Tema 8 Eventos en GUIs con Swing

Proyecto de Análisis y Diseño de Software

2º Ingeniería Informática

Universidad Autónoma de Madrid

Programación basada en eventos

- Un nuevo modelo de programación.
- La “ejecución” de una interfaz de usuario no sigue un control de flujo estrictamente secuencial
- El usuario tiene un alto grado de libertad en todo momento: amplio conjunto de acciones posibles
- Sería muy difícil representar todos los caminos posibles con un modelo de programación tradicional

Programación basada en eventos

- Modelo utilizado en las interfaces de usuario basadas en ventanas, y las aplicaciones web enriquecidas (Flash, Java/JFX, Silverlight)
- El usuario toma la iniciativa, no el programa
- El programa se divide en subprogramas asociados a ventanas o componentes gráficas
- Las **componentes están a la espera de las acciones del usuario**
- Las **acciones del usuario generan eventos** que se acumulan en una cola
- El sistema de eventos extrae eventos de la cola y los envía a los programas
- Los **programas procesan los eventos recibidos** respondiendo según el tipo de evento
- **Cada tipo de componente se caracteriza por una forma propia de respuesta a los eventos**

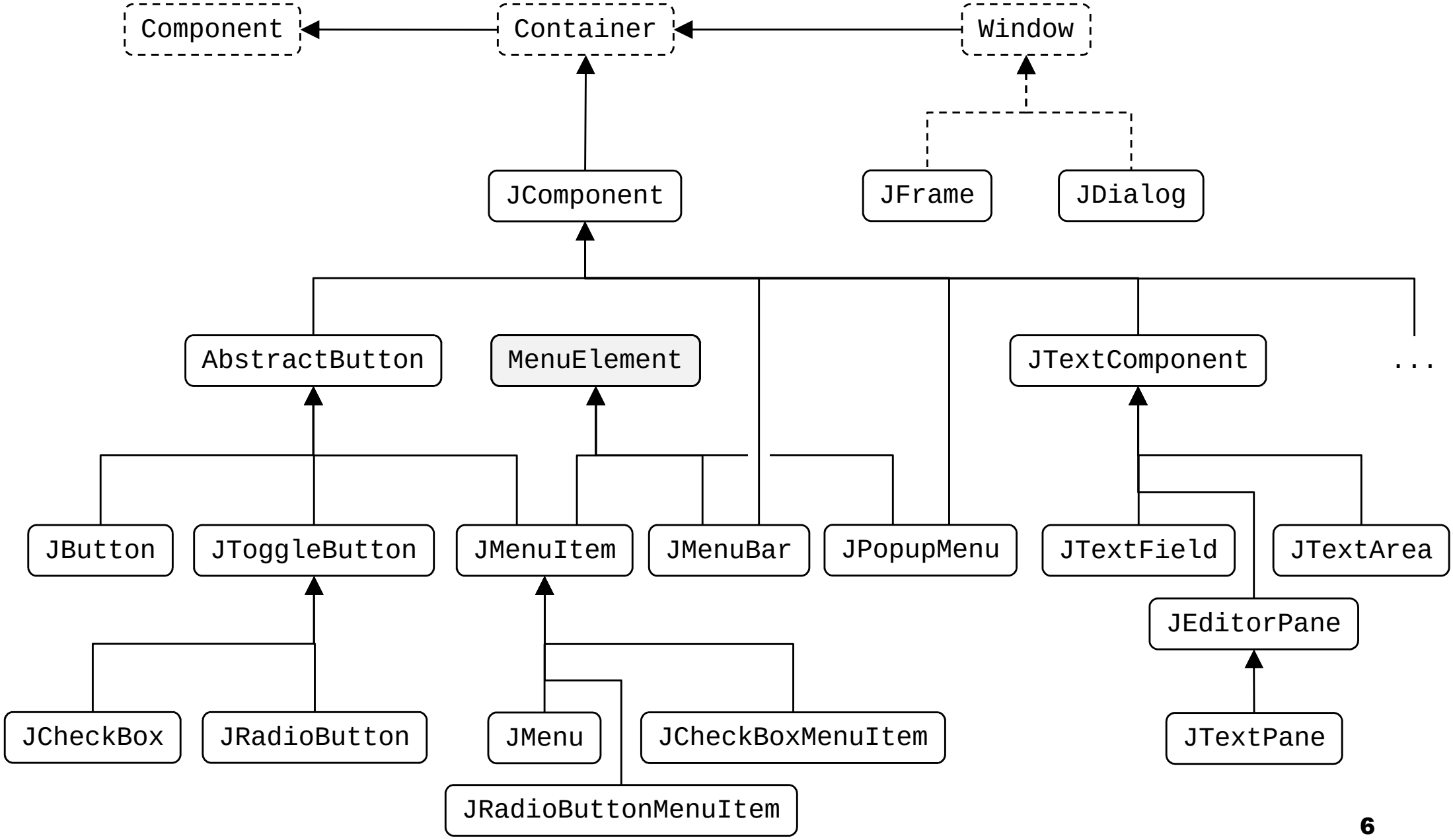
La librería JFC/Swing/AWT

- Paquetes: `javax.swing`, **`java.awt.event`**, `java.awt`
- Componentes
  - Componentes predefinidas
  - Agregación de componentes
  - Las interfaces se dibujan a sí mismas: funciones de dibujo
  - Creación de nuevas componentes
- **Interacción con el usuario: gestión de eventos**
  - **Emisión de eventos**
  - **Recepción y procesamiento de eventos**
- Layout de componentes

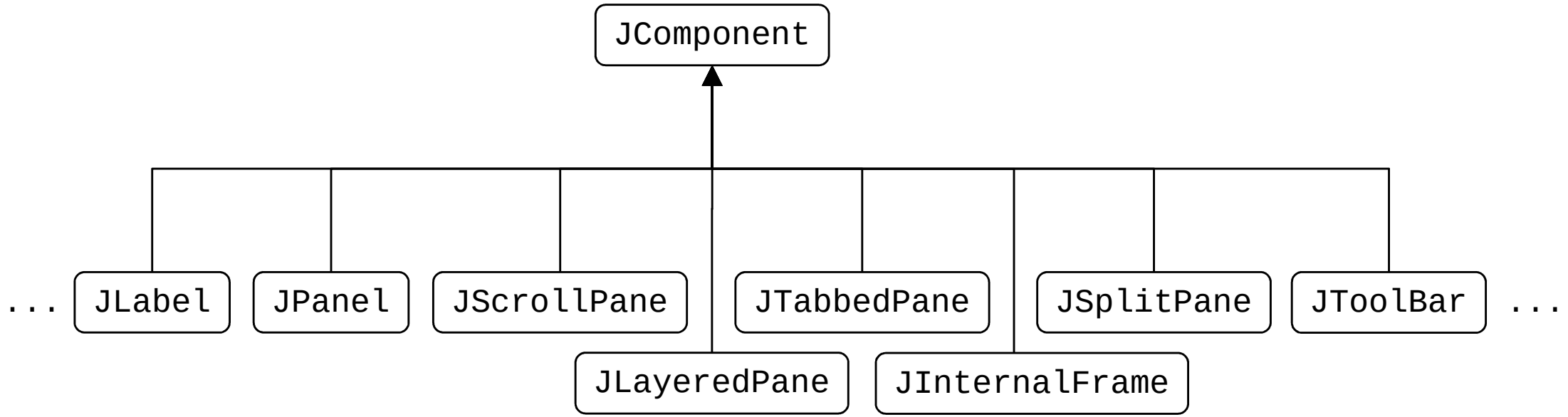
Construcción de una GUI

- Componer interfaces con las clases predefinidas
  - La clase `Container`, adición de subcomponentes
  - Control de la apariencia de las componentes manipulando su estado
- Definir la disposición de las partes de un contenedor
  - Posiciones absolutas
  - Layout managers
- **Gestionar los eventos: modelo emisor / receptor**
  - **Eventos de acción generados por las clases predefinidas**
  - **Gestión directa del input del usuario**
- Definir componentes personalizadas
  - La clase `Graphics`
  - Utilización de las funciones de dibujo

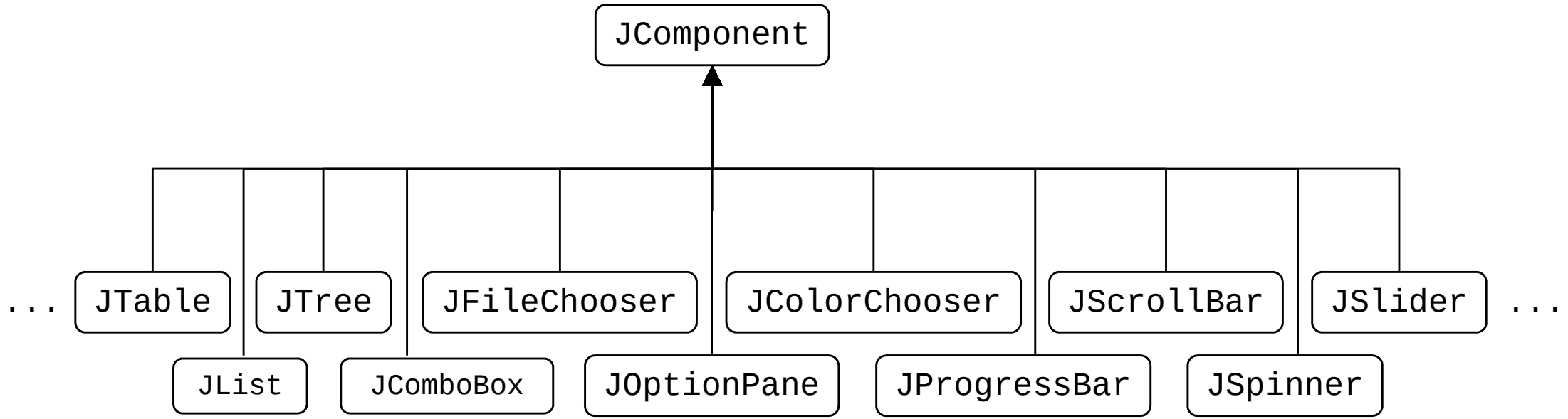
Jerarquía de componentes Swing



Jerarquía de componentes Swing



Jerarquía de componentes Swing



Interacción con el usuario

Gestión de eventos

Responder a eventos: ejemplo

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

  public void mouseClicked (MouseEvent e) {
    System.out.println ("El usuario me ha pinchado");
  }

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

Responder a eventos

- Implementar la interfaz listener correspondiente al tipo de evento
  - Existe una correspondencia de tipo de evento / tipo de listener
- Implementar todos los métodos de la interfaz
  - Cada método corresponde a una variedad del tipo de evento
  - Las clases que implementan listeners pueden ser componentes Swing, o cualquier otra clase. Esto queda a elección del programador
  - El método ha de completarse en el menor tiempo posible. Si no será necesario usar hilos
- Registrarse como listener de un emisor
  - Cada tipo de componente puede emitir cierto tipo de eventos
- El sistema Swing/AWT se ocupa del resto

El modelo de eventos

- Los eventos son objetos de distintas subclases de `AWTEvent`
- Se generan eventos cuando:
  - Se produce input del usuario: `MouseEvent`, `KeyEvent`
  - Un widget se acciona: `ActionEvent`, `ItemEvent`, `AdjustmentEvent`
  - Una ventana es manipulada: `WindowEvent`
  - Otras causas: `ContainerEvent`, `ComponentEvent`, `PaintEvent`, etc.
- Los eventos se producen en el contexto de una componente: **emisor**
- Otras componentes pueden registrarse para distintos tipo de eventos emitidos por un emisor: **receptores**
- Para ser receptora de un tipo de mensajes, una clase debe implementar la interfaz **listener** correspondiente
- Los eventos se ejecutan en un hilo especial de gestión de eventos. No se ejecuta un método de tratamiento del evento hasta que ha terminado de ejecutarse el anterior

Elementos que intervienen en el procesamiento de un tipo de evento

A cada tipo de evento *xxx*`Event` corresponde:

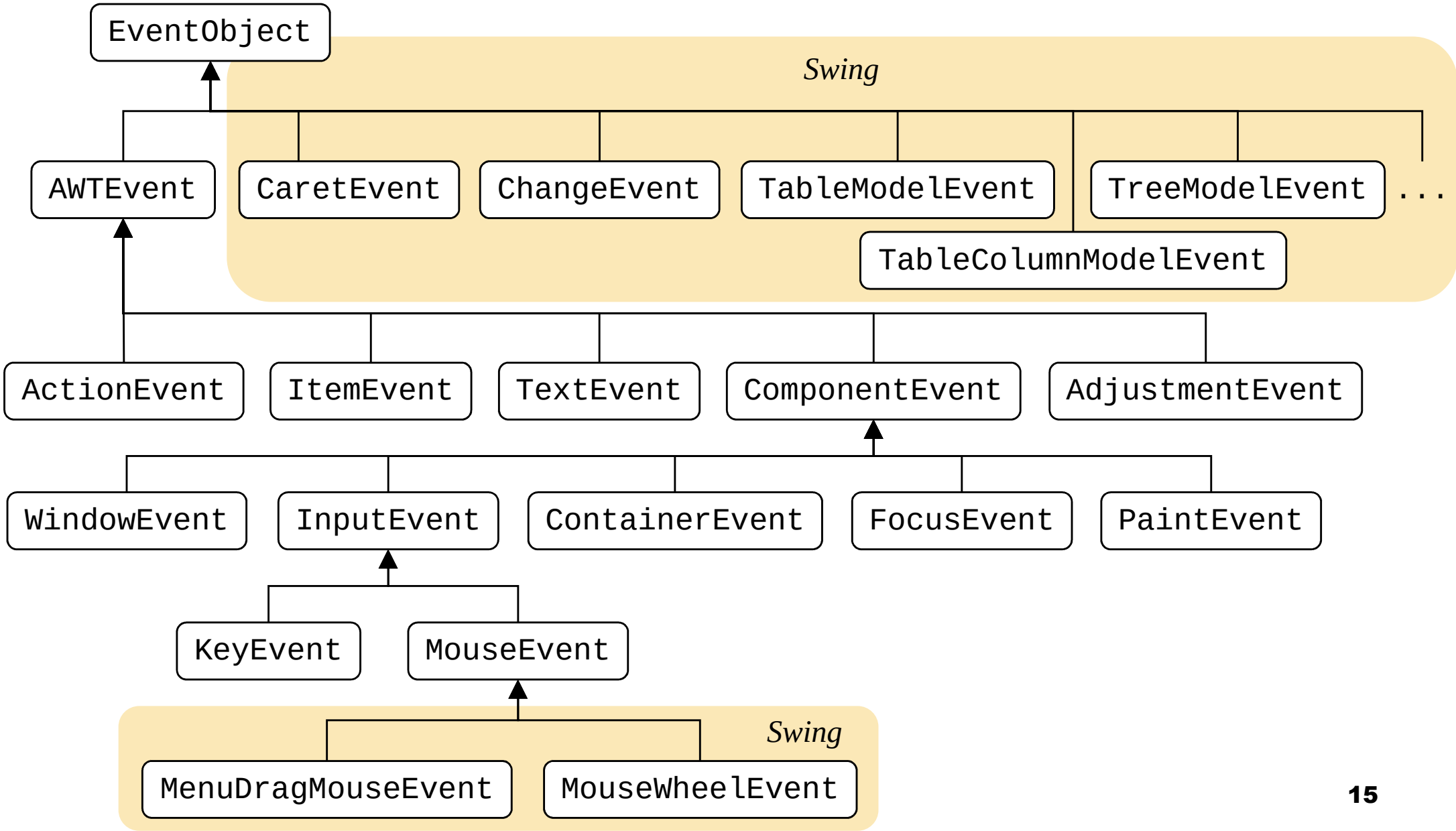
- Un tipo de receptor *xxx*`Listener` (excepto `MouseEvent` tiene 2)
- Una lista de clases de componentes que pueden producir el evento
- Un método `add`*xxx*`Listener` para registrar receptores de esos eventos
  - Este método está definido en las clases que generan el evento
  - Una componente sólo puede registrar listeners del tipo de eventos que genera la componente

Ejemplo

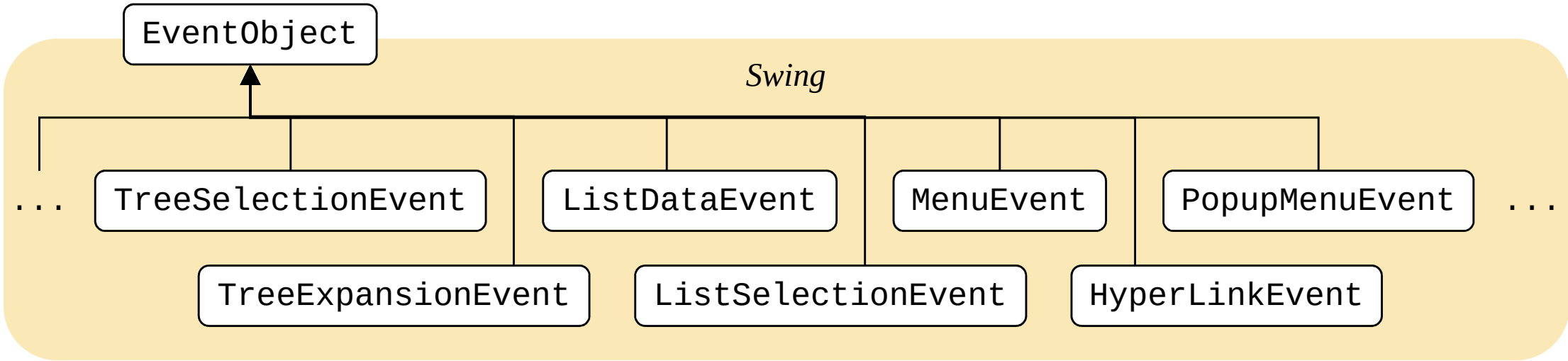
Clase de evento:	`ActionEvent`
Objetos que lo emiten:	`JButton`, `JMenuItem`, `JCheckBox`, `JRadioButton`, `JComboBox`, `JTextField`
Tipo de interfaz listener:	`ActionListener`
Métodos a implementar en clase listener:	`actionPerformed (ActionEvent)`
Método para registrar listener:	`addActionListener (ActionListener)`

Una componente sólo puede registrar listeners del tipo de eventos que genera la componente

Clases de eventos: `java.awt.event`



Clases de eventos: `javax.swing.event`



Eventos emitidos por cada clase

	Mouse	MouseMotion	Kev	Action	Window	Document	Item	Contaner	Component	Adiustment	Focus
JComponent	●	●	●						●		●
Jlabel	●	●	●						●		●
Jbutton	●	●	●	●					●		●
JCheckBox	●	●	●	●			●		●		●
JComboBox	●	●	●	●			●		●		●
Jlist	●	●	●						●		●
JTextField	●	●	●	●		●			●		●
JTextArea	●	●	●			●			●		●
JTextComponent	●	●	●			●			●		●

Eventos emitidos por cada clase

	Mouse	MouseMotion	Kev	Action	Window	Document	Item	Contaner	Component	Adiustment	Focus
JScrollBar	●	●	●						●	●	●
JMenuItem	●	●	●	●					●		●
JCheckBoxMenuItem	●	●	●	●			●		●		●
JRadioButtonMenuItem	●	●	●	●			●		●		●
Container	●	●	●					●	●		●
JPanel	●	●	●					●	●		●
JScrollPane	●	●	●					●	●		●
Window	●	●	●		●			●	●		●
Frame	●	●	●		●			●	●		●
JDialog	●	●	●		●			●	●		●

Métodos de los listener (I)

- **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)

Métodos de los listener (II)

- **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)`

Métodos de los listener (III)

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

Contenido de las clases de eventos

Las distintas clases de eventos incluyen:

- Constantes (variables estáticas)
  - ID de los distintos eventos de una clase

    P.e. `MouseEvent.MOUSE_MOVED`, `KeyEvent.KEY_RELEASED`
  - Constantes para ciertas propiedades de los eventos

    (valores devueltos por métodos)

    P.e. `ItemEvent.SELECTED`, `ItemEvent.DESELECTED`
- Métodos
  - Devuelven información adicional sobre el evento

    P.e. `getX()`, `getY()` de `MouseEvent`, `getKeyChar()` de `KeyEvent`, `getID()` de `AWTEvent`

Información contenida en los eventos (I)

- **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()`

Información contenida en los eventos (II)

- **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()

¿Qué deben hacer los métodos de los receptores?

- Modificar características de la interfaz
  - Cambiar colores, fonts, etiquetas, etc.
  - Mover objetos, cambiar su tamaño
  - Ocultar, mostrar, añadir, eliminar componentes
  - Abrir un cuadro de diálogo
  - etc.
- **Ejecutar funcionalidad de la aplicación**
  - **Normalmente se refleja algún resultado en la interfaz**

¿Procesar eventos o ignorarlos?

- Eventos de bajo nivel recogidos en widgets y elaborados en forma de eventos de alto nivel
  - Botones: `MouseEvent` → `ActionEvent`
  - Widgets de texto: `MouseEvent`, `KeyEvent` → `DocumentEvent`, `ActionEvent`
  - Widgets de selección: `MouseEvent` → `ItemEvent`, `ActionEvent`, `ListSelectionEvent`
  - etc.
- Eventos de cambio de estado de componentes: procesar los eventos inmediatamente vs. acceder al estado cuando se necesite
  - `ItemEvent`, `DocumentEvent`, `ComponentEvent`, `ContainerEvent`, `AdjustmentEvent`, etc.

Arquitectura AWT de procesamiento de eventos