

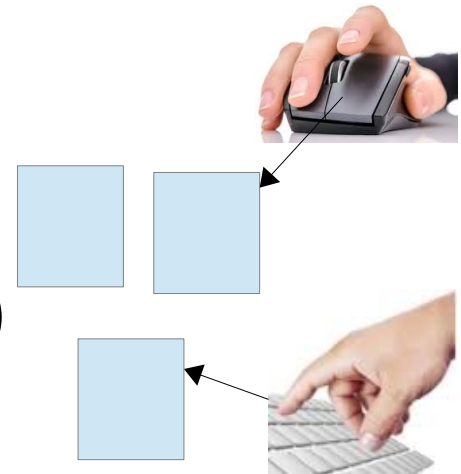
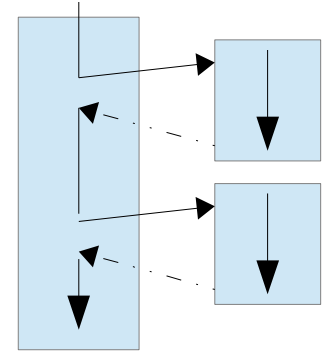
EIF206 - Programación 3

4 - Eventos

(Prof. José Sánchez, Ph.D.)

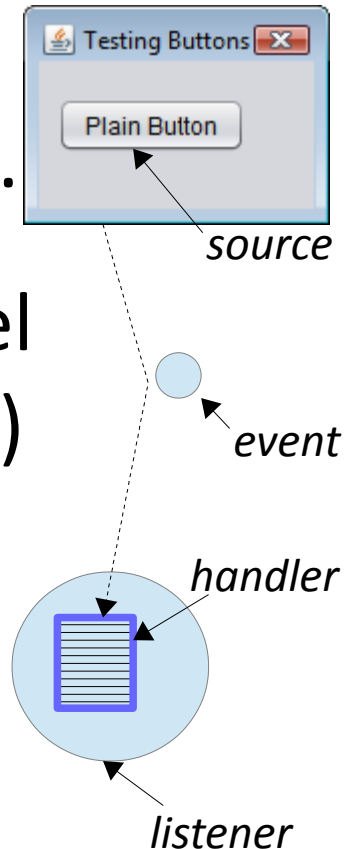
Programación Dirigida por Eventos (PDE)

- En programación tradicional el flujo del programa es secuencial
- En PDE el flujo del programa es determinado por eventos:
 - Acciones del usuario (mouse, teclado, etc.)
 - Sensores
 - Mensajes de otros programas o hilos



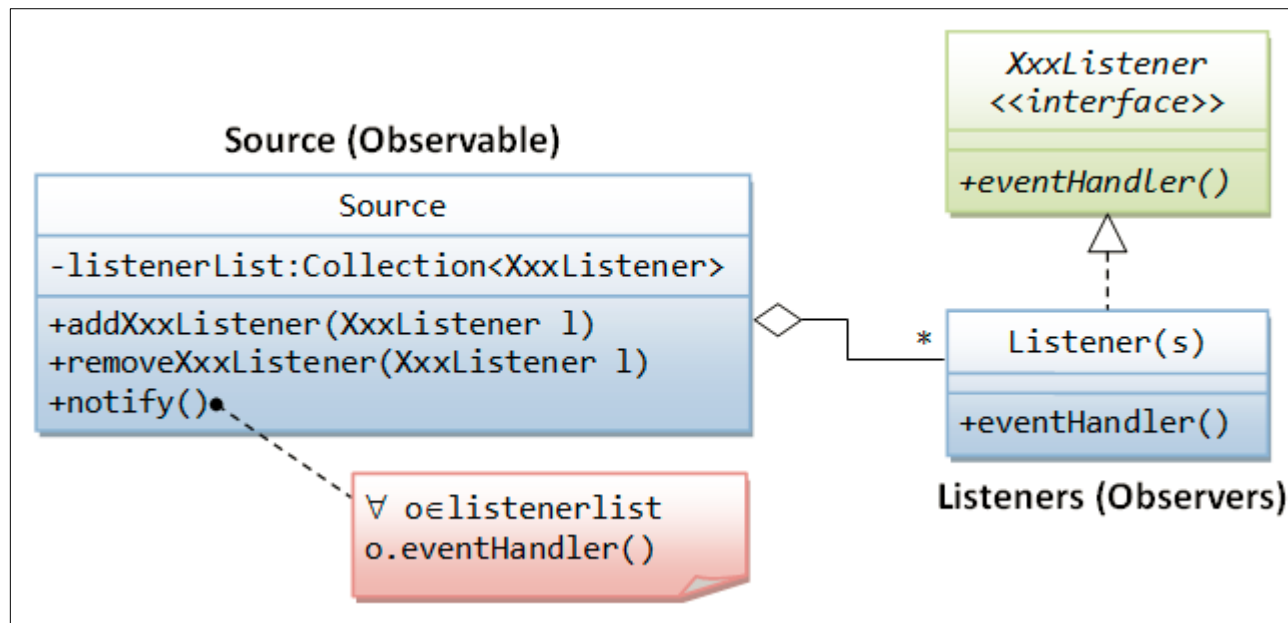
Componentes en PDE

- Fuente (*source*): el elemento (botón, campo, etc) sobre el que ocurre el evento.
- Evento (*event*): objeto con información del evento (ej. cuál botón, cuántos *clicks*, etc.)
- Comportamiento (*handler*): función o método que debe ejecutarse.
- Escucha (*listener*): objeto que contiene al *handler*.



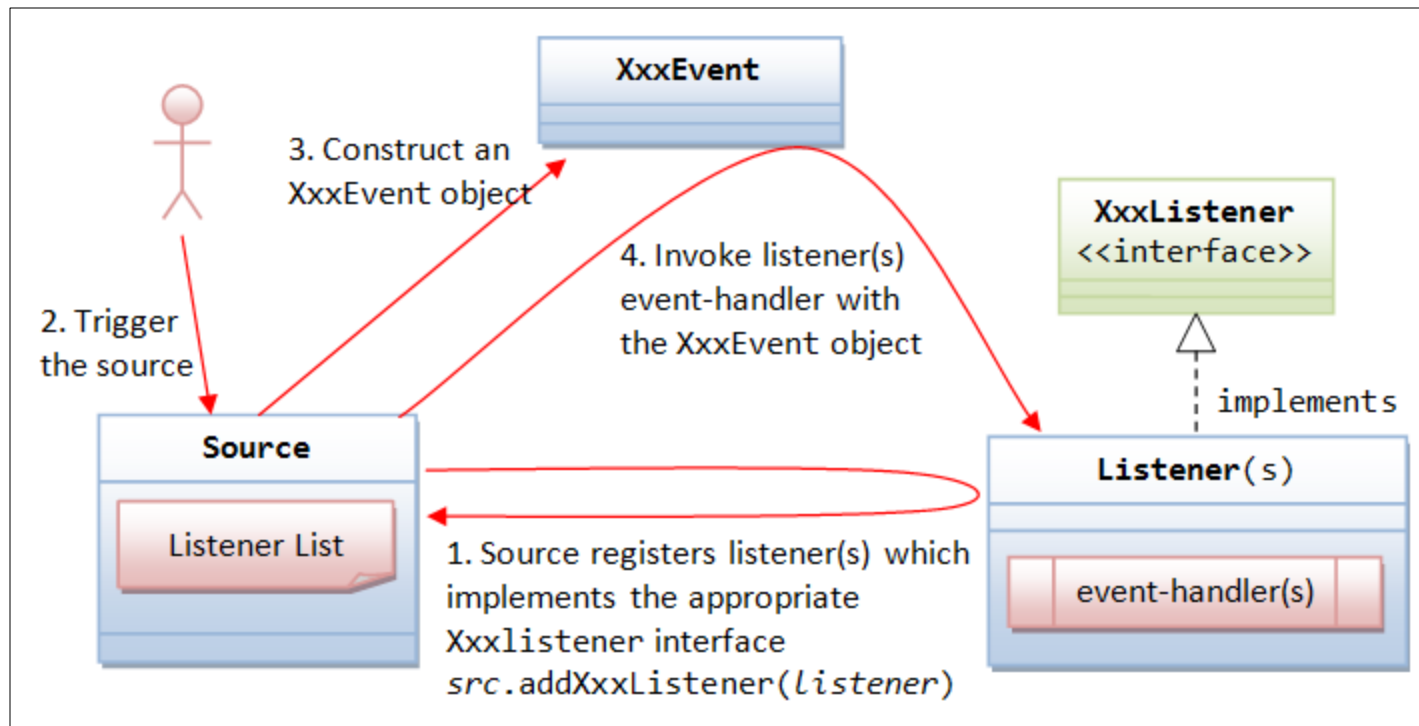
Estructura de Componentes en PDE

- Los elementos (*source*) que generan eventos deben tener una lista de a quiénes (*listeners*) le será notificado
- Debe haber métodos (*add*) para registrar listeners.
- Los *listeners* deben cumplir la interface respectiva.



https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI.html

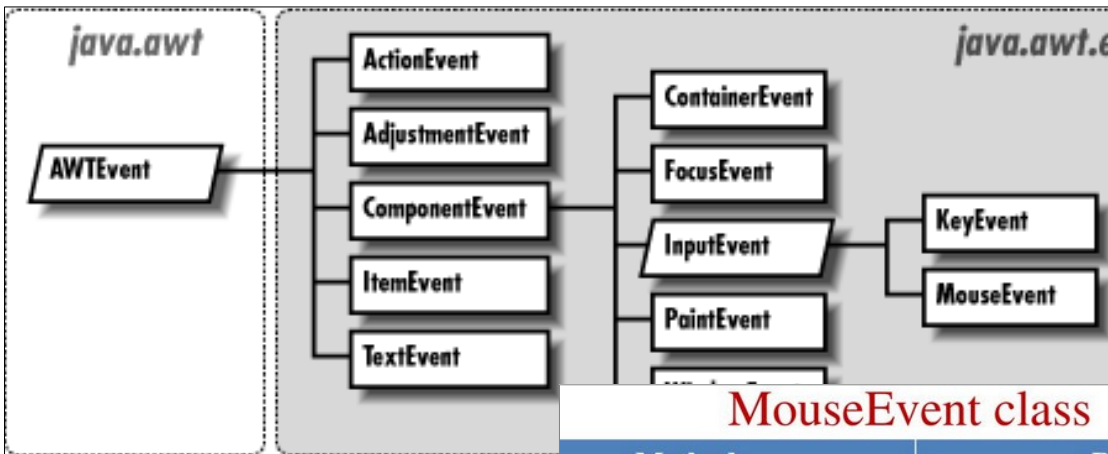
Comunicación entre Componentes



https://www3.ntu.edu.sg/home/ehchua/programming/java/J4a_GUI.html

Tipos de Eventos

- Los usuarios pueden generar eventos al presionar una tecla (*KeyEvent*), al mover o presionar el *mouse* (*MouseEvent*), etc.



KeyEvent class

Method	Purpose
<code>int getKeyChar()</code>	Obtains the Unicode character associated with this event.
<code>int getKeyCode()</code>	Obtains the key code associated with this event. The key code identifies the particular key on the keyboard that the user pressed or released. For example, VK_A specifies the key labeled A, and VK_ESCAPE specifies the Escape key.
<code>boolean isActionKey()</code>	Returns true if the key firing the event is an action key. Examples of action keys include Cut, Copy, Paste, Page Up, Caps Lock, the arrow and function keys.

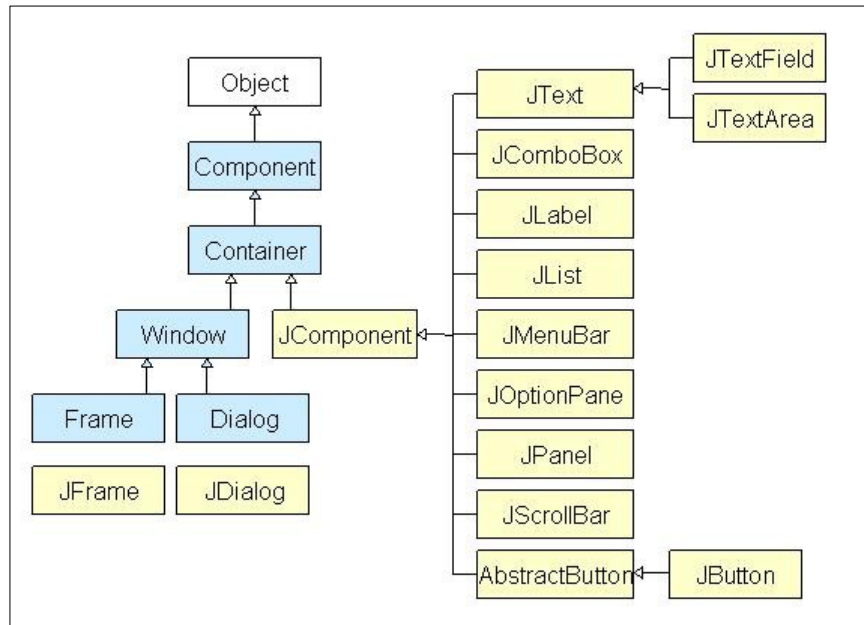
MouseEvent class

Method	Purpose
<code>int getClickCount()</code>	Returns the number of quick, consecutive clicks the user has made (including this event). For example, returns 2 for a double click.
<code>int getButton()</code>	Returns which mouse button, if any, has a changed state. One of the following constants is returned: NOBUTTON, BUTTON1, BUTTON2, or BUTTON3.
<code>int getX()</code> <code>int getY()</code>	Return the (x,y) position at which the event occurred, relative to the component that fired the event.

<https://slideplayer.com/slide/14031049/>

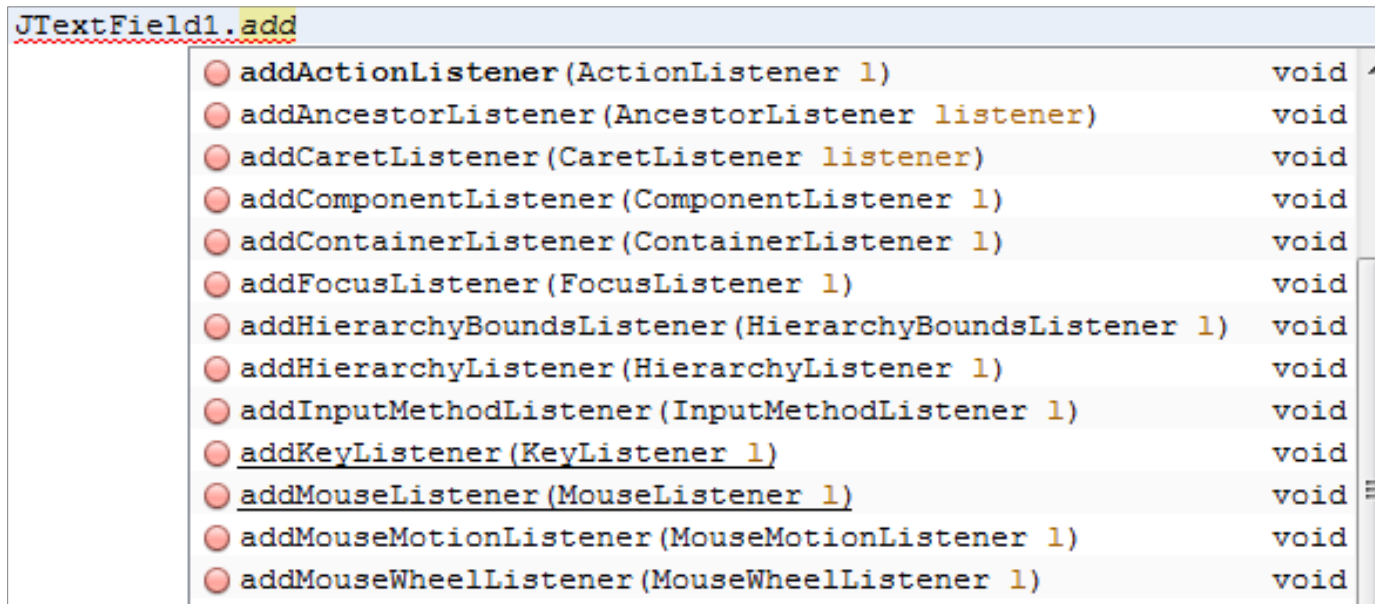
Fuentes de Eventos

- Las fuentes (Source) que reciben las acciones del usuario y generan los eventos a nivel de programación son los componentes de las interfaces de usuario (*JButton*, *TextField*, *JFrame*, etc.)



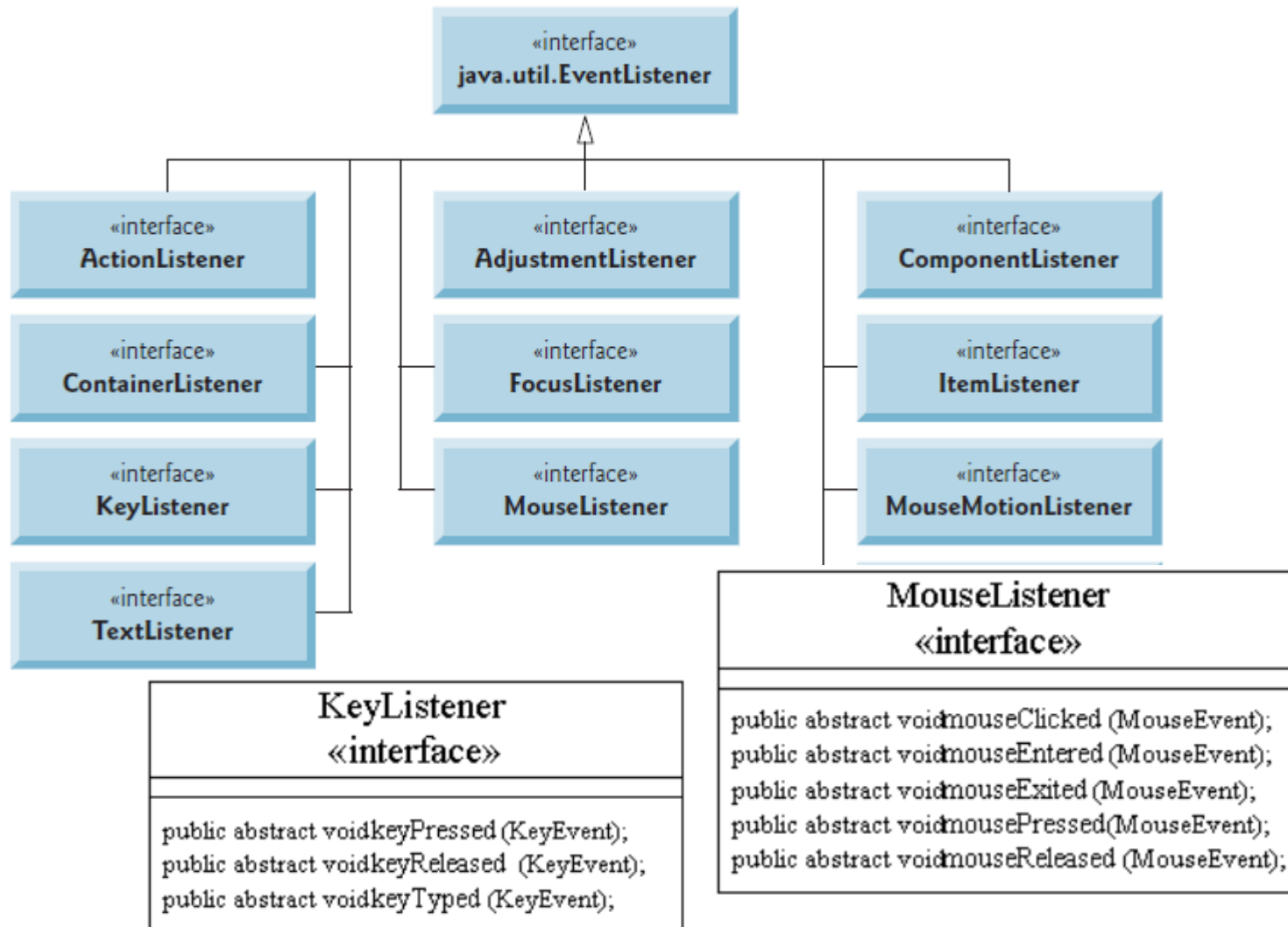
Fuentes de Eventos ..

- Los componentes (ej. *JTextField*) tienen listas de *listeners* para cada tipo de evento.
- Los *listener* tendrán que registrarse, invocando el método *addxxListener* respectivo.
- Los listeners son objetos de clases que implementan las interfaces respectivas (*KeyListener*, *MouseListener*, etc.)



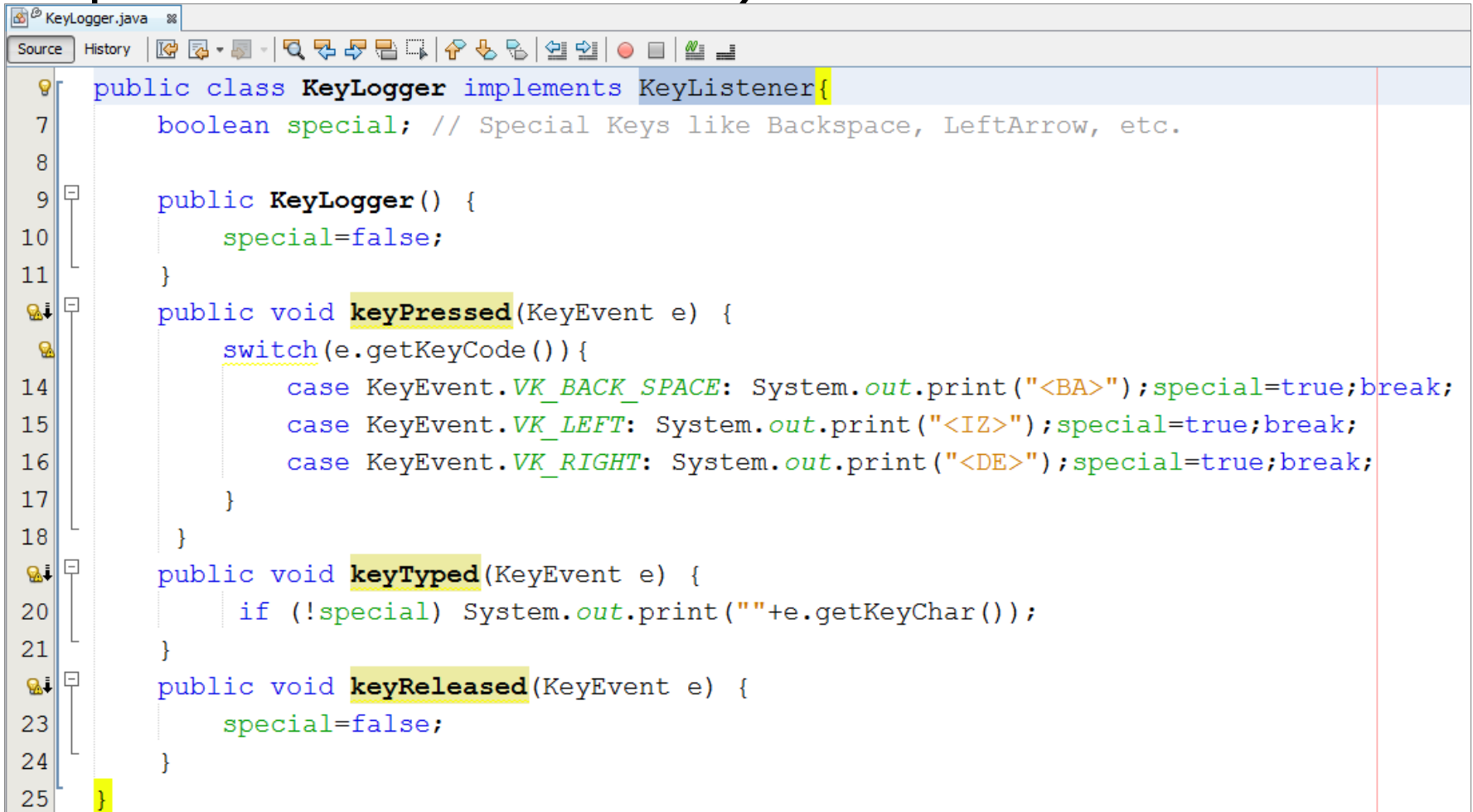
Interfaces *Listener*

- Para cada tipo de evento hay interfaces que deben implementar los *listeners* correspondientes



Ejemplo: KeyLogger

- Una clase KeyLogger que “escuche” las teclas que se digiten en un componente (ej. *TextField*) y las registre.
- Implementa la interface *KeyListener*.



The screenshot shows a code editor window titled 'KeyLogger.java'. The code implements the `KeyListener` interface. It includes a `boolean special` field to handle special keys. The `KeyListener()` constructor initializes `special` to `false`. The `keyPressed` method uses a `switch` statement to handle `VK_BACK_SPACE`, `VK_LEFT`, and `VK_RIGHT`, printing their respective symbols and setting `special` to `true`. The `keyTyped` method prints the character only if `special` is `false`. The `keyReleased` method resets `special` to `false`.

```
public class KeyLogger implements KeyListener{
    boolean special; // Special Keys like Backspace, LeftArrow, etc.

    public KeyLogger() {
        special=false;
    }

    public void keyPressed(KeyEvent e) {
        switch(e.getKeyCode()) {
            case KeyEvent.VK_BACK_SPACE: System.out.print("<BA>");special=true;break;
            case KeyEvent.VK_LEFT: System.out.print("<IZ>");special=true;break;
            case KeyEvent.VK_RIGHT: System.out.print("<DE>");special=true;break;
        }
    }

    public void keyTyped(KeyEvent e) {
        if (!special) System.out.print(""+e.getKeyChar());
    }

    public void keyReleased(KeyEvent e) {
        special=false;
    }
}
```

Ejemplo: KeyLogger ...

- Un objeto (*logger*) se registra como un *KeyListener* de un componente `TextField` de una pantalla (*p.nombre*)

The screenshot displays an IDE with two windows. The top window, titled 'Application.java', shows the following Java code:

```
1 package progra3.keylogger;
2
3 public class Application {
4
5     public static void main(String args[]) {
6         Pantalla p = new Pantalla();
7         KeyLogger logger = new KeyLogger();
8         p.nombre.addKeyListener(logger);
9         p.setVisible(true);
10    }
11 }
```

The bottom window, titled 'Output - KeyLogger (run)', shows the output of the program:

```
PROGRAMACION<IZ><IZ><IZ><IZ><BA>A<DE><DE><DE><DE> III
```

Below the output window, a small application window is visible. It contains two text input fields. The first field is labeled 'Id' and is empty. The second field is labeled 'Nombre' and contains the text 'PROGRAMACION IIII'.

Ejemplo: KeyLogger con varias fuentes

- Un mismo objeto (*logger*) se registra como un *KeyListener* de de varios componentes

The screenshot displays an IDE with two windows. The top window, titled 'Application.java', shows the following Java code:

```
1 package progra3.keylogger;
2
3 public class Application {
4
5     public static void main(String args[]) {
6         Pantalla p = new Pantalla();
7         KeyLogger logger = new KeyLogger();
8         p.nombre.addKeyListener(logger);
9         p.id.addKeyListener(logger);
10        p.setVisible(true);
11    }
12 }
```

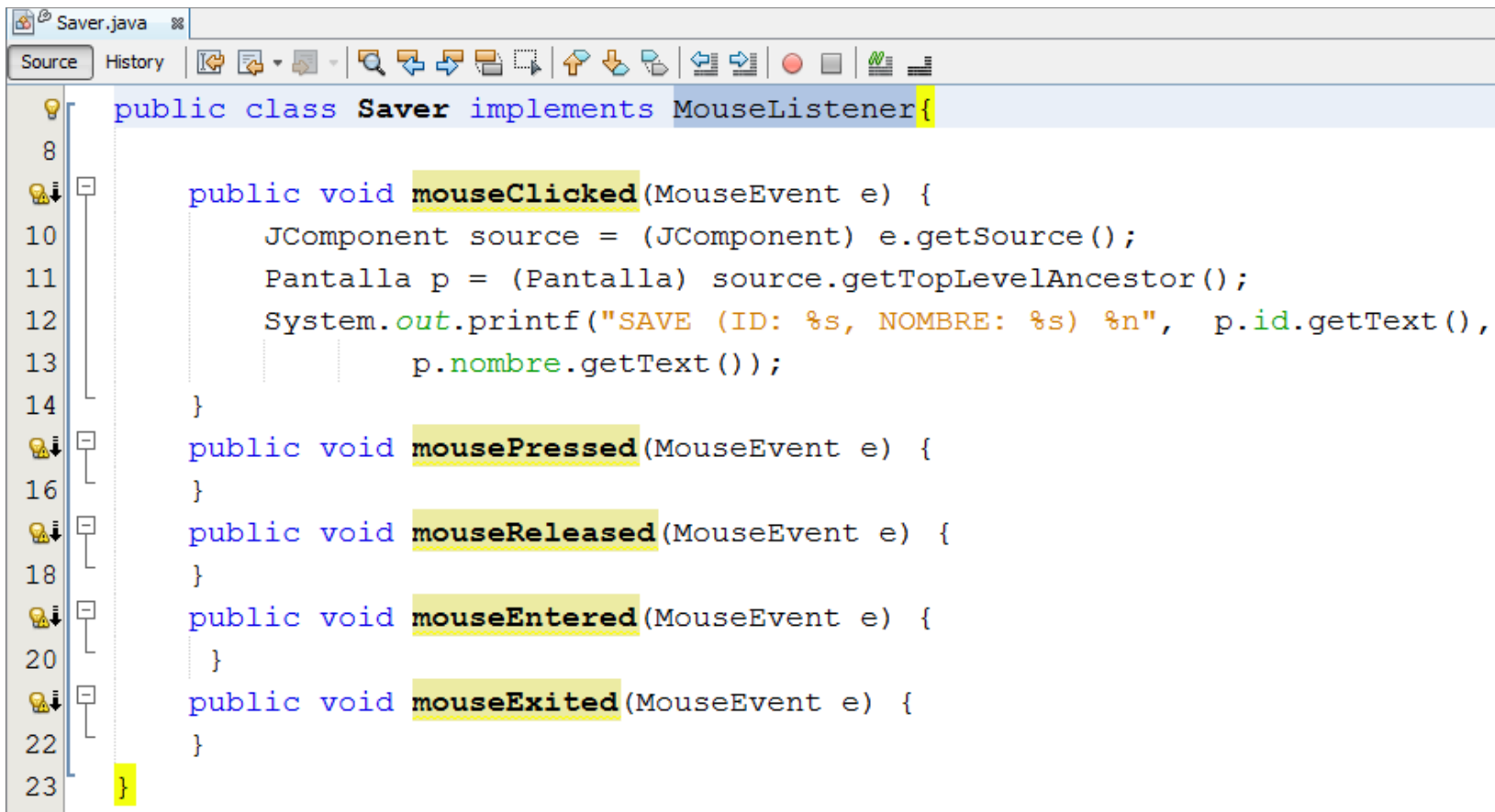
The bottom window, titled 'Output - KeyLogger (run)', shows the output of the program:

```
125<BA>6JOA<BA><BA>UAN
```

Below the output, a small application window is visible. It has two text input fields: 'Id' with the value '126' and 'Nombre' with the value 'JUAN'.

Ejemplo: Saver

- Una clase Saver que “salve” los datos de una pantalla cuando se haga *click* sobre un botón.
- Implementa la interface *MouseListener*.
- Solo el caso *mouseClicked* es relevante.



```
public class Saver implements MouseListener {

    public void mouseClicked(MouseEvent e) {
        JComponent source = (JComponent) e.getSource();
        Pantalla p = (Pantalla) source.getTopLevelAncestor();
        System.out.printf("SAVE (ID: %s, NOMBRE: %s) %n", p.id.getText(),
            p.nombre.getText());
    }

    public void mousePressed(MouseEvent e) {
    }

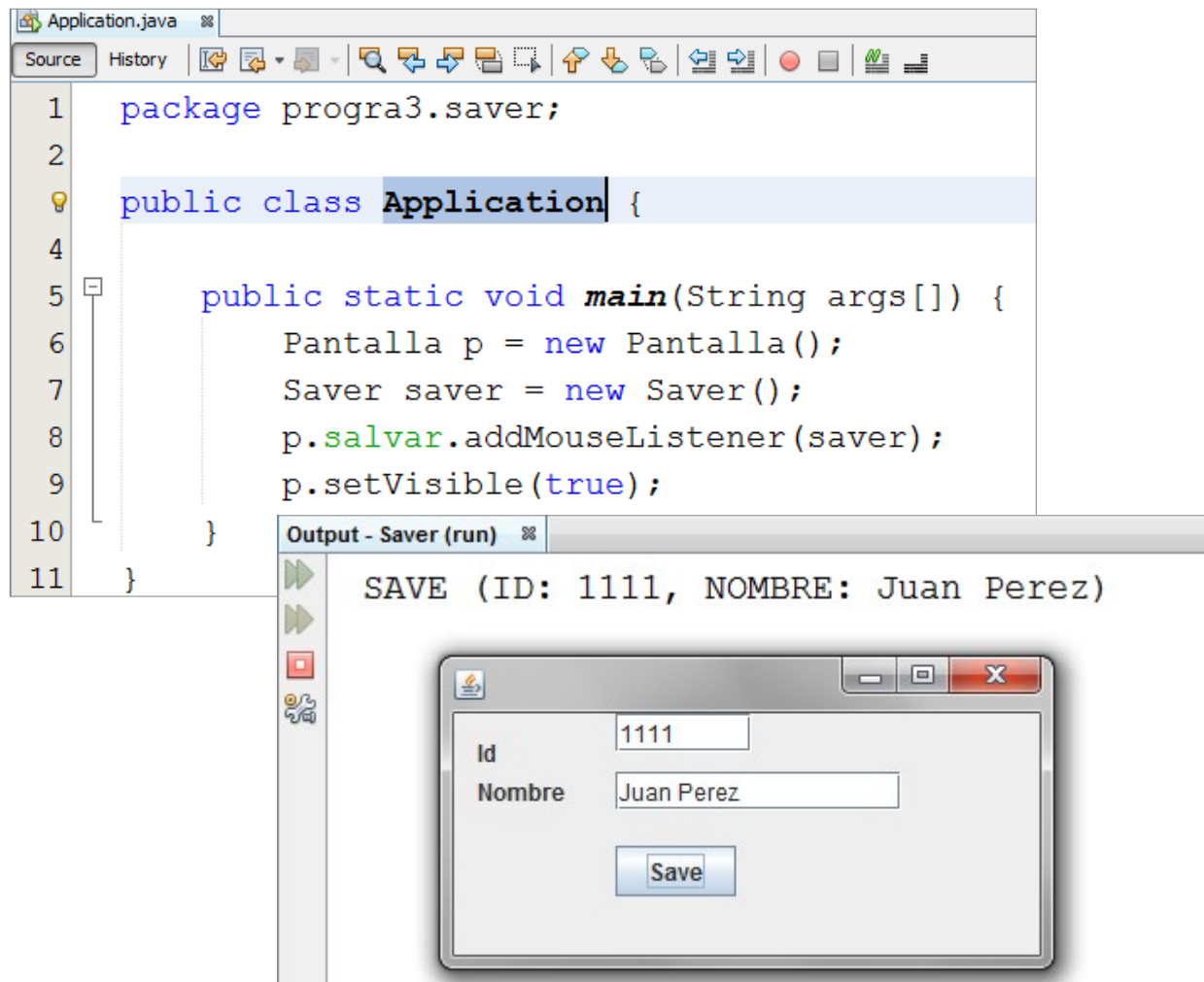
    public void mouseReleased(MouseEvent e) {
    }

    public void mouseEntered(MouseEvent e) {
    }

    public void mouseExited(MouseEvent e) {
    }
}
```

Ejemplo: Saver ..

- Un objeto (*saver*) se registra como un *MouseListener* de un componente *JButton* de una pantalla (*p.salvar*)



The screenshot displays an IDE with two main components. The top component is a code editor for 'Application.java', showing the following code:

```
1 package progra3.saver;  
2  
3 public class Application {  
4  
5     public static void main(String args[]) {  
6         Pantalla p = new Pantalla();  
7         Saver saver = new Saver();  
8         p.salvar.addMouseListener(saver);  
9         p.setVisible(true);  
10    }  
11 }
```

The bottom component is an 'Output - Saver (run)' window showing the output: 'SAVE (ID: 1111, NOMBRE: Juan Perez)'. Below the output, a small application window is visible. This window contains two text input fields: 'Id' with the value '1111' and 'Nombre' with the value 'Juan Perez'. Below these fields is a 'Save' button.