Application Basics (COEN 268)

Activities

Santa Clara University

Center for Science, Technology, and Society

Application Components

- **Activities**: They dictate the UI and handle the user interaction to the smart phone screen.
- **Services**: They handle background processing associated with an application.
- **Broadcast Receivers**: They handle communication between Android OS and applications.
- **Content Providers**: They handle data and database management issues.

Application Components (2)

- **Fragments**: Represents a portion of user interface in an Activity
- **Views**: UI elements that are drawn on-screen including buttons, lists, forms
- **Layouts**: view hierarchies that control screen format and appearance
- **Intents**: Messages wiring components together
- **Resources**: External elements, strings, constants, pictures
- **Manifest**: Configuration file for the application

Context

Santa Clara University

Center for Science, Technology, and Society

Context – Abstract Class (cont'd)

From: android.content.context:

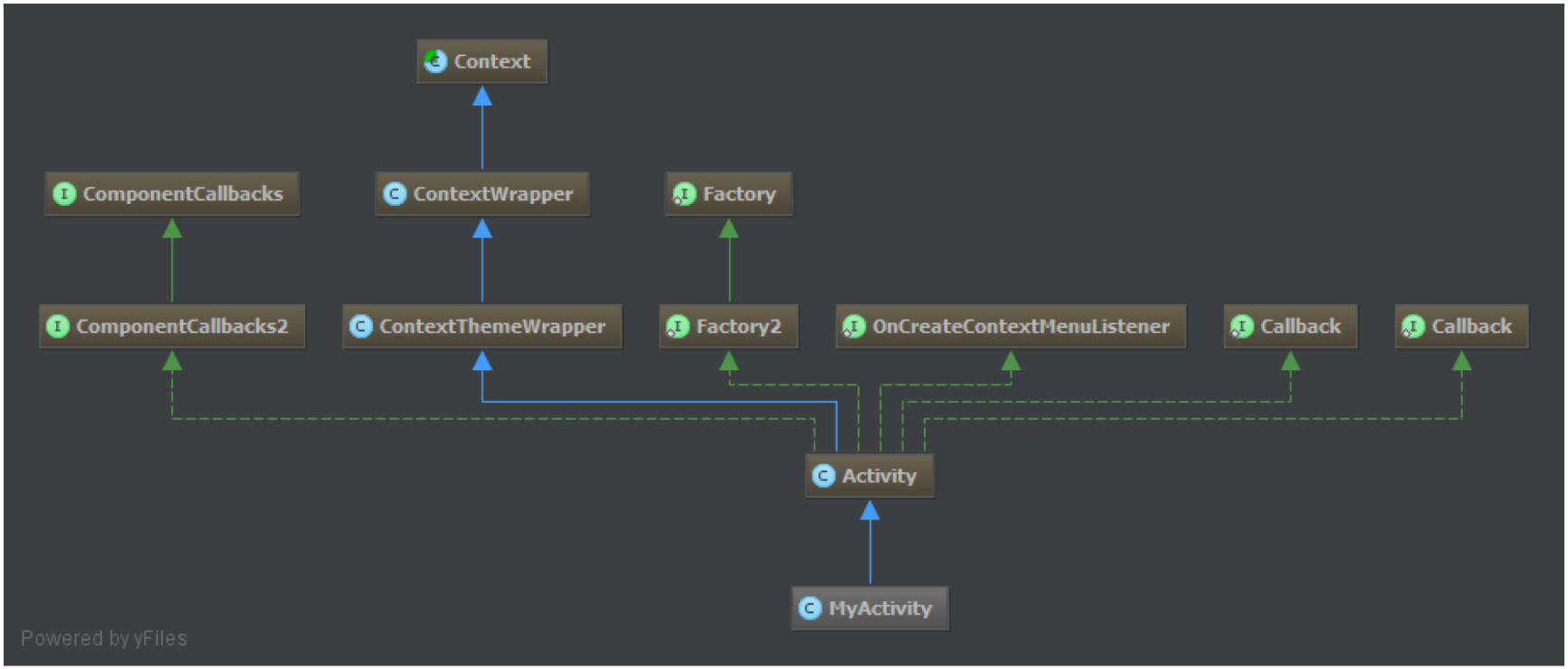
- Interface to global information about an application environment.
- an abstract class whose implementation is provided by the Android system.
- allows access to application-specific resources and classes
- allows up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

Activity

Santa Clara University

Center for Science, Technology, and Society

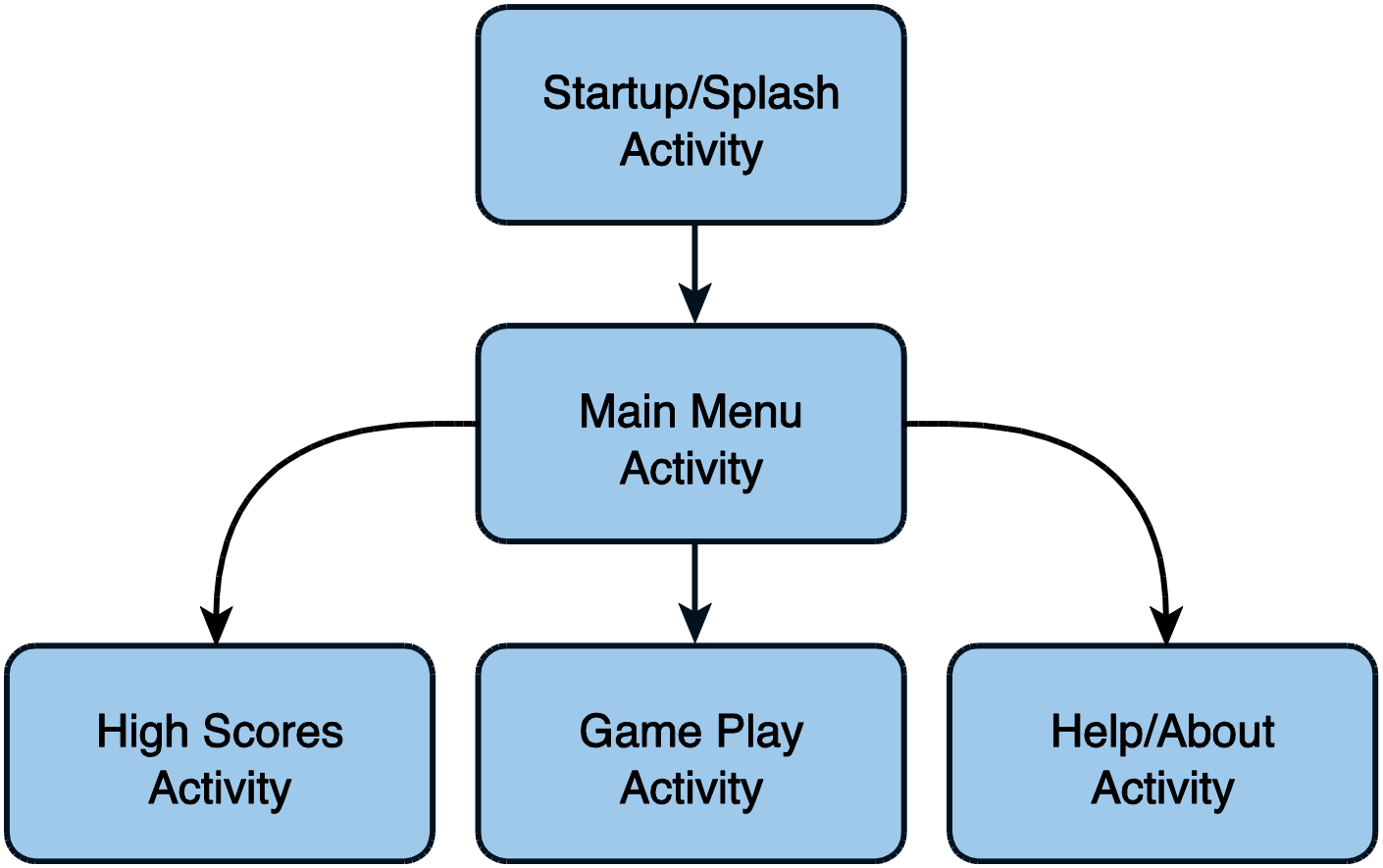
Activity Class



Performing Application Tasks with Activities

- **A simple game application might have the following five activities:**
  1. **Startup or splash screen**
  2. **Main menu screen**
  3. **Game play screen**
  4. **High scores screen**
  5. **Help/About screen**

Simple State Diagram



Activity

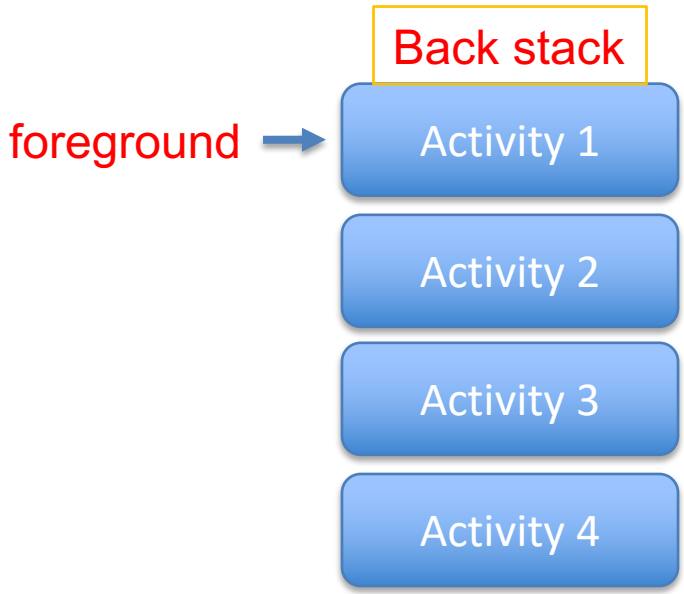
- What is Activity?
  - An **activity** is a single, focused thing that the user can do
  - Almost all activities interact with the user
  - The **Activity** class takes care of creating a window for your app in which you can place your UI with setContentView(View)

Lifecycle of an Android **Activity**

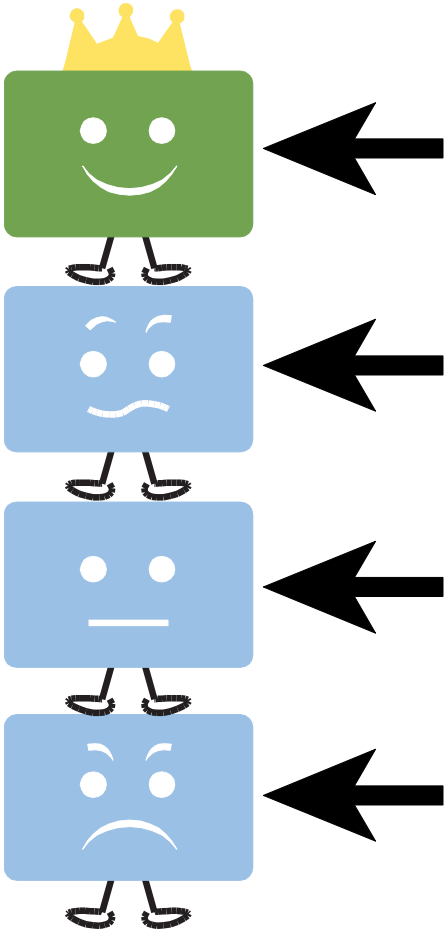
- Android apps can be multi-process.
- Android allows multiple apps to run concurrently (provided memory and processing power are available).
- Applications can have background behavior.
- Applications can be interrupted and paused when events such as phone calls occur.
- In multi-window mode, more than one applications are visible to the user but only one single application `Activity` is in active state at any given time.

Lifecycle of an Android **Activity(2)**

- Android keeps track of all `Activity` objects running by placing them on an `Activity` stack.
- The `Activity` stack is referred to as the "back stack."



Lifecycle of an `Activity` (3)



I am the top Activity.
User can see and interact with me!

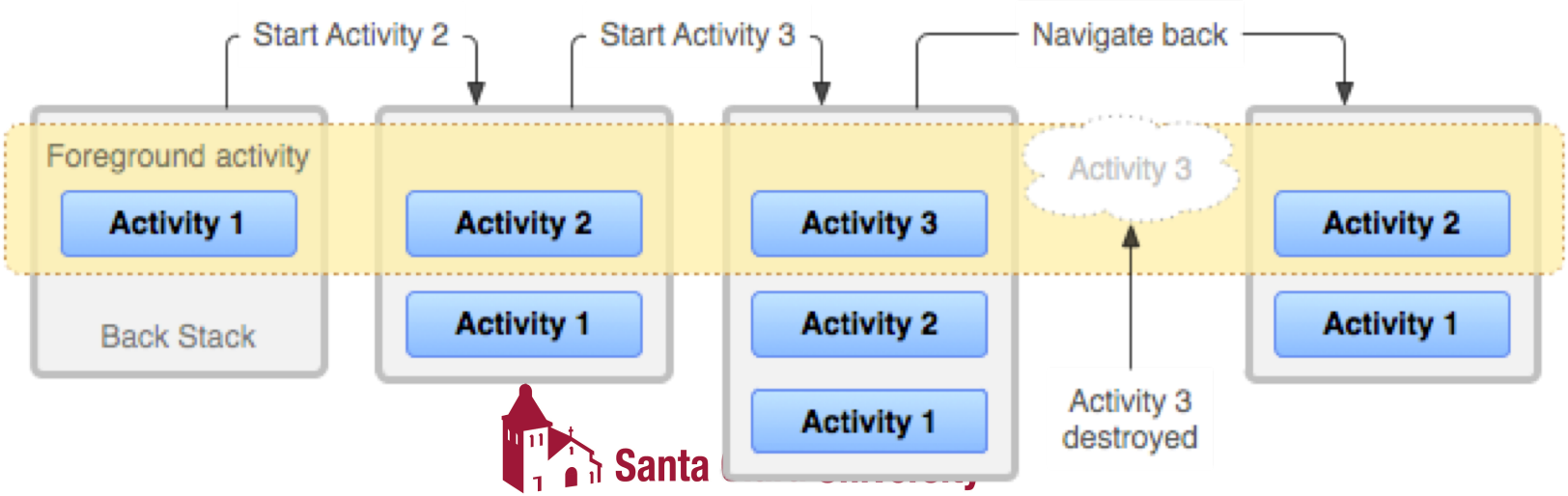
I am the second Activity in the stack.
If the user hits Back or the top Activity is destroyed, the user can see and interact with me again!

I am an Activity in the middle of the stack.
Users cannot see and interact with me until everyone above me is destroyed.

I am an Activity at the bottom of the stack.
If those Activities above me use too many resources, I will be destroyed!

Activity Stack

- When a new `Activity` starts, the `Activity` on the top of the stack (the current foreground `Activity`) pauses, and the new `Activity` pushes onto the top of the stack.
- When that `Activity` finishes, it is removed from the `Activity` stack, and the previous `Activity` in the stack resumes.



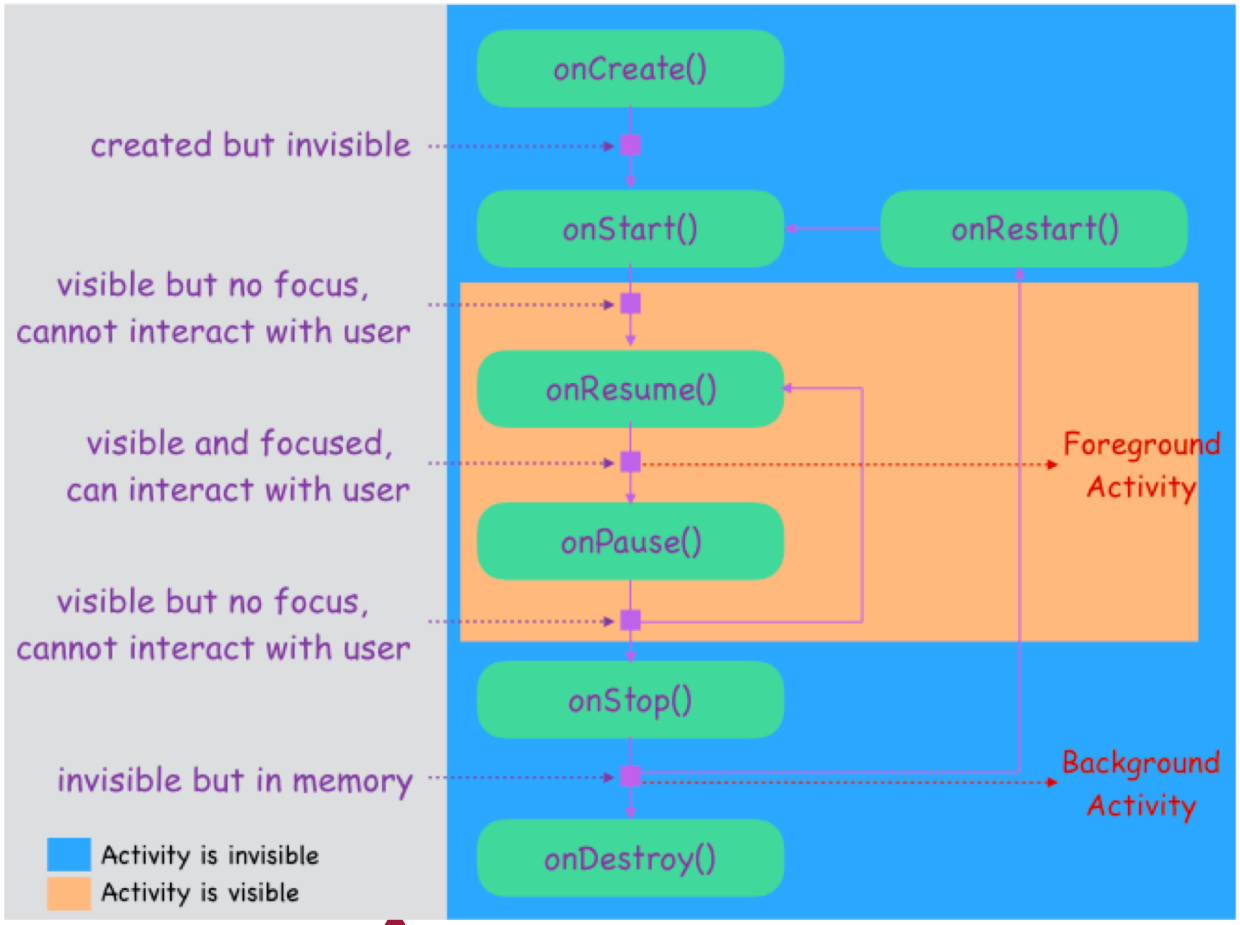
The out-of-memory killer

- To manage limited resourced on a device, Android system can terminate running processes.
- Applications without any active components are added to a least recently used list (LRU list).
- Processes at the beginning of the list will be the ones killed by the out-of-memory killer.

`Activity` Callbacks to Manage App

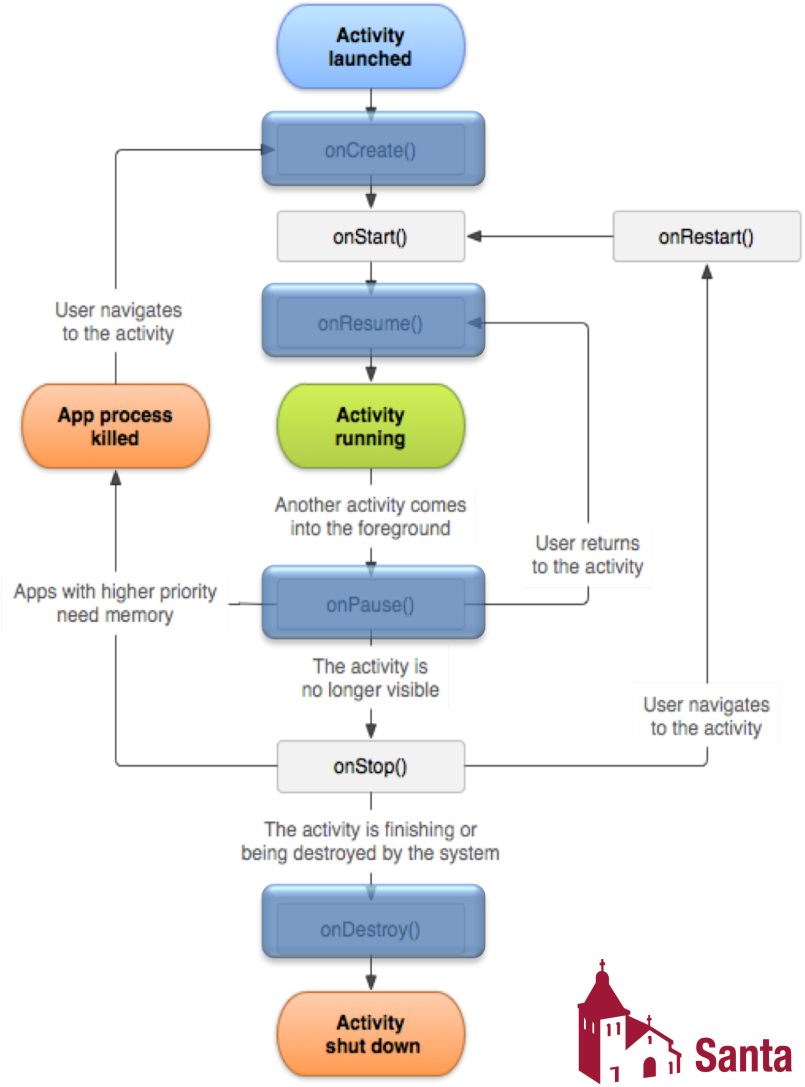
```
public class MyActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState);
    protected void onStart();
    protected void onRestart();
    protected void onResume();
    protected void onPause();
    protected void onStop();
    protected void onDestroy();
}
```

Activity Life Cycle



From codevoila

Activity Diagram




```
public class MyActivity 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();
}
```

Initializing Static `Activity` Data in `onCreate()`

- When an `Activity` first starts, `onCreate()` is called.
- `onCreate()` has a single parameter, a `Bundle` (null if newly started `Activity`).
- If this `Activity` is a restarted `Activity`, `Bundle` contains previous state information so that it can reinitiate.
- Perform setup (layout and data binding), such as `setContentView()`, in `onCreate()`.

Activity Class

```
import android.app.Activity;

public class MainActivity extends Activity {
    ...

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

Initializing and Retrieving `Activity` Data `onResume()`

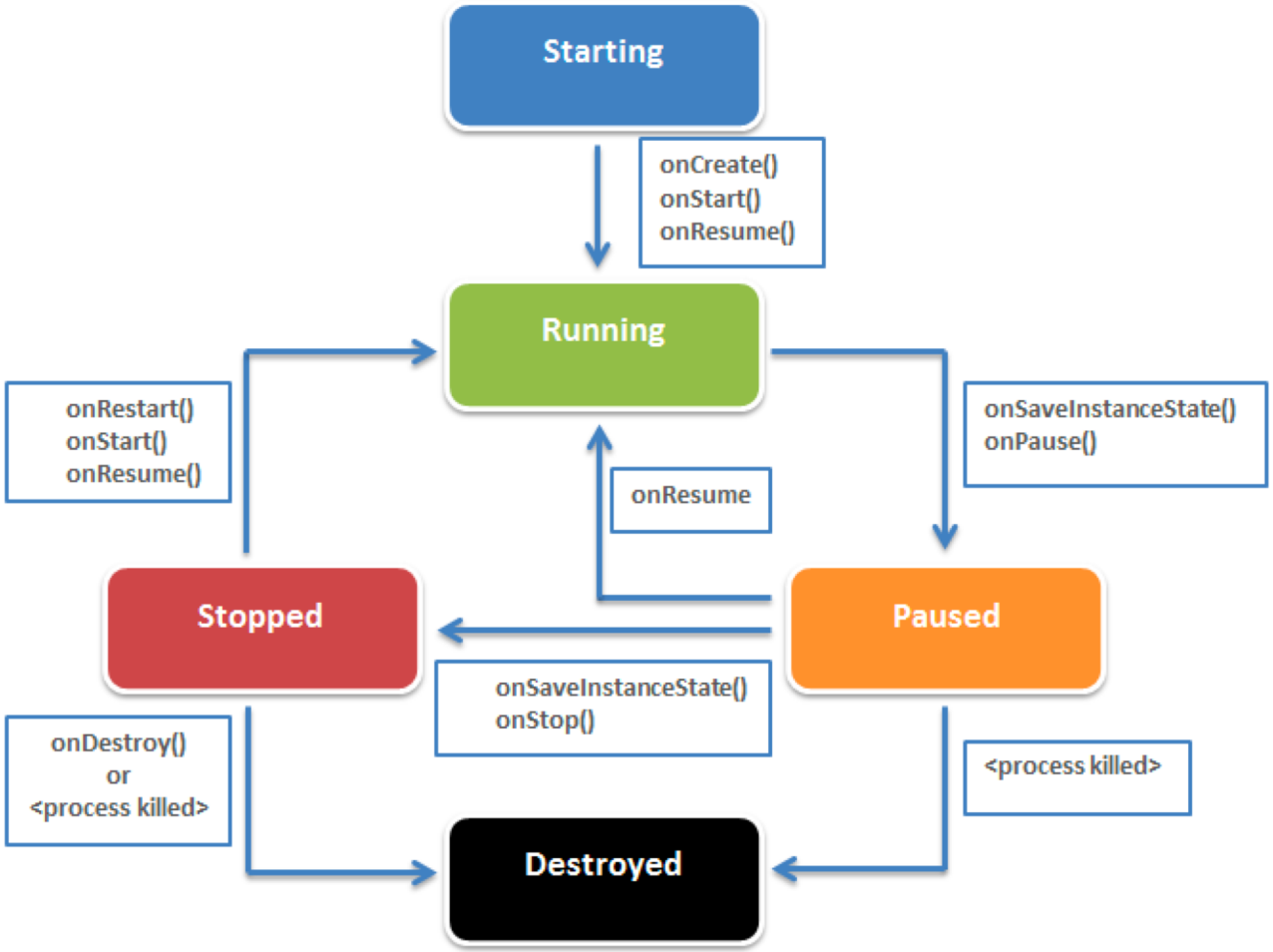
- When the `Activity` reaches the top of the stack and becomes the foreground process, `onResume()` is called.
- This is the most appropriate place to retrieve any instances of resources that the `Activity` needs to run.
- These resources are the most process intensive, so we keep them around only while the `Activity` is in the foreground.

Important Tip

Any resources and data retrieved in `onResume()` should be released in `onPause()`

If they aren't, some resources may not be cleanly released if the process is terminated!

Activity State Diagram



Avoiding Activities Being Killed

- If the `Activity` is killed after `onPause()`, chances are the `onStop()` and `onDestroy()` methods will not be called.
- The more resources released by an `Activity` in the `onPause()` method, the less likely the `Activity` is to be killed while in the background without further state methods being called.

Saving **Activity** State

- If an `Activity` is vulnerable to being killed by Android, you can save state info to a `Bundle` with **`onSaveInstanceState().`**
  - This call is not guaranteed, so use `onPause()` for essential data commits.
- What is recommended?
  - Save important data to persistent storage in `onPause()`, but use `onSaveInstanceState()` to store any data that can be used to rapidly restore the current screen to the state it was in (as the name of the method might imply)

Saving **Activity** State **(2)**

- When this `Activity` is returned to later, this `Bundle` is passed in to the `onCreate()` method, allowing the `Activity` to return to the exact state it was in when the `Activity` paused.

- You can also read `Bundle` information after the `onStart()` callback using `onRestoreInstanceState()`.

- When the `Bundle` information is there, restoring the previous state will be faster and more efficient than starting from scratch.

Multi-window LifeCycle

- **In** multi-window mode, only one Activity has the focus, and is being considered Active.
- All other Activities are in the paused state
- System gives paused-but-visible activities higher priority than non-visible activities.
- If user interacts with one of the paused activities, the Activity is resumed, and the previously active Activity is paused.

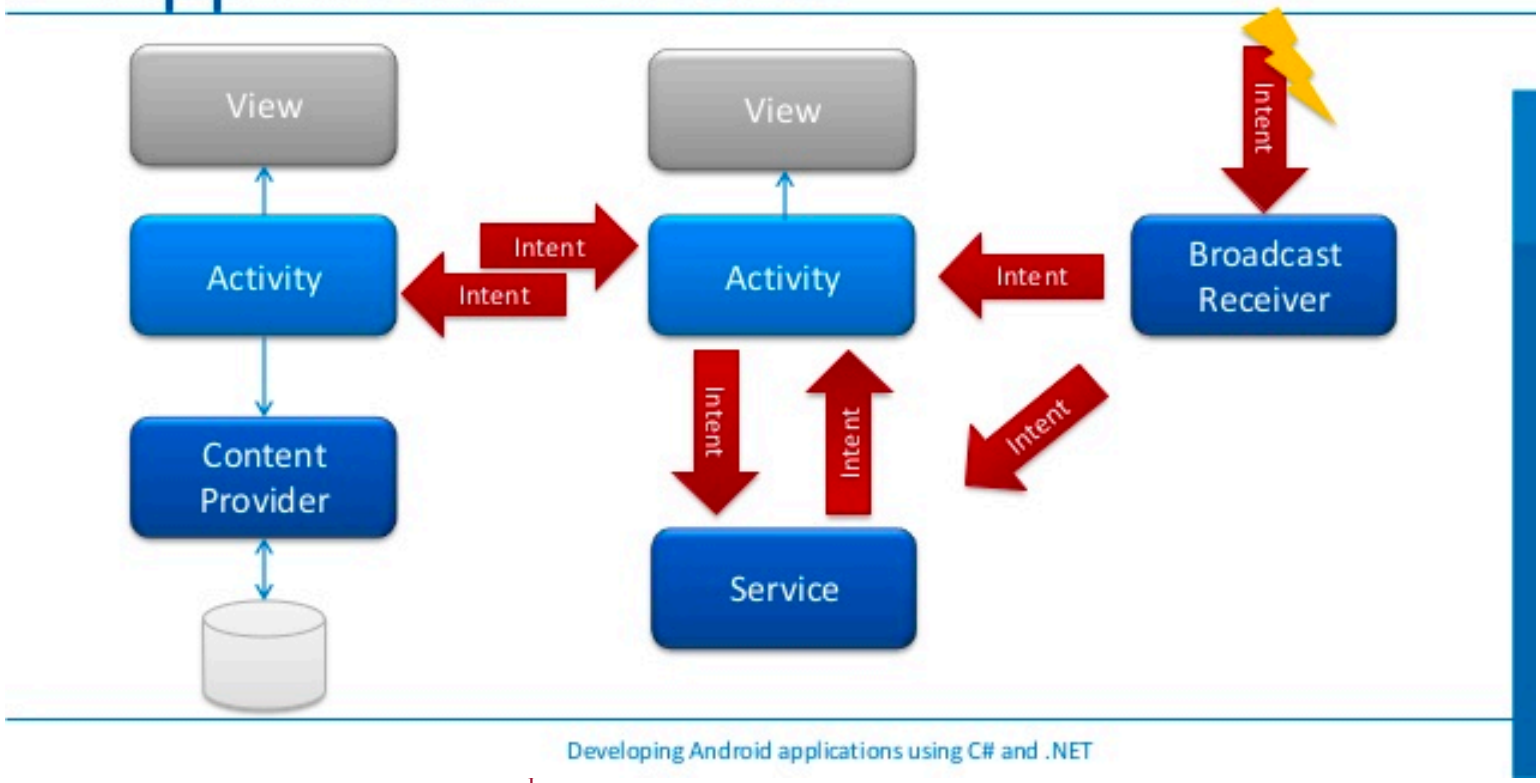
Transitioning between Activities with Intents

- Android applications can have multiple entry points.
- A specific `Activity` can be designated as the main `Activity` to launch by default.
- Other activities might be designated to launch under specific circumstances.

Launching a New `Activity` by Class Name

- You can start activities in several ways.
- The simplest method:
  - Use the `Application Context` object to call `startActivity()`.
  - `startActivity()` takes a single parameter, an `Intent`.
- An `Intent` (`android.content.Intent`) is an asynchronous message mechanism. It is used by Android to match task requests with the appropriate `Activity` or `Service` (launching it, if necessary) and to dispatch broadcast `Intent` events to the system at large.

App model - Intents



Launching a New `Activity` by Class Name (2)

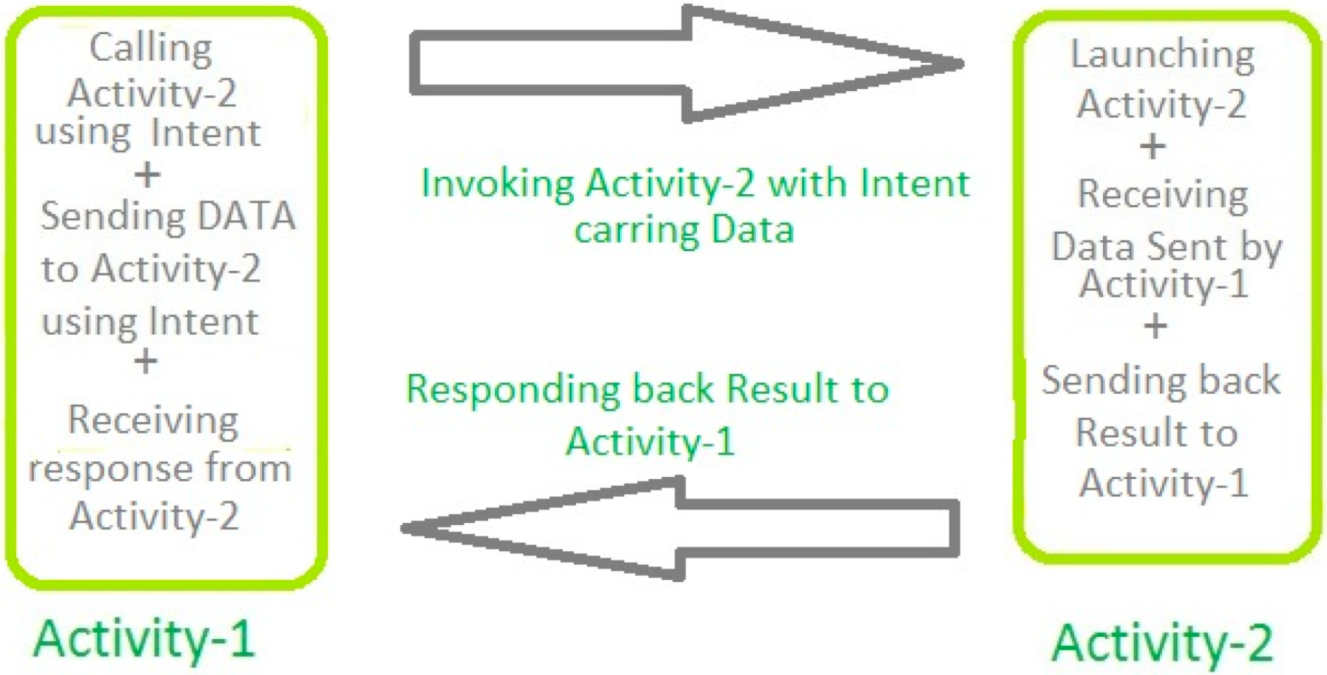
```
public class MainActivity extends AppCompatActivity {
    public static final String EXTRA_MESSAGE =
"com.example.myfirstapp.MESSAGE";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    /** Called when the user taps the Send button */
    public void sendMessage(View view) {
        Intent intent = new Intent(this, DisplayMessageActivity.class);
        EditText editText = (EditText) findViewById(R.id.editText);
        String message = editText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
    }
}
```

Creating Intents with Action and Data

- The guts of the `Intent` object are composed of two main parts:
  - The action to be performed
  - (Optionally) the data to be acted upon
- You can also specify action/data pairs using `Intent Action` types and `Uri` objects.
- Therefore, an `Intent` is basically saying "do this" (the action) to "that" (the URI describing on what resource the action is performed).

Explicit Intent

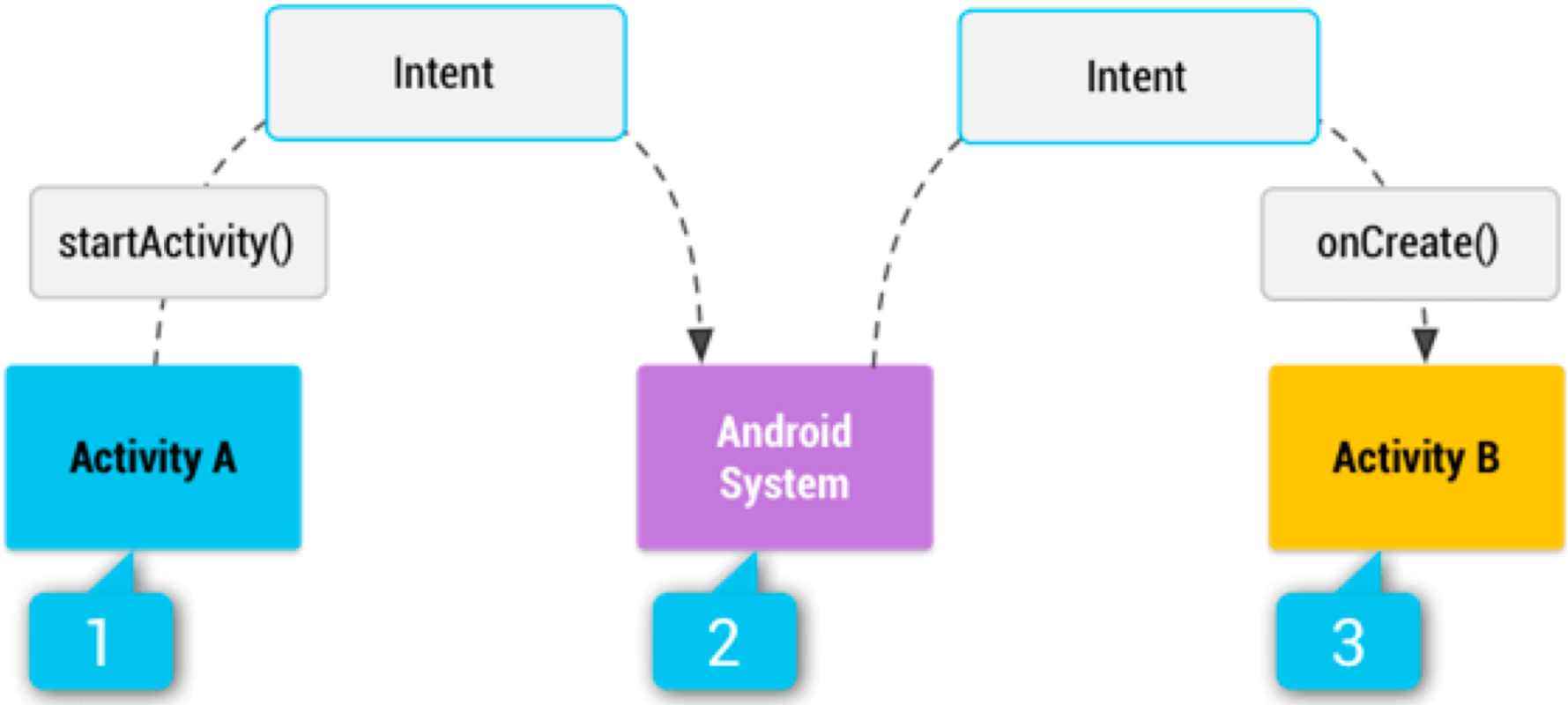


Working of Explicit Intent (With Response Data from Activity-2)

Creating Intents with Action Data (2)

- The most common action types are defined in the `Intent` class, including:
  - `ACTION_MAIN`
    - Describes the main entry point of an `Activity`
  - `ACTION_EDIT`
    - Used in conjunction with a URI to the data edited
- You also find action types that generate integration points with activities in other applications:
  - The browser or Phone Dialer

Implicit Intent



Launching an `Activity` Belonging to Another App

- With the appropriate permissions, applications might also launch external activities within other applications.
  - For example, a customer relationship management (CRM) app might launch the Contacts app to browse the Contacts database, choose a specific contact, and return that contact's unique identifier to the CRM application for use.

Launching an `Activity` Belonging to Another Application (2)

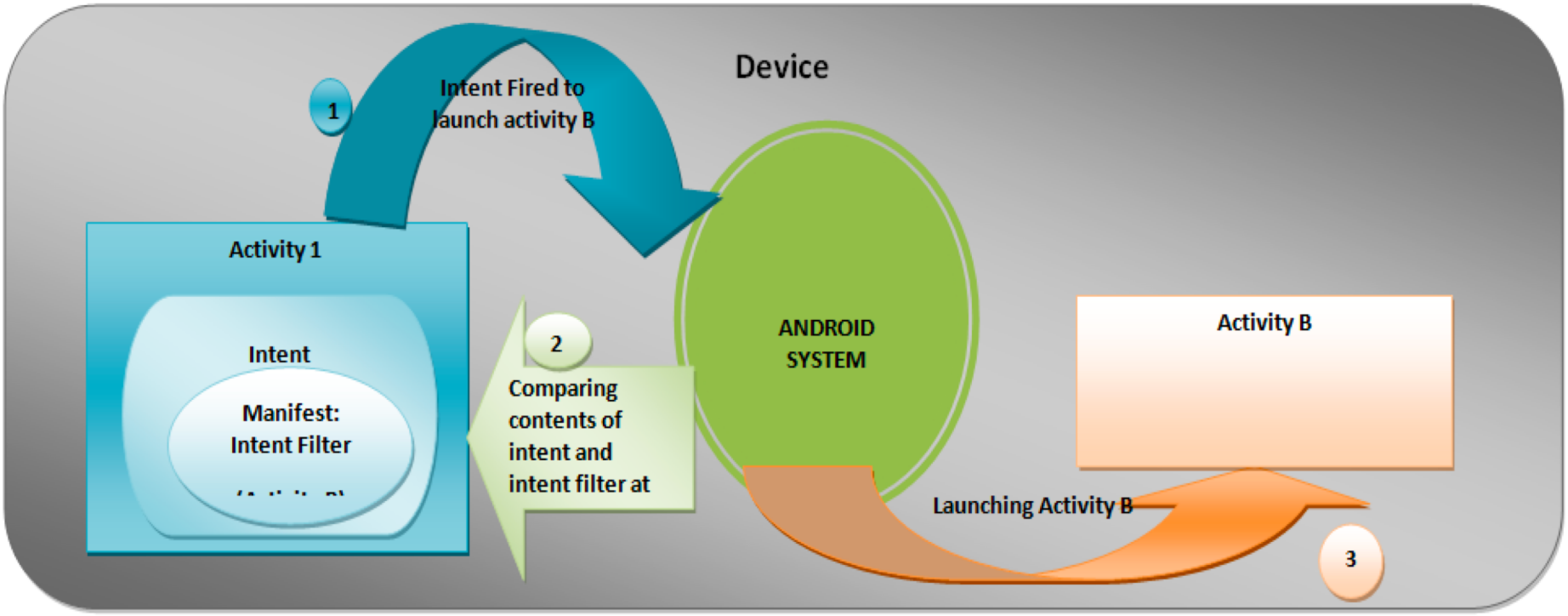
Explicit :

```
Intent intent = new Intent(this,Target.class)
startActivity(intent);
```

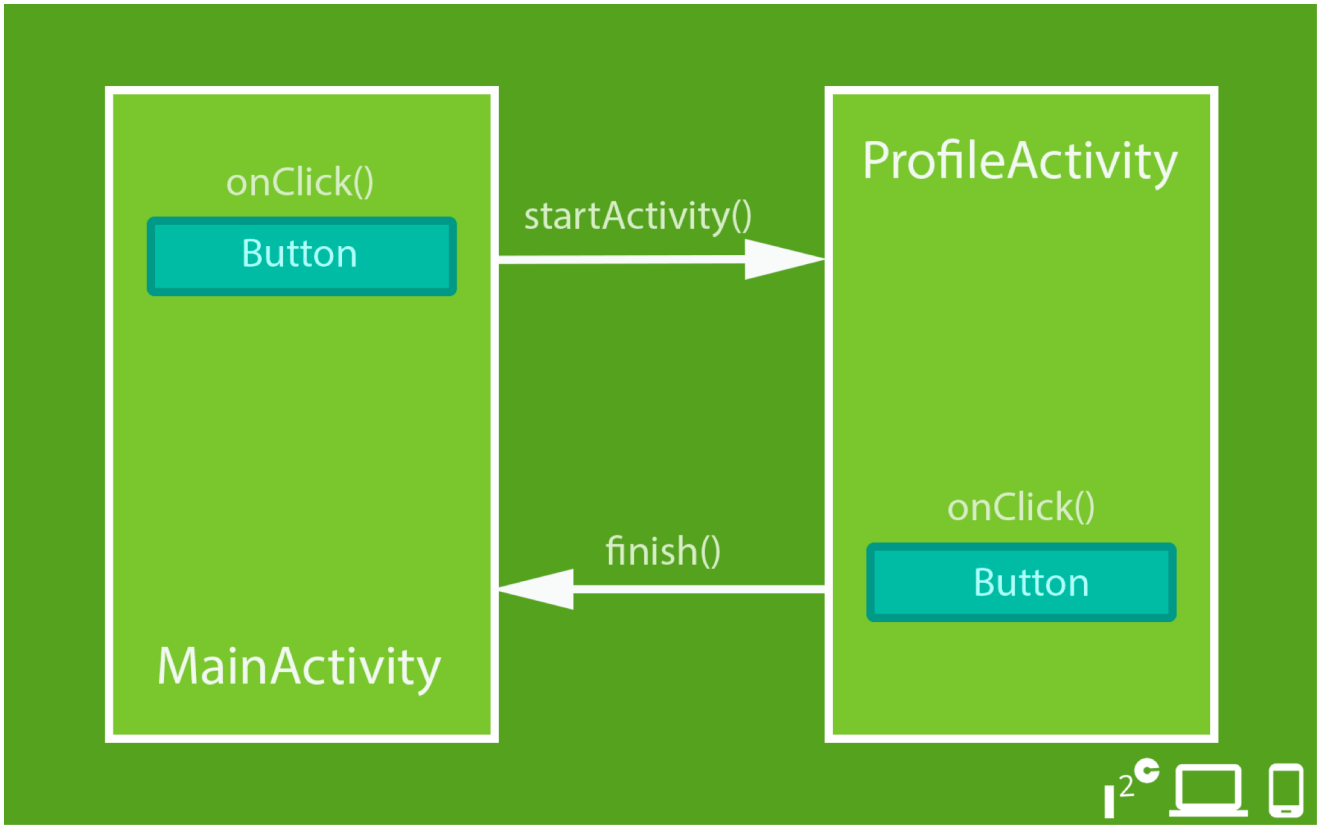
Implicit:

```
Intent intent = new Intent(ACTION_VIEW,Uri.parse("http://www.scu.edu");
startActivity(intent);
```

Implicit Intent



Exercise



https://www.learn2crack.com/2016/10/android-switching-between-activities-example.html

Logging

Santa Clara University

Center for Science, Technology, and Society

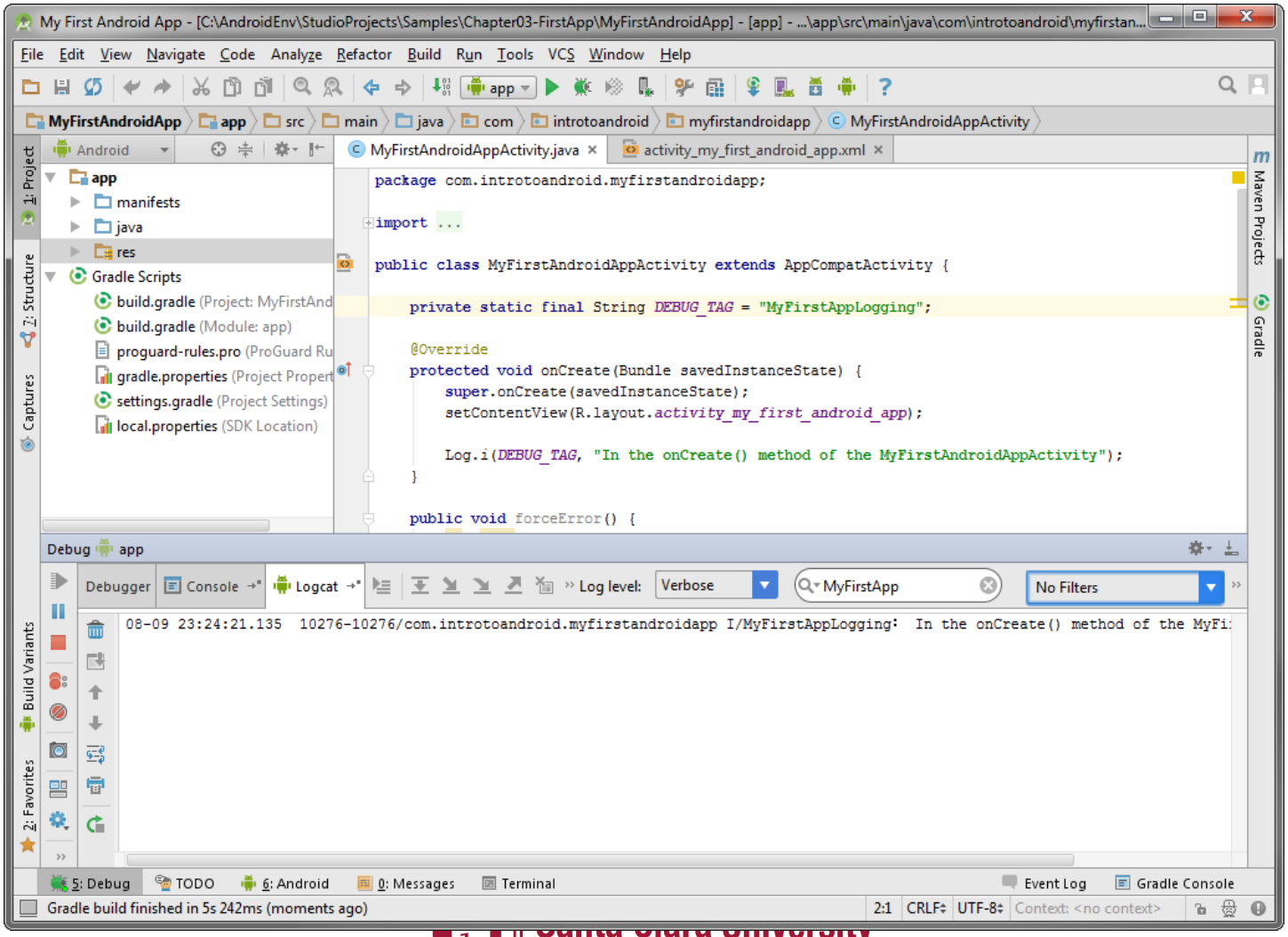
Logging Support

- Before you start diving into the various features of the Android SDK, you should familiarize yourself with logging.
- Logging is a valuable resource for debugging and learning Android.
- Android logging features are in the `Log` class of the `android.util` package.

Logging Support (2)

- **Logging Methods**
  - `Log.e()`
  - `Log.w()`
  - `Log.i()`
  - `Log.d()`
  - `Log.v()`
- **Add the `Log` class using the import statement:**
  - `import android.ultil.Log`
- **Add a variable for collecting your debug messages:**
  - `private static final String DEBUG_TAG = "MyFirstAppLogging"`
- **In the call back method add the log statement:**
  - `Log.i(DEBUG_TAG, "In the onCreate() method of the MyFirstAndroidAppActivity Class");`

Logging Support (3)



Manifest

Santa Clara University

Center for Science, Technology, and Society

Configuring Android Application Using Manifest

- What is the Android manifest file?
  - Specially formatted XML file required for each Android application
  - Defines application name
  - Defines components the application relies on
  - Defines permissions the application requires to run
  - Named `AndroidManifest.xml`

Layers of Manifest

```
<manifest>
  <application>
    <activity>
    <activity-alias>
    <meta-data>
    <service>
    <receiver>
    <provider>
    <uses-library>
  </application>
<manifest>
```

Configuring Android Manifest

AndroidManifest.xml information is used by the Android system to

- Install and upgrade the application package
- Display the application's name, description, and icon to users
- Specify application system requirements
  - Device configurations required (e.g., D-pad navigation)
  - Platform features relied upon (e.g., multitouch)
- Specify features required by the application for market filtering
- Register application activities and when they should be launched
- Manage application permissions
- Configure and define services, broadcast receivers, and content providers
- Specify intent filters for activities, services, and broadcast receivers
- Enable application settings and configurations

Editing Manifest

- Use Android Studio to manually edit the manifest.XML
- Android manifest files include a single <manifest> tag with a single <application> tag.

Sample File

A Sample AndroidManifest.xml

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.android.borderlessbuttons"
   android:versionCode="1"
   android:versionName="1.0">

  <!--
     This sample requires API 11 for use of theme attributes such as
?android:buttonBarStyle
     and ?android:borderlessButtonStyle, as well as LinearLayout's
android:showDividers attribute.
     A similar effect can be achieved by setting a clickable view's
background to
     ?android:selectableItemBackground.

     This sample requires API 14 for use of theme attributes such as
     ?android:listPreferredItemPaddingLeft.
  -->
  <!-- Min/target SDK versions (<uses-sdk>) managed by build.gradle -->

   <application android:label="@string/app_name"
      android:icon="@drawable/ic_launcher"
      android:theme="@style/Theme.Sample"
      android:allowBackup="true">

      <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>

   </application>

</manifest>
```

Managing your Application Identity

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.android.borderlessbuttons"
    android:versionCode="1"
    android:versionName="1.0" > >
```

Setting the Application Name and Icon

- **Overall application settings are configured with the <application> tag of the Android manifest file.**
- **Here, you set information such as the application icon (android:icon) and friendly name (android:label).**
- **These settings are attributes of the <application> tag.**

```
<application android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name" >
```

Setting the Application Name and Icon (2)

- You can also set optional application settings as attributes in the <application> tag:
  - android:description
  - android:debuggable="true”
  - android:supportsRtl="true"

```
https://developer.android.com/guide/topics/manifest/application-
   element.html
```

Setting the Application Name and Icon

/Users/feskafi/AndroidStudioProjects/11sight-android-sdk/app/src/main

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   package="com.elevensight.elevensight">


   <application
      android:name=".sdk.SDKManager"
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
      android:supportsRtl="true"
      android:theme="@style/AppTheme"
      tools:replace="name">
      <activity android:name=".ui.WelcomeActivity">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
      <activity android:name=".ui.MainActivity" />
      <activity android:name=".ui.SigninActivity" />
      <activity android:name=".ui.SettingsActivity"></activity>
      <service
         android:name=".fcm.MyFirebaseMessagingService">
         <intent-filter>
            <action android:name="com.google.firebase.MESSAGING_EVENT"/>
         </intent-filter>
      </service>
      <service
         android:name=".fcm.MyFirebaseInstanceIDService">
         <intent-filter>
            <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
         </intent-filter>
      </service>
   </application>


</manifest>
```

Santa Clara University

Center for Science, Technology, and Society