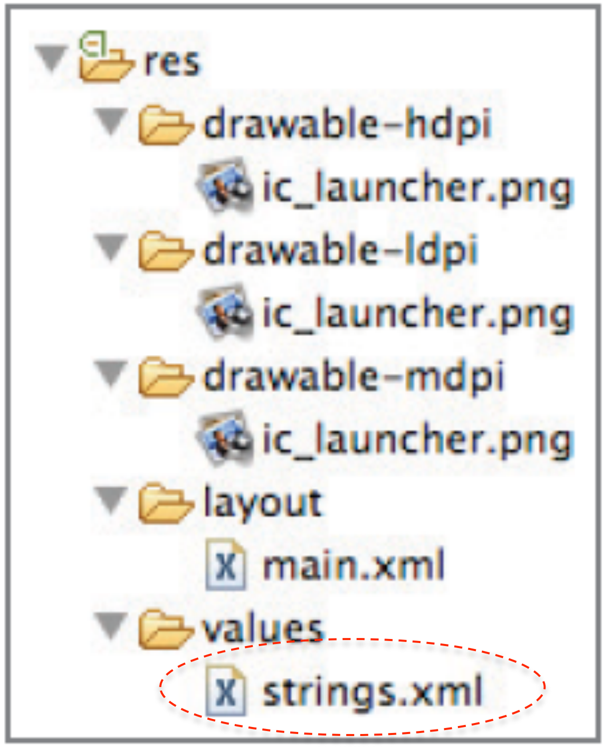
Externalizing resources

- It's always good practice to keep non-code resources external to your code.
- Android dynamically selects resources from resource trees that contain different values for alternative hardware configurations, languages, and locations.
- R class file is automatically generated to enable resource reference in code.


```
res
  drawable-hdpi
    ic_launcher.png
  drawable-ldpi
    ic_launcher.png
  drawable-mdpi
    ic_launcher.png
  layout
    main.xml
  values
    strings.xml
```

Define string resources – an example

- Define a string in strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello!</string>
</resources>
```

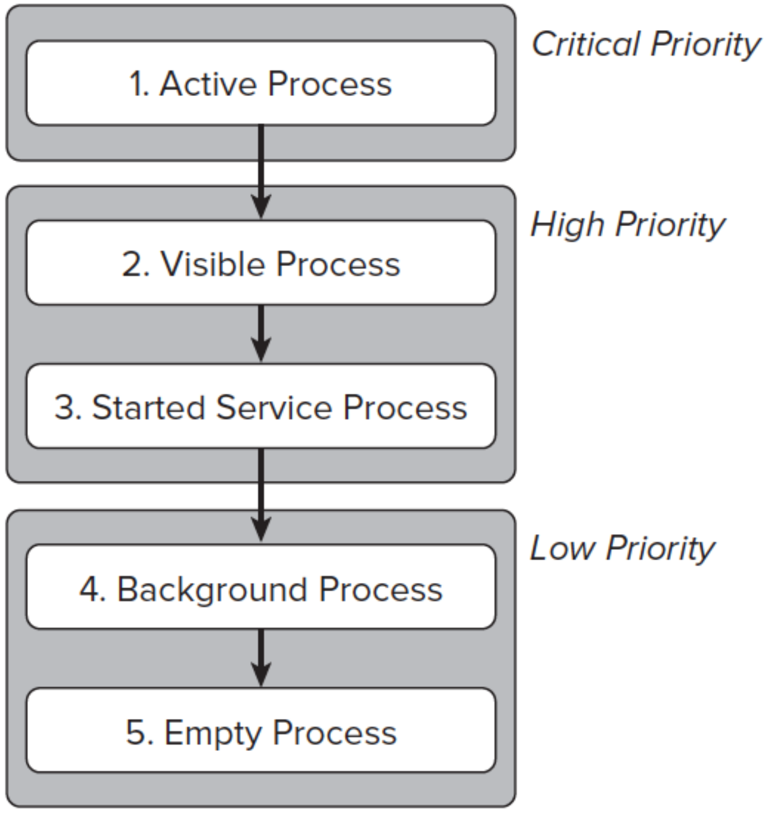
- Use the defined string.

```
Resources myResources = getResources();

CharSequence styledText = myResources.getText(R.string.stop_message);
Drawable icon = myResources.getDrawable(R.drawable.app_icon);

int opaqueBlue = myResources.getColor(R.color.opaque_blue);
```

```
String string =  getString (R.string.hello);
```

Android application lifecycle

- Android applications have limited control over their own lifecycle.
- Application priority
  - Equals to its highest-priority component.
- All Android applications continue running and in memory until the system needs resources for other applications.



Quick exercise

- Two applications A and B have the same priority.
- A spends longer time staying in that priority level than B.
- B depends on a content provider supplied by A.

- Which application might be killed the first and why?

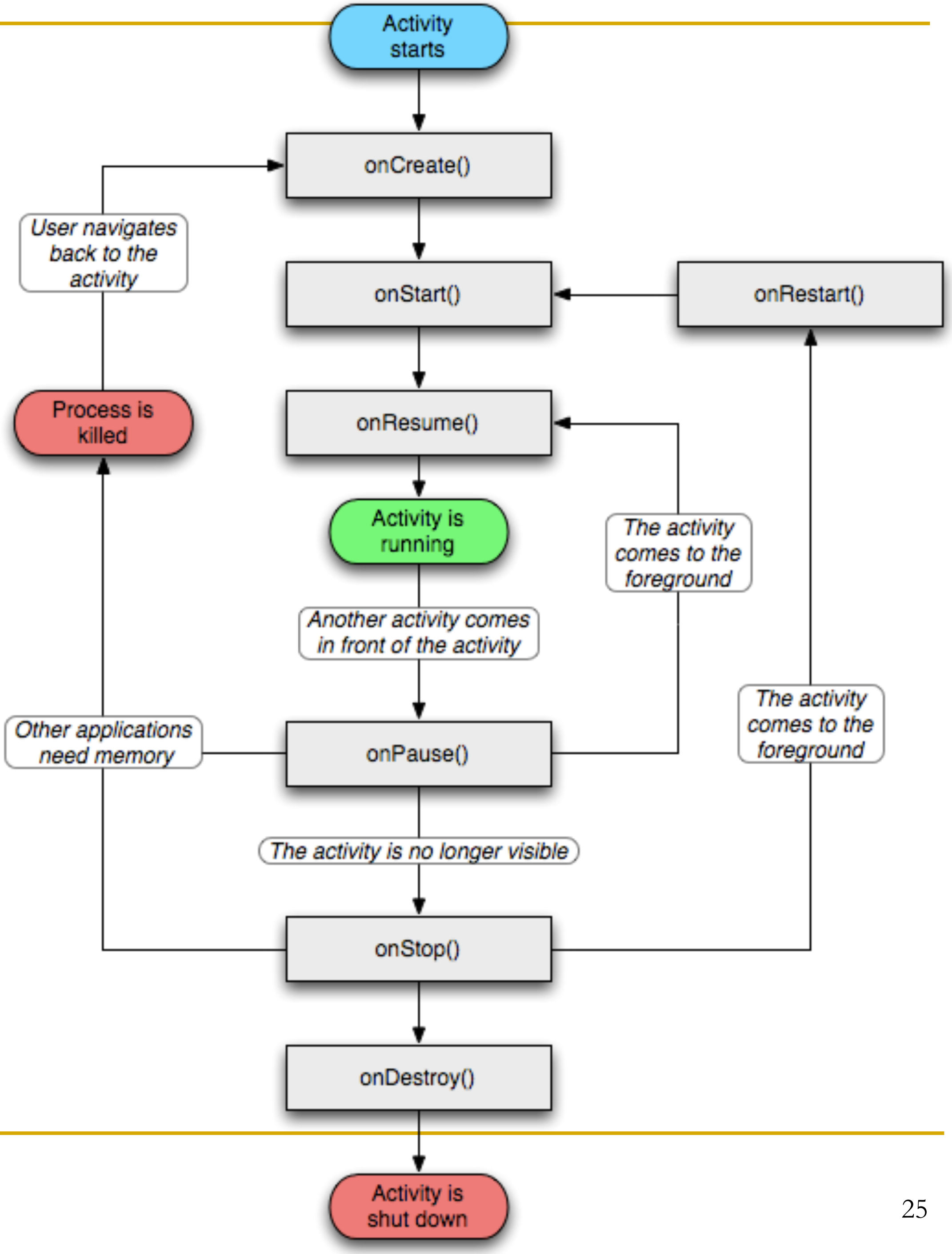
What is an activity?

- An activity is a window that contains the user interface of your application.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="net.learn2develop.Activities"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon"
      android:label="@string/app_name">
      <activity android:name=".MainActivity"
                android:label="@string/app_name">
          <intent-filter>
              <action android:name="android.intent.action.MAIN" />
              <category
                  android:name="android.intent.category.LAUNCHER" />
          </intent-filter>
      </activity>
    </application>
    <uses-sdk android:minSdkVersion="9" />
</manifest>
```

Activity state

- Active state
- Paused state
- Stopped state
- Inactive state

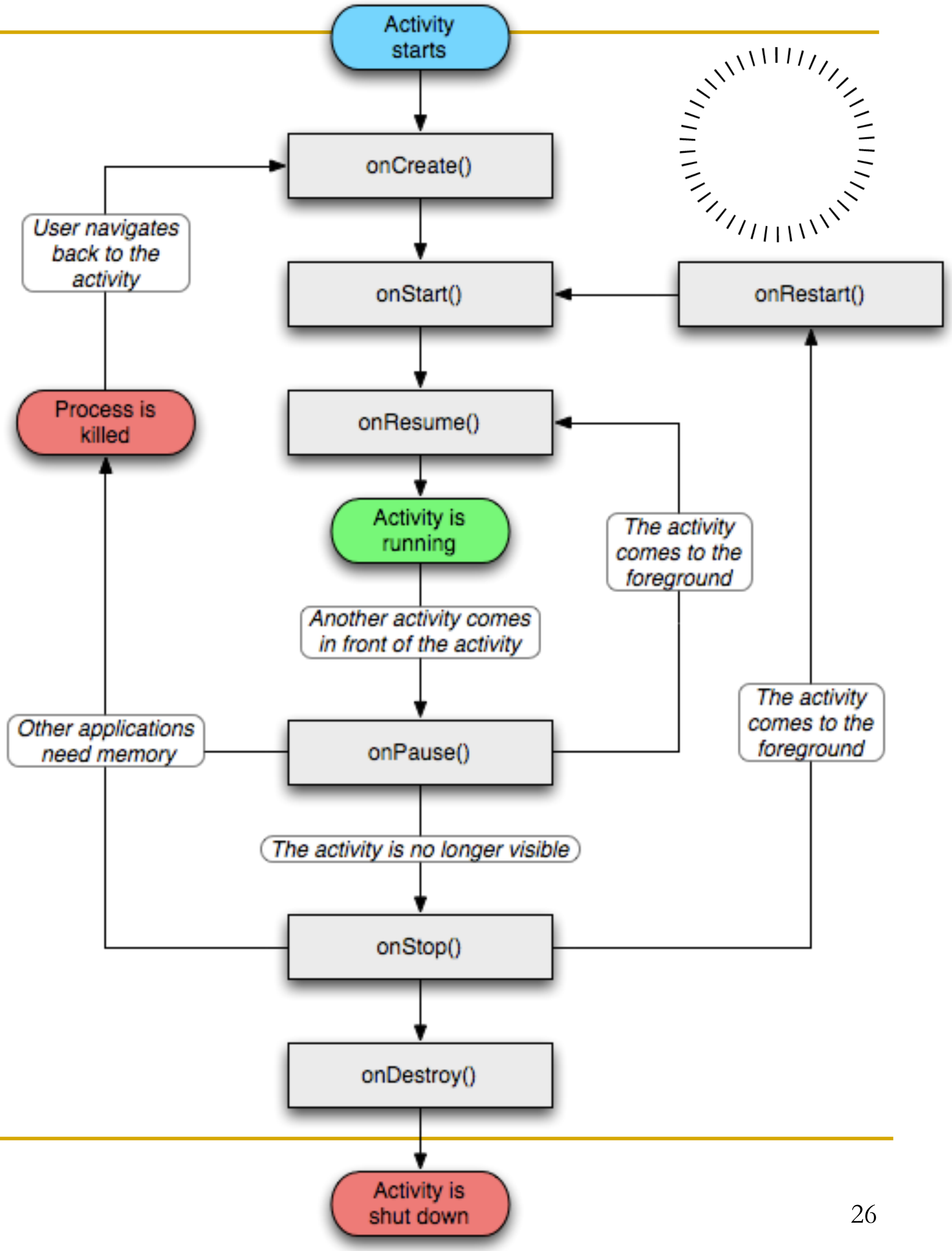


Activity starts
onCreate()
onStart()
onRestart()
onResume()
Activity is running
User navigates back to the activity
Process is killed
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Quick exercise

- Which event handler should be used to make state information persistent?

  - A. onRestart()
  - B. onStart()
  - C. onStop()
  - D. onResume()

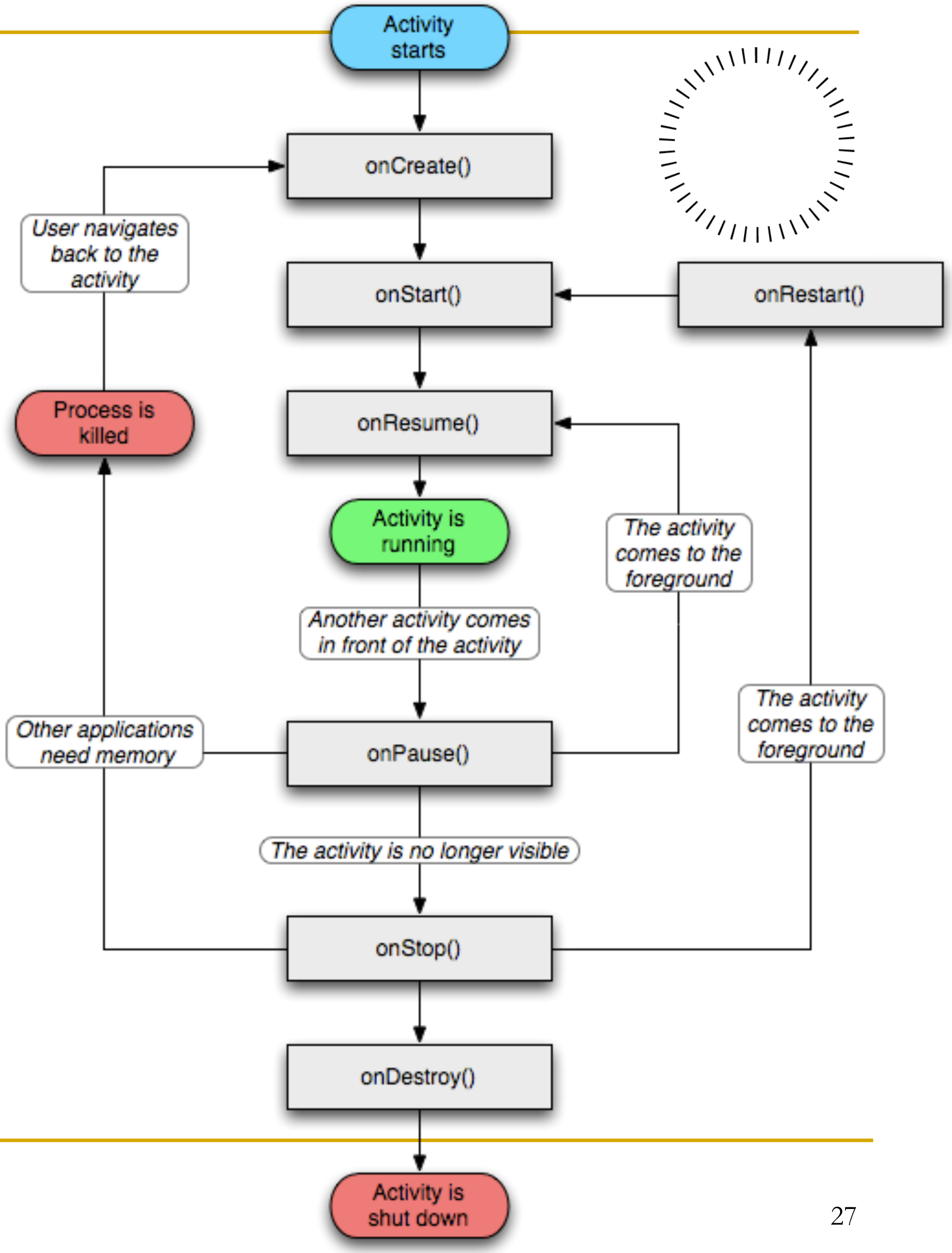


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Quick exercise

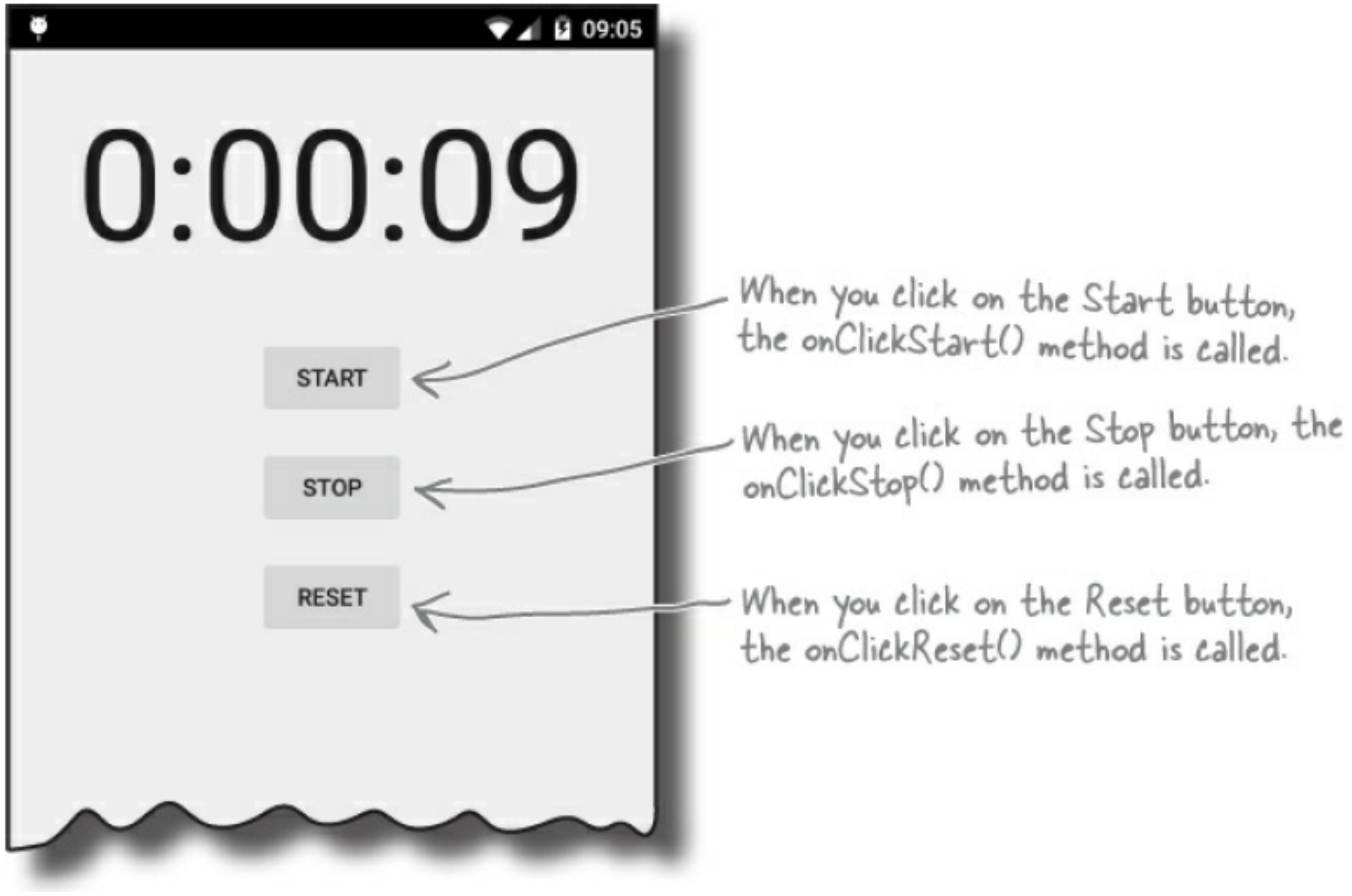
- What event handlers will definitely be called when an activity (and its UI) is pushed to the background and later made visible again?

  - A. onCreate()
  - B. onRestart()
  - C. onStart()
  - D. onStop()
  - E. onResume()

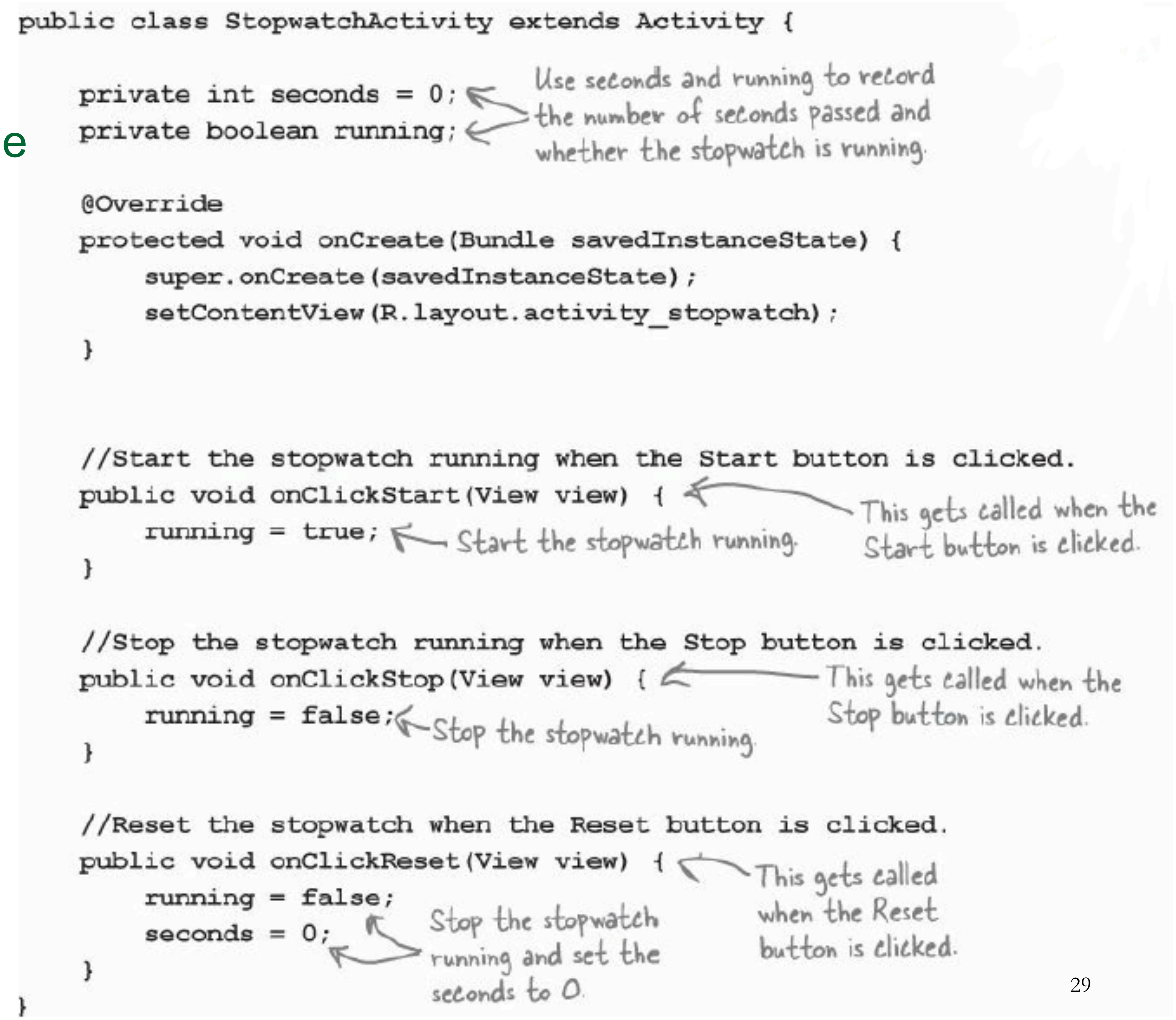

Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Case study – working with activity life cycle

- Develop a stop watch app.


09:05
0:00:09
START
STOP
RESET
When you click on the Start button, the onClickStart() method is called.
When you click on the Stop button, the onClickStop() method is called.
When you click on the Reset button, the onClickReset() method is called.

Java code



```
public class StopwatchActivity extends Activity {

    private int seconds = 0;
    private boolean running;

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


    //Start the stopwatch running when the Start button is clicked.
    public void onClickStart(View view) {
        running = true;
    }

    //Stop the stopwatch running when the Stop button is clicked.
    public void onClickStop(View view) {
        running = false;
    }

    //Reset the stopwatch when the Reset button is clicked.
    public void onClickReset(View view) {
        running = false;
        seconds = 0;
    }
}
```

Use seconds and running to record the number of seconds passed and whether the stopwatch is running.

This gets called when the Start button is clicked.

Start the stopwatch running.

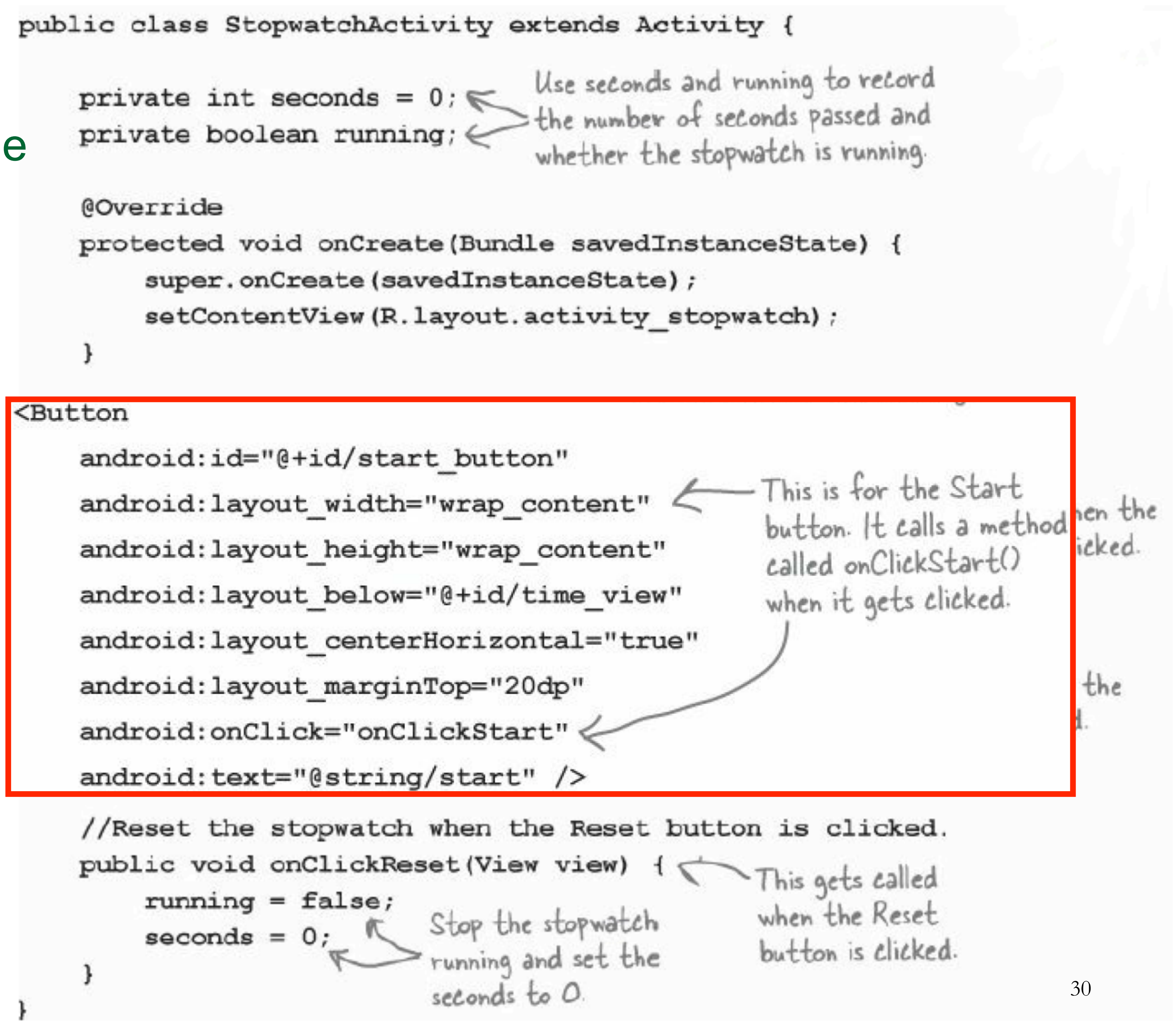
This gets called when the Stop button is clicked.

Stop the stopwatch running.

This gets called when the Reset button is clicked.

Stop the stopwatch running and set the seconds to 0.

Java code



```
public class StopwatchActivity extends Activity {

    private int seconds = 0;
    private boolean running;

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

<Button
    android:id="@+id/start_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/time_view"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="20dp"
    android:onClick="onClickStart"
    android:text="@string/start" />

    //Reset the stopwatch when the Reset button is clicked.
    public void onClickReset(View view) {
        running = false;
        seconds = 0;
    }
}
```

Use seconds and running to record the number of seconds passed and whether the stopwatch is running.

This is for the Start button. It calls a method called onClickStart() when it gets clicked.

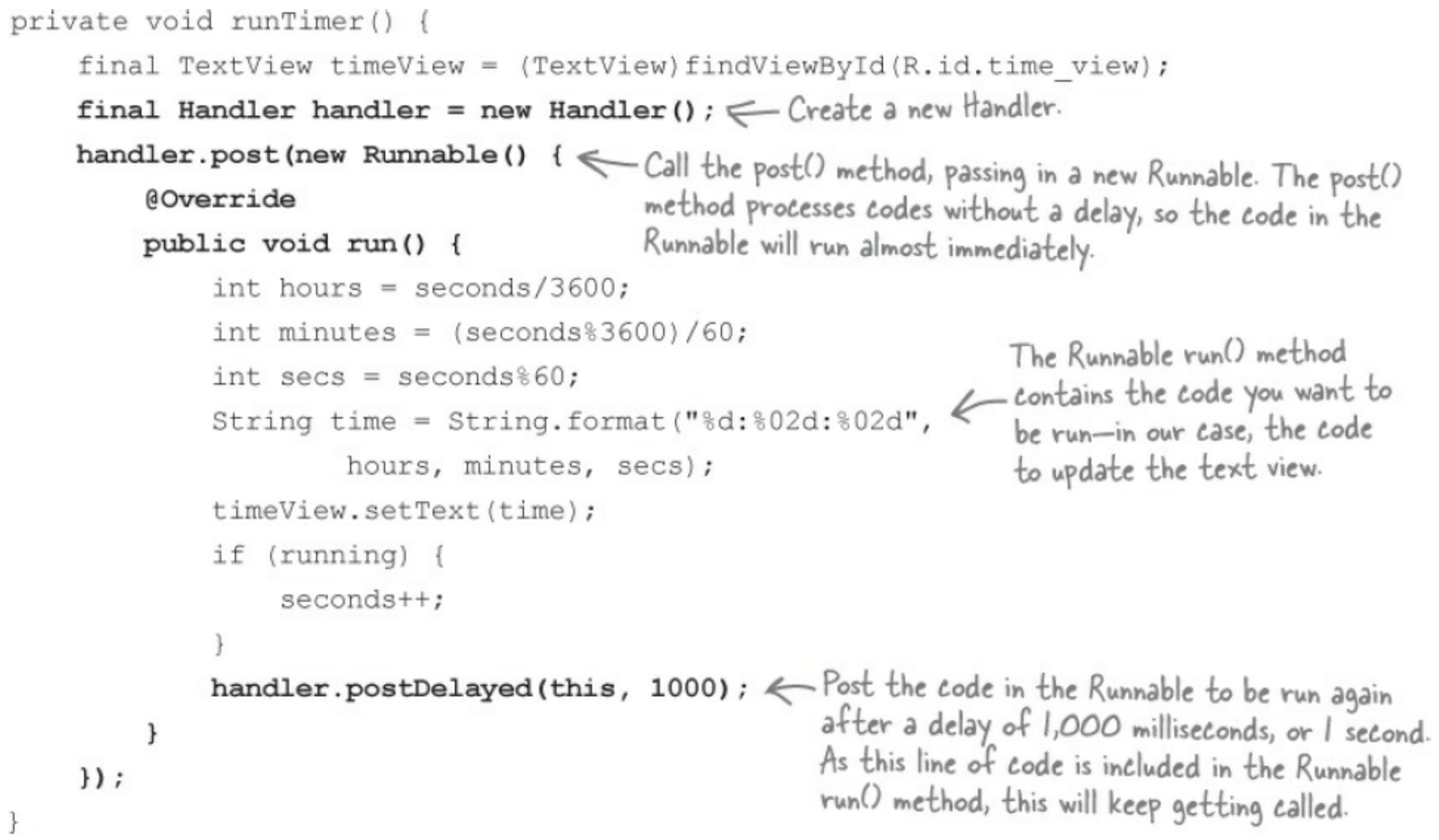
hen the icked.

the d.

This gets called when the Reset button is clicked.

Stop the stopwatch running and set the seconds to 0.

Controlling repeated update to clock counter



```
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.time_view);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
            int hours = seconds/3600;
            int minutes = (seconds%3600)/60;
            int secs = seconds%60;
            String time = String.format("%d:%02d:%02d",
                    hours, minutes, secs);
            timeView.setText(time);
            if (running) {
                seconds++;
            }
            handler.postDelayed(this, 1000);
        }
    });
}
```

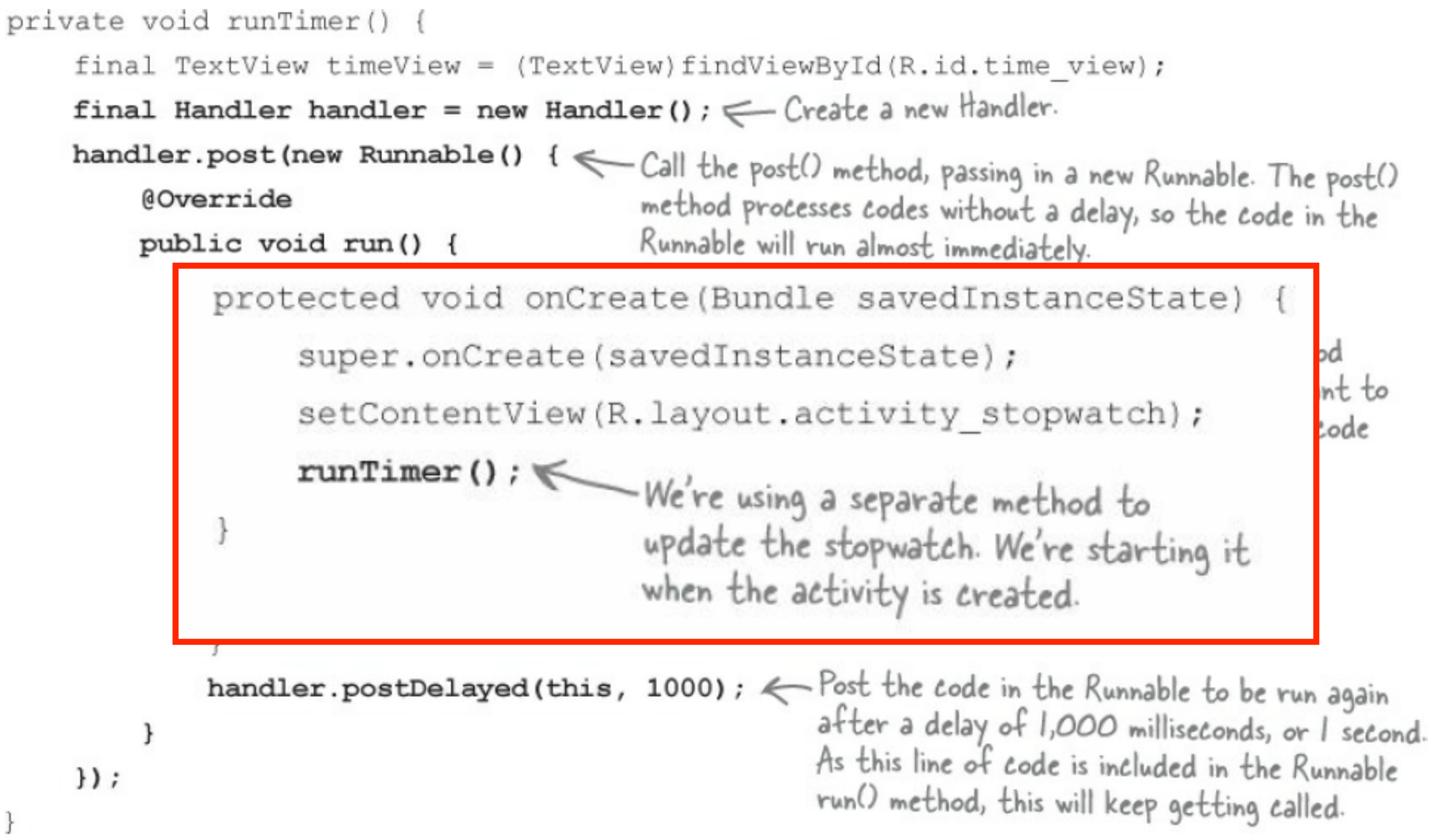
Create a new Handler.

Call the post() method, passing in a new Runnable. The post() method processes codes without a delay, so the code in the Runnable will run almost immediately.

The Runnable run() method contains the code you want to be run—in our case, the code to update the text view.

Post the code in the Runnable to be run again after a delay of 1,000 milliseconds, or 1 second. As this line of code is included in the Runnable run() method, this will keep getting called.

Controlling repeated update to clock counter

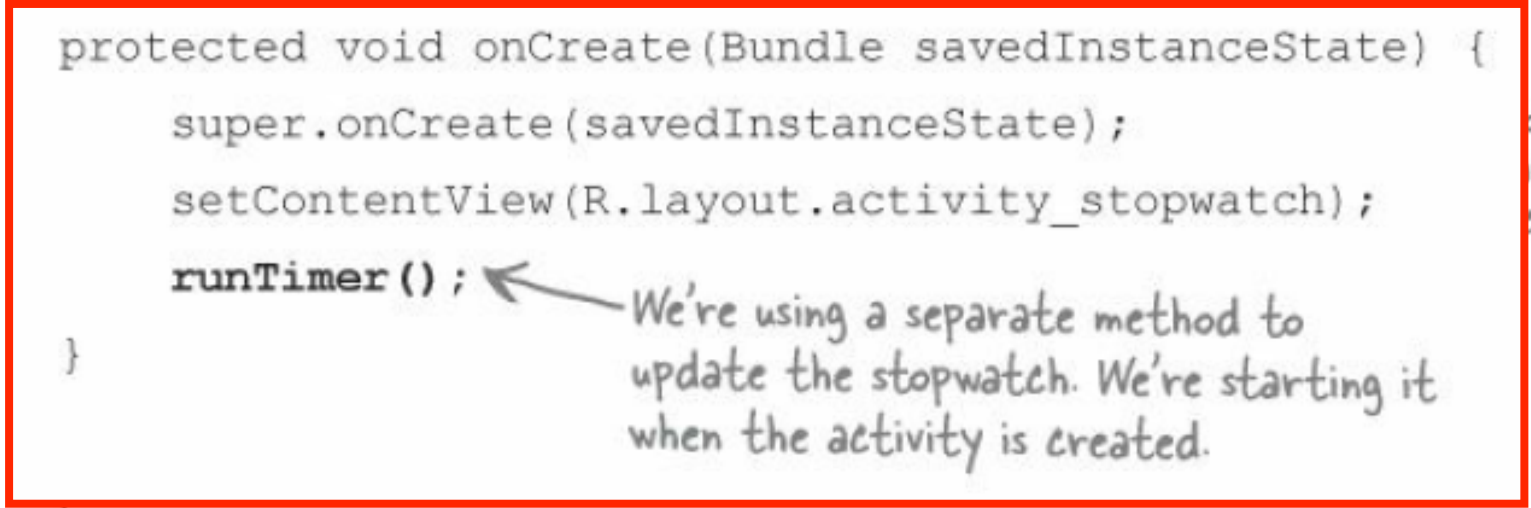


```
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.time_view);
    final Handler handler = new Handler();
    handler.post(new Runnable() {
        @Override
        public void run() {
```

Create a new Handler.

Call the post() method, passing in a new Runnable. The post() method processes codes without a delay, so the code in the Runnable will run almost immediately.

od
nt to
code



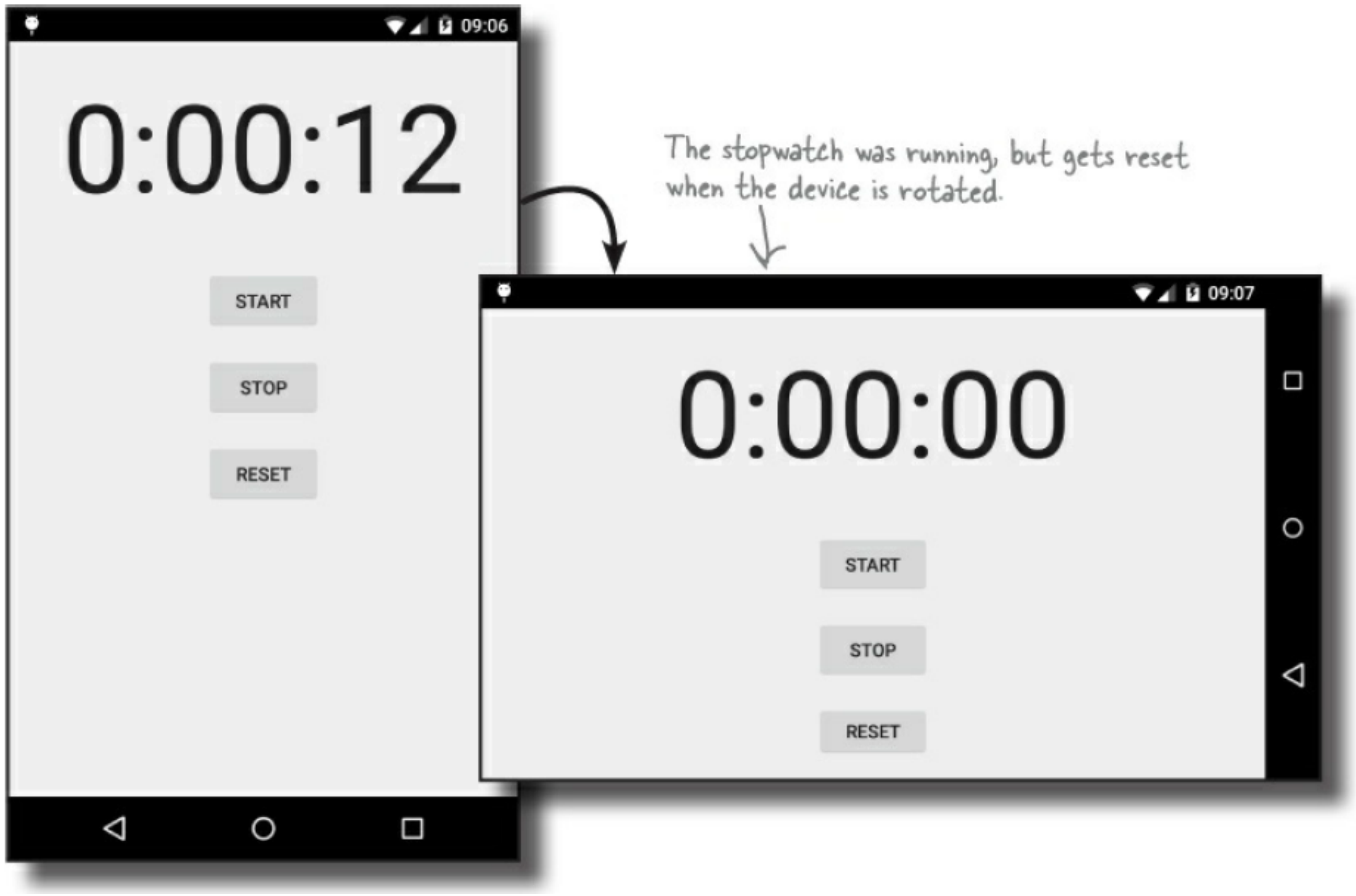
```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_stopwatch);
    runTimer();
}
```

We're using a separate method to update the stopwatch. We're starting it when the activity is created.

```
            handler.postDelayed(this, 1000);
        }
    });
}
```

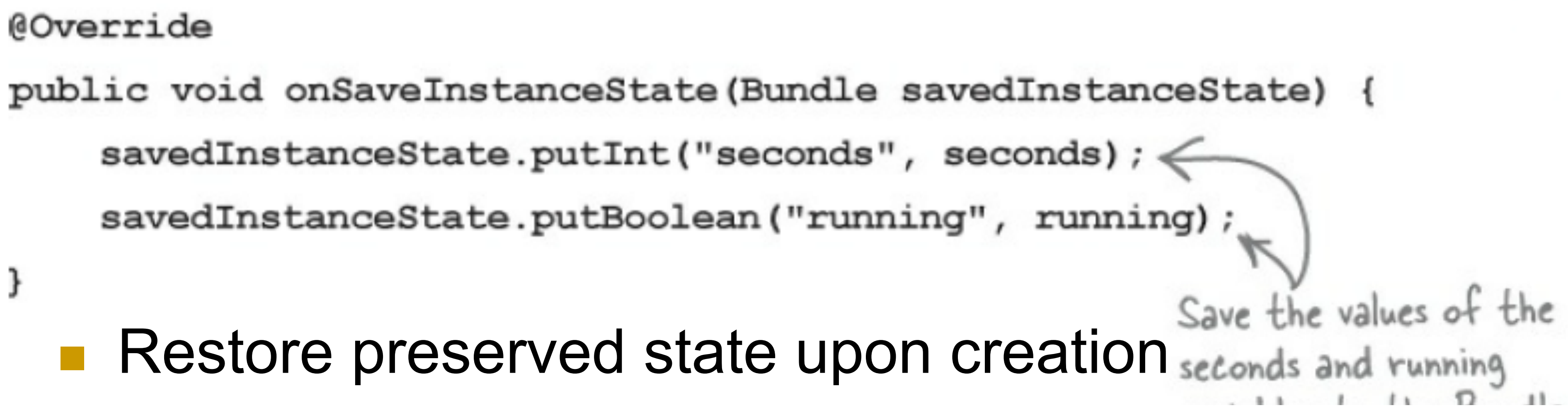
Post the code in the Runnable to be run again after a delay of 1,000 milliseconds, or 1 second. As this line of code is included in the Runnable run() method, this will keep getting called.

Test our stop watch app


09:06
0:00:12
START
STOP
RESET
The stopwatch was running, but gets reset when the device is rotated.
09:07
0:00:00
START
STOP
RESET

Keep the watch going after orientation change

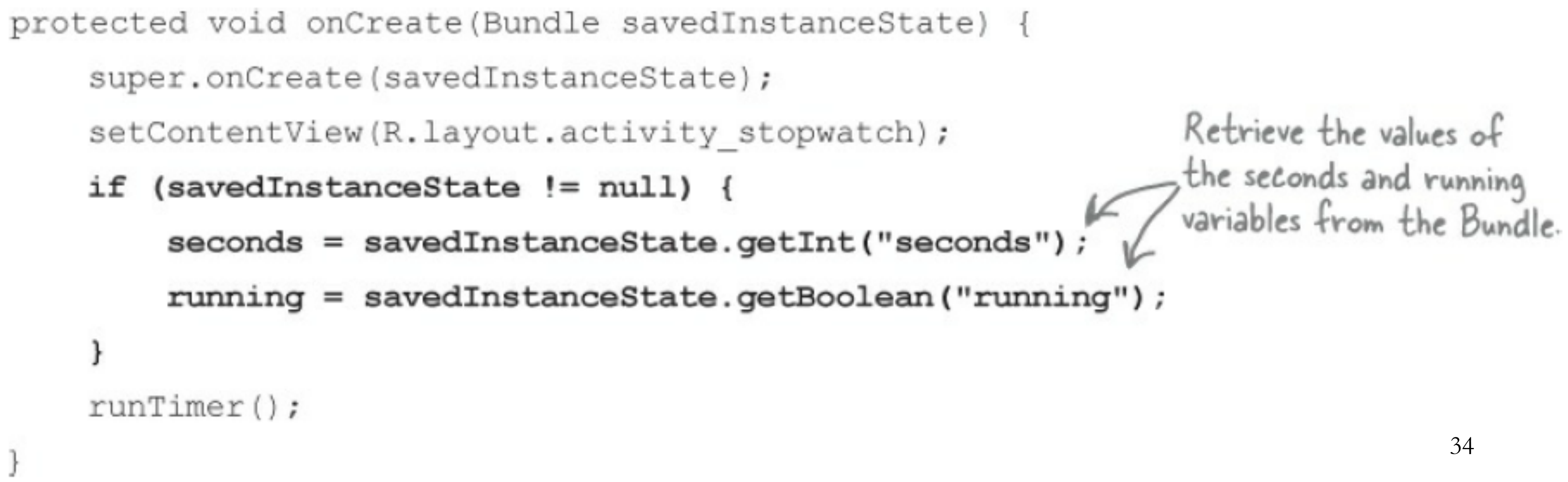
- Keep state info before an activity is destroyed.



```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    savedInstanceState.putInt("seconds", seconds);
    savedInstanceState.putBoolean("running", running);
}
```

Save the values of the seconds and running

- Restore preserved state upon creation



```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_stopwatch);
    if (savedInstanceState != null) {
        seconds = savedInstanceState.getInt("seconds");
        running = savedInstanceState.getBoolean("running");
    }
    runTimer();
}
```

Retrieve the values of the seconds and running variables from the Bundle.

Questions to ponder

- Why does Android want to re-create an activity just because I rotated the screen?
- Why doesn't Android store every instance variable automatically? Why do I have to write all of that code myself?
- How can we stop the watch when the app is no longer in the foreground?

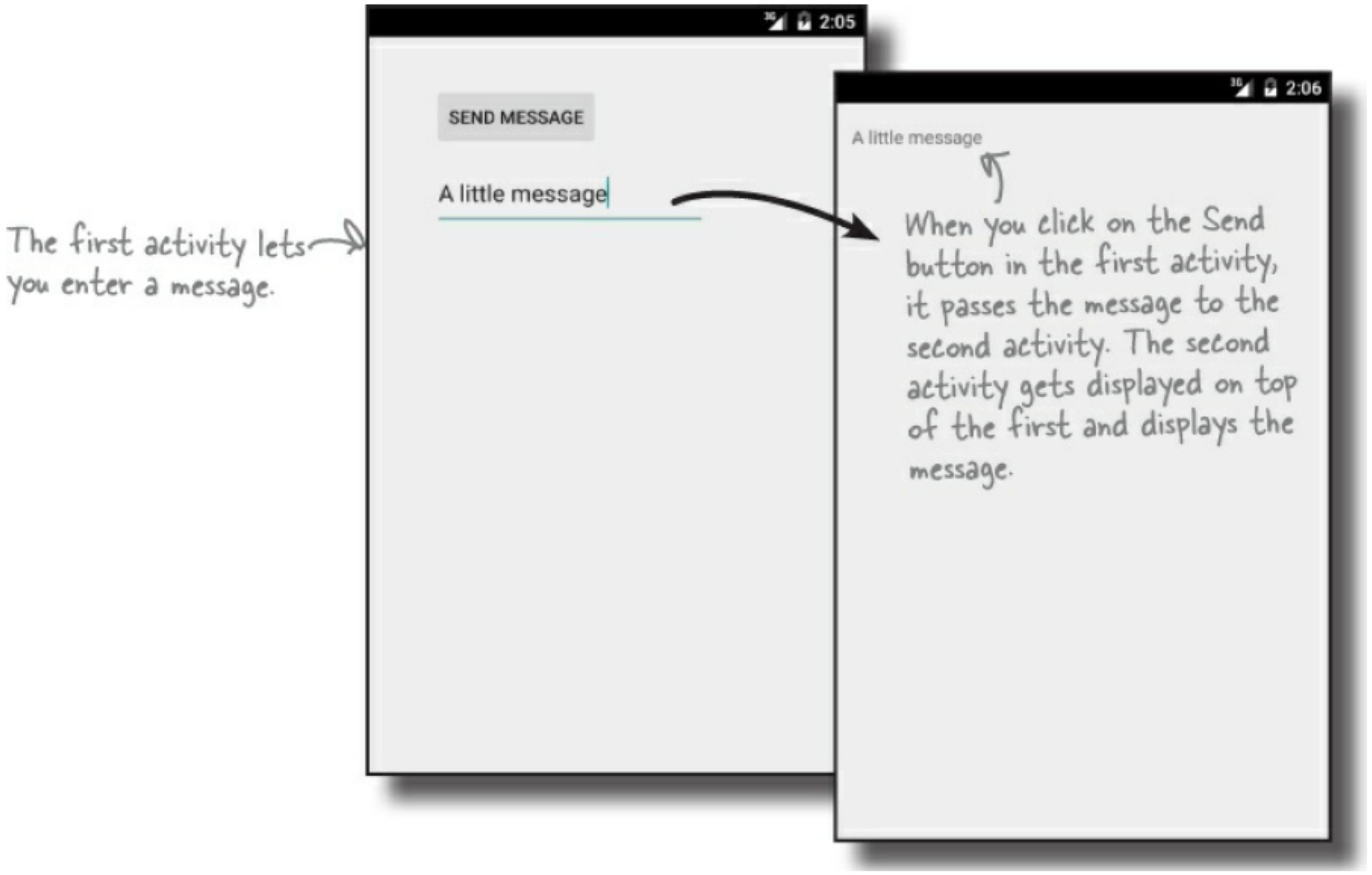
Important things to remember

- Only the main thread can update the user interface.
- A device configuration change results in the activity being destroyed and re-created.
- Your activity inherits the lifecycle methods from the Android Activity class.
  - If you override any of these methods, you need to call up to the method in the superclass.

Intent

- Intents support message-passing within or across applications.
- Main usage
  - Explicitly start a particular Service or Activity using its full class name
  - Start an Activity or Service to perform an action with (or on) a particular piece of data
  - Broadcast that an event has occurred

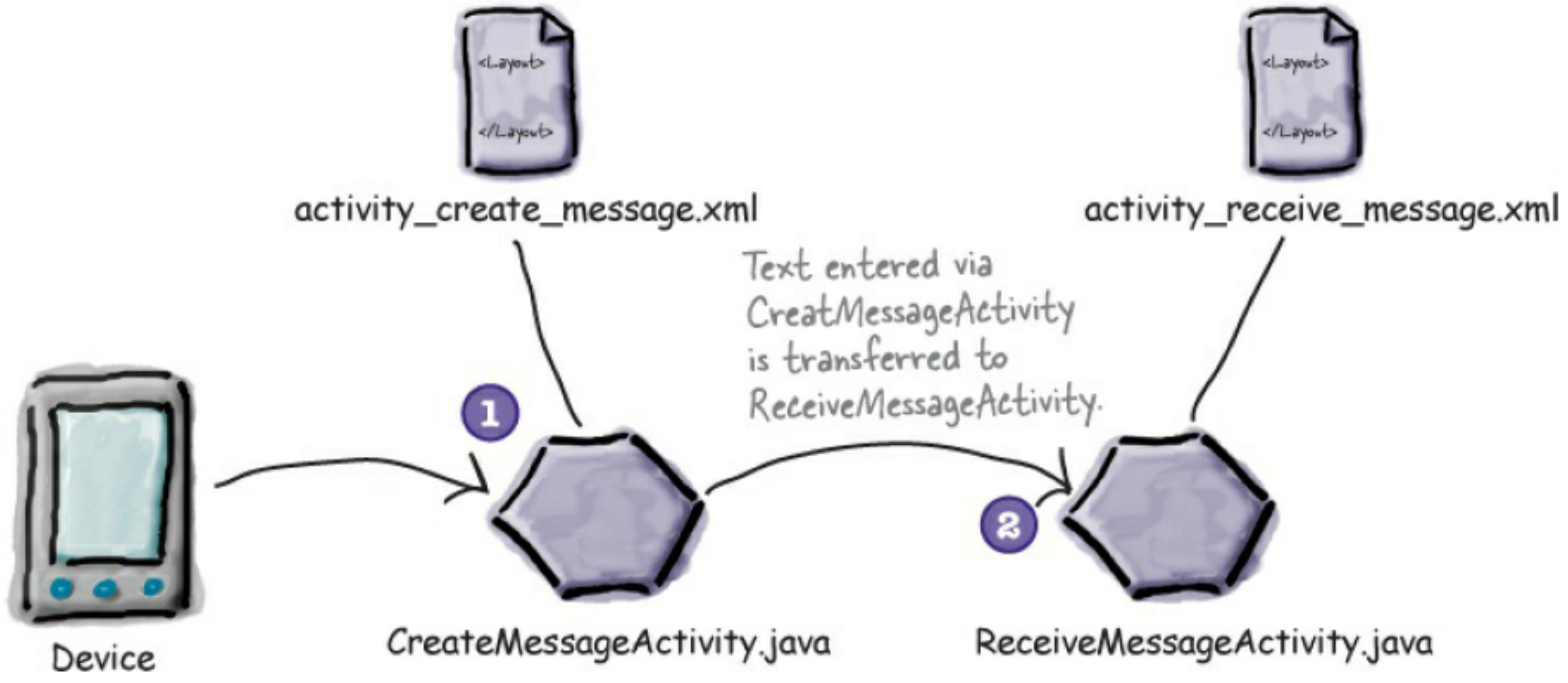
Case study – activity and intents



2:05
SEND MESSAGE
A little message
2:06
A little message
The first activity lets you enter a message.
When you click on the Send button in the first activity, it passes the message to the second activity. The second activity gets displayed on top of the first and displays the message.

Project structure

- An app with two activities.



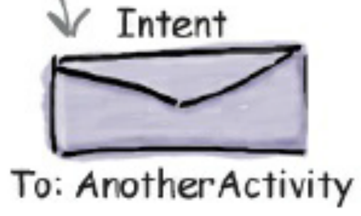
<Layout>
</Layout>
activity_create_message.xml
<Layout>
</Layout>
activity_receive_message.xml
Text entered via CreatMessageActivity is transferred to ReceiveMessageActivity.
1
2
Device
CreateMessageActivity.java
ReceiveMessageActivity.java

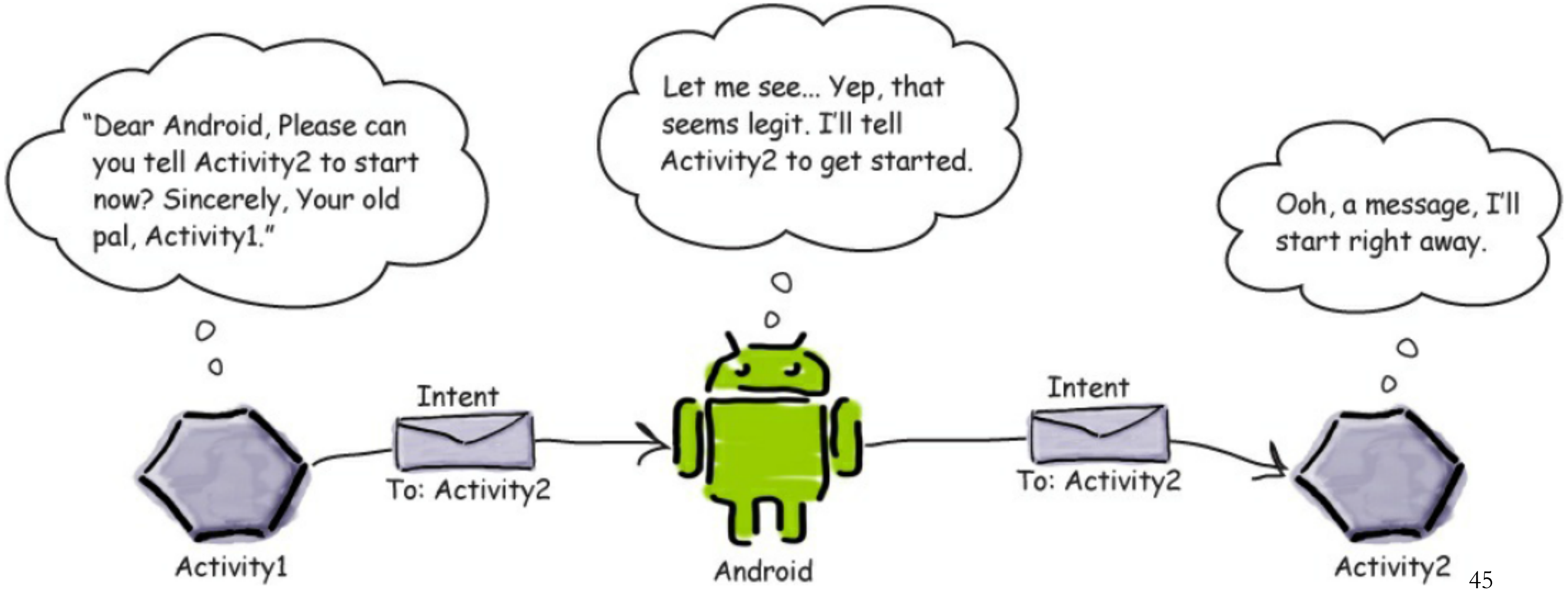
Use intent to chain activities together

```
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..



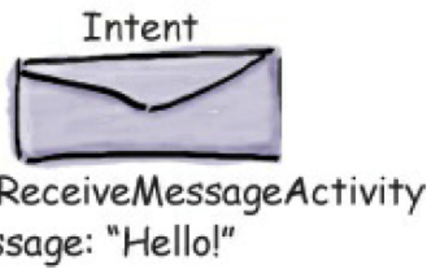

"Dear Android, Please can you tell Activity2 to start now? Sincerely, Your old pal, Activity1."
Let me see... Yep, that seems legit. I'll tell Activity2 to get started.
Ooh, a message, I'll start right away.
Intent
To: Activity2
Intent
To: Activity2
Activity1
Android
Activity2

Pass data through intent

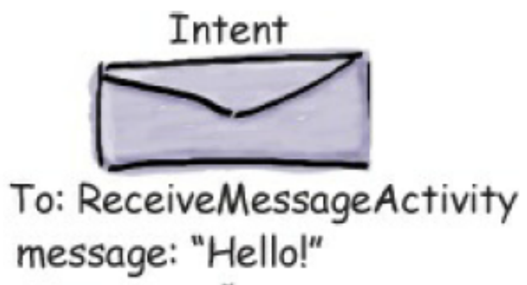
- Add information to an intent

putExtra() lets you put extra information in the message you're sending.

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



- Retrieve information from an intent



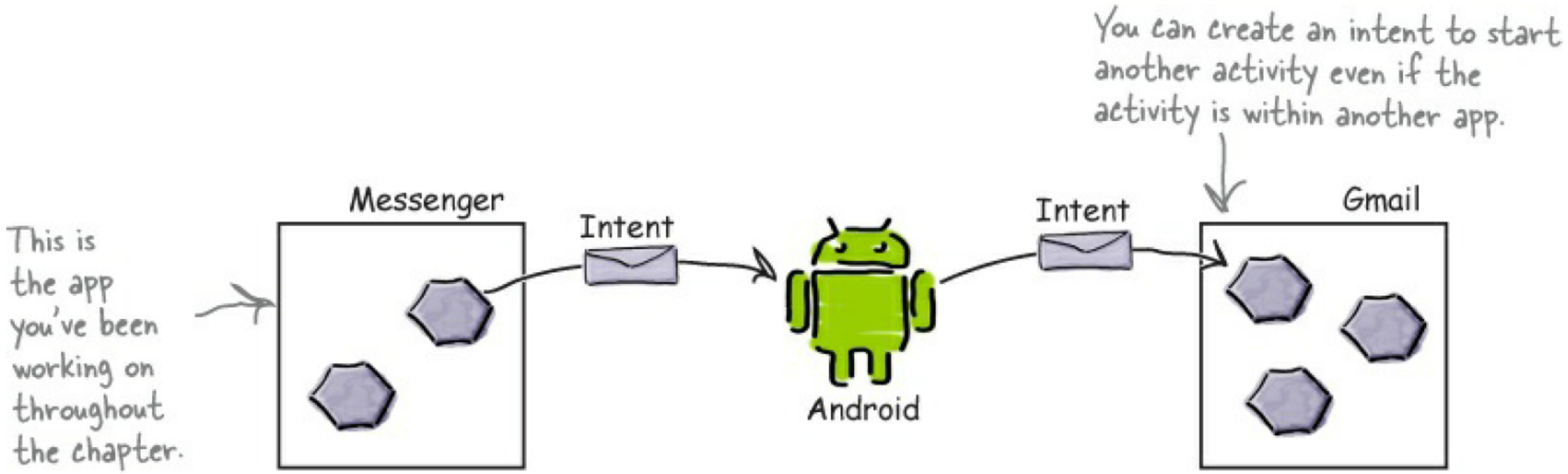
```
Intent intent = getIntent();
String string = intent.getStringExtra("message");
```

Get the intent.

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

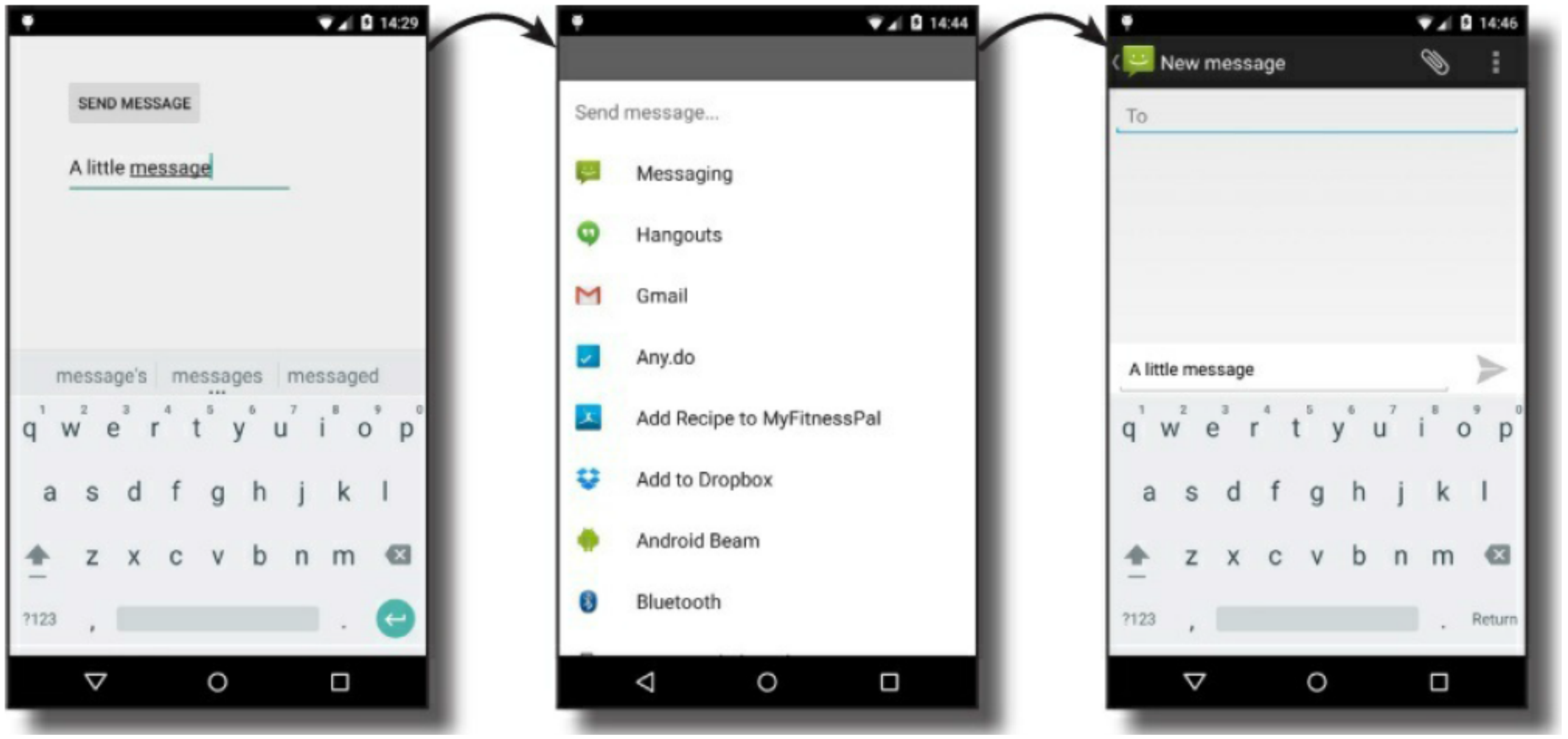
Activate third-party activities

- Intent can start activity in other apps.


You can create an intent to start another activity even if the activity is within another app.
This is the app you've been working on throughout the chapter.
Messenger
Intent
Android
Intent
Gmail

  - Create an intent that specifies an action.

Example – send message by email



Things to do …

- Create implicit intent

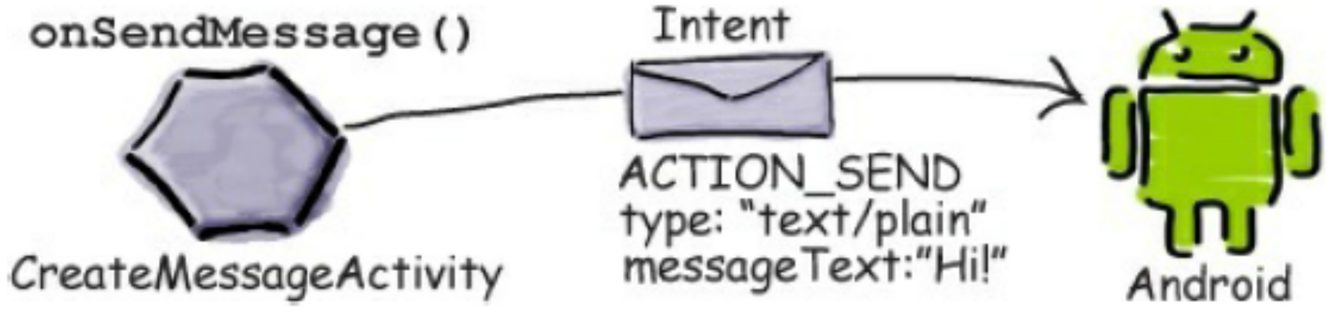
```
Intent intent = new Intent(Intent.ACTION_SEND);
```

- Add extra information

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

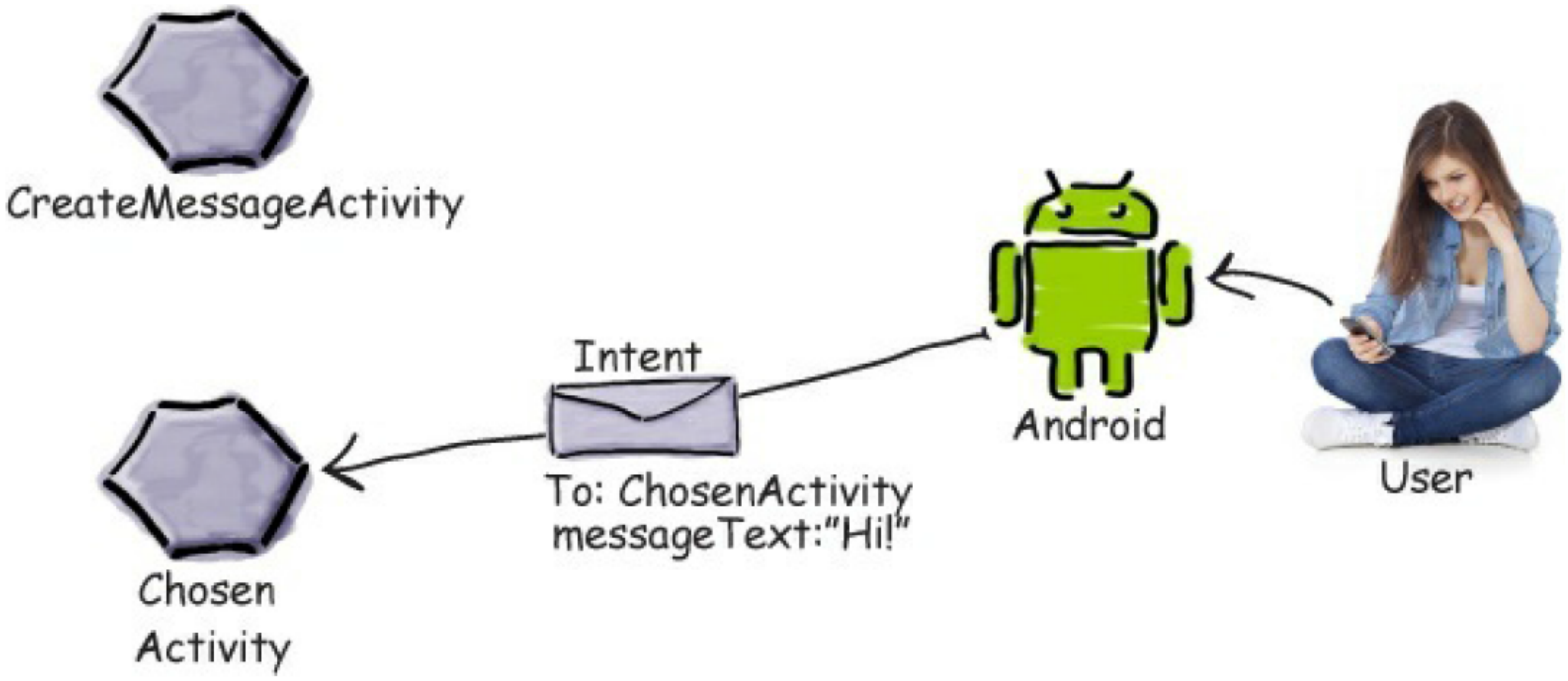
These attributes relate to Intent.ACTION_SEND. They're not relevant for all actions.

- Pass intent to Android



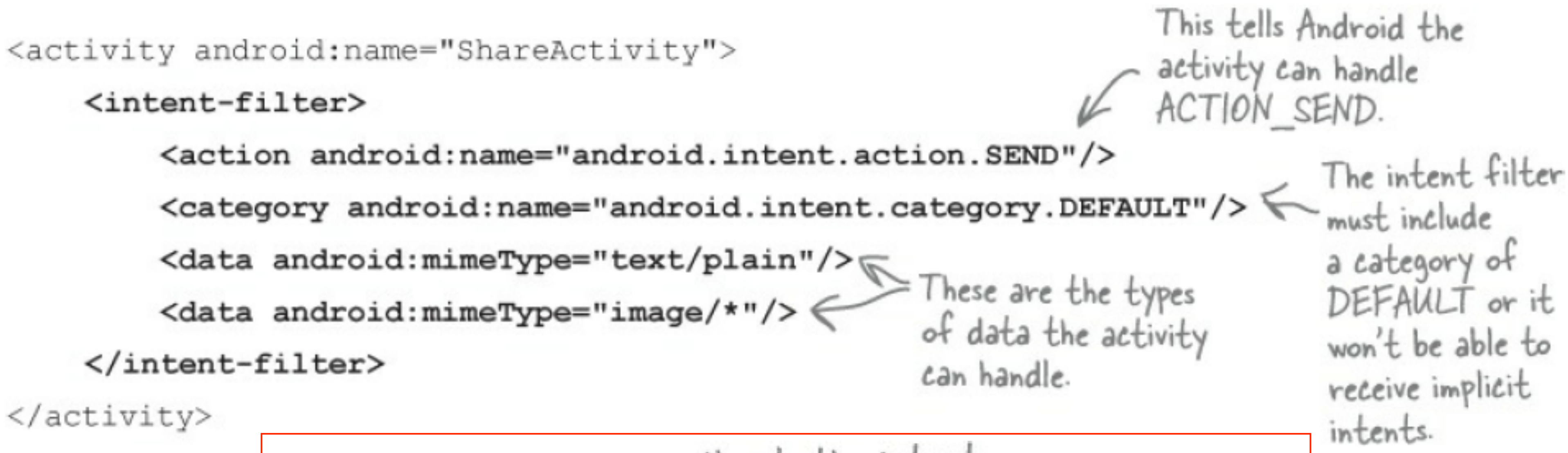
Fire an implicit intent

- User chooses an activity



Intent filters

- The intent filter tells Android which activities can handle which actions

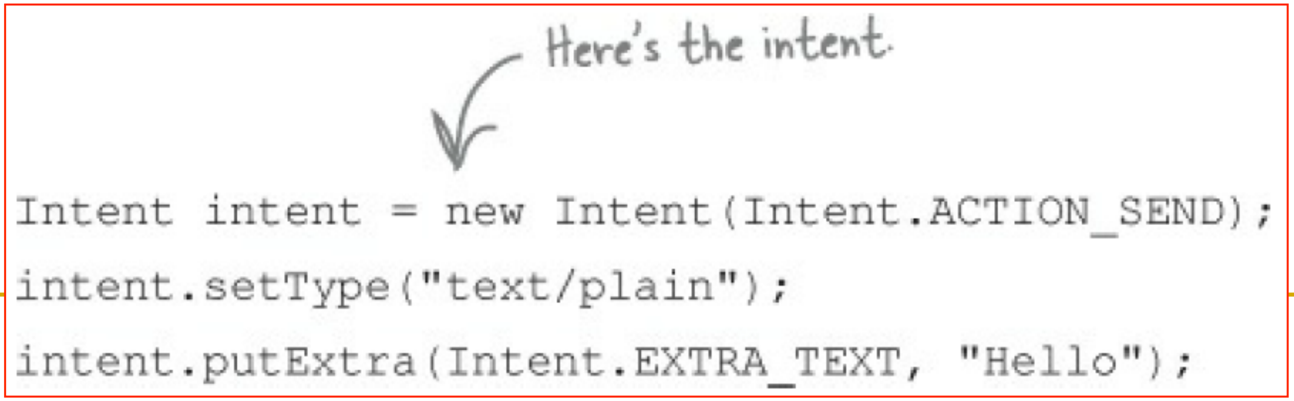


```
<activity android:name="ShareActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
        <data android:mimeType="image/*"/>
    </intent-filter>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



Here's the intent.

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

- Which activity can be used to send a plaintext message?

```
<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>
```