intent using `Intent intent = new Intent(this, Target.class);`
- To start an activity, call `startActivity(intent)`. If no activities are found, it throws an `ActivityNotFoundException`.
- Use the `putExtra()` method to add extra information to an intent.
- Use the `getIntent()` method to retrieve the intent that started the activity.
- Use the `get*Extra()` methods to retrieve extra information associated with the intent. `getStringExtra()` retrieves a String, `getIntExtra()` retrieves an int, and so on.
- An activity action describes a standard operational action an activity can perform. To send a message, use `Intent.ACTION_SEND`.
- To create an implicit intent that specifies an action, use `Intent intent = new Intent(action);`
- To describe the type of data in the intent, use the `setType()` method.
- Android resolves intents based on the named component, action, type of data, and categories specified in the intent. It compares the contents of the intent with the intent filters in each app's *AndroidManifest.xml*. An activity must have a category of `DEFAULT` if it is to receive an implicit intent.
- The `createChooser()` method allows you to override the default Android activity chooser dialog. It allows you to specify a title for the dialog, and doesn't give the user the option of setting a default activity. If no activities can receive the intent it is passed, it displays a message. The `createChooser()` method returns an `Intent`.
- You retrieve the value of a string resource using `getString(R.string.stringname);`

References

- Head First Android Development. 2nd Edition. A Brain friendy guide. Dawn Griffiths and David Griffiths.O'reilly. ISBN:978-1-449-36218-8. Chapter 3