CS 580 Client-Server Programming
Fall Semester, 2012
Doc 6 Android Activity Life Cycle & Intents
Sept 13, 2012

Mars Comments

readLine

needs new line char(s)

Protocol does not have them

available

Seems to return 0 on sockets

Activity Life Cycle

Activity

Code that does some work

Single, focused thing that a user can do

Usually each screen(View) has its own activity

An application may have multiple screens, hence multiple activities

An application runs in its own Linux process

Activities can be viewless

Application

One or more screens (view)

Each screen has an activity

When go to new screen previous activity is stored on back stack

Back button

- Kills current activity
- Makes activity on top of back stack current

Home button

- Suspends current application
- Application and its activities just paused

Tasks

Sequence of activities the user follows to accomplish an objective

A user can

- Interrupt a task to start a new task
- Resume the first task where they left off

Tasks & Applications

Many applications are self contained

So task is sequence of activities from the application

Some applications use activities from other applications

Use phone
Show contacts
Use Web browser
Play music

So task is sequence of activities from multiple applications

Interrupting a Task

User presses Home and starts an application

Notifications

Activity Stack

Back Stack

History of activities used by user

May include activities of different applications

Back button

- Removes top of activity stack
- Makes next activity active

Home button

- Activity stack remains
- Starting another application starts new activity stack

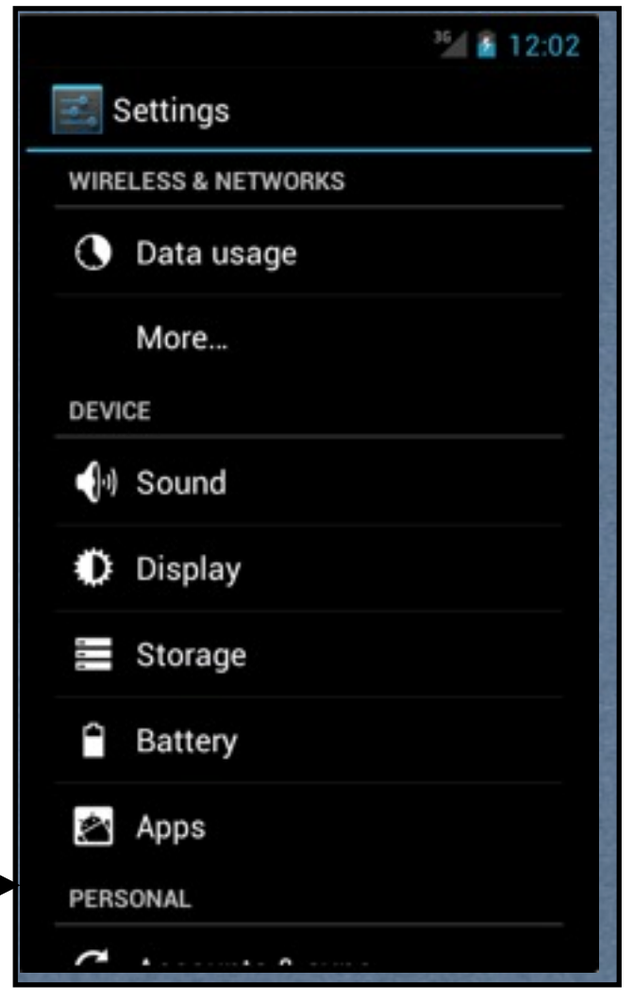
Stack only goes back to the start of the application at Home

Back Stack Example

Back Stack

Settings

Start Settings app

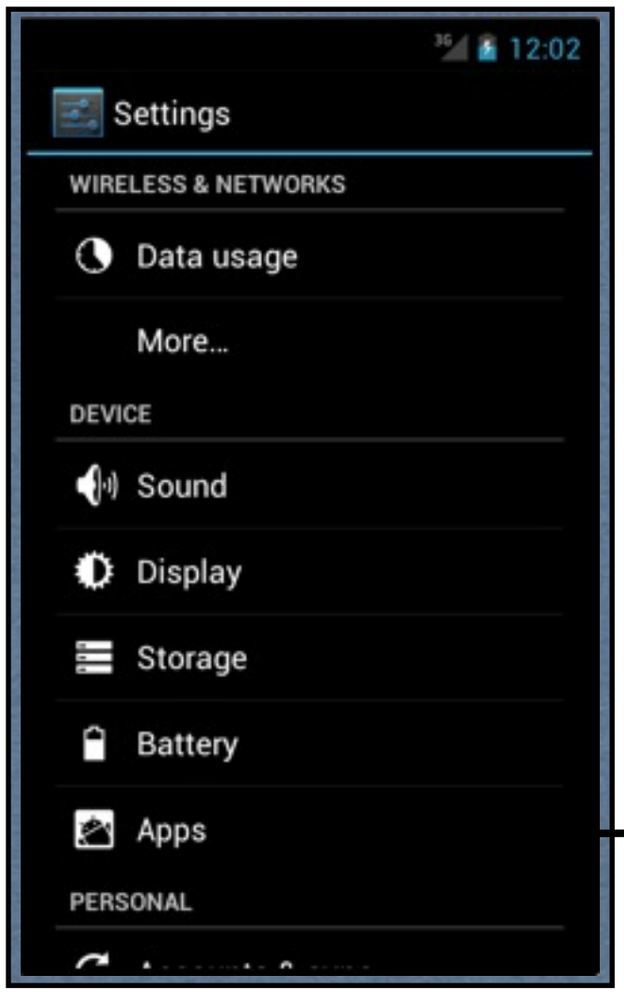


Back Stack Example

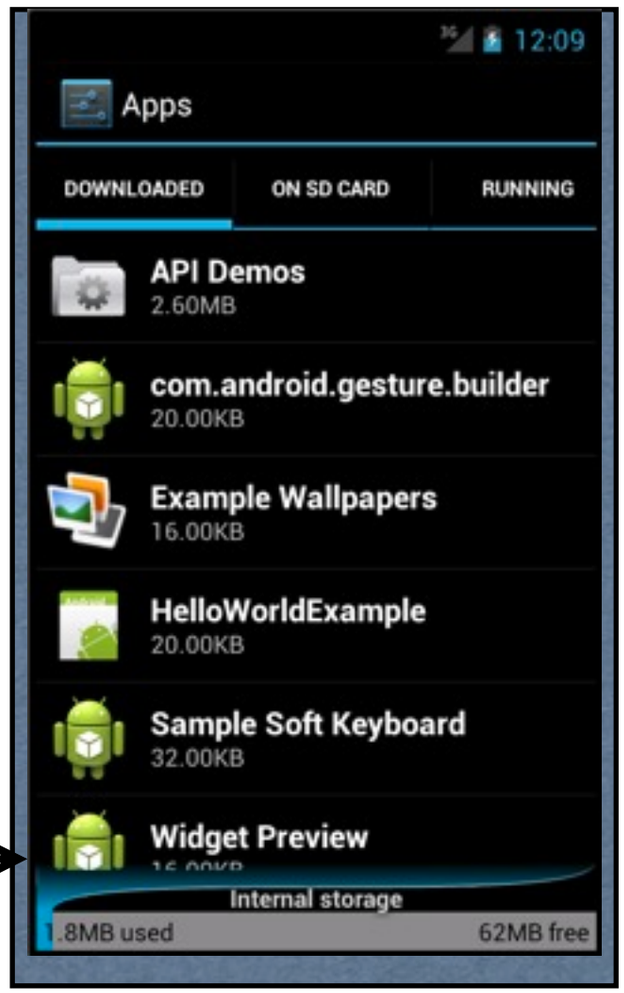
Settings

Back Stack

Apps
Settings



Apps activity

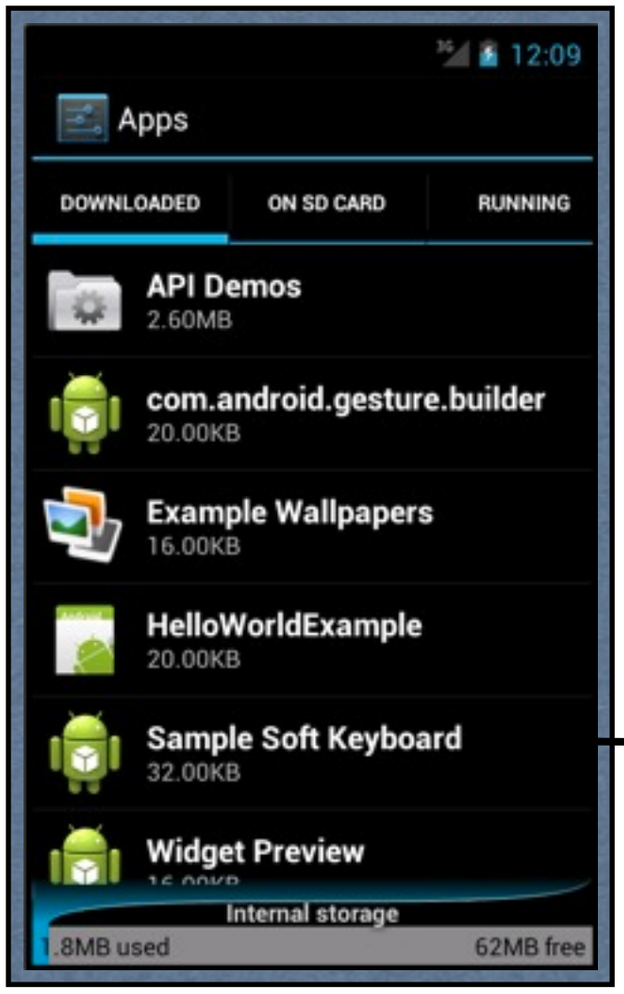


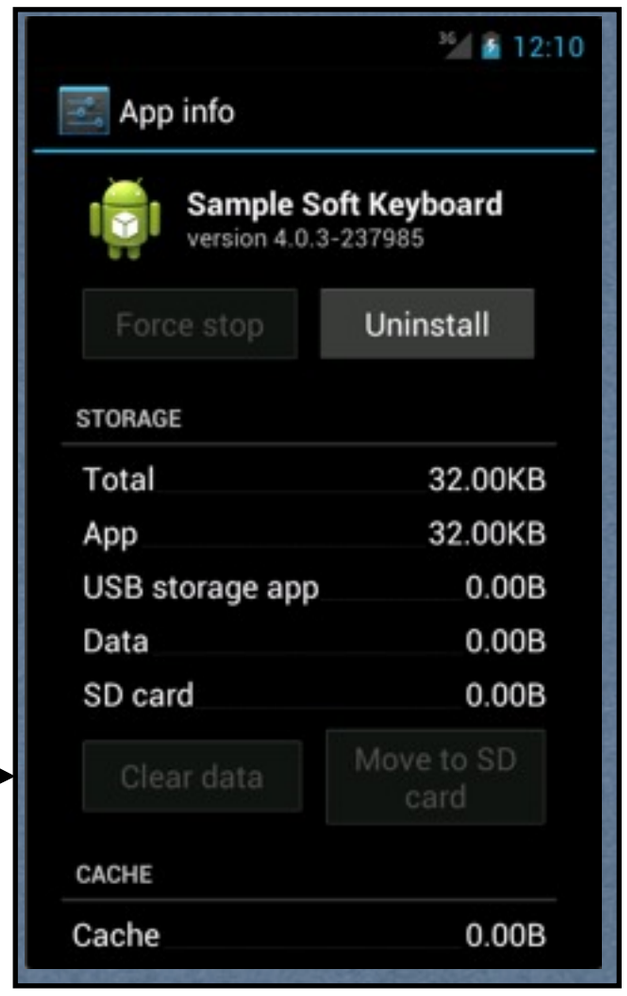
Back Stack Example

Apps
Settings

Back Stack

Soft Keyboard
Apps
Settings



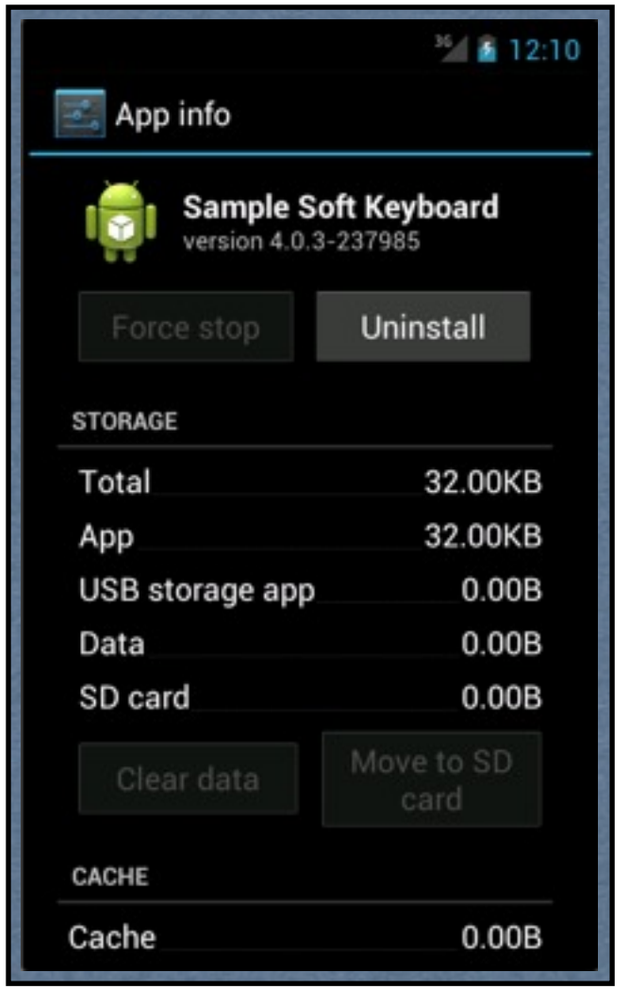


Back Stack Example - Back Button

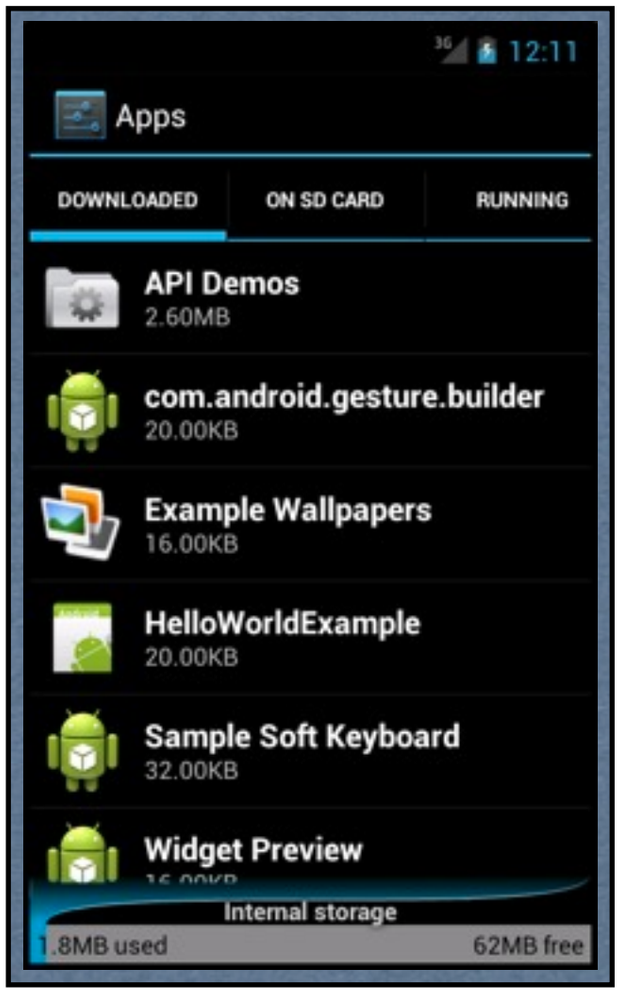
Soft Keyboard
Apps
Settings

Back Stack

Apps
Settings



Click back buttton

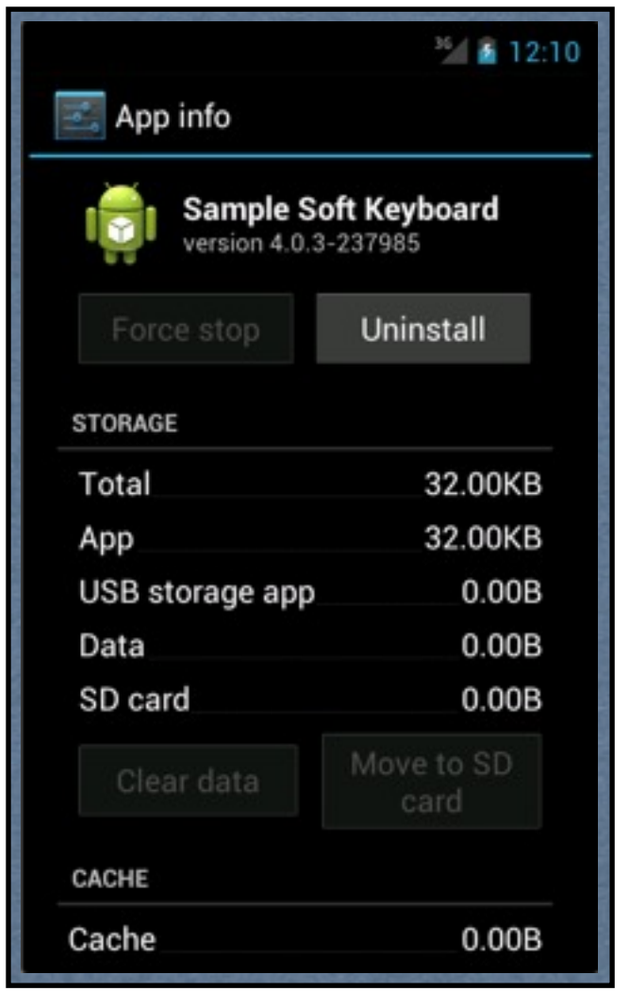


Back Stack Example - Home Button

Soft Keyboard
Apps
Settings

Back Stack

Soft Keyboard
Apps
Settings



Click home buttton

The home button brings you back to the home screen. A new back stack will be created if we start a new app. The old back stack remains. If the user restarts the old app then activity on top of its back stack is used, the app does not start from beginning.

Applications & Activity Stacks

Launching a non-running application

- Create new activity stack
- Put application's beginning activity on stack

Launching a running application

- Show activity on top of applications activity stack
- That activity may be from another application

Exceptions

- Some background activities return to their initial screen
  - Contacts & Gallery
- Some activities continue to run while in the background
  - Music player

See http://developer.android.com/guide/practices/ui_guidelines/activity_task_design.html for a complete description

Activity Lifecycle States

Active (Resumed)

- Running activity in foreground of screen

Paused

- Lost focus, but still visible
- Retains all state information
- In extreme memory situations may be killed

Stopped

- Not visible
- Retains all state information
- Often will be killed

How activities can be killed

Kill the app

All activities in app back stack are killed

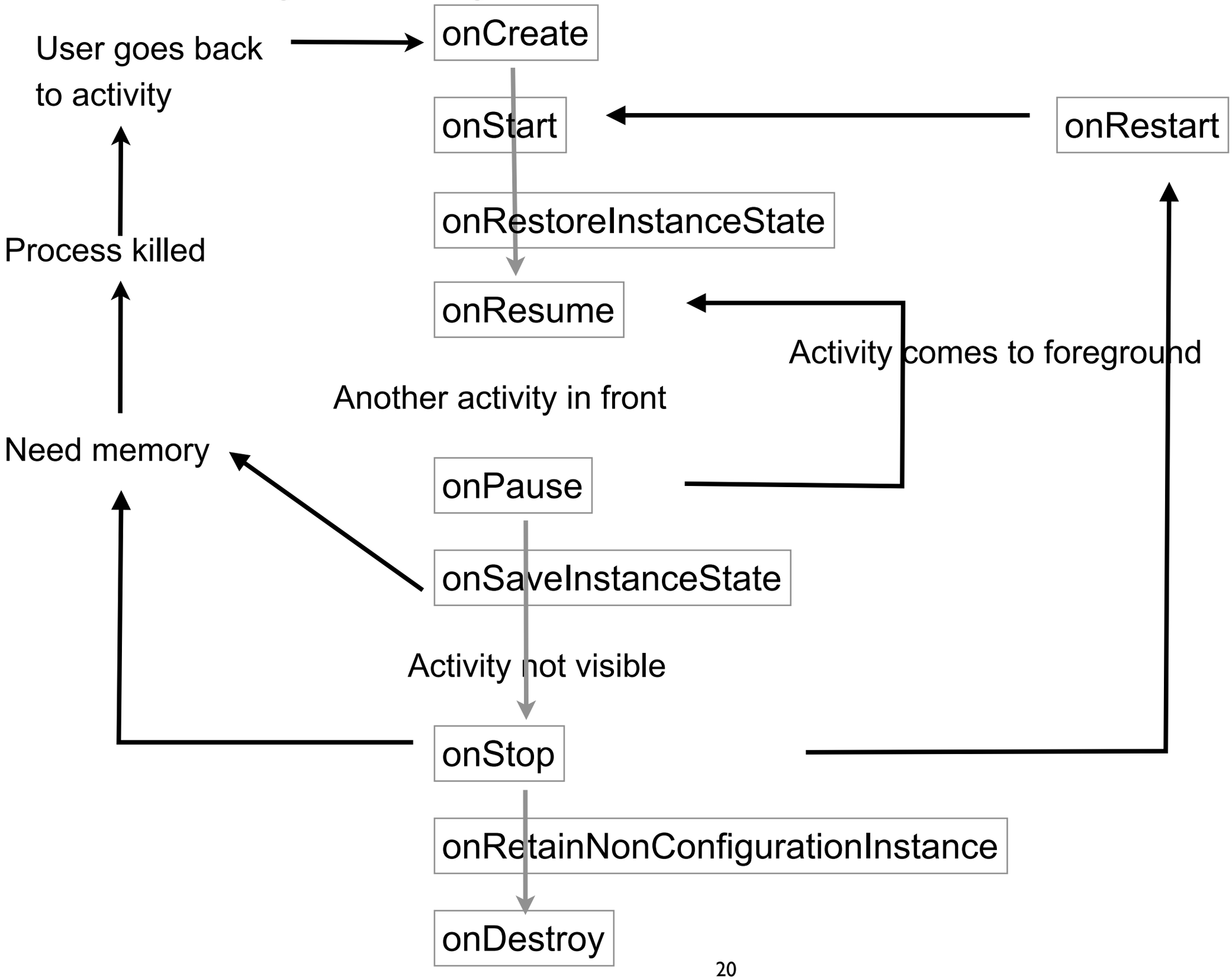
Back button

Current activity is killed

Lack of Memory

If run out of memory OS will kill activities in back stack

Activity Life Cycle Methods



Important Issue

If OS kills activity in back stack to reclaim memory

We have to insure activity

- Looks and acts the same
- When user goes back to the activity

Saving State

When low on memory system will kill activities

- In activity stack
- Not visible

When user goes back to killed activity

- Activity must appear as it did before it was killed

Must save state of activity

- System will save state of views

Types of State to Save

Dynamic instance state

- State of instance variables of activity
- Needed so activity object can operate

Persistent state

- Information that should be available next time application is run
- Contact information in Address book

Overlap

- Persistent state is usually subset of dynamic state

Saving Persistent State

Do it in the onPause() method

It will always be called

One method that will always be called before activity is killed

onStop)() and onDestroy() are not always called

onStop()

Called when activity is no longer visible

Not always called

onDestroy()

Used to free resources like threads

There are situations when
"system will simply kill the activity's hosting process
without calling this method"

Saving/Restoring Dynamic Instance State

protected void onSaveInstanceState(Bundle outState)

Called after onPause

Save data in bundle

Restore state in

onCreate or

onRestoreInstanceState

Intents

Intents - Calling Activities

Android application consists of multiple actvities

Activity represents one screen or view

Going from one screen to another

- Requires calling activity

Can't call new activity directly

- Use intent to indicate activity to start

Intent Resolution - Explicit Intents

Specify the component (class) an intent is to run

Common way to call your own code

```
Intent go = new Intent();
go.setClassName("edu.sdsu.cs696", "edu.sdsu.cs696.Hello");
startActivity(go);
```

Intent Resolution - Implicit Intents

Provide information about activity you want to run

display web page

System determines which component to run

If more than one activity can handle request
User is asked to select

Each activity declares in manifest what it can handle

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

Intent

Abstract description of an operation to be performed

Methods in android.content.Context	Action
startActivity(Intent) startActivityForResult(Intent, int)	Launch an Activity
sendBroadcast(Intent)	send it to any interested BroadcastReceiver components
startService(Intent) bindService(Intent, ServiceConnection, int)	

Intent Data

Primary Attributes

action
Action to be taken

data
Data to operation on as Uri

Secondary Attributes

category
additional information about the action to execute

type
Mime type of intent data

component
Explicit class to run

extras
Data for other component

http://code.google.com/android/reference/android/content/Intent.html

Implicit Intents

action

If given, must be listed by the component as one it handles.

String, which we can create

type

Retrieved from the Intent's data, if not already supplied in the Intent.

If given, must be listed by the component as one it handles

data that is not a content: URI and where no explicit type,

The scheme of the intent data (such as http: or mailto:) is considered

If given, must be listed by the component as one it handles

Categories

If given, all must be listed by the component as ones it handles

Intents Handled By Google Android Apps

Scheme	Action	Description
http://web_address https://web_address	VIEW	Open a browser window to the URL specified.
"" (empty string) http://web_address https://web_address	WEB_SEARCH	Opens the file at the location on the device in the browser.
tel: phone_number	CALL	Calls the entered phone number.
tel:phone_number voicemail:	DIAL	Dials but does not actually initiate the call the number given
geo:latitude,longitude geo:latitude,longitude?z=zoom geo:0,0?q=my+street+address geo:0,0?q=business+near+city	VIEW	Opens the Maps application to the given location

Intent Examples

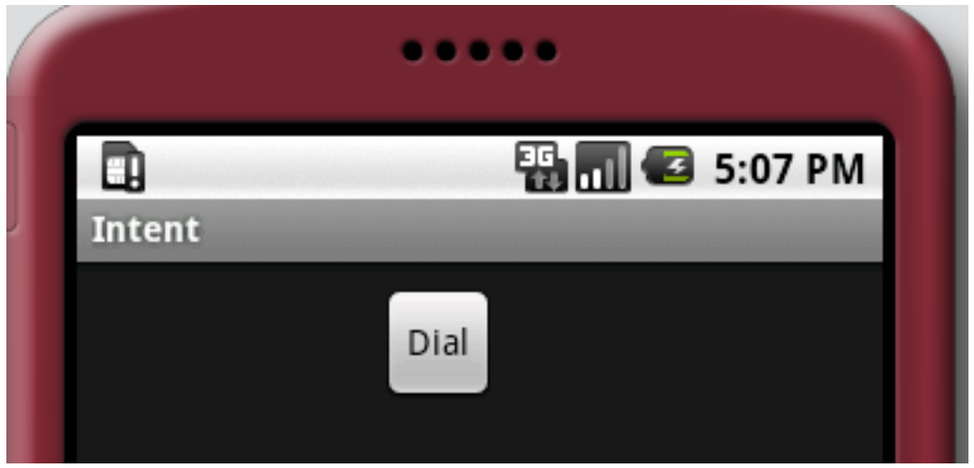
First Intent Example - Dial Phone

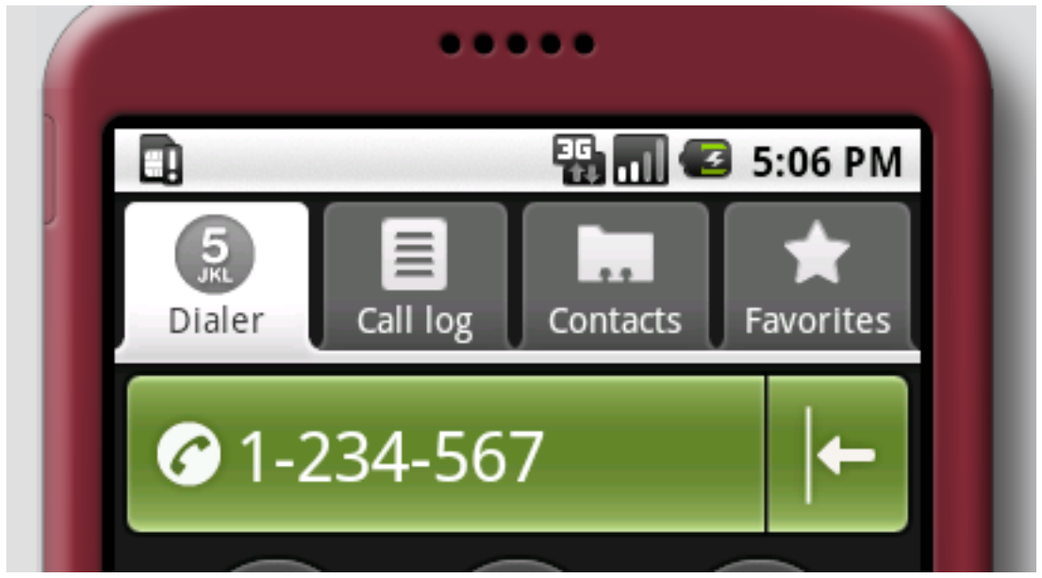
Activity with button

When button is pressed

- Phone activity is run
- Phone number is entered
- Phone number is hard coded

Implicit Intent to another application





IntentExample.java

```
public class IntentExample extends Activity implements View.OnClickListener {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.intent);
        Button ok = (Button) findViewById(R.id.go);
        ok.setOnClickListener(this);
    }

    public void onClick(View v) {
        Intent dial = new Intent();
        dial.setAction(android.content.Intent.ACTION_DIAL);
        dial.setData(Uri.parse("tel:1234567"));
        startActivity(dial);
    }
}
```

Other Ways to create the Intent

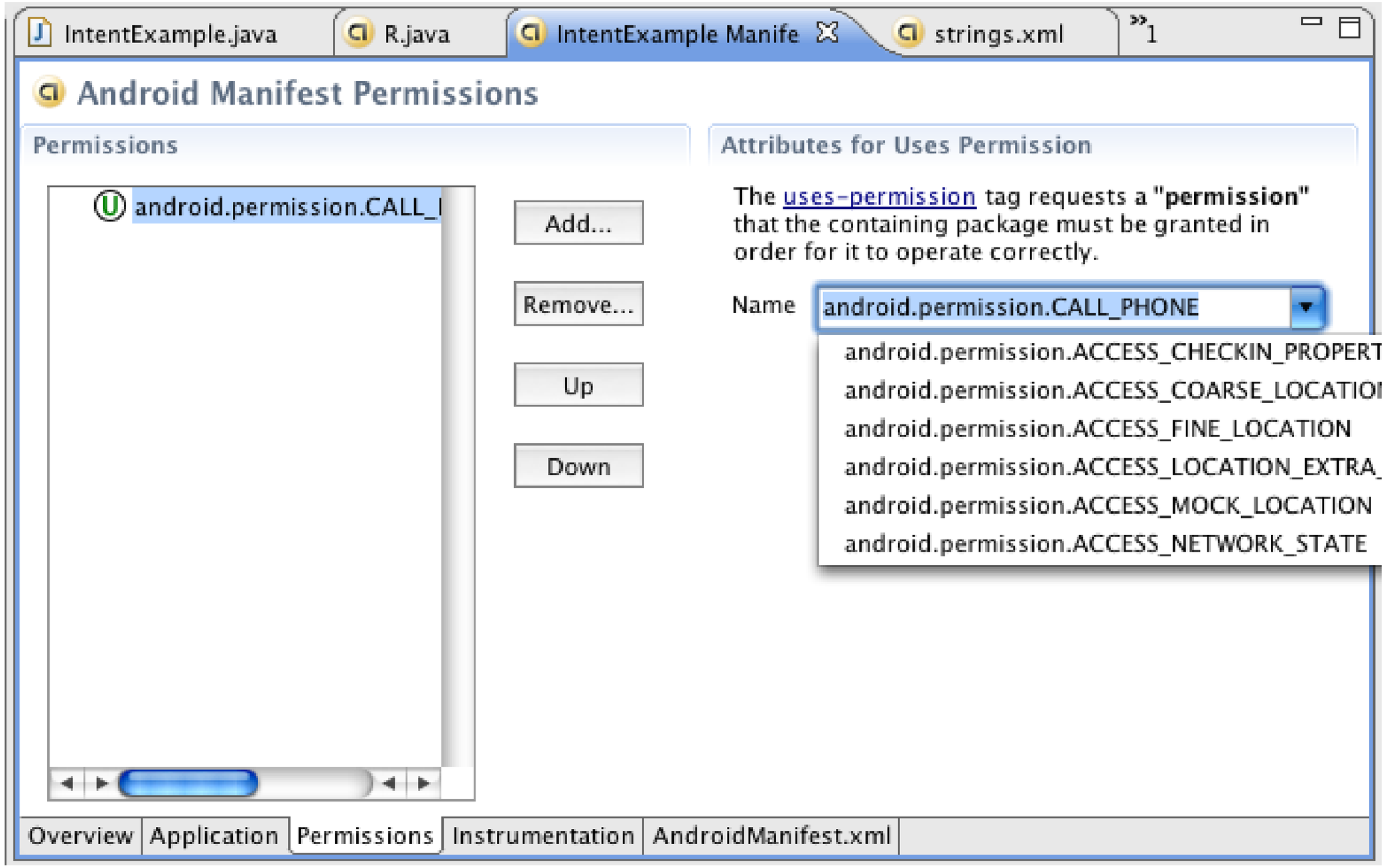
```
 public void onClick(View v) {
            Intent dial = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:1234567"));
     startActivity(dial);
}
```

IntentExamples Manifest

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="edu.sdsu.cs696"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".IntentExample"
                  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>
    <uses-sdk android:minSdkVersion="2" />
      <uses-permission android:name="android.permission.CALL_PHONE"></uses-
permission>
</manifest>
```


To use the phone you need to include the uses-permission in the manifest

Adding the Permission



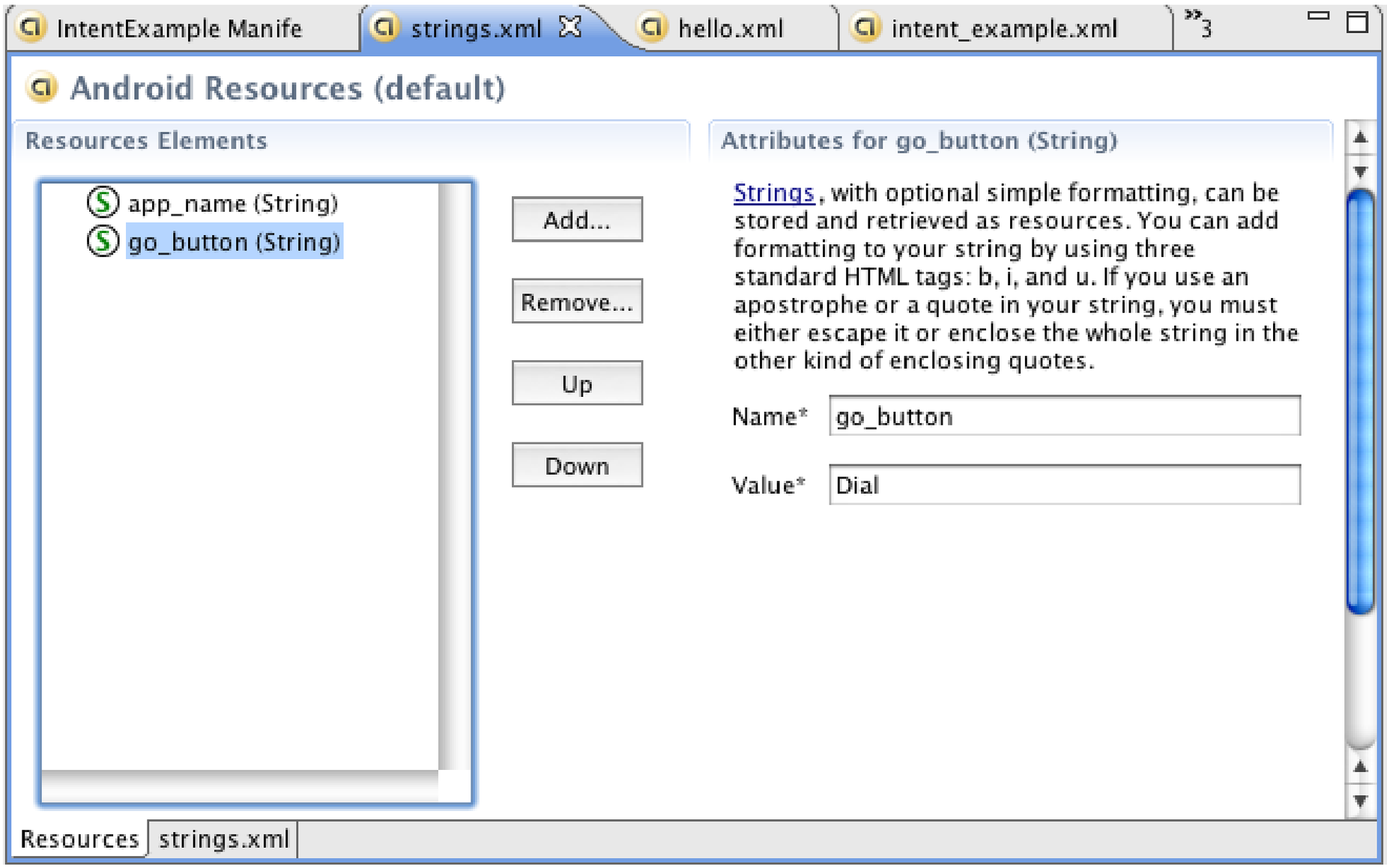
intent.xml

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:id="@+id/layout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<Button
        android:id="@+id/go"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/go_button"
        android:gravity="center"
        android:layout_x="120px"
        android:layout_y="10px"
>
</Button>
</AbsoluteLayout>
```

strings.xml

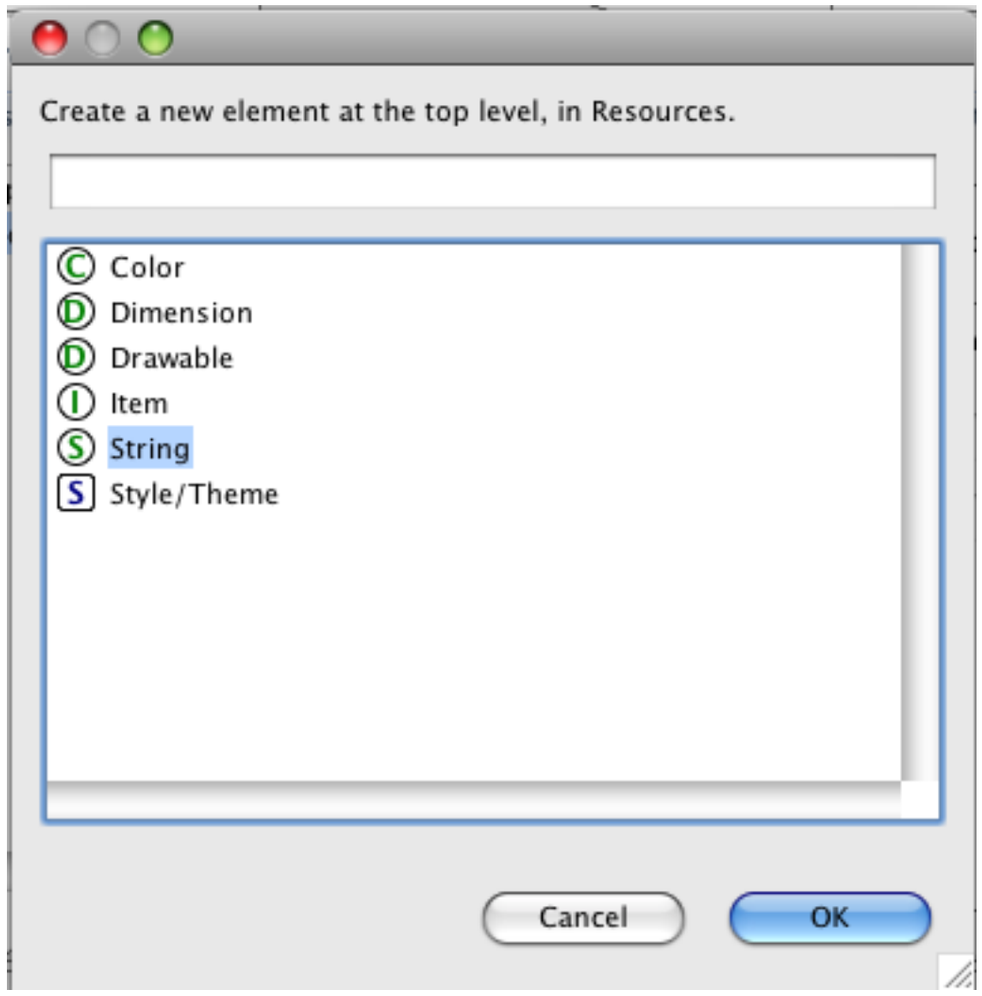
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, IntentExample!</string>
    <string name="app_name">Intent</string>
      <string name="go_button">Dial</string>
</resources>
```

Editing Resources



Add a new String

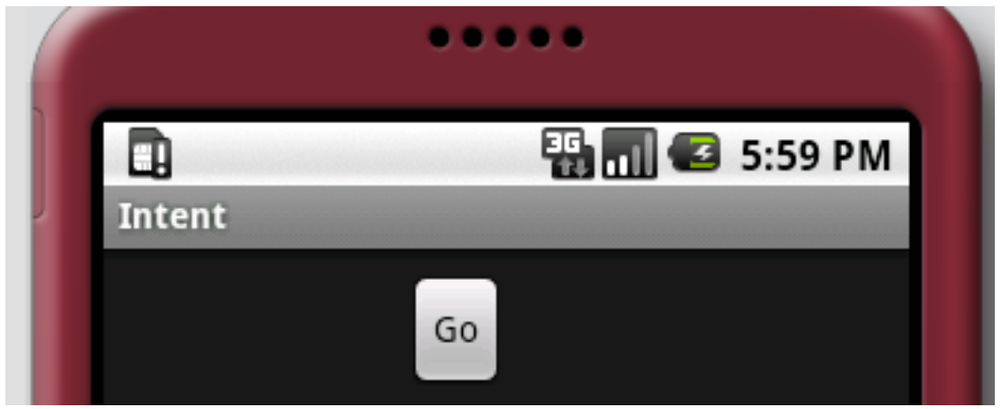
Click on the "Add" button in the Resource view of Strings.xml
(See previous slide)



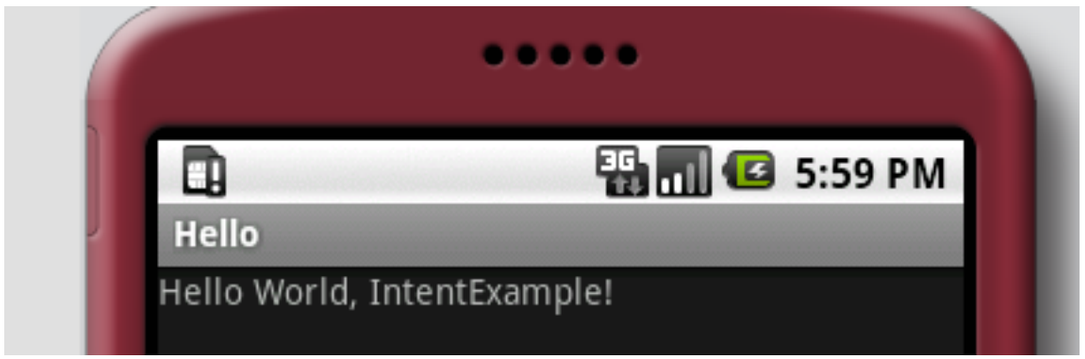
Class Name Intent Example

Button click will call another activity in same application
Will call the class directly
Useful when activity is private to application

IntentExample.java



Hello.java



Clicking the "Go" button will launch the Hello activity

IntentExample.java

```
package edu.sdsu.cs696;

public class IntentExample extends Activity implements View.OnClickListener {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.intent);
        Button ok = (Button) findViewById(R.id.go);
        ok.setOnClickListener(this);
    }

    public void onClick(View v) {
        Intent go = new Intent();
        go.setClassName("edu.sdsu.cs696", "edu.sdsu.cs696.Hello");
        startActivity(go);
    }
                                                    //Full class name required
}
```

Hello.java

```
package edu.sdsu.cs696;

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

public class Hello extends Activity {
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.hello);
        }
}
```

AndroidManifest.xml

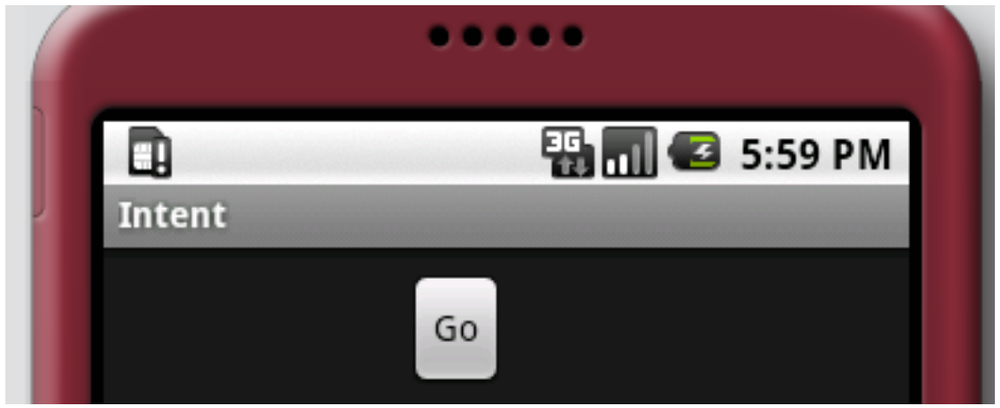
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="edu.sdsu.cs696"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".IntentExample"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:label="Hello" android:name="Hello">
            </activity>
    </application>
</manifest>
```


An activity has to be listed in the manifest before it can be run

Implicit Example - Intent Filter

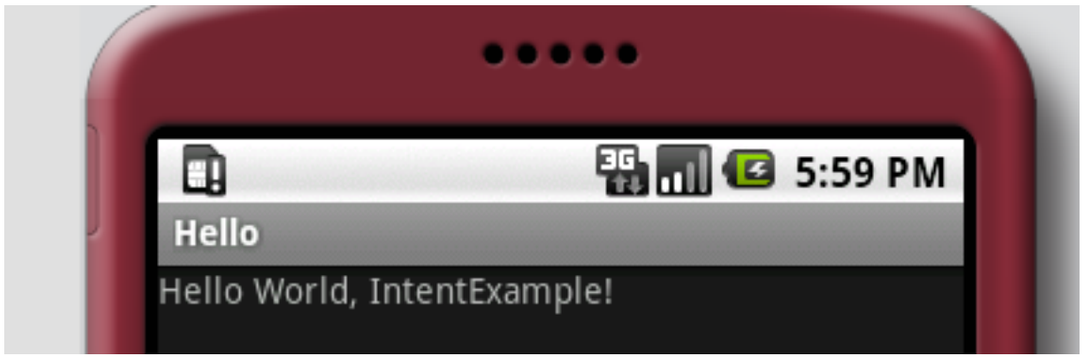
Button click will call another component using intent filters
Indirect access of activity

IntentExample.java



Clicking the "Go" button will launch the Hello activity

Hello.java



IntentExample

```
public class IntentExample extends Activity implements View.OnClickListener {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.intent_example);
        Button ok = (Button) findViewById(R.id.go);
        ok.setOnClickListener(this);
    }

    public void onClick(View v) {
        Intent go = new Intent();
        go.setAction("cs696.sender.add");
        startActivity(go);
    }
}
```

Hello.java

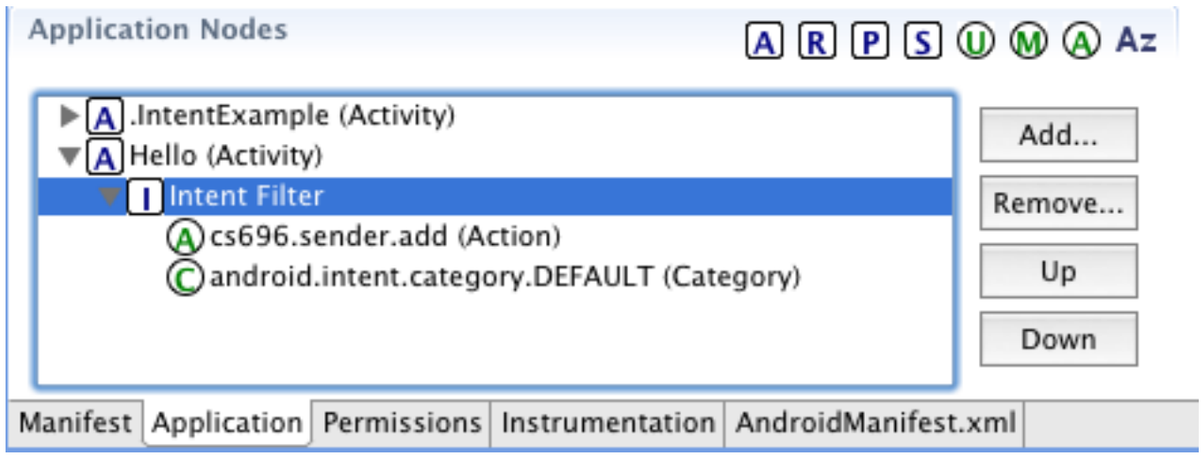
```
package edu.sdsu.cs696;

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

public class Hello extends Activity {
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.hello);
        }
}
```

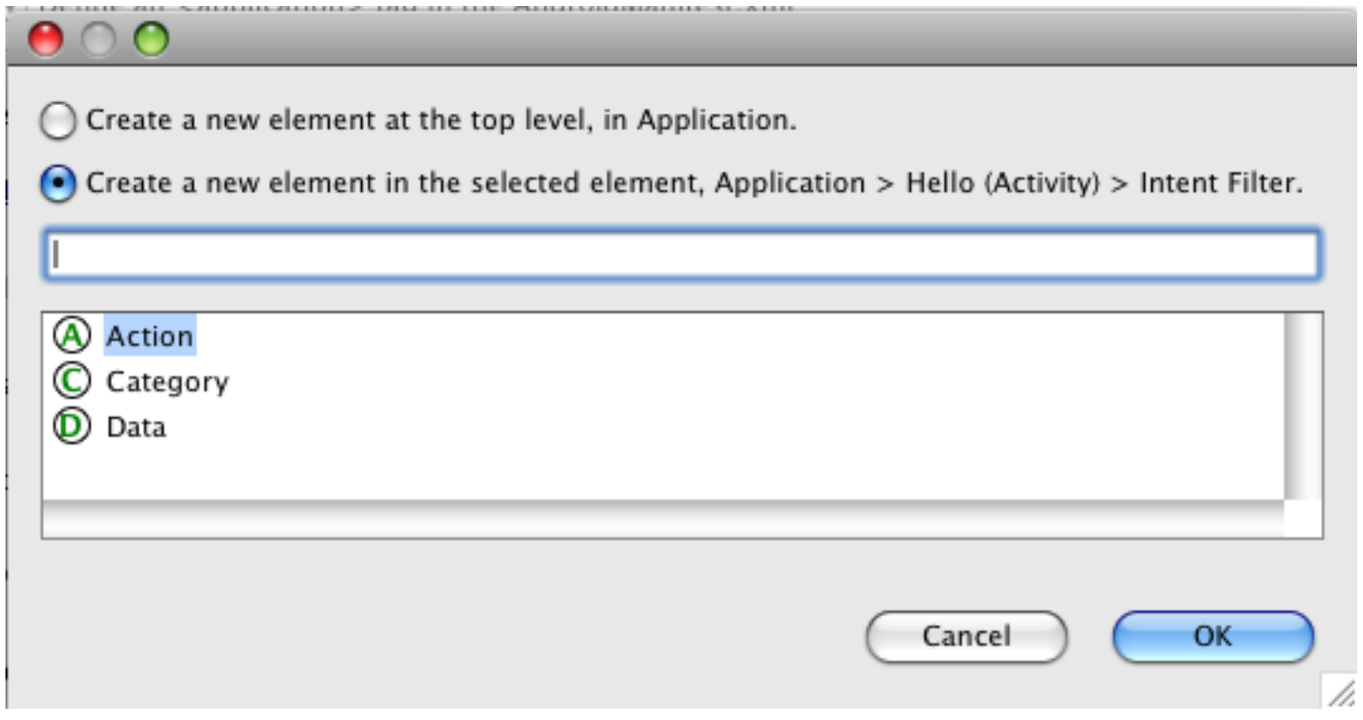
AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="edu.sdsu.cs696"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".IntentExample"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:label="Hello" android:name="Hello">
        <intent-filter>
                        <action android:name="cs696.sender.add"></action>
                        <category android:name="android.intent.category.DEFAULT">
                        </category>
                    </intent-filter>
                </activity>
    </application>
</manifest>
```


The default category is needed. As far I can tell you always need the default category when doing implicit intents.

Adding Categories etc

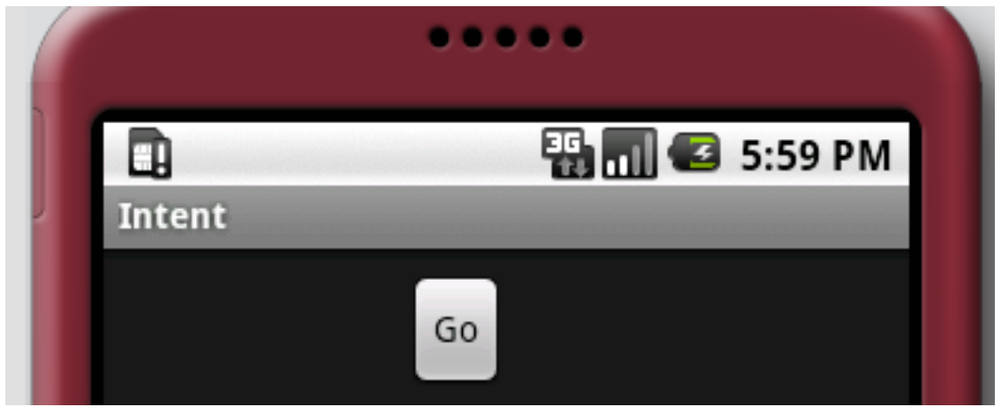




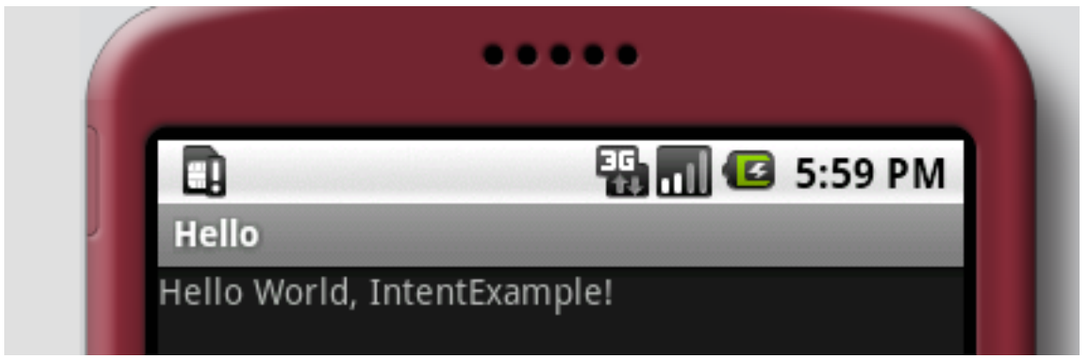
Implicit Example - Filter with Categories

Button click will call another component

IntentExample.java



Hello.java



Clicking the "Go" button will launch the Hello activity

Category Example

```
public class IntentExample extends Activity implements View.OnClickListener {

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.intent);
        Button ok = (Button) findViewById(R.id.go);
        ok.setOnClickListener(this);
    }

    public void onClick(View v) {
        Intent go = new Intent();
        go.setAction("cs696.sender.add");
        go.addCategory("foo");
        startActivity(go);
    }
}
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="edu.sdsu.cs696"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".IntentExample"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:label="Hello" android:name="Hello">
        <intent-filter>
                        <action android:name="cs696.sender.add"></action>
                        <category android:name="android.intent.category.DEFAULT">
                        </category>
                        <category android:name="foo"></category>
                    </intent-filter>
                </activity>
    </application>
</manifest>
```

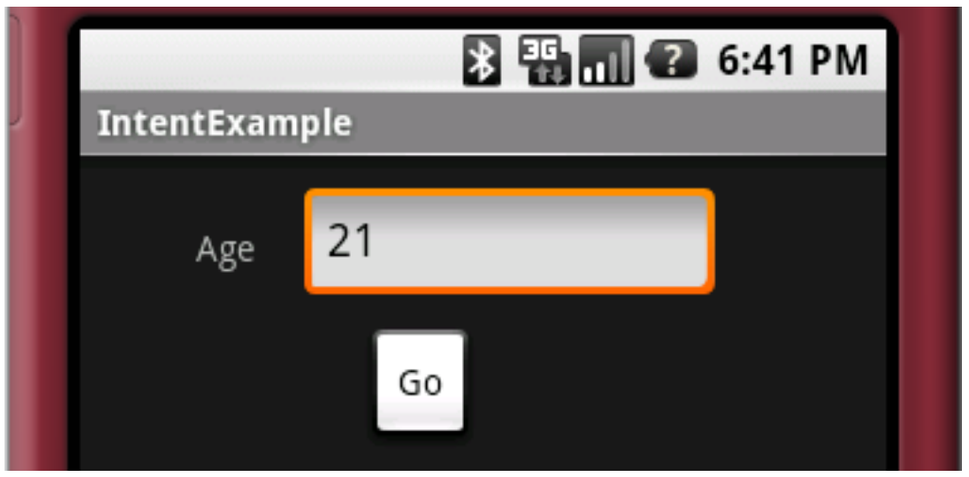
Some Application Settings

Name*	.IntentExample	Browse...
Theme		Browse...
Label	@string/app_name	Browse...
Icon		Browse...
Launch mode		
Screen orientation		
Config changes		Select...
Permission		
Multiprocess		
Process		Browse...
Task affinity		Browse...
Allow task reparenting		
Finish on task launch		
Clear task on launch		
Always retain task state		
State not needed		
Exclude from recents		
Enabled		

There is a lot more to learn

Intent - Passing Data

IntentExample

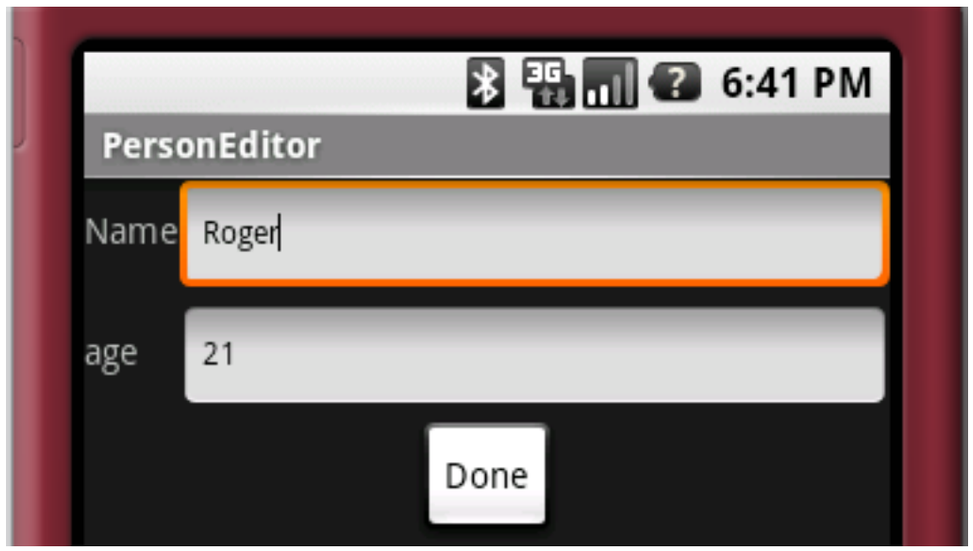


Displays/Edits age

Go button

- Calls PersonEditor
- Passes data
  - Name
  - Age

PersonEditor



Displays/Edits Name and age

Done button

- Returns edited data back
- Age = 0 cancels edit

IntentExample.java

```
public class IntentExample extends Activity implements View.OnClickListener {
    private EditText numberText;
    private static final int INTENT_EXAMPLE_REQUEST = 123;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.intent);
        Button ok = (Button) findViewById(R.id.go);
        ok.setOnClickListener(this);
        numberText = (EditText) this.findViewById(R.id.number);
        numberText.setText("21");

}
```


When we want a reply back from an Intent request we supply a request number. The request number is return with the answer. That way it is possible to know where the request originated from.

IntentExample.Java continued

Sending the data to PersonEditor

```
public void onClick(View v) {
        Intent go;
        go = new Intent();
        go.setAction("android.intent.action.EDIT");
        go.addCategory("person_editor");
        String newAge = numberText.getText().toString();
        go.putExtra("age", newAge);
        go.putExtra("name", "Roger");
        startActivityForResult(go, INTENT_EXAMPLE_REQUEST);
}
```


The name was sent just to show we can send multiple items. They can be of any base type or serializable. See the putExtra methods in the Intent class.

IntentExample.Java continued

Getting the Results back

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if (requestCode != INTENT_EXAMPLE_REQUEST) {
        numberText.setText("Not from me");
        return;
    }
    switch (resultCode) {
    case RESULT_OK:
        String editedAge = data.getStringExtra("age");
        numberText.setText(editedAge);
        break;
    case RESULT_CANCELED:
        numberText.setText("Cancelled");
        break;
    }
  }
}
```


When an activity is started after a return from a startActivityForResult request, the method onActivityResult is called. The requestCode is the request code sent in the startActivityForResult method. The result code is the set by the called activity and is RESULT_OK or RESULT_CANCELED by convention. The intent is used to send data back to this activity. RESULT_OK and RESULT_CANCELED are constants defined in the Activity class.

PersonEditor.java

```
public class PersonEditor extends Activity implements View.OnClickListener {
    private EditText ageText;
      private EditText nameText;

      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.person_editor);
            Button done = (Button) findViewById(R.id.edit_done);
            done.setOnClickListener(this);
            ageText = (EditText) this.findViewById(R.id.edit_age);
            nameText = (EditText) this.findViewById(R.id.edit_name);
            Bundle personData = getIntent().getExtras();
            String age = personData.getString("age");
            String name = personData.getString("name");
            if ((age != null) && (name != null)) {
                  ageText.setText(age);
                  nameText.setText(name);
            }
}
```


Showing how to access the intent that started the activity and extracting the extras from the intent. If the key-value pair was not set in the intent the value will be returned as null.

PersonEditor.java

Returning the data

```
public void onClick(View v) {
        String newAge = ageText.getText().toString();
        Intent result = getIntent();
        result.putExtra("age", newAge);
        if (newAge.equals("0"))
            setResult(RESULT_CANCELED, result);
        else
            setResult(RESULT_OK, result);
        finish();
}
```


setResult(int, Intent) returns information to the calling activity. The first parameter is the result code passed back in onActivityResult. The intent is the intent passed back in onActivityResult.

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="edu.sdsu.cs683.example" android:versionCode="1"
      android:versionName="1.0.0">
      <application android:icon="@drawable/icon" android:label="@string/app_name">
            <activity android:name=".IntentExample" android:label="@string/app_name">
                  <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                        <category android:name="android.intent.category.LAUNCHER" />
                  </intent-filter>
            </activity>
            <activity android:label="PersonEditor" android:name="PersonEditor">
                  <intent-filter>
                        <action android:name="android.intent.action.EDIT"></action>
                        <category android:name="person_editor"></category>
                        <category android:name="android.intent.category.DEFAULT">
                   </category>
                  </intent-filter>
            </activity>
      </application>
</manifest>
```


The intent filter for the activity must contain all the categories used by the intent to select the activity. It can contain more categories. The example does not work without the default category.