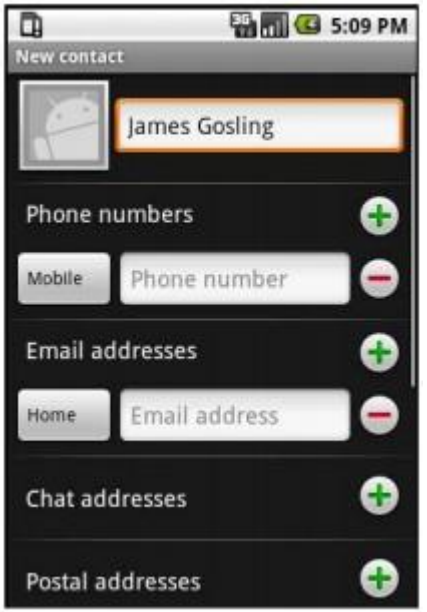
Activities And Intents

Activity Callbacks

- An ”**activity**” is an application component that provides a screen with which users can interact
- **Activity is usually a single screen**
  - Implemented as a single class extending Activity
  - Displays user interface controls (views)
  - Reacts on user input / events
- **An application typically consists of several activities**
  - Each screen is typically implemented by one activity
  - Each activity can then start another activity (new screen)
  - An activity may return a result to the previous activity



- "***main***" activity is presented to the user when launching the application for the first time.
- Each activity receives callbacks due to a change in its state during its ***lifecycle*** — whether the system is creating it, stopping it, resuming it, or destroying it.

Activity Callbacks

```
public class Activity extends ApplicationContext {
    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

// **onCreate** called when the activity is first created. Provides you with a Bundle containing the activity's previously frozen state, if there was one. Always followed by onStart().

// **onStart** called when the activity is becoming visible to the user. Followed by onResume() if the activity comes to the foreground, or onStop() if it becomes hidden.

// **onRestart** called after your activity has been stopped, prior to it being started again. Always followed by onStart()

// **onResume** called when the activity will start interacting with the user. At this point your activity is at the top of the activity stack, with user input going to it. Always followed by onPause().

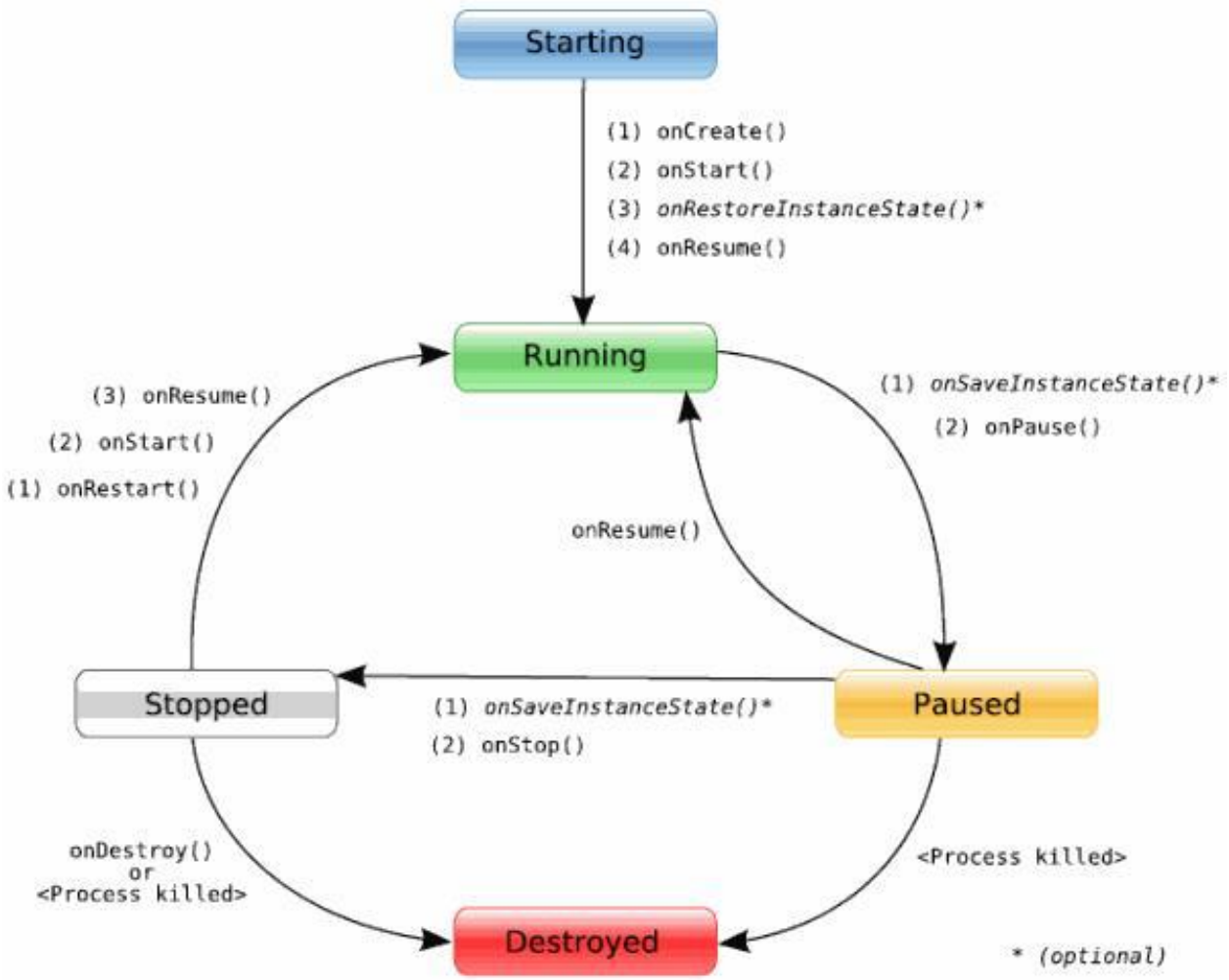
Activity Callbacks

// **onPause** called when the system is about to start resuming a previous activity. Implementations of this method must be very quick because the next activity will not be resumed until this method returns. Followed by either onResume() or onStop().

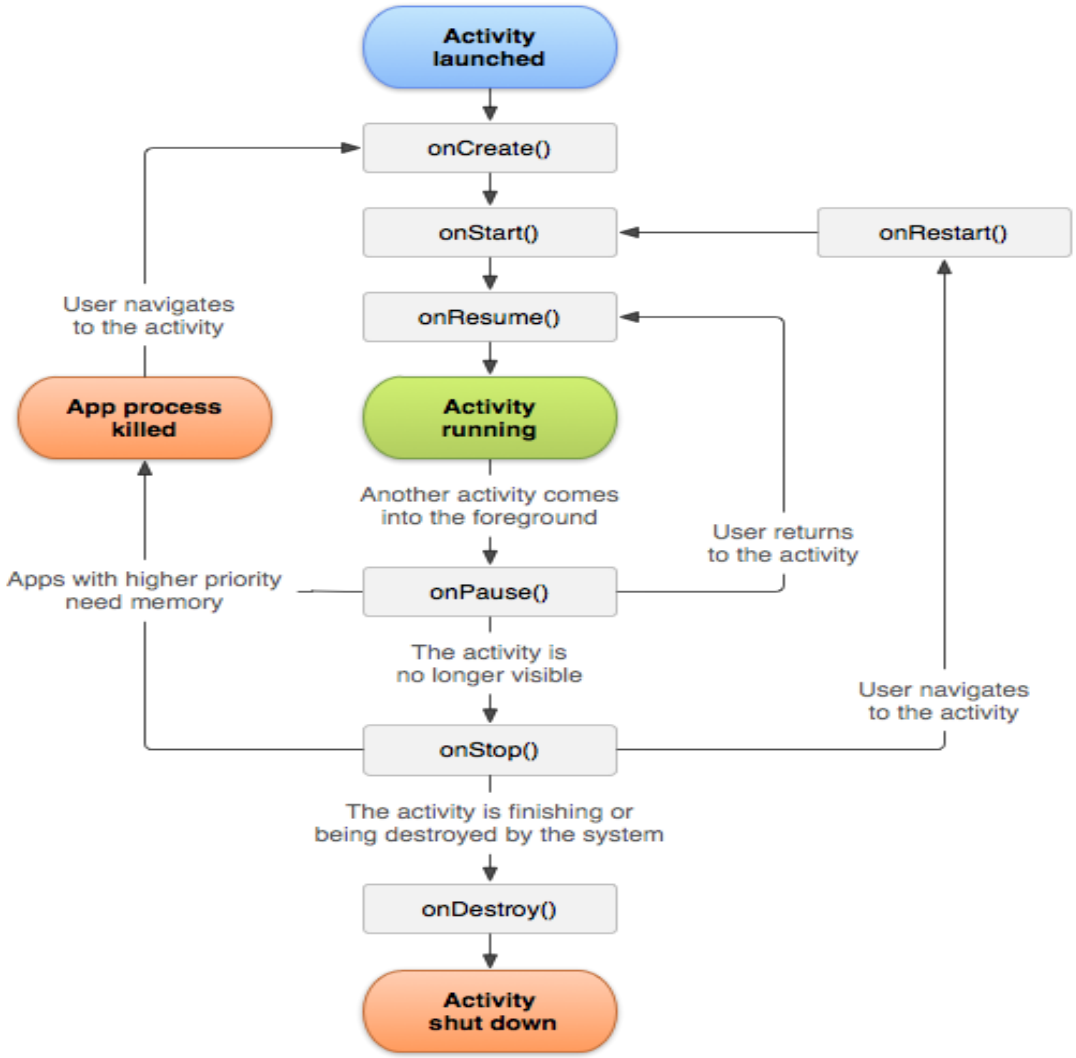
// **onStop** called when the activity is no longer visible to the user. Followed by either onRestart() if this activity is coming back to interact with user, oronDestroy() if this activity is going away.

// **onDestroy** is the final call you receive before your activity is destroyed. This can happen either because the activity is finishing (someone called finish() on it, or because the system is temporarily destroying this instance of the activity to save space. You can distinguish between these two scenarios with the isFinishing() method.

Activity Callbacks



Activity Callbacks



Utilizing Activity Lifecycle Functions

- Simple exercise to see the activity lifecycle in action
- each overridden function is explicit and a Toast command is added to show on screen when the function is entered
- Run it on an Android device and try various cases:
  - Changing the screen orientation destroys and recreates the activity from scratch.
  - Pressing the Home button pauses the activity, but does not destroy it.
  - Pressing the Application icon might start a new instance of the activity, even if the old one was not destroyed.
  - Letting the screen sleep pauses the activity and the screen awakening resumes it. (This is similar to taking an incoming phone call.)

```
package org.example.hello;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Toast;

public class Hello extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Toast.makeText(this, "onCreate", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onStart() {
        super.onStart();
        Toast.makeText(this, "onStart", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onResume() {
        super.onResume();
        Toast.makeText(this, "onResume", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Toast.makeText(this, "onRestart", Toast.LENGTH_SHORT).show();
    }

    @Override
    protected void onPause() {
        Toast.makeText(this, "onPause", Toast.LENGTH_SHORT).show();
        super.onPause();
    }

    @Override
    protected void onStop() {
        Toast.makeText(this, "onStop", Toast.LENGTH_SHORT).show();
        super.onStop();
    }

    @Override
    protected void onDestroy() {
        Toast.makeText(this, "onDestroy", Toast.LENGTH_SHORT).show();
        super.onDestroy();
    }
}
```

Activity Life Cycle Samples

- **Turn Display**

  > onCreate(null) -> onStart -> onResume() -> [Turn Display]

  > onSaveInstanceState() -> onPause() -> onStop() -> onDestroy()

  -> onCreate() -> onStart() -> onRestoreInstanceState() -> onResume()

- **Home**

  > onCreate(null) -> onStart -> onResume() -> [Home Button]

  > onSaveInstanceState() -> onPause() -> onStop() -> [Start App]

  > onRestart() -> onStart() -> onResume()

- **Phone Call Interrupt**

  > onCreate(null) -> onStart -> onResume() -> [Phone Call]

  > onSaveInstanceState() -> onPause() -> onStop() -> [Hang Up or press Back]

  > onRestart() -> onStart() -> onResume()

Notes on Activity

- **Starting Activity:**
  - void **startActivity** (Intent intent)
  - void **startActivityForResult** (Intent intent, int requestCode)
    - void **onActivityResult** (int requestCode, int resultCode, Intent data)
- **Shutting Down Activity:**
  - void **finish** ()
  - void **finishActivity** (int requestCode)
- **Saving Activity Status**
  - void **onSaveInstanceState** (Bundle outState)
  - void **onRestoreInstanceState** (Bundle savedInstanceState)
- **When Activity A starts Activity B:**
  - A's onPause()
  - B's onCreate(), onStart(), and onResume() (Activity B now has user focus.)
  - A's onStop() , if A is no longer visible

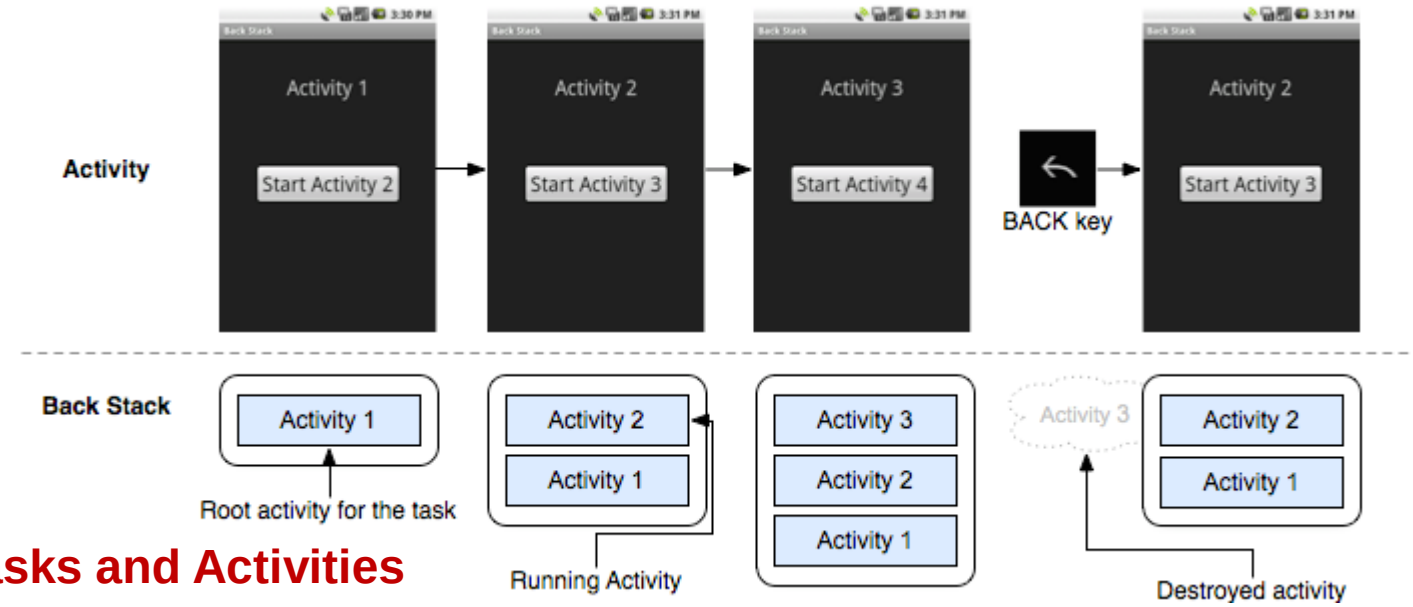
Starting Activity

```
public class MyActivity extends Activity {
    ...

    static final int PICK_CONTACT_REQUEST = 0;

    protected boolean onKeyDown(int keyCode, KeyEvent event) {
        if (keyCode == KeyEvent.KEYCODE_DPAD_CENTER) {
            // When the user center presses, let them pick a contact.
            startActivityForResult(
                new Intent(Intent.ACTION_PICK,
                new Uri("content://contacts")),
                PICK_CONTACT_REQUEST);
           return true;
        }
        return false;
    }

    protected void onActivityResult(int requestCode, int resultCode,
            Intent data) {
        if (requestCode == PICK_CONTACT_REQUEST) {
            if (resultCode == RESULT_OK) {
                // A contact was picked.  Here we will just display it
                // to the user.
                startActivity(new Intent(Intent.ACTION_VIEW, data));
            }
        }
    }
}
```

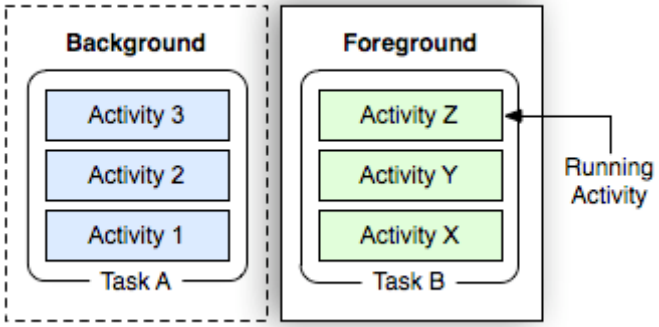
Saving Activity's State

```
public class CalendarActivity extends Activity {
    ...

    static final int DAY_VIEW_MODE = 0;
    static final int WEEK_VIEW_MODE = 1;

    private SharedPreferences mPrefs;
    private int mCurViewMode;

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        SharedPreferences mPrefs = getSharedPreferences();
        mCurViewMode = mPrefs.getInt("view_mode", DAY_VIEW_MODE);
    }

    protected void onPause() {
        super.onPause();

        SharedPreferences.Editor ed = mPrefs.edit();
        ed.putInt("view_mode", mCurViewMode);
        ed.commit();
    }
}
```

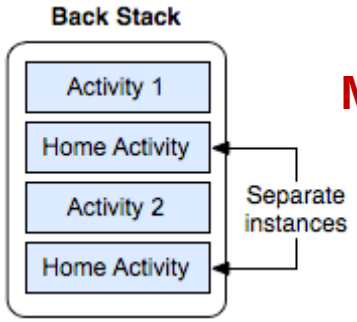
Back Stack



Tasks and Activities



Multiple Tasks



Multiple Activity Instances

Activities and AndroidManifest.xml

- Any new activity must be defined in AndroidManifest.xml file

HelloAndroid Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="org.example.hello"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="8" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Hello"
                  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>
```

Intents

Intents

- **Intent**
  - Intents are used as a message-passing mechanism within your application and between applications.
    - Explicit start Activity/Service using its class name
    - Start Activity/Service for an action with specific data
    - Broadcast that an event has occurred
  - Consists of
    - **Component name** (name of component/activity to handle the intent)
    - **Action to be performed** (MAIN / VIEW / EDIT / PICK / DELETE / …)
    - **Data to act on** (expressed as URI)
    - Categories
    - Extras (primitives / primitives / Strings / Serializable)
    - Flags

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("http://www.fhnw.ch"));
```

```
Intent intent = new Intent(Intent.ACTION_EDIT);
intent.setData(Uri.parse("content://contacts/people/1"));
```

Component Name Field

- **Specifies the name of the component (name of the activity if the component is activity) that should handle the intent**
  - Class name of the target component (for example ”FirstActivity")

```
<activity android:name=".FirstActivity"
        android:label="@string/app_name">
    <intent-filter>
        <action android:name=”edu.ck.examples.IntentActivity" />
        <category
android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

- **Setting component name is optional**
  - If it is set, the Intent object is delivered to an instance of the designated class.

Constructor: Intent(Context packageContext, Class<?> cls)

Example: `Intent intent = new Intent(this, FirstActivity.class);`

Explicit Intent

  - If it is not set, Android uses other information (i.e., action field) in the Intent object to locate a suitable target - this is called “intent resolution”

Constructor: Intent(String action)
Intent(String action, Uri uri)

Example: `Intent intent = new Intent(“edu.ck.examples.IntentActivity”);`

Implicit Intent

Intent Actions

- A string naming the action to be performed
- The Intent class defines a number of predefined action constants, including
  - ACTION_CALL, ACTION_EDIT, ACTION_MAIN, ACTION_SYNC, ACTION_BATTERY_LOW, etc.
- You can also define your own action strings for activating the components in your application
- The action largely determines how the rest of the intent is structured - particularly the data and extras fields - much as a method name determines a set of arguments and a return value.

Intent Actions

Constant	Target component	Action
ACTION_CALL	activity	Initiate a phone call.
ACTION_EDIT	activity	Display data for the user to edit.
ACTION_MAIN	activity	Start up as the initial activity of a task, with no data input and no returned output.
ACTION_SYNC	activity	Synchronize data on a server with data on the mobile device.
ACTION_BATTERY_LOW	broadcast receiver	A warning that the battery is low.
ACTION_HEADSET_PLUG	broadcast receiver	A headset has been plugged into the device, or unplugged from it.
ACTION_SCREEN_ON	broadcast receiver	The screen has been turned on.
ACTION_TIMEZONE_CHANGED	broadcast receiver	The setting for the time zone has changed.

Note that Actions' "**Constant**" could be replaced with an equivalent String constant.
e.g.,: **ACTION_CALL** = "android.intent.action.CALL"

Data Field

- The URI of the data to be acted on and the MIME type of that data.
- Different actions are paired with different kinds of data specifications.
  - If the action field is ACTION_EDIT, the data field would contain the URI of the document to be displayed for editing.
  - If the action is ACTION_CALL, the data field would be a ***tel:*** URI with the number to call.
  - If the action is ACTION_VIEW and the data field is an ***http:*** URI, the receiving activity would be called upon to download and display whatever data the URI refers to.
- Examples of Action/Data Pairs
  - ACTION_VIEW “content://contacts/people/”” //Display a list of people to browse through.
  - ACTION_VIEW “content://contacts/people/1” //Display information about person whose id is "1”.
  - ACTION_DIAL “content://contacts/people/1” //Display the phone dialer with the person filled in.
  - ACTION_VIEW “tel:123” //Display the phone dialer with the given number filled in.
  - ACTION_DIAL “tel:123” //Dial the phone dialer with the given number filled in.

Example of Activity Actions

- **Starting Activity:**

Launch new activity (without receiving a result)

- void **startActivity** (Intent intent)

```
Intent i = new Intent (this, OtherActivity.class);
startActivity(i);
```

Launch new activity and expect result

- void **startActivityForResult** (Intent intent, int requestCode)
  - When activity exits, onActivityResult() method is called with given requestCode (int > 0)
  - void **onActivityResult** (int requestCode, int resultCode, Intent data)

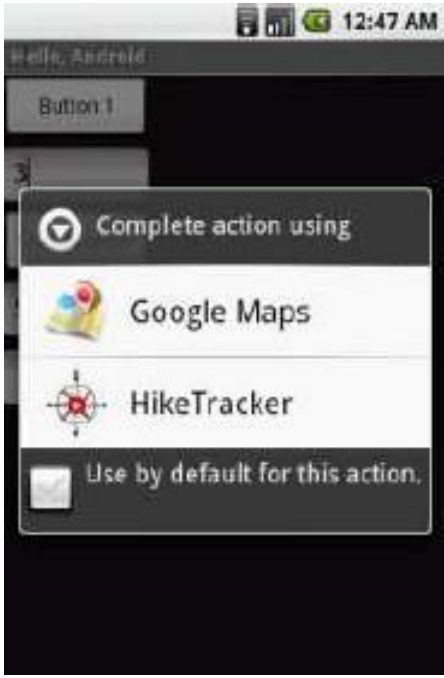
- **Shutting Down Activity:**
  - void **finish** ()

Intents and Intent Filters

- **Intent Filters**

  > Description of what intents an activity can handle

  > Activities publish their intent filters in a manifest file




```
<intent-filter >
    <action android:name="android.intent.action.VIEW"/>
    <category android:name="android.intent.category.DEFAULT"/>
    <category android:name="android.intent.category.BROWSABLE"/>
    <data android:scheme="geo"/>
 </intent-filter>
```

- Upon invocation of startActivity(intent) the system looks at the intent filters of all installed applications
  - Candidate activity that owns the filter must pass matching tests for action, category, and data fields

.MAIN Action & .LAUNCHER Category

- Activities that can initiate applications have filters with "android.intent.action.MAIN" specified as the action
  - This is a way an application gets started fresh, without a reference to any particular data.

- If they are to be represented in the application launcher, they also specify the "android.intent.category.LAUNCHER" category:

```
<intent-filter . . . >
   <action android:name="android.intent.action.MAIN" />
   <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

- The Android system populates the application launcher by finding all the activities with intent filters that specify the "android.intent.action.MAIN" action and "android.intent.category.LAUNCHER" category. It then displays the icons and labels of those activities in the launcher.