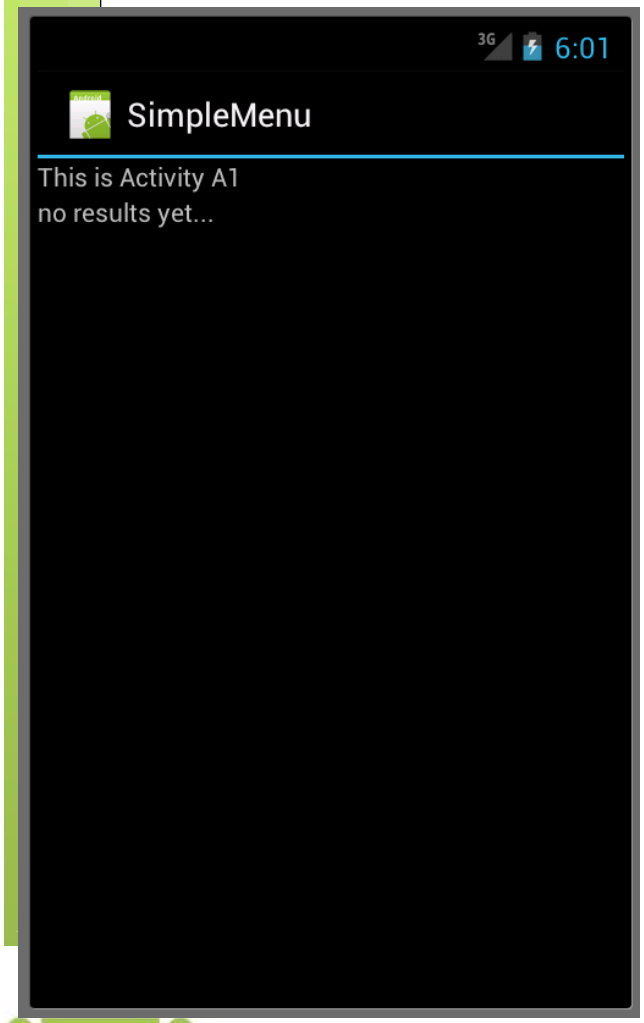
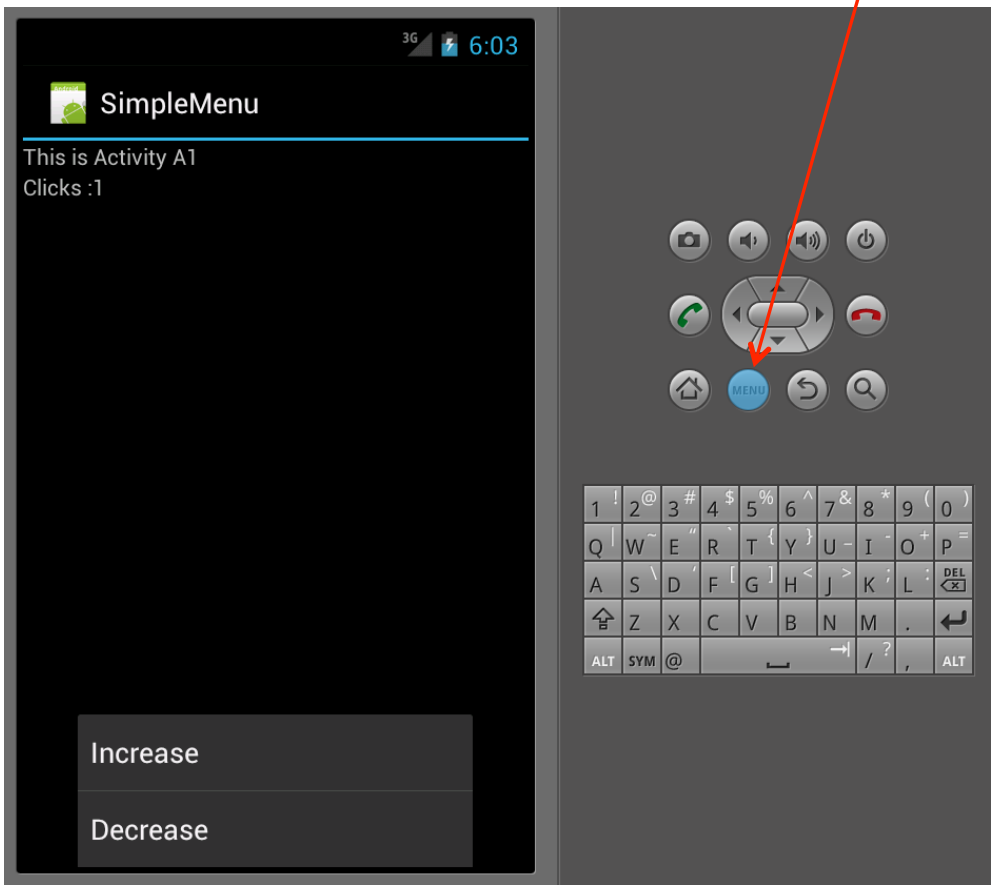
Basic UI elements: Android Menus (basics)

Marco Ronchetti
Università degli Studi di Trento

SimpleMenu





Layout & Strings

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:orientation="vertical" >
   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello" />
<TextView
      android:id="@+id/tf1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/output" />
</LinearLayout>
```

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello">This is Activity A1</string>
   <string name="app_name">SimpleMenu</string>
   <string name="output">no results yet...</string>
</resources>
```

SimpleMenu – A1

```
public class A1 extends Activity {
   int nClicks=0;
   protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.layout1);
   }
   public boolean onCreateOptionsMenu(Menu menu){
        super.onCreateOptionsMenu(menu);
        int base=Menu.FIRST;
        MenuItem item1=menu.add(base,1,1,"Increase");
        MenuItem item2=menu.add(base,2,2,"Decrease");
        return true;
   }
   public boolean onOptionsItemSelected(MenuItem item) {
        TextView tf = (TextView) findViewById(R.id.tf1);
        if (item.getItemId()==1) increase();
        else if (item.getItemId()==2) decrease();
        else return super.onOptionsItemSelected(item);
         tf.setText("Clicks :"+nClicks);
         return true;
    }
```

```
package it.unitn.science.latemar;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

Menu is created

Respond to a Menu event

```
private void increase() {
                    nClicks++;
          }
          private void decrease() {
                    nClicks--;
          }
}
```

SimpleMenu – A1

```
public class A1 extends Activity {
    int nClicks=0;
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.layout1);
    }
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        int base=Menu.FIRST;
        MenuItem item1=menu.add(base,1,1,"Increase");
        MenuItem item2=menu.add(base,2,2,"Decrease");
        return true;
    }
    public boolean onOptionsItemSelected(MenuItem item) {
        TextView tf = (TextView) findViewById(R.id.tf1);
        if (item.getItemId()==1) increase();
        else if (item.getItemId()==2) decrease();
        else return super.onOptionsItemSelected(item);
         tf.setText("Clicks :"+nClicks);
         return true;
     }
```

ID

Group

Order

Menu is created

This could be a resource

Respond to a Menu event

```
package it.unitn.science.latemar;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

```
private void increase() {
                nClicks++;
        }
        private void decrease() {
                nClicks--;
        }
}
```

Calling Activities in other apps: Android Intents

Marco Ronchetti
Università degli Studi di Trento

Re-using activities

When you create an application, you can assemble it from

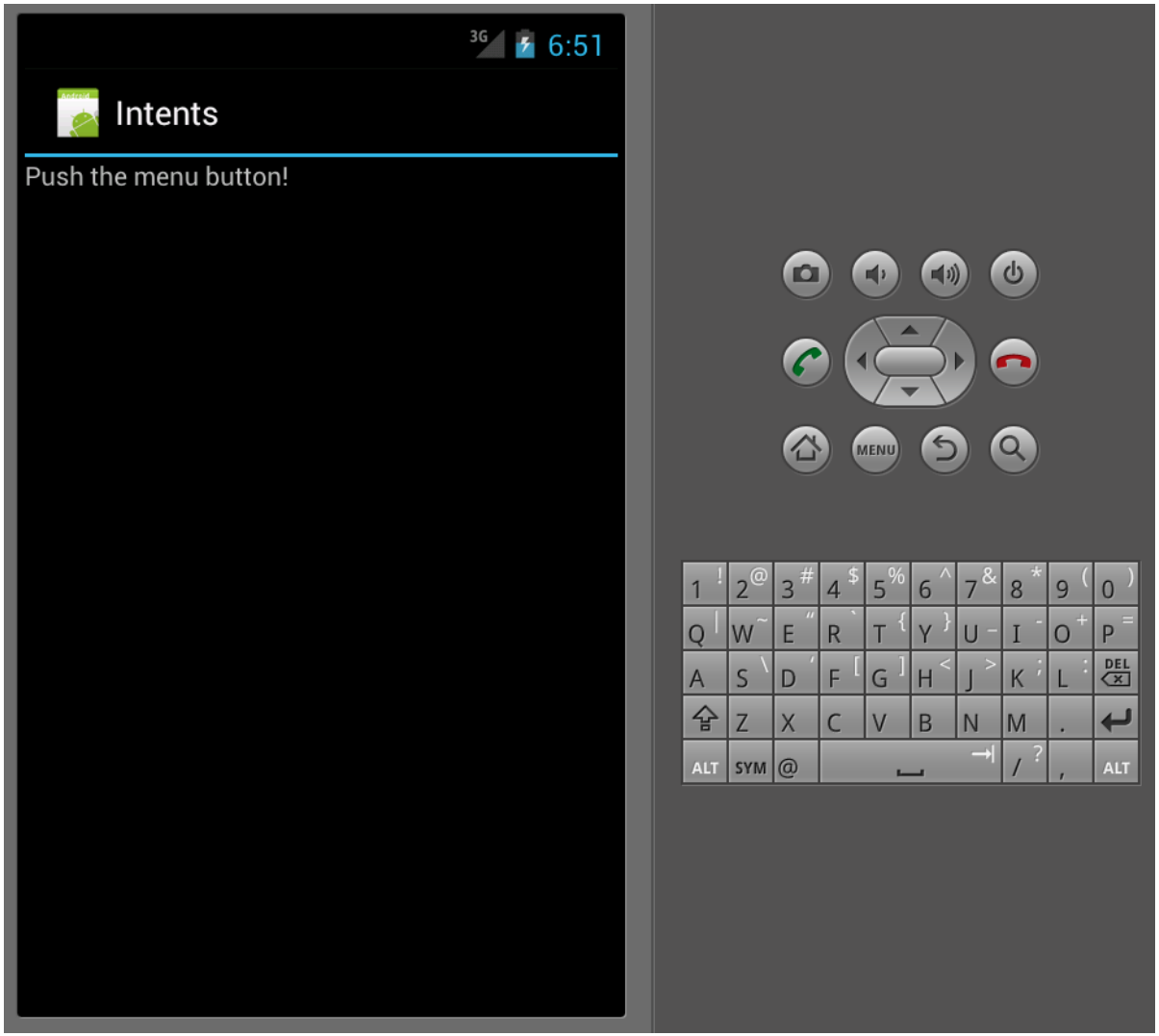
- activities that you create
- activities you re-use from other applications.

An app can incorporate activities from other apps.

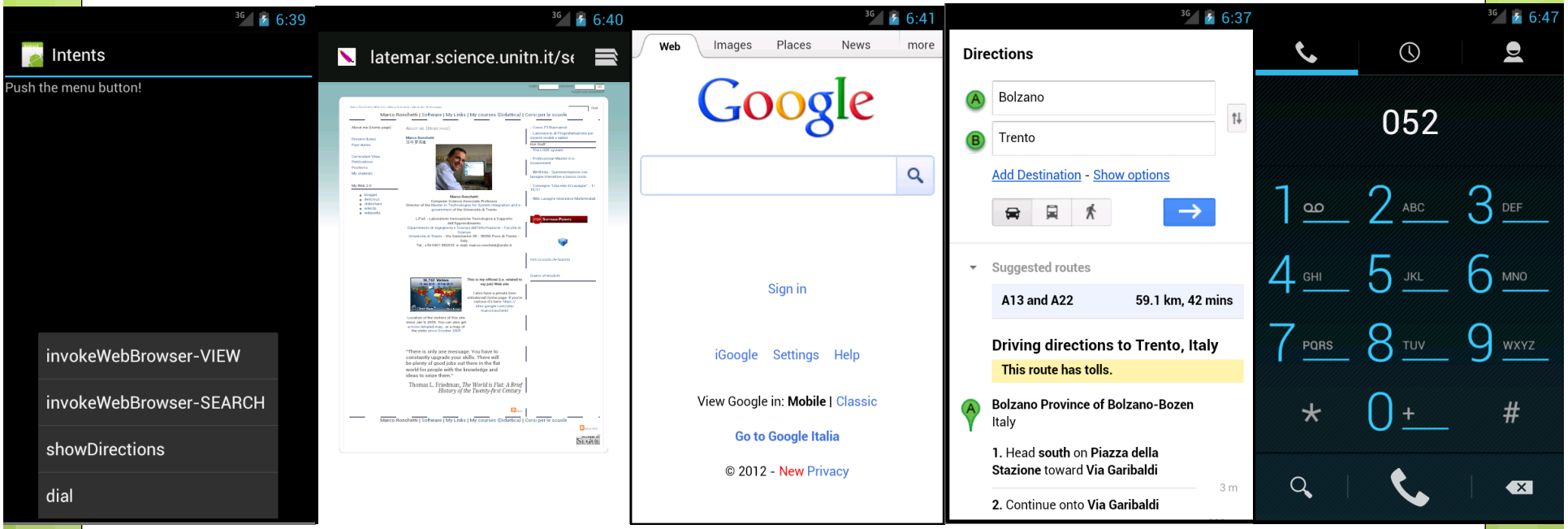
Yes, but how? By means of Intents

These activities are bound at runtime: newly installed applications can take advantage of already installed activities

Our App



Our activities



Our code – IntentUtils - 1

```
package it.unitn.science.latemar;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;

public class IntentUtils {

  public static void invokeWebBrowser(Activity activity) {
        Intent intent=new Intent(Intent.ACTION_VIEW);
        intent.setData(Uri.parse("http://latemar.science.unitn.it"));
        activity.startActivity(intent);
  }

  public static void invokeWebSearch(Activity activity) {
        Intent intent=new Intent(Intent.ACTION_WEB_SEARCH,
            Uri.parse("http://www.google.com"));
        activity.startActivity(intent);
  }
```

Our code – IntentUtils - 2

```
public static void dial(Activity activity) {
        Intent intent=new Intent(Intent.ACTION_DIAL);
        activity.startActivity(intent);
}

public static void showDirections(Activity activity){
        Intent intent = new Intent(android.content.Intent.ACTION_VIEW,
          Uri.parse("http://maps.google.com/maps? saddr=Bolzano&daddr=Trento")
        activity.startActivity(intent);
}
}
```

Our Code: IntentsActivity -1

```
package it.unitn.science.latemar;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;

public class IntentsActivity extends Activity {
@Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      TextView tv=new TextView(this);
      tv.setText("Push the menu button!");
      setContentView(tv);
   }

  public boolean onCreateOptionsMenu(Menu menu){
                super.onCreateOptionsMenu(menu);
                int base=Menu.FIRST;
                MenuItem item1=menu.add(base,1,1,"invokeWebBrowser-VIEW");
                MenuItem item2=menu.add(base,2,2,"invokeWebBrowser-SEARCH");
                MenuItem item3=menu.add(base,3,3,"showDirections");
                MenuItem item5=menu.add(base,4,4,"dial");
                return true;
   }
```

Our Code: IntentsActivity -2

```
public boolean onOptionsItemSelected(MenuItem item) {
        System.err.println("item="+item.getItemId());
        if (item.getItemId()==1)
                IntentUtils.invokeWebBrowser(this);
        else if (item.getItemId()==2)
                IntentUtils.invokeWebSearch(this);
        else if (item.getItemId()==3)
                IntentUtils.showDirections(this);
        else if (item.getItemId()==4)
                IntentUtils.dial(this);
        else
                return super.onOptionsItemSelected(item);
        return true;
  }
}
```

14

Intent structure and resolution

Marco Ronchetti
Università degli Studi di Trento

Intent structure

Who will perform the action?

- **Component name** **(can be unnamed)**

Which action should be performed?

- **Action identifier (a string)**

Which data should the action act on ?

- **Data** The URI of the data to be acted on

How we classify the action to be performed?

- **Category** A (usually codified) string.

How do we directly pass data?

- **Extras** Key-value pairs for additional information that should be delivered to the component handling the intent

How do we specify behavior modification?

- **Flags** Flags of various sorts.

Examples of action/data pairs

ACTION_VIEW content://contacts/people/1

- Display information about the person whose identifier 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. Note how the VIEW action does what what is considered the most reasonable thing for a particular URI.

ACTION_DIAL tel:123

- Display the phone dialer with the given number filled in.

ACTION_EDIT content://contacts/people/1

- Edit information about the person whose identifier is "1".

ACTION_VIEW content://contacts/people/

- Display a list of people, which the user can browse through.

Standard Actions Identifiers

ACTION_MAIN
ACTION_VIEW
ACTION_ATTACH_DATA
ACTION_EDIT
ACTION_PICK
ACTION_CHOOSER
ACTION_GET_CONTENT
ACTION_DIAL
ACTION_CALL
ACTION_SEND
ACTION_SENDTO
ACTION_ANSWER
ACTION_INSERT
ACTION_DELETE
ACTION_RUN
ACTION_SYNC
ACTION_PICK_ACTIVITY
ACTION_SEARCH
ACTION_WEB_SEARCH
ACTION_FACTORY_TEST

For details see http://developer.android.com/reference/android/content/Intent.html#CATEGORY_ALTERNATIVE

Standard Categories

String	CATEGORY_ALTERNATIVE	Set if the activity should be considered as an alternative action to the data the user is currently viewing.
String	CATEGORY_APP_BROWSER	Used with ACTION_MAIN to launch the browser application.
String	CATEGORY_APP_CALCULATOR	Used with ACTION_MAIN to launch the calculator application.
String	CATEGORY_APP_CALENDAR	Used with ACTION_MAIN to launch the calendar application.
String	CATEGORY_APP_CONTACTS	Used with ACTION_MAIN to launch the contacts application.
String	CATEGORY_APP_EMAIL	Used with ACTION_MAIN to launch the email application.
String	CATEGORY_APP_GALLERY	Used with ACTION_MAIN to launch the gallery application.
String	CATEGORY_APP_MAPS	Used with ACTION_MAIN to launch the maps application.
String	CATEGORY_APP_MARKET	This activity allows the user to browse and download new applications.
String	CATEGORY_APP_MESSAGING	Used with ACTION_MAIN to launch the messaging application.
String	CATEGORY_APP_MUSIC	Used with ACTION_MAIN to launch the music application.
String	CATEGORY_BROWSABLE	Activities that can be safely invoked from a browser must support this category.
String	CATEGORY_CAR_DOCK	An activity to run when device is inserted into a car dock.
String	CATEGORY_CAR_MODE	Used to indicate that the activity can be used in a car environment.
String	CATEGORY_DEFAULT	Set if the activity should be an option for the default action (center press) to perform on a piece of data.
String	CATEGORY_DESK_DOCK	An activity to run when device is inserted into a car dock.
String	CATEGORY_DEVELOPMENT_PREFERENCE	This activity is a development preference panel.
String	CATEGORY_EMBED	Capable of running inside a parent activity container.
String	CATEGORY_FRAMEWORK_INSTRUMENTATION_TEST	To be used as code under test for framework instrumentation tests.
String	CATEGORY_HE_DESK_DOCK	An activity to run when device is inserted into a digital (high end) dock.
String	CATEGORY_HOME	This is the home activity, that is the first activity that is displayed when the device boots.
String	CATEGORY_INFO	Provides information about the package it is in; typically used if a package does not contain a CATEGORY_LAUNCHER to provid
String	CATEGORY_LAUNCHER	Should be displayed in the top-level launcher.
String	CATEGORY_LE_DESK_DOCK	An activity to run when device is inserted into a analog (low end) dock.
String	CATEGORY_MONKEY	This activity may be exercised by the monkey or other automated test tools.
String	CATEGORY_OPENABLE	Used to indicate that a GET_CONTENT intent only wants URIs that can be opened with ContentResolver.openInputStream.
String	CATEGORY_PREFERENCE	This activity is a preference panel.
String	CATEGORY_SAMPLE_CODE	To be used as an sample code example (not part of the normal user experience).
String	CATEGORY_SELECTED_ALTERNATIVE	Set if the activity should be considered as an alternative selection action to the data the user has currently selected.
String	CATEGORY_TAB	Intended to be used as a tab inside of an containing TabActivity.
String	CATEGORY_TEST	To be used as a test (not part of the normal user experience).
String	CATEGORY_UNIT_TEST	To be used as a unit test (run through the Test Harness).

Implicit intents and intent resolution

Implicit intents do not name a target (the field for the component name is blank).

In the absence of a designated target, the Android system must find the best component (or components) to handle the intent.

It does so by comparing the contents of the Intent object to *intent filters*, structures associated with components that can potentially receive intents.

Filters advertise the capabilities of a component and delimit the intents it can handle. They open the component to the possibility of receiving implicit intents of the advertised type. If a component does not have any intent filters, it can receive only explicit intents. A component with filters can receive both explicit and implicit intents.

Intent Filters

Only three aspects of an Intent object are consulted when the object is tested against an intent filter:

action
data (both URI and data type)
category

The extras and flags play no part in resolving which component receives an intent.

AndroidManifest.xml

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

    <uses-sdk android:minSdkVersion="15" />

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

Intent messaging is a facility for late run-time binding between components in the same or different applications.

The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed — or, often in the case of broadcasts, a description of something that has happened and is being announced.

There are separate mechanisms for delivering intents to each type of component.

Using intents with activities

An Intent object is passed to Context.startActivity() or Activity.startActivityForResult() to launch an activity or get an existing activity to do something new. (It can also be passed to Activity.setResult() to return information to the activity that called startActivityForResult().)

Other uses of Intents

An Intent object is passed to Context.startService() to initiate a service or deliver new instructions to an ongoing service. Similarly, an intent can be passed to Context.bindService() to establish a connection between the calling component and a target service. It can optionally initiate the service if it's not already running.

Intent objects passed to any of the broadcast methods (such as Context.sendBroadcast(), Context.sendOrderedBroadcast(), or Context.sendStickyBroadcast()) are delivered to all interested broadcast receivers. Many kinds of broadcasts originate in system code.

In each case, the Android system finds the appropriate activity, service, or set of broadcast receivers to respond to the intent, instantiating them if necessary. There is no overlap within these messaging systems: Broadcast intents are delivered only to broadcast receivers, never to activities or services. An intent passed to startActivity() is delivered only to an activity, never to a service or broadcast receiver, and so on.

Sensors

Marco Ronchetti
Università degli Studi di Trento

Sensor categories

Motion sensors

- measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes.

Environmental sensors

- measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

Position sensors

- measure the physical position of a device. This category includes orientation sensors and magnetometers.

Basic code for managing sensors

```
public class SensorActivity extends Activity, implements SensorEventListener {
   private final SensorManager sm;
   private final Sensor sAcc;
   public SensorActivity() {
      sm= (SensorManager)getSystemService(SENSOR_SERVICE);
      sAcc= sm.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
   }
   protected void onPause() {
      super.onPause();
      sm.unregisterListener(this);
   }
   protected void onResume() {
      super.onResume();
      sm.registerListener(this, sAcc, SensorManager.SENSOR_DELAY_NORMAL);
   }
   public void onAccuracyChanged(Sensor sensor, int accuracy) { }
   public void onSensorChanged(SensorEvent event) { }
}
```

SensorManager

SensorManager sm=Context.getSystemService(SENSOR_SERVICE);

List<Sensor> getSensorList(int type)

- get the list of available sensors of a certain type. Sensor

Sensor getDefaultSensor(int type)

- Use this method to get the default sensor for a given type

void registerListener(SensorEventListener listener, Sensor sensor, int rate)

- Registers a SensorEventListener for the given sensor.

void unregisterListener(SensorEventListener listener, Sensor sensor)

- Unregisters a listener for the sensors with which it is registered.

void unregisterListener(SensorEventListener listener)

- Unregisters a listener for all sensors.

- Some methods for transforming data (Vector to matrix representation etc.)

Sensor types

int constants of the Sensor class describing sensor types:

TYPE_ACCELEROMETER

TYPE_ALL A constant describing all sensor types.

TYPE_AMBIENT_TEMPERATURE

TYPE_GRAVITY

TYPE_GYROSCOPE

TYPE_LIGHT

TYPE_LINEAR_ACCELERATION

TYPE_MAGNETIC_FIELD

TYPE_PRESSURE

TYPE_PROXIMITY

TYPE_RELATIVE_HUMIDITY

TYPE_ROTATION_VECTOR

Accelerometer

*"Sensor's values are in **meters/second^2** units. A sensor measures the acceleration applied to the device. For this reason, when the device is sitting on a table (and obviously not accelerating), the accelerometer reads a magnitude of **g** = 9.81 m/s^2. Similarly, when the device is in free-fall and therefore dangerously accelerating towards to ground at 9.81 m/s^2, its accelerometer reads a magnitude of 0 m/s^2."* (Android Developers – sensors)

Orientation sensor

"A compass is a navigational instrument for determining ***direction*** *relative to the Earth's* ***magnetic poles****. It consists of a magnetized pointer (usually marked on the North end) free to align itself with* ***Earth's magnetic field****."* (Compass EN Wiki)

In Android's terminology it is called **Orientation sensor**.

Gyroscope

*"A gyroscope is an instrument consisting of a rapidly **spinning wheel** so mounted as to use the tendency of such a wheel to maintain a fixed position in space, and to resist any force which tries to change it. The way it will move if a twisting force is applied depends on the extent and orientation of the force and the way the gyroscope is mounted. A free vertically spinning gyroscope remains vertical as the carrying vehicle tilts, so providing an artificial horizon. A horizontal gyroscope will maintain a certain bearing, and therefore indicate a vessel's heading as it turns. Modern gyroscopes (including those built-in in smartphones) no longer have a spinning wheel."* (Gyroscope Cambridge Encyclopedia)

*"All values are in **radians/second** and measure the rate of rotation around the **X**, **Y** and **Z** axis. The coordinate system is the same as is used for the acceleration sensor."* (Android Developers – sensors) Rotation is positive in the **counter-clockwise** direction.

Sensor class

float getMaximumRange()

- maximum range of the sensor in the sensor's unit.

int getMinDelay()

- minimum delay allowed between two events in microsecond or zero if this sensor only returns a value when the data it's measuring changes

String getName()

float getPower()

- the power in mA used by this sensor while in use

float getResolution()

- resolution of the sensor in the sensor's unit.

int getType()

String getVendor()

int getVersion()

```
SensorManager sm= (SensorManager)getSystemService(SENSOR_SERVICE);
List<Sensor> sensorList = sm.getSensorList(Sensor.TYPE_ALL);
StringBuilder sensorString = new StringBuilder("Sensors:\n");
for(int i=0; i<sensorList.size(); i++) {
    sensorString.append(sensorList.get(i).getName()).append(", \n");
}
```

HTC EVO 4G
BMA150 3-axis Accelerometer
AK8973 3-axis Magnetic field sensor
AK8973 Orientation sensor
CM3602 Proximity sensor
CM3602 Light sensor

Samsung Nexus-S
KR3DM 3-axis Accelerometer
AK8973 3-axis Magnetic field sensor
AK8973 Orientation sensor
GP2A Light sensor
GP2A Proximity sensor
K3G Gyroscope sensor
Gravity Sensor
Linear Acceleration Sensor
Rotation Vector Sensor

Code examples

- http://www.vogella.com/articles/AndroidSensor/article.html accelerometer and compass examples
- http://developer.android.com/guide/topics/sensors/sensors_overview.html and following pages
- http://developer.android.com/resources/samples/ApiDemos/src/com/example/android/apis/os/RotationVectorDemo.html a more complex example with 3D graphics

Interface SensorEventListener

abstract void onAccuracyChanged(Sensor sensor, int accuracy)

- Called when the accuracy of a sensor has changed.

abstract void onSensorChanged(SensorEvent event)

- Called when sensor values have changed.

Sensors limitation - 1

From Jim Steele

Using available sensors in the Android platform: current limitations and expected improvements

Comparing the sensors on these two phones demonstrates the **sensor fragmentation** now found in Android:

1) **Non-standard sensor availability**: The Nexus-S has a gyroscope (from ST Micro), but the EVO does not. In fact, most Android devices do not have a gyroscope. There is no standard availability of sensors across devices.

2) **Non-standard sensor capability**: The BMA150 is a Bosch Sensortec 10-bit accelerometer, and the KR3DM is a ST Micro 12-bit accelerometer (using a special part number). In fact, there is no standard capability requirement for sensors across devices to ensure consistent
resolution, noise floor, or update rate.

Sensors limitation - 2

3) **Sensors not fully specified**: The AK8973 is an AKM magnetometer, which is only 8-bits. Analyzing this data stream shows it is low-pass filtered. This fact is not published on the phone or even the sensor datasheet. Many sensors have characteristics not specified such as bias changes, non-uniform gain, and skew (coupling between measurement axes). **Algorithms that use sensors without knowing these extra characteristics may produce incorrect information**.

4) **Broken virtual sensors**: The AKM sensor driver abstracts out an orientation virtual sensor which is derived from the combination of two sensors: the accelerometer and magnetometer. However, support for this virtual sensor was dropped early on, so the TYPE_ORIENTATION sensor is deprecated and the method **SensorManager.getOrientation()** should be used instead. Furthermore, the new virtual sensors introduced in Android 2.3 (Gingerbread) are not supported on all devices.

The sensor differences between just these two phones is substantial. So when a developer is faced with **writing apps utilizing sensors across as many devices as possible, it is a daunting task**.

Furthermore, **the Android platform is not optimized for real-time sensor data acquisition**.

What can you do with accelerometer and gyroscope?

http://www.starlino.com/imu_guide.html

A Guide To using IMU (Accelerometer and Gyroscope Devices) in Embedded Applications.

"This guide is intended to everyone interested in inertial MEMS (Micro-Electro-Mechanical Systems) sensors, in particular Accelerometers and Gyroscopes as well as combination IMU devices (Inertial Measurement Unit)."

- what does an accelerometer measure
- what does a gyroscope (aka gyro) measure
- how to convert analog-to-digital (ADC) readings that you get from these sensor to physical units (those would be g for accelerometer, deg/s for gyroscope)
- how to combine accelerometer and gyroscope readings in order to obtain accurate information about the inclination of your device relative to the ground plane

http://www.starlino.com/dcm_tutorial.html

DCM Tutorial – An Introduction to Orientation Kinematics

Emulator limits

The emulator does not emulate sensors, so what can you do without a physical device?

BUT…

There is an app that emulates many sensors, and that you can use as data provider!

SensorSimulator

OpenIntents SensorSimulator lets you simulate sensor data with the mouse in real time. Moreover, you can simulate your battery level and your gps position too, using a telnet connection.

Now you can also record a sequence with states from a real device.

See

http://code.google.com/p/openintents/wiki/SensorSimulator

Fragments

Fragments

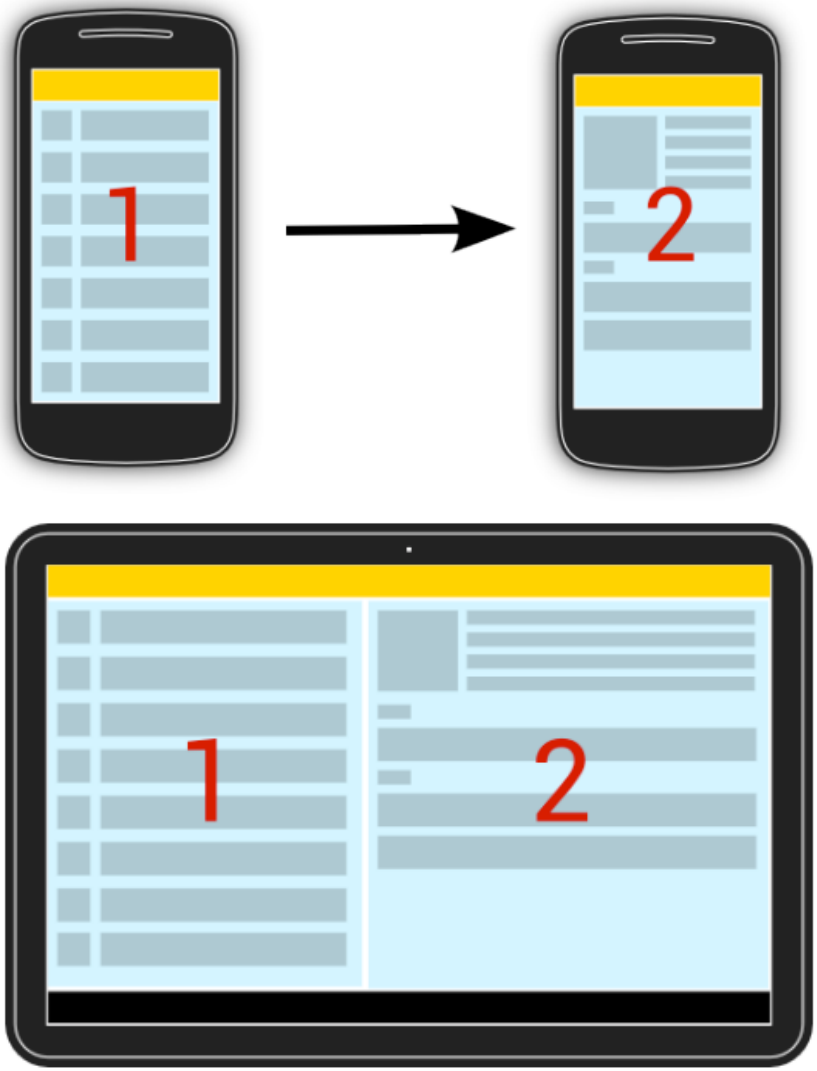
A fragment is a self-contained, modular section of an application's user interface and corresponding behavior that can be embedded within an activity.

Fragments can be assembled to create an activity during the application design phase, and added to, or removed from an activity during application runtime to create a dynamically changing user interface.

Fragments may only be used as part of an activity and cannot be instantiated as standalone application elements.

A fragment can be thought of as a functional "sub-activity" with its own lifecycle similar to that of a full activity.

Using fragments



Fragments lifecycle

Method	Description
onAttach()	The fragment instance is associated with an activity instance.The activity is not yet fully initialized
onCreate()	Fragment is created
onCreateView()	The fragment instance creates its view hierarchy. The inflated views become part of the view hierarchy of its containing activity.
onActivityCreated()	Activity and fragment instance have been created as well as thier view hierarchy. At this point, view can be accessed with the `findViewById()` method. example.
onResume()	Fragment becomes visible and active.
onPause()	Fragment is visibile but becomes not active anymore, e.g., if another activity is animating on top of the activity which contains the fragment.
onStop()	Fragment becomes not visible.

Defining a new fragment (from code)

To define a new fragment you either extend the android.app.Fragment class or one of its subclasses, for example, ListFragment, DialogFragment, PreferenceFragment or WebViewFragment.

Defining a new fragment (from code)

```
public class DetailFragment extends Fragment {
 @Override
 public View onCreateView(LayoutInflater inflater,
        ViewGroup container, Bundle savedInstanceState) {
    View view=inflater.inflate(
        R.layout.fragment_rssitem_detail,
        container, false);
   return view;
 }
 public void setText(String item) {
    TextView view = (TextView)
                getView().findViewById(R.id.detailsText);
    view.setText(item);
 }
}
```

XML-based fragments

```
<RelativeLayout xmlns:android="http://schemas.android.com/
apk/res/android" xmlns:tools="http://schemas.android.com/
tools" android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".FragmentDemoActivity" >
<fragment android:id="@+id/fragment_one"
android:name="com.example.myfragmentdemo.FragmentOne"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_alignParentLeft="true"
android:layout_centerVertical="true" tools:layout="@layout/
fragment_one_layout" />
</RelativeLayout>
```

Adding-removing fragments at runtime

The FragmentManager class and the FragmentTransaction class allow you to add, remove and replace fragments in the layout of your *activity*.

Fragments can be dynamically modified via transactions. To dynamically add fragments to an existing layout you typically define a container in the XML layout file in which you add a *Fragment*.

```
FragmentTransaction ft =
getFragmentManager().beginTransaction();
ft.replace(R.id.your_placehodler, new
YourFragment());
ft.commit();
```

A new *Fragment* will replace an existing *Fragment* that was previously added to the container.

Finding if a fragment is already part of your Activity

```
DetailFragment fragment = (DetailFragment)
        getFragmentManager().
        findFragmentById(R.id.detail_frag);


if (fragment==null) {
                // start new Activity
} else {
                fragment.update(...);
}
```

Communication: activity -> fragment

In order for an activity to communicate with a fragment, the activity must identify the fragment object via the ID assigned to it using the findViewById() method. Once this reference has been obtained, the activity can simply call the public methods of the fragment object.

Communication: fragment-> activity

Communicating in the other direction (from fragment to activity) is a little more complicated.

A) the fragment must define a listener interface, which is then implemented within the activity class.

```
public class MyFragment extends Fragment {
  AListener activityCallback;
  public interface AListener {
            public void someMethod(int par1, String par2);
  }
  …
```

Communication: fragment-> activity

B. the onAttach() method of the fragment class needs to be overridden and implemented. The method is passed a reference to the activity in which the fragment is contained. The method must store a local reference to this activity and verify that it implements the interface.

```
public void onAttach(Activity activity) {
  super.onAttach(activity);
  try { activityCallback = (AListener) activity;
  } catch (ClassCastException e) {
      throw new ClassCastException(
            activity.toString()
            + " must implement ToolbarListener");
} }
```

Communication: fragment-> activity

C. The next step is to call the callback method of the activity from within the fragment. When and how this happens is entirely dependent on the circumstances under which the activity needs to be contacted by the fragment. For the sake of an example, the following code calls the callback method on the activity when a button is clicked:

```
public void buttonClicked(View view) {
    activityCallback.someMethod(arg1, arg2);
}
```

Communication: fragment-> activity

All that remains is to modify the activity class so that it implements the ToolbarListener interface.

```
public class MyActivity extends
FragmentActivity implements
MyFragment.AListener {
 public void someMethod(String arg1, int arg2)
      {
      // Implement code for callback method
      }
.
.
}
```

Esempio

vedi
http://www.vogella.com/tutorials/AndroidFragments/article.html

sez. 10