# IPM 11/12 – P4
# Handling Events in AWT

Licenciatura em Ciência de Computadores

*Miguel Tavares Coimbra*

# Why *Events*?

- Components need a way to:
  - Receive messages.
  - Send messages.

- Examples:
  - A TextField needs to know that a key was pressed.
  - A destroy Button wants to tell the corresponding window that it was pressed.

In Java AWT we can use *Events*

# What are AWT *Events*?

- Like everything in java, an *Event* is an *Object.*
  - java.lang.Object
    - java.util.EventObject
      - java.awt.AWTEvent

- Based on a Consumer / Producer framework.
  - Producer generates *Event* objects.
  - Consumers register as listeners of *Events*.

# Example

Who are these?

Listener1

Produces the *Event*

Button → Event

Listener2

Listener3

Sends it to Listeners

Who is the first to process the event?
Do all listeners process it?

...
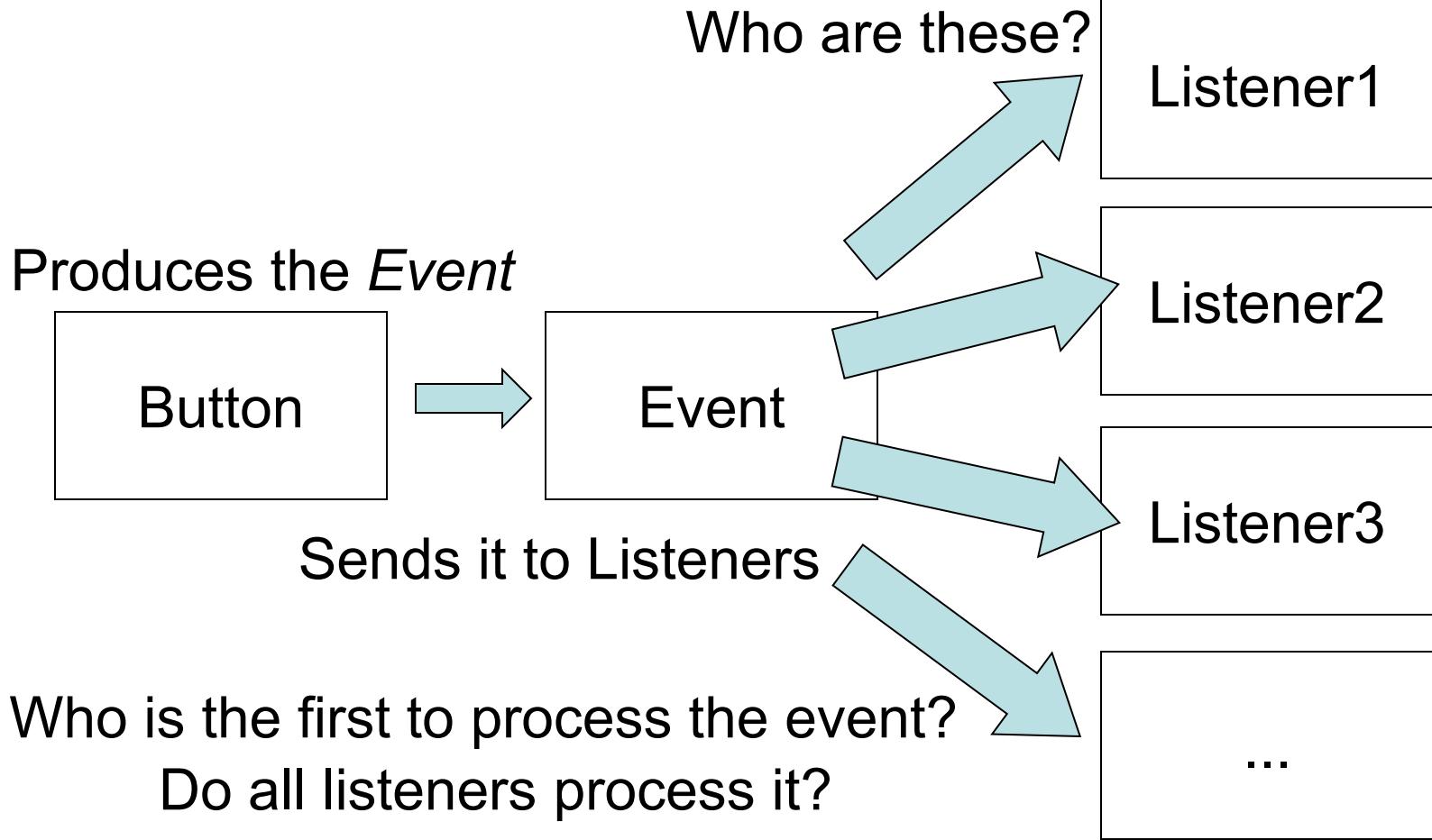
# Adding Listeners

- Producers add their Listeners
  - And not the other way around!
- Function: *AddListener(listener);*
- Example:

  Button mybutton = new Button("Ok");
  mybutton.addActionListener(this);
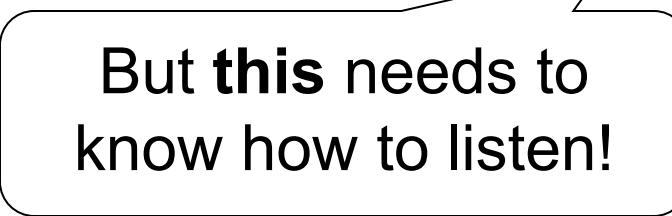
  > But **this** needs to know how to listen!

U.PORTO

# More examples

```
Button button = new Button("Ok");
ActionListener ac = new ActionListener()
button.addActionListener(ac);
```

```
Button button = new Button("Ok")
button.addActionListener(new ActionListner() { ... });
```

# Can I be a *Listener*?

- Yes....
  - But you need to implement the listener interface.
    - What is an interface?
- This way the *Event* object can be received as a parameter.

```
public class Contador extends Frame implements ActionListener
{
        public void actionPerformed(ActionEvent e) {
                // Insert your killer code here
        }
}
```

# A more complex example

- Imagine you have a TextField object.
  - You want to block every keypress that does not correspond to a number.

  Class myTextField extends TextField (

  ....

  )
- **Pause: What does extend mean?**

# Inheritance

- **Object-Oriented Programming**

  - Inheritance

  - *Child* classes inherit methods and properties of *parent* classes.

    class child extends parent {

      // killer code here

    }

- **Great way to reuse and expand code.**

  - Also means we must **always** use good programming practices since we might later reuse this code.

# Back to our example

- Imagine you have a TextField object.
  - You want to block every keypress that does not correspond to a number.

  Class myTextField extends TextField {

  ....

  }
- So **myTextField** inherits all methods and properties of **TextField**.

# Managing Events

- Now myTextField wants to listen to KeyEvents.

- Need to implement the KeyListener interface:

```
public class myTextField extends TextField implements KeyListener
 {
         public void keyTyped(KeyEvent e) { // Do Something }
         public void keyPressed(KeyEvent e) { // Do Something }
         public void keyReleased(KeyEvent e) { // Do Something }
 }
```

# Consuming Events

I can then implement code that ignores non-numerical key presses.

- Which key was pressed?
    - Check the KeyEvent object!
- Who pressed it?
    e.getSource();
- But my Event will then be passed to TextField class.
    - Consume the event
    *e.consume();*

# Types of Events

- ActionEvent – Button, MenuItem, TextField
- ItemEvent – Checkbox, CheckboxMenuItem, Choice
- AdjustmentEvent – Scrollbar
- WindowEvent – Dialog, Frame
- KeyEvent – Any component
- FocusEvent – Any component
- ContainerEvent – Container
- TextEvent - Text

# ActionEvent example

```java
import java.applet.Applet;
import java.awt.Button;
public class TestButton extends Applet {
    private Button button = null;

    public TestButton() { super(); }

    public void init() {
            button = new Button();
            button.setLabel("Press me!");
            button.addActionListener(new java.awt.event.ActionListener() {
                    public void actionPerformed(java.awt.event.ActionEvent e) {
                            button.setLabel("OK"); }
                    });
            add(button);
            }
    }
```

# TextEvent example

```
import java.applet.Applet;
import java.awt.Label;
import java.awt.TextField;

public class TestTextField extends Applet {
    private TextField textField = null;
    private Label label = null;

    public void init() {
            label = new Label("Password:");
            textField = new TextField();
            textField.setColumns(10);
            textField.setEchoChar('*');
            textField.addTextListener(new java.awt.event.TextListener() {
                    public void textValueChanged(java.awt.event.TextEvent e) {
                            label.setText(textField.getText()); } });
            textField.addActionListener(new java.awt.event.ActionListener() {
                    public void actionPerformed(java.awt.event.ActionEvent e) {
                            label.setText("Password"); } });
            this.add(label, null);
            this.add(textField, null);
            }
    }
```

# Summing up

- I can reuse and improve previous classes

  NewClass extends OldClass

- For a class to implement an interface, it must implement all the defined methods.

  NewClass implements AmazingInterface

- Events are:

  - Produced by components.

  - Consumed by registered Listeners.

# Resources

1. Developer Resources for Java Technology
   http://java.sun.com/

2. Essentials of the Java programming language
   http://java.sun.com/developer/onlineTraining/Programming/BasicJava1/