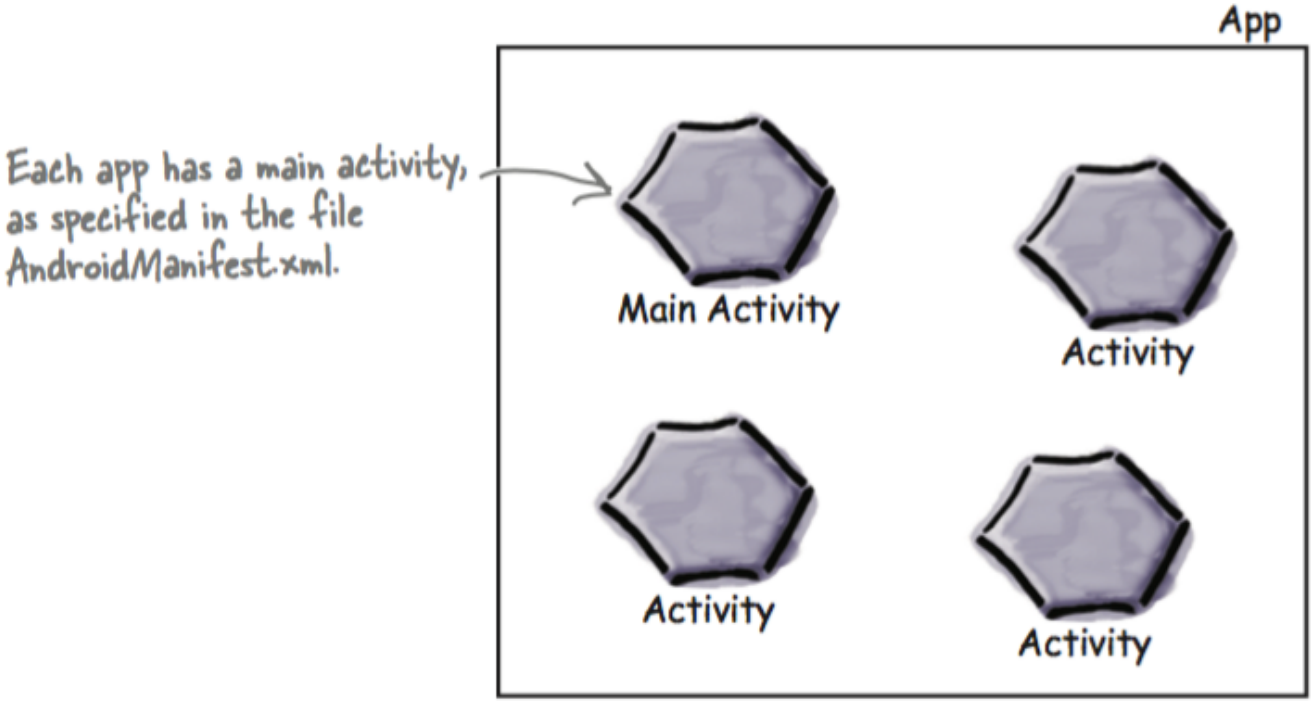
4
El ciclo de vida de una actividad

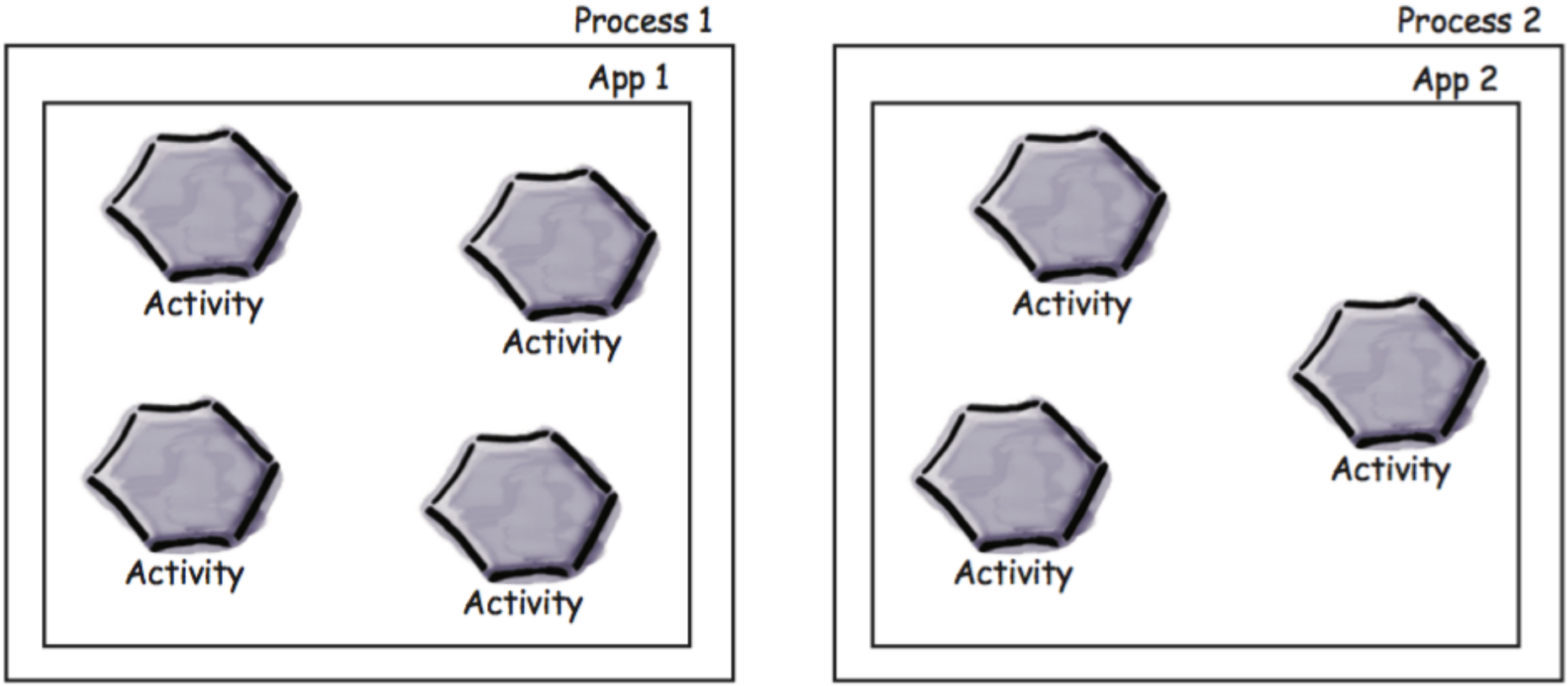
Como funcionan las actividades

An app is a collection of activities, layouts, and other resources.
One of these activities is the main activity for the app.



Como funcionan las actividades

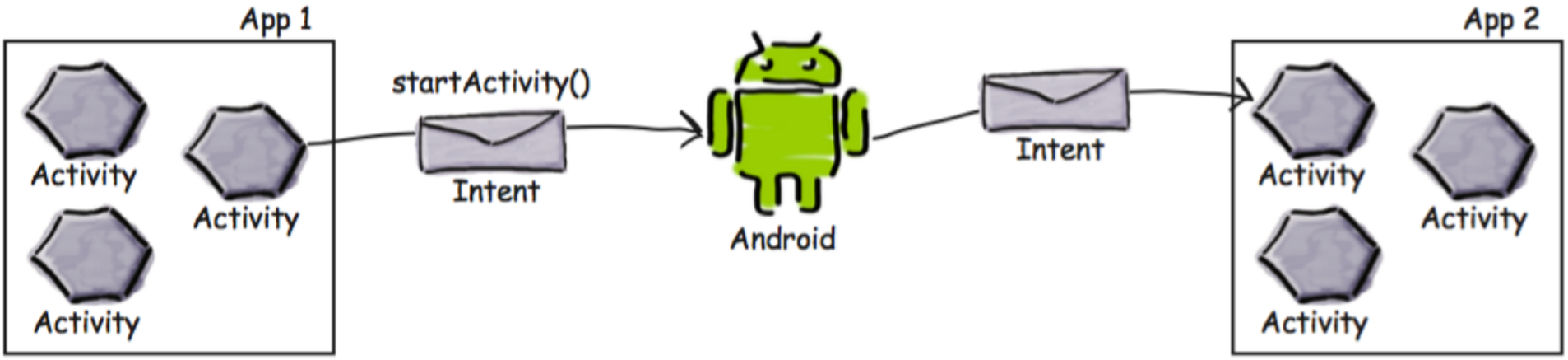
By default, each app runs within its own process.
This helps keep your apps safe and secure. You can read more about this in Appendix i (which covers the Android runtime, or ART) at the back of this book.



Como funcionan las actividades

You can start an activity in another application by passing an intent with startActivity().

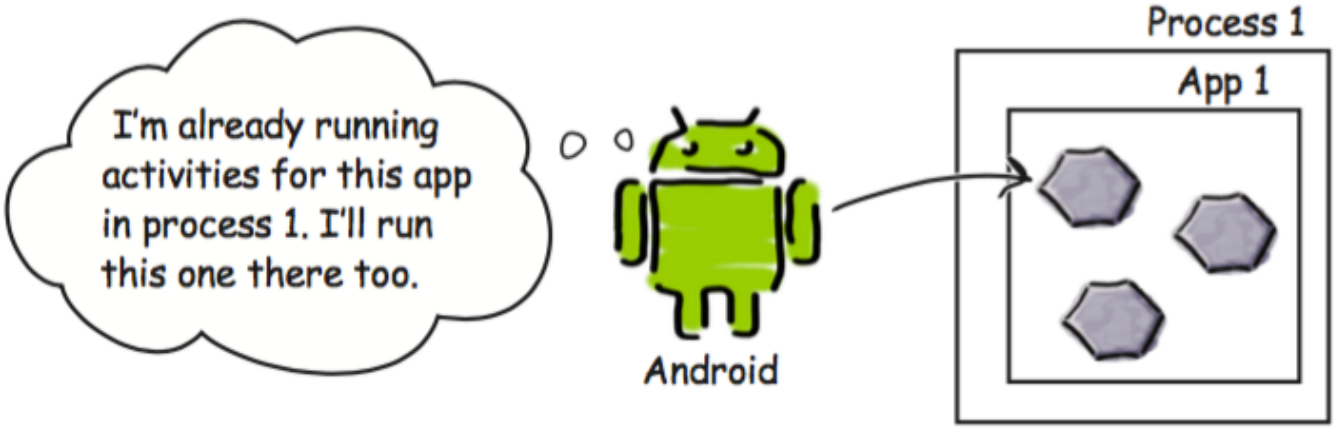
The Android system knows about all the installed apps and their activities, and uses the intent to start the correct activity.



Como funcionan las actividades

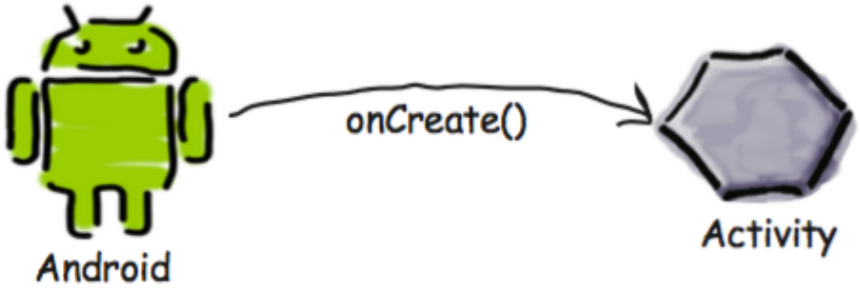
When an activity needs to start, Android checks if there's already a process for that app.

If one exists, Android runs the activity in that process. If one doesn't exist, Android creates one.



When Android starts an activity, it calls its onCreate() method.

`onCreate()` is always run whenever an activity gets created.



La aplicación Stopwatch (cronómetro)

This is the number of seconds.

When you click this button, the seconds begin to increment.

When you click this button, the seconds stop incrementing.

When you click this button, the seconds tally goes back to 0.

activity_stopwatch.xml

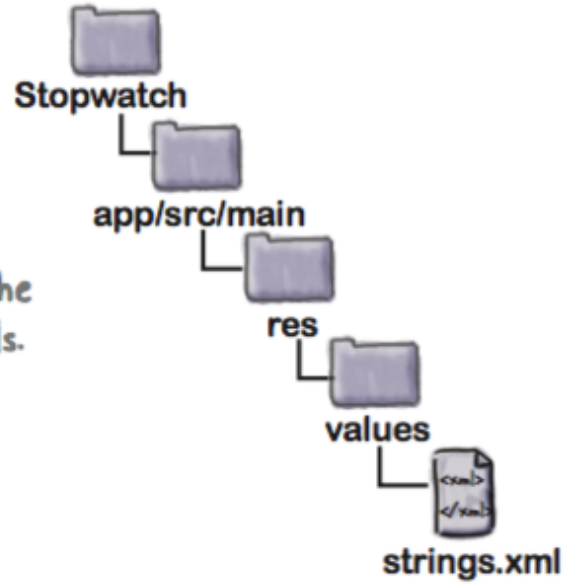
The app is composed of one activity and one layout.

StopwatchActivity.java

EL archivo strings.xml de Stopwatch

```
...
    <string name="start">Start</string>
    <string name="stop">Stop</string>
    <string name="reset">Reset</string>
...
```

These are the button labels.

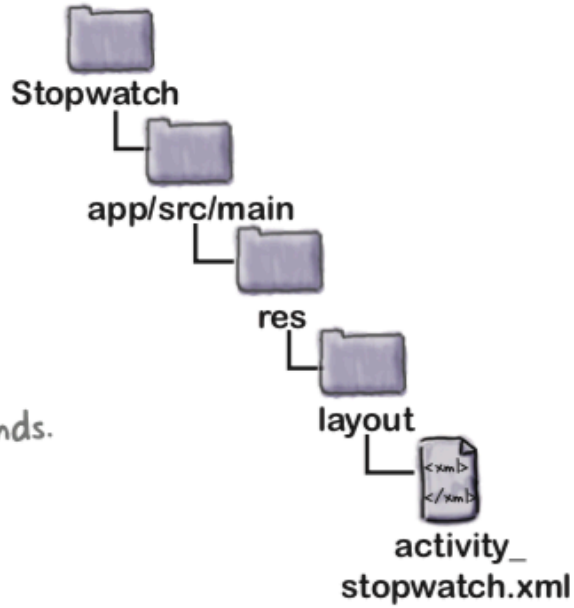


El código de Stopwatch

```
<?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:orientation="vertical"
    android:padding="16dp"
    tools:context=".StopwatchActivity">

    <TextView
        android:id="@+id/time_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"
        android:textAppearance="@android:style/TextAppearance.Large"
        android:textSize="56sp" />
```

We'll use this text view to display the number of seconds.

These attributes make the stopwatch timer nice and big.



El código de Stopwatch

```
<Button
    android:id="@+id/start_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="20dp"
    android:onClick="onClickStart"
    android:text="@string/start" />

<Button
    android:id="@+id/stop_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="8dp"
    android:onClick="onClickStop"
    android:text="@string/stop" />

<Button
    android:id="@+id/reset_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:layout_marginTop="8dp"
    android:onClick="onClickReset"
    android:text="@string/reset" />
</LinearLayout>
```

This code is for the Start button.

When it gets clicked, the Start button calls the onClickStart() method.

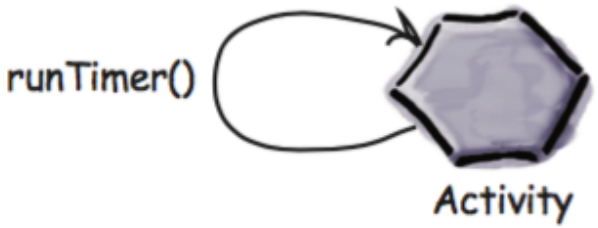
This is for the Stop button.

When it gets clicked, the Stop button calls the onClickStop() method.

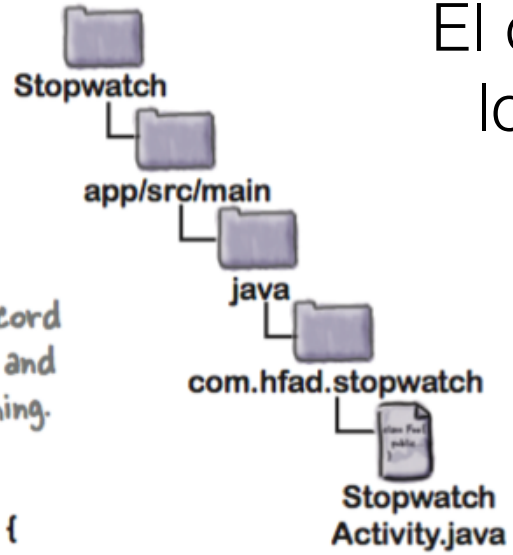
This is for the Reset button.

When it gets clicked, the Reset button calls the onClickReset() method.

Como funciona el código de la actividad



El código para los botones




```
package com.hfad.stopwatch;

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

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.

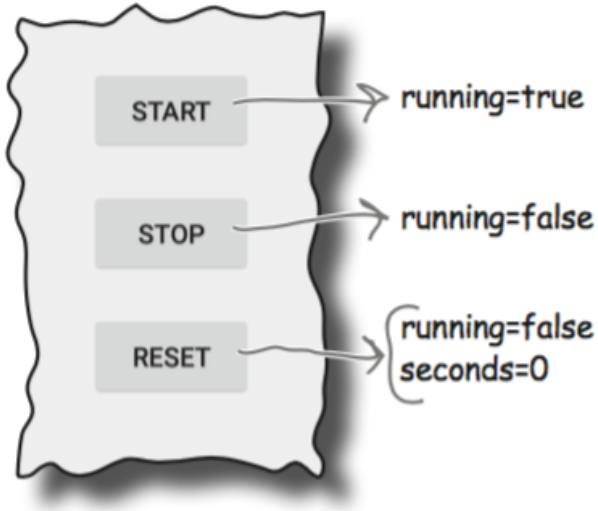
Start the stopwatch running.

This gets called when the Stop button is clicked.

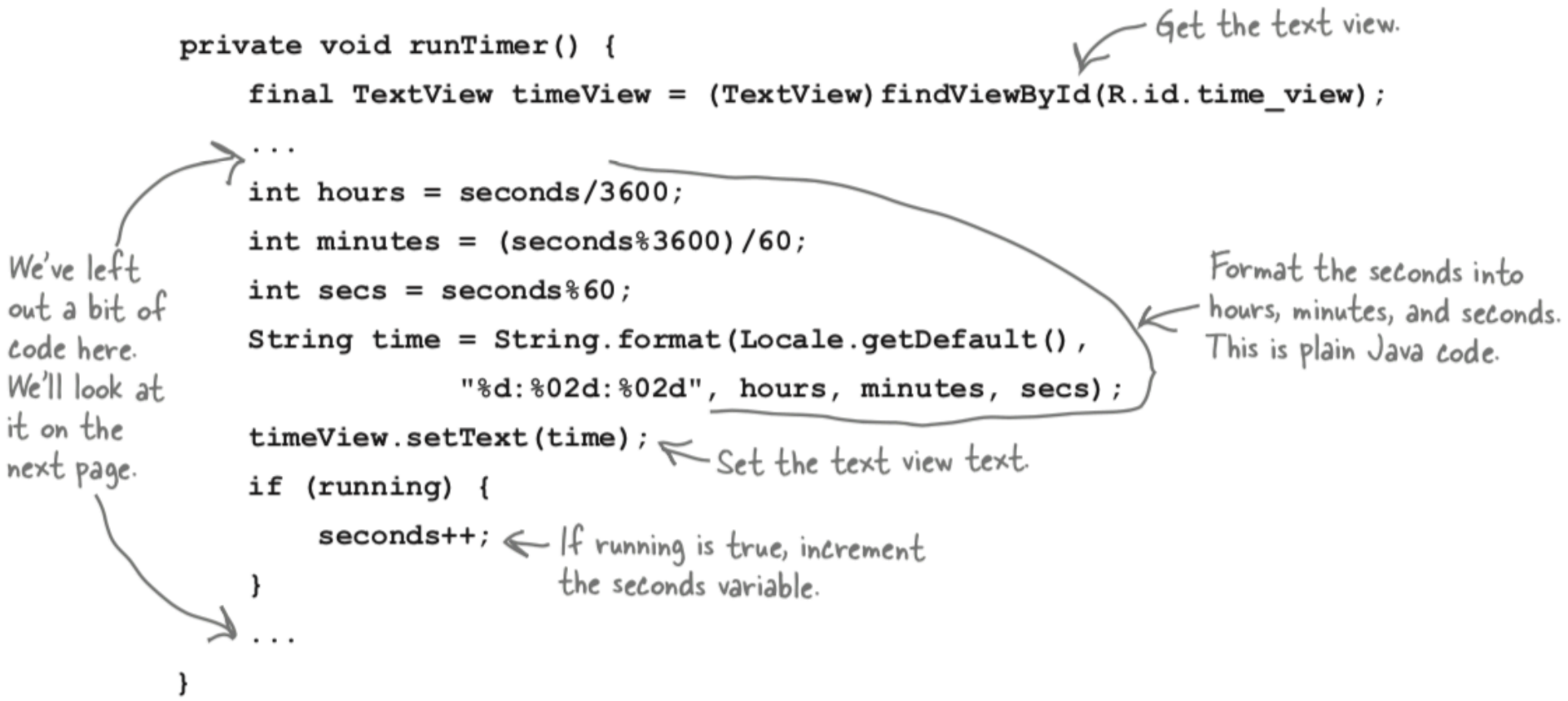
Stop the stopwatch running.

This gets called when the Reset button is clicked.

Stop the stopwatch running and set the seconds to 0.



El método runTimer()



El método post() publica código que debe ser ejecutado tan pronto como sea posible.

```
final Handler handler = new Handler();
handler.post(Runnable);
```
← You put the code you want to run in the Handler's run() method.

El método postDelayed() funciona como post(), excepto que publica código que deberá ser ejecutado en el futuro.

```
final Handler handler = new Handler();
handler.postDelayed(Runnable, long);
```
← Use this method to delay running code by a specified number of milliseconds.

El método runTimer() completo

```
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.

```
protected void onCreate(Bundle savedInstanceState) {
    ...
    runTimer();
}
```

```
package com.hfad.stopwatch;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import java.util.Locale;
import android.os.Handler;
import android.widget.TextView;

public class StopwatchActivity extends Activity {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;

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

    //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;
    }
```

We're using these extra classes so we need to import them.

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

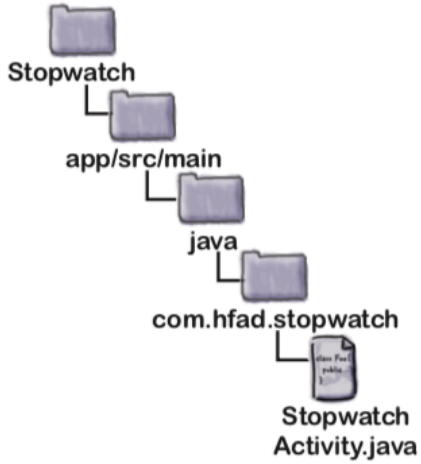
We're using a separate method to update the stopwatch. We're starting it when the activity is created.

This gets called when the Start button is clicked.

Start the stopwatch.

This gets called when the Stop button is clicked.

Stop the stopwatch.



El código de completo de StopWatch

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

//Sets the number of seconds on the timer.
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(Locale.getDefault(),
                    "%d:%02d:%02d", hours, minutes, secs);
            timeView.setText(time);
            if (running) {
                seconds++;
            }
            handler.postDelayed(this, 1000);
        }
    });
}
}
```

This gets called when the Reset button is clicked.

Stop the stopwatch and set the seconds to 0.

Get the text view.

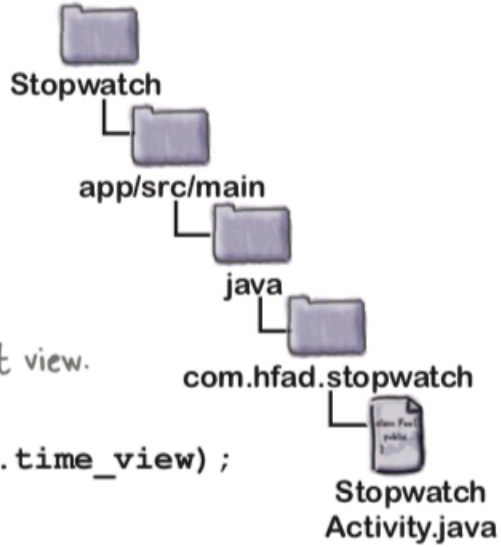
Use a Handler to post code.

Format the seconds into hours, minutes, and seconds.

Set the text view text.

If running is true, increment the seconds variable.

Post the code again with a delay of 1 second.

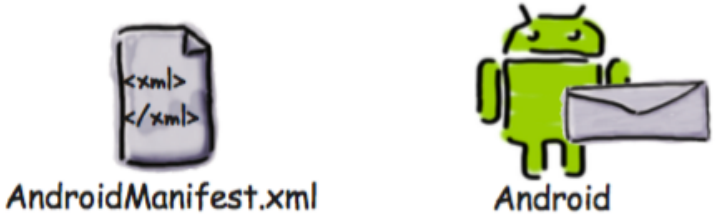


El código completo de StopWatch

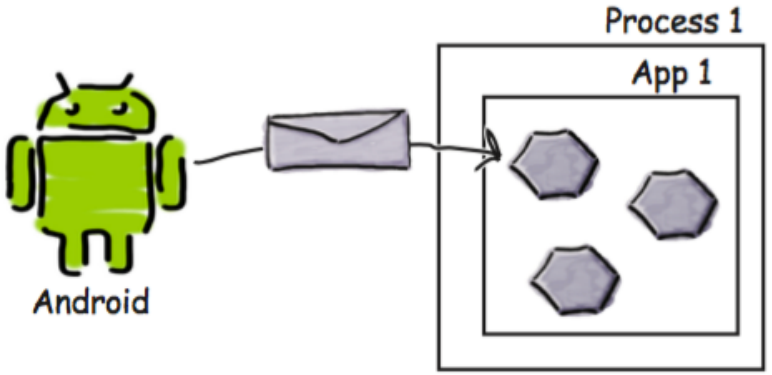
Que pasa al ejecutar la aplicación

1 **The user decides she wants to run the app.**
She clicks on the icon for the app on her device.

2 **The AndroidManifest.xml file for the app specifies which activity to use as the launch activity.**
An intent is constructed to start this activity using `startActivity(intent)`.

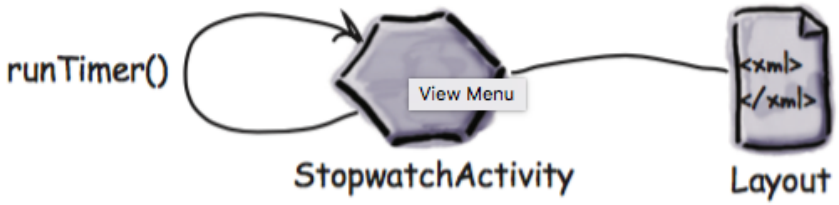


3 **Android checks if there's already a process running for the app, and if not, creates a new process.**
It then creates a new activity object—in this case, for `StopwatchActivity`.

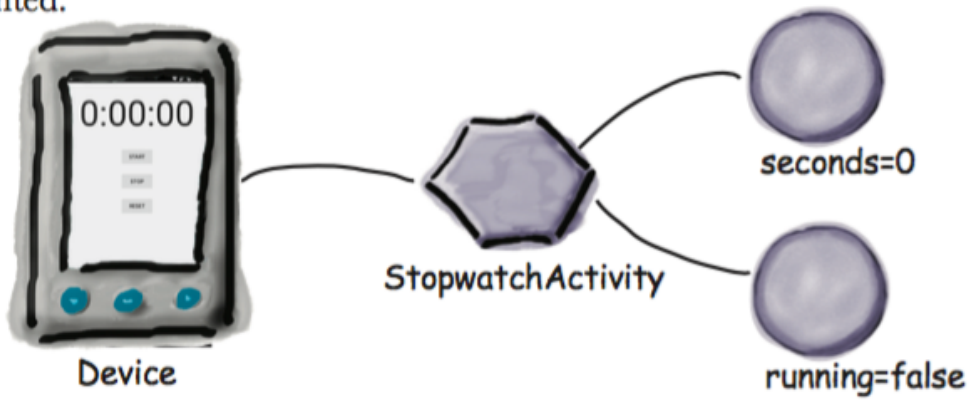


Que pasa al ejecutar la aplicación

4 **The onCreate() method in the activity gets called.**
The method includes a call to `setContentView()`, specifying a layout, and then starts the stopwatch with `runTimer()`.

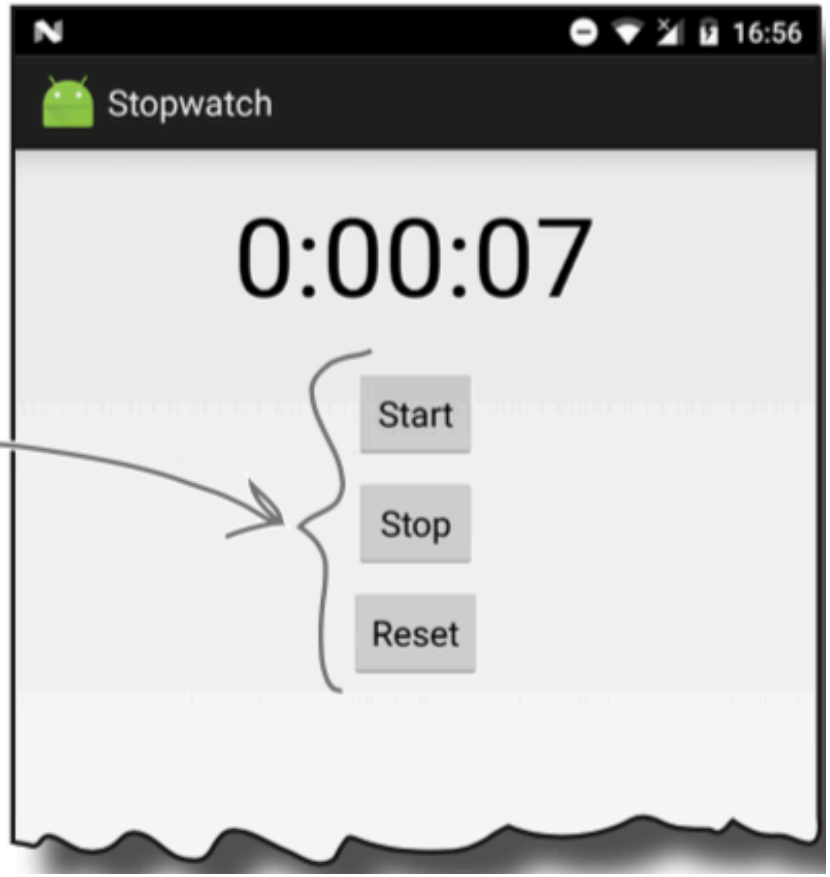


5 **When the onCreate() method finishes, the layout gets displayed on the device.**
The `runTimer()` method uses the `seconds` variable to determine what text to display in the text view, and uses the `running` variable to determine whether to increment the number of seconds. As `running` is initially `false`, the number of seconds isn't incremented.

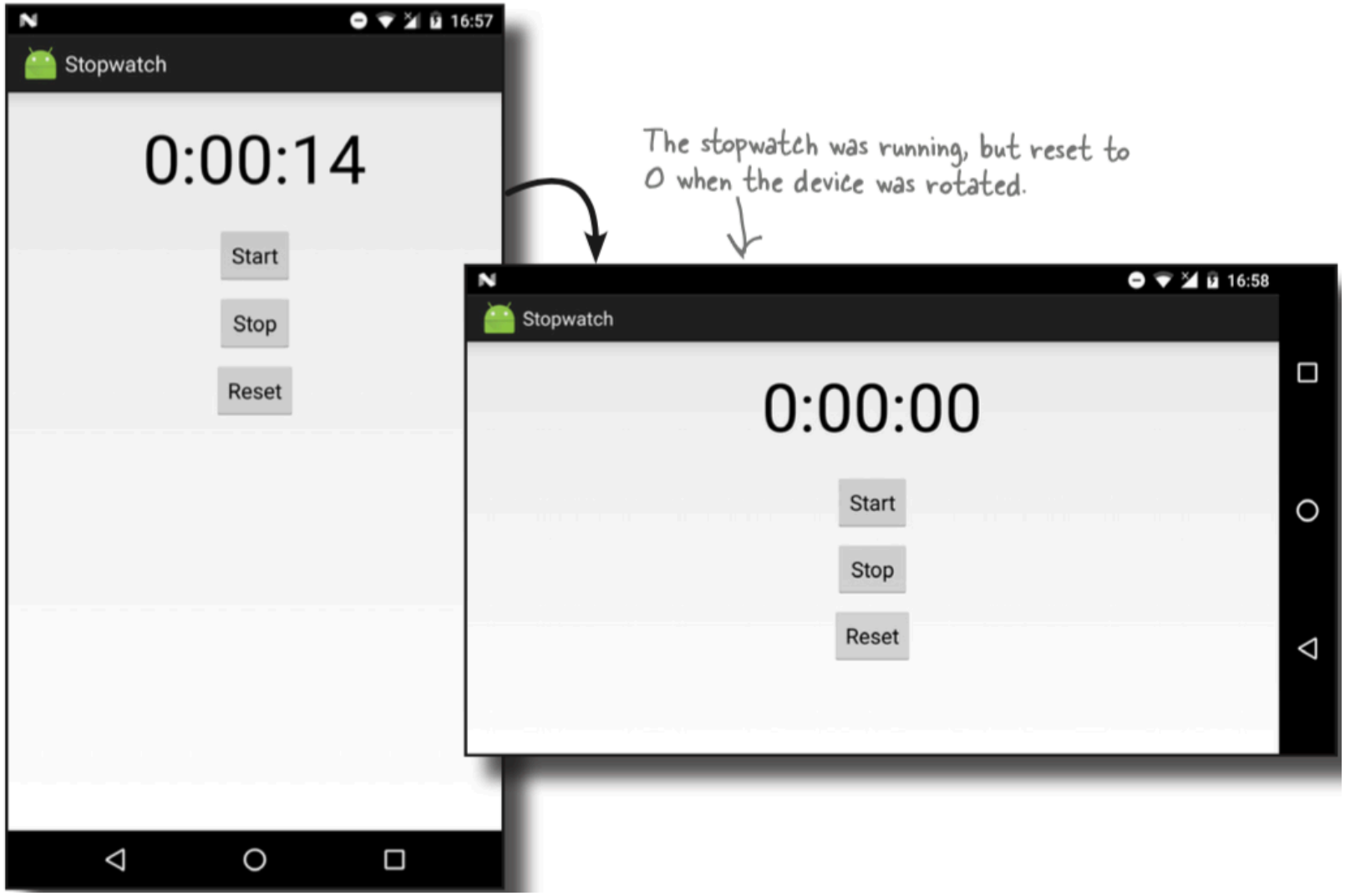


Prueba

These buttons work as you'd expect. The Start button starts the stopwatch, the Stop button stops it, and the Reset button sets the stopwatch back to 0.



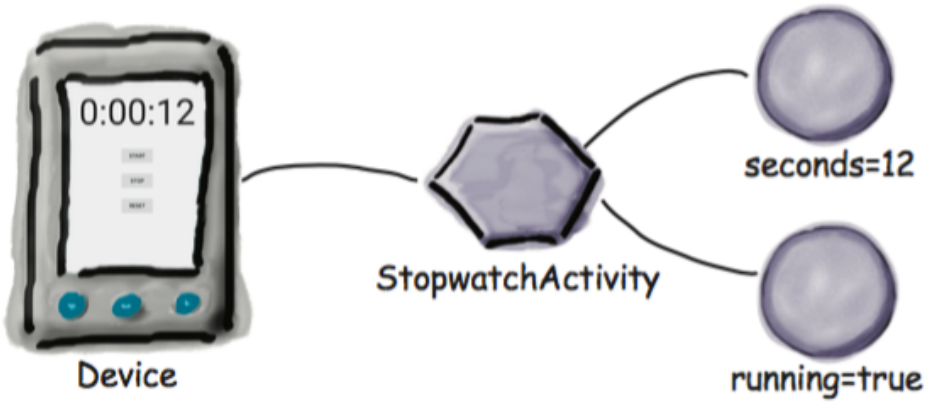
Prueba … apareció un problema



Que acaba de pasar ?

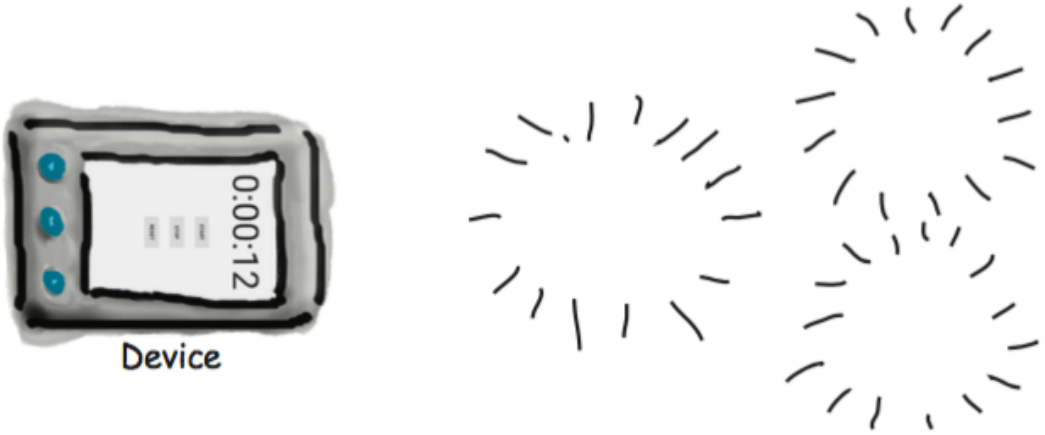
1 **The user starts the app, and clicks on the start button to set the stopwatch going.**

The `runTimer()` method starts incrementing the number of seconds displayed in the `time_view` text view using the `seconds` and `running` variables.



2 **The user rotates the device.**

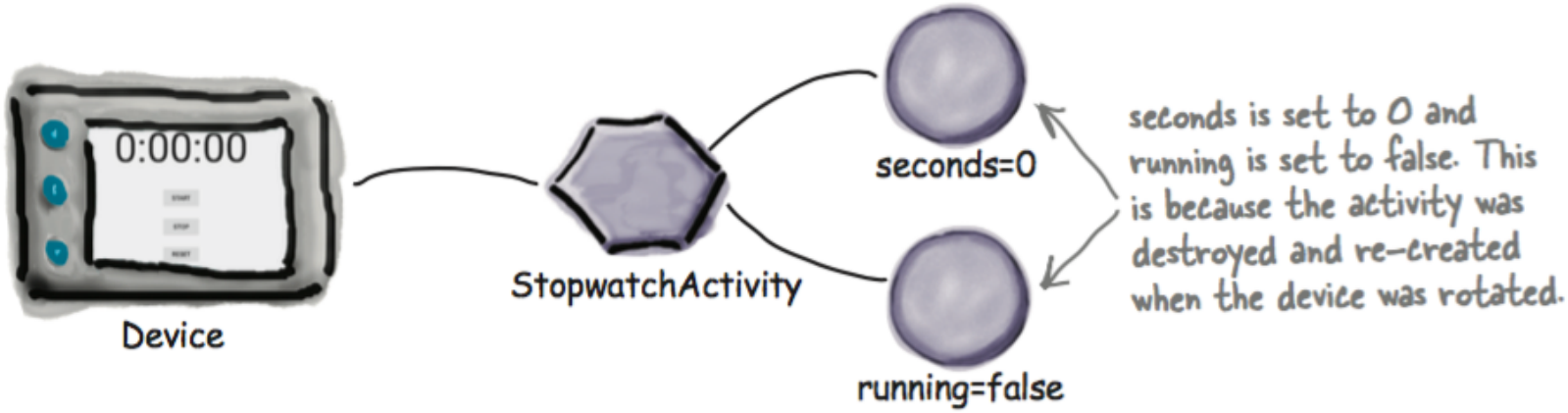
Android sees that the screen orientation and screen size has changed, and it destroys the activity, including any variables used by the `runTimer()` method.



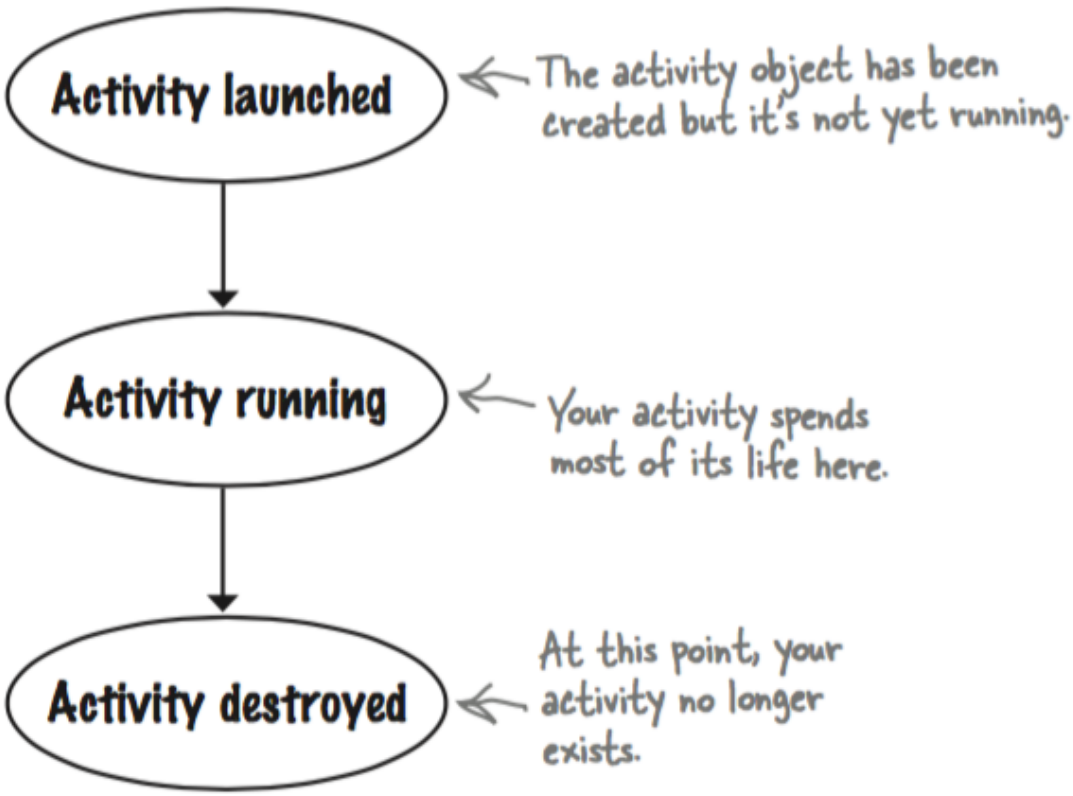
Que acaba de pasar ?

3 **StopwatchActivity is then re-created.**

The `onCreate()` method runs again, and the `runTimer()` method gets called. As the activity has been re-created, the `seconds` and `running` variables are set to their default values.



Los estados de una actividad (del nacimiento a la muerte)



El ciclo de vida de una actividad (de la creación a la destrucción)

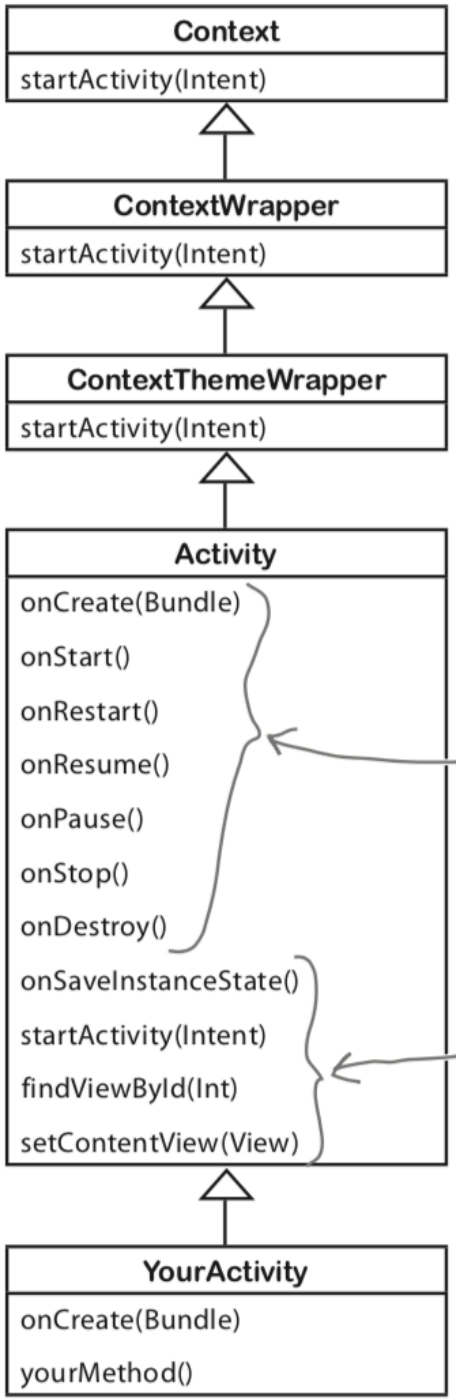
1. **The activity gets launched.**
   The activity object is created and its constructor is run.

2. **The onCreate() method runs immediately after the activity is launched.**
   The `onCreate()` method is where any initialization code should go, as this method always gets called after the activity has launched and before it starts running.

3. **An activity is running when it's visible in the foreground and the user can interact with it.**
   This is where an activity spends most of its life.

4. **The onDestroy() method runs immediately before the activity is destroyed.**
   The `onDestroy()` method enables you to perform any final clean up such as freeing up resources.

5. **After the onDestroy() method has run, the activity is destroyed.**
   The activity ceases to exist.



Context abstract class
(android.content.Context)
An interface to global information about the application environment. Allows access to application resources, classes, and operations.

ContextWrapper class
(android.content.ContextWrapper)
A proxy implementation for the `Context`.

ContextThemeWrapper class
(android.view.ContextThemeWrapper)
Allows you to modify the theme from what's in the `ContextWrapper`.

Activity class
(android.app.Activity)
The `Activity` class implements default versions of the lifecycle methods. It also defines methods such as `findViewById(Int)` and `setContentView(View)`.

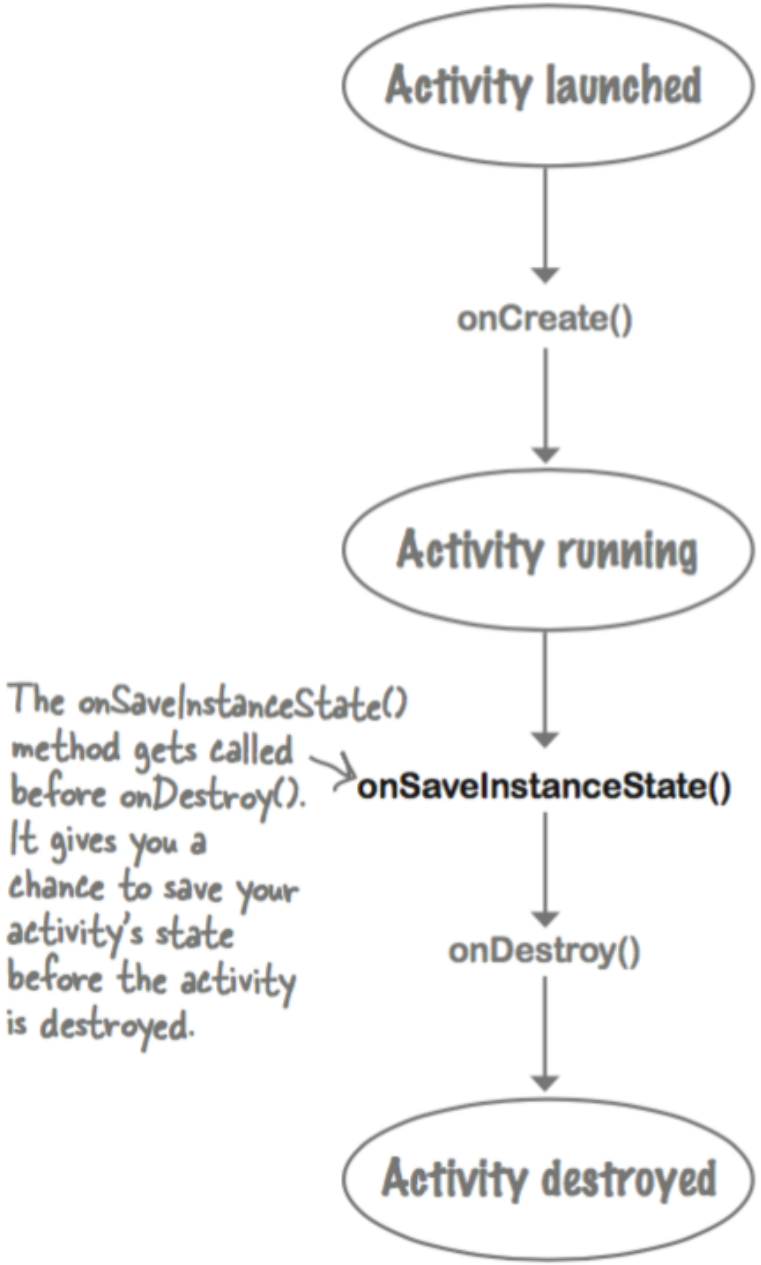
These are the activity lifecycle methods. You'll find out more about them through the rest of the chapter.

These aren't lifecycle methods, but they're still very useful. You've already used most of them in earlier chapters.

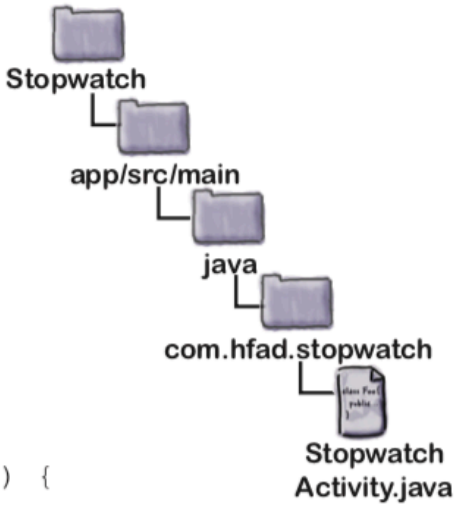
YourActivity class
(com.hfad.foo)
Most of the behavior of your activity is handled by superclass methods your activity inherits. All you do is override the methods you need.

La actividad hereda los métodos del ciclo de vida

Almacenando y restaurando el estado actual



Almacenando y restaurando el estado actual




```
...

public class StopwatchActivity extends Activity {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;

    @Override
    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();
    }

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

Restore the activity's state by getting values from the Bundle.

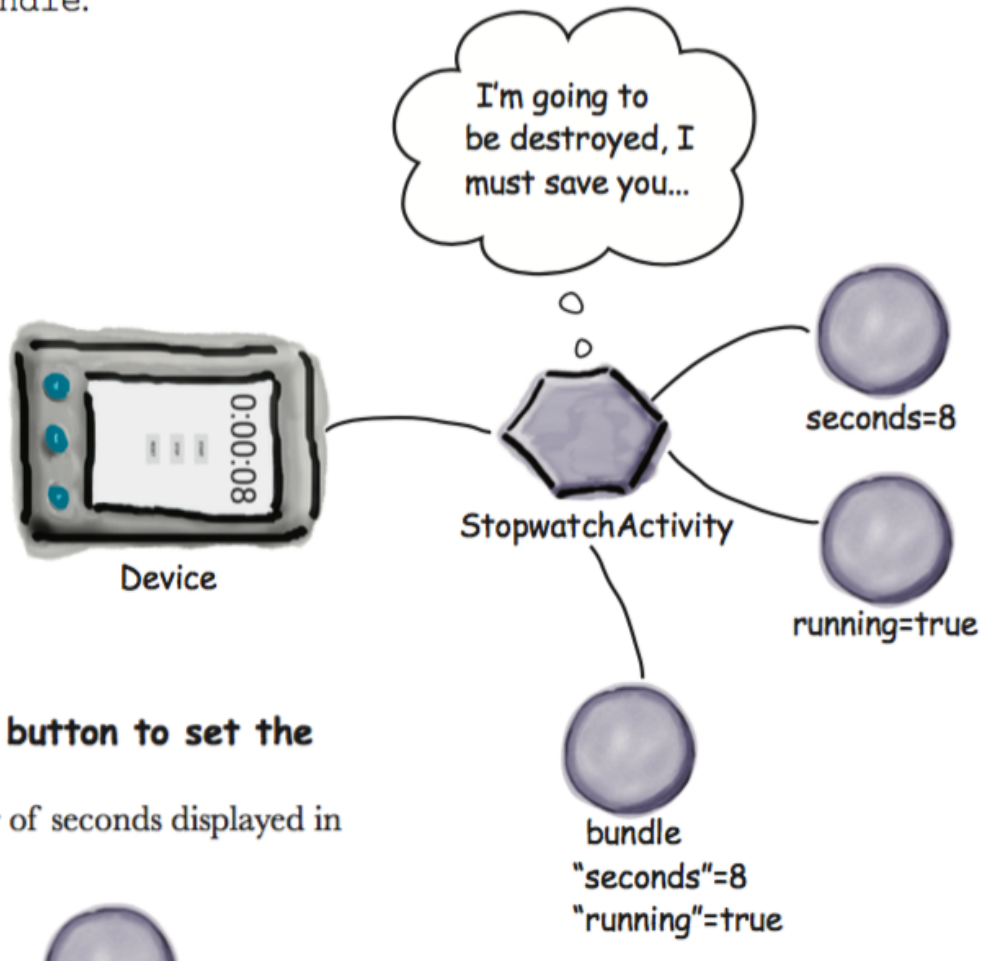
Save the state of the variables in the activity's onSaveInstanceState() method.

We've left out some of the activity code, as we don't need to change it.

Que pasa al ejecutar la aplicación

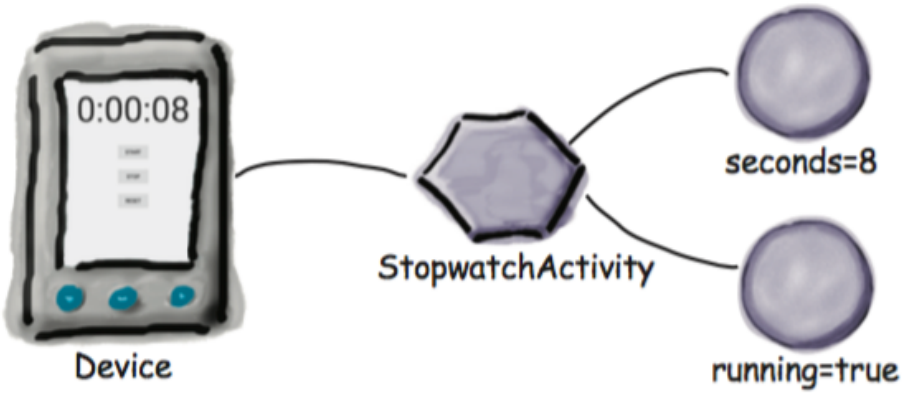
8 The user rotates the device.

Android views this as a configuration change, and gets ready to destroy the activity. Before the activity is destroyed, `onSaveInstanceState()` gets called. The `onSaveInstanceState()` method saves the seconds and running values to a `Bundle`.



1 The user starts the app, and clicks on the start button to set the stopwatch going.

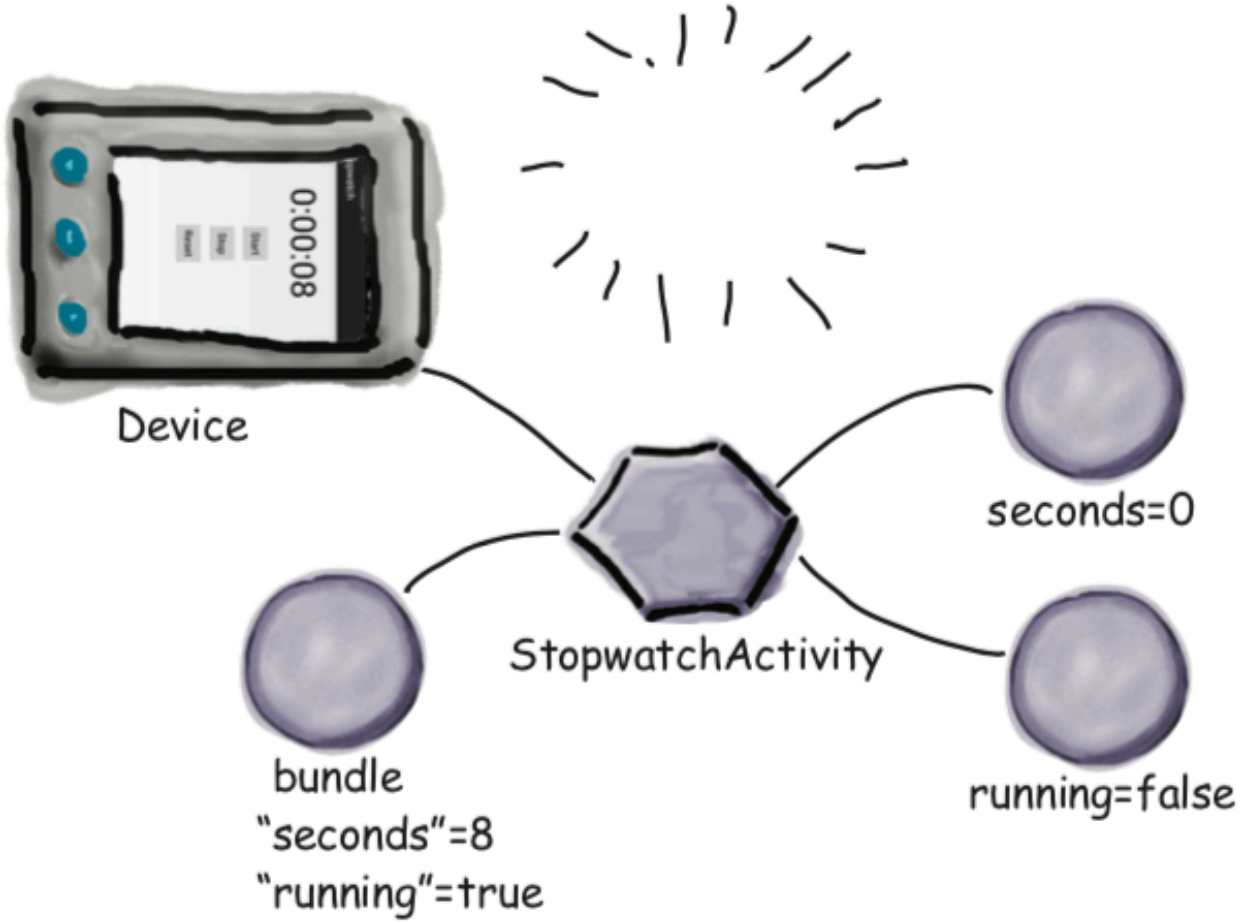
The `runTimer()` method starts incrementing the number of seconds displayed in the `time_view` text view.



Que pasa al ejecutar la aplicación

3 **Android destroys the activity, and then recreates it.**

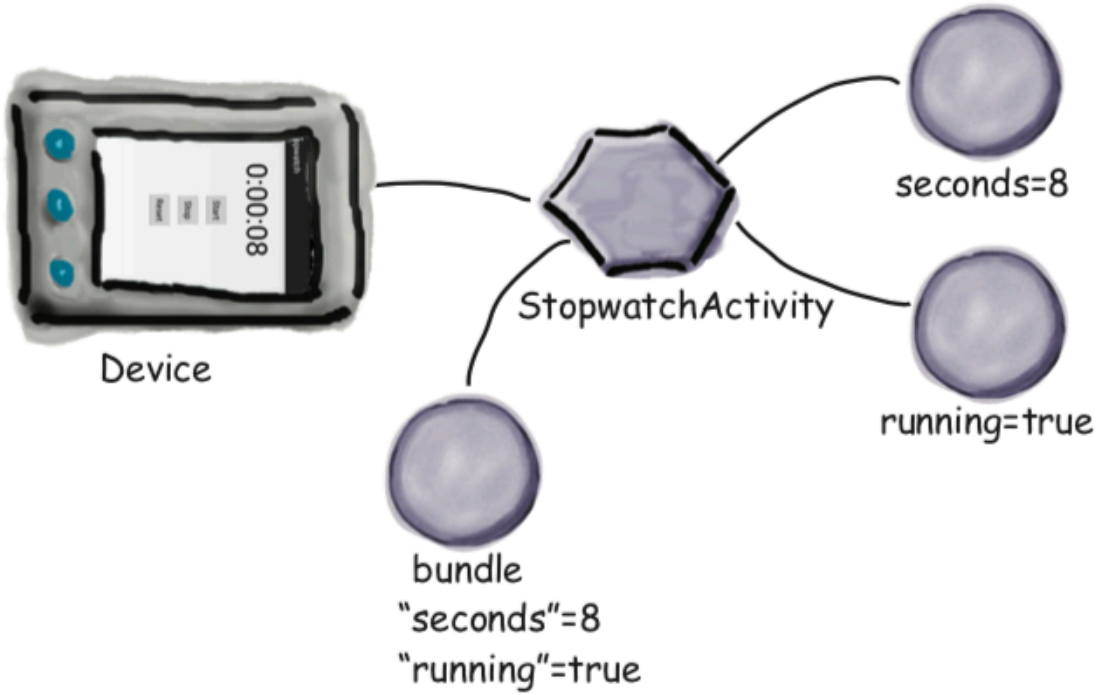
The `onCreate()` method gets called, and the `Bundle` gets passed to it.



Que pasa al ejecutar la aplicación

4 **The Bundle contains the values of the seconds and running variables as they were before the activity was destroyed.**

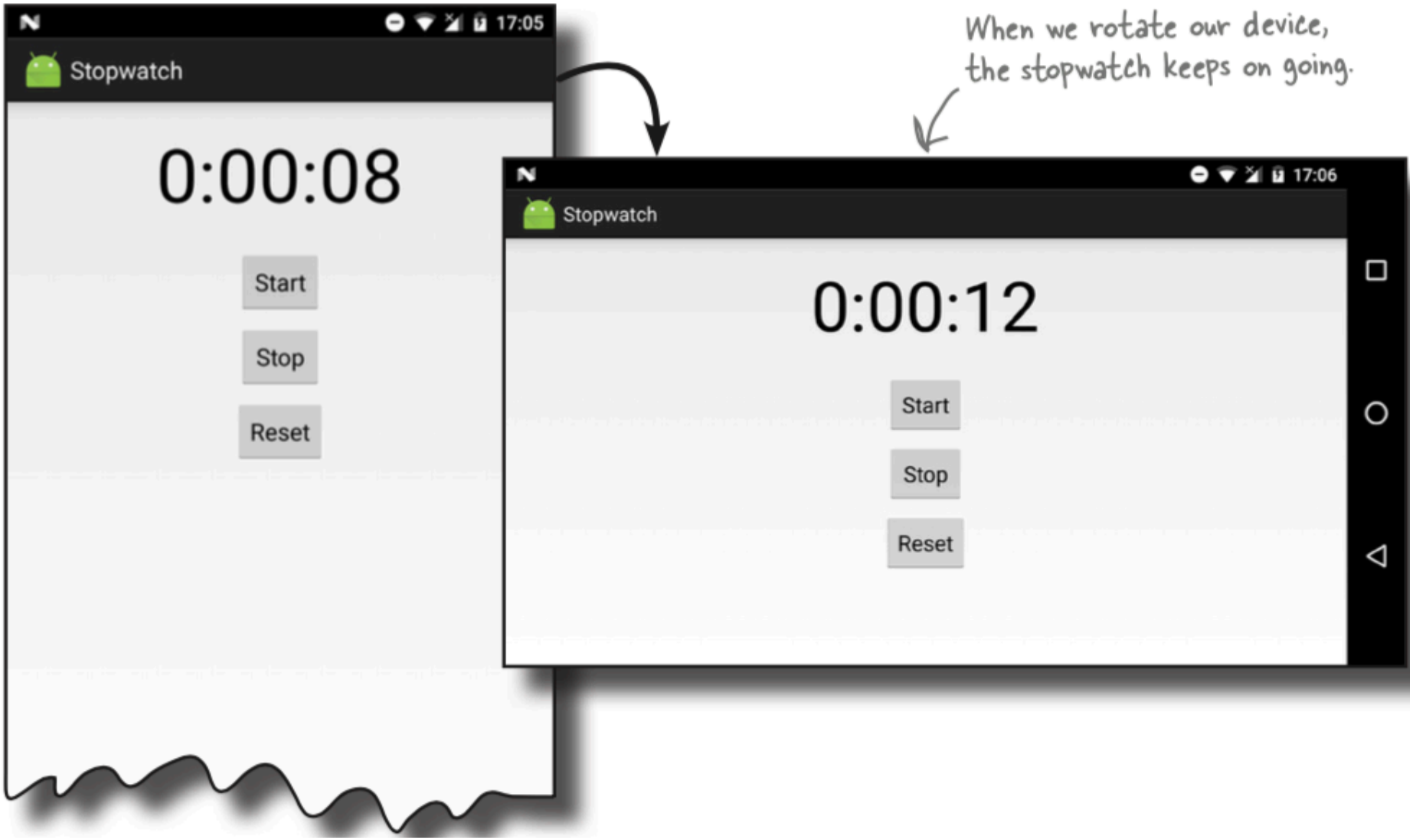
Code in the `onCreate()` method sets the current variables to the values in the `Bundle`.



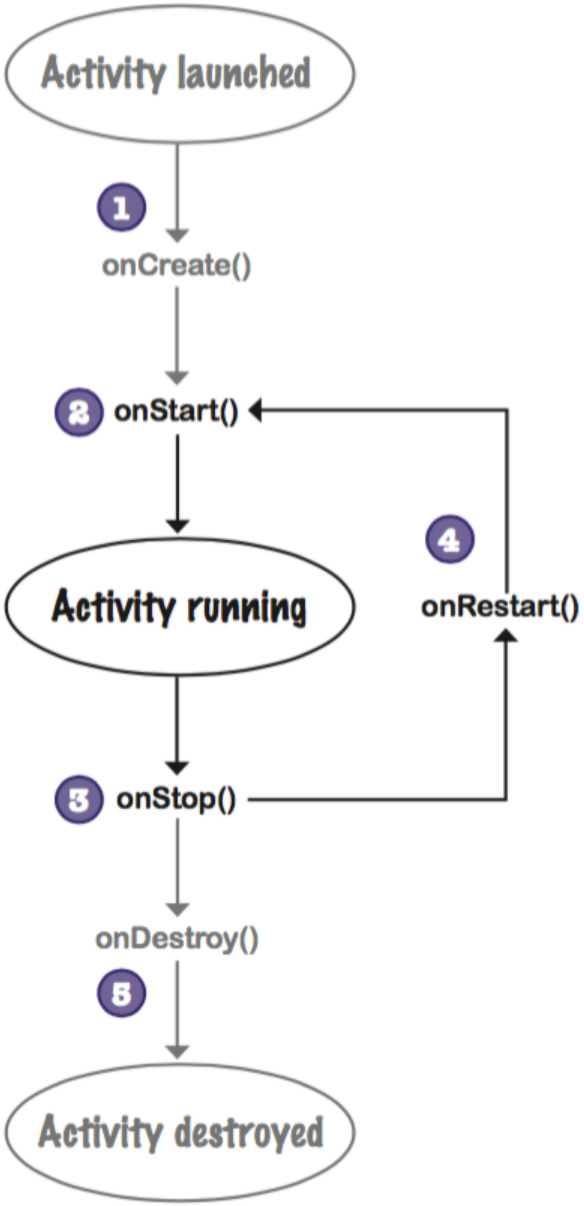
5 **The runTimer() method gets called, and the timer picks up where it left off.**

The running stopwatch gets displayed on the device and continues to increment.

Prueba


17:05
Stopwatch
0:00:08
Start
Stop
Reset
When we rotate our device, the stopwatch keeps on going.
17:06
Stopwatch
0:00:12
Start
Stop
Reset

Inicio, paro y restablecimiento de una actividad



1. **The activity gets launched, and the onCreate() method runs.**
Any activity initialization code in the `onCreate()` method runs. At this point, the activity isn't yet visible, as no call to `onStart()` has been made.

2. **The onStart() method runs after the onCreate() method. It gets called when the activity is about to become visible.**
After the `onStart()` method has run, the user can see the activity on the screen.

3. **The onStop() method runs when the activity stops being visible to the user.**
After the `onStop()` method has run, the activity is no longer visible.

4. **If the activity becomes visible to the user again, the onRestart() method gets called followed by onStart().**
The activity may go through this cycle many times if the activity repeatedly becomes invisible and visible again.

5. **Finally, the activity is destroyed.**
The `onStop()` method will usually get called before `onDestroy()`, but it may get bypassed if the device is extremely low on memory.

Reiniciando el cronómetro si fue detenido por onStop()

```
public class StopwatchActivity extends Activity {
    private int seconds = 0;
    private boolean running;
    private boolean wasRunning;

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

A new variable, wasRunning, records whether the stopwatch was running before the onStop() method was called so that we know whether to set it running again when the activity becomes visible again.

Restore the state of the wasRunning variable if the activity is re-created.

Reiniciando el cronómetro si fue detenido por onStop()

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

Save the state of the wasRunning variable.

```
@Override
protected void onStop() {
    super.onStop();
    wasRunning = running;
    running = false;
}
```

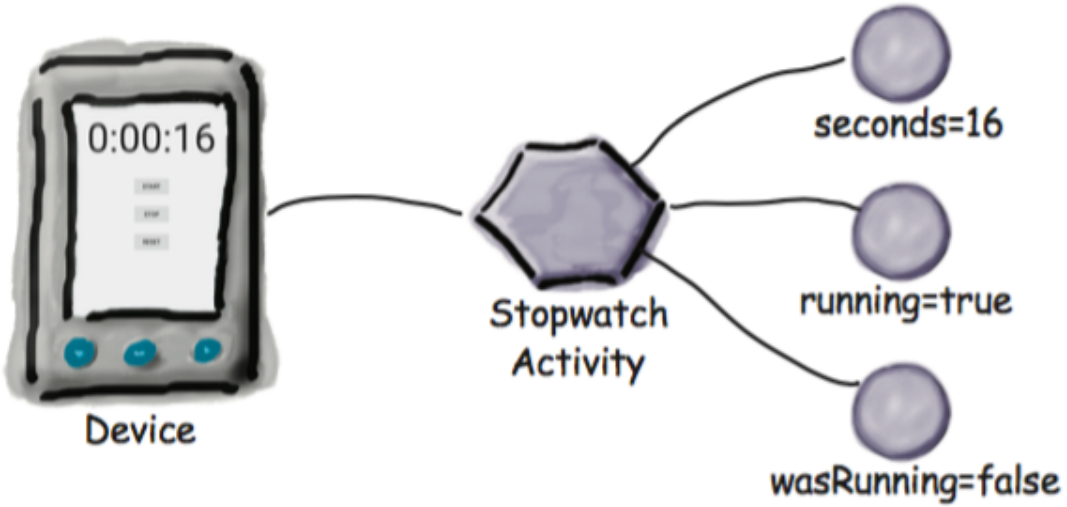
Record whether the stopwatch was running when the onStop() method was called.

```
@Override
protected void onStart() {
    super.onStart();
    if (wasRunning) {
        running = true;
    }
}
```

Implement the onStart() method. If the stopwatch was running, set it running again.

Que pasa al ejecutar la aplicación

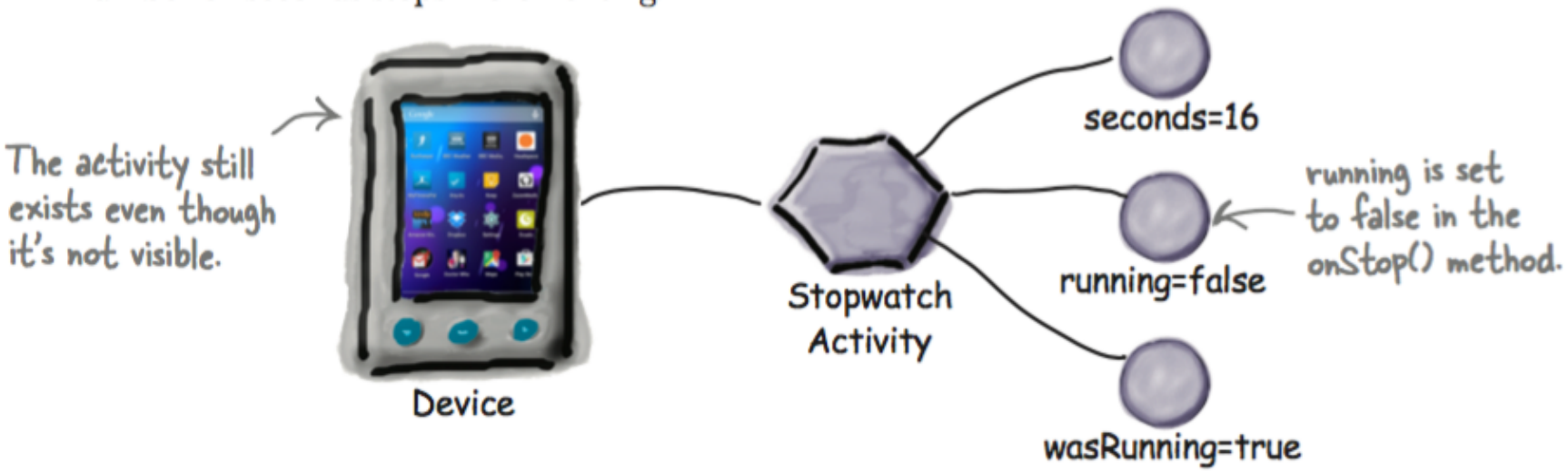
1 **The user starts the app, and clicks the Start button to set the stopwatch going.** The `runTimer()` method starts incrementing the number of seconds displayed in the `time_view` text view.



Que pasa al ejecutar la aplicación

3 **The user navigates to the device home screen so the Stopwatch app is no longer visible.**

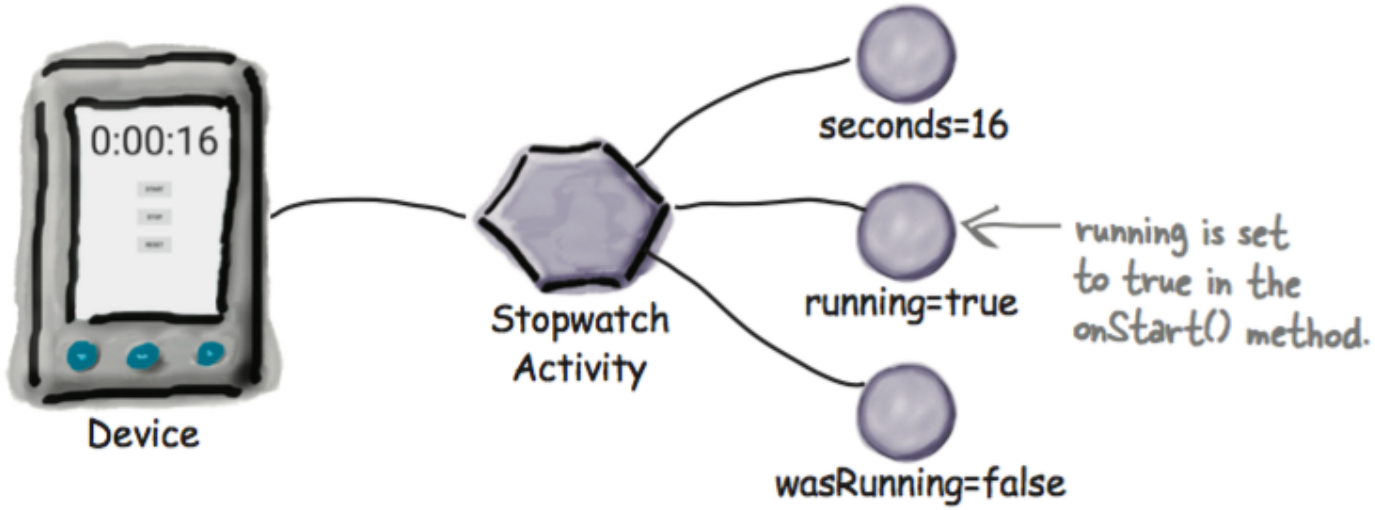
The `onStop()` method gets called, `wasRunning` is set to `true`, `running` is set to `false`, and the number of seconds stops incrementing.



Que pasa al ejecutar la aplicación

3 **The user navigates back to the Stopwatch app.**

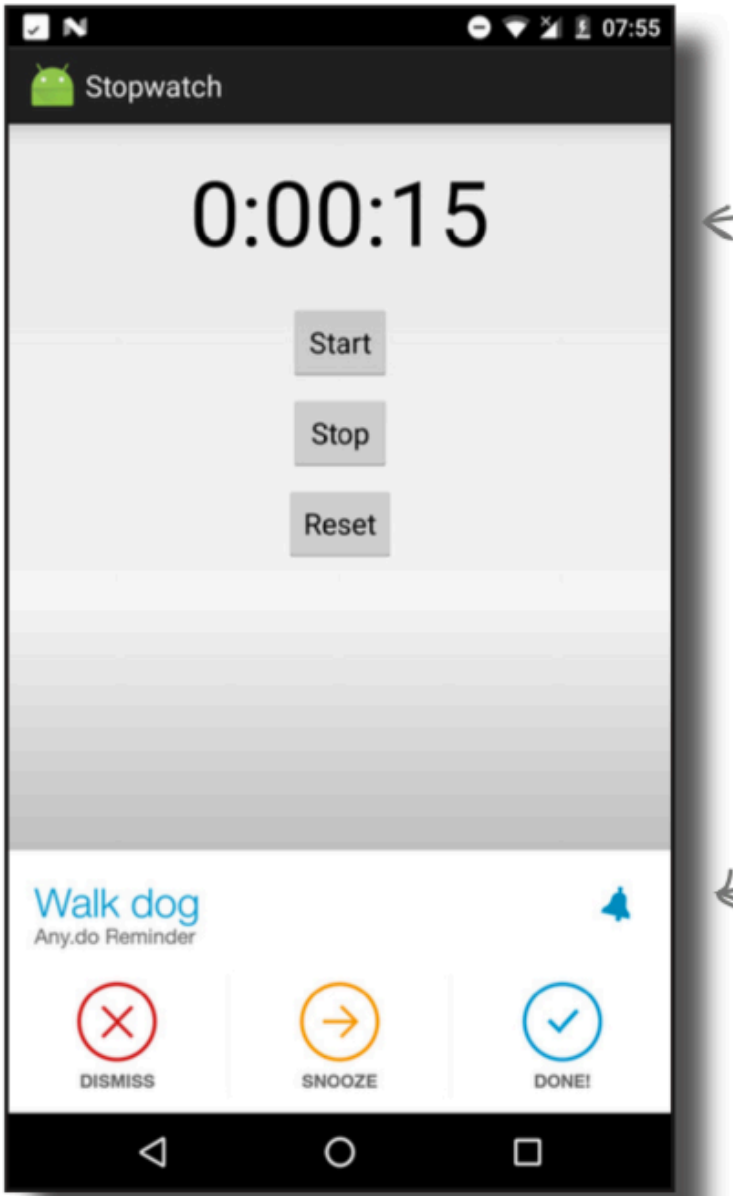
The `onStart()` method gets called, `running` is set to `true`, and the number of seconds starts incrementing again.



Prueba


17:09
Stopwatch
0:00:16
Start
Stop
Reset
We set our stopwatch going, then switched to the device's home screen.
Google
Google
Create
Runkeeper
17:09
Stopwatch
0:00:16
Start
Stop
Reset
The stopwatch paused while the app wasn't visible.
17:09
Stopwatch
0:00:26
Start
Stop
Reset
The stopwatch started again when we went back to it.

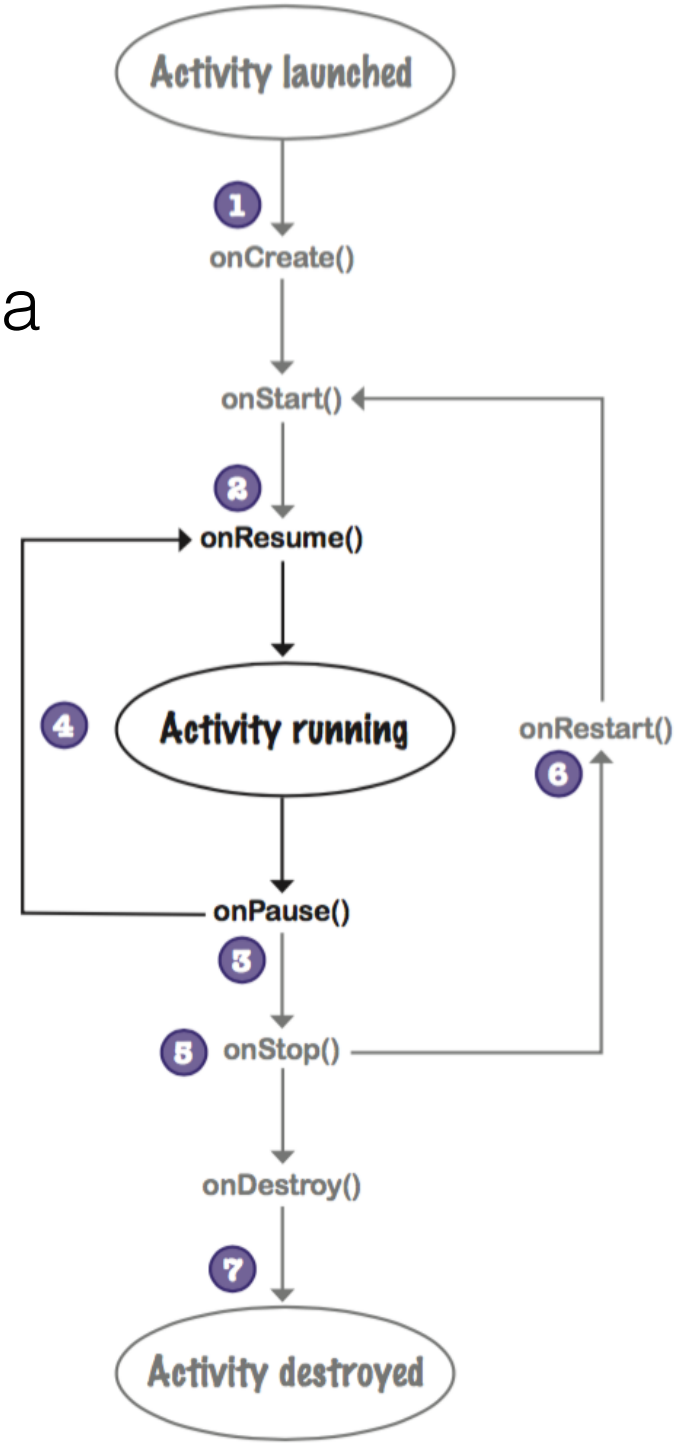
Que pasa si la aplicación es visible parcialmente



The stopwatch activity is still visible, but it's partially obscured and no longer has the focus. When this happens, it pauses.

This is an activity from another app that's appeared on top of the stopwatch.

EL ciclo de vida en el fondo



1. **The activity gets launched, and the onCreate() and onStart() methods run.**
   At this point, the activity is visible, but it doesn't have the focus.

2. **The onResume() method runs after the onStart() method. It gets called when the activity is about to move into the foreground.**
   After the `onResume()` method has run, the activity has the focus and the user can interact with it.

3. **The onPause() method runs when the activity stops being in the foreground.**
   After the `onPause()` method has run, the activity is still visible but doesn't have the focus.

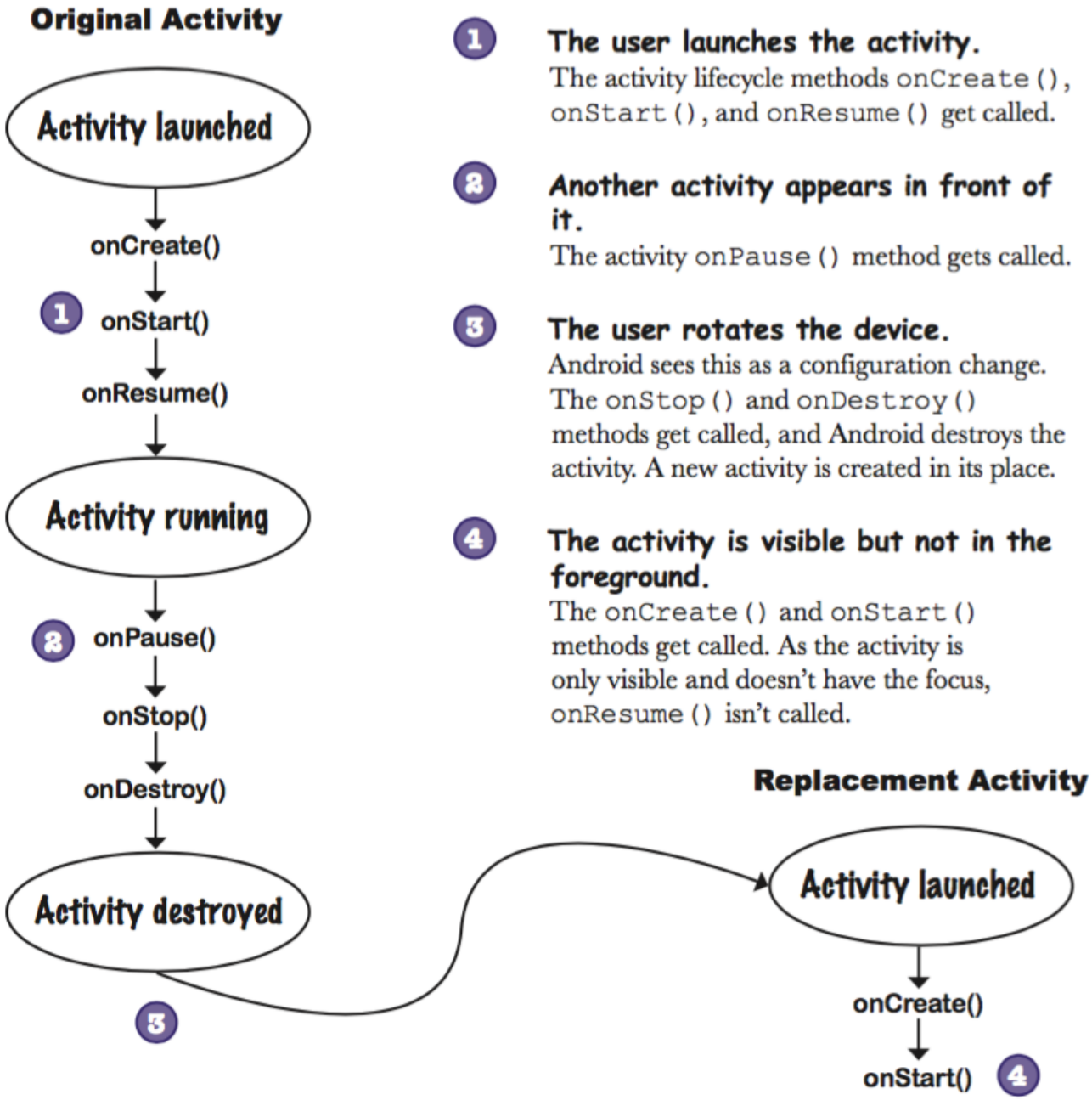
4. **If the activity moves into the foreground again, the onResume() method gets called.**
   The activity may go through this cycle many times if the activity repeatedly loses and regains the focus.

5. **If the activity stops being visible to the user, the onStop() method gets called.**
   After the `onStop()` method has run, the activity is no longer visible.

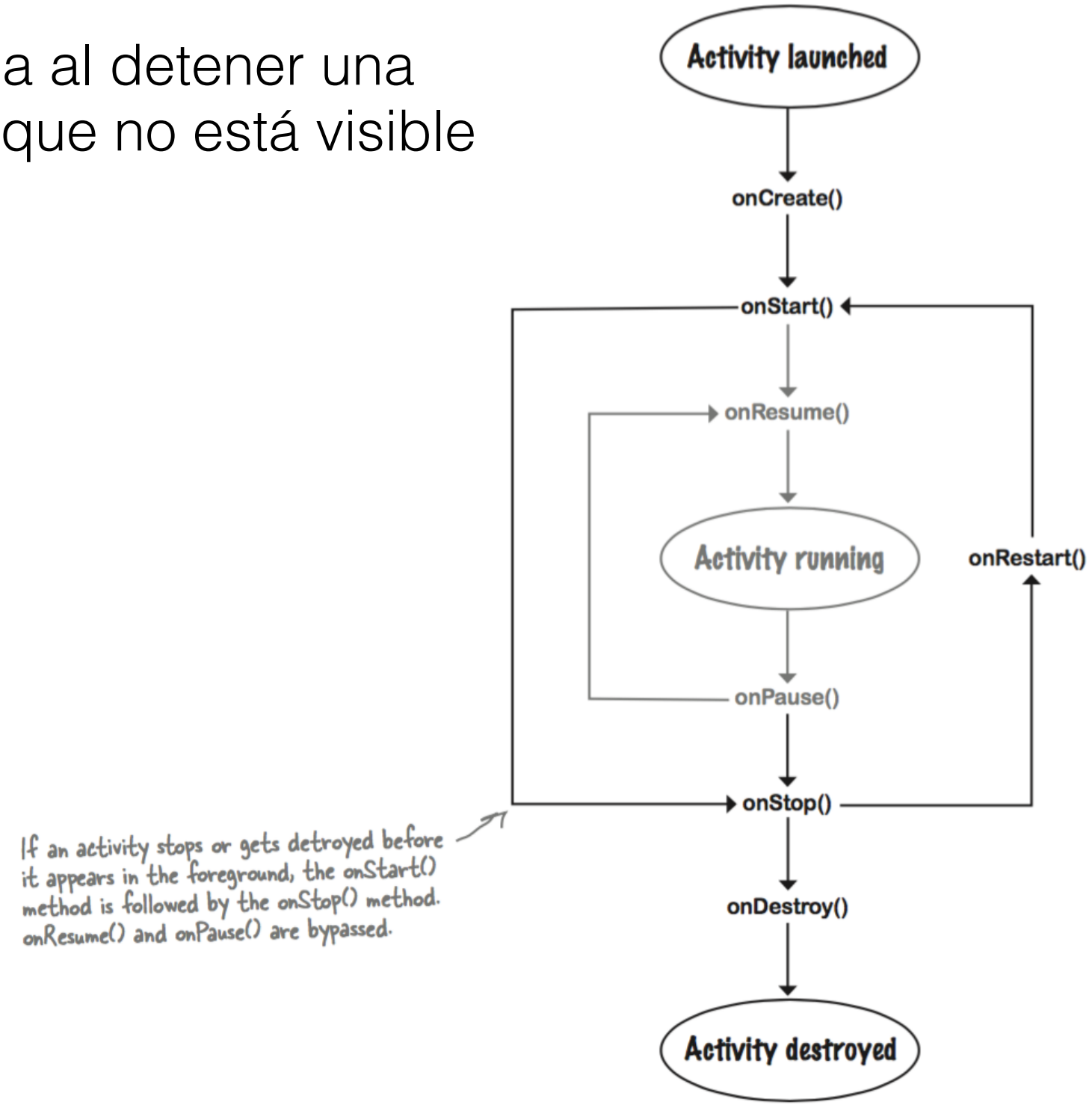
6. **If the activity becomes visible to the user again, the onRestart() method gets called, followed by onStart() and onResume().**
   The activity may go through this cycle many times.

7. **Finally, the activity is destroyed.**
   As the activity moves from running to destroyed, the `onPause()` method gets called before the activity is destroyed. The `onStop()` method usually gets called too.

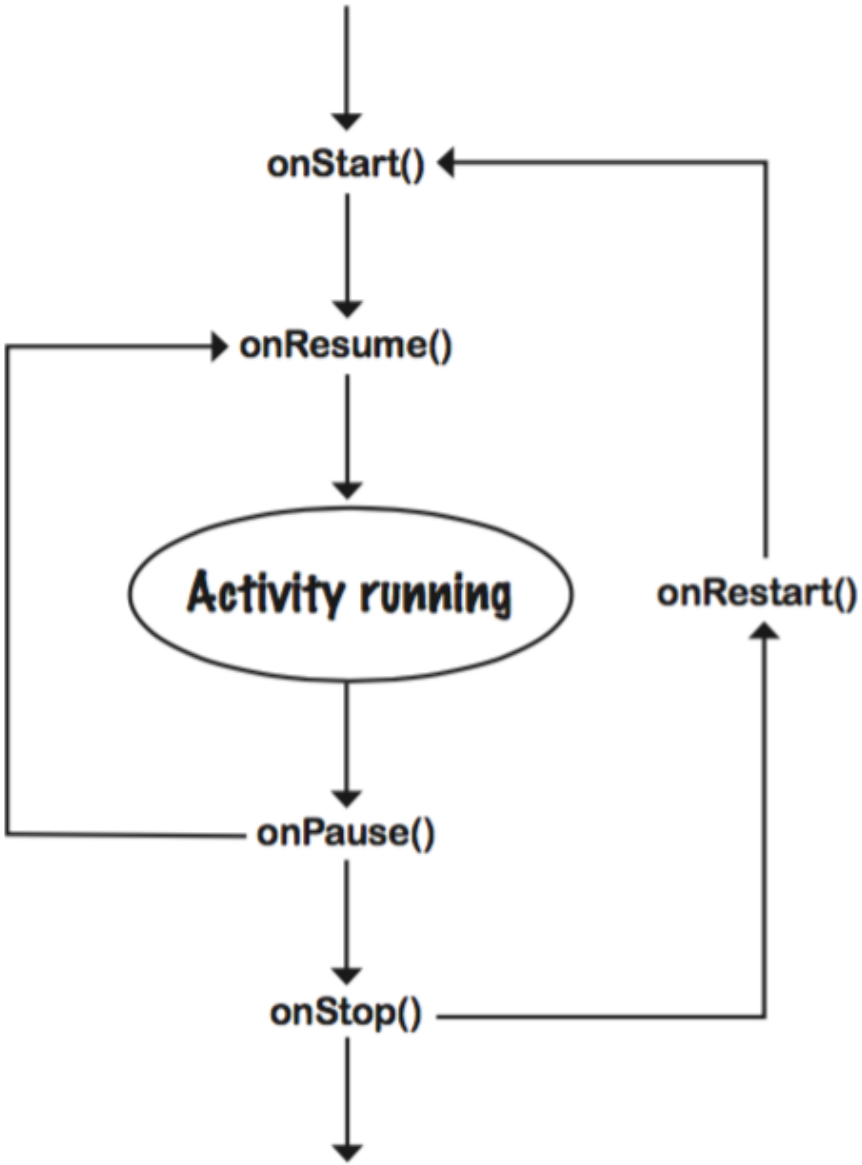
Que pasa al rotar el dispositivo estando la aplicación en pausa



Que pasa al detener una aplicación que no está visible



Parando el cronómetro si la actividad está en pausa



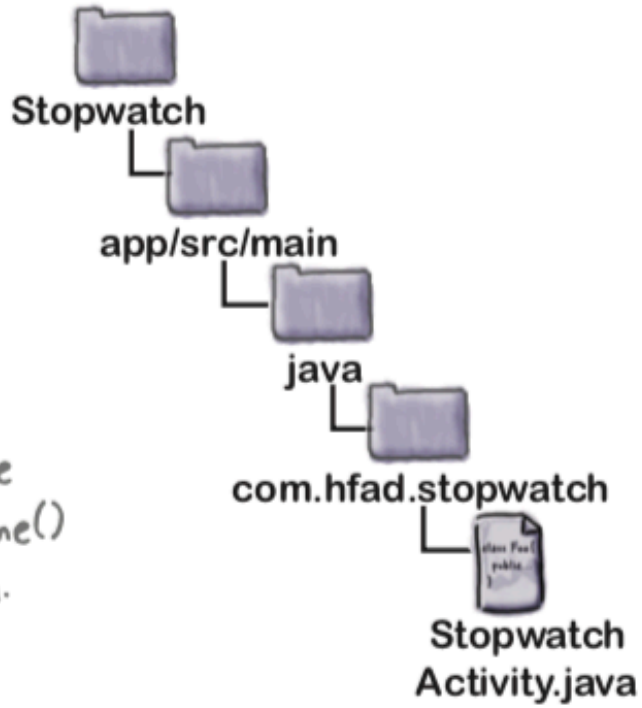
Parando el cronómetro si la actividad está en pausa

~~@Override~~
~~protected void onStart() {~~
~~super.onStart();~~
~~if (wasRunning) {~~
~~running = true;~~
~~}~~
~~}~~

Delete the onStart() method.

```
@Override
protected void onResume() {
    super.onResume();
    if (wasRunning) {
        running = true;
    }
}
```

Add the onResume() method.



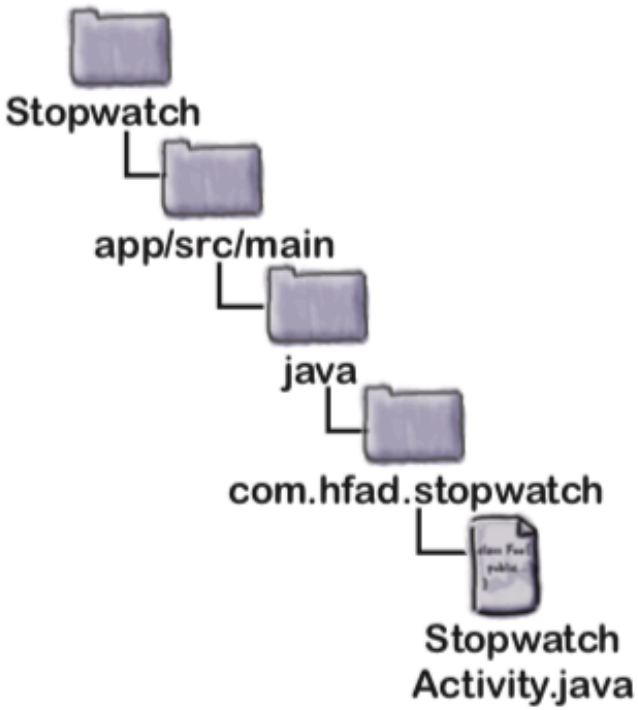
Parando el cronómetro si la actividad está en pausa

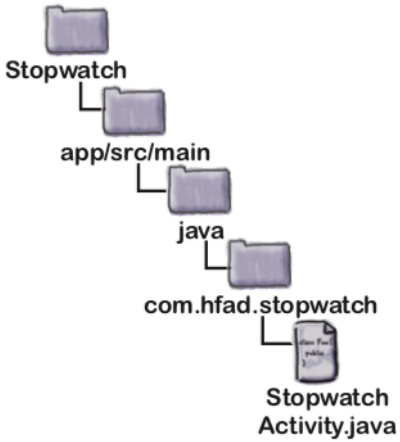
```
@Override
protected void onStop() {
    super.onStop();
    wasRunning = running;
    running = false;
}
```

Delete the onStop() method.

```
@Override
protected void onPause() {
    super.onPause();
    wasRunning = running;
    running = false;
}
```

Add the onPause() method.






```
package com.hfad.stopwatch;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import java.util.Locale;
import android.os.Handler;
import android.widget.TextView;

public class StopwatchActivity extends Activity {
    //Number of seconds displayed on the stopwatch.
    private int seconds = 0;
    //Is the stopwatch running?
    private boolean running;
    private boolean wasRunning;

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

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

Use seconds, running, and wasRunning respectively to record the number of seconds passed, whether the stopwatch is running, and whether the stopwatch was running before the activity was paused.

Get the previous state of the stopwatch if the activity's been destroyed and recreated.

Save the state of the stopwatch if it's about to be destroyed.

El código completo de la actividad

El código completo de la actividad

```
@Override
protected void onPause() {
    super.onPause();
    wasRunning = running;
    running = false;
}

@Override
protected void onResume() {
    super.onResume();
    if (wasRunning) {
        running = true;
    }
}

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

If the activity's paused, stop the stopwatch.

If the activity's resumed, start the stopwatch again if it was running previously.

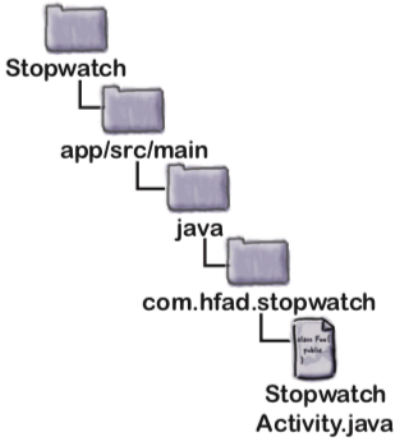
This gets called when the Start button is clicked.

```
    //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;
    }

    //Sets the number of seconds on the timer.
    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(Locale.getDefault(),
                            "%d:%02d:%02d", hours, minutes, secs);
                timeView.setText(time);
                if (running) {
                    seconds++;
                }
                handler.postDelayed(this, 1000);
            }
        });
    }
}
```

This gets called when the Stop button is clicked.

This gets called when the Reset button is clicked.

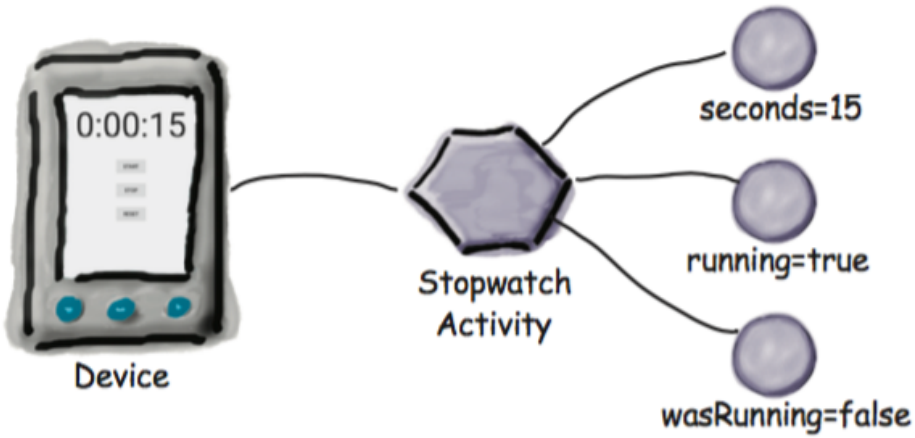
The runTimer() method uses a Handler to increment the seconds and update the text view.



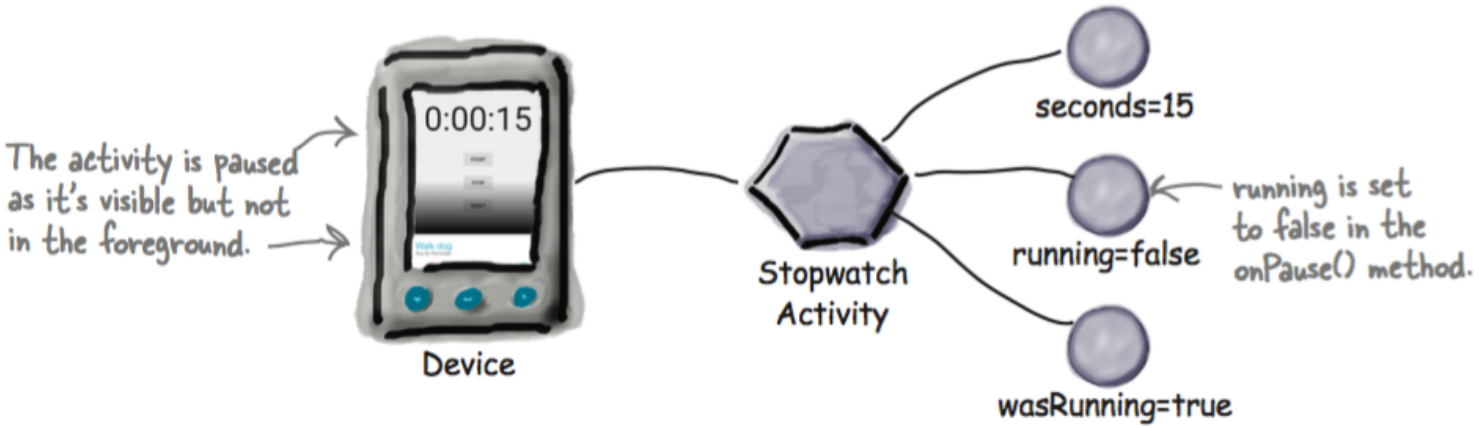
El código completo de la actividad

Que pasa al ejecutar la aplicación

1 **The user starts the app, and clicks on the start button to set the stopwatch going.**
The `runTimer()` method starts incrementing the number of seconds displayed in the `time_view` text view.

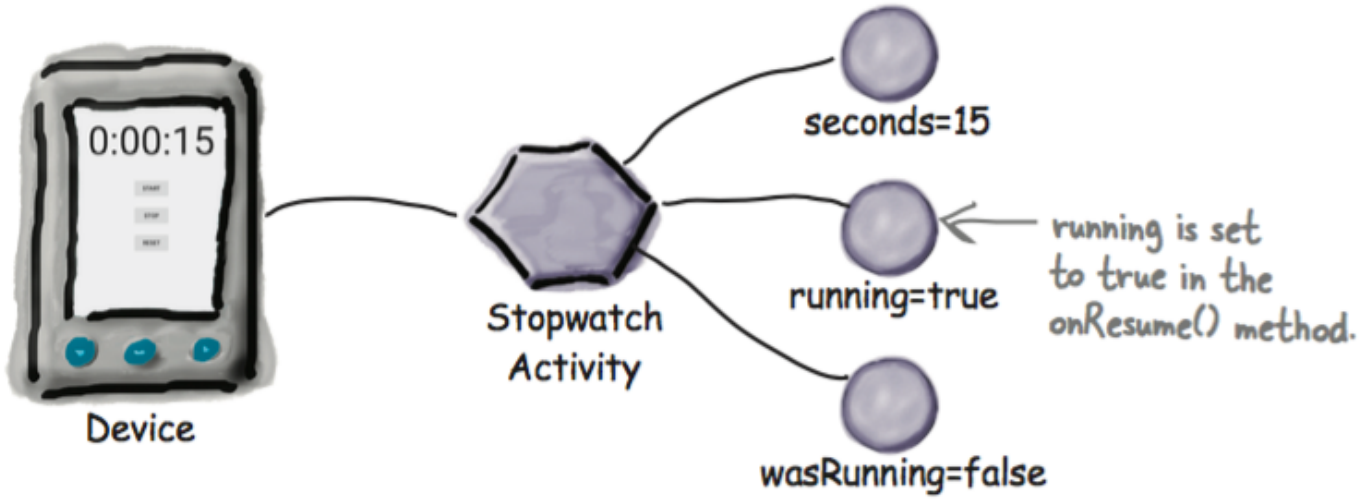


2 **Another activity appears in the foreground, leaving StopwatchActivity partially visible.**
The `onPause()` method gets called, `wasRunning` is set to `true`, `running` is set to `false`, and the number of seconds stops incrementing.



Que pasa al ejecutar la aplicación

3 **When StopwatchActivity returns to the foreground, the onResume() method gets called, running is set to true, and the number of seconds starts incrementing again.**



Prueba


07:57
Stopwatch
0:00:34
Start
Stop
Reset
We started our stopwatch.
08:00
Stopwatch
0:02:28
Start
Stop
Reset
The stopwatch paused when the activity was partially obscured.
Walk dog
Any.do Reminder
DISMISS
SNOOZE
DONE!
08:00
Stopwatch
0:02:43
Start
Stop
Reset
The stopwatch started again when the activity came back into the foreground.

Los métodos del ciclo de vida

Method	When it's called	Next method
onCreate()	When the activity is first created. Use it for normal static setup, such as creating views. It also gives you a `Bundle` giving the previously saved state of the activity.	`onStart()`
onRestart()	When your activity has been stopped just before it gets started again.	`onStart()`
onStart()	When your activity is becoming visible. It's followed by `onResume()` if the activity comes into the foreground, or `onStop()` if the activity is made invisible.	`onResume()` or `onStop()`
onResume()	When your activity is in the foreground.	`onPause()`
onPause()	When your activity is no longer in the foreground because another activity is resuming. The next activity isn't resumed until this method finishes, so any code in this method needs to be quick. It's followed by `onResume()` if the activity returns to the foreground, or `onStop()` if it becomes invisible.	`onResume()` or `onStop()`
onStop()	When the activity is no longer visible. This can be because another activity is covering it, or because the activity's being destroyed. It's followed by `onRestart()` if the activity becomes visible again, or `onDestroy()` if the activity is going to be destroyed.	`onRestart()` or `onDestroy()`
onDestroy()	When your activity is about to be destroyed or because the activity is finishing.	None

BULLET POINTS

- Each app runs in its own process by default.
- Only the main thread can update the user interface.
- Use a `Handler` to schedule code, or post code to a different thread.
- 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.
- `onSaveInstanceState(Bundle)` enables your activity to save its state before the activity gets destroyed. You can use the `Bundle` to restore state in `onCreate()`.
- You add values to a `Bundle` using `bundle.put*("name", value)`. You retrieve values from the bundle using `bundle.get*("name")`.
- `onCreate()` and `onDestroy()`, deal with the birth and death of the activity.
- `onRestart()`, `onStart()` and `onStop()` deal with the visibility of the activity.
- `onResume()` and `onPause()` deal with when the activity gains and loses the focus.