Unit 8. Events in Swing-based GUIs

Software Analysis and Design Project

Universidad Autónoma de Madrid

Event-based programming

- Another programming paradigm: one very suitable for GUIs
- A program “execution” through a GUI does not follow a strictly sequential flow
- The user has the freedom to decide the next step at any given time, chosing from alternatives offered by the program
- It would very difficult to capture all possible paths of execution in a traditional program (based on conditionals, iterations,…)

Event-based Programming

- Often used in window-based GUI as well as in web-enhanced applications (Flash, Java/JFX, Silverlight)
- The user takes the initiative, rather than de program
- Each program gets divided into modules associated with independent windows or other graphical components
- The **components just wait for user´s actions**
- **User actions generate events,** which are queued for processing
- An event-based system gest events from the queue and send them to the corresponding program unit
- Each **program processes the events it receives** by giving a certain response action that depends on the event received
- **Each type of component is characterized by its own way of reacting to events**

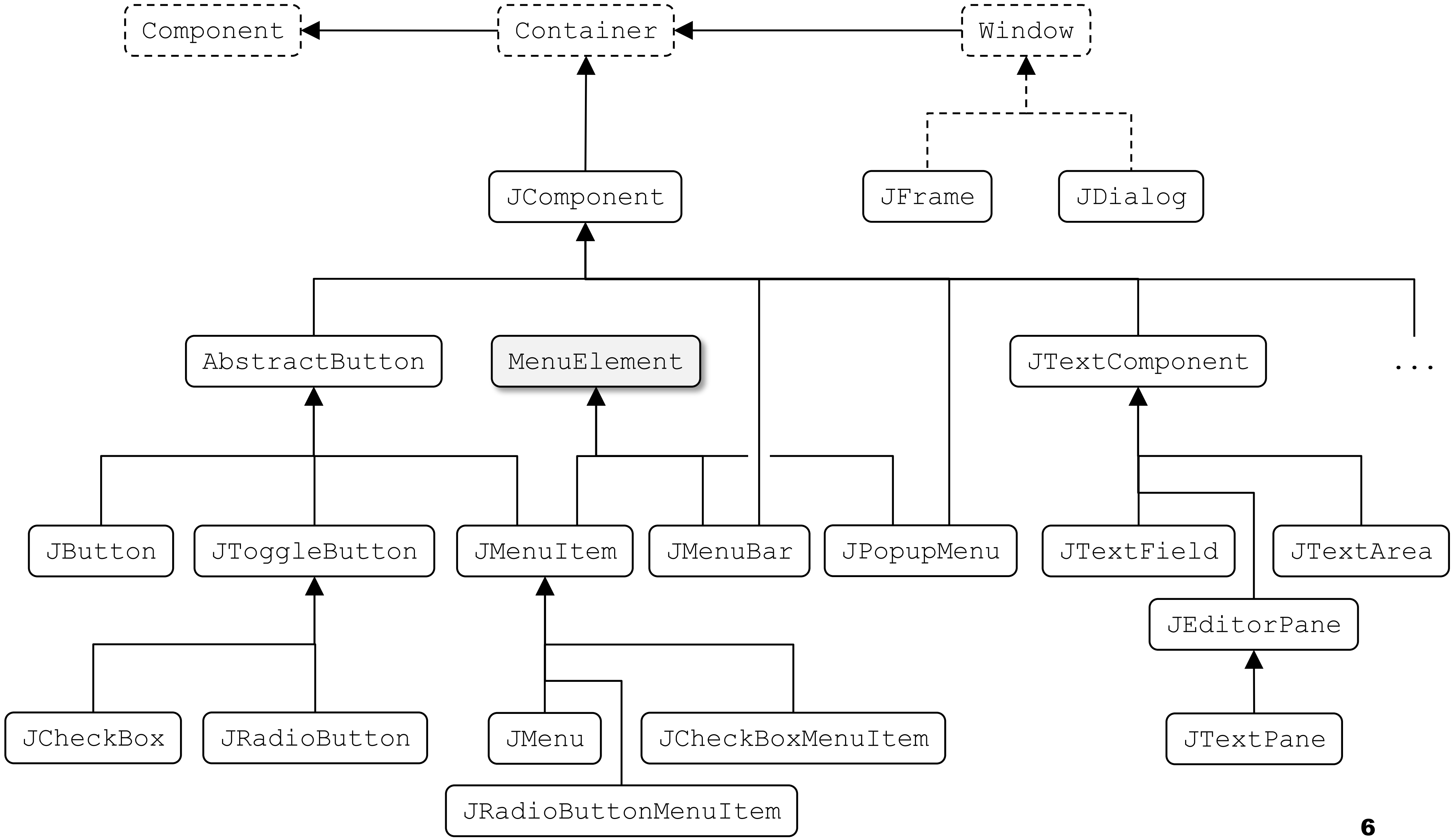
Libraries: JFC/Swing/AWT

- Packages: `javax.swing`, **`java.awt.event`**, `java.awt`
- Components
  - Predefined components
  - Aggregating components
  - Interfaces draw themselves: drawing functions
  - Creation of customized new components
- **User Interaction thru event handling**
  - **Firing events**
  - **Capturing and processing events**
- Layout of components

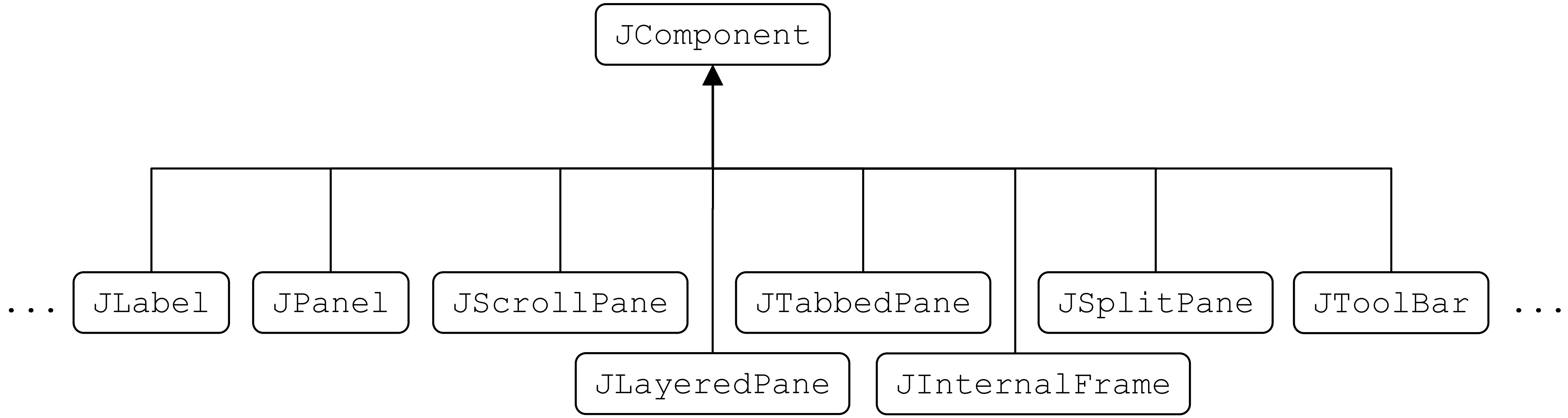
GUI Construction Steps

- Compose interfaces combining predefined classes
  - The `Container` class, to add components to containers
  - Control the visual aspect of components setting their state (visibility, color, alignment, …)
- Define the ubication of a container´s components
  - Absolute coordinates
  - Layout managers
- **Managing events: an emission/reception model**
  - **Managing events generated by predefined class as consequence of user´s actions**
  - **Directly managing user´s input**
- Defind personalized components
  - The `Graphics` class
  - Using low-level drawing functions

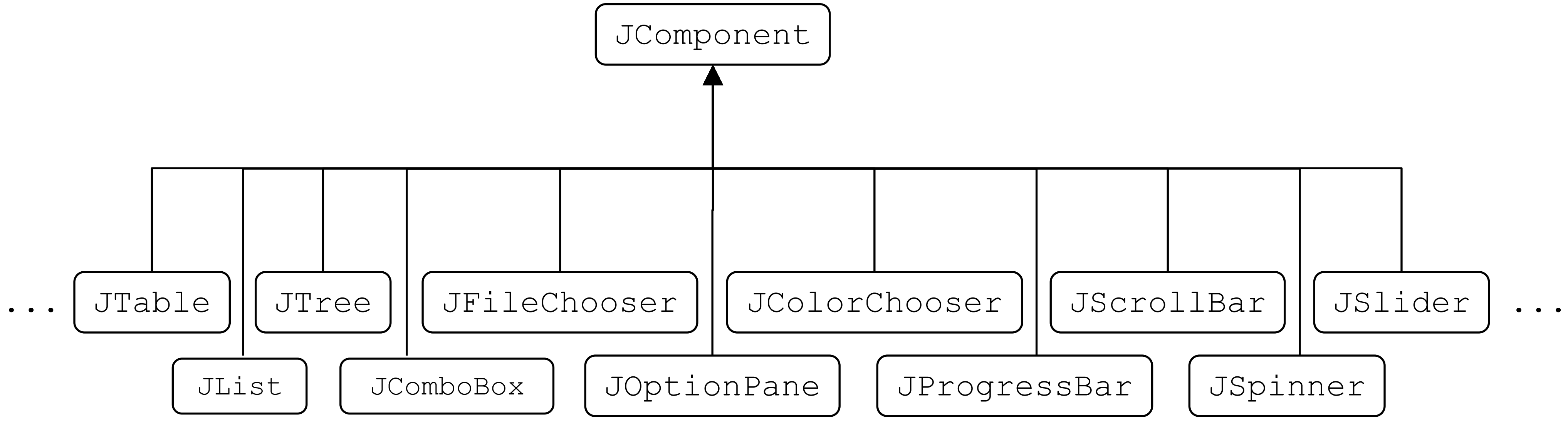
Hierarchy of Swing components



Hierarchy of Swing components



Hierarchy of Swing components



Interacting with the User

Event management

Reacting to events: an example

```
class MyWindow extends JFrame implements MouseListener {
 MyWindow () { addMouseListener (this); }

  public void mouseClicked (MouseEvent e) {
    System.out.println ("The user just clicked on me");
  }

  public void mouseEntered (MouseEvent e) {}
  public void mouseExited (MouseEvent e) {}
  public void mousePressed (MouseEvent e) {}
  public void mouseReleased (MouseEvent e) {}
}
```

Reacting to Events

- Implement the interface listener corresponding to each type of event
  - It exists a correspondence: event type $\rightarrow$ listener type
- Implement all methods of the interface
  - Each method corresponds to a particular kind of event
  - Classes implementing listeners may be Swing components or other classes. Programmers can chose the best alternative
  - These methods must execute as quickly as possible. Otherwise, it may be necessary to use parallel threads
- The listener must register itself as listerner of the emisor
  - Each type of component can generate certain types of events
- The Swing/AWT System handles the rest

The event model

- Events are objects of subclasses extending `AWTEvent`
- Events are generated when:
  - User generates a direct input: `MouseEvent`, `KeyEvent`
  - User acts upon a widget: `ActionEvent`, `ItemEvent`, `AdjustmentEvent`
  - User changes a window: `WindowEvent`
  - Other causes: `ContainerEvent`, `ComponentEvent`, `PaintEvent`, etc.
- Events are generated in the context of a specific component: **emisor**
- Other components can register to receive different types of events generated an emisor: **receptors**
- To be a receptor of a type of events, a class must implement the corresponding **listener** interface
- Events execute in a special thread for event management. Thus, the method for handling an event does not excecute until completion of the method for handling the previous event

Elements involved when processing events of a given type

For each type of event such as *xxx*`Event` there will be:

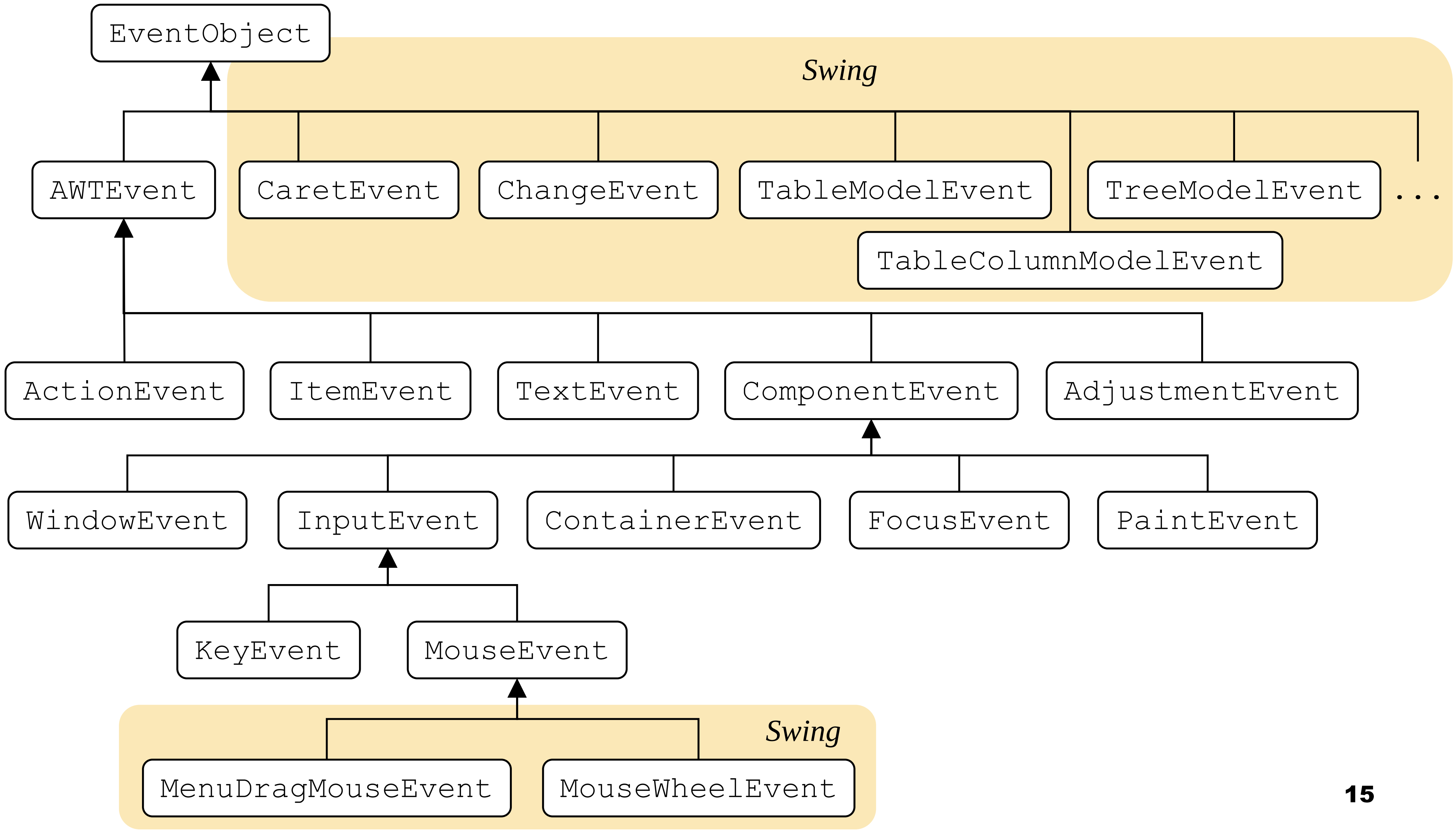
- One type of listener *xxx*`Listener` (except that `MouseEvent` has 2)
- A list of component clases that can generate events of this type
- A method `add`*xxx*`Listener` to register listeners for events of this type
  - This method is defined in classes of components that can generate events of this type
  - A component can only register listeners for the types of events that the component itself can generate

Example

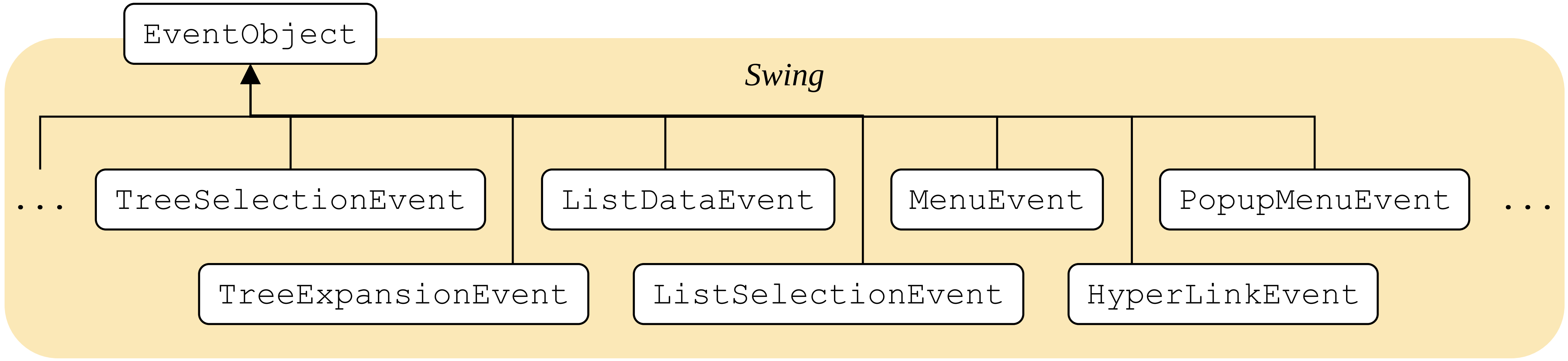
Event class:	`ActionEvent`
Object that generate it:	`JButton`, `JMenuItem`, `JCheckBox`, `JRadioButton`, `JComboBox`, `JTextField`
Listener interface:	`ActionListener`
Methods to implement in the listener class:	`actionPerformed (ActionEvent)`
Method to register as a listener:	`addActionListener (ActionListener)`

A component can only register listeners for the types of events that the component itself can generate

Classes of events: `java.awt.event`



Classes of events: `javax.swing.event`



Events generated by each class

	Mouse	MouseMotion	Key	Action	Window	Document	Item	Contaner	Component	Adjustment	Focus
JComponent	●	●	●						●		●
JLabel	●	●	●						●		●
JButton	●	●	●	●					●		●
JCheckBox	●	●	●	●			●		●		●
JComboBox	●	●	●	●			●		●		●
JList	●	●	●						●		●
JTextField	●	●	●	●		●			●		●
JTextArea	●	●	●			●			●		●
JTextComponent	●	●	●			●			●		●

Events generated by each class

	Mouse	MouseMotion	Key	Action	Window	Document	Item	Container	Component	Adjustment	Focus
JScrollBar	●	●	●						●	●	●
JMenuItem	●	●	●	●					●		●
JCheckBoxMenuItem	●	●	●	●			●		●		●
JRadioButtonMenuItem	●	●	●	●			●		●		●
Container	●	●	●					●	●		●
JPanel	●	●	●					●	●		●
JScrollPane	●	●	●					●	●		●
Window	●	●	●		●			●	●		●
JFrame	●	●	●		●			●	●		●
JDialog	●	●	●		●			●	●		●

Methods to include in each listener (1/3)

- **`MouseListener`**
  - `mouseClicked(MouseEvent)`
  - `mousePressed(MouseEvent)`
  - `mouseReleased(MouseEvent)`
  - `mouseEntered(MouseEvent)`
  - `mouseExited(MouseEvent)`
- **`MouseMotionListener`**
  - `mouseMoved(MouseEvent)`
  - `mouseDragged(MouseEvent)`
- **`KeyListener`**
  - `keyTyped(KeyEvent)`
  - `keyPressed(KeyEvent)`
  - `keyReleased(KeyEvent)`
- **`ActionListener`**
  - `actionPerformed(ActionEvent)`

Methods to include in each listener (2/3)

- **`ItemListener`**
  - `itemStateChanged(ItemEvent)`
- **`ListSelectionListener`**
  - `valueChanged(ListSelectionEvent)`
- **`DocumentListener`**
  - `insertUpdate(DocumentEvent e)`
  - `removeUpdate(DocumentEvent e)`
  - `changedUpdate(DocumentEvent e)`
- **`WindowListener`**
  - `windowActivated(WindowEvent)`
  - `windowDeactivated(WindowEvent)`
  - `windowOpened(WindowEvent)`
  - `windowClosing(WindowEvent)`
  - `windowClosed(WindowEvent)`
  - `windowIconified(WindowEvent)`
  - `windowDeiconified(WindowEvent)`

Methods to include in each listener (3/3)

- **`ContainerListener`**
  - `componentAdded(ContainerEvent)`
  - `componentRemoved(ContainerEvent)`
- **`ComponentListener`**
  - `componentShown(ComponentEvent)`
  - `componentHidden(ComponentEvent)`
  - `componentMoved(ComponentEvent)`
  - `componentResized(ComponentEvent)`
- **`AdjustmentListener`**
  - `adjustmentValueChanged(AdjustmentEvent)`
- **`FocusListener`**
  - `focusGained(FocusEvent)`
  - `focusLost(FocusEvent)`
- `...`

Contents of the event classes

Many event classes include:

- Constants (static final variables)
  - The identifying ID if the different events of the class

    e.g. `MouseEvent.MOUSE_MOVED`, `KeyEvent.KEY_RELEASED`
  - Constants representing certain properties of events

    (corresponding to values returned by methods accessing events)

    For instance: `ItemEvent.SELECTED`, `ItemEvent.DESELECTED`
- Methods
  - Return additional information about the event

    such as: `getX()`, `getY()` for `MouseEvent`, `getKeyChar()` for `KeyEvent`, `getID()` for `AWTEvent`

Information included within events (1/2)

- **`AWTEvent`**
  - `getID(), getSource(), toString()`
- **`InputEvent`**
  - `getWhen(), isShiftDown(), isControlDown(), isAltDown()`
  - `getModifiers()` → `BUTTON1_MASK, BUTTON2_MASK, BUTTON3_MASK`
- **`MouseEvent`**
  - `getClickCount(), getX(), getY()`
- **`KeyEvent`**
  - `getKeyChar(), getKeyString()`
- **`ActionEvent`**
  - `getActionCommand()` → `String`
  - `getModifiers()` → `ALT_MASK, CTRL_MASK, META_MASK, SHIFT_MASK`
- **`WindowEvent`**
  - `getWindow()`

Information included within events (2/2)

- **`ItemEvent`**
  - `getItem()` → `Object (String ó Integer)`, `getItemSelectable()`
  - `getStateChange()` → `SELECTED`, `DESELECTED`
- **`DocumentEvent`**
  - `getDocument()` → `Document`
- **`ContainerEvent`**
  - `getChild()`, `getContainer()`
- **`ComponentEvent`**
  - `getComponent()`
- **`AdjustmentEvent`**
  - `getValue()`, `getAdjustable()`
  - `getAdjustmentType()` → `UNIT_INCREMENT`, `UNIT_DECREMENT`, `BLOCK_INCREMENT`, `BLOCK_DECREMENT`, `TRACK`
- **`FocusEvent`**
  - `getOppositeComponent()`, `isTemporarty()`, `paramString()`

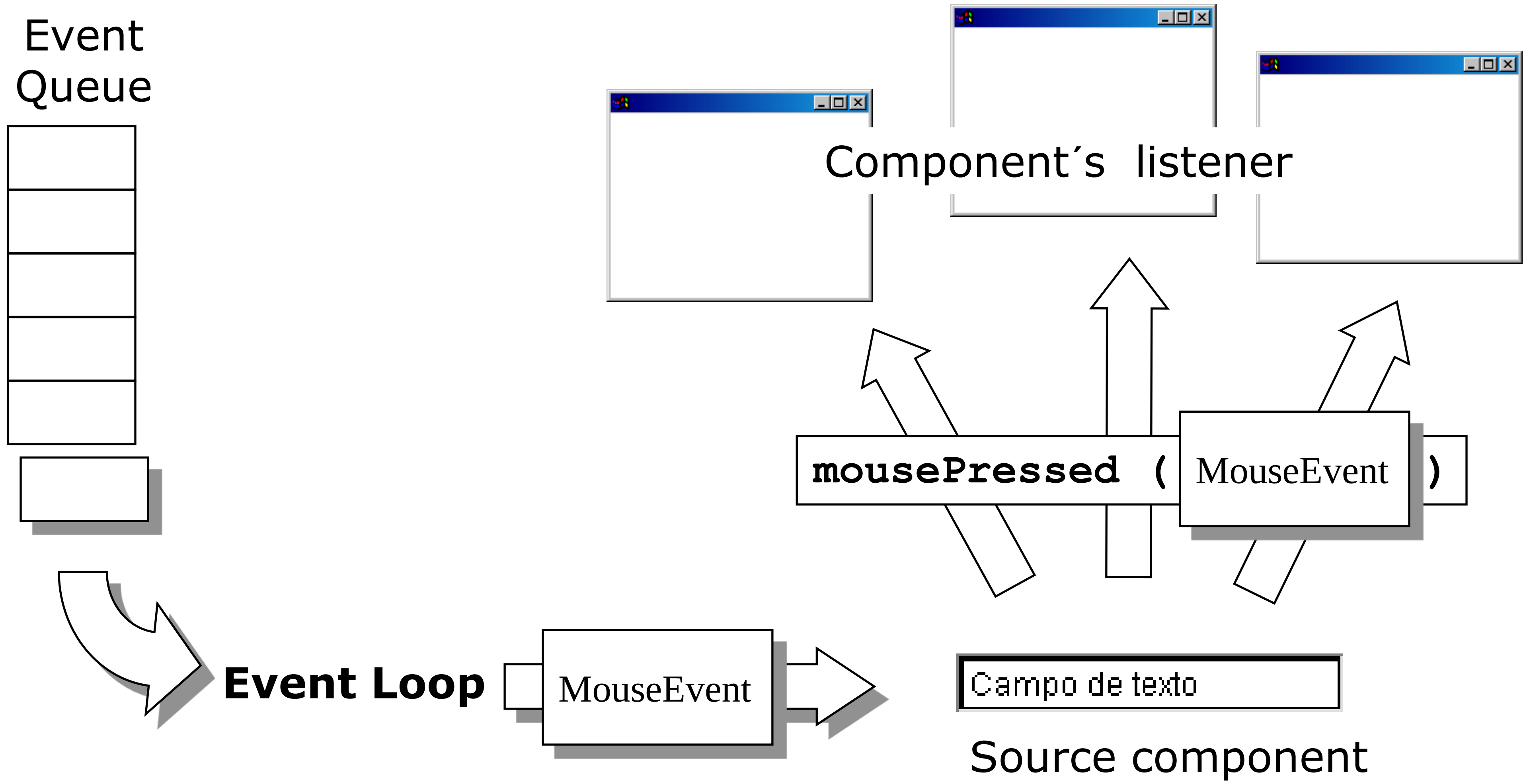
What must event handlers do?

- Modify aspect or features of the GUI
  - Change colors, fonts, labels, …
  - Change widgets size or ubication
  - Hide, show, add or remove components
  - Open a dialog box (dialog window)
  - etc.
- **Execute part of the application's functionality**
  - **Typically, this produces a result or change in the GUI**

To process or to ignore events?

- Low-level events that widgets encapsulate into and reformulated as higher-level events
  - Buttons: `MouseEvent` → `ActionEvent`
  - Text Widgets: `MouseEvent`, `KeyEvent` → `DocumentEvent`, `ActionEvent`
  - Selection Widgets: `MouseEvent` → `ItemEvent`, `ActionEvent`, `ListSelectionEvent`
  - etc.
- Event for component's state change: process these events inmediately or get access to the component state when required
  - `ItemEvent`, `DocumentEvent`, `ComponentEvent`, `ContainerEvent`, `AdjustmentEvent`, etc.

AWT Architecture for event processing


Event Queue
Component´s listener
mousePressed (MouseEvent)
Event Loop
MouseEvent
Campo de texto
Source component