Systems Programming

Graphical User Interfaces

Julio Villena Román (LECTURER)
<jvillena@it.uc3m.es>

CONTENTS ARE MOSTLY BASED ON THE WORK BY:
José Jesús García Rueda

Universidad Carlos III de Madrid

Systems Programming

GUIs based on Java Swing

Julio Villena Román (LECTURER)

<jvillena@it.uc3m.es>

CONTENTS ARE MOSTLY BASED ON THE WORK BY:

José Jesús García Rueda

Universidad Carlos III de Madrid

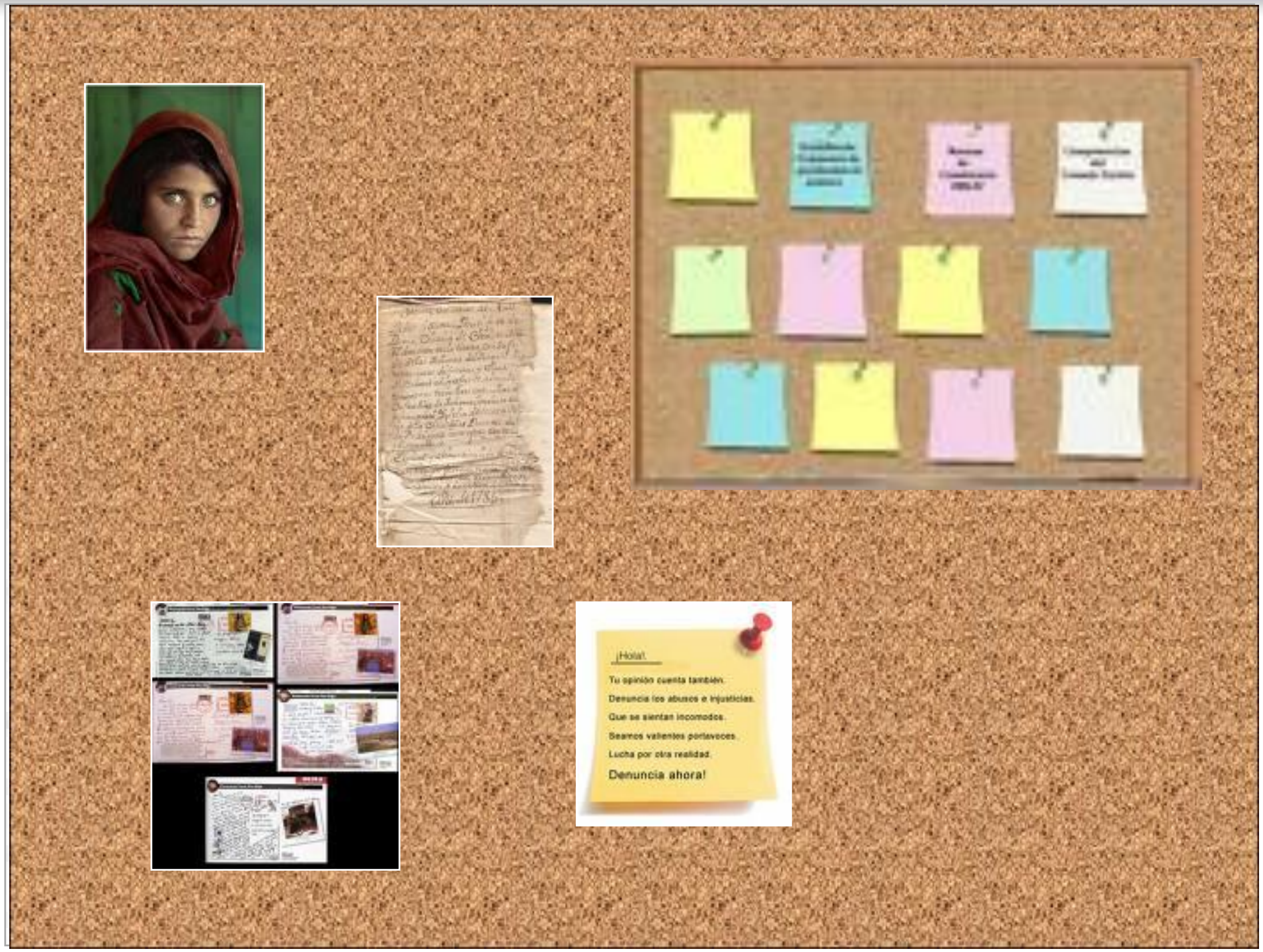
Introduction

- What are the GUIs?
- Well known examples…

Basic concepts

- Graphical application
- Containers
- Actions
- Events
- Graphical elements:
  - Menu bar
  - Title bar
  - Minimize and maximize buttons
  - Closing button
  - Scroll
  - Window frame
  - Icons
  - Buttons
  - Text areas

The wall metaphor



¡Hola!

Tu opinion cuenta tambien.

Denuncia los abusos e injusticias.

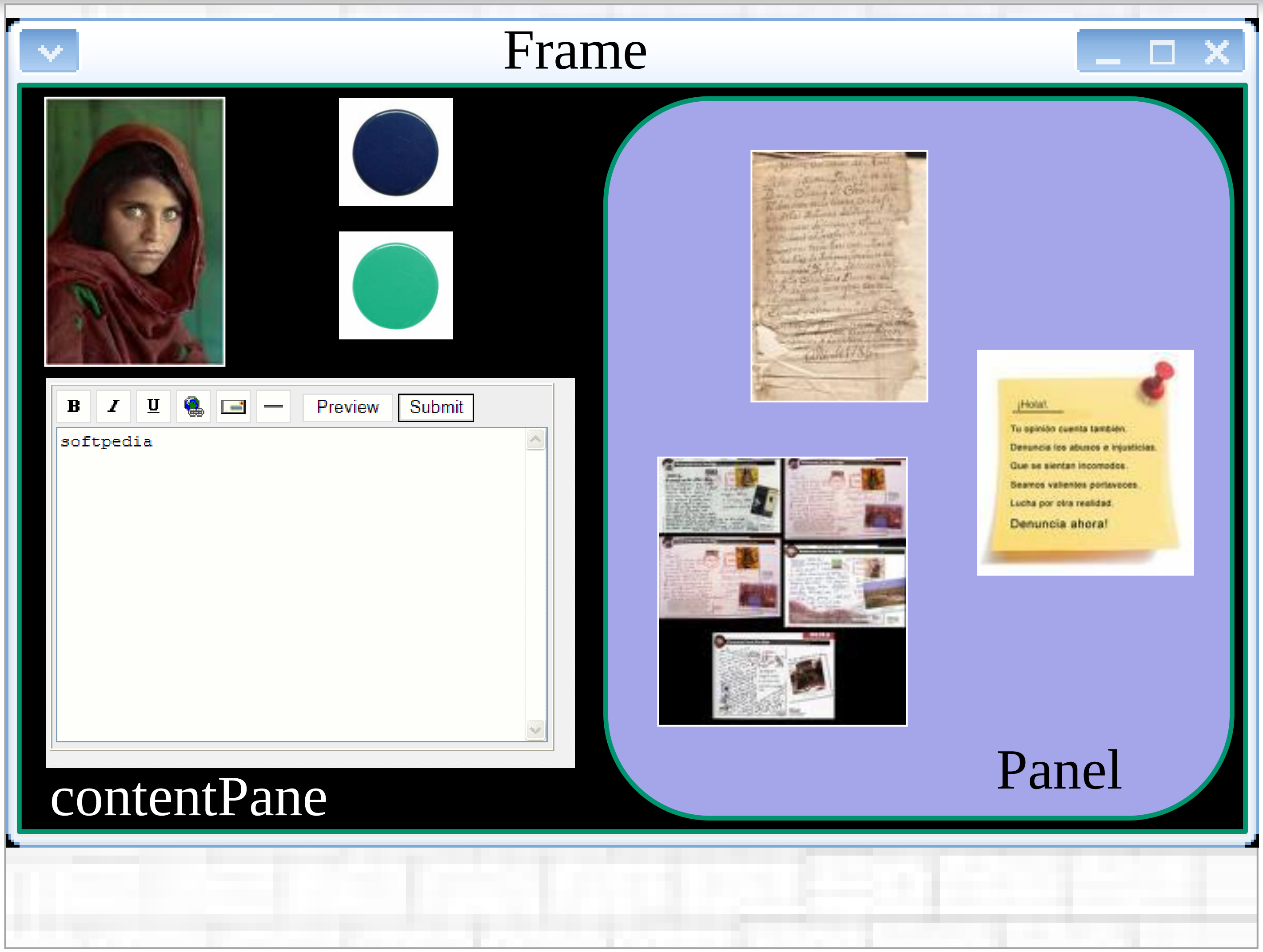
Que se sientan incomodos.

Seamos valientes portavoces.

Lucha por otra realidad.

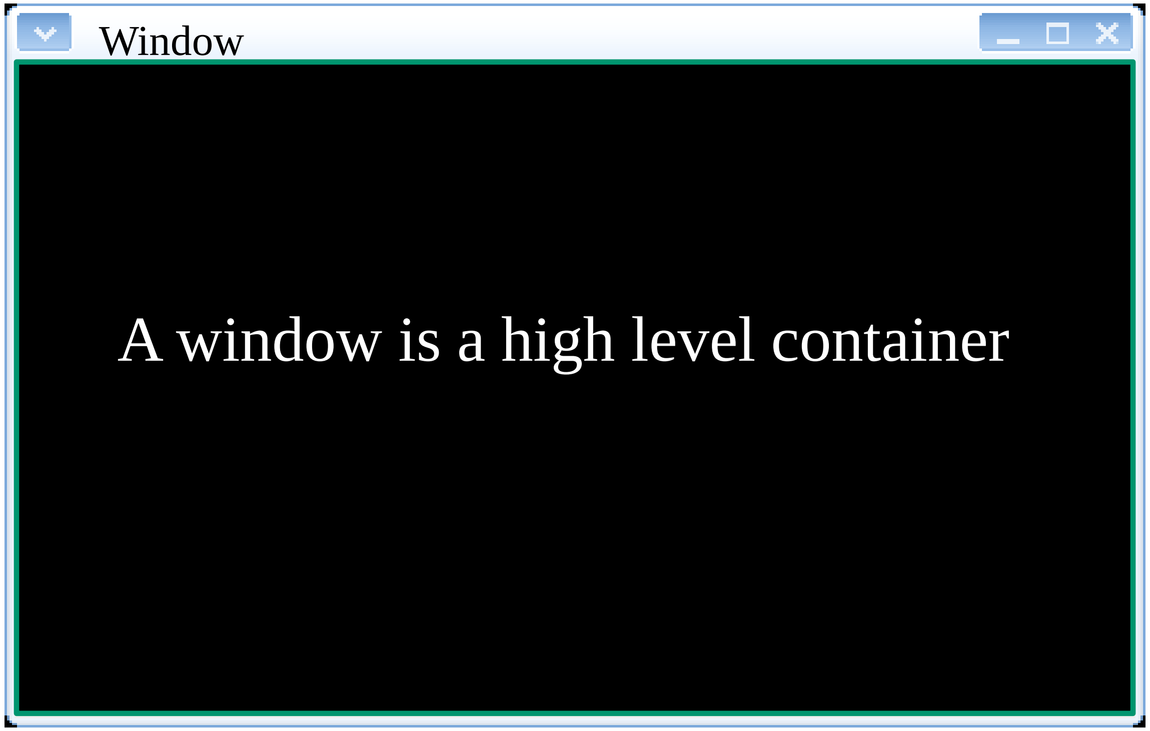
Denuncia ahora!

The wall metaphor


Frame
Preview
Submit
softpedia
¡Hola!
Tu opinión cuenta también.
Denuncia los abusos e injusticias.
Que se sientan incomodos.
Seamos valientes portavoces.
Lucha por otra realidad.
Denuncia ahora!
Panel
contentPane

Creating the wall

- How to create a window in Java?



How to create a window in Java?

```
import javax.swing.* ;

public class Example extends JFrame {

              ethods starts everything*/
              atic void main (String argv[]) {

            Example window= new Example();

            window.setSize (400, 400);
            window.setVisible(true);

      }
}
```

A window in Java is just a class extending Jframe, the generic window.

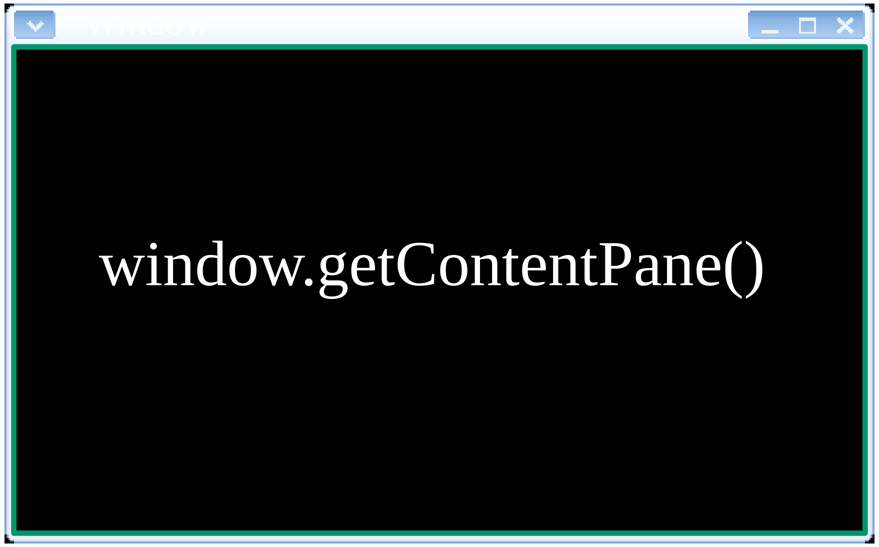
The classes needed to build GUIs are included in the Swing package

The window must be made visible explicitly

Universidad Carlos III de Madrid

Covering the wall with cork

- Every high level container in Swing (windows, for instance) will be “covered” with a “contentPane”
- The rest of the graphical components will be placed on it
  - Including other containers



Adding it to the code

```
import javax.swing.* ;

public class Example extends JFrame {

      /* This methods starts everything*/
      public static void main (String argv[]) {

            Example window = new Example();

            window.getContentPane().add(…);

            window.setSize (400, 400);
            window.setVisible(true);

      }
}
```

Universidad Carlos III de Madrid

What elements can I “attach to the cork”?

- In the **`contentPane`** you can put elements from the Swing package:
  - Labels: `JLabel`
  - Buttons: `JButton`
  - Text boxes: `JTextField`, `JTextArea`
  - Checkboxes: `JCheckBox`
  - Option buttons: `JRadioButton`
  - Lists: `JList`
  - Scroll bars: `JScrollBar`
- All the Swing components extend `JComponent`

Universidad Carlos III de Madrid

And how can I attach them?

```
JButton button;
JLabel label;

public Example() {

      label = new JLabel(“A label");
      button = new JButton(“A button");
      button.setSize(100, 70);
      getContentPane().add(button);
      getContentPane().add(label);

}
```

Universidad
Carlos III de Madrid

LITTLE PAUSE

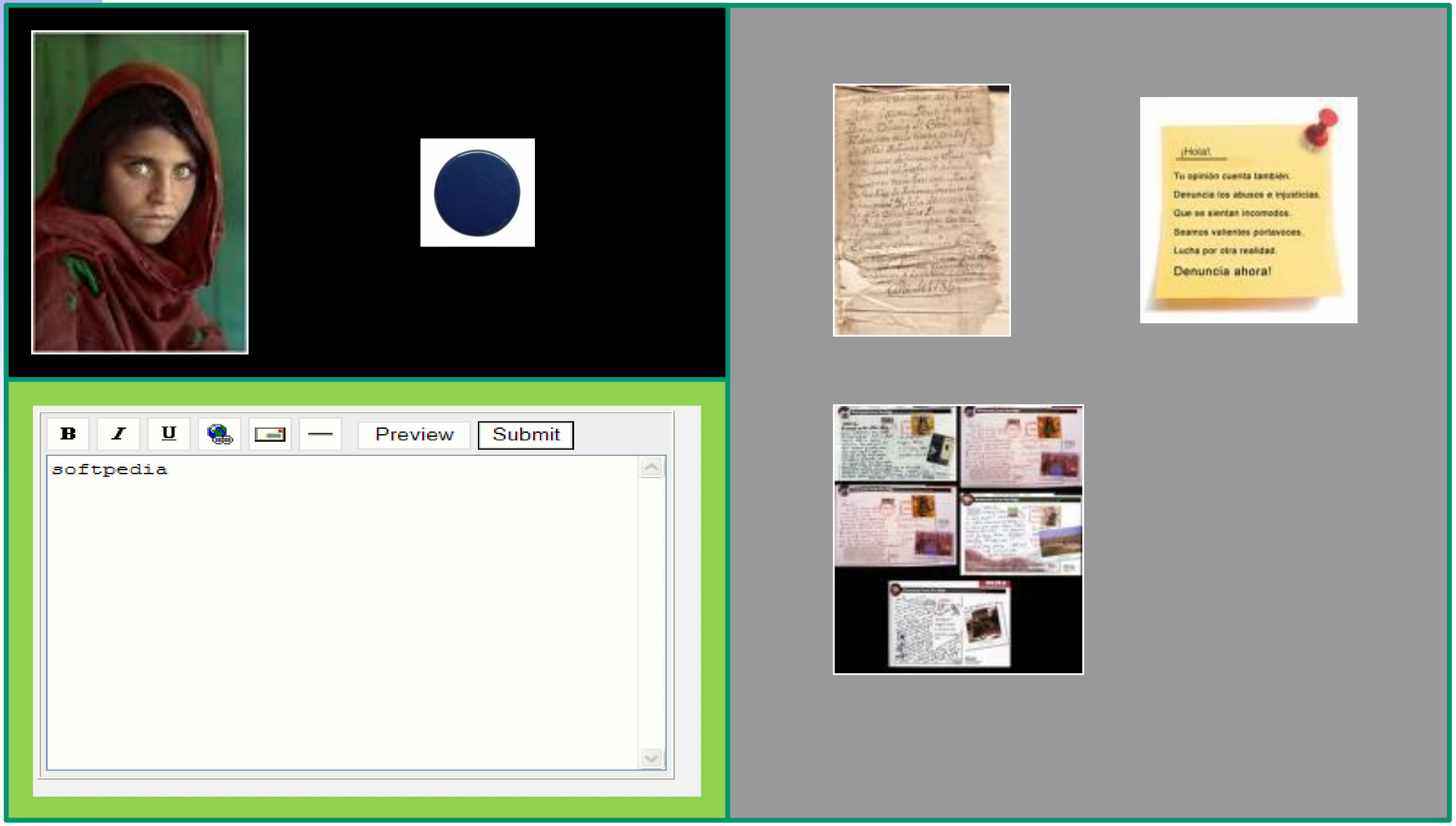
... A good time to take a look at the Java API, in order to get to know where to find information on the different graphical components and how to use them...

And how can I attach “corks to the cork”?

- We will use CONTENT PANELS:
  `JPanel`
- They are medium level containers:
  - They simplify the window organization
- A panel may contain other panels

Universidad Carlos III de Madrid

Panel hierarchy



softpedia

¡Hola!

Tu opinión cuenta también.

Denuncia los abusos e injusticias.

Que se sientan incomodos.

Seamos valientes portavoces.

Lucha por otra realidad.

Denuncia ahora!

Universidad
Carlos III de Madrid

Example of panel

```
JButton button;
JLabel label;
JPanel panel;

public Example() {

      panel = new JPanel();
      getContentPane().add(panel);

      label = new JLabel("A label");
      button = new JButton("A button");
      button.setSize(100, 70);
      panel.add(button);
      panel.add(label);

}
```

How can I put together all those components?

- You can use either coordinates…

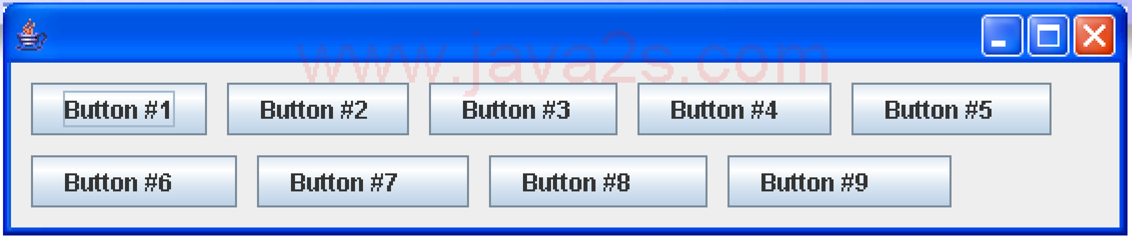
```
label.setBounds(100, 70, 50, 50);
```

- …or LAYOUTS:
  - These are like templates to organize graphical components.
  - They are associated to panels.
  - We'll see three types here.
- To use coordinates you have to neutralize the layout first:

```
panel.setLayout(null)
```

- You need to import `java.awt.*` in order to use layouts!

Example using FlowLayout



```
JButton button;
JLabel label;
JButton otherButton;
JPanel panel;

public Example() {
  panel = new JPanel();
  getContentPane().add(panel);

  label = new JLabel("A label");
  button = new JButton("A button");
  otherButton = new JButton("Other button");
  panel.add(button);
  panel.add(label);
  panel.add(otherButton);
}
```

It places the elements in a row, one after the other

FlowLayout is the default! (in panels)

Universidad Carlos III de Madrid

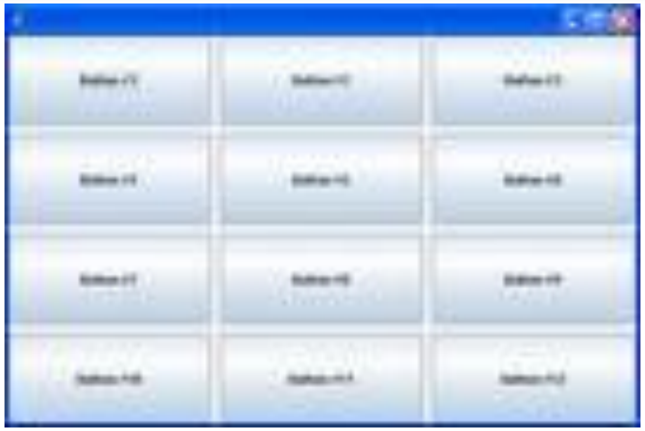
Example using GridLayout

```
public Example() {

   panel = new JPanel();
   panel.setLayout(new GridLayout(2, 2));
   getContentPane().add(panel);

   label = new JLabel(“A label");
   button = new JButton(“A button");
   otherButton = new JButton(“Other button");
   panel.add(button);
   panel.add(label);
   panel.add(otherButton);

}
```



It places the elements in a grid

Example with BorderLayout

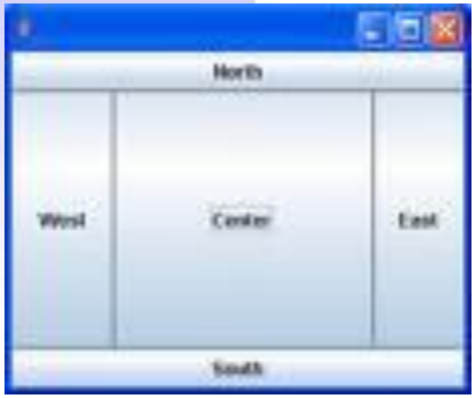
It is the default in high level containers

```
public Example() {

   panel = new JPanel();
   panel.setLayout(new BorderLayout());
   getContentPane().add(panel);

   label = new JLabel(“A label");
   button = new JButton(“A button");
   otherButton = new JButton(“Other button");
   panel.add(button, BorderLayout.SOUTH);
   panel.add(label, BorderLayout.WEST);
   panel.add(otrobutton, BorderLayout.NORTH);

}
```



It divides the container in five sections:
North, south, east, west and center

Universidad Carlos III de Madrid

Systems Programming

Events

Julio Villena Román (LECTURER)

<jvillena@it.uc3m.es>

CONTENTS ARE MOSTLY BASED ON THE WORK BY:

José Jesús García Rueda

Session objectives

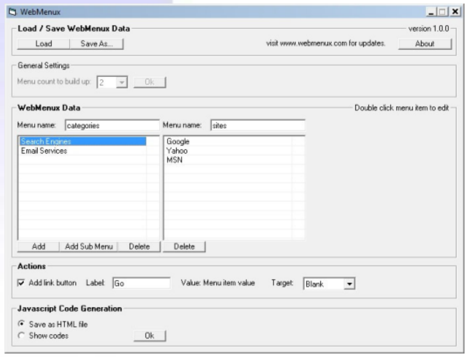
- Being able to add behaviour to the graphical elements in the interface…
- …modifying those elements as a result of the actions on them
- In other words, to cover the whole cycle:
  1. Receiving events that take place on the graphical elements
  2. Processing them
  3. Showing feedback on the screen

Graphical application architecture

Interface

Processing

Persistence



WebMenux

Load / Save WebMenux Data — version 1.0.0

Load | Save As... — visit www.webmenux.com for updates. — About

General Settings

Menu count to build up: 2 — Ok

WebMenux Data — Double click menu item to edit

Menu name: categories — Menu name: sites

Search Engines	Google
Email Services	Yahoo
	MSN

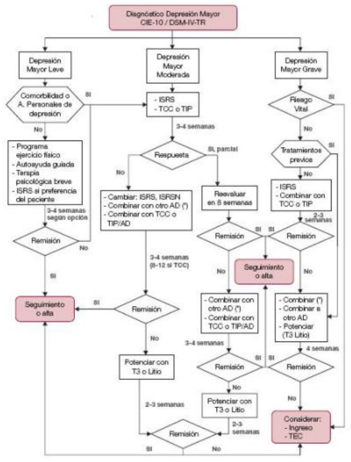
Add | Add Sub Menu | Delete — Delete

Actions

[✓] Add link button — Label: Go — Value: Menu item value — Target: Blank

Javascript Code Generation

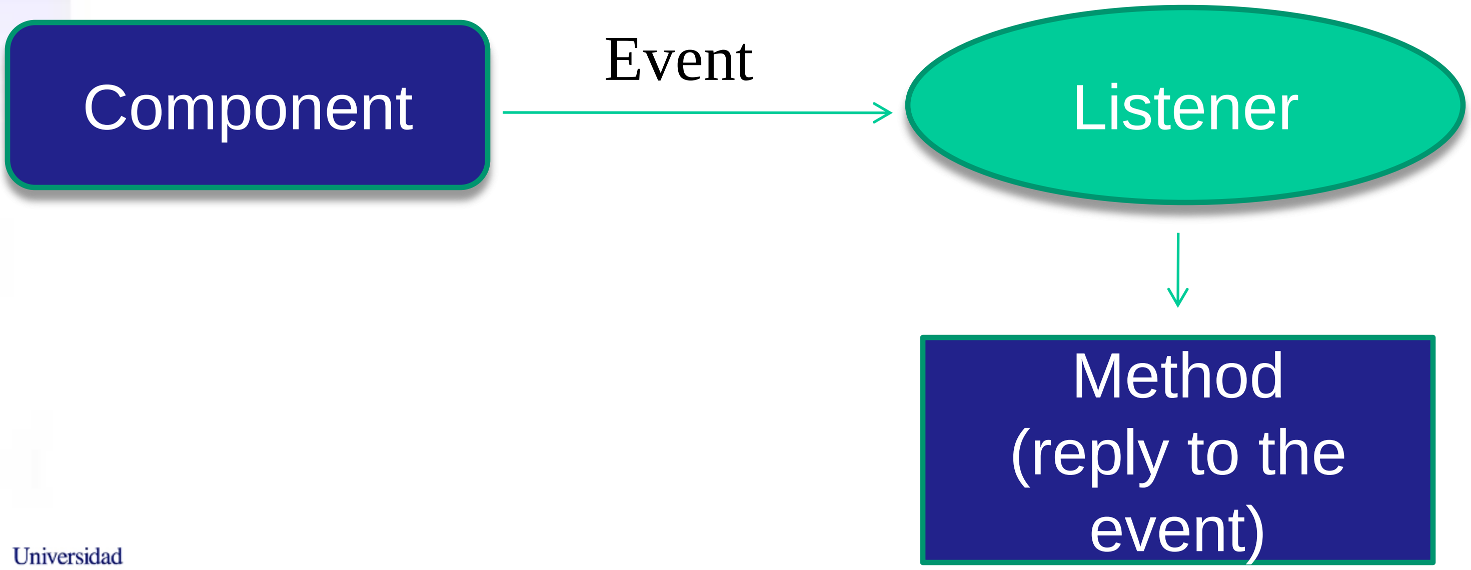
[✓ Save as HTML file] Show codes — Ok



How is this link created?

Anybody listening?

- When users act on the interface, something should happen.
- For being so, we will have to program **events managers** (listeners)



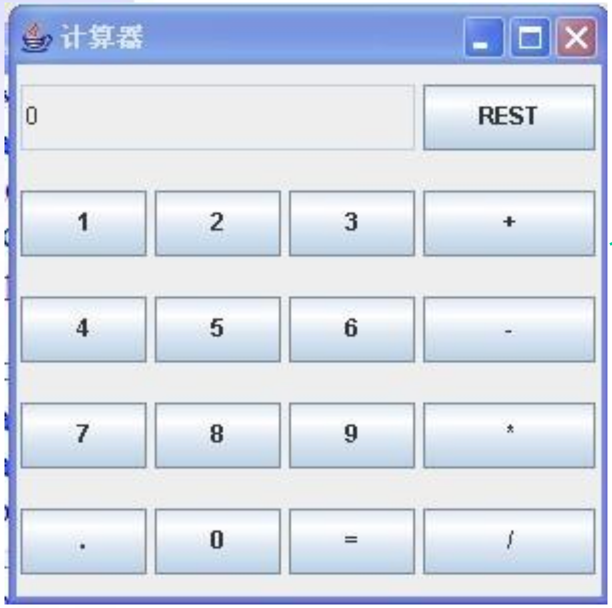
Universidad Carlos III de Madrid

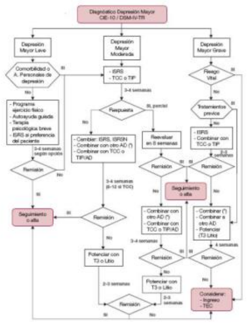
Examples of listeners

- **`WindowListener`**
  - For managing window events
- **`ActionListener`**
  - For managing buttons and other simple components events
- You will have to consult the API constantly!

Active waiting

- Once the GUI is “painted” on the screen…
- … the program stays in a “stand-by” mode, not running any active code





When something happens on the interface, the associated listener wakes up

And translated into code?

This package includes the listeners

Listeners are interfaces, usually

```
import java.awt.event.*;

public class ListenerExample implements ActionListener {

    public void actionPerformed (ActionEvent e) {

        System.out.println(“Inside the listener");

    }

}
```

This method is awaken automatically

Who listens to whom?

- If we have several graphical components…
- …and we can create as many listeners as we wish…
- Who listens to whom?

- We'll have to associate, explicitly, the listeners to the components
- Multiple combinations are possible:
  - Several listeners associated to the same component.
  - One listener associated to several components.

How to set up the association?

```
import javax.swing.*;
Import java.awt.event.*;

public class Example2 extends JFrame {

   JButton myButton = new JButton ("Click here");
   ListenerExample myListener = new ListenerExample();

   public Example2 () {
         getContentPane().add(myButton);
         myButton.addActionListener(myListener);
   }

   public static void main (String[] arg) {
         Example2 window = new Example2();
         window.setSize(200, 200);
         window.setVisible(true);
      }
}
```

Creating an instance of the corresponding listener

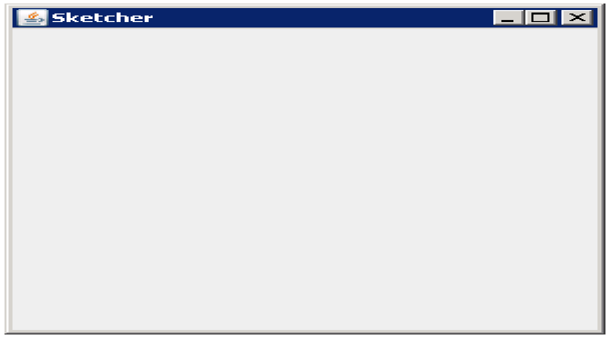
Associating the listener to the component

Which part of the listener is awaken?

- Listeners have **different methods** to listen to different events.
- Java **automatically** invokes the suitable method, depending on the event.
- The body of these methods will be programmed by us. We can invoke other methods from these.
- When the method running is over, the program moves on to stand-by again, awaiting for new events.
- These methods receive an **event** object as argument.

Example: WindowListener

- Among its methods we find:
  - `void windowClosing (WindowEvent evt)`
  - `void windowOpened (WindowEvent evt)`
  - `void windowClosed (WindowEvent evt)`
  - `void windowIconified (WindowEvent evt)`
  - `void windowDeiconified (WindowEvent evt)`
  - `void windowActivated (WindowEvent evt)`
  - `void windowDeactivated (WindowEvent evt)`



Universidad Carlos III de Madrid

May I get more information about an event?

- The event received as an argument by the listeners' methods is provided automatically by Java
- "**Asking**" to that event object we can find out more things about what really happened
- Asking, as always, is done by invoking methods of the event object

Example

Argument provided by Java automatically

```
import java.awt.event.*;

public class ListenerExample implements ActionListener {

    public void actionPerformed (ActionEvent e) {

        String source = e.getActionCommand();
        System.out.println("Button: " + source);

    }

}
```

It gives back the label of the component that started the event

Universidad Carlos III de Madrid

Event oriented programming

- GUIs in Java is just an example of a more general and very important programming technique: Events Oriented Programming
- In a program everything is sequential: the time when each action is going to happen is predictable…
- …How can we take into account those events in the world outside our program that we don't know exactly when will happen?
  - When will that door open?
  - When will this pot of water boil?
  - When will the user push this button?
- Programs have mechanisms to react ("wake up") when specific events take place outside the program

Code organization

- Everything explained about GUIs is under the principles and rules of the OO programming paradigm…
- …so everything we know about OO up to now is perfectly valid here
- We have just added new pieces to the meccano…
  - …that can be mixed with the rest in the way we consider most suitable.
- Examples:
  - Creating the listeners as independent classes
  - Creating the listeners as inner classes
  - Making the graphical components themselves act as listeners
  - Associating a listener to more than one graphical component

Adapters

- Some listeners interfaces have lots of methods…
- …and we will have to implement them all (listeners are interfaces)
- Adapters are classes that implement all the methods of a specific listener
- Being classes, we just have to extend them rewriting the methods we need
- For every Listener interface, there is an Adapter class:
  - `WindowListener` → `WindowAdapter`
  - `KeyListener` → `KeyAdapter`
  - `MouseListener` → `MouseAdapter`

"We're all ears!"