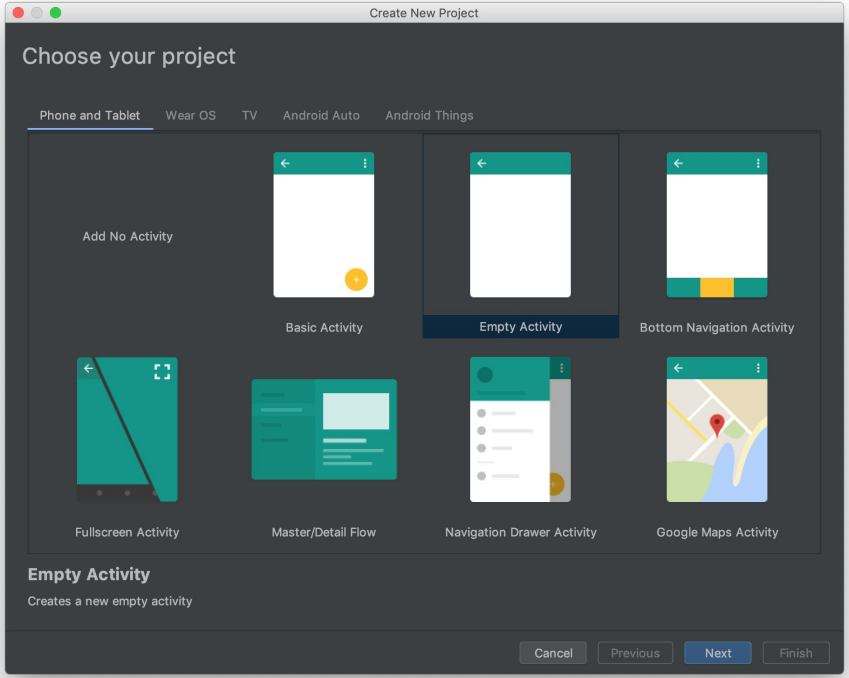
Creating a Stopwatch

Create a New Project

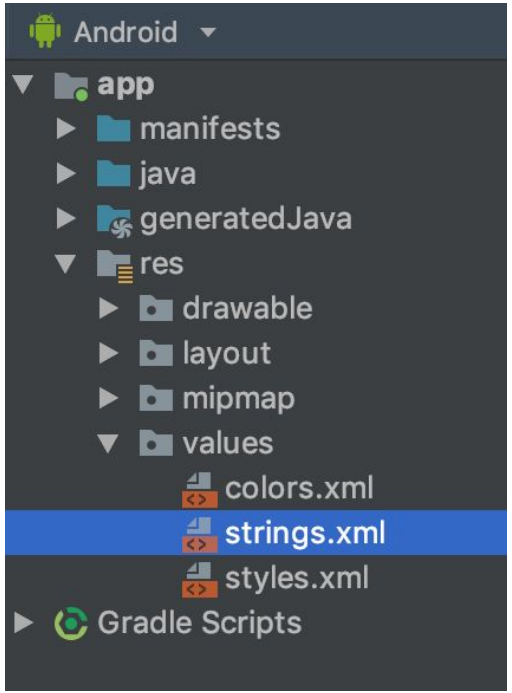


Maydm

Create a New Project

1. Create a new project in Android Studio
2. Give the project the name Stopwatch
3. Set the company domain as **yourname**.com
4. Target Android Devices using API 19 (Android 4.4 KitKat)

Add String Resources



Find the strings.xml file. It's located under

app > res (resources) > values

Open the strings.xml file and add three strings to the resources element:

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

Maydm

Add String Resources

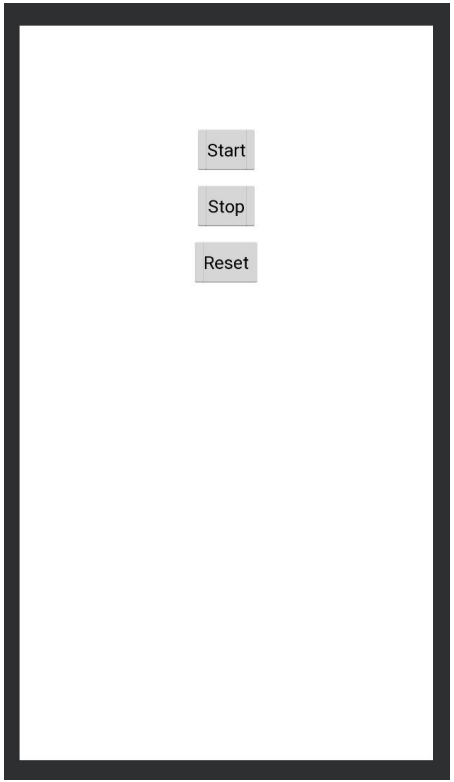
```
<resources>
   <string name="app_name">Stopwatch</string>
   <string name="start">Start</string>
   <string name="stop">Stop</string>
   <string name="reset">Reset</string>
</resources>
```

Maydm

Layout

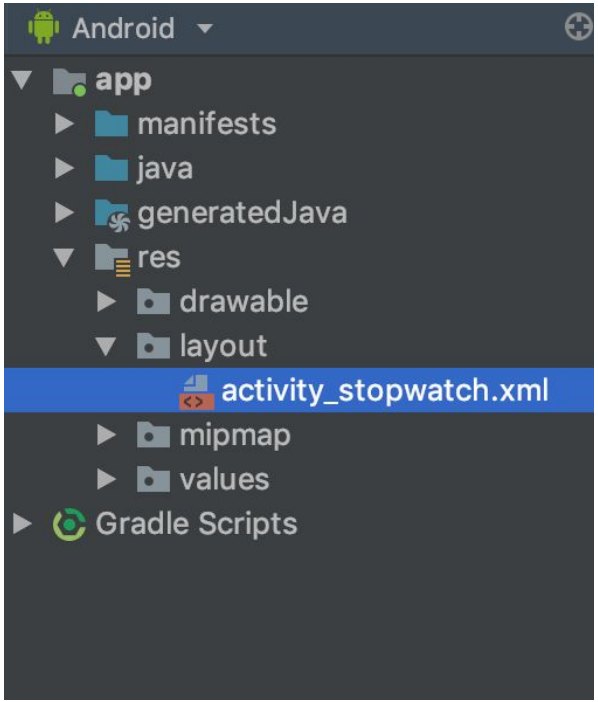
The layout for the stopwatch will be pretty simple. There will be four elements:

1. Text field that displays the elapsed time
2. Start button
3. Stop button
4. Reset button



Maydm

Update Layout Code



Find the activity_stopwatch.xml file. It's located under

app > res > layout

Open the activity_stopwatch.xml file.

Maydm

Update Layout Code

The file defaults to the Design view.

Click on Text at the bottom of the view to switch to the XML text file.

Leave the top line (the XML prolog) and delete the rest, so all you have is:

<?xml version="1.0" encoding="utf-8"?>

Maydm

Update Layout Code

Create a LinearLayout element using the following code:

```
<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"
    >
</LinearLayout>
```

Maydm

Update Layout Code

Inside the LinearLayout, add the TextView:

```
<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" />
```

Maydm

Update Layout Code

After the TextView element, add the Start button:

```
<Button
    android:id="@+id/start_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:onClick="onClickStart"
    android:text="Start" />
```

Maydm

Update Layout Code

After the Start Button, add the Stop button:

```
<Button
    android:id="@+id/stop_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:onClick="onClickStop"
    android:text="Stop" />
```

Maydm

Update Layout Code

After the Stop Button, add the Reset button:

```
<Button
    android:id="@+id/reset_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center_horizontal"
    android:onClick="onClickReset"
    android:text="Reset" />
```

Maydm

Adding the Timer Methods

Each of the buttons has an **onClick** attribute. This attribute specifies which activity method will run whenever a user clicks it.

Because the methods are named well, we know what each will do just by the name.

Now we need to add those methods! Find the StopwatchActivity.java file. It's located under

app > java > com.**yourname**.stopwatch

Maydm

Adding the Timer Methods

Replace the existing code with the following:

```
package com.yourname.stopwatch;

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


public class StopwatchActivity extends Activity {}
```

Maydm

Adding the Timer Methods

Inside the StopwatchActivity class, we'll set up some private variables. Because the variables are well named, we can figure out what they represent.

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

Maydm

Adding the Timer Methods

Now we add an Override to the onCreate() method to initialize the Activity and set the View.

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

Maydm

Adding the Timer Methods

Our first timer method is onClickStart. It takes the view as an argument and sets the running variable to true. (You don't need to add the comments.)

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

Maydm

Adding the Timer Methods

Our next method is onClickStop. It is exactly the same as onClickStart but it set sets running to false.

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

Maydm

Adding the Timer Methods

The last method is onClickReset. It sets running to false and resets the seconds.

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

Maydm

Making the Timer Run

Next we'll add a runTimer() method. This method has to do a lot:

- Increment the seconds variable to track the time
- Check the running variable to see if it's set to true
- Access the text view from the layout
- Format the seconds variable into something readable
- Display the results in the text view

Maydm

Making the Timer Run

```
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.time_view);

    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++;
    }
}
```

Maydm

Making the Timer Run

The runTimer() method needs to loop continuously while running in order to increment the seconds variable and update the text view every second.

This has to be done in a way that doesn't block the main Android thread. Only the main Android thread can update the user interface.

To get around this, we will use a Handler.

Handlers

A **Handler** is a class provided by Android that allows you to schedule code to run at some point in the future.

It can also post code that needs to run on a different thread from the main Android thread.

We'll use the Handler to schedule the stopwatch code to run every second.

Handlers

To use a Handler, wrap the code to be scheduled in a **Runnable** object, with the code to be run inside the Runnable's **run()** method.

Then use the Handler's **post()** and **postDelayed()** methods to specify when the code should run.

Handler.post() Method

The post() method posts code that needs to run ASAP, which usually runs almost immediately. It takes one parameter, a Runnable object.

The post() method looks something like this:

```
final Handler handler = new Handler();

handler.post(Runnable);
```

Maydm

Handler.postDelayed() Method

The postDelayed() method works just like post() except it schedules the code to run in the future. It takes two parameters, a Runnable object and a **long**, a type of integer. The long specifies the number of milliseconds to delay the code.

The postDelayed() method looks like this:

```
final Handler handler = new Handler();

handler.postDelayed(Runnable, long);
```

Maydm

Making the Timer Run

Head back to the StopwatchActivity.java file. We need to import a few more classes that we'll be using in the runTimer() method:

```
import java.util.Locale;
import android.os.Handler;
import android.widget.TextView;
```

Making the Timer Run

In the onCreate block, add a call to the runTimer() method.

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

Maydm

The runTimer() Method

Below the onClick methods, start to add the runTimer() method:

```
private void runTimer() {
    final TextView timeView = (TextView)findViewById(R.id.time_view);

    final Handler handler = new Handler();
```

Maydm

The runTimer() Method

Next we'll add the handler.post() method with the Runnable:

```
handler.post(new Runnable() {
    @Override
    public void run() {
        int hours = seconds/3600;
        int minutes = (seconds%3600)/60;
        int secs = seconds%60;
```

Maydm

The runTimer() Method

Inside the run() method, add a String **time** that will turn the variables into the correct format:

```
String time = String.format(Locale.getDefault(),
        format: "%d:%02d:%02d", hours, minutes, secs);
```

Maydm

The String.format() Method

Let's look at the String.format() method. It takes three parameters:

- Locale, which is applied to the format() method
- The **format string**, a combination of plain text & **format specifiers**
- The **arguments** (variables) that will be added to the format string in order of the format specifiers: hours, minutes, seconds.

```
String time = String.format(Locale.getDefault(),
        format: "%d:%02d:%02d", hours, minutes, secs);
```

Maydm

The String.format() Method

Format specifiers are special characters that tell format() how to format the arguments. There are many specifiers but common ones include:

`%d` - integer
`%t` `%T` - time and date
`%s` `%S` - string

`%f` - floating decimal point
`%c` - character
`%b` `%B` - Boolean

```
String time = String.format(Locale.getDefault(),
        format: "%d:%02d:%02d", hours, minutes, secs);
```

Maydm

The String.format() Method

In our case, the `%d` format specifier also includes a flag to further format the number: `%02d` tells format to pad the number to two spaces, using zeros if necessary.

So if `minutes = 1`, then `%02d` will print it as 01.

```
String time = String.format(Locale.getDefault(),
        format: "%d:%02d:%02d", hours, minutes, secs);
```

Maydm

The runTimer() Method

Back to the runTimer() method. Now that we have time formatted properly, we update the timeView with the setText() method and, if running is true, increment the seconds variable.

```
timeView.setText(time);
if (running) {
    seconds++;
}
```

Maydm

The runTimer() Method

Last, and certainly not least, we call the postDelayed() method. We pass in **this**, which refers to the Runnable object, and 1000 milliseconds as the delay.

Make sure you have all your closing brackets and parentheses!

Maydm

What happens next?

1. User clicks the app icon.
2. An intent is constructed to start using startActivity(intent).
    a. The AndroidManifest.xml file specifies which activity to use as the launch activity.
3. Android checks whether there is already a process running for the app then creates a new activity object -- here, it's StopwatchActivity.
4. The onCreate() method in the activity is called.
    a. The layout is specified with setContentView() and runTimer() starts the stopwatch.
5. When onCreate() finishes, the layout is displayed on the device.
    a. runTimer() determines the text to display from the seconds variable, which is 0, so the text starts out at 0:00:00.

Maydm

Questions to think about

Q: Why does Android run each app in a separate process?
A: Security and stability.

Q: Why use onCreate() in the activity and not the constructor?
A: Android has to set up the environment first, and then calls onCreate.

Q: Why not just write a loop in onCreate() to update the timer?
A: The onCreate() method must finish before the screen will appear. A loop inside of onCreate would prevent onCreate from finishing.

Test Drive the App

See how your code works on an actual device. Make sure the start, stop, and reset buttons all work like they should.

If something isn't working, review your code to find the bug.

Test Drive the App

Did you try to rotate the screen? If not, do so now.

What happened?!

On rotation the counter resets to 0.

It's surprisingly common for apps to break on screen rotation. We'll fix it but first we'll look at why it happens.

Why the Timer Resets on Rotation

1. User clicks the app icon and starts the stopwatch by clicking the Start button.
2. The user rotates the device.
3. Android sees the orientation and screen size have changed, so it destroys the activity -- including any variables that are in use by the runTimer() method.
4. Android recreates the StopwatchActivity, which means the variables revert back to their default values.

Discussion

Why does Android consider screen rotation a change to the device configuration?

Activity Lifecycle

onCreate() runs immediately after activity is launched.

Activity spends most of its life in this stage.

onDestroy() runs before it's destroyed for final cleanup.

Maydm

Lifecycle Methods

Because the activity extends the **`android.app.Activity`** class, it gains access to all the methods available to it, including:

- `onCreate(Bundle)`
- `onStart()`
- `onRestart()`
- `onResume()`
- `onPause()`
- `onStop()`
- `onDestroy()`
- `onSaveInstanceState()`
- `startActivity(Intent)`
- `findViewById(Int)`
- `setContentView(View)`

Maydm

Lifecycle Methods

Most behavior is handled by the superclass methods that are inherited from the Activity class. You can override the methods you want to change, like this:

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

Fixing the Rotation Problem

The best way to deal with configuration changes is by saving the current state of the activity and reinstating it in onCreate(). This is done with the onSaveInstanceState() method.

This method is called before an activity is destroyed, which will allow you to save any values you might want to keep.

onSaveInstanceState() Method

This method takes one paramenter, a **Bundle**. A Bundle allows you to gather different types of data into one object.

The method looks something like this:

```
public void onSaveInstanceState(Bundle saveInstanceState){
    ...
}
```

Maydm

onSaveInstanceState() Methods

onCreate() will receive the Bundle as a parameter. This means if you save your variables in the Bundle, they will be available to the onCreate() method whenever the activity is recreated.

Do this with the Bundle put*() methods, which creates name/value pairs. The methods look like this:

```
bundle.put*("name", value);
```

Where bundle is the name of the Bundle, * is the type of value, and name/value are the name and value of the data you want to save.

Maydm

onSaveInstanceState() Methods

Here's how we will combine all the blocks. Add this below the onCreate() method in your StopwatchActivity file:

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

Now we can pass Bundle to onCreate() to access the saved values!

Maydm

Accessing savedInstanceState

When we set up the onCreate() method, we already passed in the Bundle savedInstanceState. Now we need to use a bundle method to access the values we stored there.

These methods will look similar to the put*() methods we just used:

```
bundle.get*("name");
```

Where bundle is the name of the Bundle, * is the type of value, and name refers to the name of the name/value pair that was saved.

Maydm

Accessing savedInstanceState

Now let's put it together with the onCreate() method. Inside the onCreate() block, add the bolded code after setContentView and before the call to runTimer():

```
setContentView(R.layout.activity_stopwatch);
if (savedInstanceState != null) {
    seconds = savedInstanceState.getInt( key: "seconds");
    running = savedInstanceState.getBoolean( key: "running");
}
runTimer();
```

Maydm

Try it out!

The stopwatch should keep working if the screen orientation changes.

If it doesn't, make sure you added the onSaveInstanceState() method inside the class, and the bundle.get* methods inside onCreate().

Journal Time

How do activities and life cycles provide architecture for apps?

Maydm