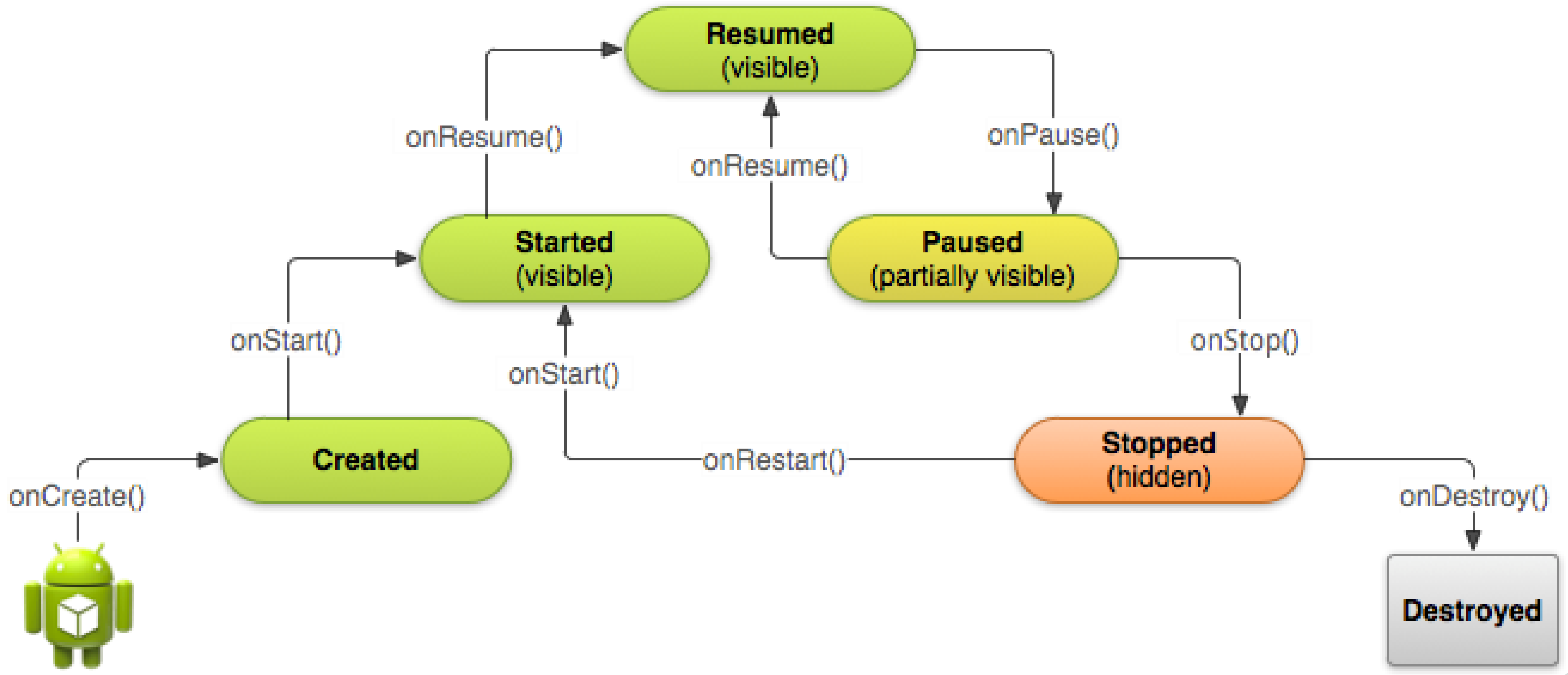
Android Development: We’re Getting There

Al-Kandari, AbdulMuhsin

Blessing, James

Activity Life Cycle Diagram



Walking Through an Activity’s Lifecycle

- Any android activity can be in one of six states: Created, Started, Resumed, Paused, Stopped and Destroyed.
- Walkthrough:
  - When an Activity is first “summoned” either by the start of the app or by an intent the onCreate() method fires, which quickly calls onStart(), which is followed closely by onResume().
  - The activity is now in the Resumed state, this is the state that the user interacts with and can be referred to as the running state.
  - From the Resumed state the activity can go to the Paused state which is called through the onPause() method, the paused state will only persist if the activity is partially obscured by another activity or fragment (an Example of this are dialogs or popups). Note: No input can be given to the activity in this state.
  - If the activity is completely hidden and not visible the Paused state will move into a Stopped state through onStop(). In this state the instance and the state information is retained. It is from this state that an activity can be Destroyed.

General Occurrences in Methods

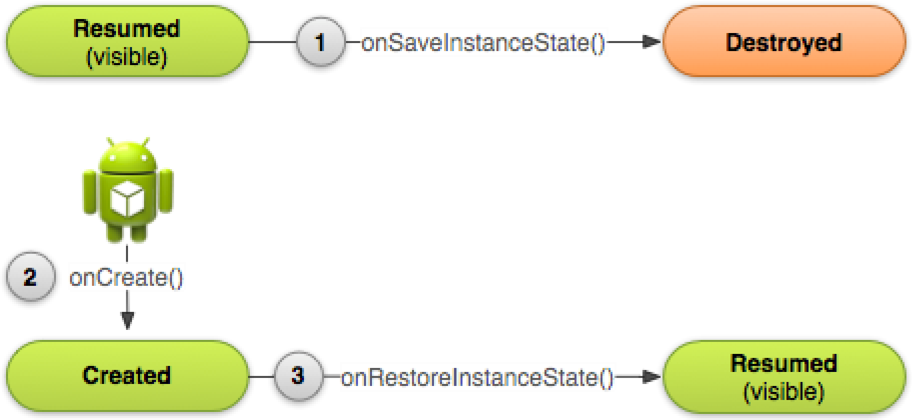
- onStart(): verify that the required system features are enabled
- onResume(): initialize needed components (begin animations, or components that are only used while the activity has user focus like the camera)
- onPause(): Stop animations, commit changes that a user would expect to be permanently saved (such as a draft email), and release system resources (broadcast recievers, gps, etc.)
  - Avoid CPU intensive work here because it can slow transition to the next activity
- onStop(): release almost all resources (in extreme cases android might kill your app process without calling onDestroy() so it is important you don’t leave resources that leak memory), also perform larger more intensive shut-down operations (e.g. writing to a database)

Notes: What You Need to Know

- Only 3 states are static: Resumed, Paused and Stopped.
- For lower complexity applications, a need to implement ALL the lifecycle methods is highly unlikely. It is still important to understand each one even if you do not need to implement it.
- Implementation of these lifecycle methods may be the only way to ensure your app behaves the way users expect.
- Implementing these methods properly stops the following problems from occurring:
  - Crashes after phone calls or after switching to different apps.
  - Consumption of system resources while the user is not using them.
  - The user can no longer progress if they leave your app and return at a later time.
  - The program crashes or the user loses progress when the screen rotates between landscape and portrait.

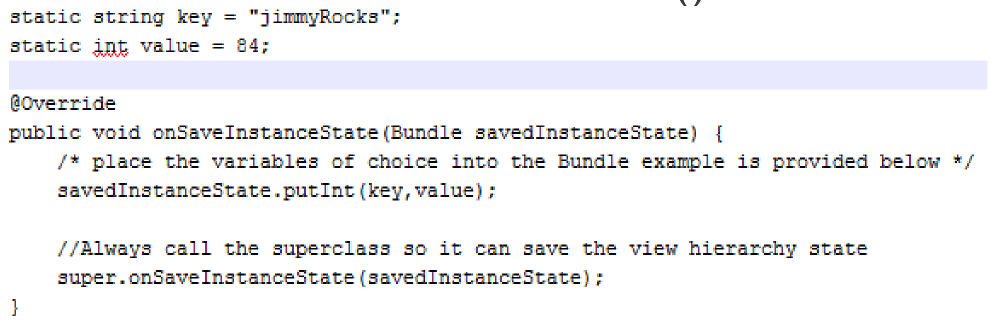
onCreate() & onDestroy(): We Didn’t Forget You, You’re Just Special

- When destroying an activity, you often come across data that you would like to reload when the activity is recreated (especially when the screen rotates as switch from landscape to portrait or vice-versa causes the activity to be destroyed)
- The data that you want to reload should not be stored permanently, Android solves this using Bundle objects and the onSaveInstanceState() and onRestoreInstanceState().



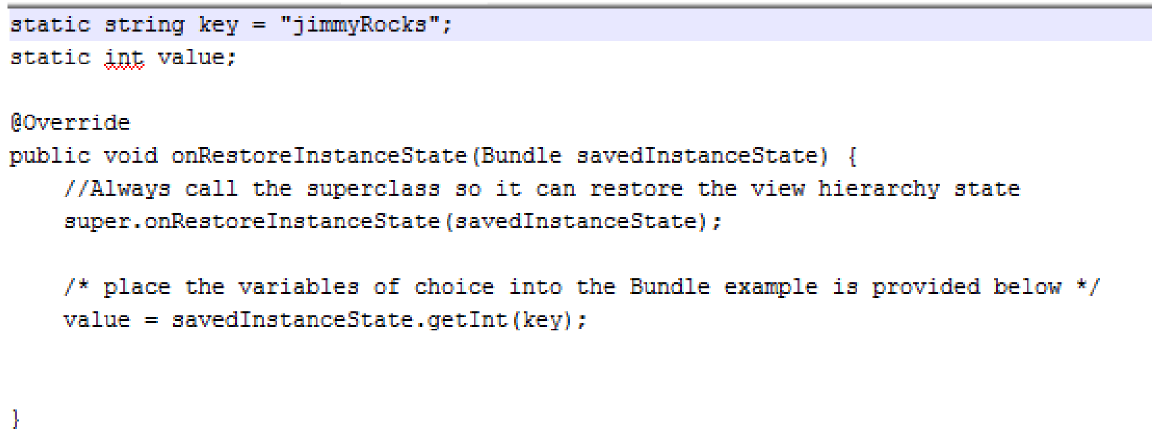
onCreate() & onDestroy(): Continued

- The format of the onSaveInstanceState() method is as follows:



```java
static string key = "jimmyRocks";
static int value = 84;

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    /* place the variables of choice into the Bundle example is provided below */
    savedInstanceState.putInt(key,value);

    //Always call the superclass so it can save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```

- The format of the onRestoreInstanceState() method is as follows

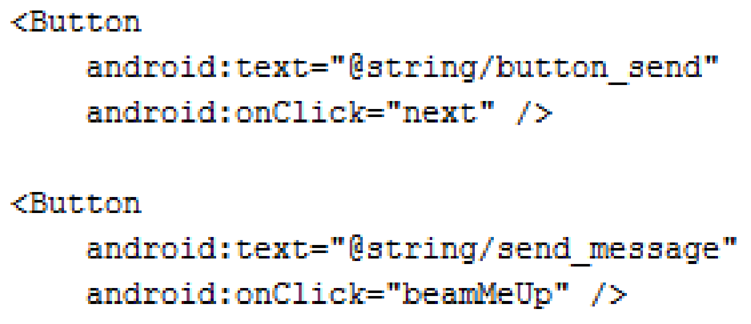


```java
static string key = "jimmyRocks";
static int value;

@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    //Always call the superclass so it can restore the view hierarchy state
    super.onRestoreInstanceState(savedInstanceState);

    /* place the variables of choice into the Bundle example is provided below */
    value = savedInstanceState.getInt(key);


}
```

Intents: Telling Your App Where to Go

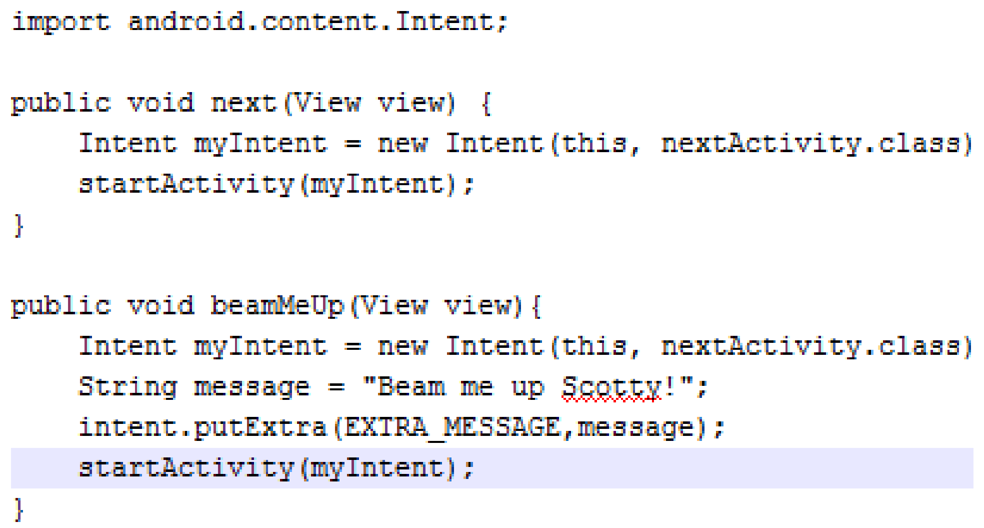
- When moving from an activity to another activity in android you have to declare your intent.
- An intent is an abstract description of an operation to be performed.
- The structure of an initializing an intent is as follows:
  - Intent myIntent = new Intent(this, nextActivity.class);
    - Intent: is the object type you are creating
    - myIntent: the name of your object
    - this: current activity
    - nextActivity.class: the name of your next activity, note that you are calling the java file not the xml file.

Intents: Moving on With A Click of a Button



```
<Button
    android:text="@string/button_send"
    android:onClick="next" />

<Button
    android:text="@string/send_message"
    android:onClick="beamMeUp" />
```

An activity with two buttons in XML



```
import android.content.Intent;

public void next(View view) {
    Intent myIntent = new Intent(this, nextActivity.class)
    startActivity(myIntent);
}

public void beamMeUp(View view){
    Intent myIntent = new Intent(this, nextActivity.class)
    String message = "Beam me up Scotty!";
    intent.putExtra(EXTRA_MESSAGE,message);
    startActivity(myIntent);
}
```

The java methods for each button

Exercise:

- Build an app that illustrates the different activity states, is able to switch between activities, and illustrates saved instances. (remember that the saved instances are activity specific not application specific)