Android

Multiple Activities and Intents

Activities and Intents

What is an Activity?

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

What does an Activity do?

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

Apps and activities

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

Layouts and Activities

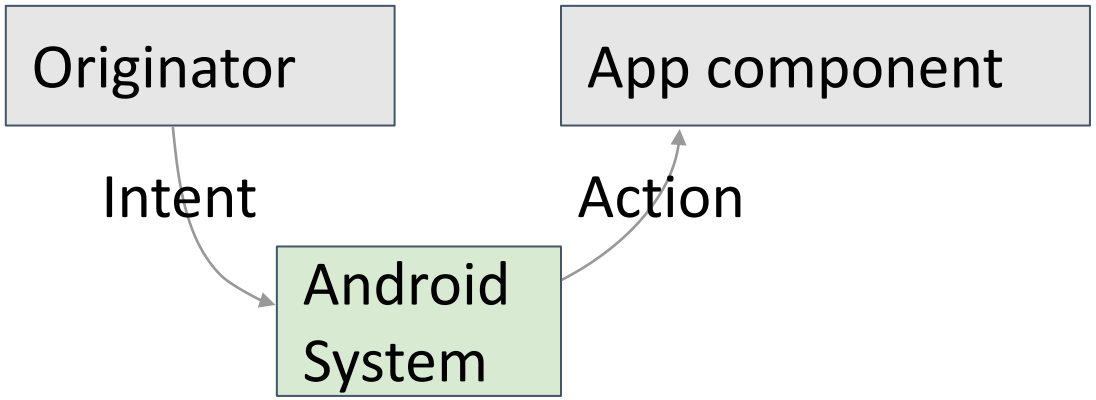
- An `Activity` typically has a UI layout
- Layout is usually defined in one or more XML files

Intents

What is an intent?

An `Intent` is a description of an operation to be performed.

An **`Intent`** is an object used to request an action from another **app component** via the Android system.



What can intents do?

- Start an `Activity`
  - A button click starts a new `Activity` for text entry
- Start n `Service`
  - Initiate downloading a file in the background
- Deliver `Broadcast`
  - The system informs everybody that the phone is now charging

Explicit and implicit intents

Explicit Intent

- Starts a specific `Activity`
  - Main activity starts for ex. the `ViewShoppingCart` `Activity`

Implicit Intent

- Asks system to find an `Activity` that can handle this request
  - Clicking Share opens a chooser with a list of apps

Starting Activities

Start an Activity with an explicit intent

To start a specific `Activity`, use an explicit `Intent`

1. Create an `Intent`
    - `Intent intent = new Intent(this, ActivityName.class);`
2. Use the `Intent` to start the `Activity`
    - **`startActivity`**`(intent);`

Start an Activity with implicit intent

To ask Android to find an `Activity` to handle your request, use an implicit `Intent`

1. Create an `Intent`
   - `Intent intent = new Intent(action, uri);`
2. Use the `Intent` to start the `Activity`
   - **`startActivity`**`(intent);`

Implicit Intents - Examples

Show a web page

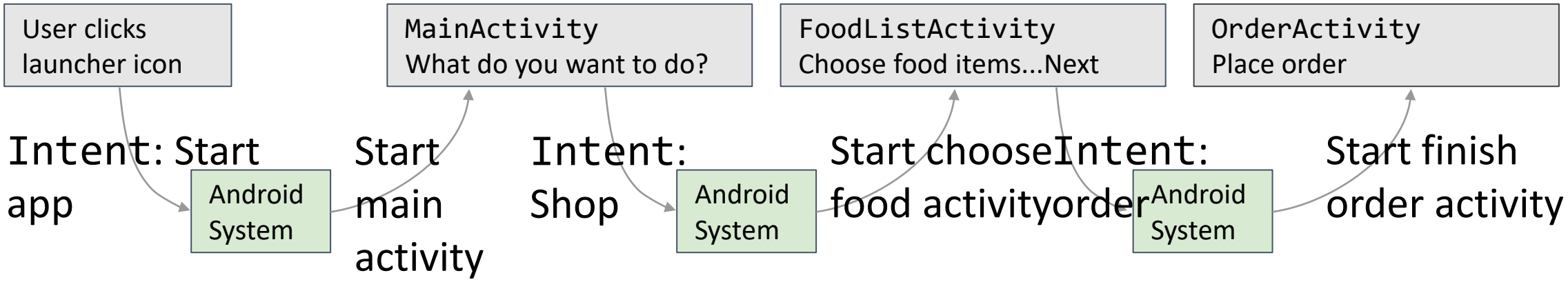
```
Uri uri = Uri.parse("http://www.google.com");
Intent it = new Intent(Intent.ACTION_VIEW,uri);
startActivity(it);
```

Dial a phone number

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

How Activities Run

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



Sending and Receiving Data

Two types of sending data with intents

- Data—one piece of information whose data location can be represented by an URI

- Extras—one or more pieces of information as a collection of key-value pairs

Sending and retrieving data

In the first (sending) `Activity`:

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

In the second (receiving) `Activity`:

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

Putting a URI as intent data

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

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

Put information into intent extras

- `putExtra(String name, int value)`
  ⇒ `intent.putExtra("level", 406);`
- `putExtra(String name, String[] value)`
  ⇒
```
String[] foodList = {"Rice", "Beans", "Fruit"};
intent.putExtra("food", foodList);
```
- `putExtras(bundle);`
  ⇒ if lots of data, first create a bundle and pass the bundle

Create a bundle and pass the bundle

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


// creating a bundle object
Bundle bundle = new Bundle();


// storing the string value in the bundle
// which is mapped to key
bundle.putString("key1", "GFG :- Main Activity");


// passing the bundle into the intent
intent.putExtras(bundle);


// starting the intent
startActivity(intent);
```

Sending data to an activity with extras

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

Intent intent = new Intent(this, SecondActivity.class);
String message = "Hello Activity!";
intent.putExtra(EXTRA_MESSAGE_KEY, message);
startActivity(intent);
```

Get data from intents

- `getData();`
  ⇒ `Uri locationUri = intent.getData();`
- `int getIntExtra (String name, int defaultValue)`
  ⇒ `int level = intent.getIntExtra("level", 0);`
- `Bundle bundle = intent.getExtras();`
  ⇒ Get all the data at once as a bundle

Get all the data at once as a bundle

```
// getting the bundle back from the android
Bundle bundle = getIntent().getExtras();


// getting the string back
String title = bundle.getString("key1", "Default");
```

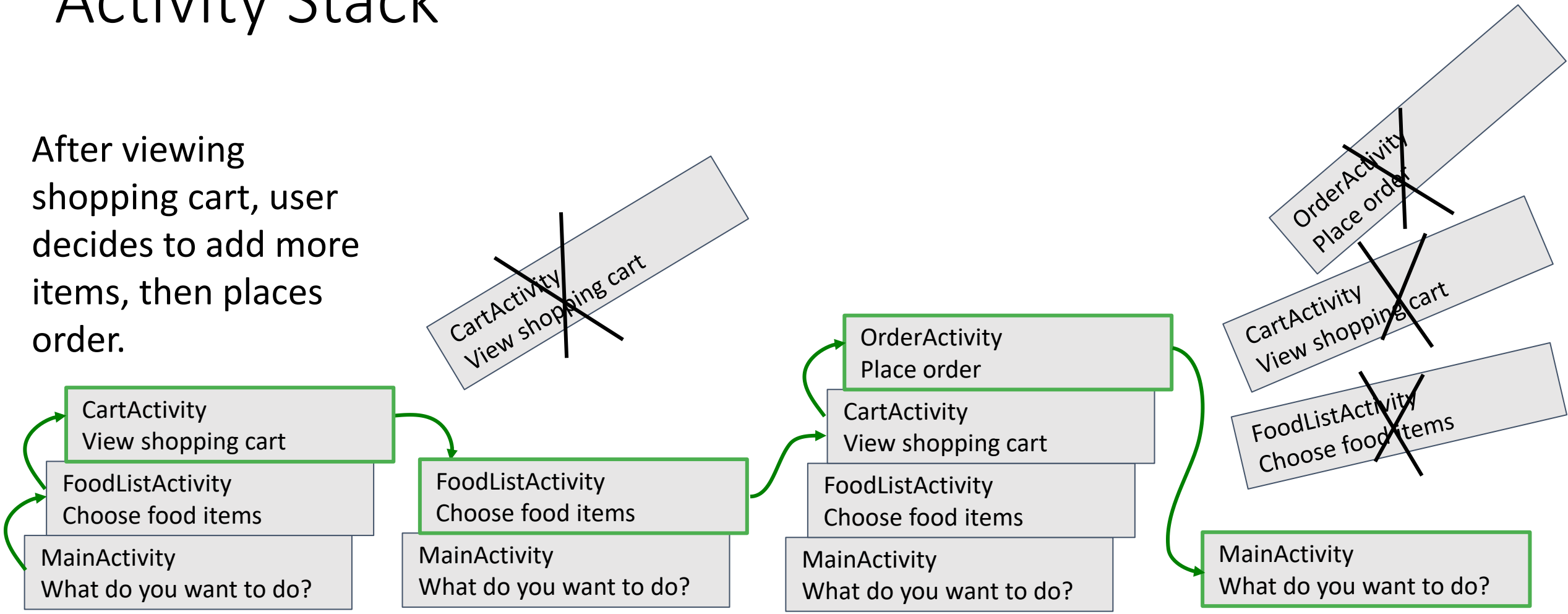
Navigation

Activity stack

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

Activity Stack

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



Two forms of navigation

Temporal or back navigation

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

Ancestral or up navigation

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

Back navigation

- Back stack preserves history of recently viewed screens
- Back stack contains all the `Activity` instances that have been launched by the user in reverse order *for the current task*
- Each task has its own back stack
- Switching between tasks activates that task's back stack

Up navigation

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

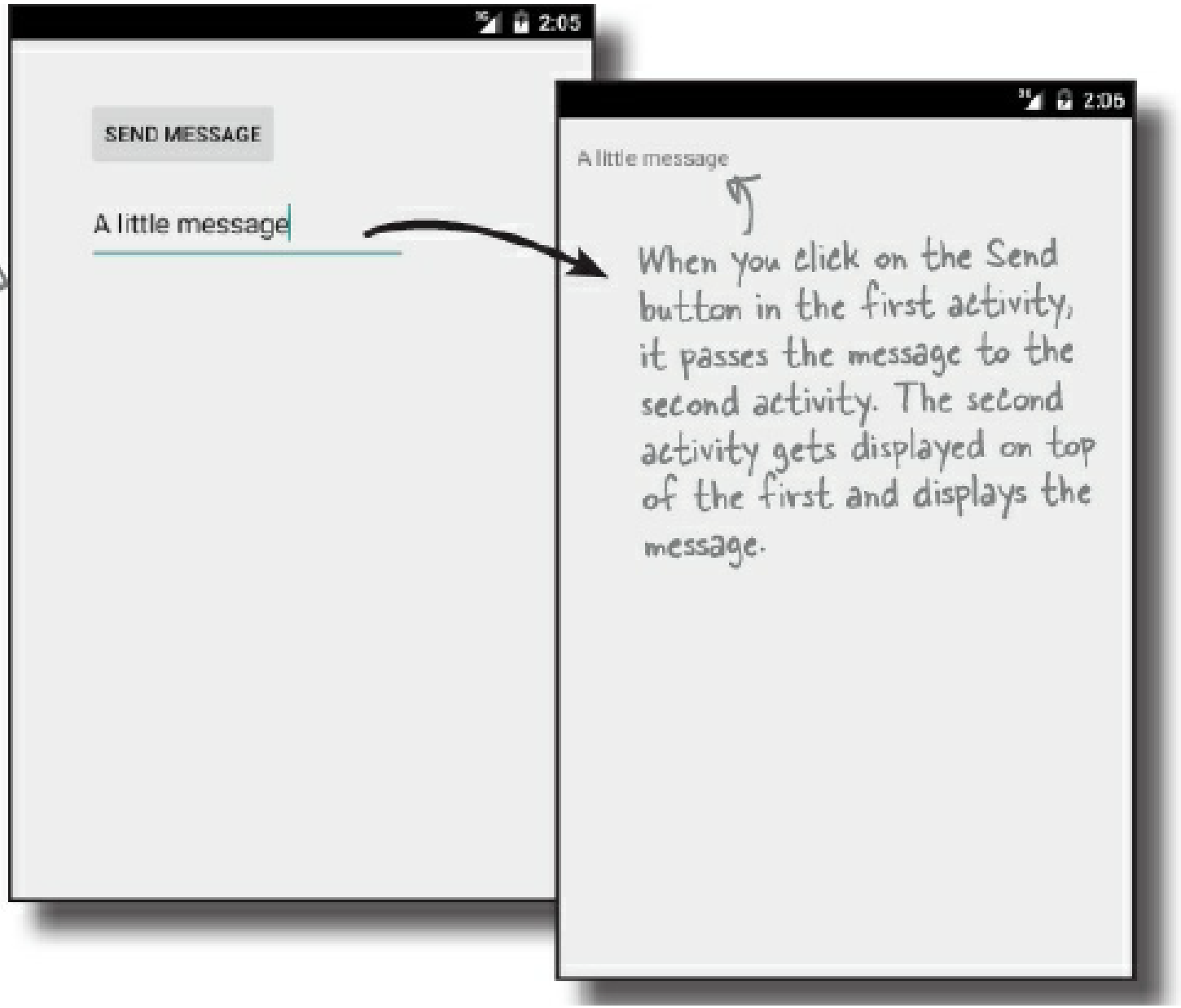
```
<activity
    android:name=".ShowDinnerActivity"
    android:parentActivityName=".MainActivity" >
</activity>
```

Application

What we'll do?

- We're going to build an app containing two activities. The first activity will allow you to type a message.
- When you click on a button in the first activity, it will launch the second activity and pass it the message.
- The second activity will then display the message.

The first activity lets you enter a message.



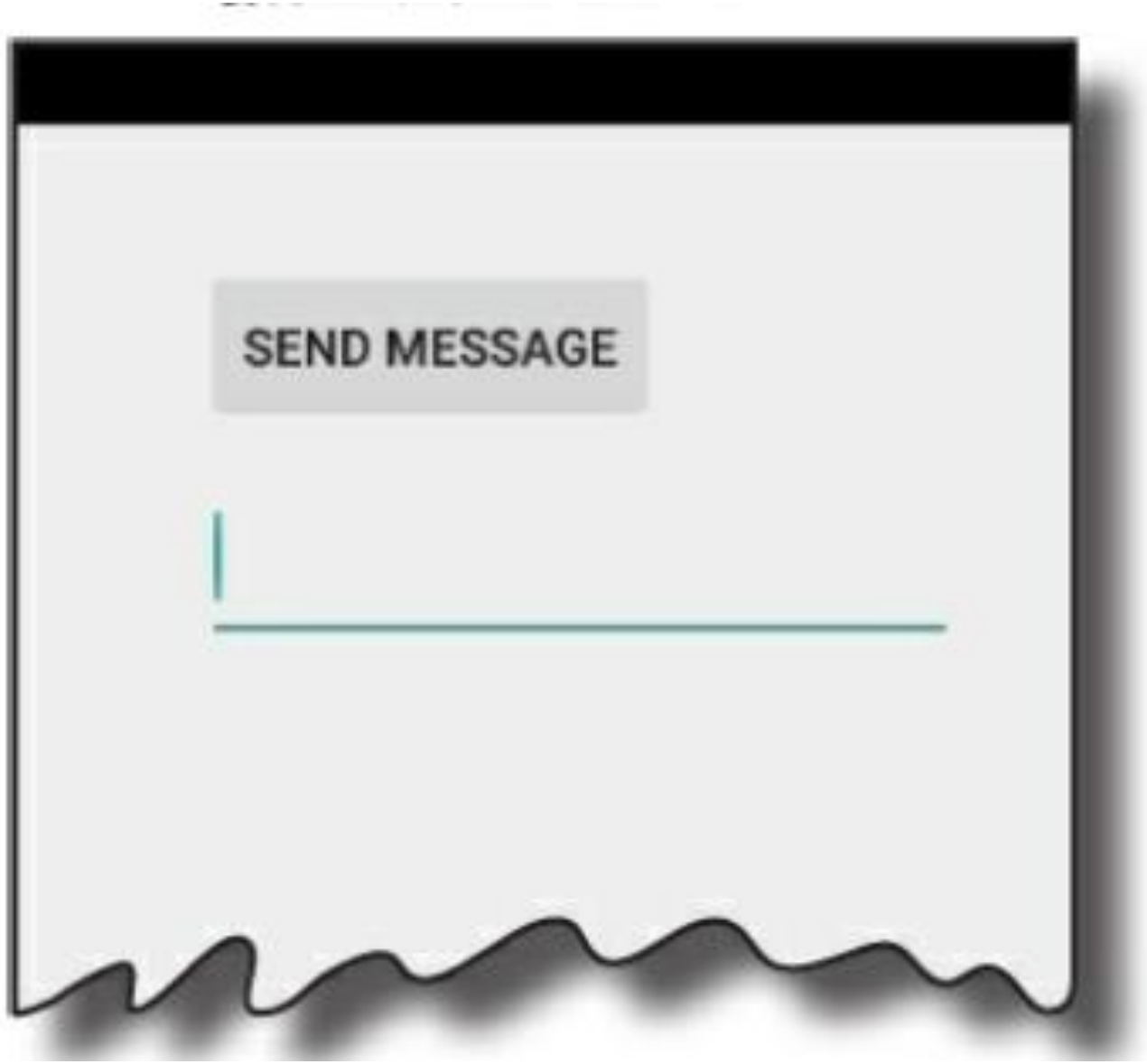
Application

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.**

Modify Layout

- In the XML for the *activity_create_message.xml* file:
- We removed the ~~<TextView>~~ that Android Studio created for us, and replaced it with <Button> and <EditText> elements.
- The <EditText> element gives you an editable text field you can use to enter data.

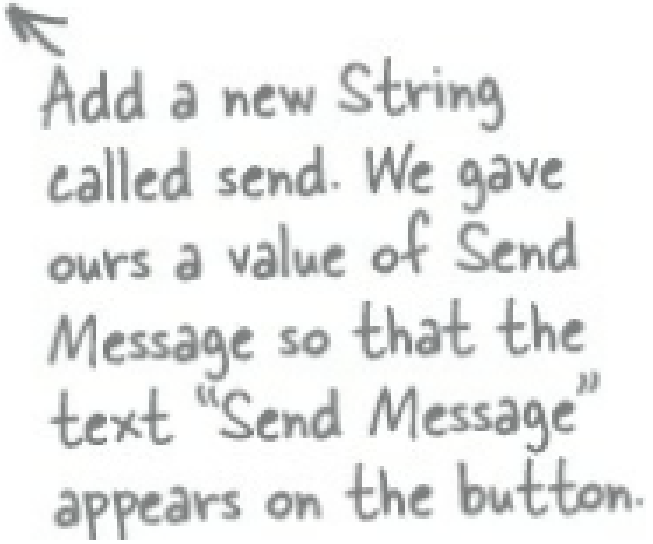
Layout



Add String Resources

- we need to add a string called “send” to *strings.xml* and give it a value. This value is the text we want to appear on the button. Do this

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



Add onClick to button

- ADD the line in the <Button> element

  - android:onClick="onSendMessage“

- It means that the onSendMessage() method in the activity will fire when the button is clicked

Add onSendMessage()

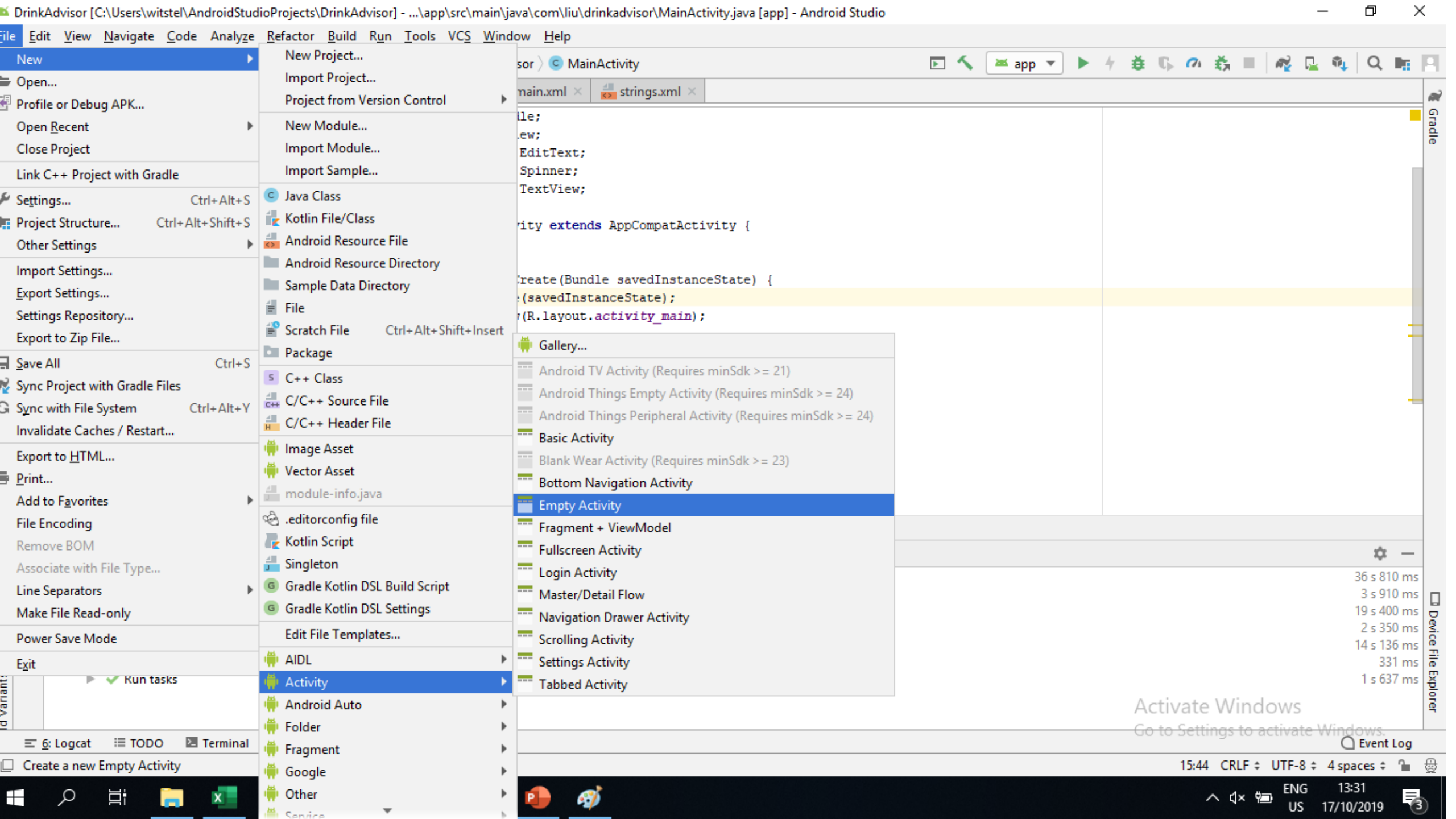
Open up the *CreateMessageActivity.java* file and add the the following code:

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

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.

Create the second activity and layout

- To create the new activity, choose File → New → Activity, and choose the option for Blank Activity.
- You will be presented with a new screen where you can choose options for your new activity.
- Give the new activity a name of “ReceiveMessageActivity” and the layout a name of “activity_receive_message”.


DrinkAdvisor [C:\Users\witstel\AndroidStudioProjects\DrinkAdvisor] - ...\app\src\main\java\com\liu\drinkadvisor\MainActivity.java [app] - Android Studio
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
New
Open...
Profile or Debug APK...
Open Recent
Close Project
Link C++ Project with Gradle
Settings... Ctrl+Alt+S
Project Structure... Ctrl+Alt+Shift+S
Other Settings
Import Settings...
Export Settings...
Settings Repository...
Export to Zip File...
Save All Ctrl+S
Sync Project with Gradle Files
Sync with File System Ctrl+Alt+Y
Invalidate Caches / Restart...
Export to HTML...
Print...
Add to Favorites
File Encoding
Remove BOM
Associate with File Type...
Line Separators
Make File Read-only
Power Save Mode
Exit
New Project...
Import Project...
Project from Version Control
New Module...
Import Module...
Import Sample...
Java Class
Kotlin File/Class
Android Resource File
Android Resource Directory
Sample Data Directory
File
Scratch File Ctrl+Alt+Shift+Insert
Package
C++ Class
C/C++ Source File
C/C++ Header File
Image Asset
Vector Asset
module-info.java
.editorconfig file
Kotlin Script
Singleton
Gradle Kotlin DSL Build Script
Gradle Kotlin DSL Settings
Edit File Templates...
AIDL
Activity
Android Auto
Folder
Fragment
Google
Other
Gallery...
Android TV Activity (Requires minSdk >= 21)
Android Things Empty Activity (Requires minSdk >= 24)
Android Things Peripheral Activity (Requires minSdk >= 24)
Basic Activity
Blank Wear Activity (Requires minSdk >= 23)
Bottom Navigation Activity
Empty Activity
Fragment + ViewModel
Fullscreen Activity
Login Activity
Master/Detail Flow
Navigation Drawer Activity
Scrolling Activity
Settings Activity
Tabbed Activity
MainActivity
app
main.xml
strings.xml
Gradle
Device File Explorer
Run tasks
6: Logcat
TODO
Terminal
Event Log
Create a new Empty Activity
15:44 CRLF UTF-8 4 spaces
Activate Windows
Go to Settings to activate Windows.
ENG US 13:31 17/10/2019

Android manifest file

- Every Android app must include a file called *AndroidManifest.xml*. You can find it in the *app/src/main* folder of your project.
- The *AndroidManifest.xml* file contains essential information about your app, such as what activities it contains, required libraries, and other declarations.
- Android creates the file for you when you create the app.

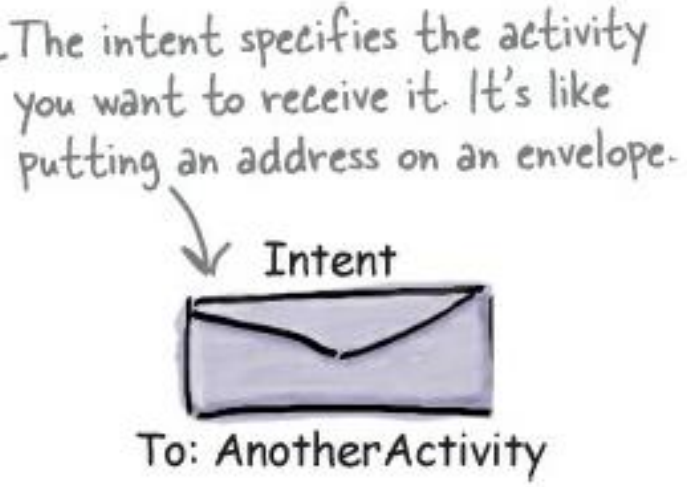
Recall Intent

- **An intent is a type of message**
- 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”.
- 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 start an activity by creating an intent and using it in the startActivity() method.**

Create the Intent

- You can create and send an intent using just a couple of lines of code. You start by creating the intent like this:

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



1. The first parameter tells Android which object the intent is from, and you can use the word `this` to refer to the current activity.
2. The second parameter is the class name of the activity that needs to receive the intent.

Start the Activity

- Once you've created the intent, you pass it to Android like this:

```
startActivity(intent);
```

This tells Android to start the activity specified by the intent.

Once Android receives the intent, it checks everything's OK and tells the activity to start.

If it can't find the activity, it throws an **`ActivityNotFoundException`**.

Open the other activity

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

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

Add the following code

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

Start activity ReceiveMessageActivity.

Modify the XML

- We need to make a couple of changes to the layout. We need to give the **<TextView>** element an ID of "**message**" so that we can reference it in our activity code, and we need to stop the String "Hello world!" from appearing.

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

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

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

The “putExtra()” Method

- **putExtra()** puts extra information in an intent
- 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);**

The “putExtra” value

- **intent.putExtra("message", value);**

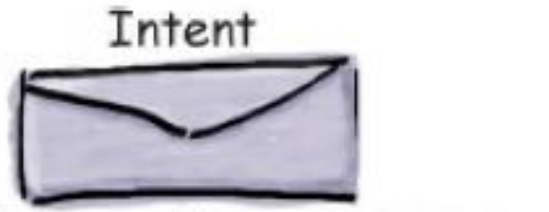
- where message is a String name for the value you’re passing in, and value is the value.
- The putExtra() method is overloaded so value has many possible types.
- As an example, it can be a primitive such as a boolean or int, an array of primitives, or a String.
- You can use putExtra() repeatedly to add numerous extra data to the intent.
- If you do this, make sure you give each one a unique name.

Retrieve extra information from an intent

- First of all, we get the intent using:
  - **Intent 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.

The “getIntent()” Method

It depends on the type of information that was sent. As an example, if you know the intent includes a String value with a name of “**message**”, you would use the following:




```
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”.

The “getIntExtra” Example

- You’re not just limited to retrieving String values. As an example, you can use
    - **int intNum = intent.getIntExtra("name", default_value);**
- to retrieve an int with a name of “**name”**.
- *default_value* specifies what int value you should use as a default.

The “onSendMessage” Code

```
//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("message", messageText);
    startActivity(intent);
}
```

Get the text from the editable text field with an ID of message.

Add the text to the intent, giving it a name of "message".

ReceiveMessageActivity.java

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_receive_message);

    Intent intent = getIntent();
    String messageText= intent.getStringExtra("message");
    TextView messageView = (TextView)findViewById(R.id.messageText);
    messageView.setText(messageText);
}
```

References

- developer.android.com/courses/adf-v2
- Chapter 3 of the textbook:

  Head First Android Development, 2nd Edition; by Dawn Griffiths, David Griffiths; publisher(s): O'Reilly Media, Inc.; ISBN · 9781491974056