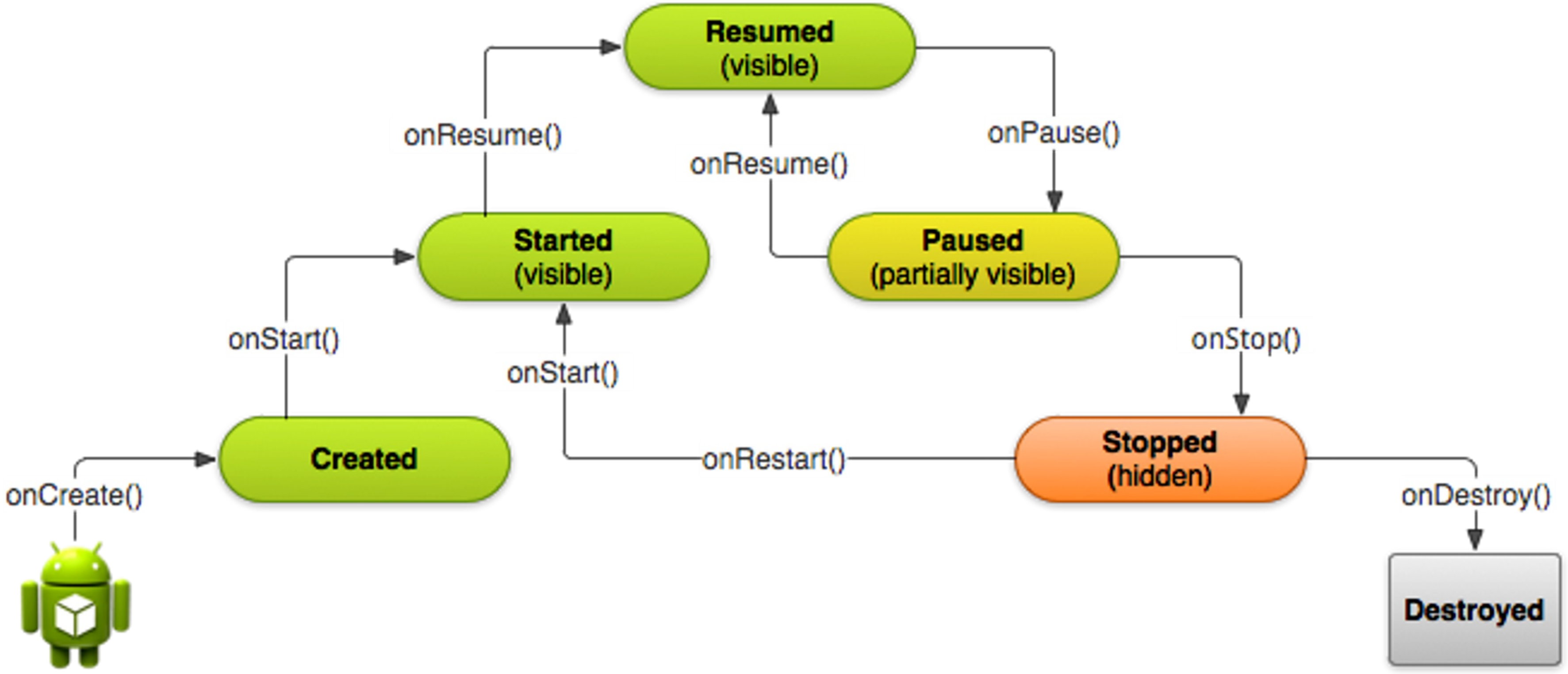
Mobile Computing

Activity Lifecycle

Step Pyramid of Lifecycle

- Each stage of the activity lifecycle is a separate step on the pyramid
- Each callback method moves the activity state one step toward the top
- As the user begins to leave the activity, the system calls other methods that move the activity state back down the pyramid



Importance of Lifecycle

- Implementing the activity lifecycle methods properly ensures the app behaves well in several ways;
  - Does not crash if the user receives a phone call or switches to another app while using your app.
  - Does not consume valuable system resources when the user is not actively using it.
  - Does not lose the user's progress if they leave your app and return to it at a later time.
  - Does not crash or lose the user's progress when the screen rotates between landscape and portrait orientation.

Static States

- Resumed (or Running)
  - In this state, the activity is in the foreground and the user can interact with it.
- Paused
  - In this state, the activity is partially obscured by another activity—the other activity that's in the foreground is semi-transparent or doesn't cover the entire screen. The paused activity does not receive user input and cannot execute any code.
- Stopped
  - In this state, the activity is completely hidden (not visible to the user) and it is considered to be in the background. The activity instance and all its state information such as member variables is retained, but it cannot execute any code.

Declaring the Main Activity

- The main activity for your app must be declared in the manifest with an **<intent-filter>** that includes
  - The **MAIN** action
  - **LAUNCHER** category

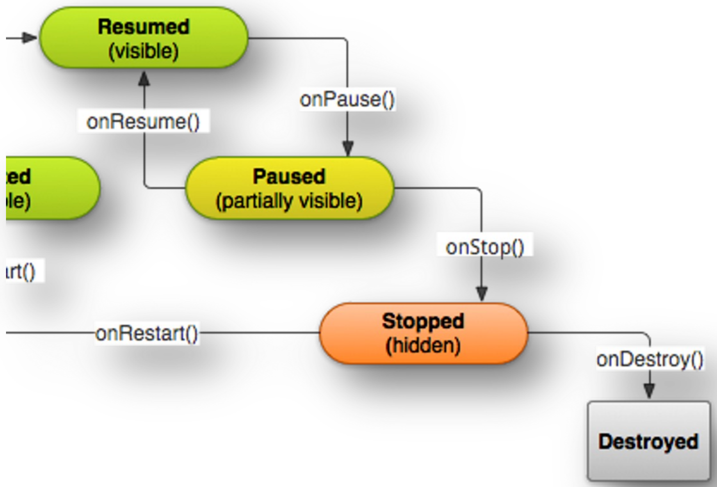
```
<activity android:name=".MainActivity" android:label="@string/app_name">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Create a New Instance

- The system creates every new instance of **Activity** by calling its **onCreate()** method
- The implementation of **onCreate()** method performs basic application startup logic that should happen only once for the entire life of the activity.
- Once the **onCreate()** finishes execution, the system calls the **onStart()** and **onResume()** methods in quick succession.
- The system never calls the **onCreate()** again in lifetime of the activity.
- The activity becomes visible to the user when **onStart()** is called, but **onResume()** quickly follows and the activity remains in the Resumed state until something occurs, i.e.
  - a phone call is received
  - the user navigates to another activity
  - the device screen turns off

Destroy the Activity

- The last callback is **onDestroy().**
- The system calls this method on the activity as the final signal that the activity instance is being completely removed from the system memory.
- Generally **onDestroy()** method does not need to implement.
- **onDestroy()** method may implemented in the following reason
  - Background threads that were created during onCreate()
  - Long-running resources that could potentially leak memory if not properly closed
- The system calls **onDestroy()** after it has already called **onPause()** and **onStop()** in all situations except:
  - When **finish()** is invoked within the **onCreate()** method.



Pausing and Resuming an Activity

- The activity pauses when,
  - A semitransparent activity appears
  - User is leaving the application
- Once the activity is fully-obstructed and not visible, it stops.
- The onPause() callback usually uses to:
  - Commit unsaved changes, but only if users expect such changes to be permanently saved when they left.
  - Release system resources, such as
    - broadcast receivers
    - handles to sensors
    - any resources that may affect battery life while your activity is paused and the user does not need them.
- When the user resumes your activity from the Paused state, the system calls the onResume() method.
- The system calls this method
  - every time the activity comes into the foreground
  - when it's created for the first time.

Stopping and Restarting an Activity

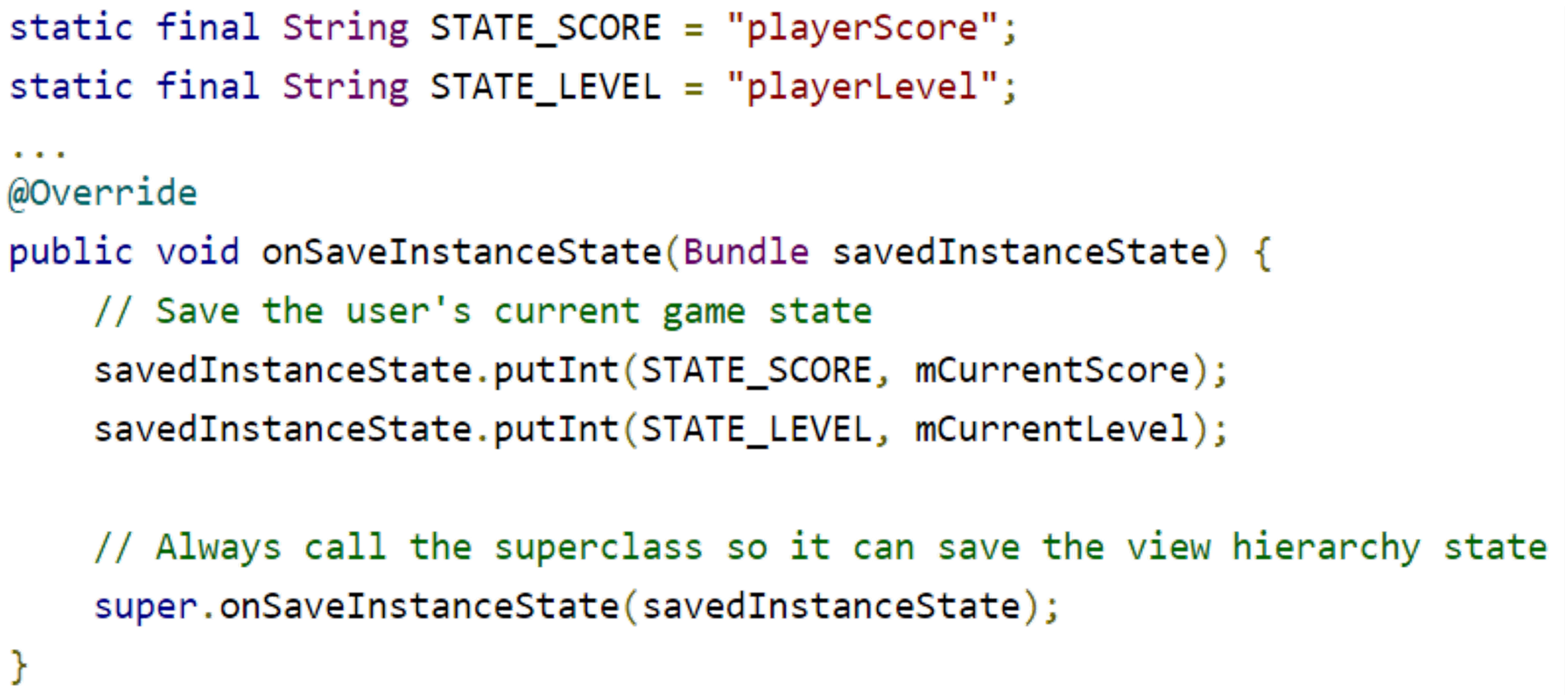
- An activity is stopped and restarted when:
  - The user opens the Recent Apps window and switches from the app to another app. The activity in your app that's currently in the foreground is **stopped**. If the user returns to your app from the Home screen launcher icon or the Recent Apps window, the activity **restarts**.
  - The user performs an action in the app that starts a new activity. The current activity is **stopped** when the second activity is **created**. If the user then presses the *Back* button, the first activity is **restarted**.
  - The user receives a phone call while using your app on his or her phone the current activity is **stopped**.
- the system retains the Activity instance in system memory when it is stopped.
  - Do not need to implement the onStop() and onRestart() most of the cases
  - May use onPause() to pause ongoing actions and disconnect from system resources.
- **onStop()** can be used to perform larger, more CPU intensive shut-down operations, such as writing information to a database.
- Even if the system destroys the activity, it still retains the state of the View objects in a **Bundle**
- When the activity comes back to the foreground from the stopped state, it receives a call to **onRestart()** and **onStart()**.

Recreating an Activity

- An activity can be destroyed due to normal app behavior by calling **finish()**
- The system may also destroy an activity if
  - It is currently stopped and hasn't been used in a long time
  - the foreground activity requires more resources
- When an activity is destroyed because the user presses *Back* or the activity finishes itself, the system's concept of that Activity instance is gone forever.
- If the system destroys the activity due to system constraints then although the actual Activity instance is gone, the system remembers that it existed.
  - if the user navigates back to it, the system creates a new instance using a set of saved data that describes the state of the activity when it was destroyed.
- The saved data is called the "**instance state**" and is a collection of key-value pairs stored in a **Bundle** object.
- Activity will be destroyed and recreated each time the user rotates the screen.
  - activity might need to load alternative resources

Bundle Object

- When an activity is recreated after it was previously destroyed, the saved state can be recovered from the **Bundle** that the system passes the activity.
- Both the **onCreate()** and **onRestoreInstanceState()** callback methods receive the same **Bundle** that contains the instance state information.
- Store data in the Bundle instance

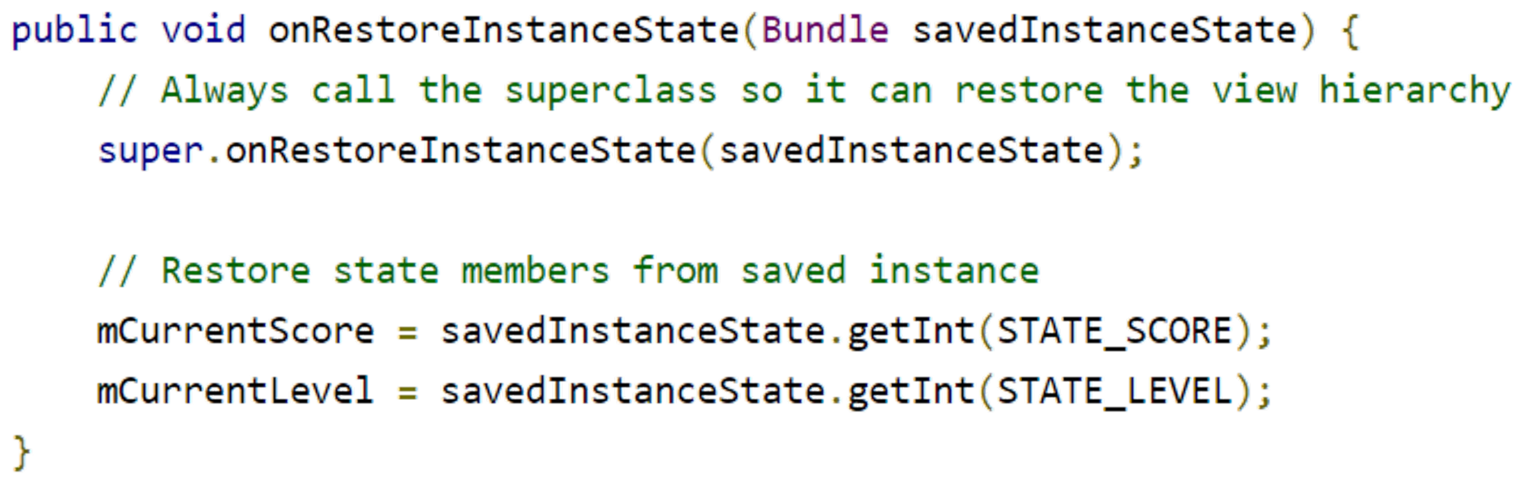


```java
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

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

Bundle Object Cont.

- Get data from the Bundle instance on restore call back



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

    // Restore state members from saved instance
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
}
```