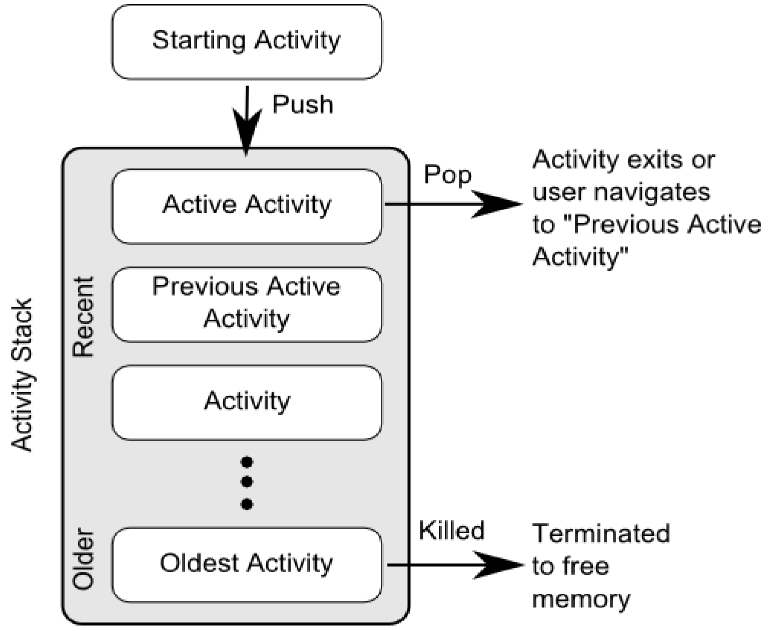
Activity

- An app is a collection of activities, layouts, and other resources.
  - ✓ One of the activities is the main activity for the app.
- By default, each app runs within its own process.
  - ✓ This helps keep your apps safe and secure. If the system identifies that resources on the device are reaching capacity it will take steps to terminate processes to free up memory.
- When an activity needs to start, Android checks whether 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.

- An activity will interact with the UI component (User Interface) by using setContentView(View)
- An activity provides the window in which the app draws its UI.
  - ✓ Generally, one activity implements one(single) screen in an app.
- To use activities in your app, you must register information about them in the app's manifest, and you should manage activity lifecycles appropriately.
- The life cycle states (and callbacks) are per activity not per application, so we can implement different behavior at different points in the lifecycle of each Activity.
- Next, the Activity Stack and life cycle will be discussed.

Activity Stack

- For each application that is running on an Android device, the runtime system maintains an Activity Stack.
- When an application is launched, the first of the application’s activities to be started is placed onto the stack.
- When a second activity is started, it is placed on the stack and the previous activity will be pushed down.



Life Cycle of Android Activity

Method	When it is called	NextMethod
onCreate()	When the activity is first created. It also gives you a Bundle that contains the previously saved state of the activity.	onStart()
onRestart()	When your activity has been stopped but just before it gets started again.	onStart()
onStart()	When your activity is becoming visible.	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	onResume() or onStop()
onStop()	When the activity is no longer visible	onRestart() or onDestroy()
onDestroy()	When your activity is about to be destroyed or because the activity is finishing.	None

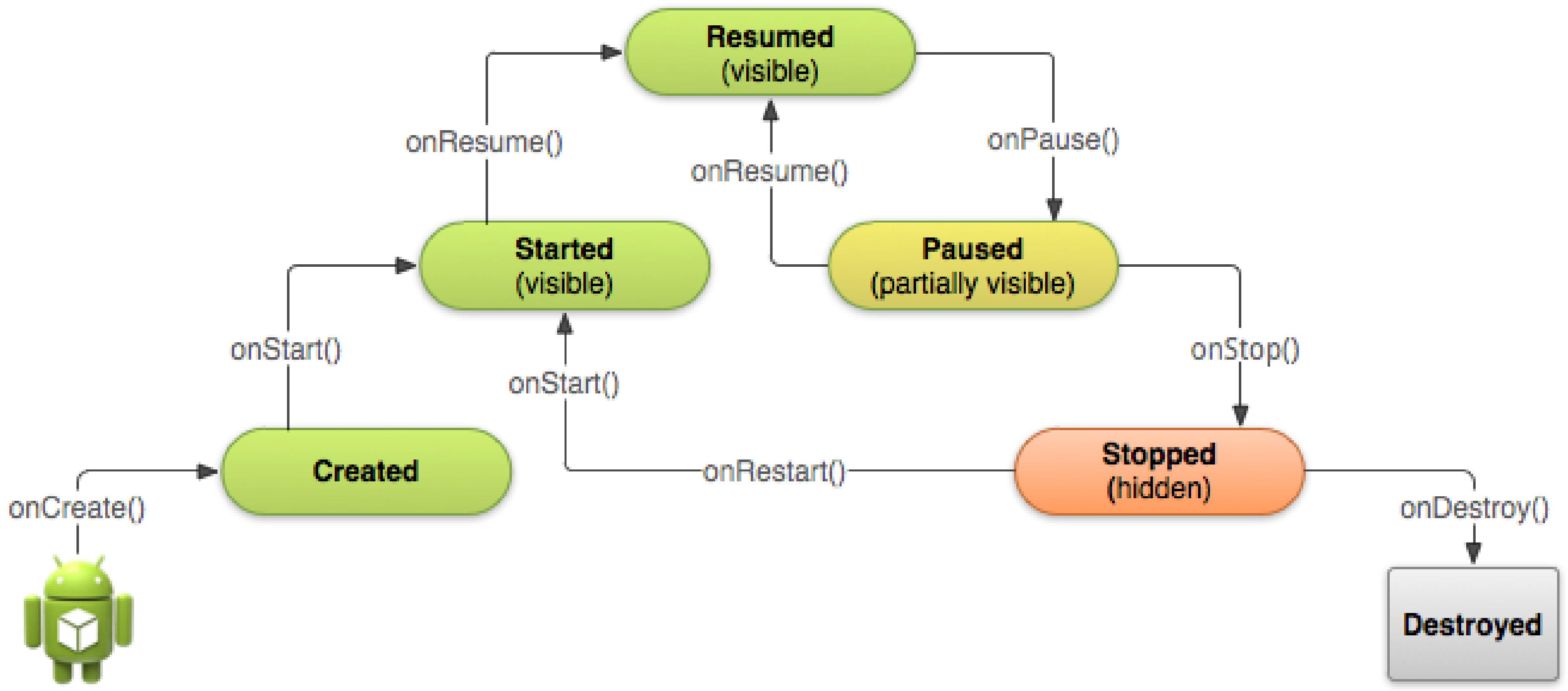


Fig Visual Representation of the Activity lifecycle

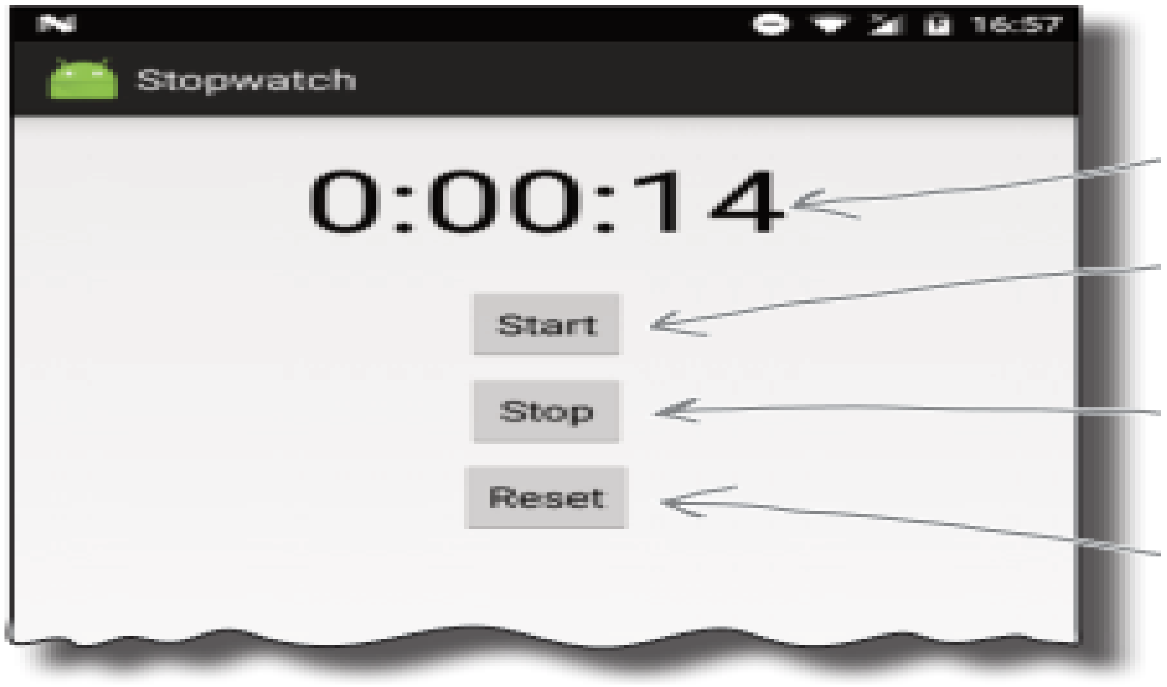
- Examples that show some situation call backs methods will be called (implemented):
- OnPause():
  - ✓ Some event interrupts app execution, like receiving a phone call, the user’s navigating to another activity, or the device screen’s turning off.
  - ✓ In Android 7.0 (API level 24) or higher, multiple apps run in multi-window mode. Because only one of the apps (windows) has focus at any time, the system pauses all of the other apps.
  - ✓ A new, semi-transparent activity (such as a dialog) opens. As long as the activity is still partially visible but not in focus, it remains paused.

- OnStop():
  - ✓ When a newly launched activity covers the entire screen.
  - ✓ The user navigates to the device’s home screen.
- OnDestroy():
  - ✓ the activity is finishing (due to the user completely dismissing the activity).
  - ✓ the system is temporarily destroying the activity due to a configuration change (such as device rotation, language).

- Next, we try to see how we can handle activity life cycle using Stop Watch Activity.

Stop Watch Activity

- ❖ Layout
- ❖ Use
  - ✓ Linear Layout
  - ✓ Relative Layout Or
  - ✓ Constraint Layout



❖Activity Code

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

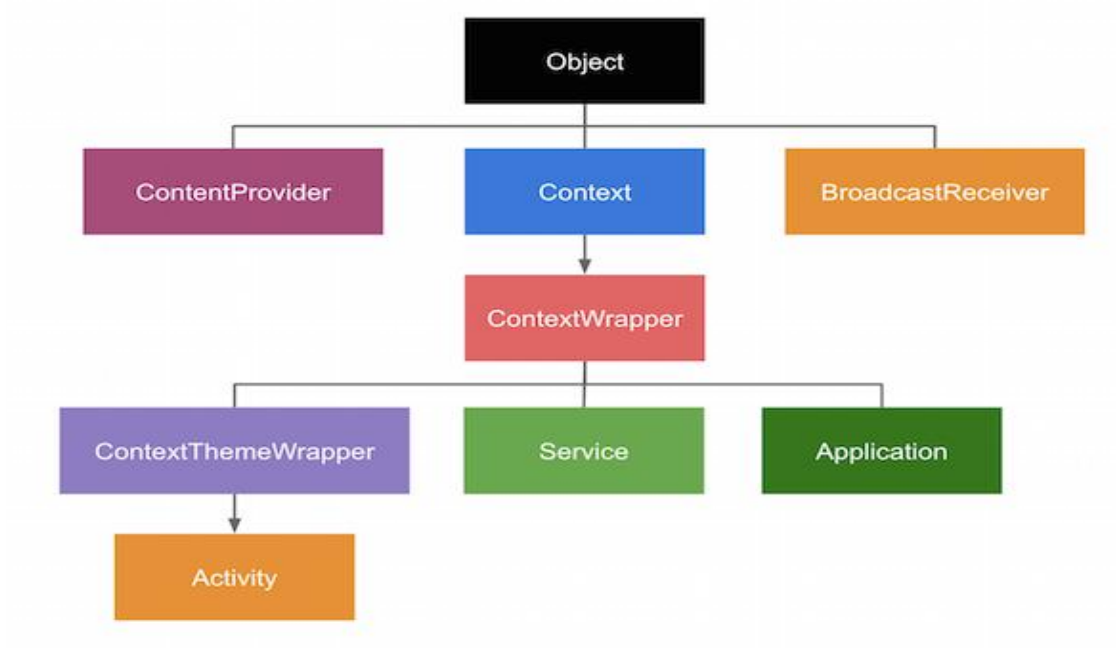
- A Handler is an Android class you can use to schedule code that should be run at some point in the future.
- To use the Handler, you wrap the code you wish to schedule in a Runnable object, and then use the Handler Post() and postDelayed() method to specify when you want the code to run.
- The post() method posts code that needs to be run as soon as possible (which is usually almost immediately).

```
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 method takes one parameter, an object of type Runnable.
- The postDelayed() method works in a similar way to the post() method except that you use it to post code that should be run in the future.
- This method takes two parameters, A Runnable and a long.

Cont’d [Activity Lifecycle]

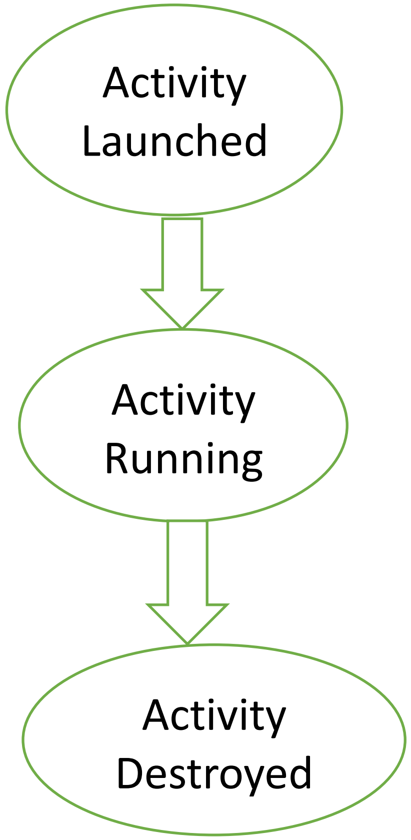
- Your Activity inherits the lifecycle methods.
- Context: An interface to global information about the application environment allows access to application resources, classes, and operations.
- Activity: The Activity class implements default versions of the lifecycle methods. It also defines methods such as findViewById(Int) and setContentView(View).



States of an Activity

- The main state of an activity is when it’s running or active. An activity is running when it’s in the foreground of the screen, it has the focus, and the user can interact with it.
- The onCreate() method gets called immediately after your activity is launched. This method is where you do all your normal activity setup such as calling setContentView().
- The onDestroy() method is the final call you get before the activity is destroyed. There are a number of situations in which an activity can get destroyed.

❖ for example, if it’s been told to finish, if the activity is being recreated due to a change in device configuration, or if Android has decided to destroy the activity in order to save space.



Save the current state (onCreate/onDestroy)

- When we change device configuration, like orientation the app will loss local variables used by the activity.

How we restore the value?

- We need to implement the onSaveInstanceState() method. This method gets called before the activity gets destroyed, which means you get an opportunity to save any values you want to retain before they get lost.
- onSaveInstanceState() method takes one parameter, a Bundle. A Bundle allows you to gather together different types of data into a single object.

```
public void onSaveInstanceState(Bundle savedInstanceState)
{
}
```

- Add these two codes in the activity

Save the values of the seconds and running variables to the Bundle.

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

Restore the state in onCreate.

```
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();
}
```

- There are additional lifecycle methods which deal with activity’s visibility. onStart(), onStop(), and onRestart().
- We inherit from Android Activity class like onCreate() and onDestroy().
- onStart() gets called when your activity becomes visible to the user.
- onStop() gets called when your activity has stopped being visible to the user. This might be because it’s completely hidden by another activity that’s appeared on top of it.
- onRestart() gets called after your activity has been made invisible, before it gets made visible again.

Implement onStop() to stop the timer

- We have to add a new variable to record 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.

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

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

- savedInstanceState.putBoolean("wasRunning", wasRunning);
- wasRunning = savedInstanceState.getBoolean("wasRunning");

Implement onPause() to stop the timer

- When an activity is visible but doesn’t have the focus, the activity is Paused. This can happen if another activity appears on top of your activity that isn’t full-size or that’s transparent.

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

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

Conclusion

- Properly use the Activity Life cycle can help ensure that your app avoids:
  - ✓ Consuming valuable system resources when the user is not actively using it.
  - ✓ Losing the user’s progress if they leave your app and return to it at a later time.
  - ✓ Crashing or losing the user’s progress when the screen rotates between landscape and portrait orientation or switches to another app while using your app.