Diseño de Interfaces de Usuario

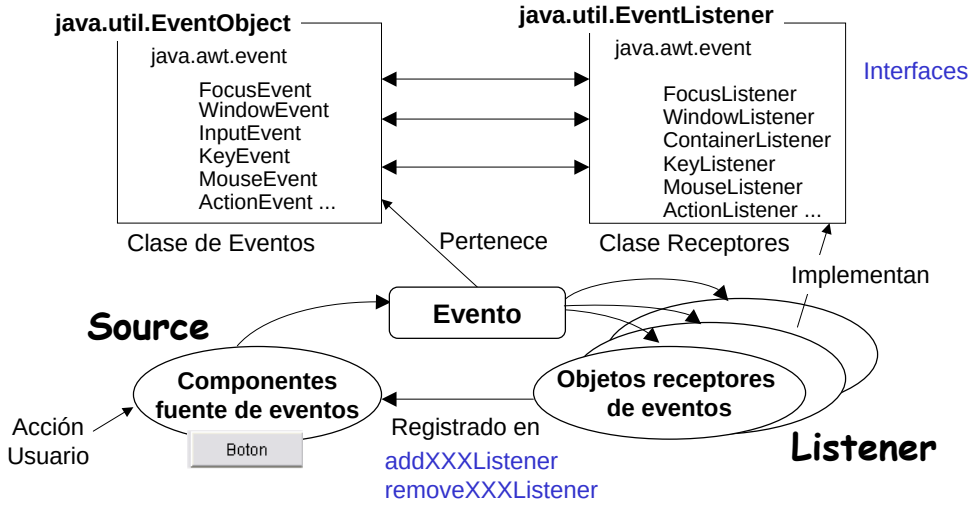
JAVA

6. Gestión de Eventos

import java.awt.event.*

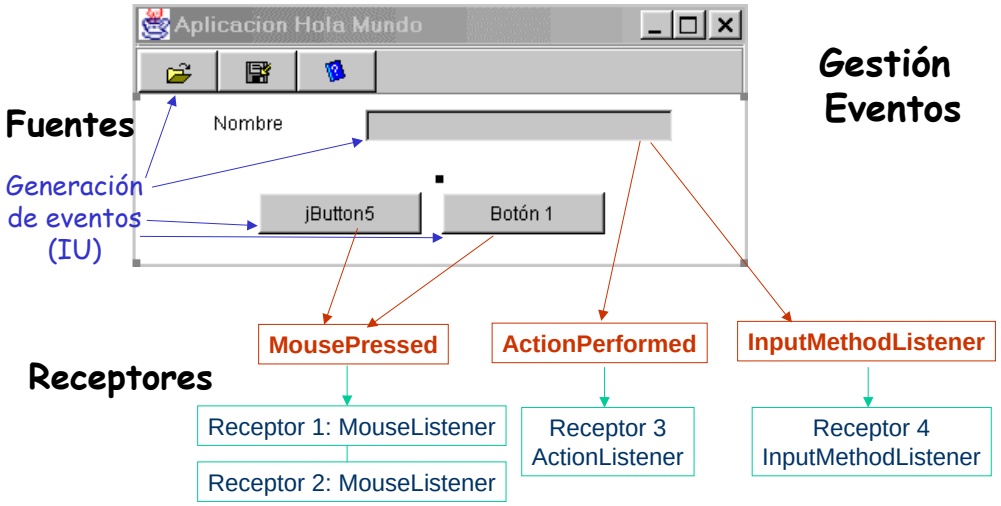
http://giig.ugr.es/~mgea/docencia/diu/diu.html
Ultima actualización: 2/Nov/2004

1. Modelo de gestión de eventos



Modelo de Delegación de Eventos

2. Ejemplo



Receptores (Listener) +Clases Adaptadoras

3. Tipos de eventos

Eventos

(bajo nivel)

ComponentEvent	Manipulación del componente
ContainerEvent	Cambio de contenido (en contenedor)
WindowEvent	Cambio de estado en ventana (abierta, icono..)
FocusEvent	Obtención del foco de teclado (TAB)
InputEvent (devices)	
KeyEvent	Evento de teclado
MouseEvent	Evento del ratón

(semánticos)

ActionEvent	Acción específica de cada componente
AdjustementEvent	Cambio de ajuste
InputMethod	Cambios en un control de entrada
ItemEvent	Cambio de selección de un ítem
TextEvent	Cambio en un objeto de texto

3.1 Tipos de eventos: Teclado

❖ **Eventos: Teclado (KeyEvent)**

Receptor: KeyListener

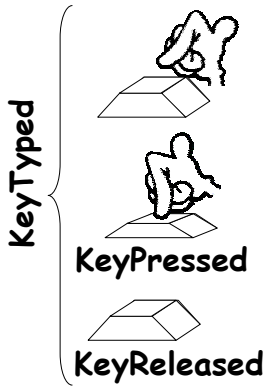
KeyListener: Métodos

KeyPressed (KeyEvent e);
KeyReleased (KeyEvent e);
KeyTyped(KeyEvent e);

Evento teclado (KeyEvent e):

e.getKeyChar();
e.getKeyCode();
e.getKeyModifiersText();
e.isAltDown();
e.isShiftDown();

Constantes: VK_B, VK_ALT, VK_F2...



3.2 Tipos de eventos: Ratón

❖ **Eventos: Ratón (MouseEvent)**

Receptores: MouseListener
MouseMotionListener

Gestión: MouseListener

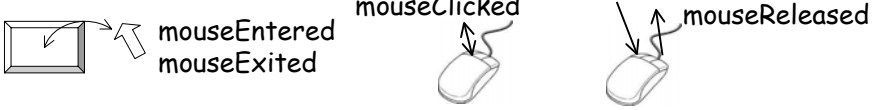
mouseClicked(mouseEvent e);
mousePressed (mouseEvent e);
mouseReleased (mouseEvent e);
mouseEntered (mouseEvent e);
mouseExited (mouseEvent e);

Gestión: MouseMotionListener

mouseMoved (mouseEvent e);
mouseDragged (mouseEvent e);

Evento ratón (MouseEvent e)

e.getX();
e.getY();
e.getClickCount();
e.isAltDown();
e.isShiftDown();
BUTTON1_MASK ...



3.3 Tipos de eventos: Semánticos

❖ **ActionEvent**: Evento que denota una acción específica de un componente (la mas usual)

Evento (ActionEvent e)	Gestión: ActionListener

e.ActionCommand();
e.getWhen();
e.getModiffiers();
	actionPerformed(ActionEvent e)
	JButton, JList, JTextField, JMenuItem, JPopupMenu, Jmenu,...

❖ **InputMethodEvent**: Evento para control de las entradas de usuario

Evento (InputMethodEvent e)	Gestión: InputMethodListener
e.CARET_POSITION_CHANGED // posición e.INPUT_METHOD_TEXT_CHANGED // texto	**inputMethodTextChanged**(InputMethodEvent event) **JTextField, JTextArea,..**

❖ **ItemEvent**: Evento que controla el cambio de estado de un componente

Evento (ItemEvent e)	Gestión: ItemListener
e.SELECTED e.DESELECTED e.ITEM_STATE_CHANGED // estado alternado	**itemStateChanged**(ItemEvent e) **JCheckBox, JList, JComboBox**

4. Objetos Receptores (Listener)

- Definen los métodos necesarios para cada gestión de Eventos
- Funcionalidad del receptor definida a través de INTERFACES
- ADAPTER: Clases genéricas que implementan los interfaces nulos

Método de construcción

1) Creación de una clase que implemente el interface
2) Clase que herede del Adaptador (clase genérica)
3) Clase anónima

4.1. Objetos Receptores: Interfaces

❖ **INTERFACES (receptores)**

ContainerListener
WindowListener
FocusListener
EventListener
KeyListener
MouseListener
MouseMotionListener
ActionListener
AdjustementListener
ItemListener
TextListener
. . .

```
public interface KeyListener
        extends EventListener {
  public void keyTyped(KeyEvent e);
  public void keyPressed(KeyEvent e);
  public void keyReleased(KeyEvent e);
}
```

- Definición de clases **abstractas** (lista de métodos obligatorios para gestionar un evento de ese tipo)
- Se debe crear una clase que **implemente** el interfaz

4.2. Objetos Receptores: Implementación

- Se deben "programar" todos los métodos del interfaz

```
Class ControlTeclado implements  java.awt.keyListener {

    public void keyTyped(KeyEvent e) {
       System.out.println("pulsada la tecla "+e.getKeyChar);
    }
    public void keyPressed(KeyEvent e) {
        System.out.println("pulsando tecla "+e.getKeyChar);
    }
    public void keyReleased(KeyEvent e) {
        System.out.println("soltando tecla "+e.getKeyChar);
    }
}
```

```
public interface KeyListener extends EventListener {
    public void keyTyped(KeyEvent e);
    public void keyPressed(KeyEvent e);
    public void keyReleased(KeyEvent e);
}
```

4.3. Objetos Receptores: Adaptadores

- Implementan INTERFACES (métodos nulos)
- No se necesita "programar" todos los métodos (sólo los que interesen)

```
Class ControlTeclado extends  java.awt.keyAdapter {

    public void keyTyped(KeyEvent e) {
       System.out.println("pulsada la tecla "+e.getKeyChar);
    }
}
```

```
public abstract class KeyAdapter implements KeyListener  {
        public void keyTyped(KeyEvent e)      { }
        public void keyPressed(KeyEvent e)   { }
        public void keyReleased(KeyEvent e)   { }
}
```


4.4 Objetos Receptores: Clases anónimas

- Se crea la clase (anónima) al registrar el receptor. No posee identificador
- Útil si la clase es específica y no se usa desde otro lugar (por defecto en JBuilder)
- Pasos:
  - Creación de la clase adaptadora (1)
  - Se registra en el fuente (2)
  - Se redefinen los métodos de interés (3)

```
panelMouse.addKeyListener
            (new java.awt.KeyAdapter()  {

    public void keyTyped (KeyEvent e) {
        System.out.println("pulsada la tecla "+e.getKeyChar);
    }
} ); // una sola sentencia
```

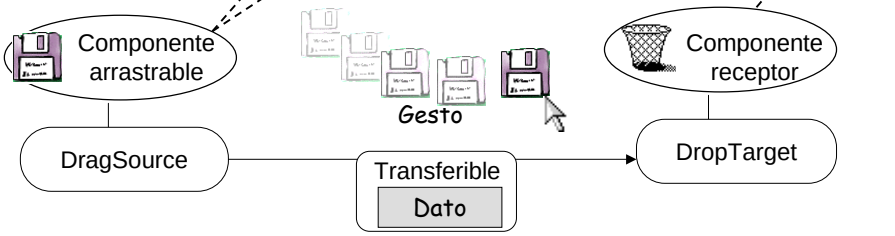
Método redefinido

5. Manipulación directa (drag & drop)

1. Objeto "arrastrable"
2. Objeto(s) receptor
3. Gestor del Gesto
4. Dato a transferir

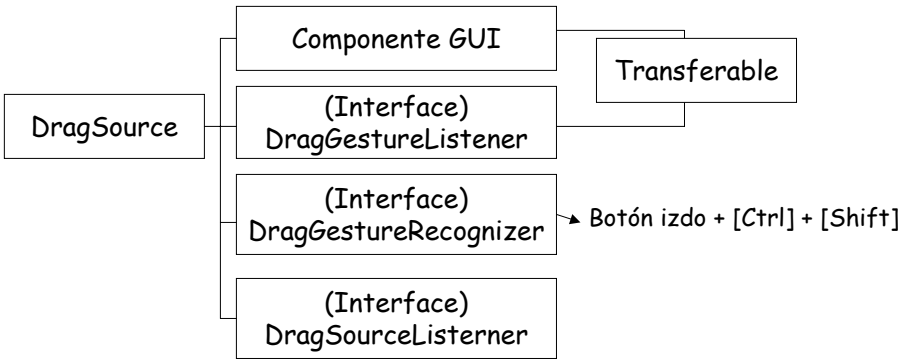
DragSourceListener
DropTargetListener
DragGestureListener

Implementa



5.1 Manipulación directa: objeto arrastrable

- **Objeto *Arrastrable*** *(DagSource)*



5.1 Manipulación directa: objeto arrastrable (cont.)

import java.awt.dnd.*

❖ **Interface de objeto *Arrastrable***

```
public interface DragSourceListener {
  void dragEnter(DragSourceDragEvent dsde);
  void dragOver(DragSourceDragEvent dsde);
  void dropActionChanged(DragSourceDragEvent dsde);
  void dragExit(DragSourceEvent dse);
  void dragDropEnd(DragSourceDropEvent dsde);
}
```


5.2 Manipulación directa: receptor

import java.awt.dnd.*

❖ **Interface del gestor del gesto**

```
public interface DragGestureListener {
  void  void dragGestureRecognized(DragGestureEvent dge);
}
```

5.2 Manipulación directa: receptor (cont.)

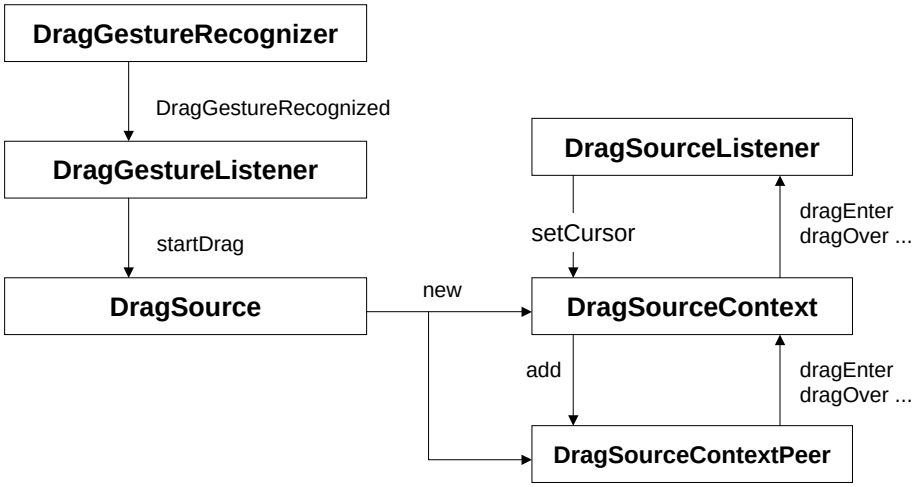
```
import java.awt.dnd.*
```

❖ **Interface del receptor del objeto**

```
public interface DropTargetListener extends EventListener {
  void dragEnter(DropTargetDragEvent dtde);
   void dragOver(DropTargetDragEvent dtde);
   void dropActionChanged(DropTargetDragEvent dtde);
   void dragExit(DropTargetEvent dte);
   void drop(DropTargetDropEvent dtde);}
}
```


5.3 Manipulación directa: Relaciones

❖ **Relaciones entre objetos**



5.4 Manipulacíon directa: Ejemplo

❖ **Ejemplo de objeto "arrastrable"**

```
public class DragLabel extends JLabel
      implements DragSourceListener, DragGestureListener {

 DragSource obj_arrastrar;

  public DragLabel(String s) {
   this.setText(s);
   obj_arrastrar = new DragSource();
   obj_arrastrar.createDefaultDragGestureRecognizer(this, DnDConstants.ACTION_COPY, this);
 }

   public void dragGestureRecognized(DragGestureEvent e) {
     StringSelection texto = new StringSelection (getText());
     obj_arrastrar.startDrag(e, obj_arrastrar.DefaultCopyDrop, texto, this);
 }
 . . .
```


5.4 Manipulacíon directa: Ejemplo (cont.)

❖ **Ejemplo de objeto receptor**

```
public class DropList extends java.awt.List
    implements DropTargetListener {

 DropTarget obj_destino;

  public DropList() {
    obj_destino = new DropTarget(this,this);
  }

  public void dragEnter(DropTargetDragEvent e) {
     System.out.println( "Ha entrado en el receptor");
     e.acceptDrag(DnDConstants.ACTION_COPY);
    }
```

5.4 Manipulacíon directa: Ejemplo (cont.)

❖ **Ejemplo de objeto receptor**(Cont.)

```
// class DropList extends java.awt.List
public void drop (DropTargetDropEvent e) {
  try {
   Transferable tr = e.getTransferable();
   if(tr.isDataFlavorSupported(DataFlavor.stringFlavor)) {
        e.acceptDrop(DnDConstants.ACTION_COPY_OR_MOVE);
        String s = (String)tr.getTransferData(DataFlavor.stringFlavor);
        add(s);   // añade a la lista
        e.getDropTargetContext().dropComplete(true);
        }
     else {
       System.err.println("Dato no aceptado");   e.rejectDrop();  }

  } catch (IOException io) { io.printStackTrace();  e.rejectDrop();   }
    catch (UnsupportedFlavorException ufe) {   ufe.printStackTrace(); e.rejectDrop();  }
  }
```