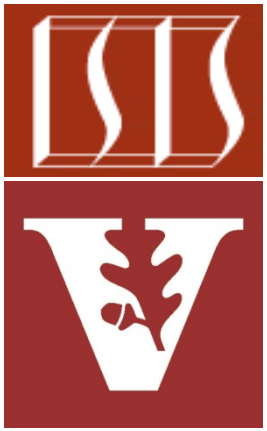
Overview of Activities



Douglas C. Schmidt
d.schmidt@vanderbilt.edu

www.dre.vanderbilt.edu/~schmidt

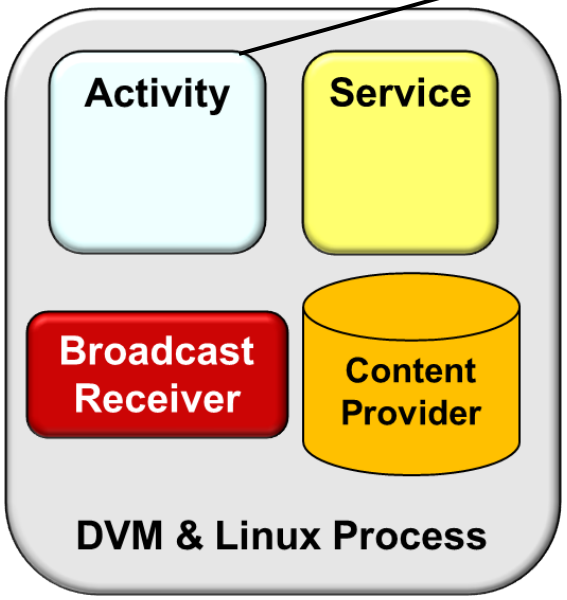
Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

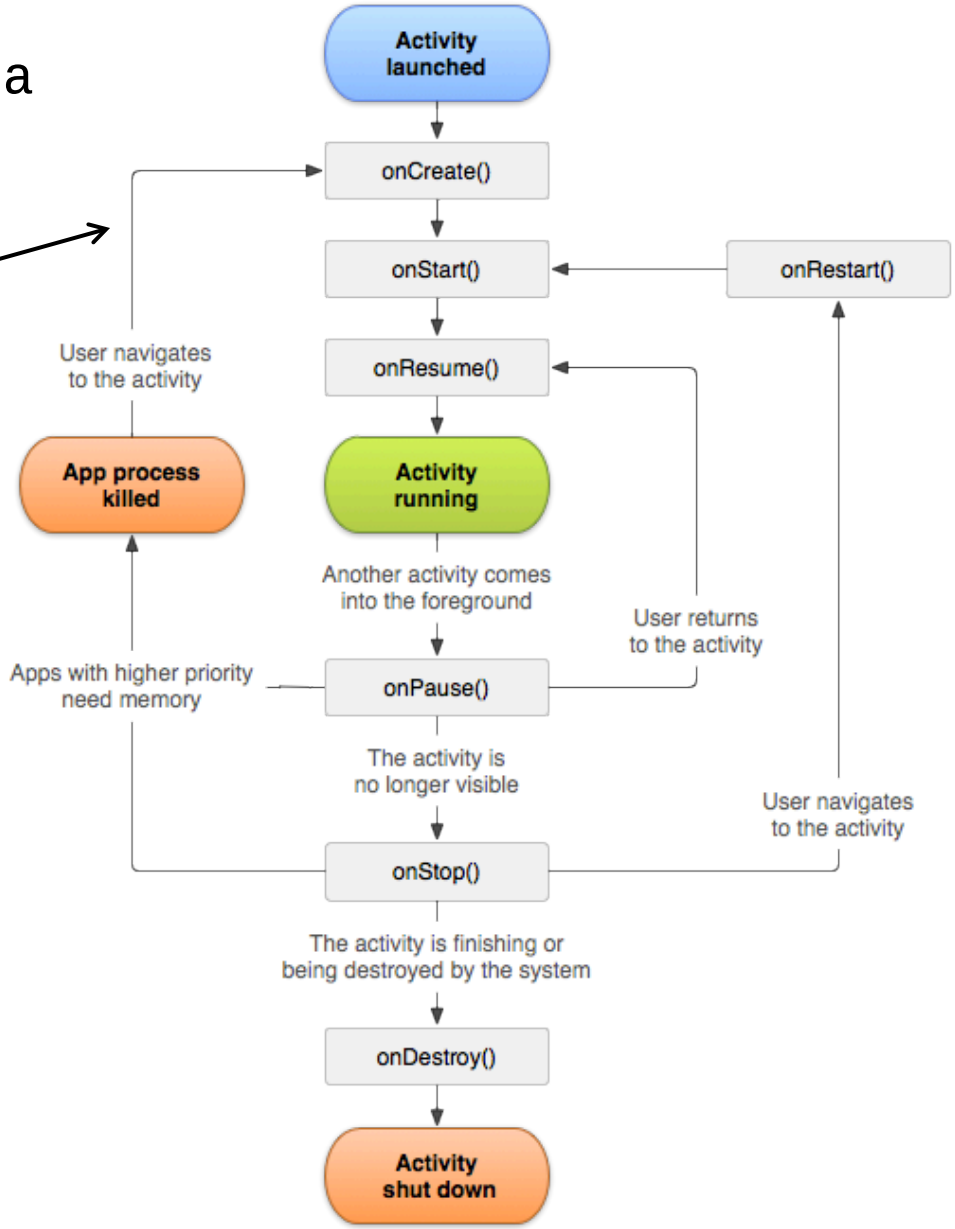
CS 282 Principles of Operating Systems II

Systems Programming for Android

Learning Objectives of this Module

- Understand how an Activity provides a visual interface for user interaction





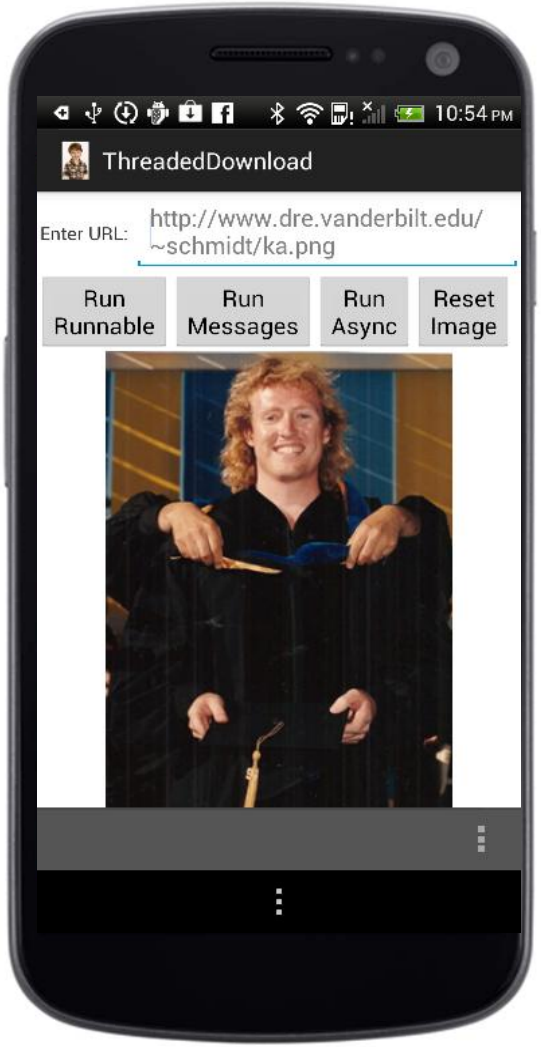
We'll emphasize commonalities & variabilities in our discussion

Overview of an Activity

- An Activity provides a visual interface for user interaction

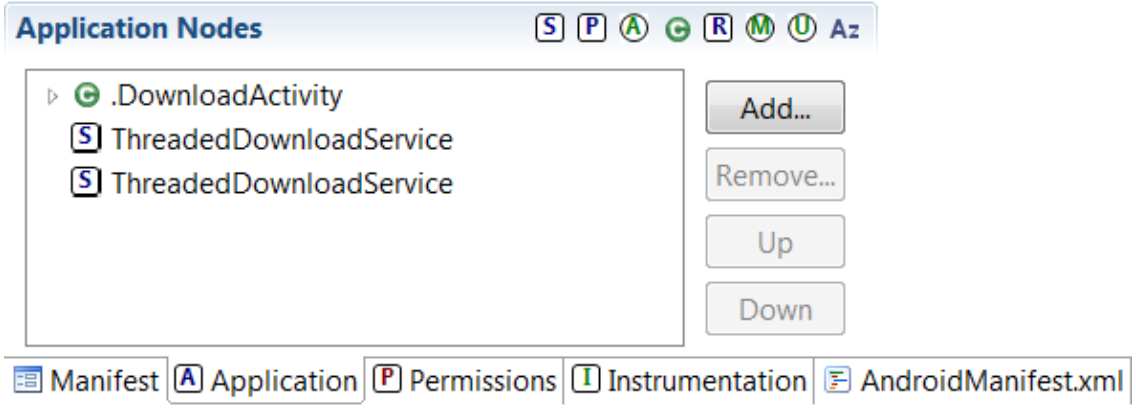
Overview of an Activity

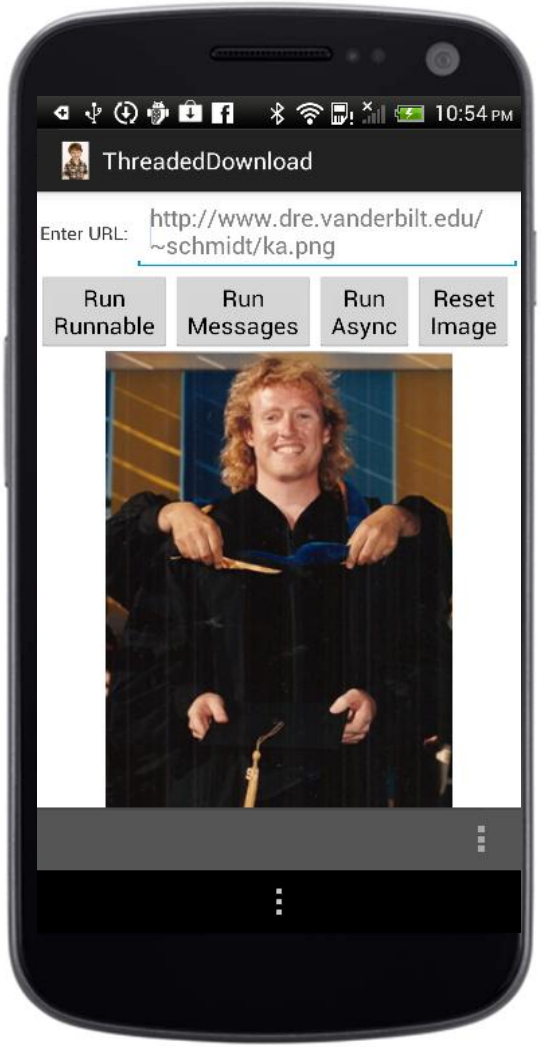
- An Activity provides a visual interface for user interaction
- Typically supports one thing a user can do, e.g.:
  - Show a login screen
  - Read an email message
  - Compose a text message
  - View a contact
  - Browse the Internet
  - etc.



Overview of an Activity

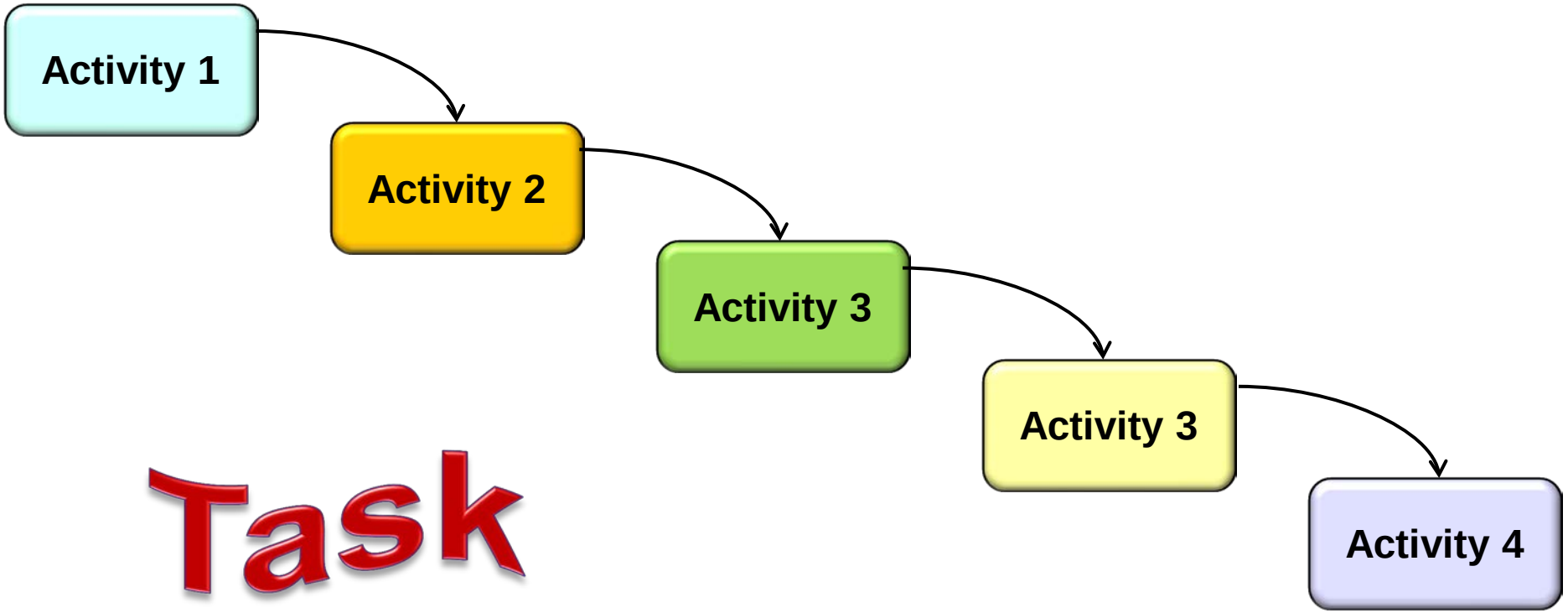
- An Activity provides a visual interface for user interaction
- Typically supports one thing a user can do, e.g.:
  - Show a login screen
  - Read an email message
  - Compose a text message
  - View a contact
  - Browse the Internet
  - etc.
- Applications can include one or more activities





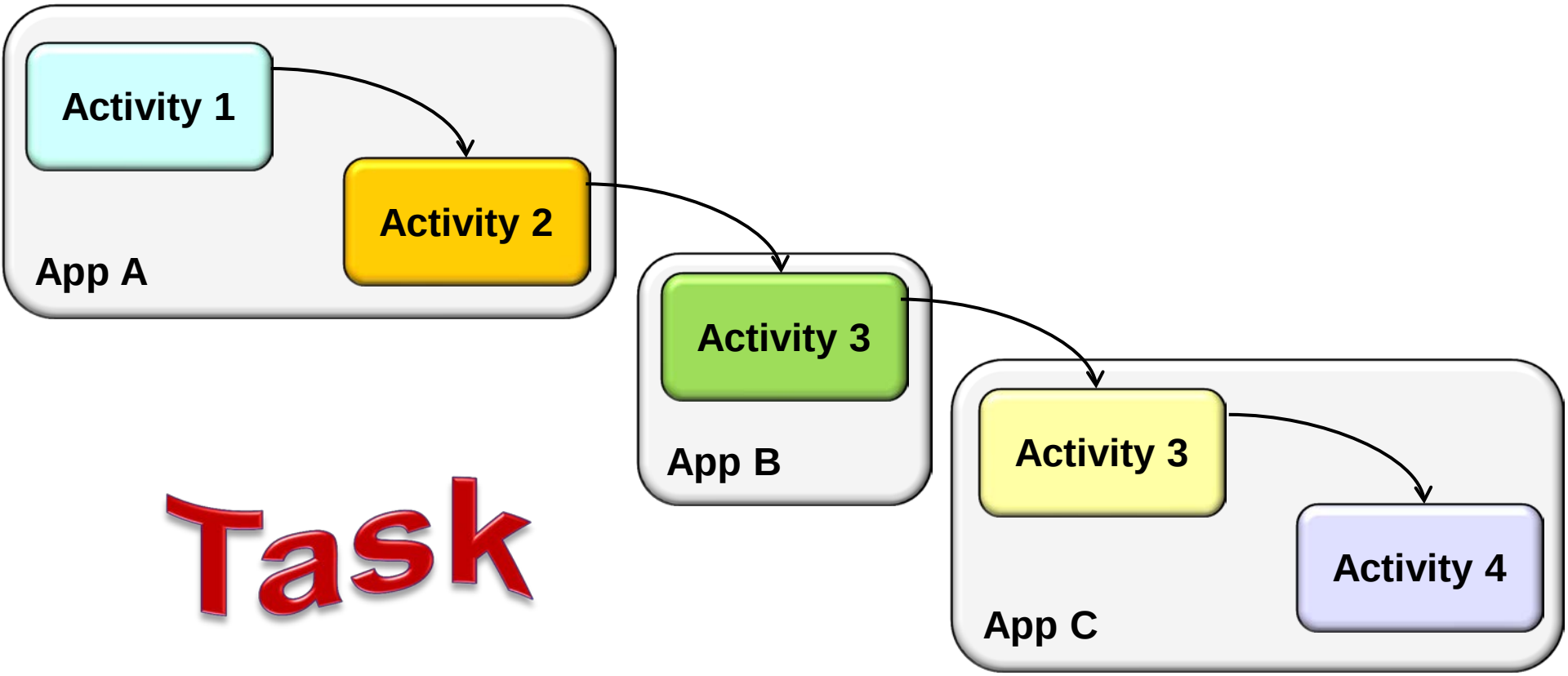
Tasks

- A Task is a chain of related Activities



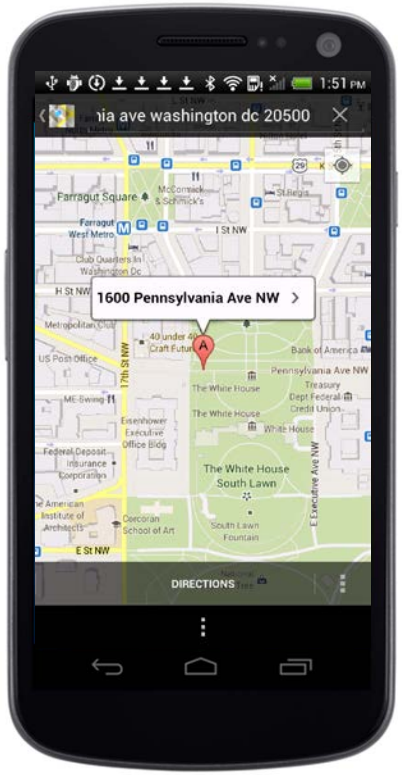
Tasks

- A Task is a chain of related Activities
  - Task are not necessarily provided by a single app



Tasks

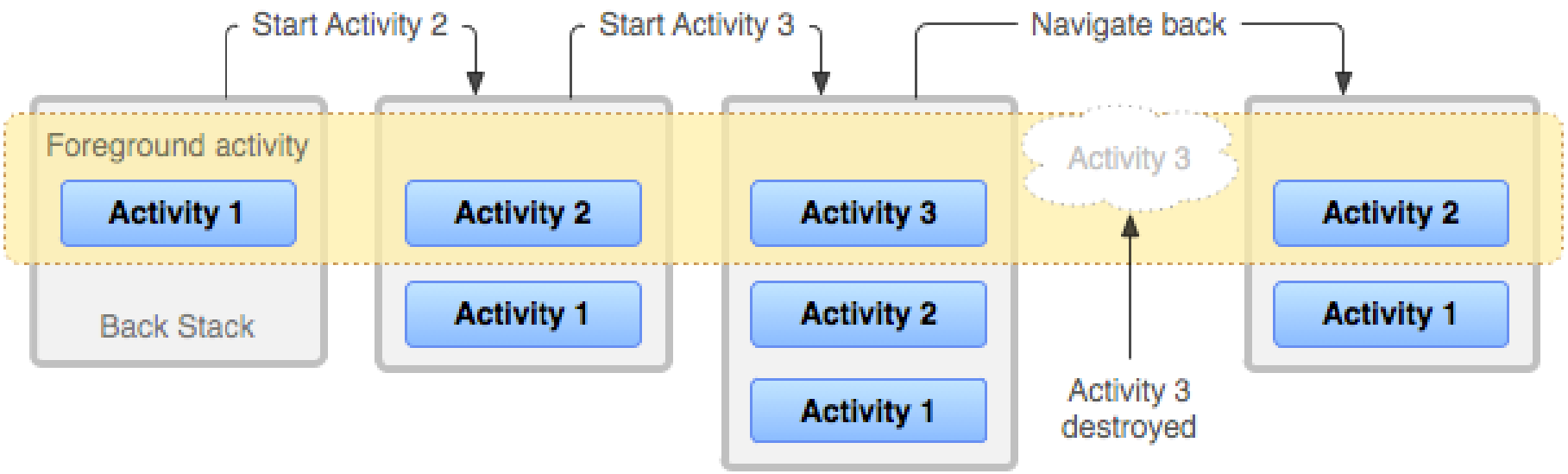
- A Task is a chain of related Activities
  - Task are not necessarily provided by a single app
- Tasks give the illusion that multiple (often unrelated) Activities were developed as part of the same app



developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html

Tasks

- The task's Activity objects are stored on a "back stack" with the currently running Activity at the top

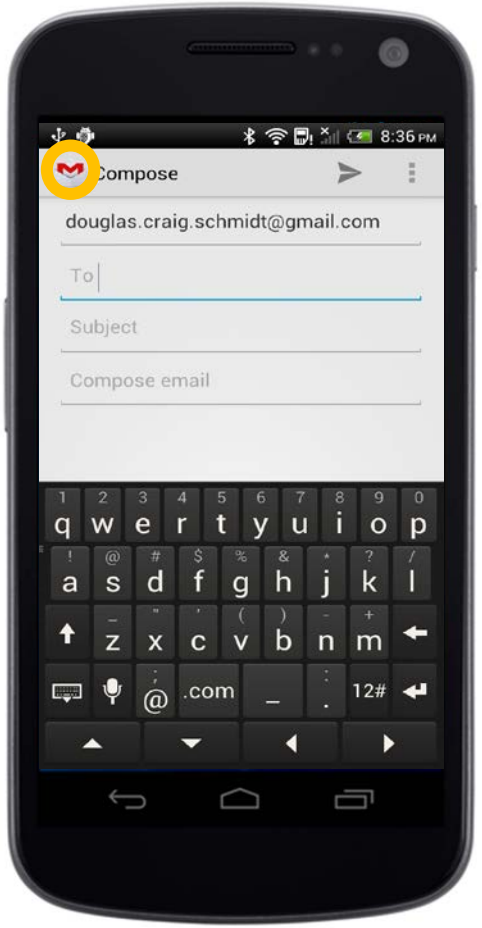


Tasks

- The task's Activity objects are stored on a "back stack" with the currently running Activity at the top
- At runtime
  - Launching an Activity places it on top of the stack

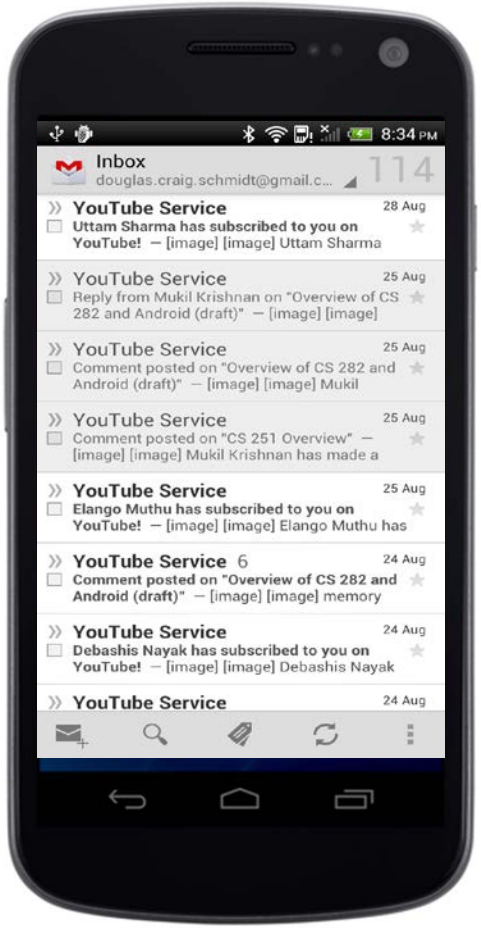
Tasks

- The task's Activity objects are stored on a "back stack" with the currently running Activity at the top
- At runtime
  - Launching an Activity places it on top of the stack
  - Finishing an Activity pops it off the stack...



Tasks

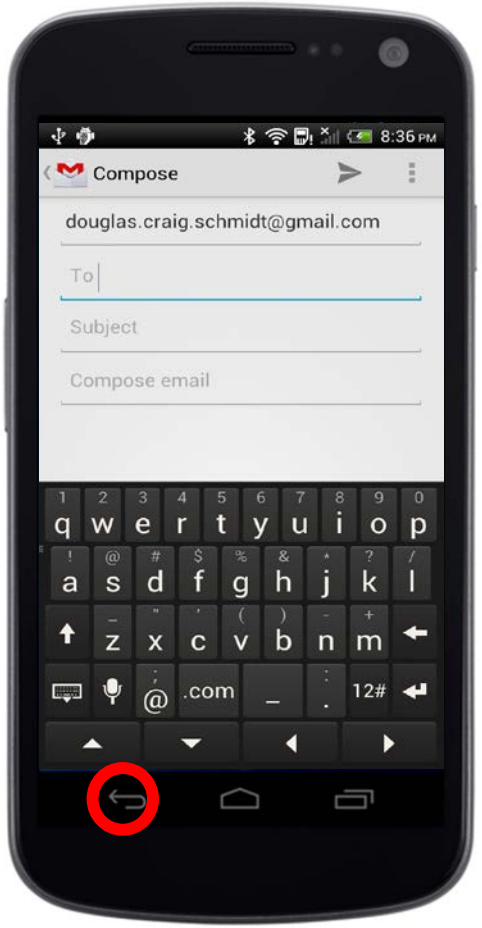
- The task's Activity objects are stored on a "back stack" with the currently running Activity at the top
- At runtime
  - Launching an Activity places it on top of the stack
  - Finishing an Activity pops it off the stack...
  - ... & returns to the previous Activity



developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html

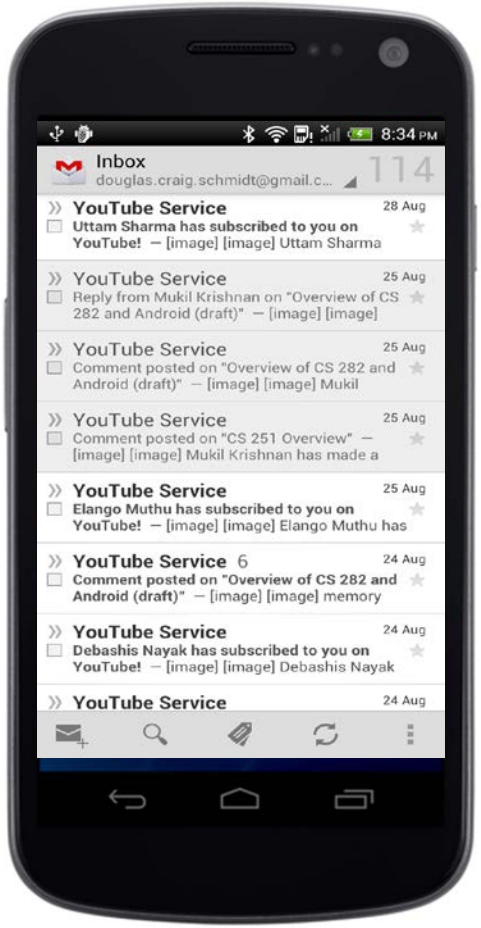
Tasks

- The task's Activity objects are stored on a "back stack" with the currently running Activity at the top
- At runtime
  - Launching an Activity places it on top of the stack
  - Hitting the BACK button pops current activity off the stack...



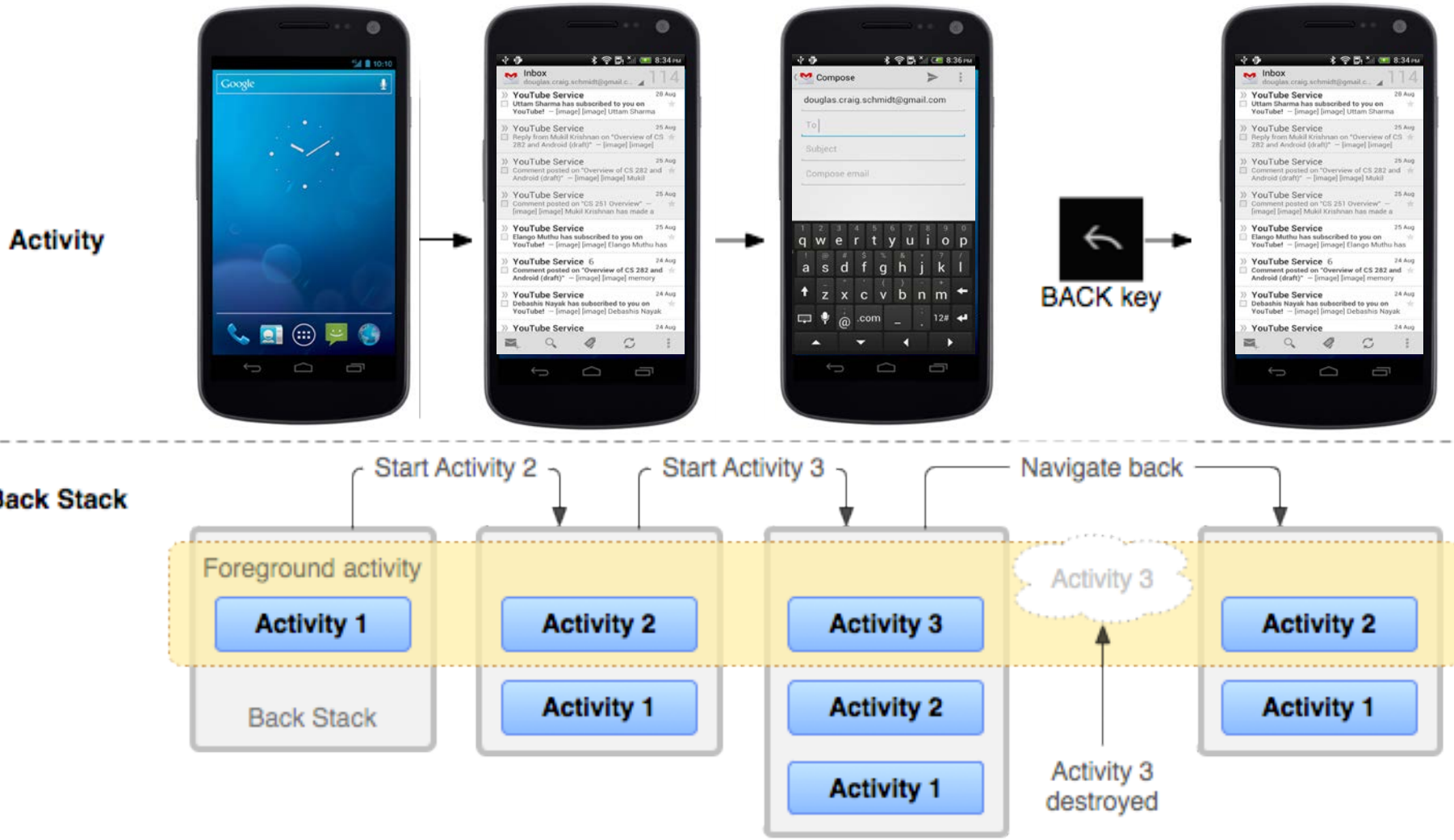
Tasks

- The task's Activity objects are stored on a "back stack" with the currently running Activity at the top
- At runtime
  - Launching an Activity places it on top of the stack
  - Hitting the BACK button pops current activity off the stack...
  - ... & returns to the previous Activity



developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html

Task Stack



developer.android.com/guide/topics/fundamentals/tasks-and-back-stack.html

Implementing an Activity

- Implementing an Activity involves several steps, e.g.:
  - Inherit from Activity class

```
public class MapLocation
             extends Activity {

   ...
}
```

Implementing an Activity

- Implementing an Activity involves several steps, e.g.:
  - Inherit from Activity class
  - Override selected lifecycle hook methods

```
public class MapLocation
             extends Activity {
   protected void onCreate
     (Bundle savedInstanceState);
   protected void onStart();
   protected void onRestart();
   protected void onResume();
   protected void onPause();
   protected void onStop();
   protected void onDestroy();
   ...
}
```

Implementing an Activity

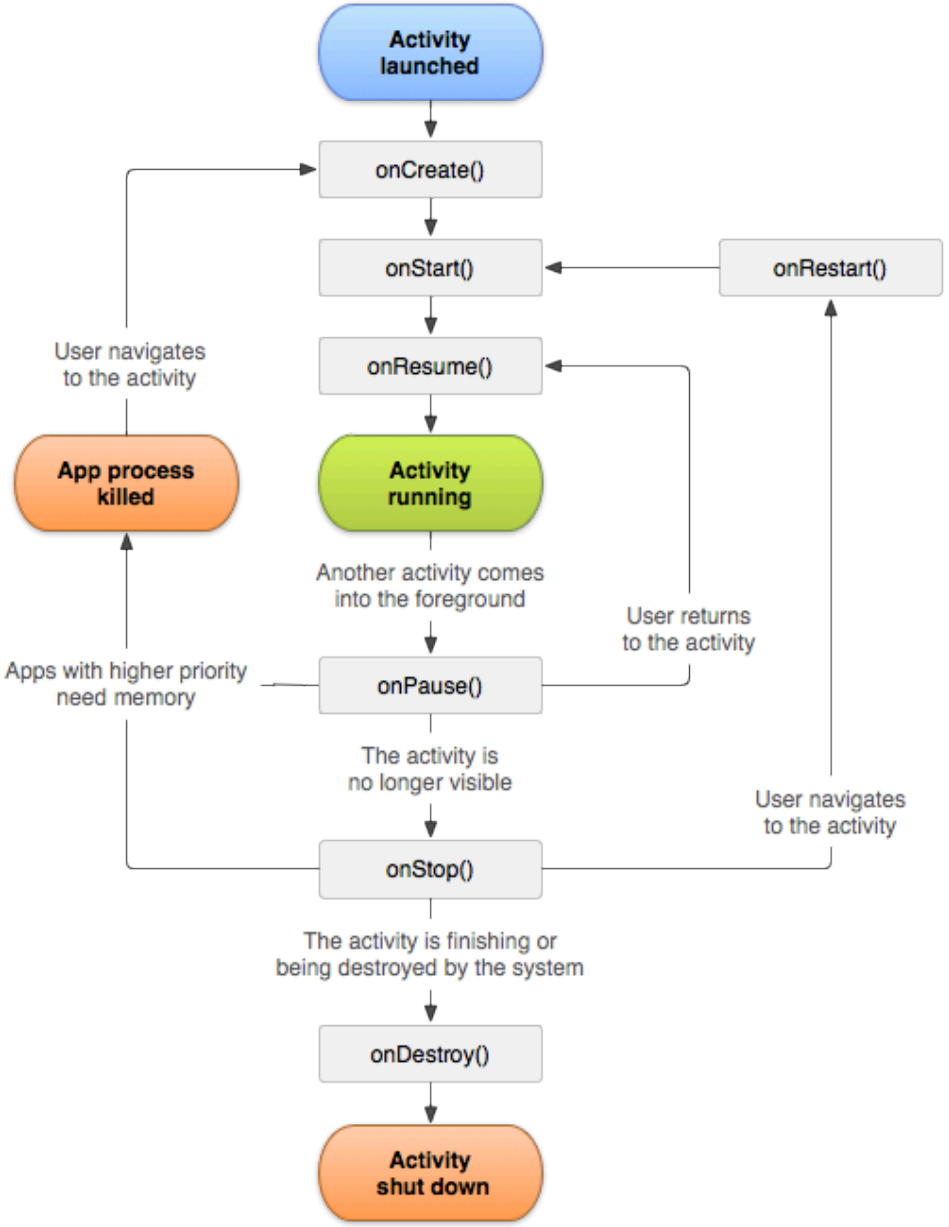
- Implementing an Activity involves several steps, e.g.:
  - Inherit from Activity class
  - Override selected lifecycle hook methods
  - Include Activity in the config file AndroidManifest.xml
  - etc.

```
public class MapLocation
             extends Activity {
   protected void onCreate
     (Bundle savedInstanceState);
   protected void onStart();
   protected void onRestart();
   protected void onResume();
   protected void onPause();
   protected void onStop();
   protected void onDestroy();
   ...
}
```

```
<activity
 android:name="course.examples.Activity.SimpleMapExample.MapLocation"
 android:label="Map A Location">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name=
                                "android.intent.category.LAUNCHER" />
            </intent-filter>
</activity>
```

Implementing an Activity

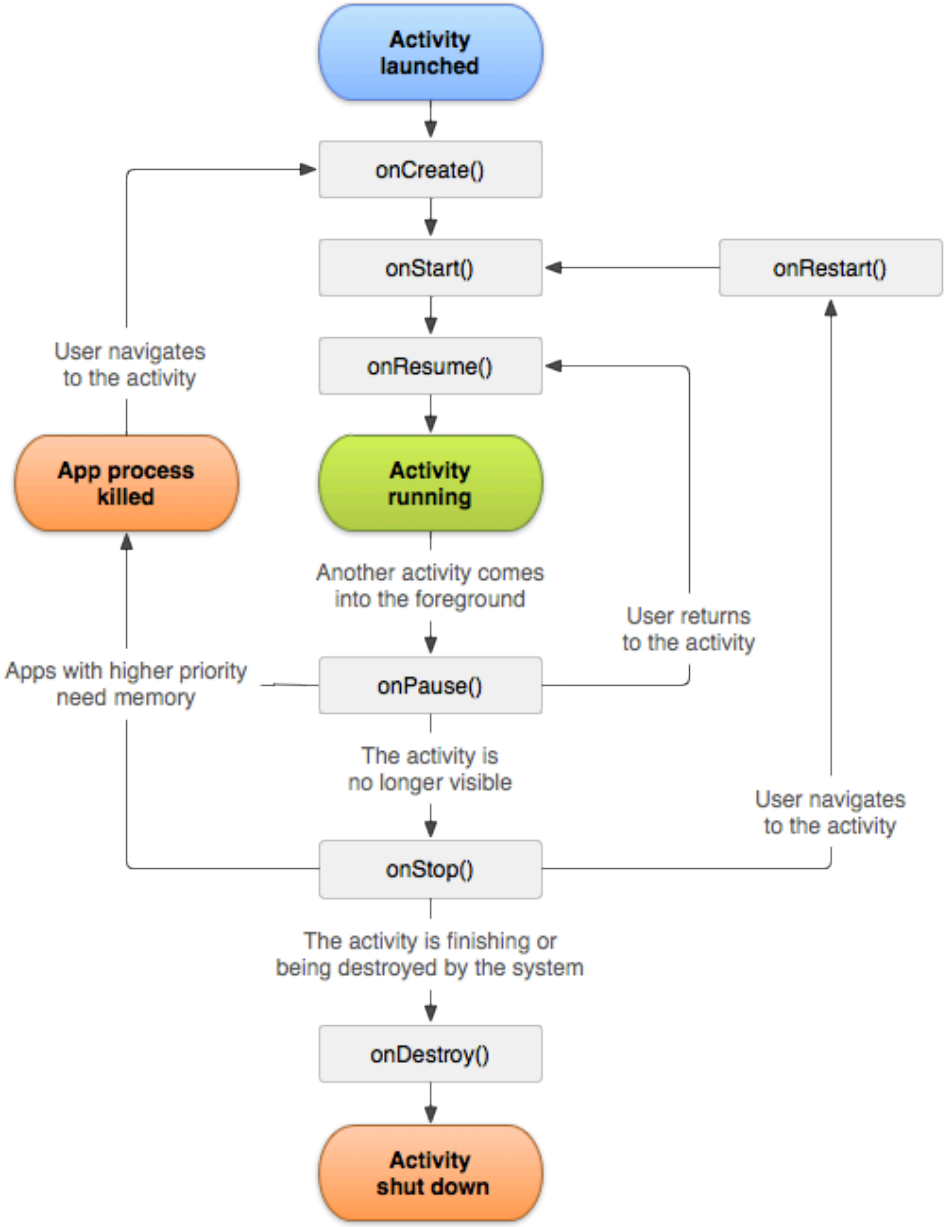
- Implementing an Activity involves several steps
- Android communicates state changes to an Activity by calling its lifecycle hook methods



Implementing an Activity

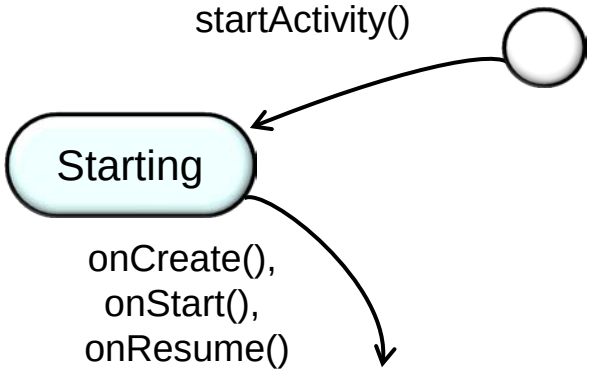
- Implementing an Activity involves several steps
- Android communicates state changes to an Activity by calling its lifecycle hook methods

- **Commonality**: Provides common interface for interacting with user, including operations performed when moving between lifecycle states
- **Variability**: Subclasses can override lifecycle hook methods to do necessary work when an Activity changes state



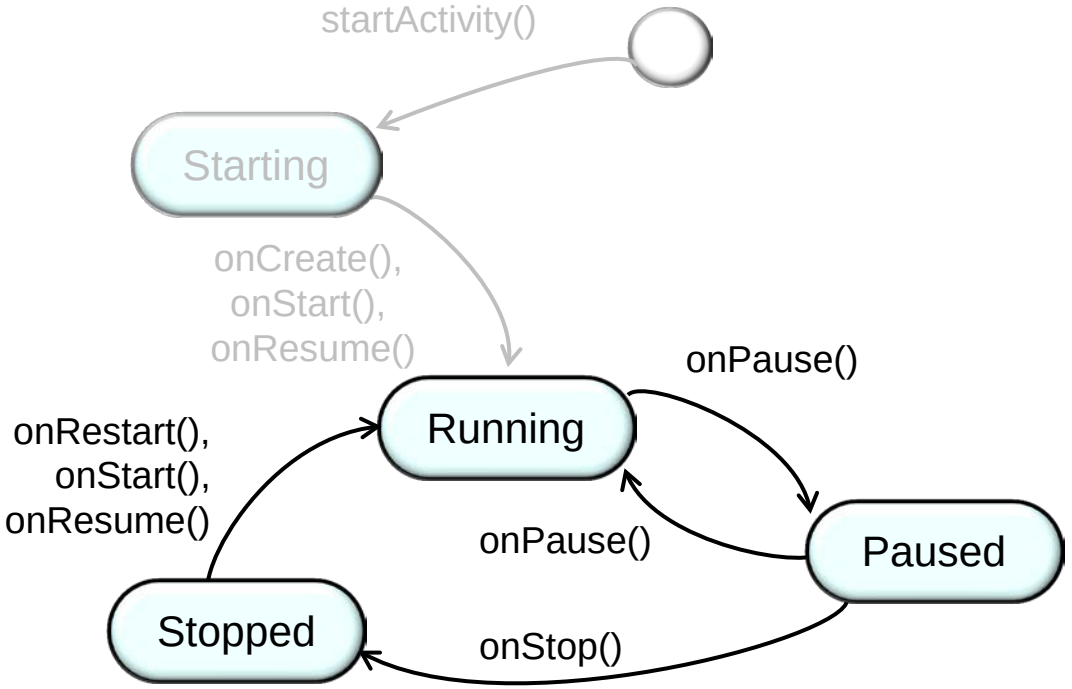
Activity Lifecycle States

- **Activity starting** – Initialization steps



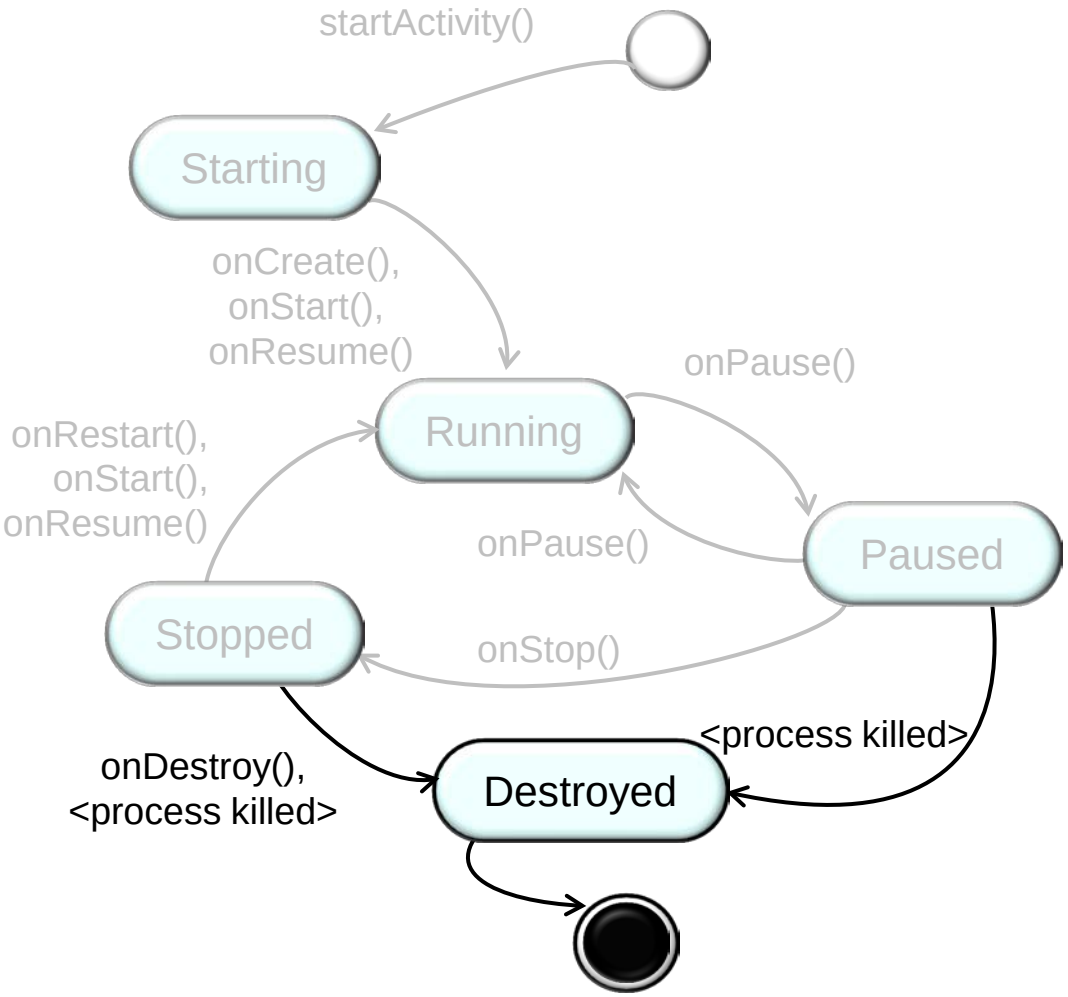
Activity Lifecycle States

- **Activity starting** – Initialization steps
- **Activity running**
  - *Running* – visible, has focus
  - *Paused* – visible, does not have focus, can be terminated
  - *Stopped* – not visible, does not have focus, can be terminated



Activity Lifecycle States

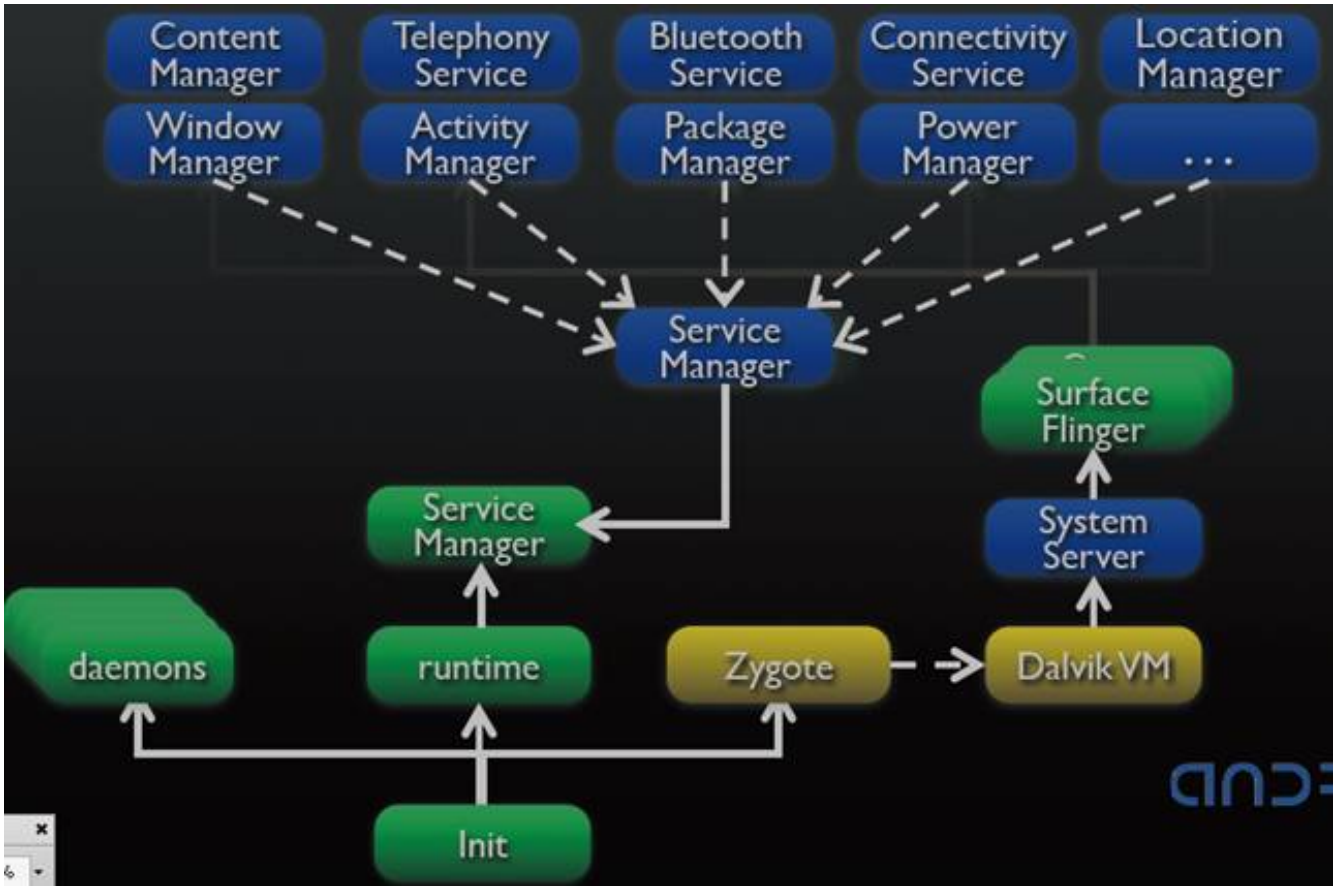
- **Activity starting** – Initialization steps
- **Activity running**
  - *Running* – visible, has focus
  - *Paused* – visible, does not have focus, can be terminated
  - *Stopped* – not visible, does not have focus, can be terminated
- **Activity shut down** – Voluntarily finished or involuntarily killed by the system



See developer.android.com/guide/components/activities.html for more info

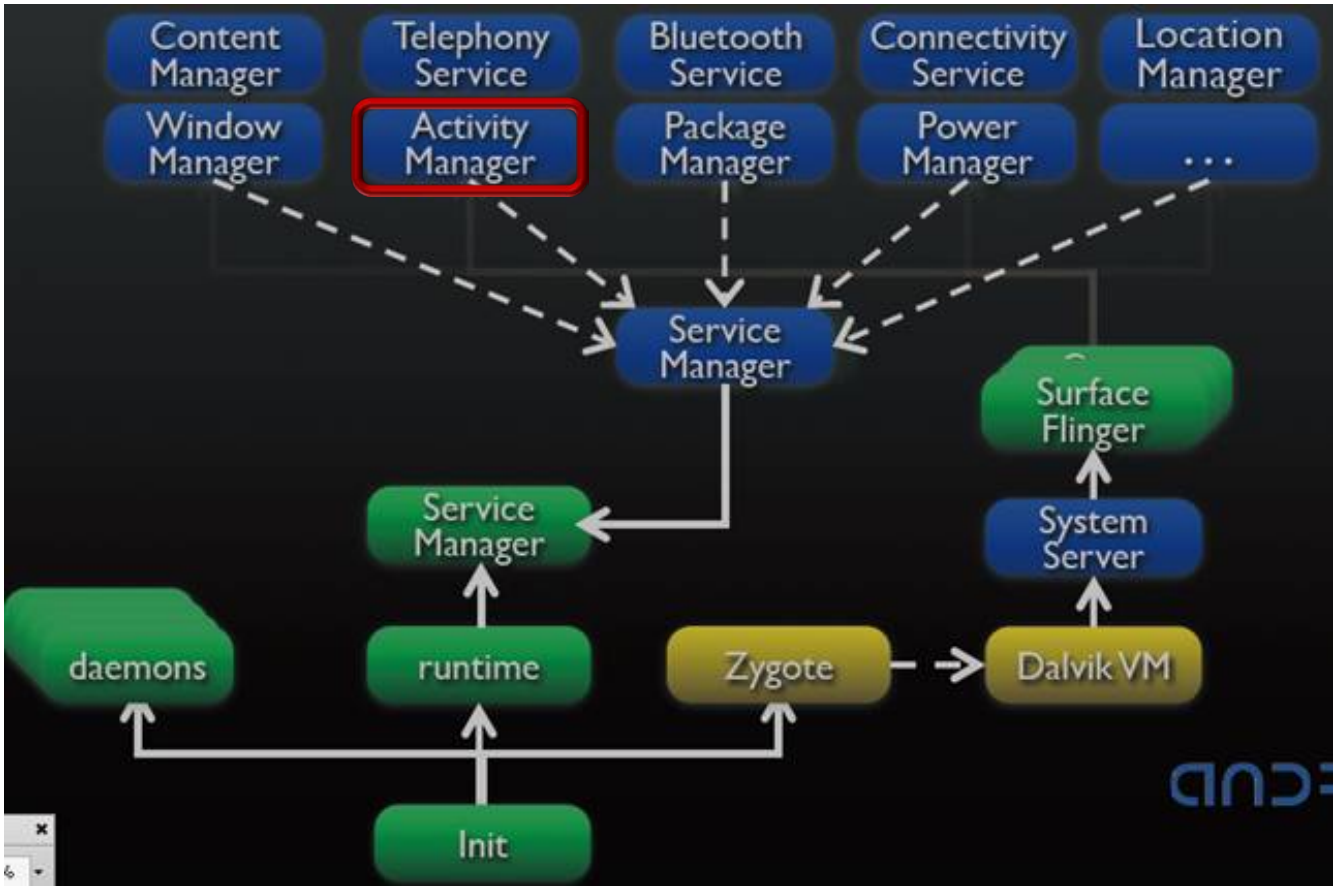
Managing the Activity Lifecycle

- Android communicates state changes to application by calling specific lifecycle methods



Managing the Activity Lifecycle

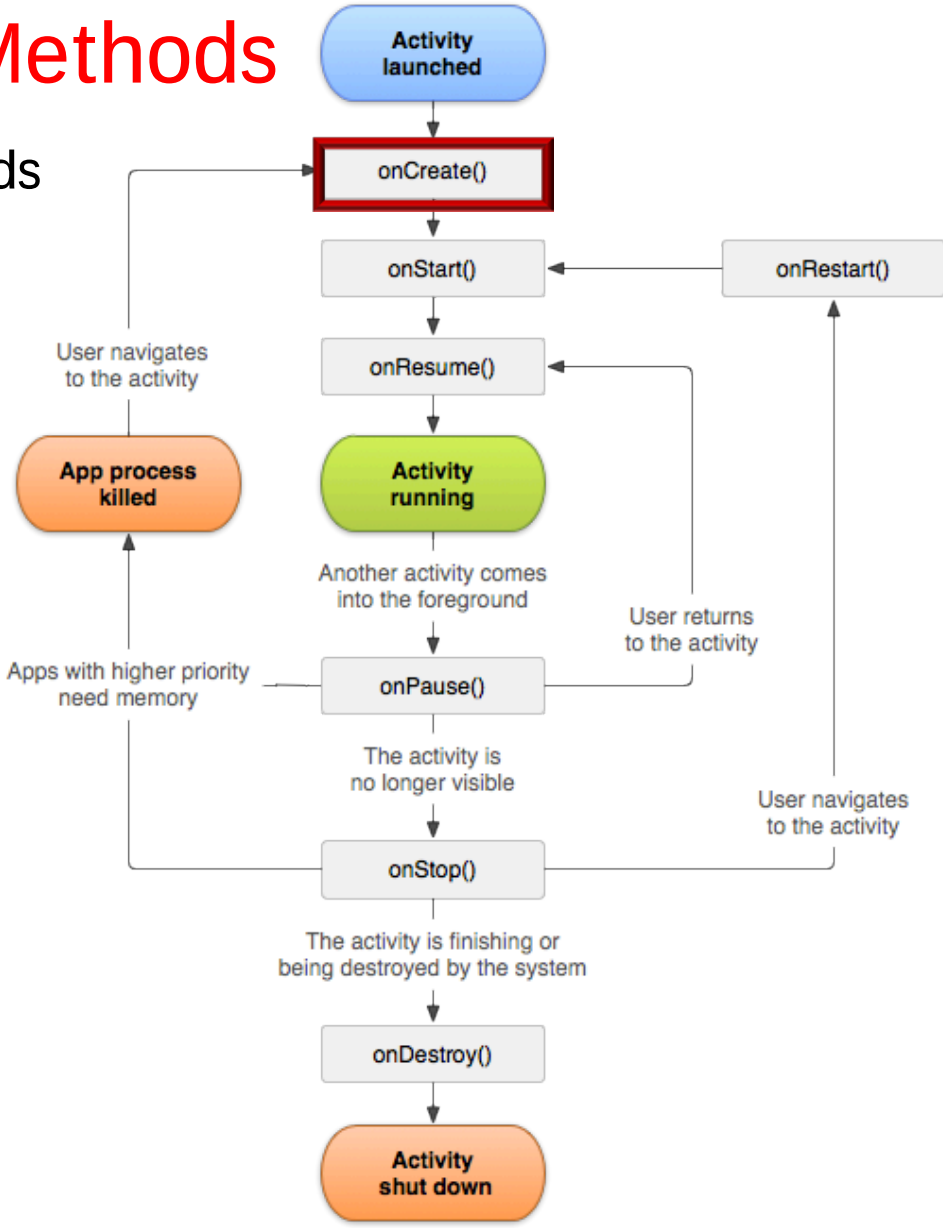
- Android communicates state changes to application by calling specific lifecycle methods
- The ActivityManager is the system service in Android that communicates these changes



developer.android.com/reference/android/app/ActivityManager.html has more

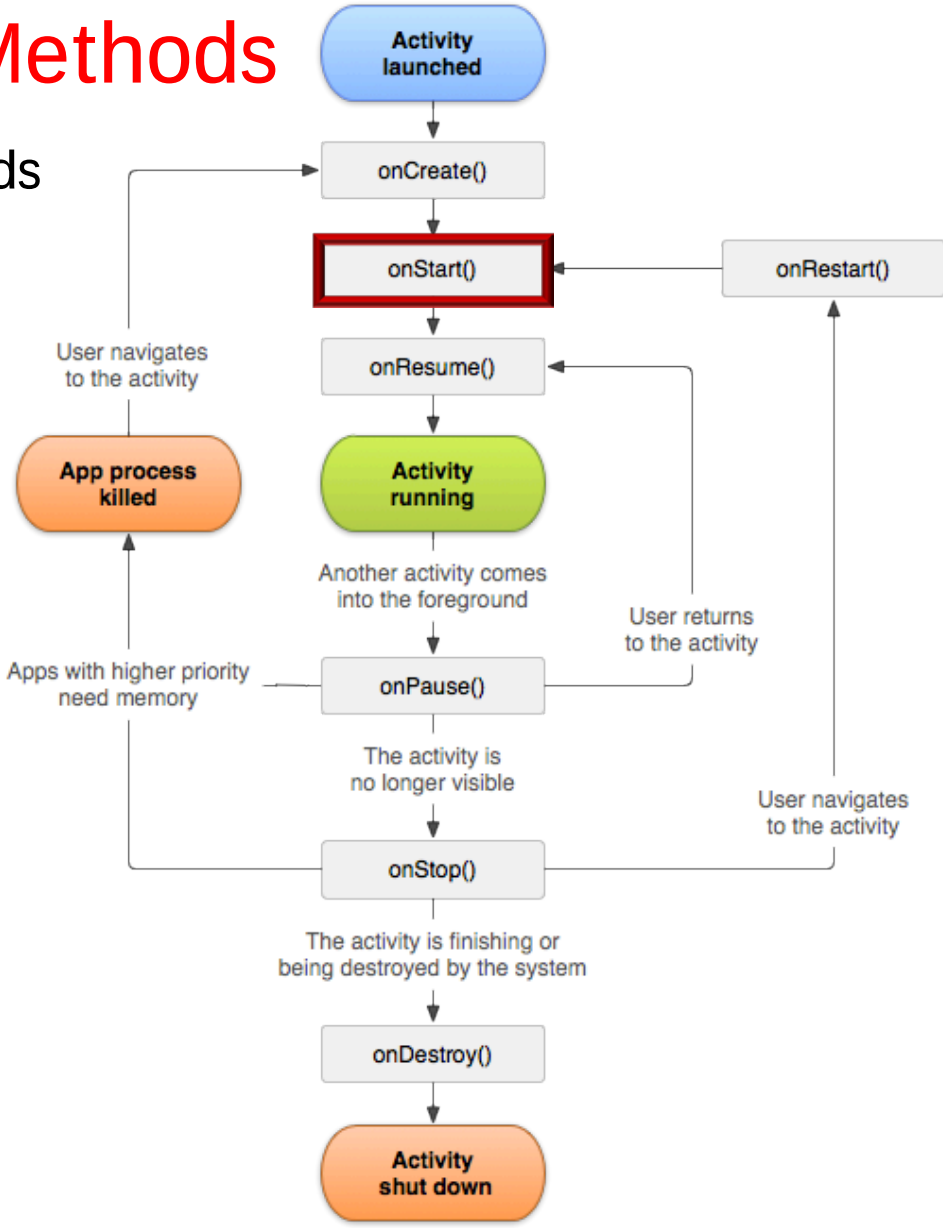
Activity Lifecycle Hook Methods

- The Android runtime calls hook methods on an Activity to control its lifecycle:
  - **onCreate()** – called to initialize an Activity when it is first created



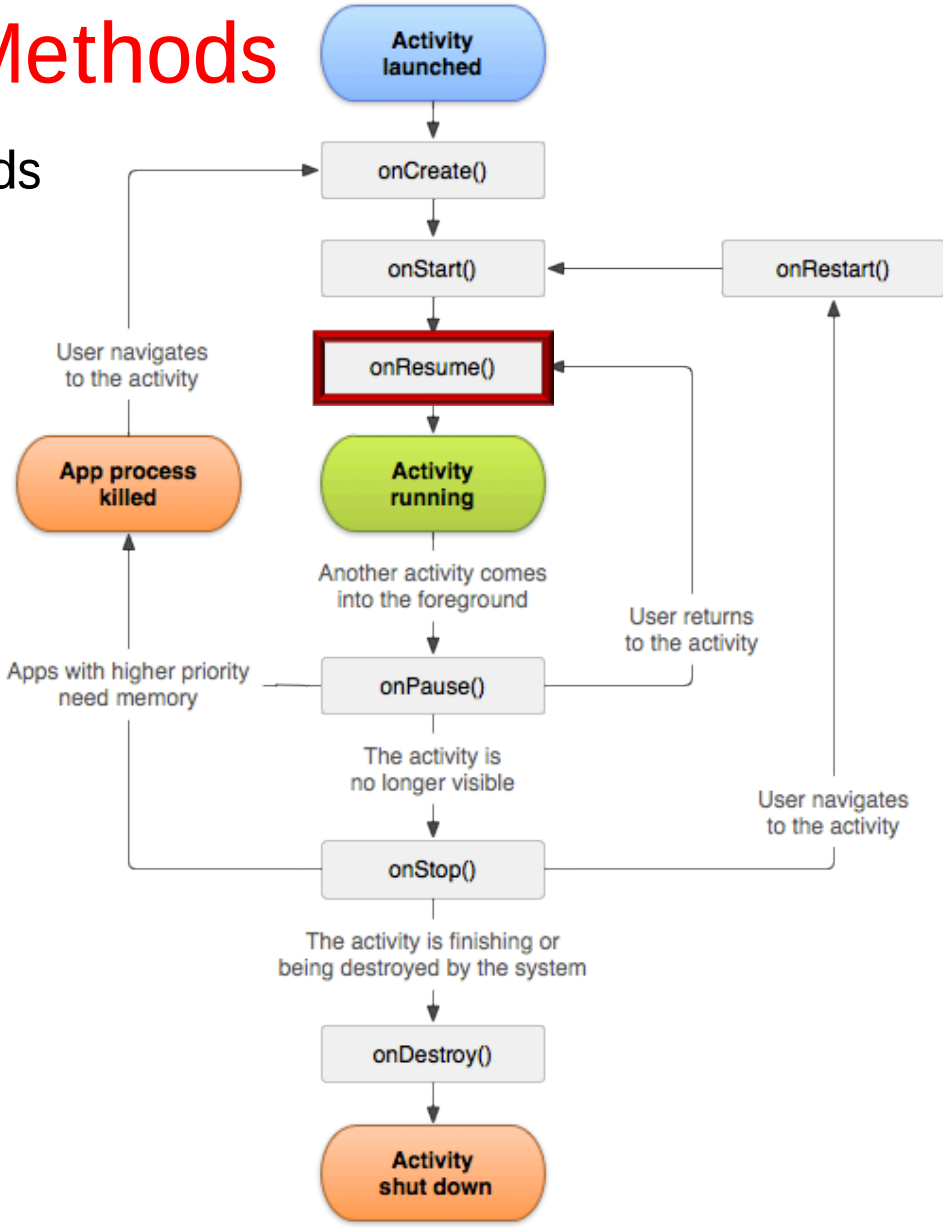
Activity Lifecycle Hook Methods

- The Android runtime calls hook methods on an Activity to control its lifecycle:
  - **onCreate()** – called to initialize an Activity when it is first created
  - **onStart()** – called when Activity is becoming visible to the user



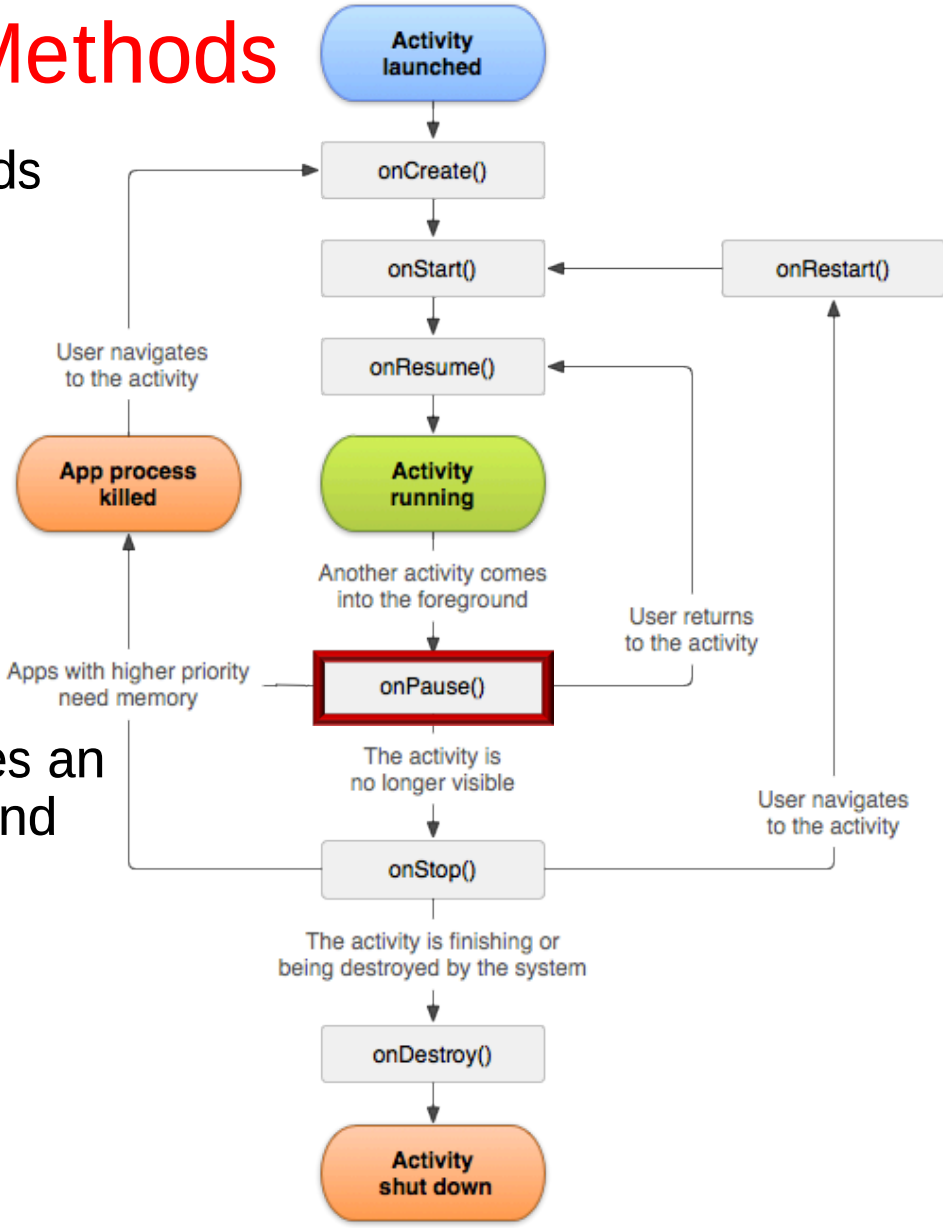
Activity Lifecycle Hook Methods

- The Android runtime calls hook methods on an Activity to control its lifecycle:
  - **onCreate()** – called to initialize an Activity when it is first created
  - **onStart()** – called when Activity is becoming visible to the user
  - **onResume()** – called when user returns to an Activity from another



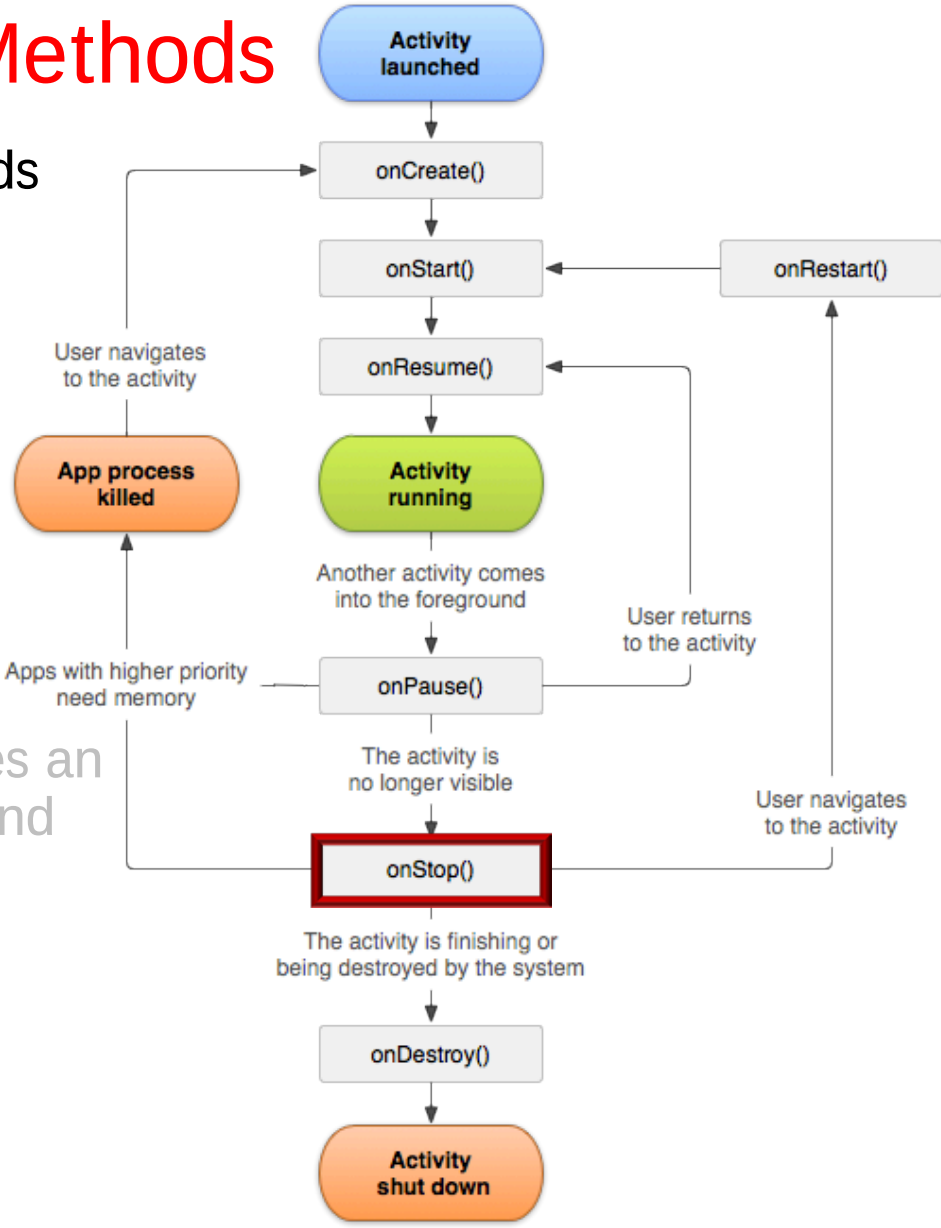
Activity Lifecycle Hook Methods

- The Android runtime calls hook methods on an Activity to control its lifecycle:
  - **onCreate()** – called to initialize an Activity when it is first created
  - **onStart()** – called when Activity is becoming visible to the user
  - **onResume()** – called when user returns to an Activity from another
  - **onPause()** – called when user leaves an Activity that's still visible in background



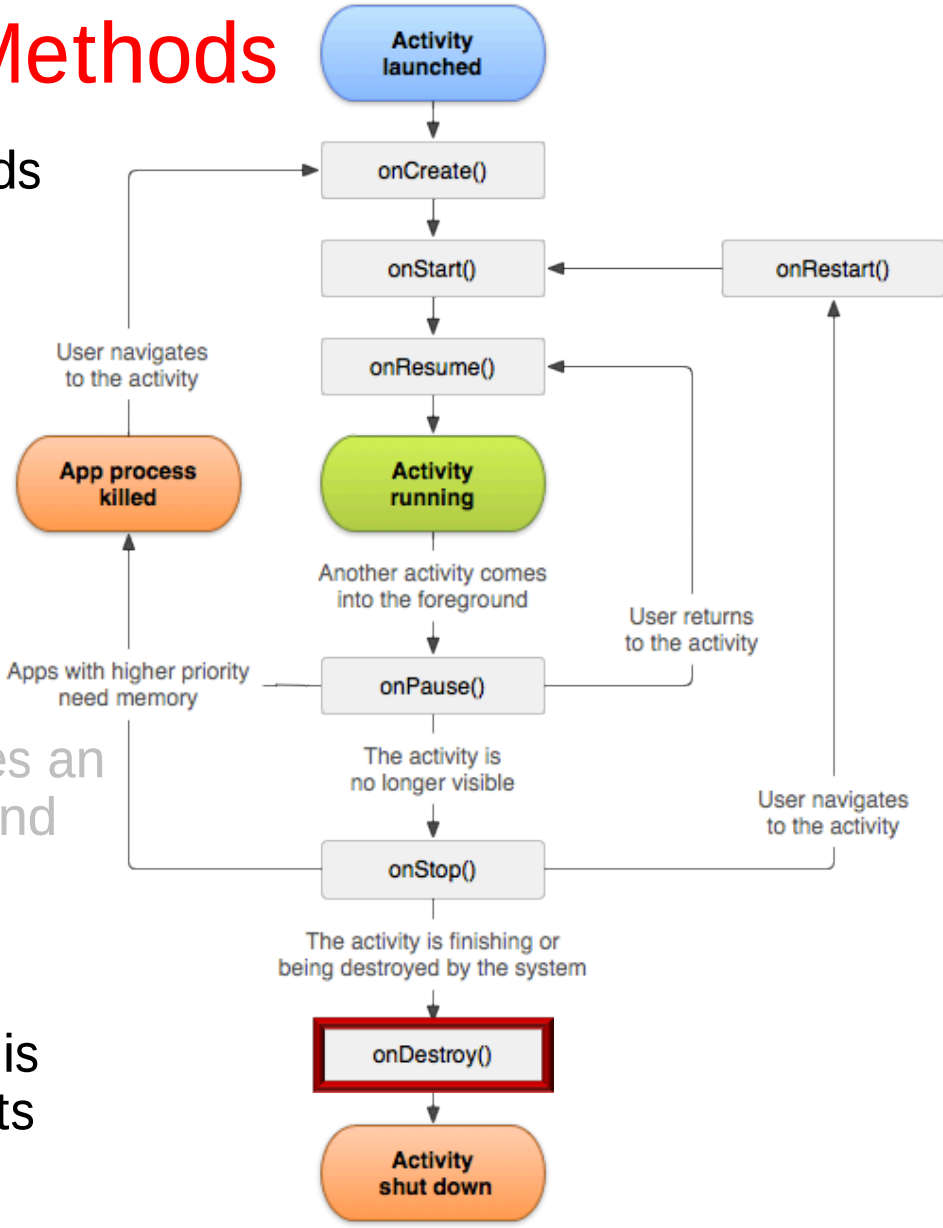
Activity Lifecycle Hook Methods

- The Android runtime calls hook methods on an Activity to control its lifecycle:
  - **onCreate()** – called to initialize an Activity when it is first created
  - **onStart()** – called when Activity is becoming visible to the user
  - **onResume()** – called when user returns to an Activity from another
  - **onPause()** – called when user leaves an Activity that's still visible in background
  - **onStop()** – called when user leaves an Activity for another



Activity Lifecycle Hook Methods

- The Android runtime calls hook methods on an Activity to control its lifecycle:
  - **onCreate()** – called to initialize an Activity when it is first created
  - **onStart()** – called when Activity is becoming visible to the user
  - **onResume()** – called when user returns to an Activity from another
  - **onPause()** – called when user leaves an Activity that's still visible in background
  - **onStop()** – called when user leaves an Activity for another
  - **onDestroy()** – called when Activity is being released & needs to clean up its allocated resources



See developer.android.com/reference/android/app/Activity.html for more info

Useful Helper Class for Activity Lifecycle Methods

```
public abstract class LifecycleLoggingActivity extends Activity {

  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Log.d(getClass().getSimpleName(),
                "onCreate()");
    if (savedInstanceState == null)
      Log.d(getClass().getSimpleName(), "activity created anew");
    else
      Log.d(getClass().getSimpleName(), "activity restarted");
  }

  public void onStart() {
    super.onStart();
    Log.d(getClass().getSimpleName(), "onStart()");
  }
  ...
```

Inherit from Activity class

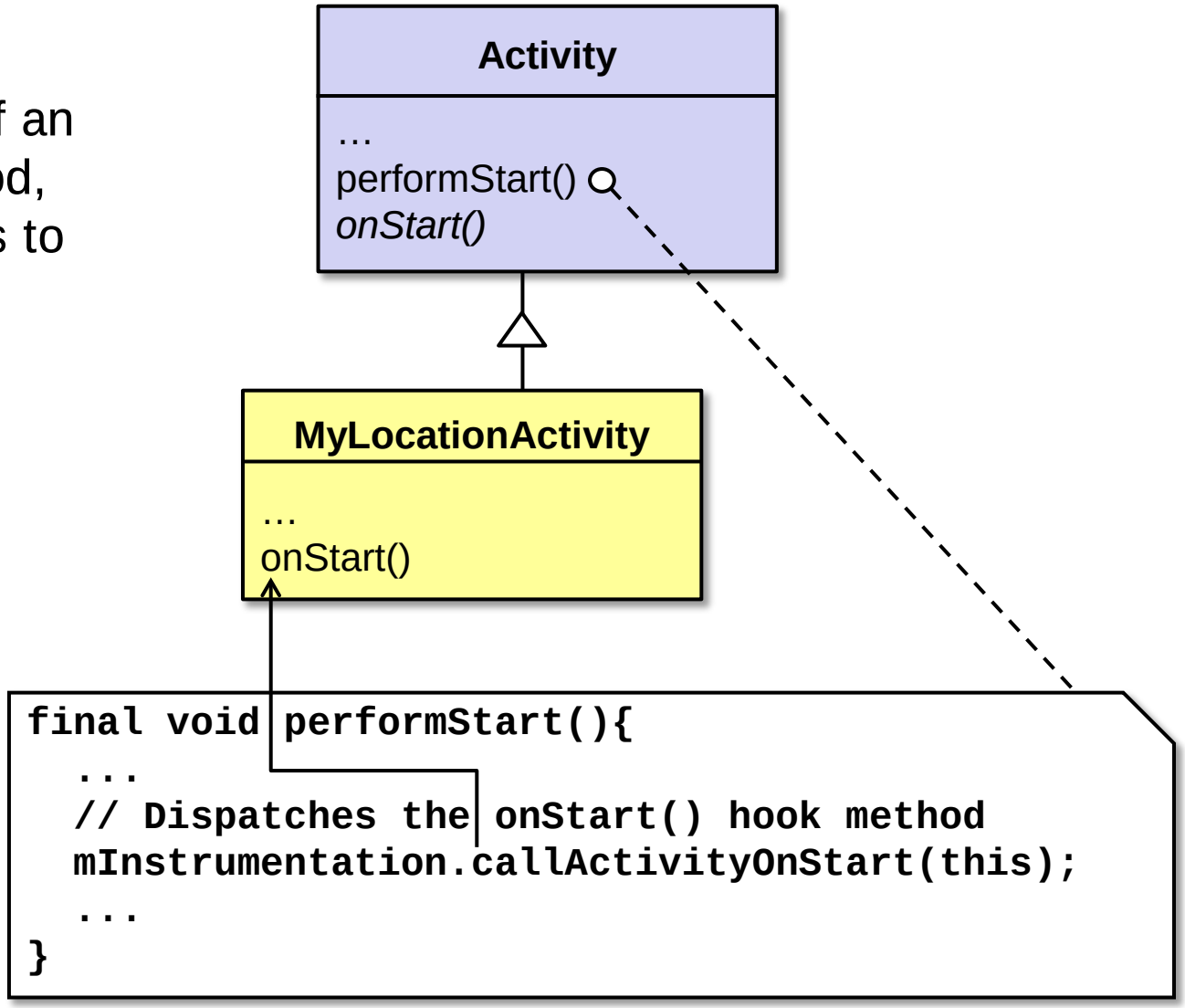
Automatically log lifecycle hook method calls

Note “inversion of control” & patterns in the Android Activity framework

Template Method GoF Class Behavioral

Intent

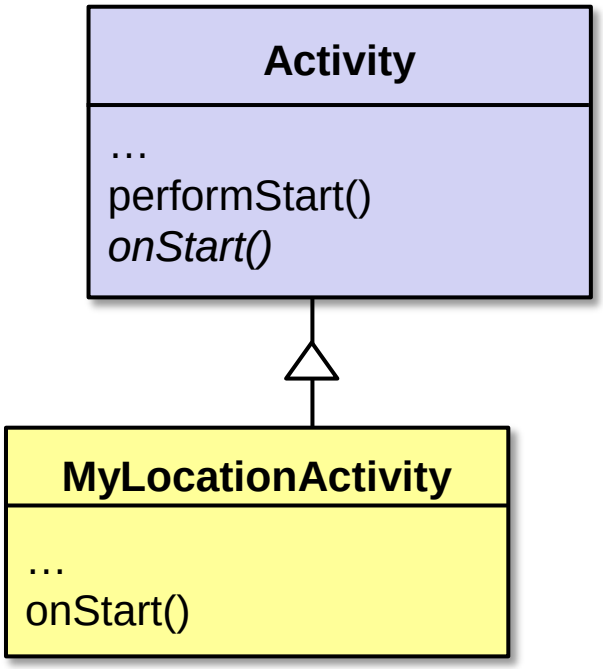
- Provide a skeleton of an algorithm in a method, deferring some steps to subclasses



Template Method GoF Class Behavioral

Applicability

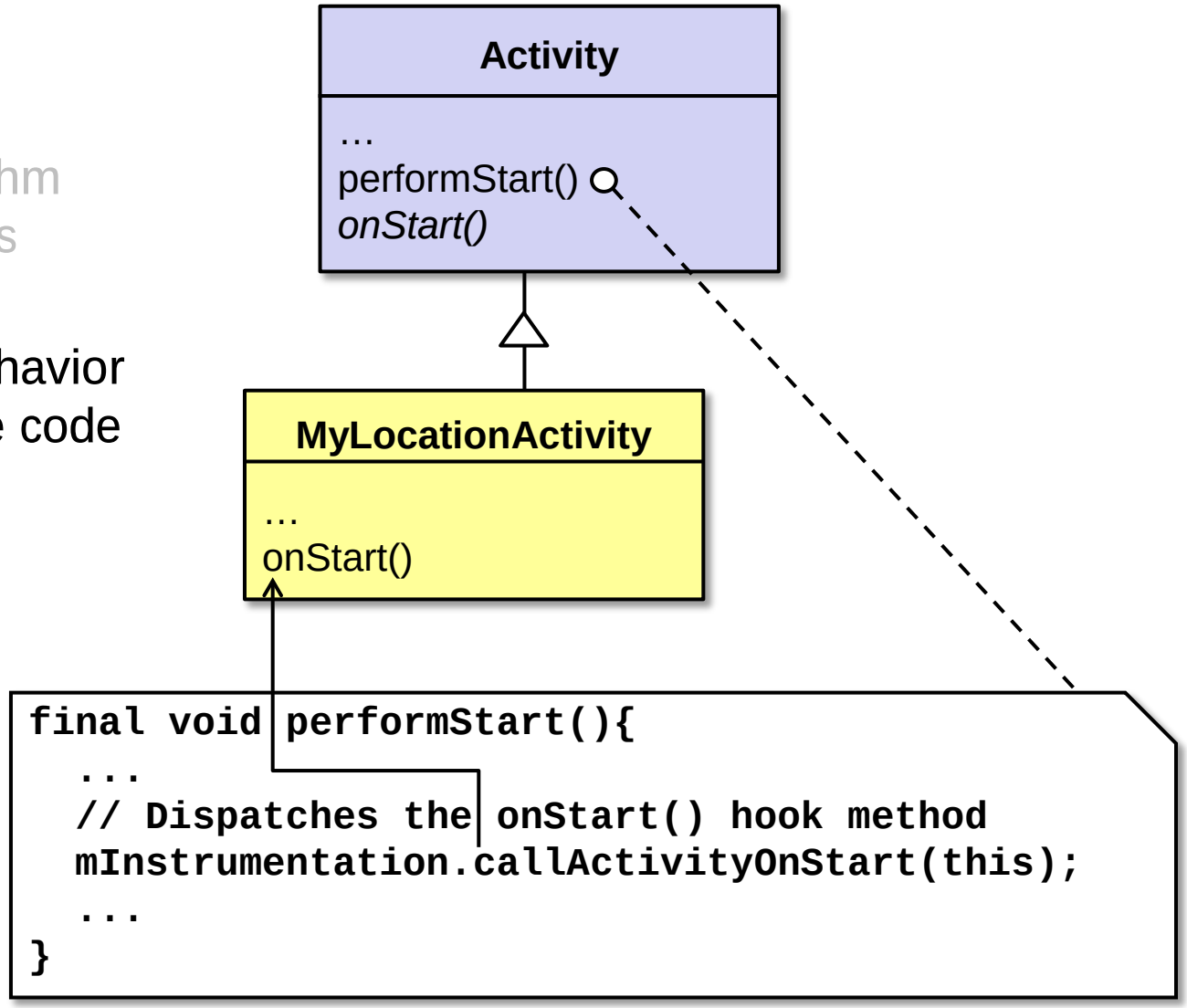
- Implement invariant aspects of an algorithm *once* & let subclasses define variant parts



Template Method GoF Class Behavioral

Applicability

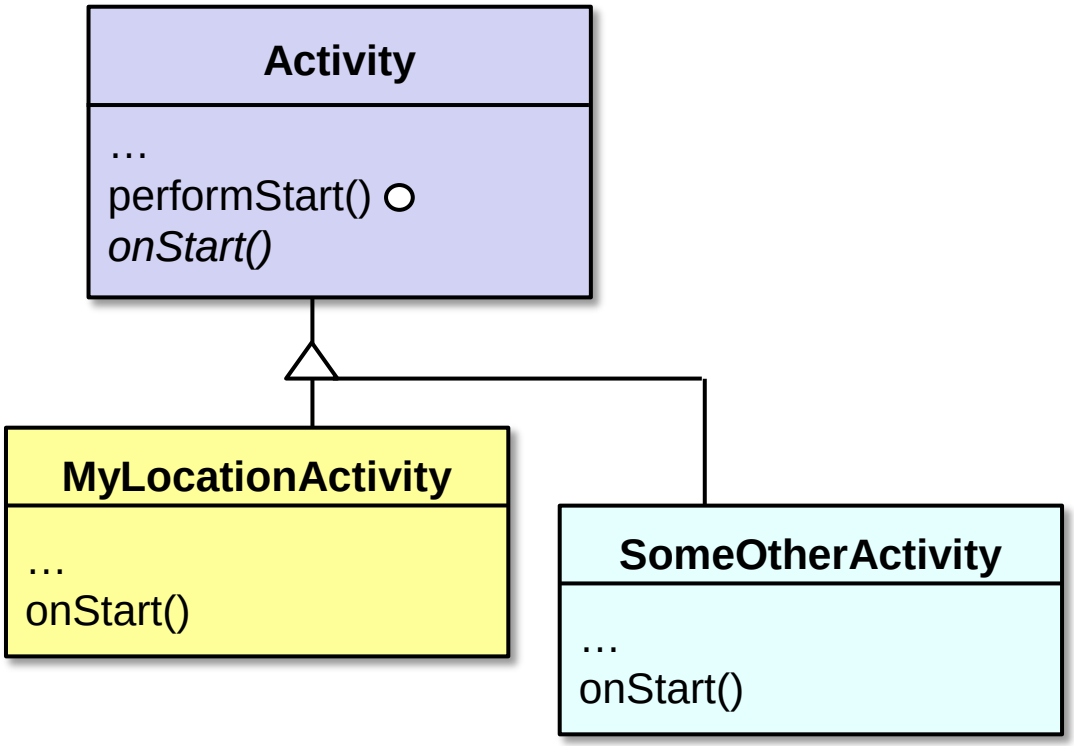
- Implement invariant aspects of an algorithm *once* & let subclasses define variant parts
- Localize common behavior in a class to increase code reuse



Template Method GoF Class Behavioral

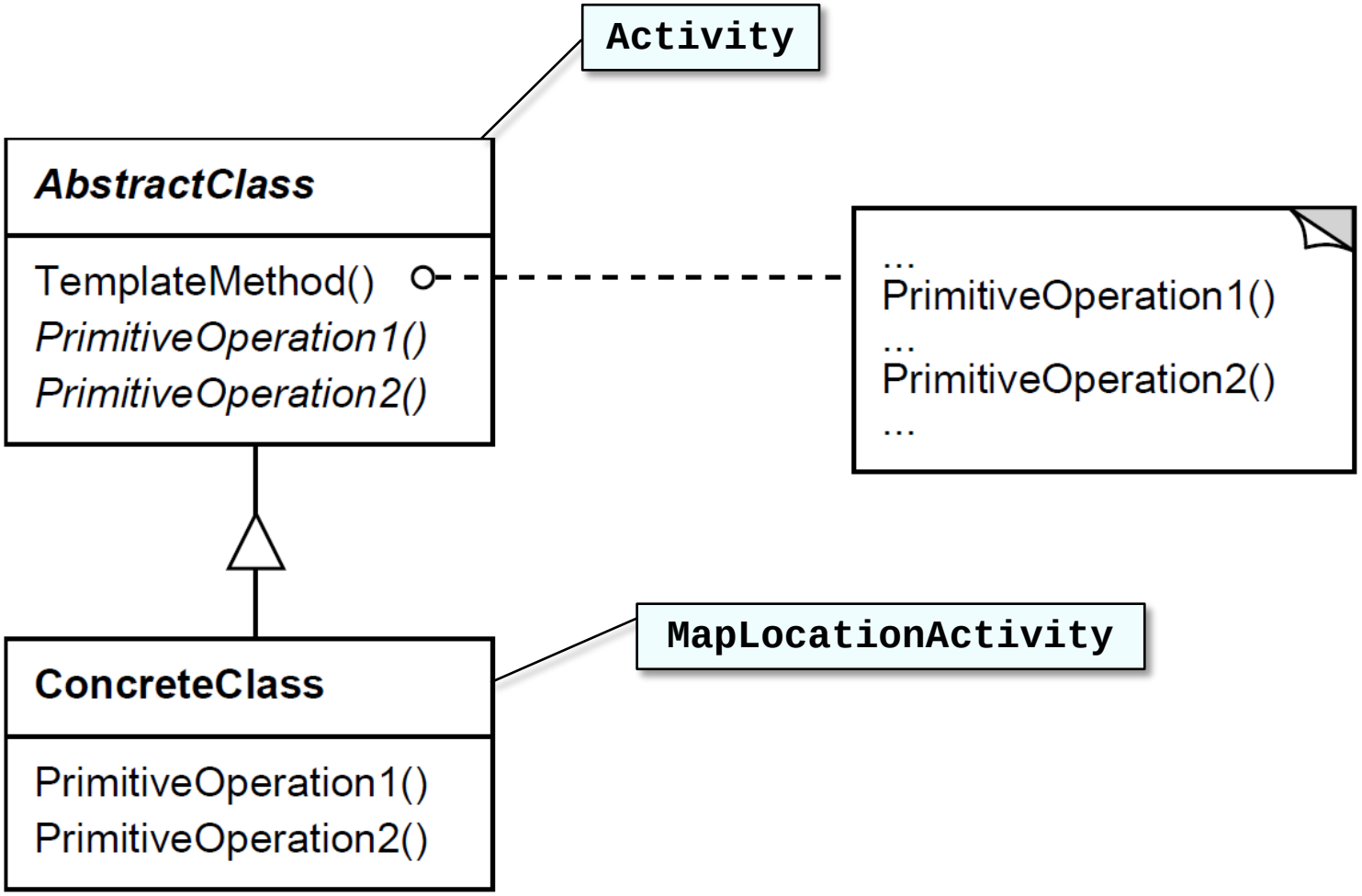
Applicability

- Implement invariant aspects of an algorithm *once* & let subclasses define variant parts
- Localize common behavior in a class to increase code reuse
- Control subclass extensions



Template Method GoF Class Behavioral

Structure & Participants



Template Method GoF Class Behavioral

Template Method example in Android

- Allow subclasses to customize certain steps in the input handling algorithm

```
public class Activity extends ContextThemeWrapper {
  ...
  final void performStart() {
    ...
    mInstrumentation.callActivityOnStart(this);
    ...
```

Template method

Call a helper method that dispatches the hook method

```
public class Instrumentation {
  public void callActivityOnStart(Activity activity) {
        activity.onStart();
  }
```

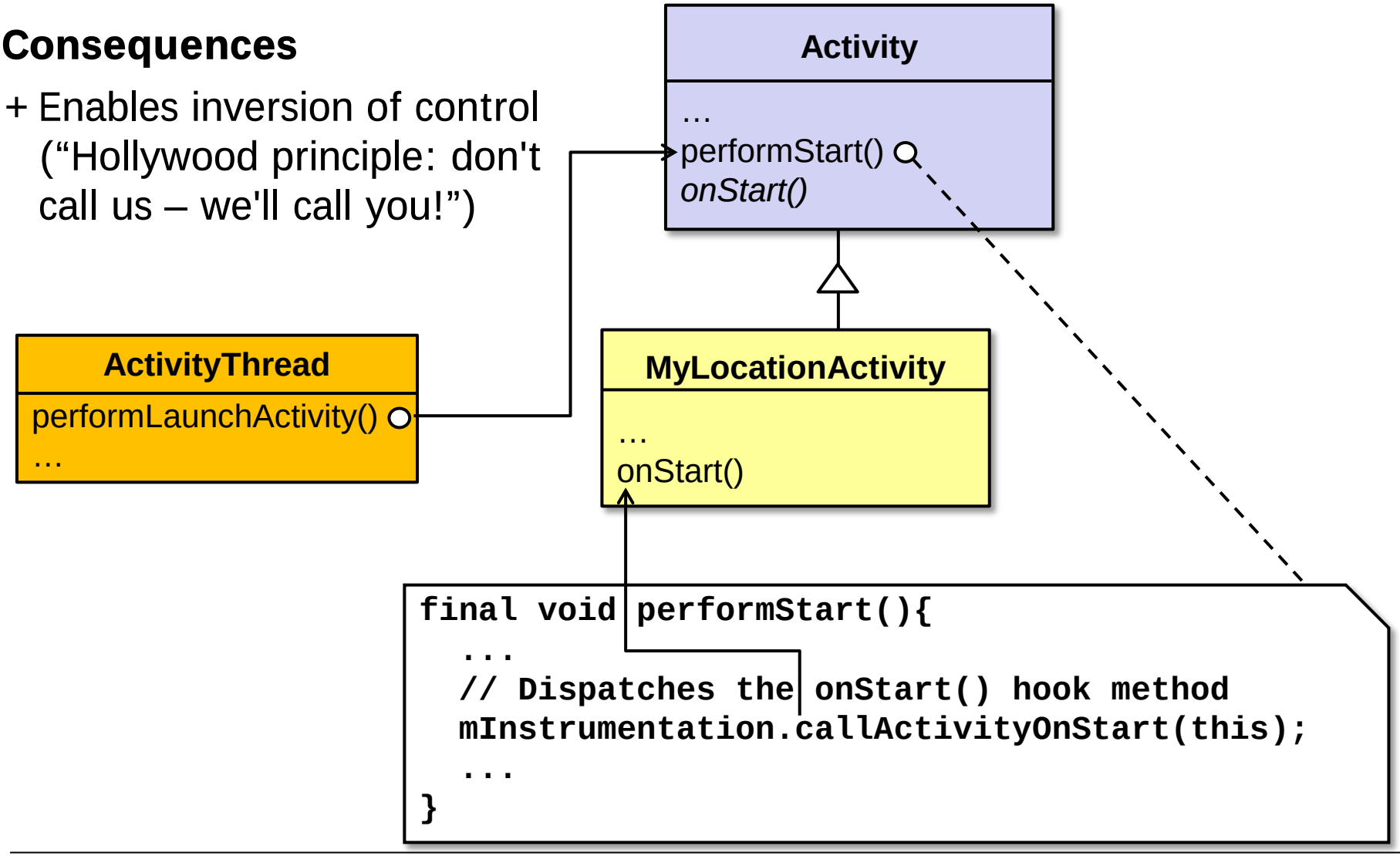
Dispatch the Activity subclasses' onStart() hook method

See frameworks/base/core/java/android/app for this code

Template Method GoF Class Behavioral

Consequences

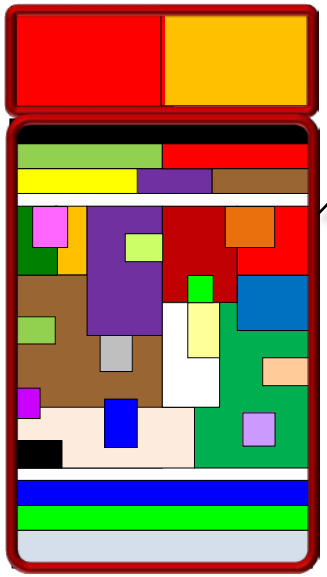
+ Enables inversion of control ("Hollywood principle: don't call us – we'll call you!")

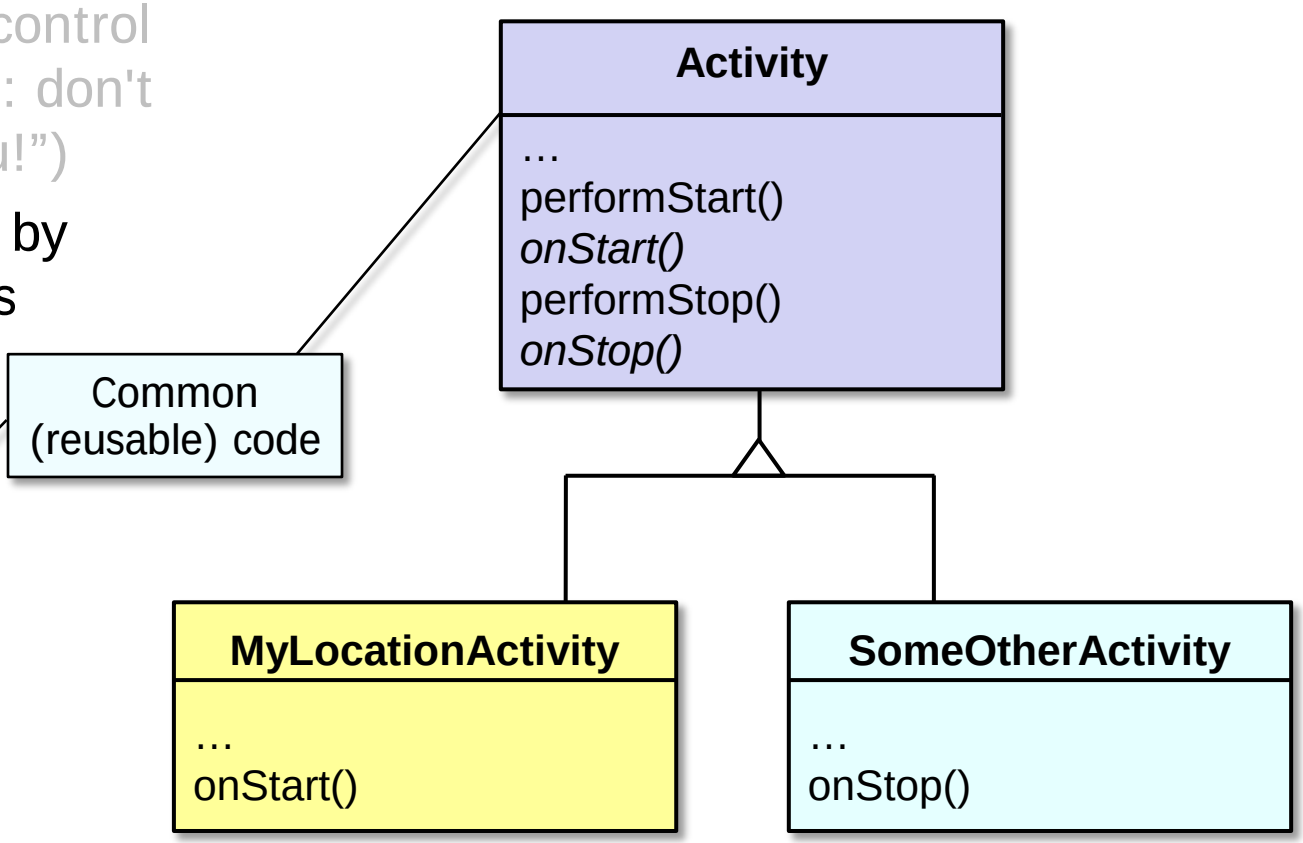


Template Method GoF Class Behavioral

Consequences

+ Enables inversion of control ("Hollywood principle: don't call us – we'll call you!")
+ Promotes code reuse by collapsing stove-pipes



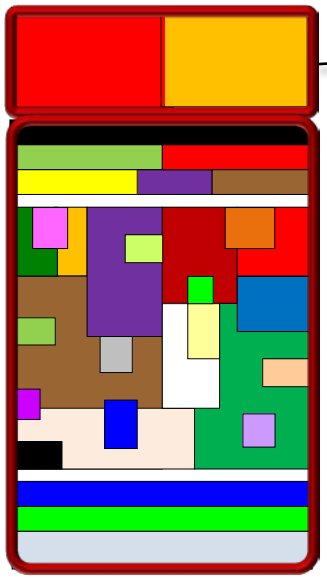


Template Method GoF Class Behavioral

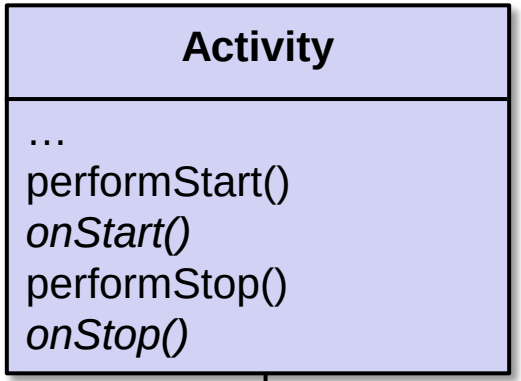
Consequences

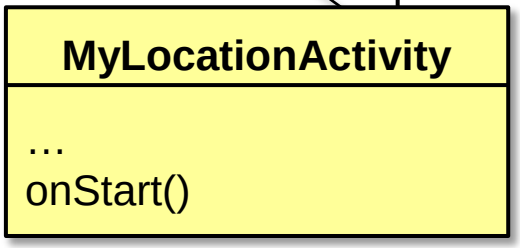
+ Enables inversion of control ("Hollywood principle: don't call us – we'll call you!")

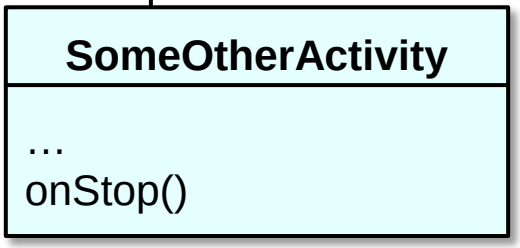
+ Promotes code reuse by collapsing stove-pipes



Variant (non-reusable) code



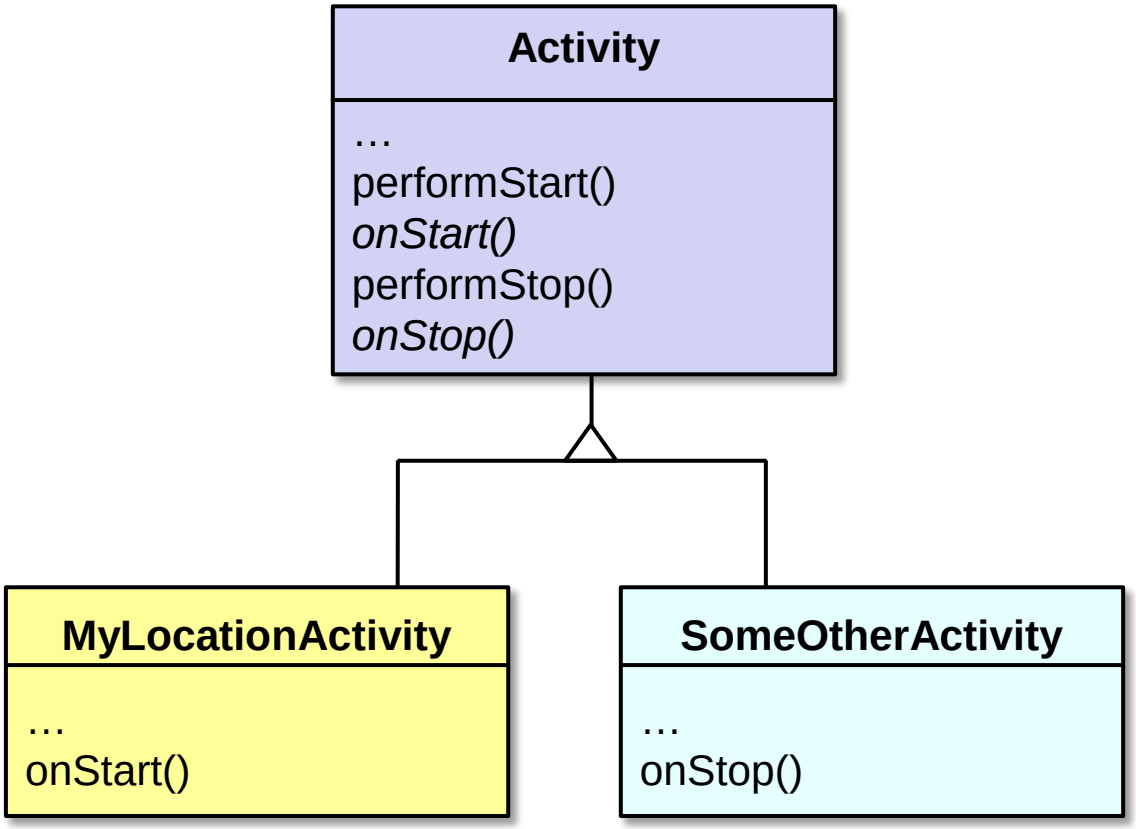




Template Method GoF Class Behavioral

Consequences

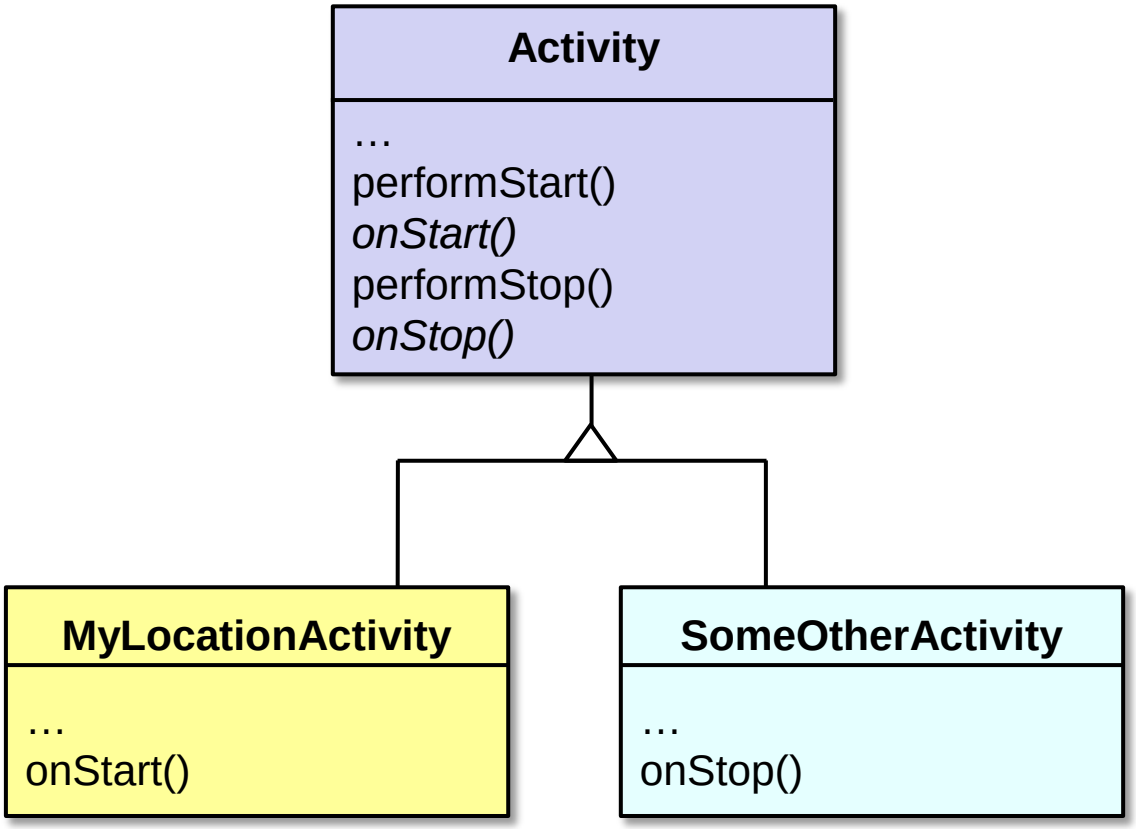
+ Enables inversion of control ("Hollywood principle: don't call us – we'll call you!")
+ Promotes code reuse by collapsing stove-pipes
+ Programmers enforce overriding rules via subclassing



Template Method GoF Class Behavioral

Consequences

– Must subclass to specialize behavior, which can lead to an explosion of subclasses
  - Compare & contrast with the *Strategy* pattern



Template Method GoF Class Behavioral

Consequences

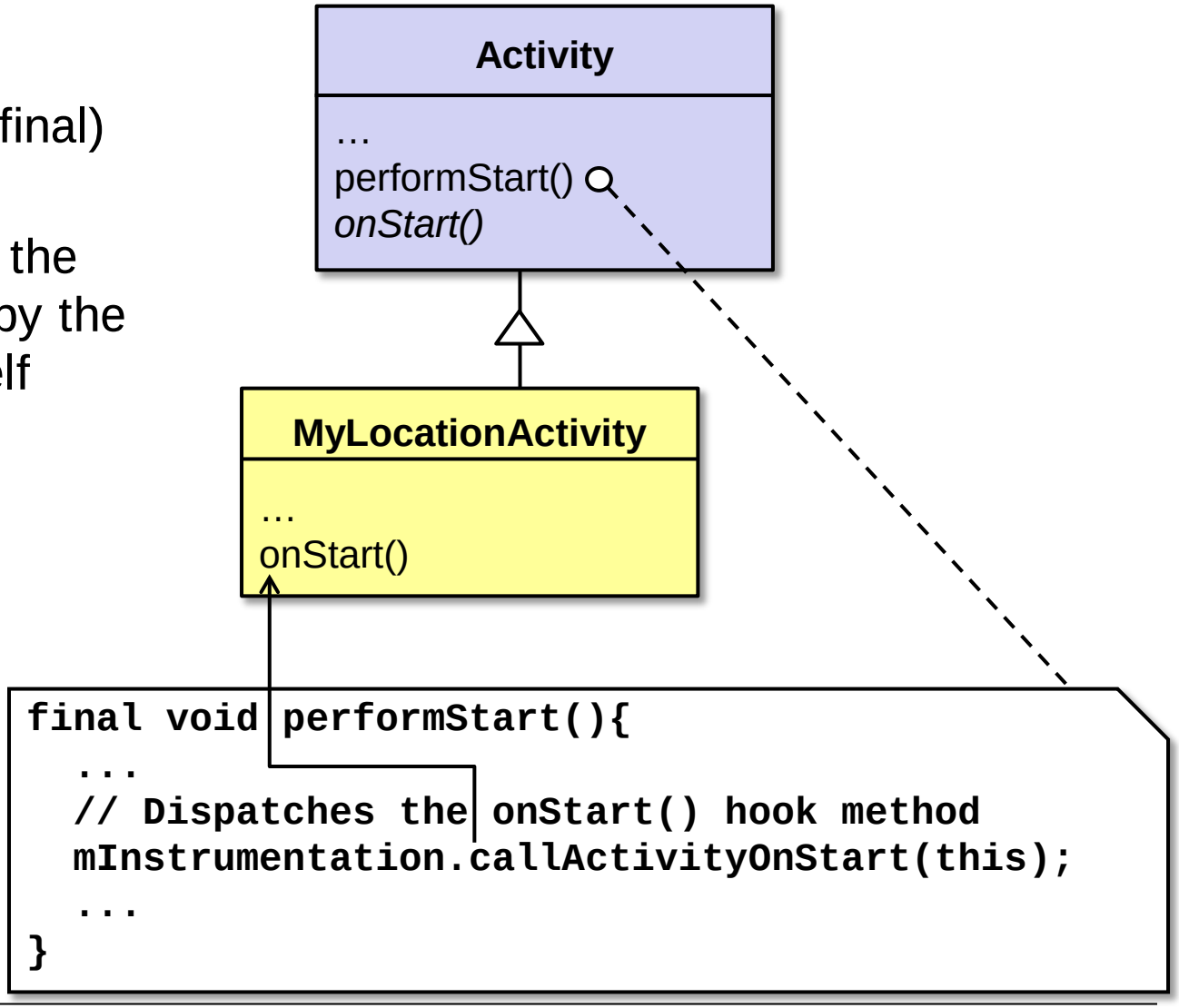
– Must subclass to specialize behavior, which can lead to an explosion of subclasses
  - Compare & contrast with the *Strategy* pattern

– Validation becomes tricky since the proper functioning of the framework depends on the proper functioning of the hook methods!

Template Method GoF Class Behavioral

Implementation

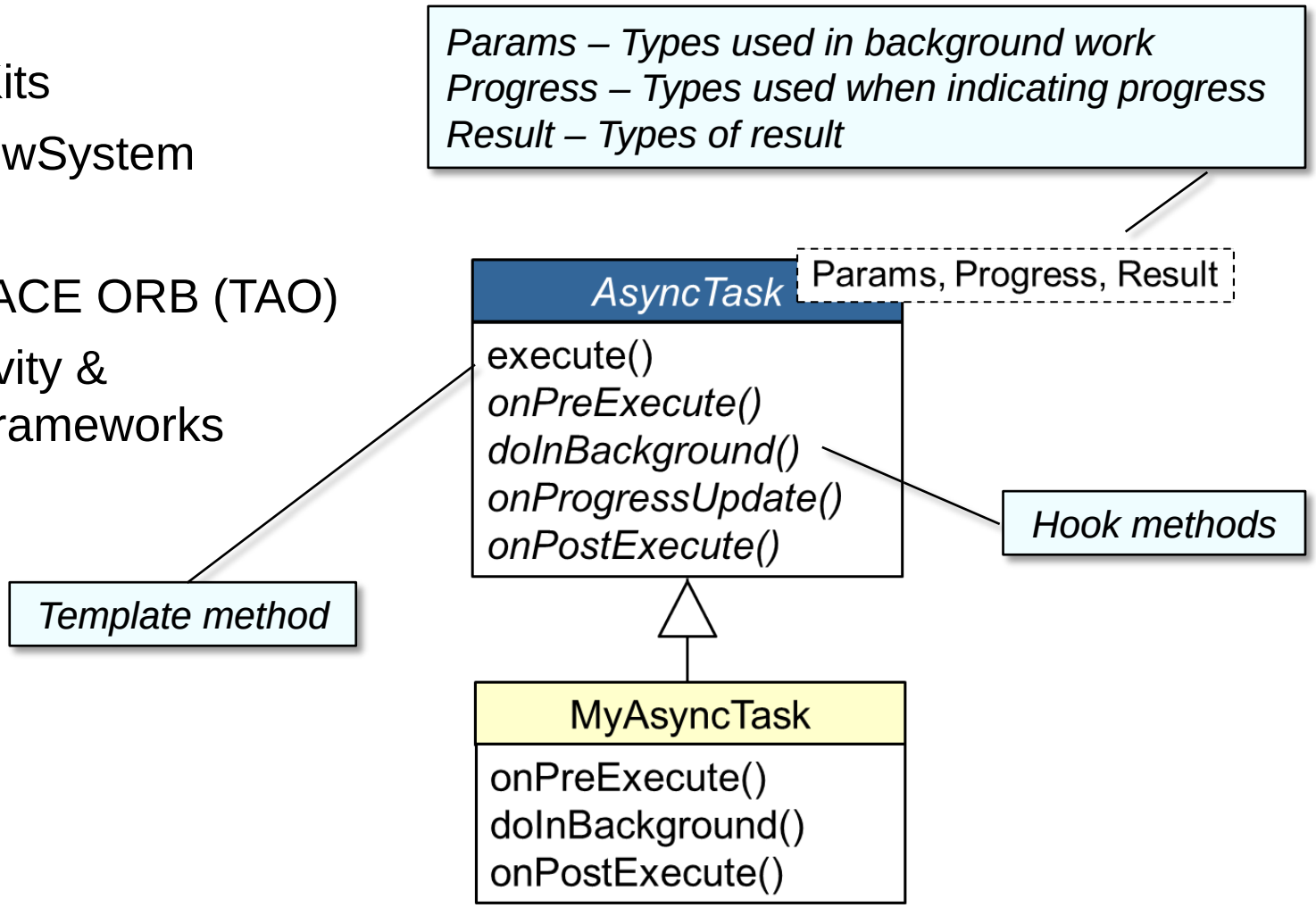
- Virtual vs. non-virtual (final) template method
  - Depends on whether the algorithm embodied by the template method itself may need to change
- Few vs. many primitive operations (hook methods)
- Naming conventions
  - For example, do*() vs. make*() vs. on*() prefixes



Template Method GoF Class Behavioral

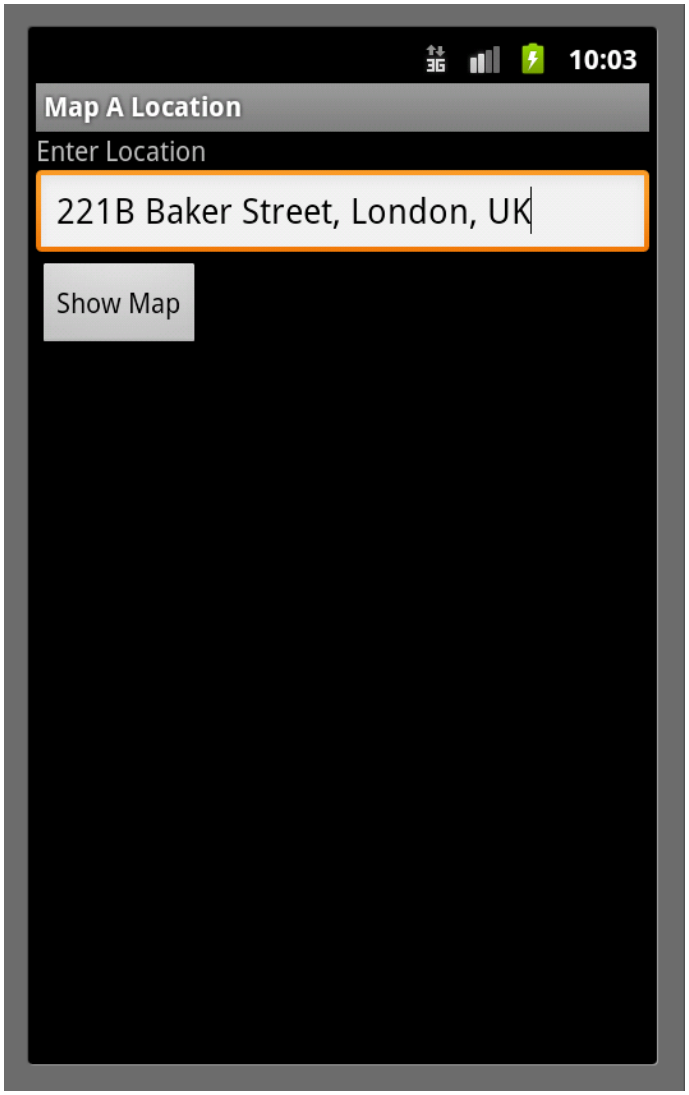
Known Uses

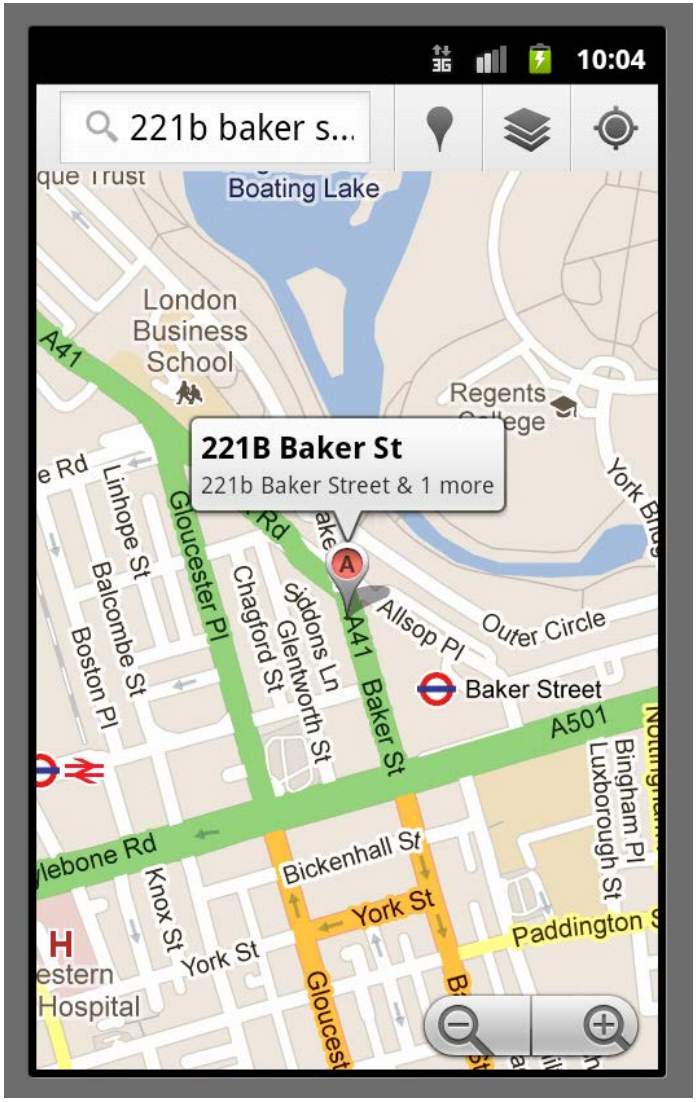
- InterViews Kits
- ET++ WindowSystem
- AWT Toolkit
- ACE & The ACE ORB (TAO)
- Android Activity & AsyncTask frameworks



developer.android.com/reference/android/os/AsyncTask.html has more info

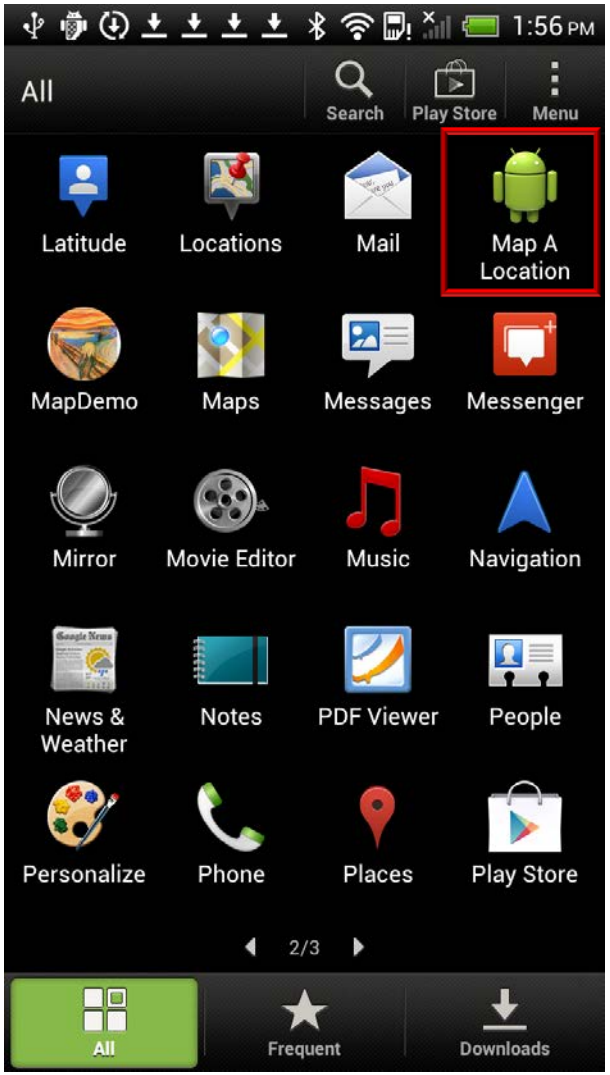
MapLocation App Example





Calling onCreate() in Map App

- The onCreate() method is called when an Activity is first being initialized during app launch
- onCreate() typically initializes global Activity state, e.g.,
  1. Calls super.onCreate()
  2. Inflates & configures UI views as necessary
  3. Sets the Activity's content view



MapLocation.onCreate()

```
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  setContentView(R.layout.main);
  final EditText addressfield = (EditText) findViewById(R.id.location);
  final Button button = (Button) findViewById(R.id.mapButton);
  button.setOnClickListener(new Button.OnClickListener() {
     public void onClick(View v) {
           try {
              String address = addressfield.getText().toString();
              address = address.replace(' ', '+');
              Intent geoIntent = new Intent(android.content.
                 Intent.ACTION_VIEW, Uri.parse("geo:0,0?q=" + address));
              startActivity(geoIntent);
           } catch (Exception e) { /* ...do something else... */ }
        }
     });
  ...
```

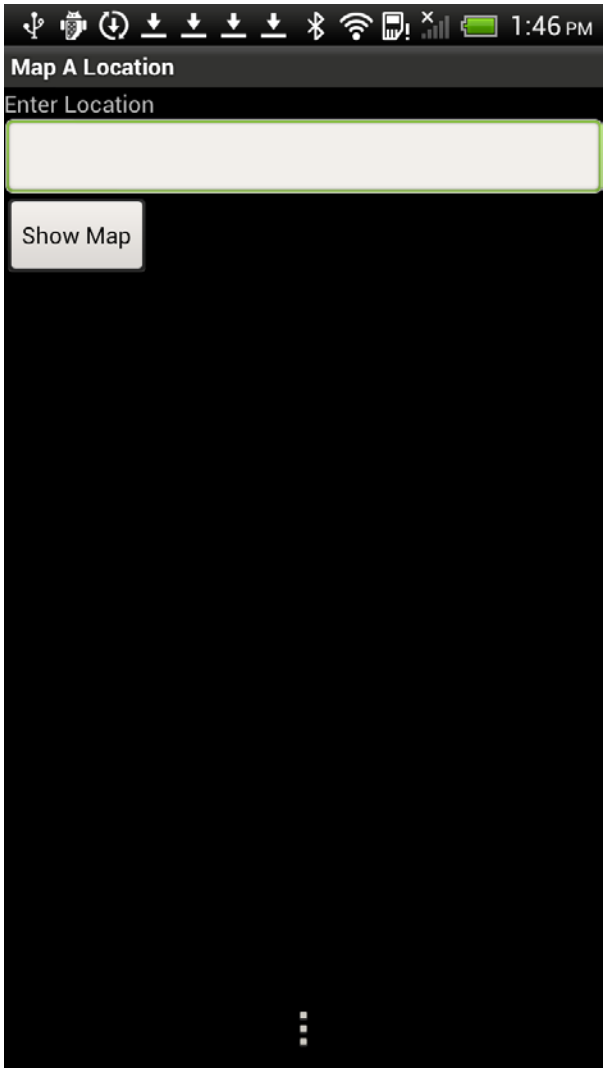
Calls super.onCreate()

Sets the Activity's content view

Configure UI views

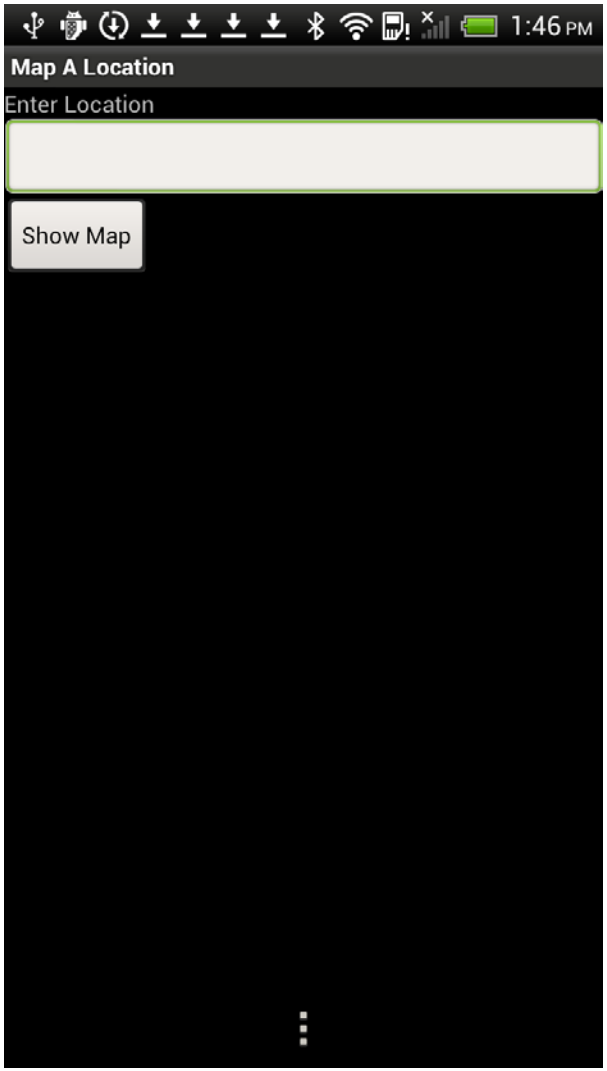
Calling onStart() in Map App

- The onStart() method is called when the main Activity for your app is about to become visible on the display
- Typical actions
  - Reset app state & behavior



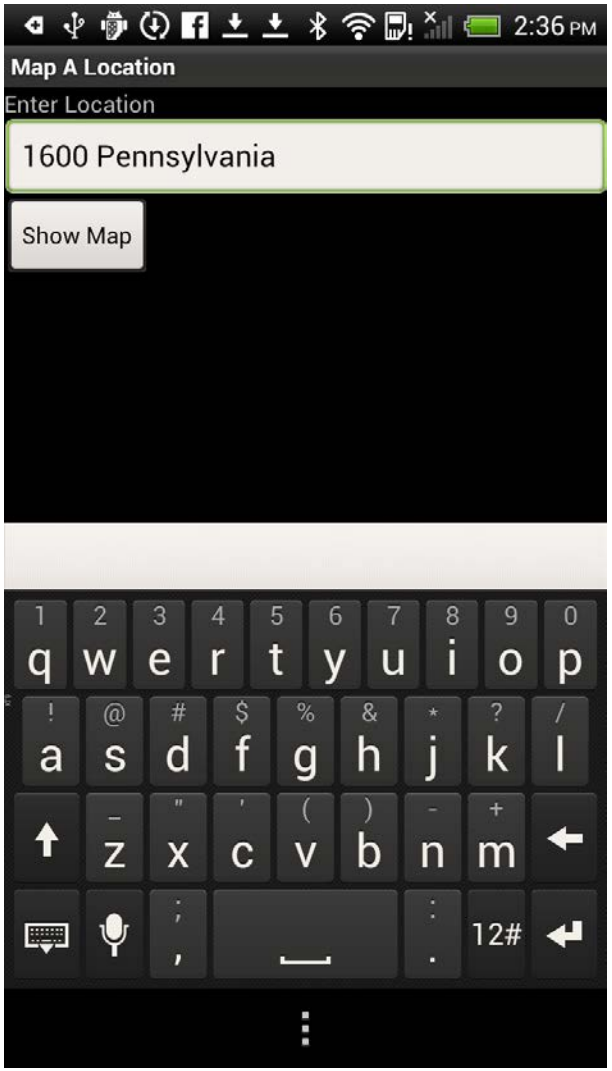
Calling onResume() in Map App

- The onResume() method is called when the main Activity for your app is about to start interacting with the user
- Typical actions
  - Start foreground-only behaviors



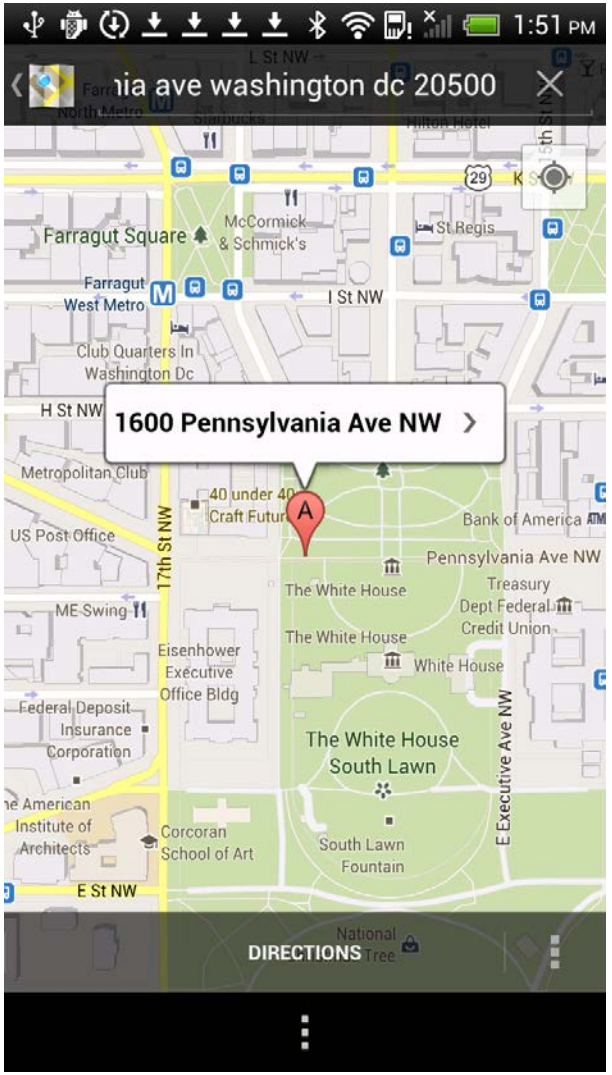
Entering Text & Launch Map Activity

- Clicking on the “Show Map” button will open a new Activity to display the map
- Note that entering text via the virtual keyboard doesn’t change the focus on the UI nor does it generate any lifecycle events



Calling onPause() in Map App

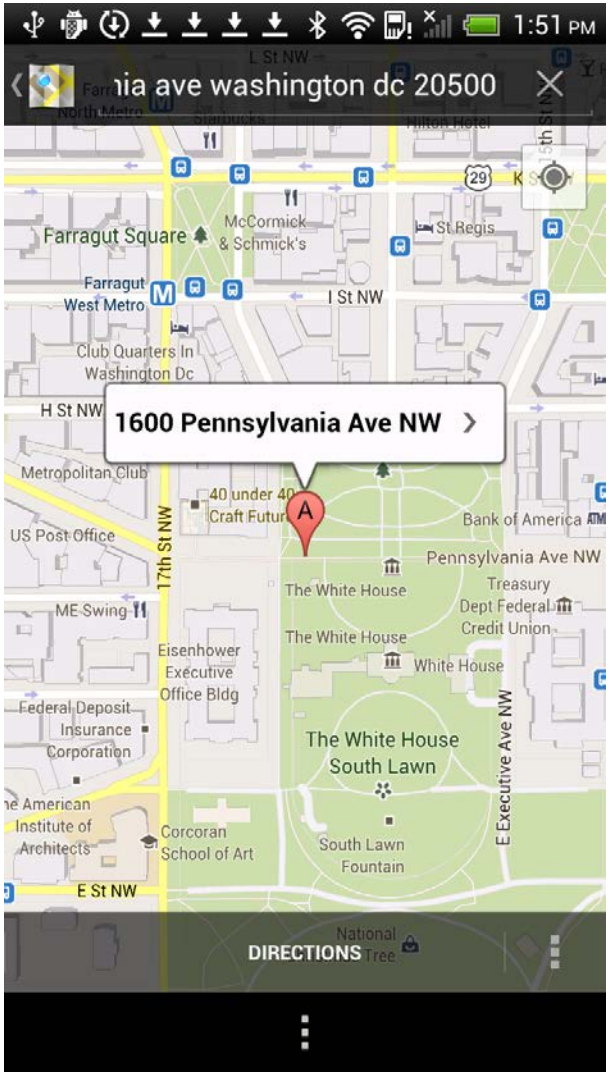
- The onPause() method is called when the focus is about to switch to another Activity
- Typical actions
  - Shutdown foreground-only behaviors



stackoverflow.com/questions/7240916/android-under-what-circumstances-would-a-dialog-appearing-cause-onpause-to-be/7384782#7384782

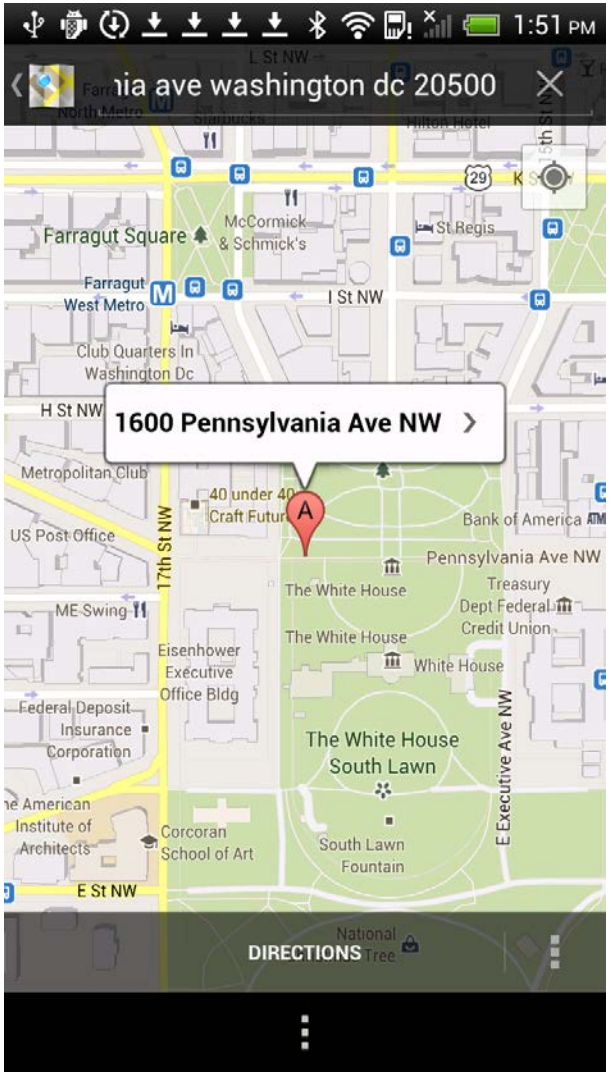
Calling onStop() in Map App

- The onStop() method is called when the Activity is no longer visible to the user (but may be restarted again later)
- Typical actions
  - Cache state



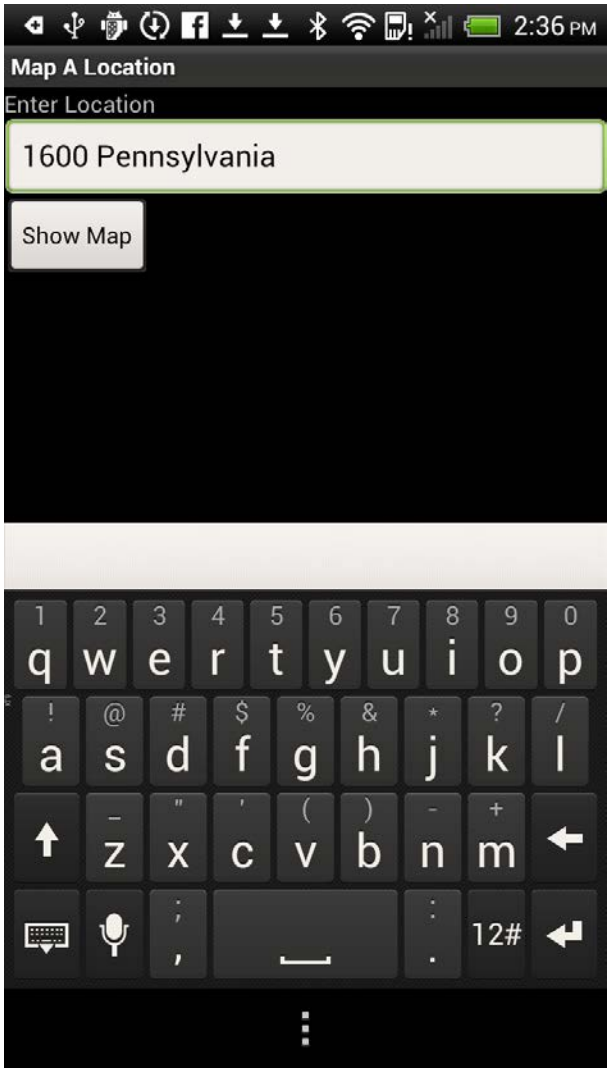
Calling onPause()/onStop() in Map App

- When the google map Activity is launched, its onCreate() & onStart() methods are called automatically by the Android ActivityManager framework
- The prior Activity's onPause() & onStop() methods were previously called



Calling onRestart() in Map App

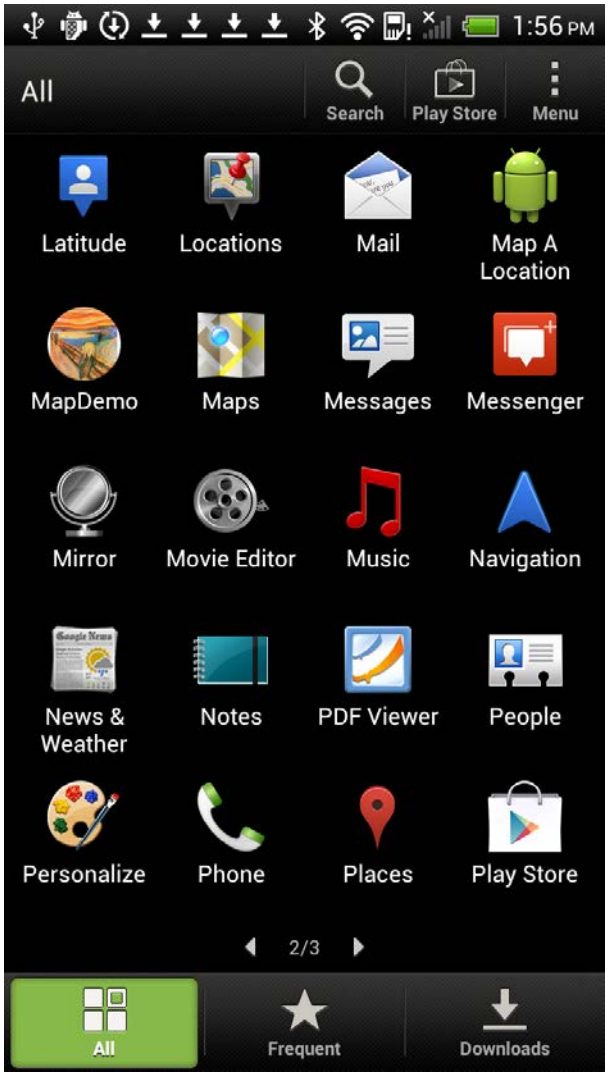
- The onRestart() method is called if the Activity has been stopped & is about to be started again
  - e.g., returning back to a previously launched Activity
- Typical actions
  - Read cached state



developer.android.com/training/basics/activity-lifecycle/stopping.html has more

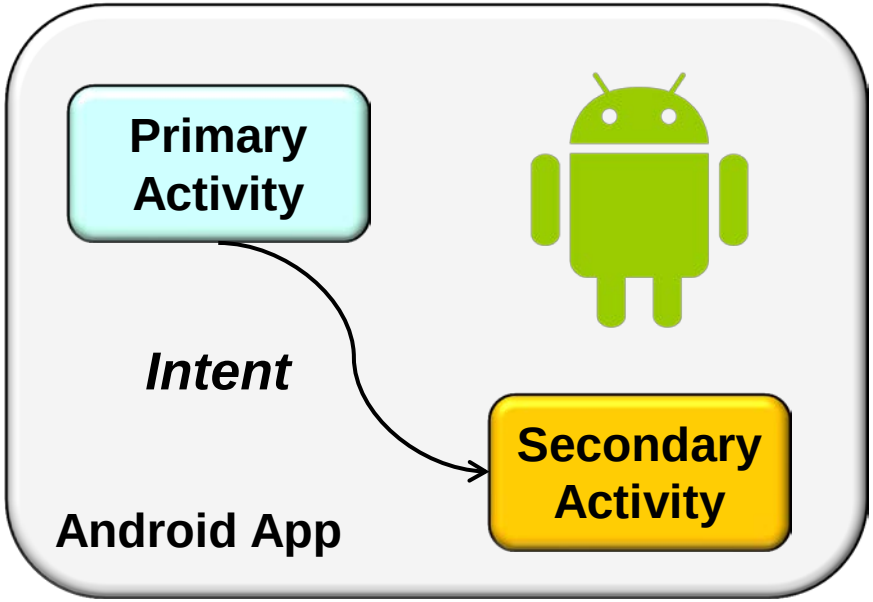
Calling onDestroy() in the Map App

- The onDestroy() method is called when the Activity is about to be destroyed
  - e.g., when the user presses the “back” button
- Typical actions
  - Save persistent state in anticipation of the Activity being recreated later on



Starting Activities

- Create an Intent specifying the Activity to start (Intents are discussed later)



developer.android.com/reference/android/content/Intent.html has more info

Starting Activities

- Create an Intent specifying the Activity to start (Intents are discussed later)
- Pass newly created Intent to one of the following methods
  - startActivity() – Launch a new Activity with no return expected
  - startActivityForResult() – Callback to return result when Activity finishes

Starting Activities

- Create an Intent specifying the Activity to start (Intents are discussed later)
- Pass newly created Intent to one of the following methods
  - startActivity() – Launch a new Activity with no return expected
  - StartActivityForResult() – Callback to return result when Activity finishes
- We use startActivity() in our example app

```
protected void onCreate(Bundle savedInstanceState) {
  public void onClick(View v) {
  …
    Intent geoIntent =
       new Intent(android.content.Intent.ACTION_VIEW,
                  Uri.parse("geo:0,0?q=" + address));
    startActivity(geoIntent);
  …
```

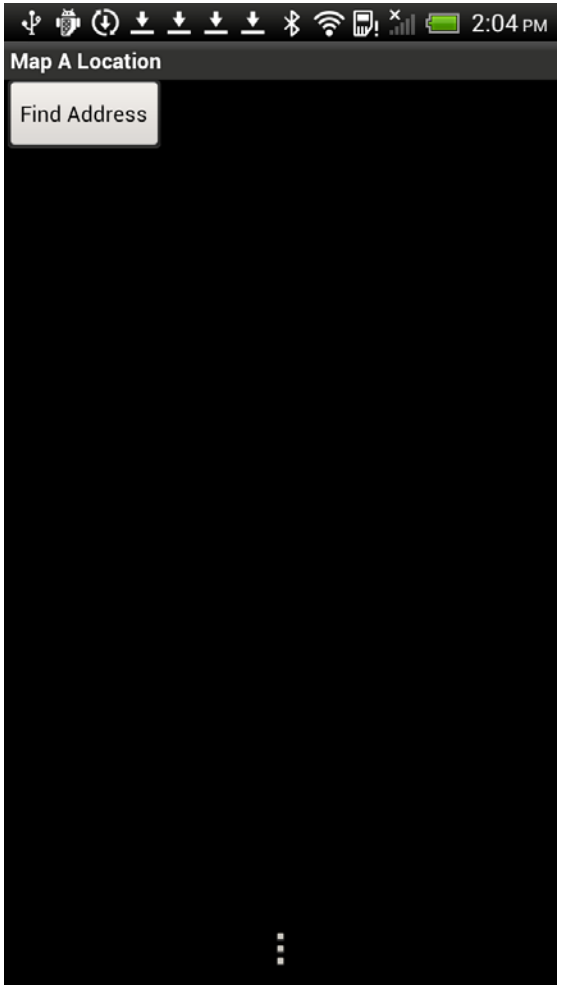
Starting Activities

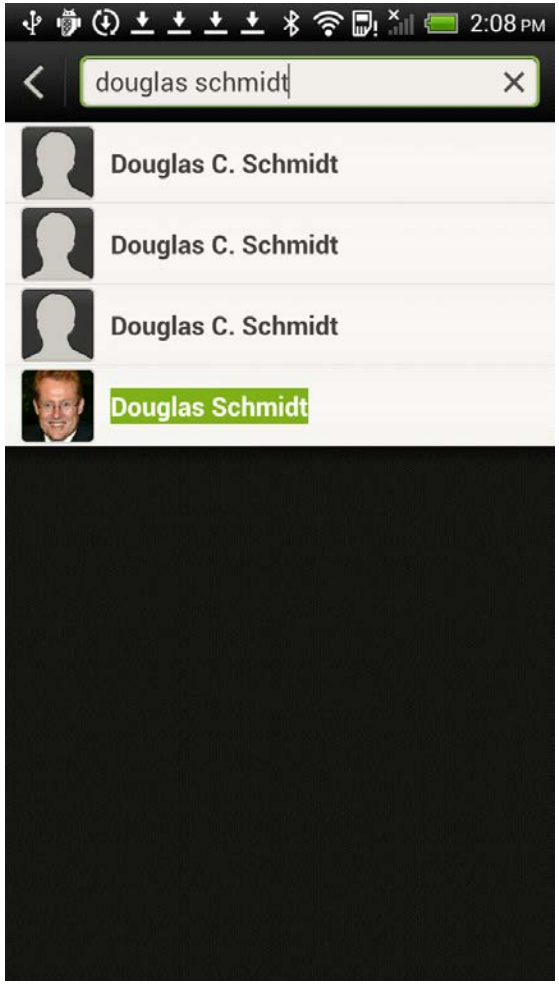
- Create an Intent specifying the Activity to start (Intents are discussed later)
- Pass newly created Intent to one of the following methods
  - startActivity() – Launch a new Activity with no return expected
  - StartActivityForResult() – Callback to return result when Activity finishes
- We use startActivity() in our example app

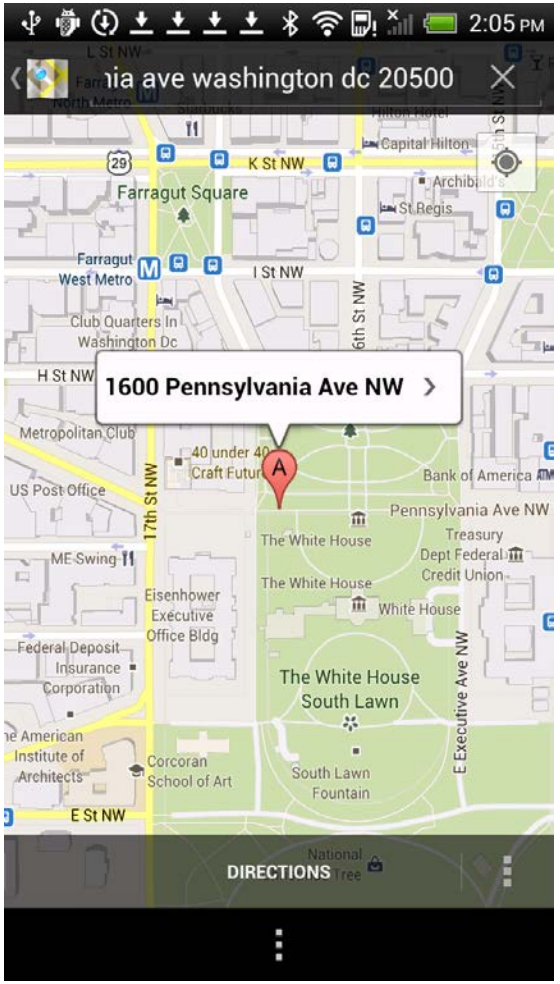
```
protected void onCreate(Bundle savedInstanceState) {
  public void onClick(View v) {
  …
    Intent geoIntent =
       new Intent(android.content.Intent.ACTION_VIEW,
                  Uri.parse("geo:0,0?q=" + address));
    startActivity(geoIntent);
  …
```

- We'll show startActivityForResult() shortly

MapLocationFromContacts







*Not really my address ☺

Using startActivityForResult()

```
private static final int PICK_CONTACT_REQUEST = 0;
...
public class MapLocation extends Activity {
  protected void onCreate(Bundle savedInstanceState) {
  ...
  button.setOnClickListener(new Button.OnClickListener() {
    public void onClick(View v) {
      try {
        Intent intent = new Intent(Intent.ACTION_PICK,
                  ContactsContract.Contacts.CONTENT_URI);
        startActivityForResult(intent,
                               PICK_CONTACT_REQUEST);
      } catch (Exception e) {}
    }});
...
```

Using startActivityForResult()

- Started Activity sets result by calling Activity.setResult()
  - public final void setResult (int resultCode)
  - public final void setResult (int resultCode, Intent data)
- resultCode (an int)
  - RESULT_CANCELED
  - RESULT_OK
  - RESULT_FIRST_USER
- Custom resultCodes can be added after this

Using startActivityForResult()

```
protected void onActivityResult(int requestCode,
                                int resultCode,
                                Intent data) {
  if (resultCode == Activity.RESULT_OK
     && requestCode == PICK_CONTACT_REQUEST) {
     …
     String address =  /* extract address from data */
     Intent geoIntent =
        new Intent(android.content.Intent.ACTION_VIEW,
                   Uri.parse("geo:0,0?q=" + address));
     startActivity(geoIntent);
  }
}
```