3
Actividades múltiples y operaciones

Aplicaciones con más de una actividad

Una actividad es una cosa que el usuario puede hacer. Si se encadenan varias actividades para hacer algo más complejo, se le llama tarea.

Se construirá una aplicación que contendrá 2 actividades. La primera permitirá enviar un mensaje, la segunda recibirá el mensaje.

Aplicaciones con más de una actividad

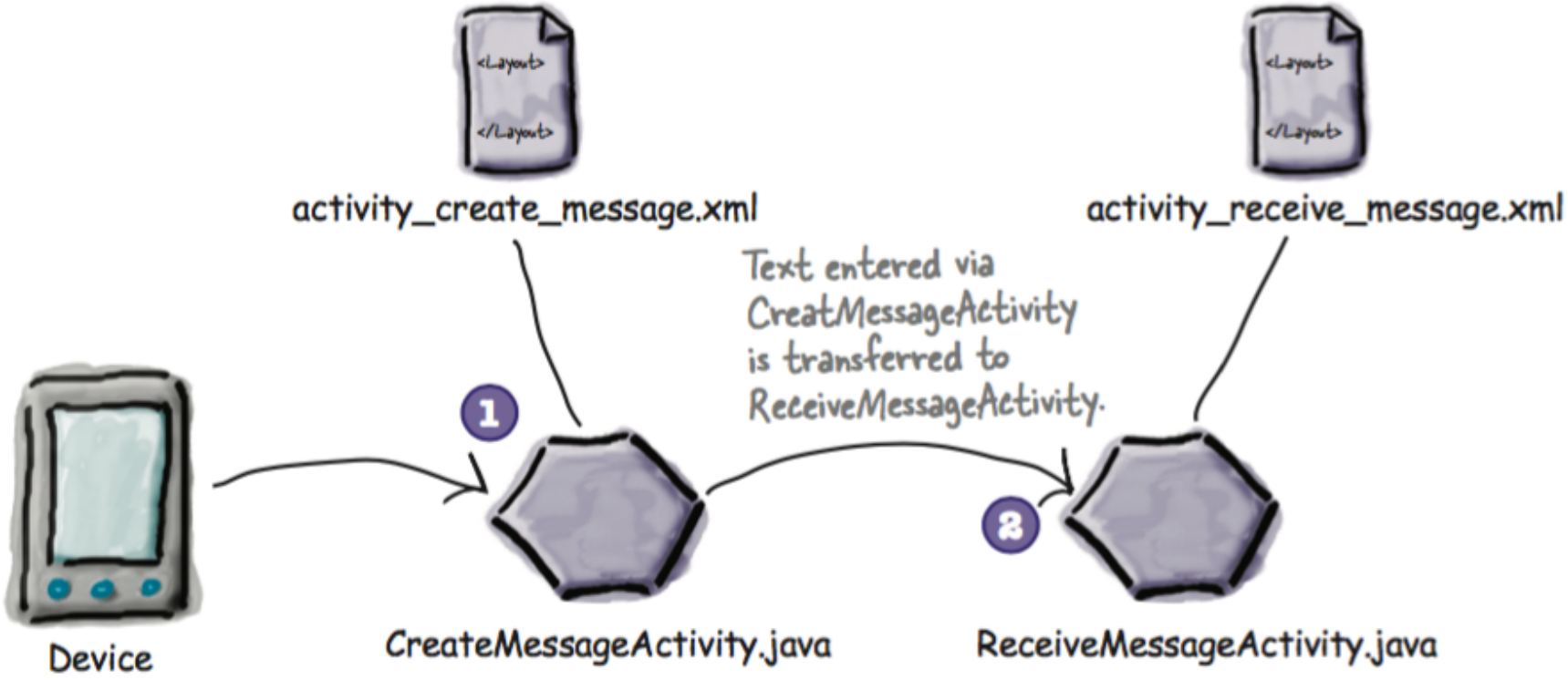
Here are the steps

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.

Estructura de la aplicación

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*.



Creación del proyecto

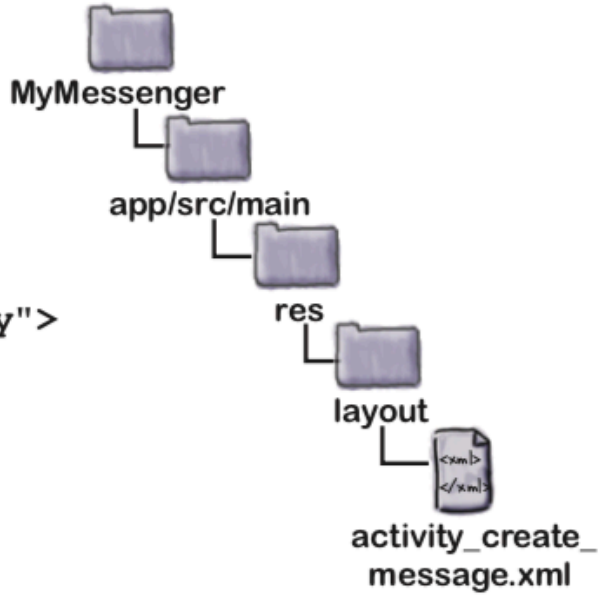
Get started: create the project

You create a project for the app in exactly the same way you did in previous chapters. Create a new Android Studio project for an application named "My Messenger" with a company domain of "hfad.com", making the package name `com.hfad.mymessenger`. The minimum SDK should be API 19 so that it will work on most devices. You'll need an empty activity named "CreateMessageActivity" with a layout named "activity_create_message" so that your code matches ours. **Make sure that you untick the Backwards Compatibility (AppCompat) option when you create the activity.**

Actualización del layout


```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    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:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.mymessenger.CreateMessageActivity">
```

We're using a linear layout with a vertical orientation.



Actualización del layout

This creates an editable text field.

```
<EditText
    android:id="@+id/message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:hint="@string/hint"
    android:ems="10" />
```

The hint attribute gives the user a hint of what text they should type into the text field. We need to add it as a String resource.

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

```
<Button
    android:id="@+id/send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:onClick="onSendMessage"
    android:text="@string/send" />

</LinearLayout>
```

This is a String resource we need to create.

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

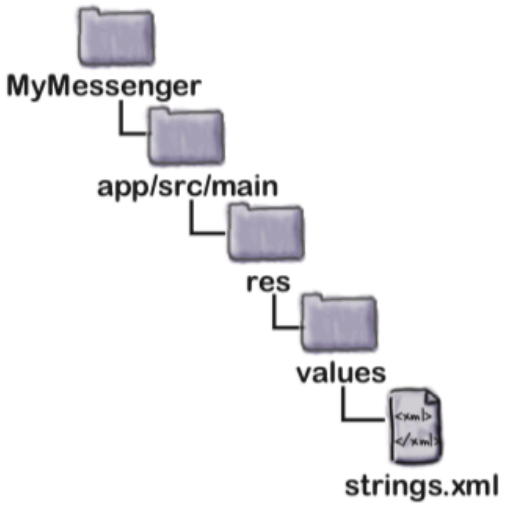
El elemento <EditText> define un campo de

Actualización de strings.xml …

```
<resources>
    ...
    <string name="send">Send Message</string>
    <string name="hint">Enter a message</string>
</resources>
```

The text "Send Message" will appear on the button.

The text "Enter a message" will appear as a hint in the text field if it's empty.



y el layout …

```
android:onClick="onSendMessage"
```

Actualizar el código Java …

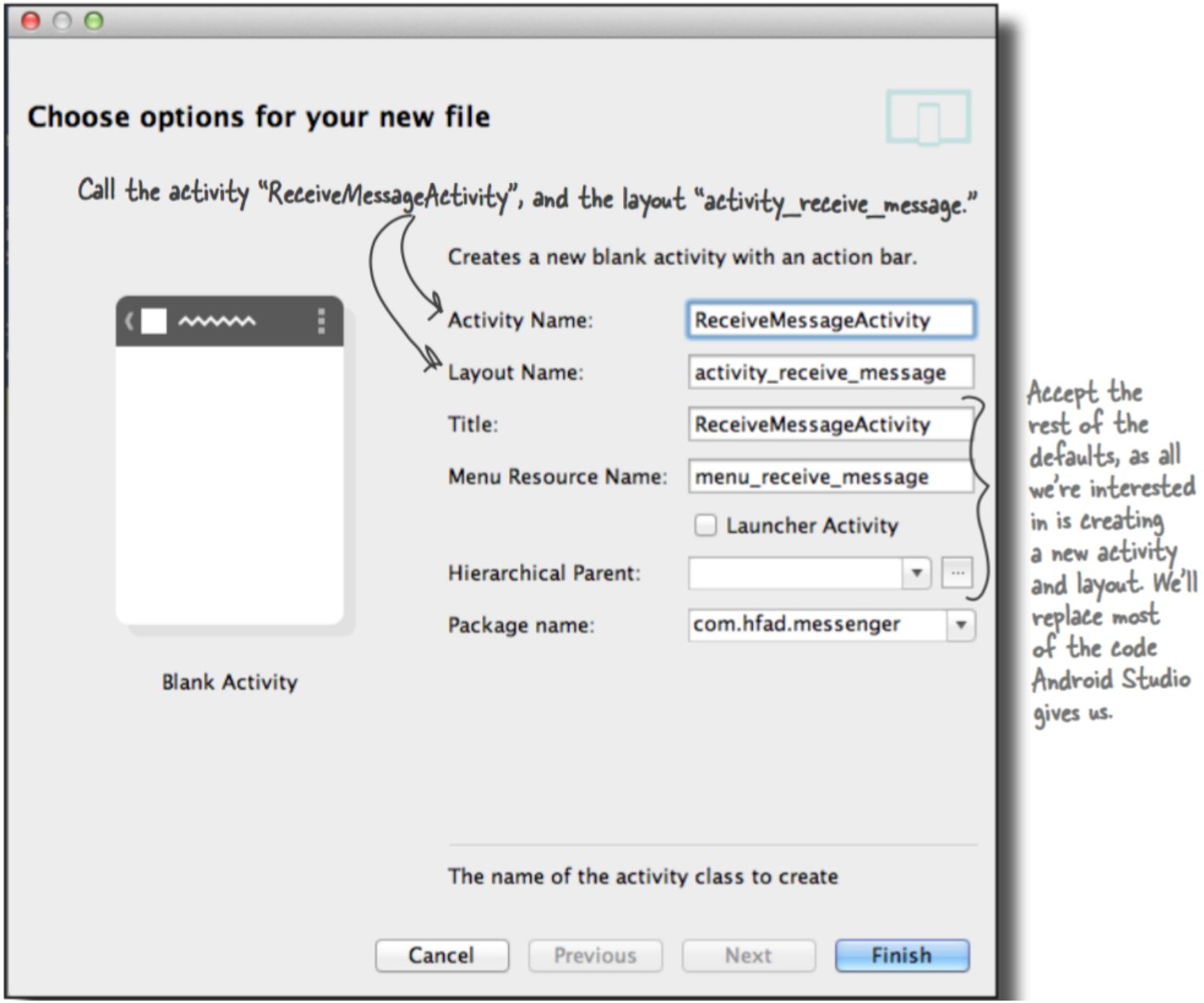
```
package com.hfad.messenger;

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.

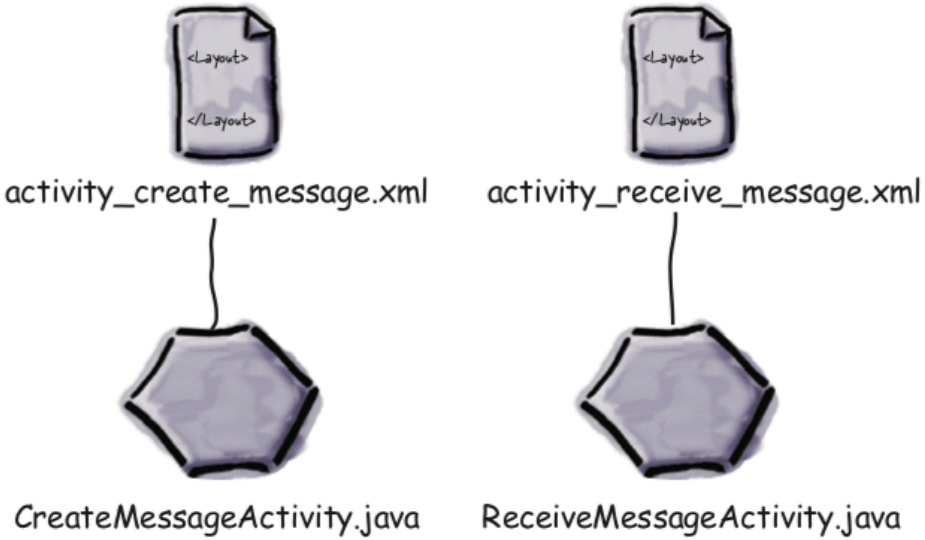
The onCreate() method gets called when the activity is created.

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.

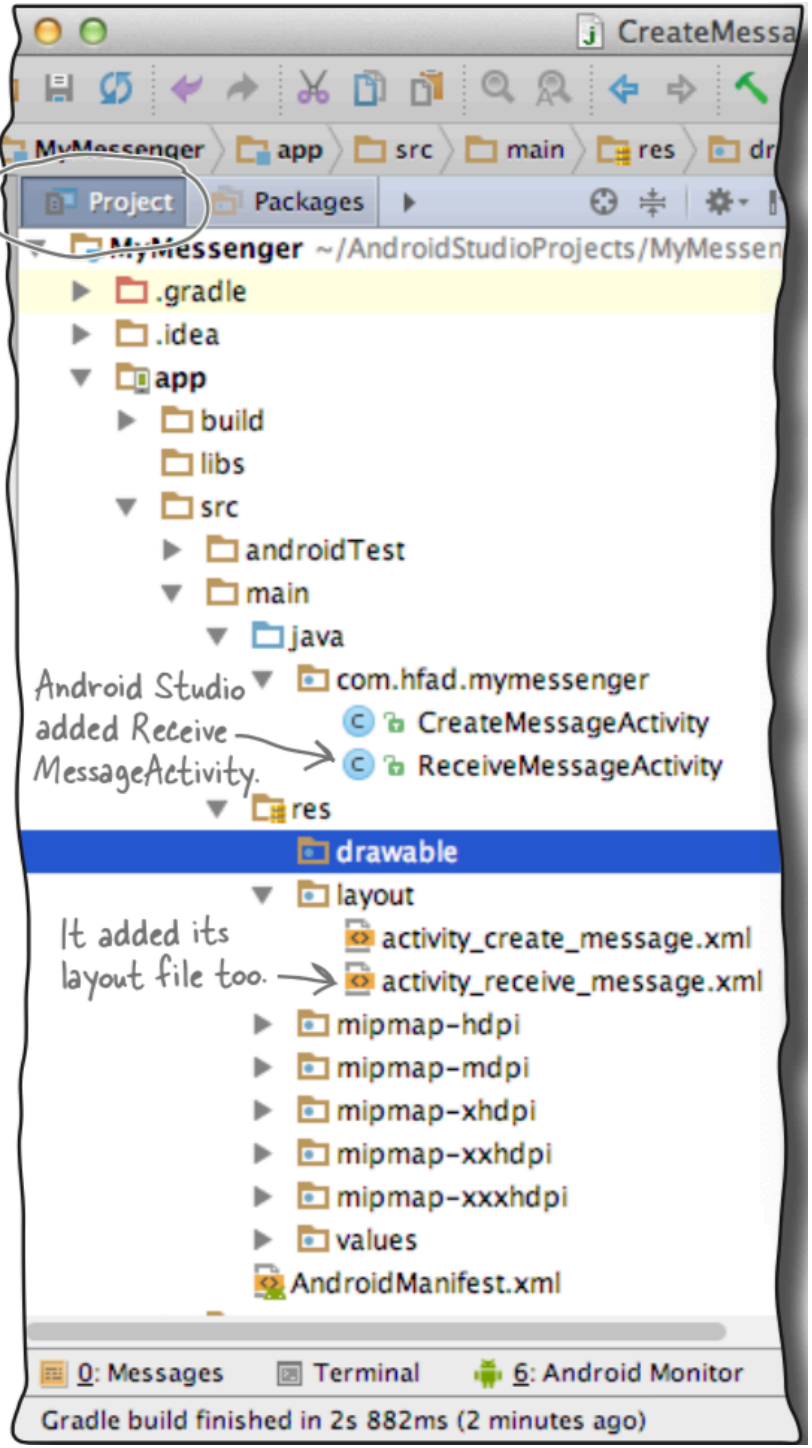
Crear la segunda actividad y su layout



Que acaba de pasar



Your explorer window may look different than ours because we've switched to the Project view.



Android Studio added ReceiveMessageActivity.

It added its layout file too.

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

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

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

        <activity android:name=".ReceiveMessageActivity"></activity>

    </application>

</manifest>
```

This is the package name we specified.

Android Studio gave our app default icons.

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, ReceiveMessageActivity. Android Studio added this code when we added the second activity.

Watch it!

If you develop Android apps without an IDE, you'll need to create this file manually.

El archivo Android manifest

Cada actividad debe declararse ...

Watch it!

The second activity was automatically declared because we added it using the Android Studio wizard.

If you add extra activities manually, you'll need to edit AndroidManifest.xml *yourself. If you use another IDE, it may not be added for you.*


```
<application
    ...
    ...>
    <activity
        android:name=".MyActivityClassName"
        ...
        ...>
    ...
    </activity>
    ...
</application>
```

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

This line is mandatory; just replace MyActivityClassName with the name of your activity.

The activity may have other properties too.

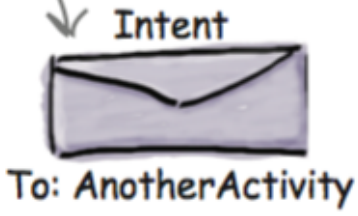
Activity

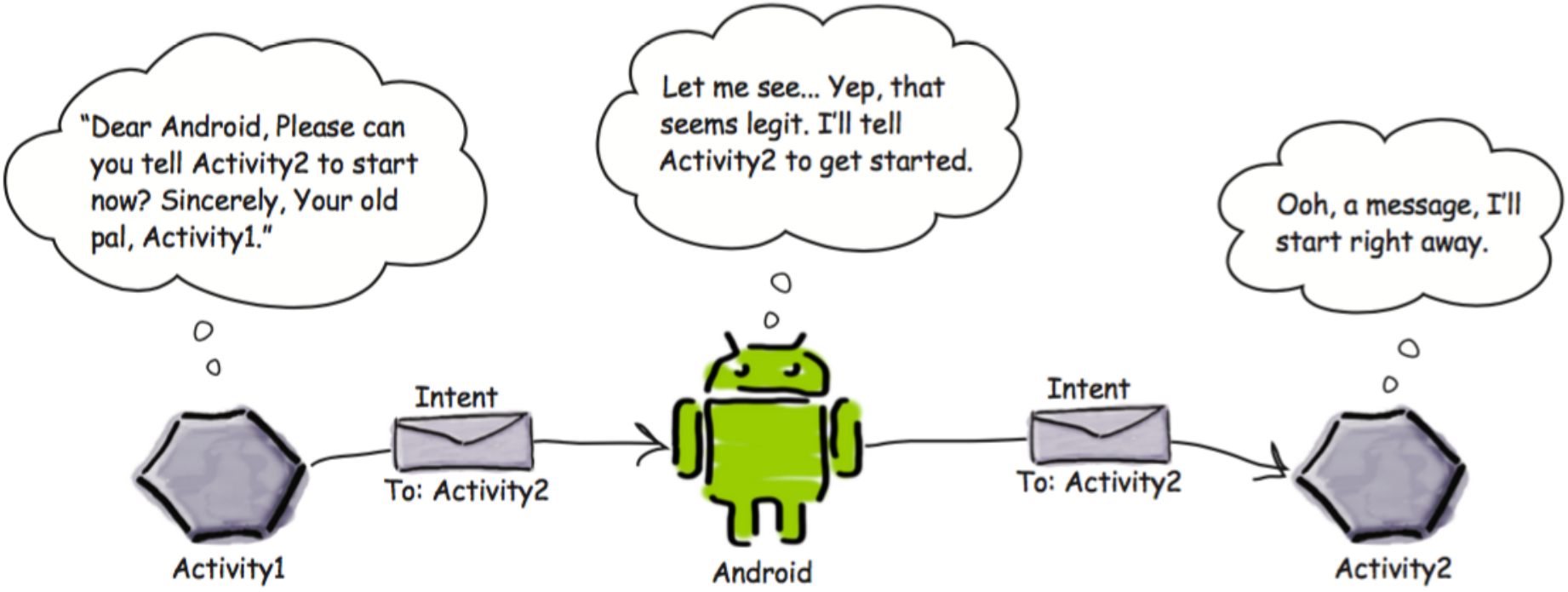
Un intent es un tipo de mensaje

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

startActivity(intent);
```

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

startActivity() starts the activity specified in the intent.



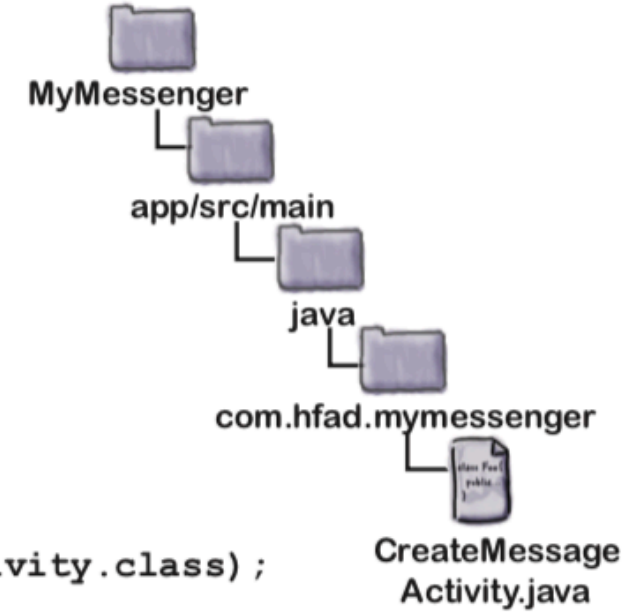


Utilizando un intent para iniciar la segunda actividad

```
package com.hfad.mymessenger;

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);
    }
```

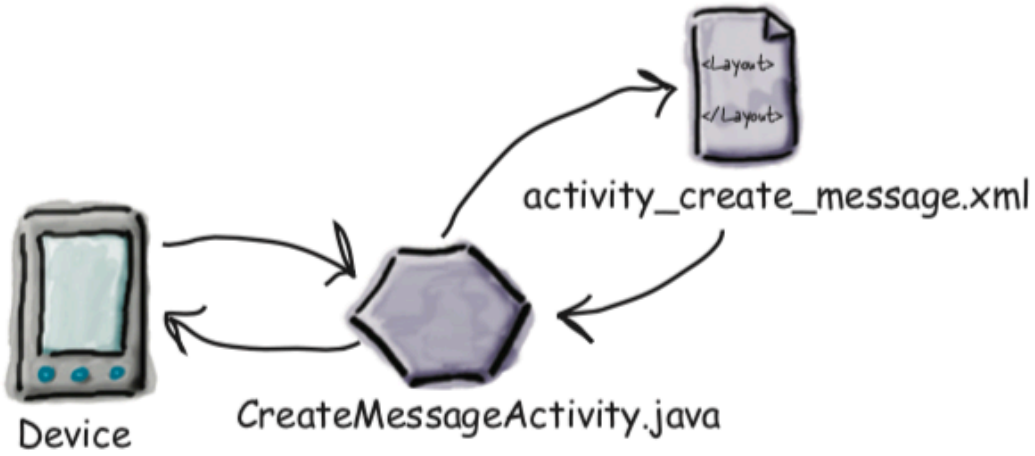
We need to import the Intent class android.content.Intent as we're using it in onSendMessage().

Start activity ReceiveMessageActivity.

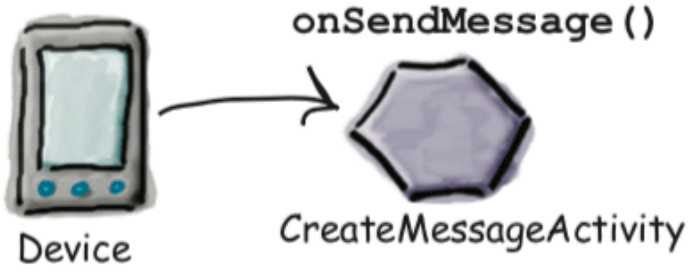


Que pasa al ejecutar la aplicación

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 layout gets displayed in a new window.

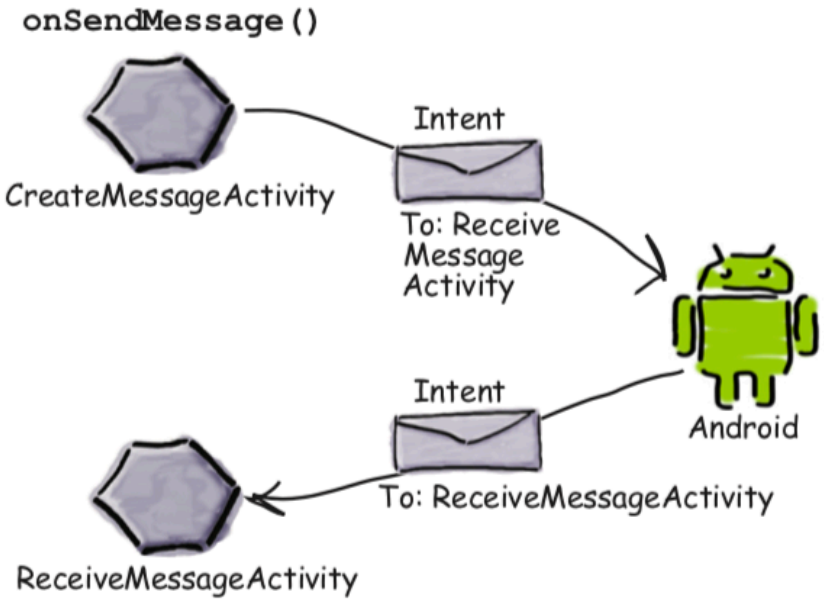


2. **The user types in a message and then clicks on the button.** The `onSendMessage()` method in `CreateMessageActivity` responds to the click.

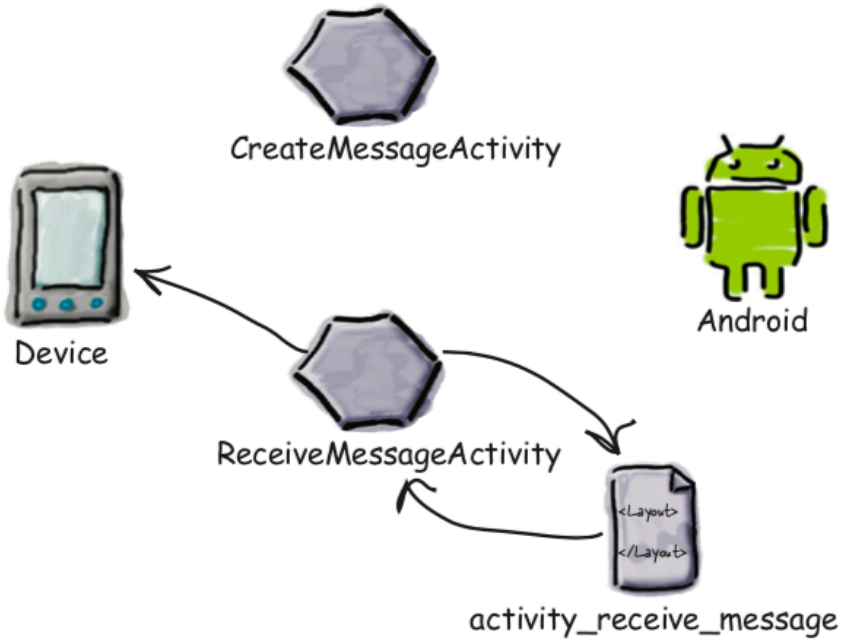


Que pasa al ejecutar la aplicación

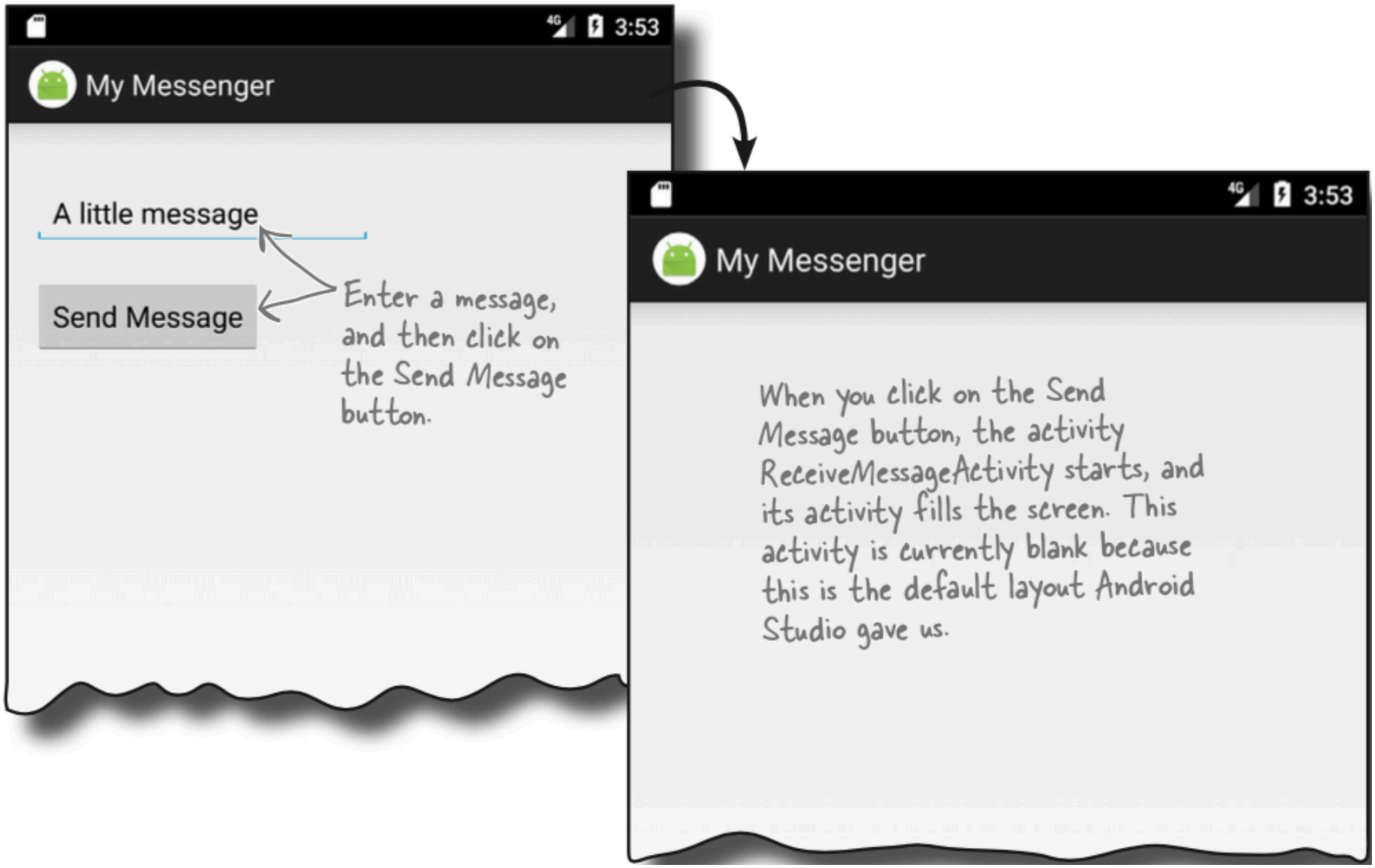
3 **The onSendMessage() method uses an intent to tell Android to start activity ReceiveMessageActivity.**
Android checks that the intent is valid, and then it tells `ReceiveMessageActivity` to start.



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

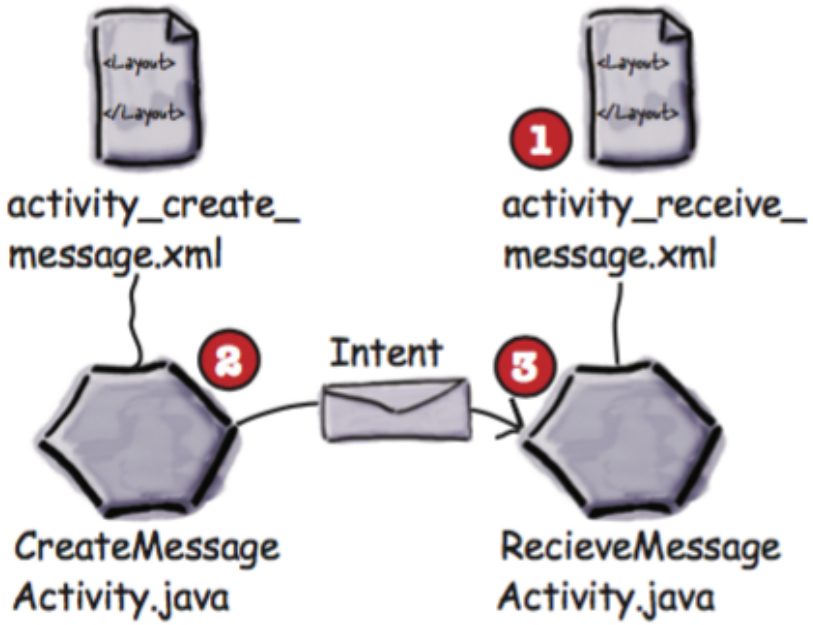


Prueba


3:53
My Messenger
A little message
Send Message
Enter a message, and then click on the Send Message button.
3:53
My Messenger
When you click on the Send Message button, the activity ReceiveMessageActivity starts, and its activity fills the screen. This activity is currently blank because this is the default layout Android Studio gave us.

Pasando el texto a la segunda actividad

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.

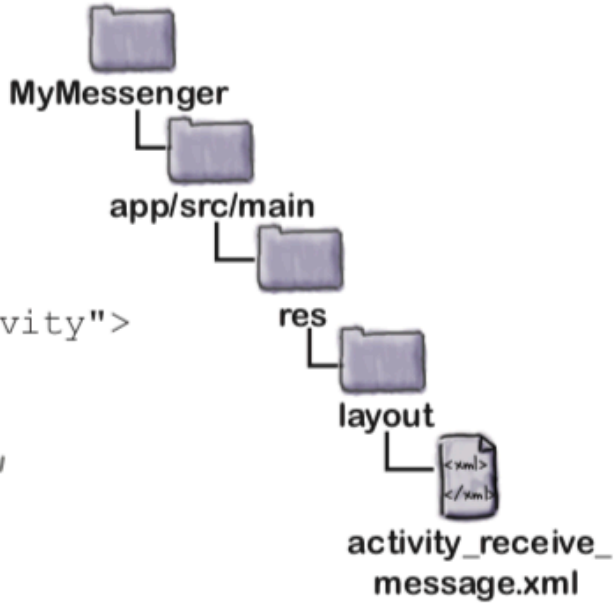


Modificación del layout creado por Android Studio

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    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:padding="16dp"
    android:orientation="vertical"
    tools:context="com.hfad.mymessenger.ReceiveMessageActivity">

    <TextView
        android:id="@+id/message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</LinearLayout>
```

This adds the text view.

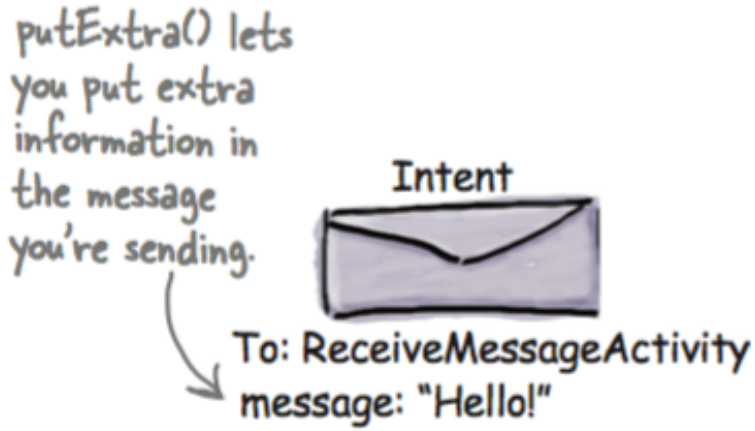
This line gives the text view an ID of "message".



Poniendo información extra en el intent

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

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



Obteniendo la información extra del intent

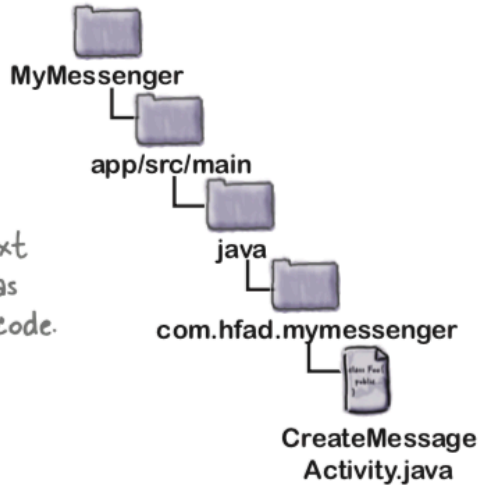

```
Intent intent = getIntent();
String string = intent.getStringExtra("message");
```

Get the intent.

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

```
int intNum = intent.getIntExtra("name", default_value);
```

Actualizando el código de la primera actividad




```
package com.hfad.mymessenger;

import android.app.Activity;
import android.os.Bundle;
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);
    }
}
```

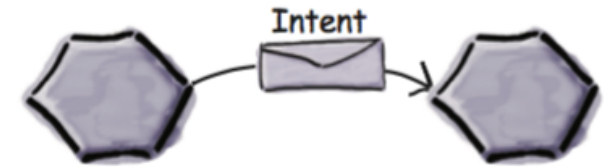
You need to import the EditText class android.widget.EditText as you're using it in your activity code.

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 constant to ReceiveMessageActivity on the next page, so don't worry if Android Studio says it doesn't exist.

Start ReceiveMessageActivity with the intent.

Actualizando el código de la segunda actividad



We need to make ReceiveMessageActivity deal with the intent it receives.

```
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.

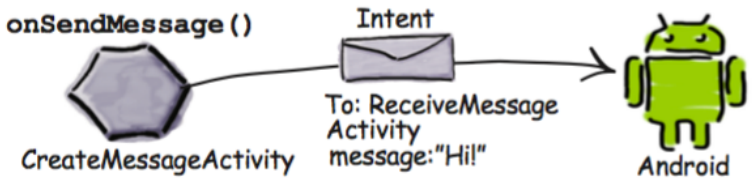
This is the name of the extra value we're passing in the intent.

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

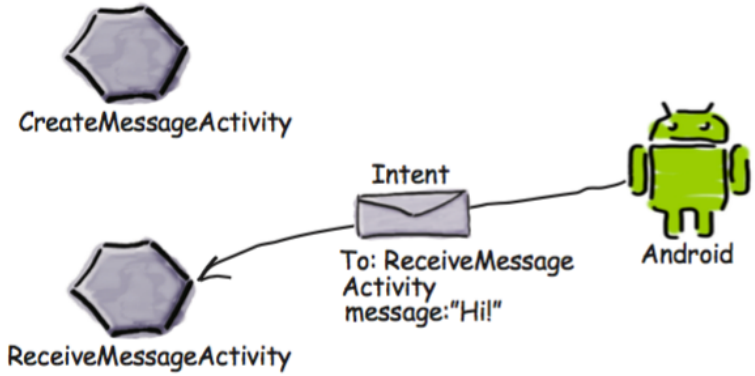
Add the text to the message text view.

Que pasa al presionar el botón Send Message

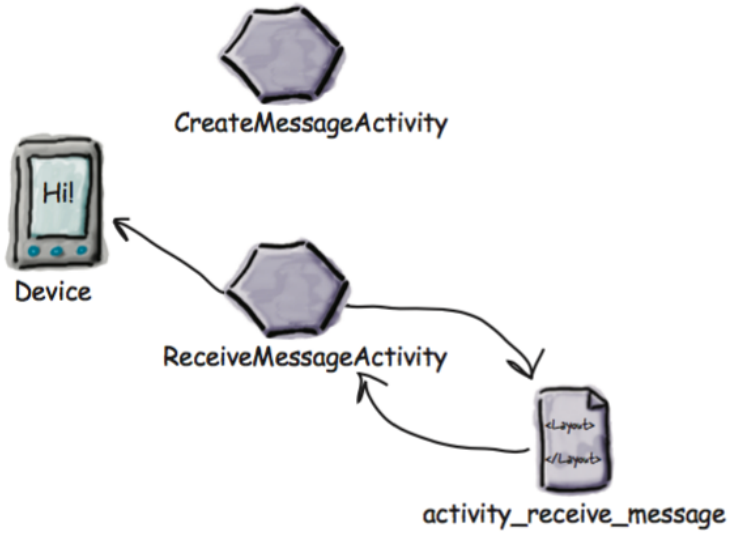
1. **When the user clicks on the button, the onSendMessage() method is called.**
Code within the `onSendMessage()` method creates an intent to start activity `ReceiveMessageActivity`, adds a message to the intent, and passes it to Android with an instruction to start the activity.



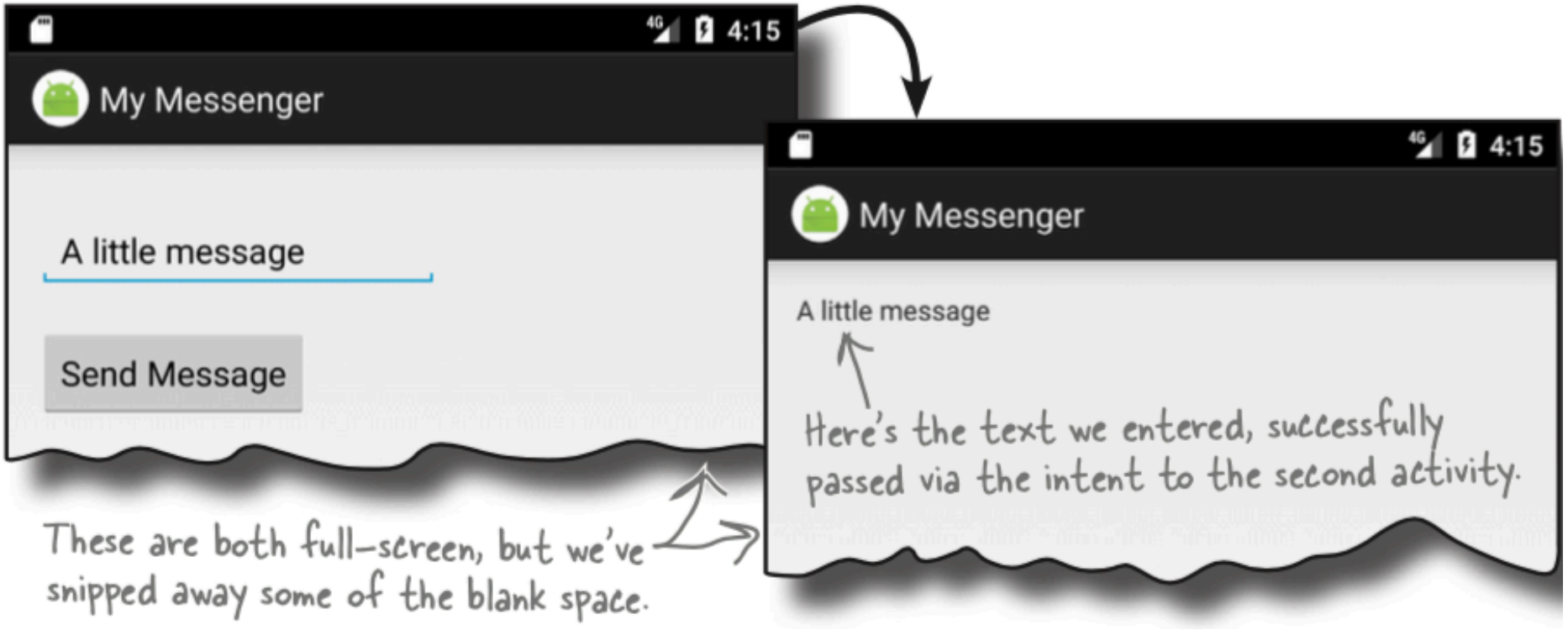
2. **Android checks that the intent is OK, and then tells ReceiveMessageActivity to start.**



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.

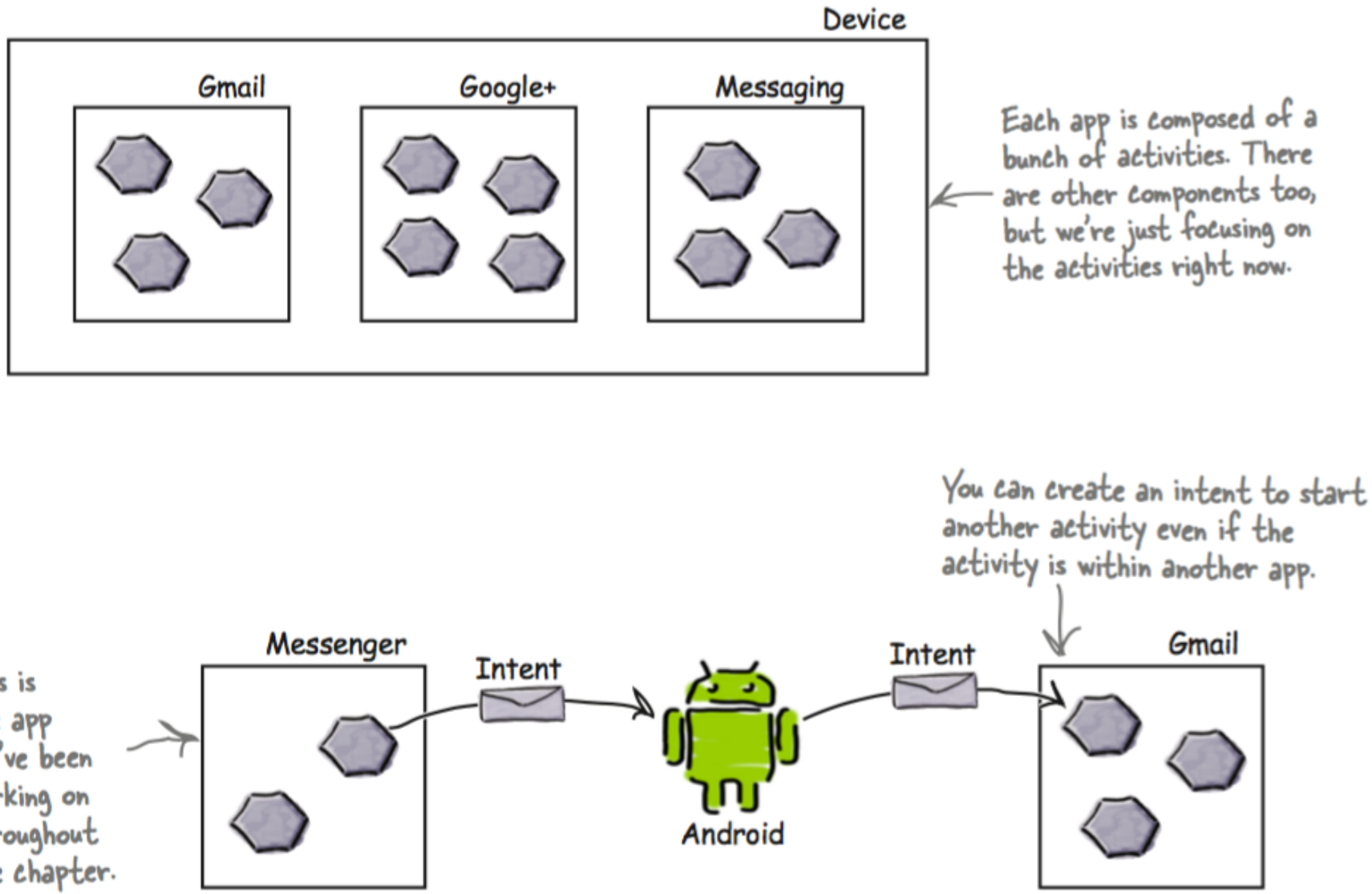


Prueba


4:15
My Messenger
A little message
Send Message
These are both full-screen, but we've snipped away some of the blank space.
4:15
My Messenger
A little message
Here's the text we entered, successfully passed via the intent to the second activity.

Cambiando la aplicación para enviar mensajes a otras personas

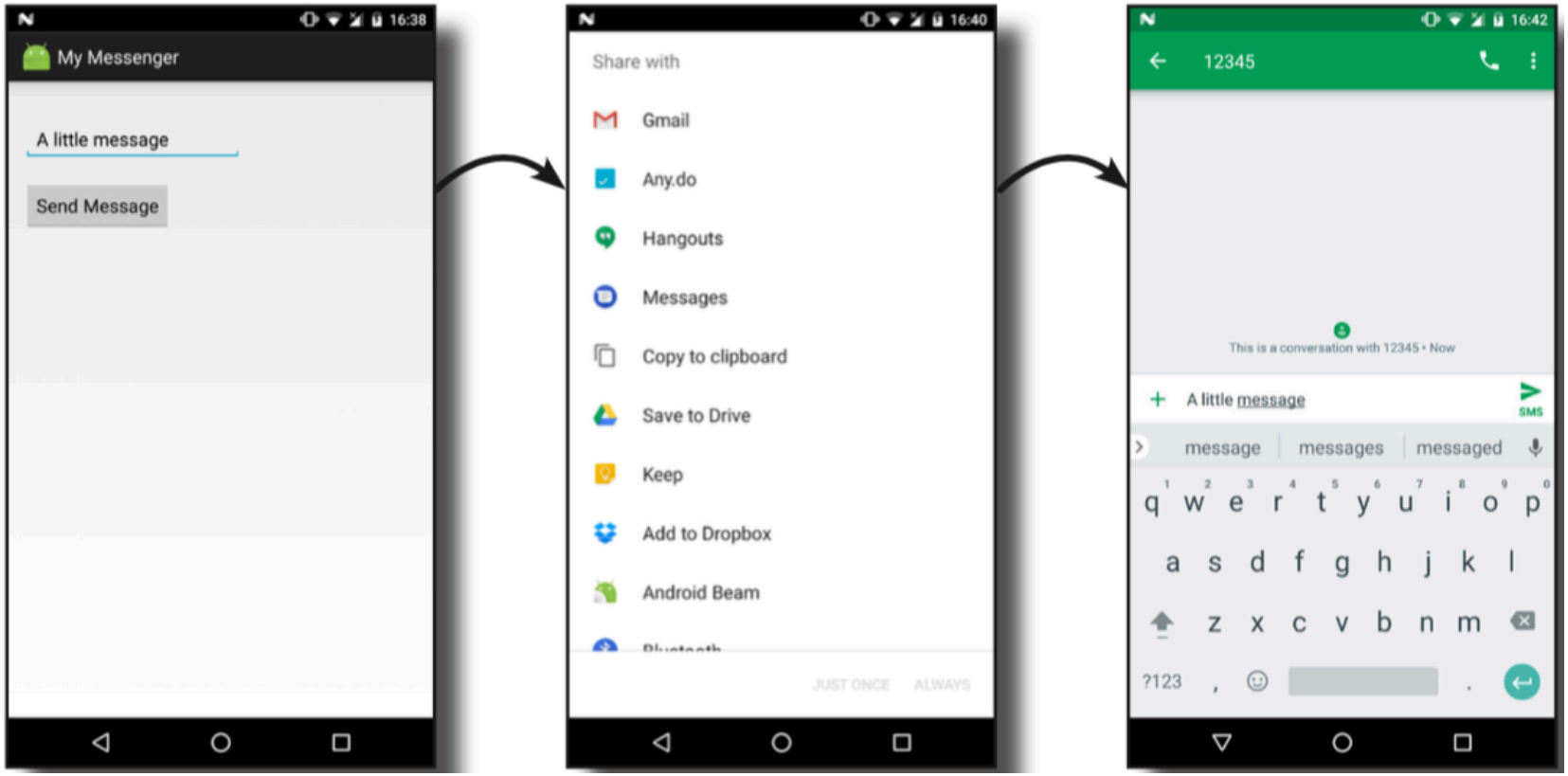
Cambiando la aplicación para enviar mensajes a otras personas



Cambiando la aplicación para enviar mensajes a otras personas

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.**
Chances are, there'll be more than one app on the user's 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.



Creando un intent con una acción específica

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

Agregando información extra

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

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

```
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.

Cambiando el intent para utilizar una acción

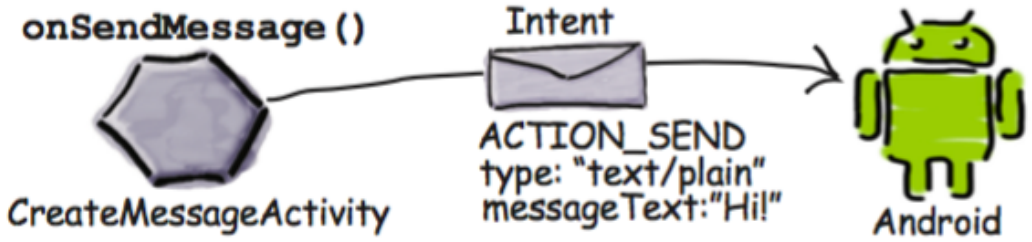
```
public class CreateMessageActivity extends Activity {

    @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.

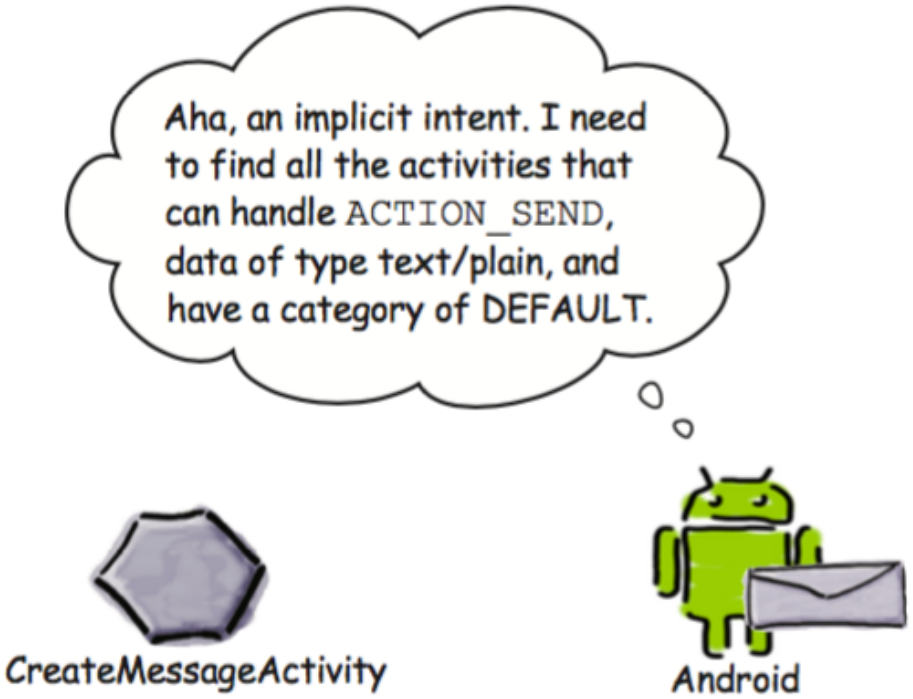
Instead of creating an intent that's explicitly for ReceiveMessageActivity, we're creating an intent that uses a send action.

Que pasa al ejecutar el código …

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.

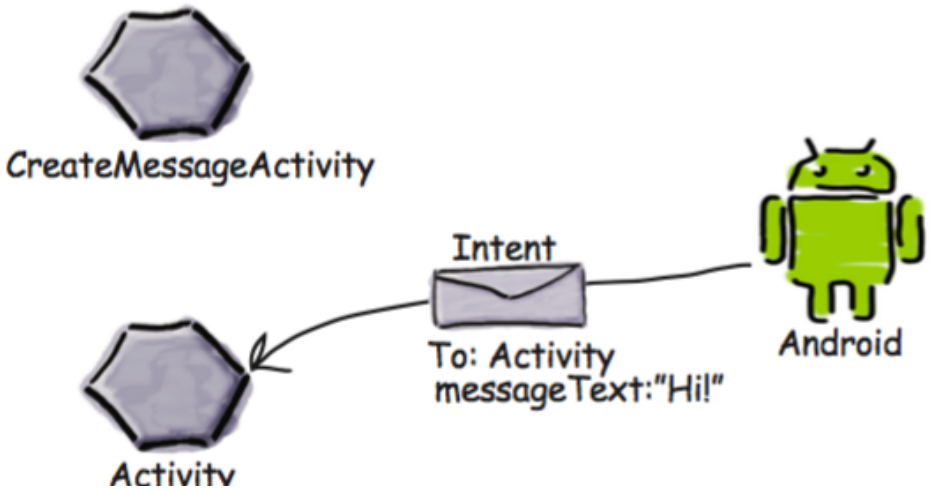


2. **Android sees that the intent can only be passed to activities able to handle ACTION_SEND and text/plain data. Android checks all the activities, looking for ones that are able to receive the intent.**
If no actions are able to handle the intent, an `ActivityNotFoundException` is thrown.

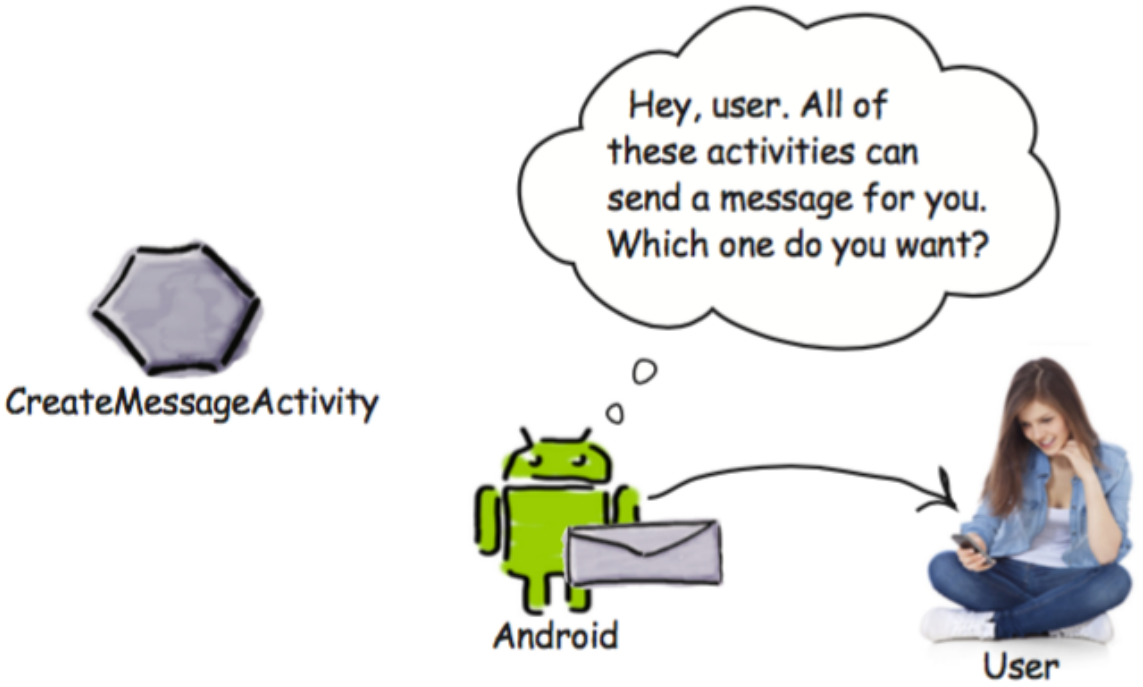


Que pasa al ejecutar el código …

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

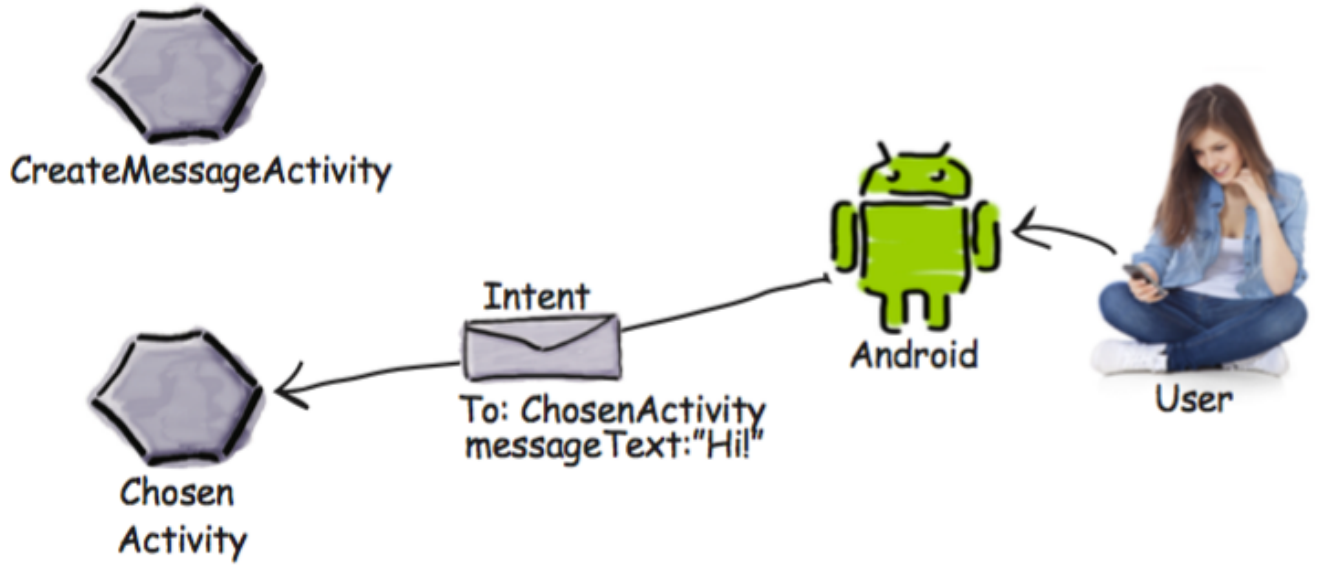


4 If more than one activity is able to receive the intent, Android displays an activity chooser dialog and asks the user which one to use.



Que pasa al ejecutar el código …

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.



El filtro de intent le dice a Android cual actividad puede manejar la acción

```
<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.

Ejecutando la aplicación en un dispositivo real

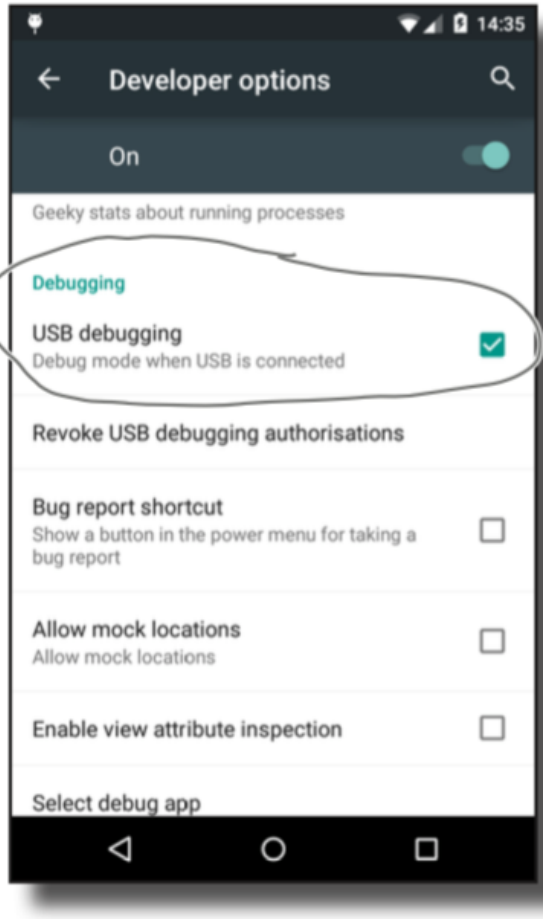
1. Enable USB debugging on your device

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.

Ejecutando la aplicación en un dispositivo real

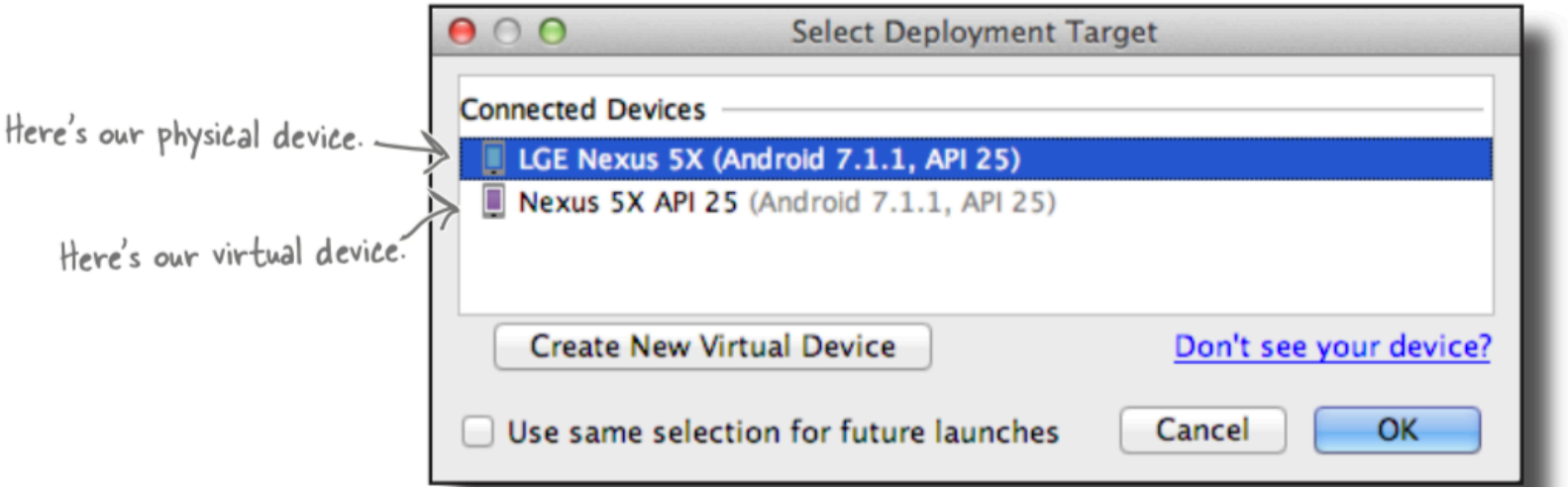
4. Stop your app running on the emulator

Before you can run your app on a different device, you need to stop it running on the current one (in this case the emulator). To do this, choose Run → "Stop 'app'", or click on the "Stop 'app'" button in the toolbar.

Click on this button in Android Studio's toolbar to stop the app running on the current device.

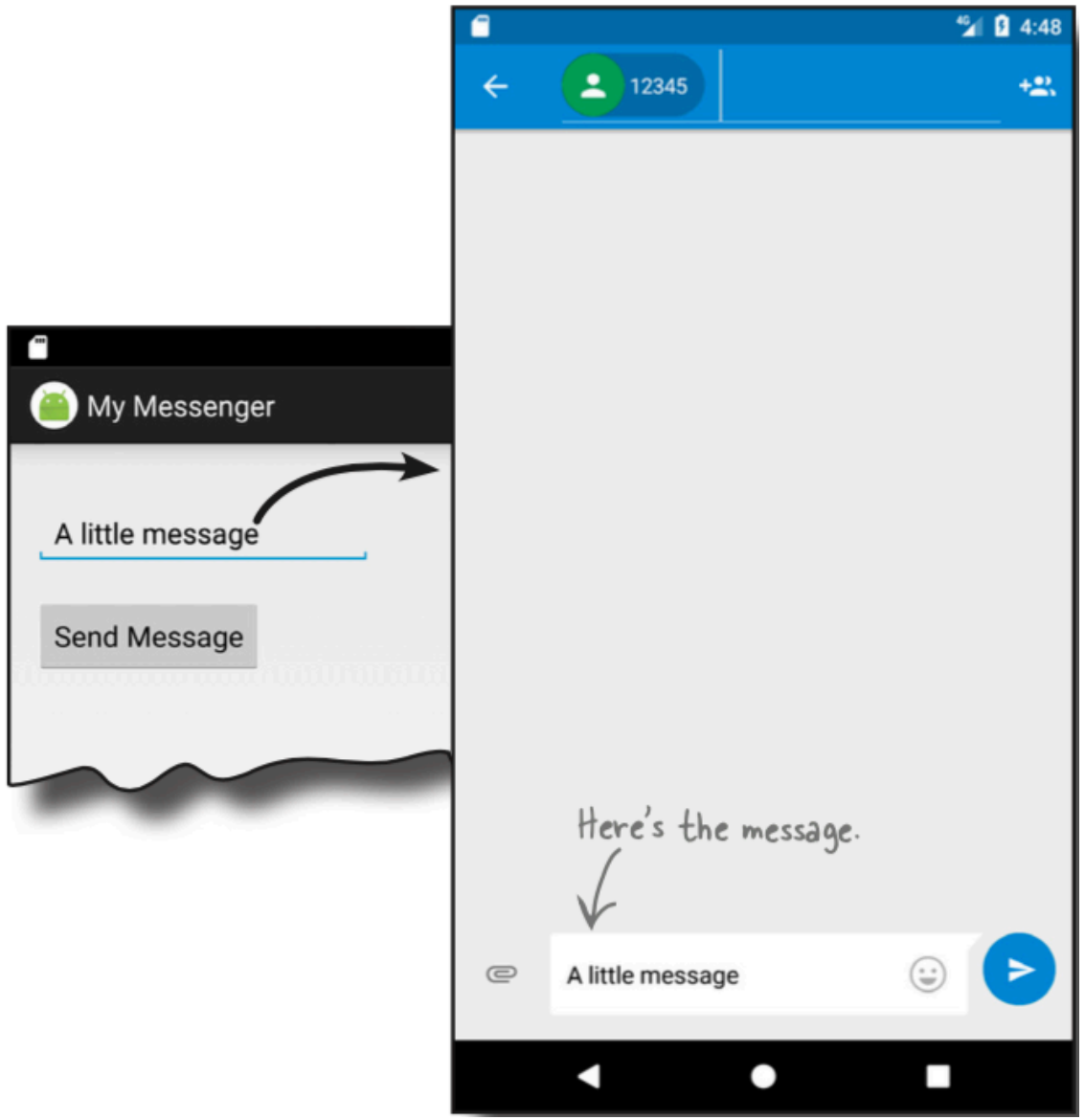
5. Run your app on the physical device

Run the app by choosing Run → "Run 'app'". Android Studio will ask you to choose which device you want to run your app on, so select your device from the list of available devices and click OK. Android Studio will install the app on your device and launch it.

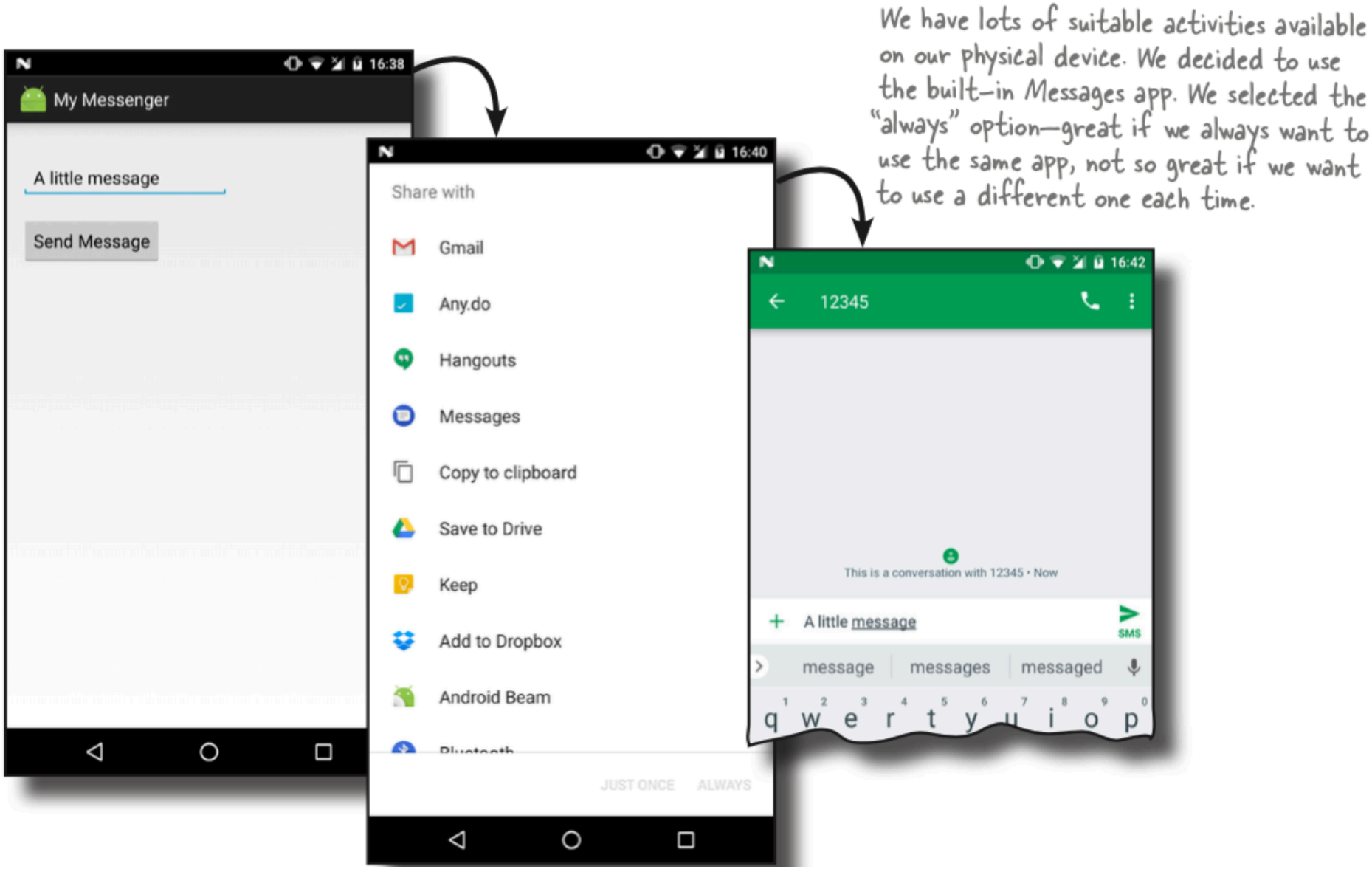


Prueba - Si hay sólo una actividad

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 that activity.



Prueba - Si hay más de una actividad



Como obligar al usuario a elegir una actividad

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

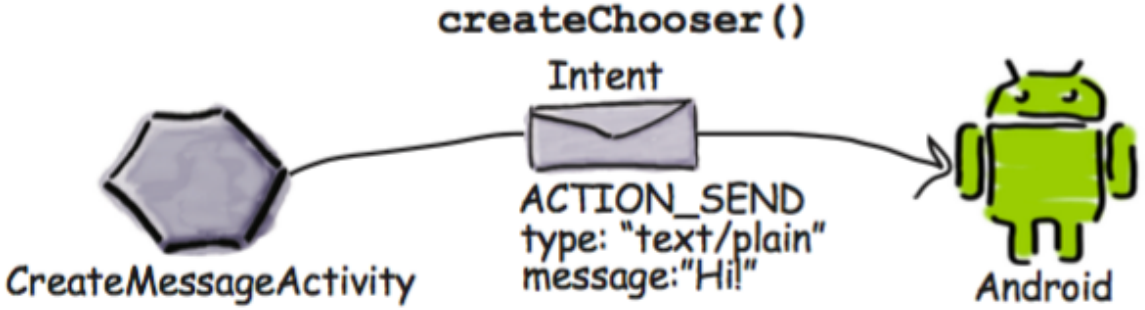
This is the intent you created earlier.

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

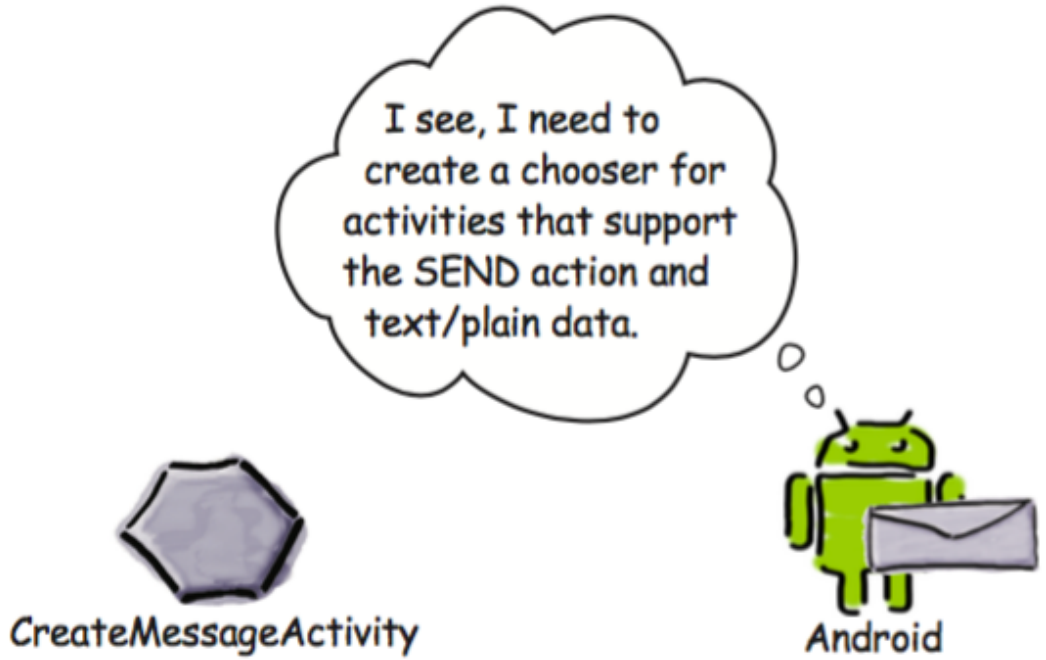
createChooser() permite especificar un título para el diálogo y no permite al usuario seleccionar una actividad para utilizarla por omisión

Que pasa al invocar 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.



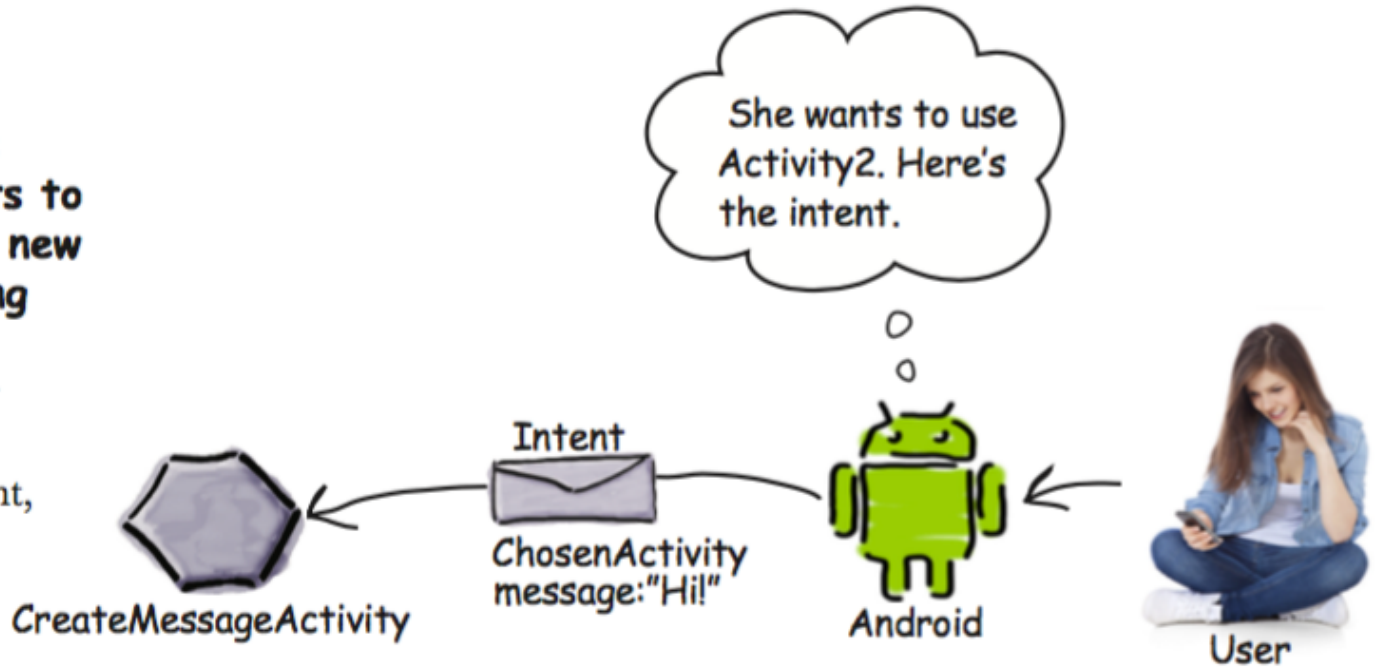
Que pasa al invocar createChooser()

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.

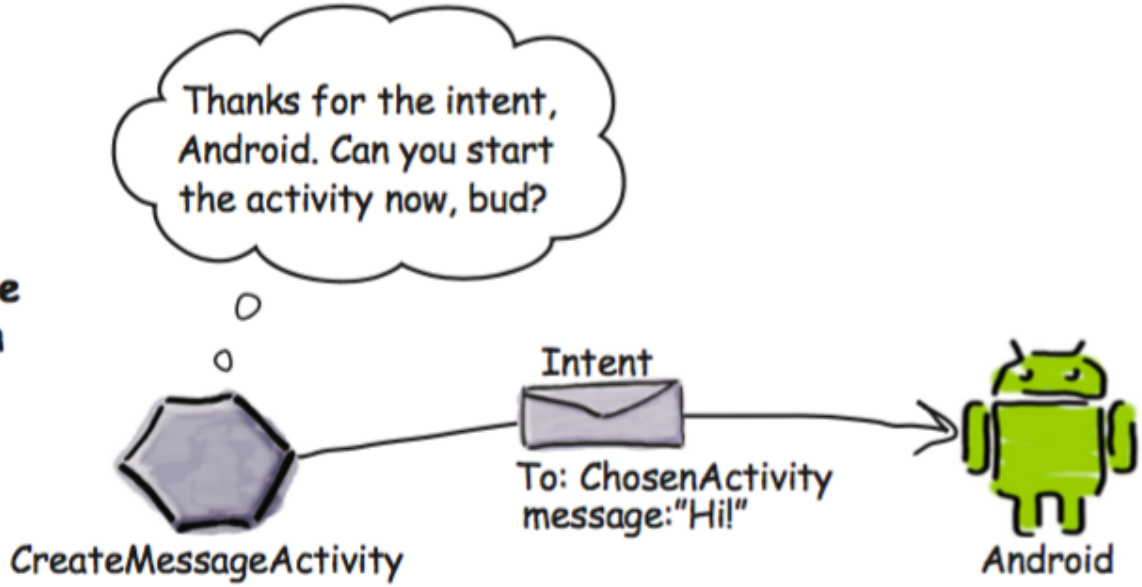
Que pasa al invocar createChooser()

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.

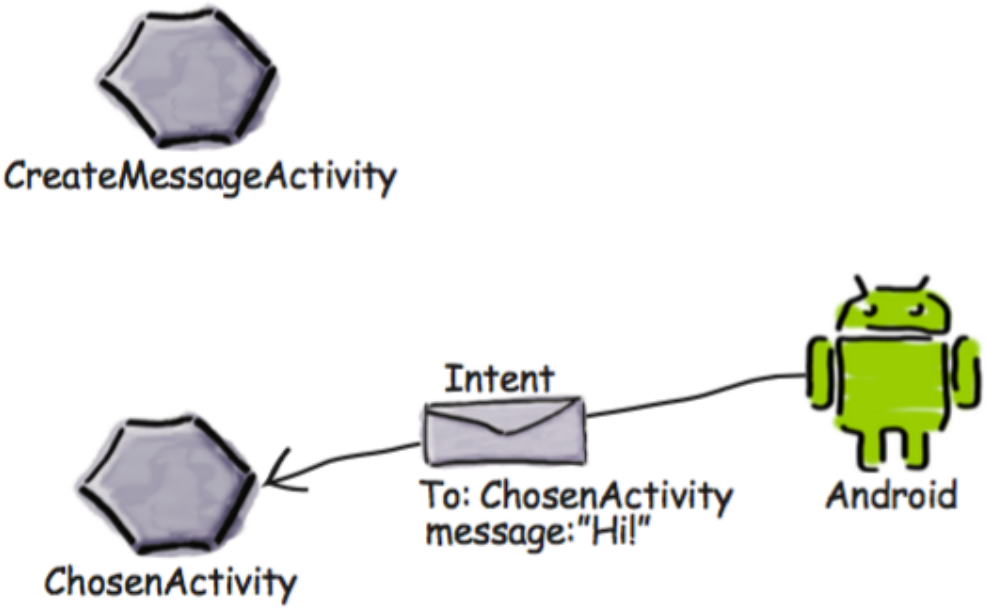


Que pasa al invocar createChooser()

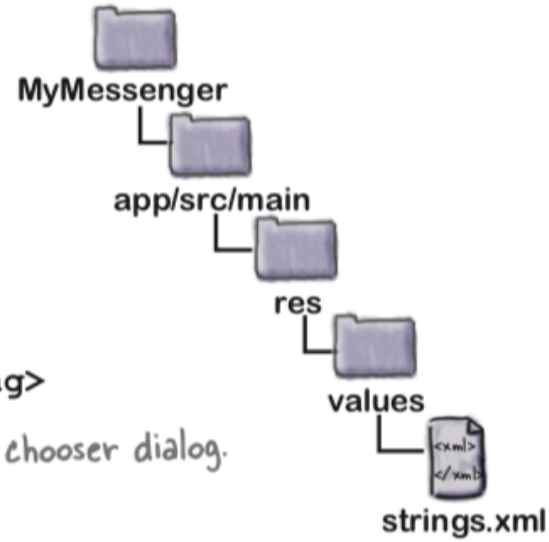
5 **The activity asks Android to start the activity specified in the intent.**



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

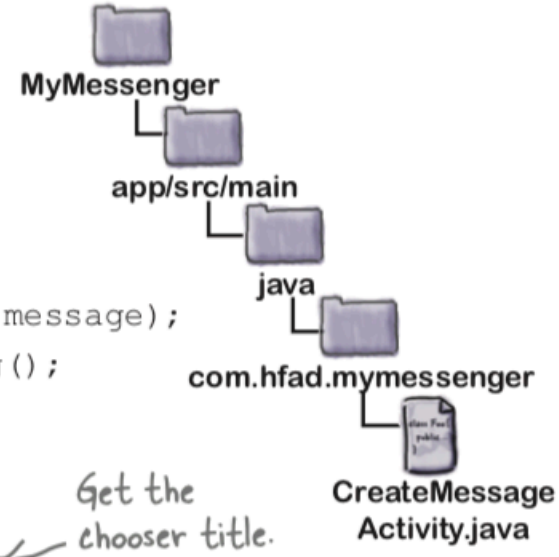


Cambiando el código para crear un chooser




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

This will be displayed in the chooser dialog.




```
...
    //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.

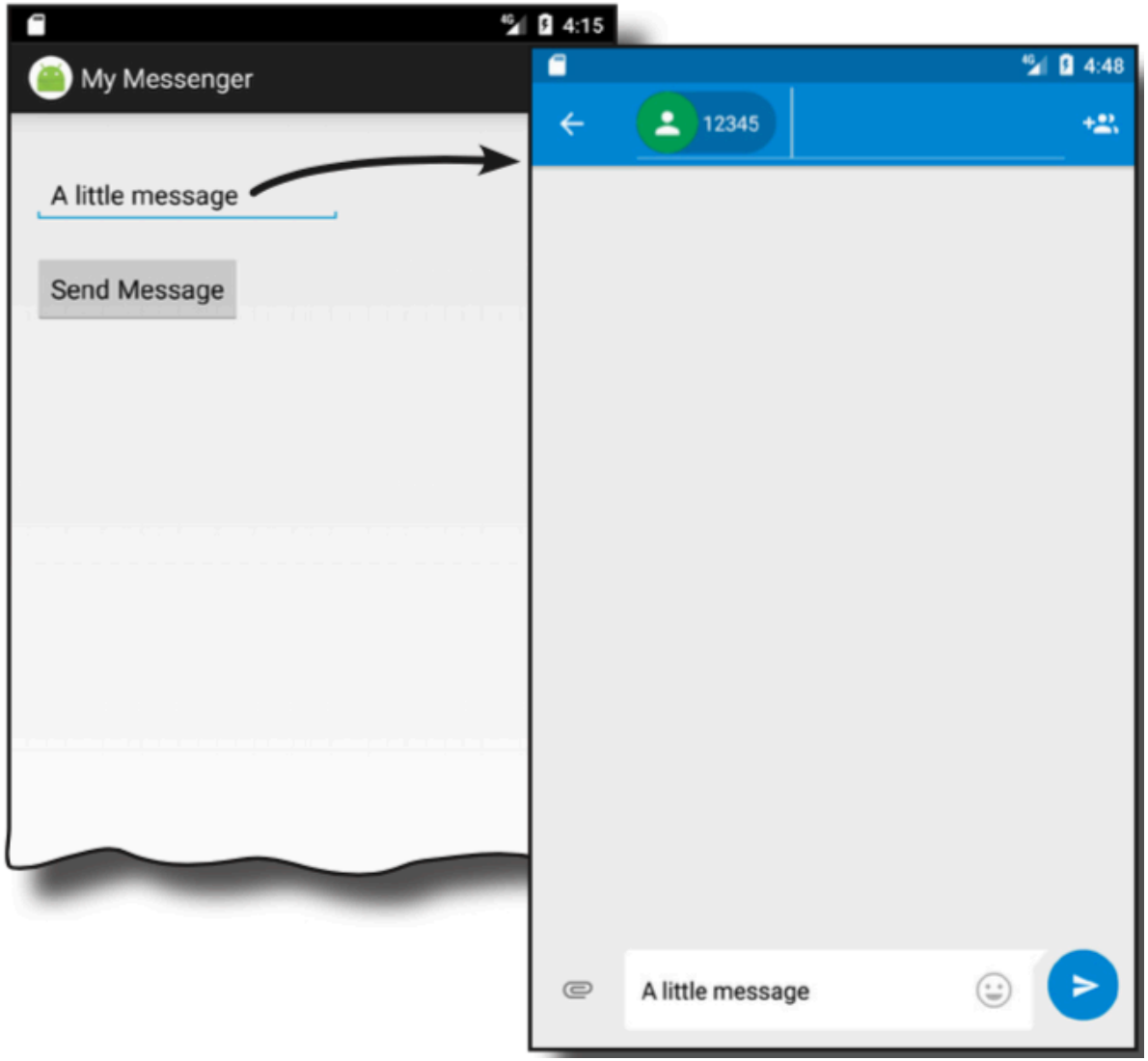
Delete this line.

Display the chooser dialog.

Start the activity that the user selected.

Prueba - Si hay sólo una actividad

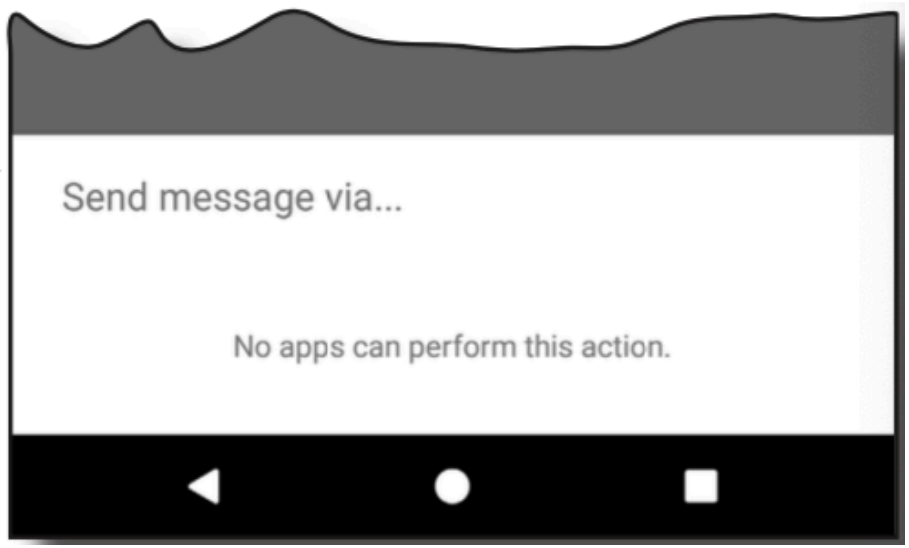
There's no change here—
Android continues to take
you straight to the activity.



Prueba - Si hay más de una actividad

Prueba

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



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);`