


# Short & Quick messages: Toast

Marco Ronchetti
Università degli Studi di Trento

# Toast

A toast is a view containing a quick little message for the user (shown for a time interval).

When the view is shown to the user, appears as a floating view over the application. It will never receive focus.

It is as unobtrusive as possible, while still showing the user the information you want them to see.

setGravity(), setDuration(), set Text(), view()



# Threads

Marco Ronchetti
Università degli Studi di Trento

# Threads

When an application is launched, the system creates a thread of execution for the application, called "main" or "UI thread"

This thread dispatches events to the user interface widgets, and draws (uses the android.widget and android.view packages).

Unlike Java AWT/Swing, separate threads are NOT created automatically.

Methods that respond to system callbacks (such as onKeyDown() to report user actions or a lifecycle callback method) always run in the UI thread.

If everything is happening in the UI thread, performing long operations such as network access or database queries will block the whole UI. When the thread is blocked, no events can be dispatched, including drawing events. From the user's perspective, the application appears to hang.

If the UI thread is blocked for more than 5 sec the user is presented with the"ANR - application not responding" dialog.

# the Andoid UI toolkit is not thread-safe !

Consequence:

you must not manipulate your UI from a worker thread—all manipulation to the user interface must be done within the UI thread.

You MUST respect these rules:

- Do not block the UI thread
- Do not access the Android UI toolkit from outside the UI thread

# An example from android developers

```
public void onClick(View v) {
        Bitmap b = loadImageFromNetwork(
                        "http://example.com/image.png");
        myImageView.setImageBitmap(b);
}
```

WRONG!
Potentially
Slow
Operation!

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork(
                "http://example.com/image.png");
            myImageView.setImageBitmap(b);
        })
            .start();
}
```

WRONG!
A non UI thread
accesses the UI!

# Still not the solution…

```
public void onClick(View v) {
  Bitmap b;
      new Thread(new Runnable() {
          public void run() {
              b = loadImageFromNetwork(
                  "http://example.com/image.png");
          })
      .start();
      myImageView.setImageBitmap(b);
}
```

**WRONG!**
**This does not wait for the thread to finish!**

# The solution

public boolean post (Runnable action)

- Causes the Runnable to be sent to the UI thread and to be run therein. It is invoked on a View from outside of the UI thread.

public boolean postDelayed (Runnable action, long delayMillis)

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork(
                "http://example.com/image.png");
            myImageView.post(
                new Runnable() {
                    public void run() {
                        mImageView.setImageBitmap(b);
                    }
                }
            });
        .start();
}
```

OK! This code will be run in the UI thread

# Java reminder: varargs

void f(String pattern, Object... arguments);

The three periods after the final parameter's type indicate that the final argument may be passed

- as an array *or*
- as a sequence of arguments.

Varargs can be used *only* in the final argument position.

```
Object a, b, c, d[10];
…
f("hello",d);
f("hello",a,b,c);
```

# Varargs example

```
public class Test {
    public static void main(String args[]){ new Test(); }

    Test(){
        String k[]={"uno","due","tre"};
        f("hello",k);
        f("hello",“alpha“,“beta“);
        // f("hello“,“alpha“,“beta“,k); THIS DOES NOT WORK!
    }

    void f(String s, String... d){
        System.out.println(d.length);
        for (String k:d) {
            System.out.println(k);
        }
    }

}
```

# AsyncTask<Params,Progress,Result>

Creates a new asynchronous task. The constructor must be invoked on the UI thread.

AsyncTask must be subclassed, and instantiated in the UI thread.

Methods to be overridden:

| method | where | when |
|---|---|---|
| void onPreExecute() | UI Thread | before |
| Result doInBackground(Params...) | Separate new thread | during |
| void onProgressUpdate(Progress...) | UI Thread | |
| void onPostExecute(Result) | UI Thread | after |

# A simpler and more elegant solution

```
public void onClick(View v) {
  new DownloadImageTask().execute("http://example.com/image.png");
}
```

```
private class DownloadImageTask extends AsyncTask<String, Void, Bitmap> {
    protected Bitmap doInBackground(String... urls) {
        return loadImageFromNetwork(urls[0]);
    }
    protected void onPostExecute(Bitmap result) {
        mImageView.setImageBitmap(result);
    }
}
```

# Using Progress

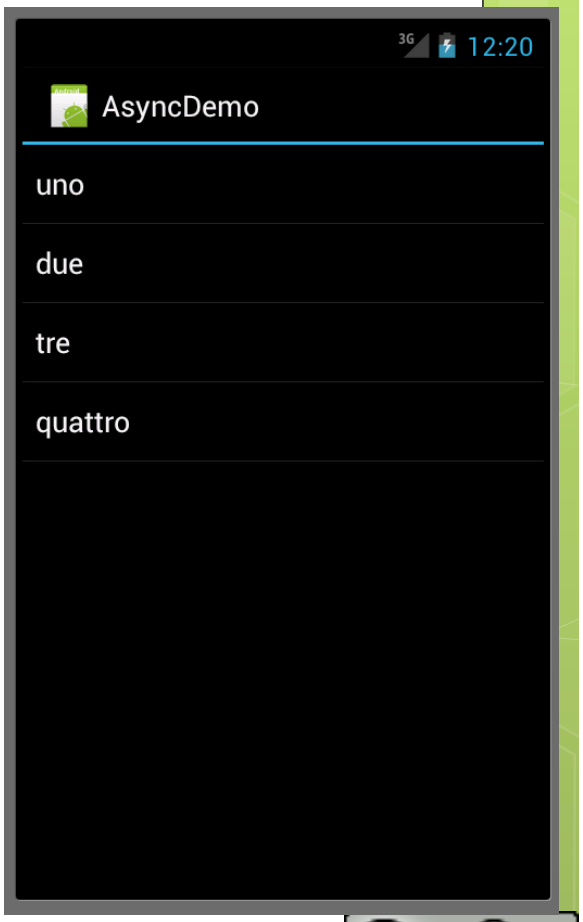
```
package it.unitn.science.latemar;
import ...
```

```
public class AsyncDemoActivity extends ListActivity {
 private static final String[] item{"uno","due","tre","quattro",
          "cinque","sei", "sette","otto","nove",
          "dieci","undici","dodici",};

@Override
 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  ListView listView = getListView();

  setListAdapter(new ArrayAdapter<String>(this,
                android.R.layout.simple_list_item_1,
                new ArrayList<String>()));

  new AddStringTask().execute();
 }
```




Adapted from the source code of
http://commonsware.com/Android/

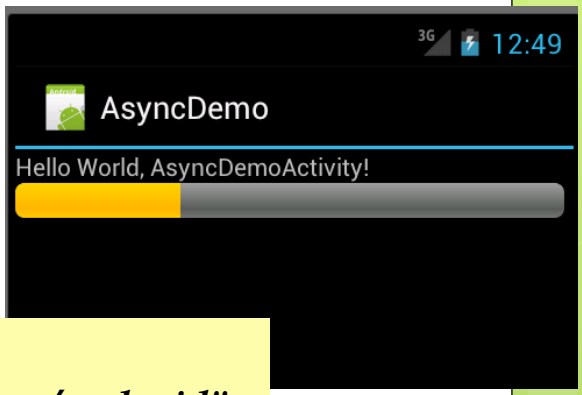
# Using Progress

This is an inner class!

```
class AddStringTask extends AsyncTask<Void, String, Void> {
  @Override
  protected Void doInBackground(Void... unused) {
    for (String item : items) {
      publishProgress(item);
      SystemClock.sleep(1000);
    }
    return(null);
  }
  @SuppressWarnings("unchecked")
  @Override
  protected void onProgressUpdate(String... item) {
    ((ArrayAdapter<String>)getListAdapter()).add(item[0]);
  }
```

```
@Override
  protected void onPostExecute(Void unused) {
    Toast
      .makeText(AsyncDemoActivity.this,
                "Done!", Toast.LENGTH_SHORT)
      .show();
  }
}
```

# Using the ProgressBar




```
<?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" />
   <ProgressBar
      android:id="@+id/pb1"
      android:max="10"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      style="@android:style/Widget.ProgressBar.Horizontal"
      android:layout_marginRight="5dp" />
</LinearLayout>
```

```
public class AsyncDemoActivity2
              extends Activity {
  ProgressBar pb;
  @Override
  public void onCreate(Bundle state) {
    super.onCreate(state);
    setContentView(R.layout.main);
    pb=(ProgressBar) findViewById(R.id.pb1);
    new AddStringTask().execute();
  }
```

# Using the ProgressBar

```
class AddStringTask extends AsyncTask<Void, Integer, Void> {
@Override
protected void doInBackground(Void... unused) {
  int item=0;
  while (item<10 ){
    publishProgress(++item);
    SystemClock.sleep(1000);
  }
}
@Override
protected void onProgressUpdate(Integer... item) {
  pb.setProgress(item[0]);
}
}
```

17

# Application Context

Marco Ronchetti
Università degli Studi di Trento

# The Context

An interface to global information about an application environment.

It allows accessing application-specific resources and classes, as well as up-calls for application-level operations such as launching activities, broadcasting and receiving intents, etc.

We have seen it in various cases:

- Activity is subclass of Context
- new Intent(Context c, Class c);
- isIntentAvailable(Context context, String action)

# A global Application Context

Is there a simple way to maintain and access the application context from everywhere it's needed?

a) Modify the Android Manifest adding the "name" parameter to the application tag

```
<application android:name="myPackage.MyApplication"> …
</application>
```

b) Write the class

```
public class MyApplication extends Application{
    private static Context context;
    public void onCreate(){
        super.onCreate();
        MyApplication.context = getApplicationContext();
    }
    public static Context getAppContext() {
        return MyApplication.context;
    }
}
```

c) Access MyApplication.getAppContext() to get your application context statically from eveywhere.



# Fragments

# Fragments

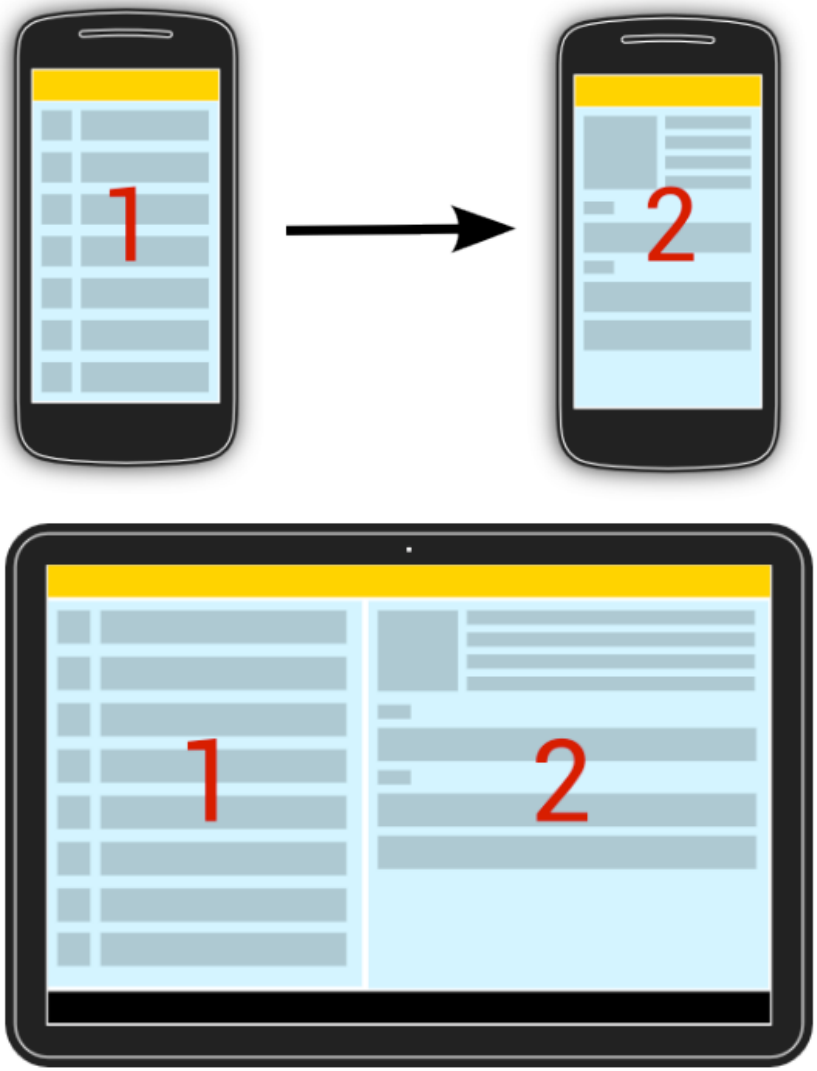
A fragment is a self-contained, modular section of an application's user interface and corresponding behavior that can be embedded within an activity.

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
- 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 AListener");
} }
```

# Communication: fragment-> activity

C. The next step is to call the callback method of the activity from within the fragment. For 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 theAListener 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