Agenda

• Listeners
• Multithreading
• Drawable resources
• SHOWCASE
• LAB:
  – Advanced: group lab
  – Beginners: introduction to OOP
Stuff

- We have Twitter! **MIT-AITI Colombia**
- Android Book?
- Homework until date:
  - Ideas generation
  - Java calc
  - Read chapter of Book
- Remember me the points, otherwise not taken into account
Video

- Our dear Steve…
Listeners

- Activities **listen** to events happening in it
- Examples:
  - Button click: onClick()
  - Button long click: onLongClick()
  - Touch screen: onTouch()
  - Introducing text in a EditText
  - …
Steps to create listeners

1. Set the Activity as a Listener (implementing the corresponding interface)
2. Localize the object responsible of the event (Eg: Button, EditText,…)
3. Tell that object that the activity is listening
4. Implement the **callback function**.
Example 1: Event-handling with Buttons

Method 1: Define call-backs using code

```java
public class StatusActivity extends Activity implements OnClickDrawable{
    Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.status_activity);
        button = (Button) findViewById(R.id.updateButton);
        button.setOnClickListener(this);
    }
    @Override
    public void onClick(View view) {
        String text = editText.getText().toString();
        twitter.setStatus(text);
    }
}
```

1. Implement the interface at the activity
2. Set the listener
3. Override the callback function

Method 2: Define call-backs in Layout XML files

```xml
<Button android:id="@+id/button1" android:layout_width="80px"
        android:layout_height="fill_parent" android:onClick="clickhandler"
        android:text="1">
</Button>
```

```java
public void clickhandler(View clickedobject) {
    switch (idOfClickedObject) {
        case R.id.button1:
            //do something
            break;
    }
}
```
Example 2: TextView

TextView interface for observing how the content of a EditText changes.

```java
private TextWatcher mTextWatcher = new TextWatcher() {
    @Override
    public void beforeTextChanged(CharSequence charSequence, int start, int count, int before) {
        // This method is called to notify you that, within <charSequence>, the <count> characters
        // beginning at <start> have just replaced old text that had length <before>.
    }

    @Override
    public void onTextChanged(CharSequence charSequence, int start, int count, int after) {
        // This method is called to notify you that, within <charSequence>, the count characters
        // beginning at <start> are about to be replaced by new text with length <after>.
    }

    @Override
    public void afterTextChanged(Editable editable) {
        // This method is called to notify you that, somewhere within <editable>, the text has been changed.
    }
};
```

// Called when the activity is first created.
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.status_activity);

    EditText editText1 = new EditText(this);
    editText1.addTextChangedListener(mTextWatcher);
```
MultiThreading

• Execution of multiple tasks “in parallel”.
  – Asynchronously

• Incentive: Execute a task that may take a long time without blocking the main task
  – Does not affect user experience

• Solution: Use of `AsyncTask()`
AsyncTask()

- Performs background operations and publishes results to the UI thread
- Must be subclassed (cannot call AsyncTask directly)
AsyncTask() (cont)

- Uses generic type parameters

```java
private class MyTask extends AsyncTask<Void, Void, Void> { ... }
```

- Input Parameters:
  - **Params**: Sent to task upon execution
  - **Progress**: Units published during background operation
  - **Result**: Type of result of the background operation

- Use Void for any unused type
Steps (methods) of AsyncTask()

- **Execution methods**
  - `doInBackground( Params... )`
  - `onProgressUpdate( Progress... )`
  - `onPostExecute( Result )`

- **Task invoked (executed) by calling**
  - `execute( Params )`
Rules of AsyncTask()

- The AsyncTask class must be loaded using the UI thread (foreground Activity)
- `execute(Params ...)` must be invoked in the UI thread
- Do not call any of the other class methods manually.
Resources

- AsyncTask description

- **Gargenta – Learning Android**
  - Chapter 6 -> Threading in Android
Drawable resources

- Icons
- Backgrounds
- Graphical titles
- Arrows
- Buttons background images…

- PROBLEM: What size in pixels??
Drawable Resources

- Start with your raw resource in vector format and generate the images for each density using the following size scale:

  - xhdpi: 2.0
  - hdpi: 1.5
  - mdpi: 1.0 (baseline)
  - ldpi: 0.75
Drawable Resources

- EXAMPLE: If you generate a 200x200 image for xhdpi devices, you should generate the same resource in 150x150 for hdpi, 100x100 for mdpi, and 75x75 for ldpi devices.

MyProject/
  res/
    drawable-xhdpi/
      awesomeimage.png
    drawable-hdpi/
      awesomeimage.png
    drawable-mdpi/
      awesomeimage.png
    drawable-ldpi/
      awesomeimage.png

Any time you reference @drawable/awesomeimage, the system selects the appropriate bitmap based on the screen's density.