U11. Interfaz Gráfica

Componentes

Disposición: Layouts

Interacción con el usuario: Gestión de eventos

Elementos básicos de una GUI

- **Componentes**:
  - Ventanas
  - Contedores
  - Menús
  - Barras
  - Botones, capos de texto, listas desplegables…..
- **Disposición (layout):** Cómo se clocan los componenetes
  - Layout managers: Gestinan la organixaión de los componentes gráficos
- **Eventos:** interactividad con el usuario
  - Desplazamiento del ratón , selección de un menú, botón pulsado…

Bibliotecas de componentes para GUI

- AWT
  - Funcionalidad dependiente de la plataforma
    - La apariencia de ventanas, menúes, etc. es distinta en Windows, Mac, Motif, y otros sistemas
- Swing
  - Independiente de la plataforma
  - Escrito totalmente en Java. No reemplaza a AWT.
  - Se apoya sobre AWT y añade JComponents.

Componentes

- Contenedores
  - Contienen oros componentes(u contenedores)
    - Panel, Frame…
- Componentes de interfaz de usuario
  - Botones, listas menús, campos de texto, casillas de verificación
- Componentes de construcción de ventanas
  - Ventanas, marcos,barras de menús, cuadros de diálogo
- Lienzo

Jerarquía de componentes

Contenedores

- **Panel**
  - Sirve para colocar botones, etiquetas, etc…
  - No existe sin una ventana que lo albergue
  - Ej: Jpanel, JScrollPane
- **Window**
  - Sirve para crear ventanas independientes
  - Normalmente se usan 2 tipos:
    - **Frame**: ventana donde se pueden colocar menús, no contenida en otras ventanas. Ej.: Jframe
    - **Dialog**: Cuadro de diálogo: JDialog

Elementos de una ventana

Jframe

- Ventana al uso
- Se utiliza en Swing como contenedor raíz
- Los contenidos se añaden en el *panel de contenidos* (***content pane***) accesible a través del método ***getContentPane*** (por defecto, un objeto de tipo **Jpanel**, aunque puede cambiarse con *setContentPane*).
- La barra de menú puede fijarse con setJMenuBar

Componentes

- **Botón** - Clase ***JButton***
  - Botón de interacción que puede tener una etiqueta
- Etiqueta - Clase ***JLabel***
  - Muestra una cadena de sólo lectura
  - Normalmente para asociar el texto con otro componente
- **Campo de texto** - Clase ***JTextField***
  - Campo de una línea que permite introducir y editar texto
- **Area de texto** - Clase ***JTextArea***
  - Campo de texto de varias líneas
  - Mayor funcionalidad
    - Añadir, reemplazar e insertar texto
    - Barras de desplazamiento
- **Menús** - Clase ***JMenuBar, JMenu, y JMenuItem***
  - Para definir una barra de menús, cada menú y los elementosde cada menú

Inicialización

- Primero se crean los componenetes y se configuran
- Por último se agregan al contenedor con el método add de éste
- Cómo se coloquen los componentes, depende del gestor de disposición del contenedor

Disposición de los componentes

- Tipos de disposiciones
  - **FlowLayout:**
    - Los componentes se ponen de izquierda a derecha hasta llenar la línea, y se pasa a la siguiente. Cada línea se centra
      - Por defecto, en paneles y applets
  - **BorderLayout**
    - Se, ponen los componentes en un lateral o en el centro
    - Se indica con uma dirección: “East”,”West”,”North”,”South”,”Center”
      - Por defecto en ventanas Jframe
  - **GridLayout**
    - Se colocan en una rejilla rectangular (filas x columnsa)
    - Se añaden en orden de izquierda → derecha y arriba → abajo
- Para poner una disposición se utiliza el método *setLayout():*

```
GridLayout nuevolayout = new GridLayout(3,2);
setLayout(nuevolayout);
```

- El *layout manager* elige la mejor posición y tamaño de cada componente de acuerdo al espacio disponible.

Disposición

FlowLayout

GridLayout(3,2)

Contenedores anidados

- Para conseguir diseños sofisticados lo mejor es anidar contenedores
- Cada contenedor tendrá su propio Layout

Ejemplo

// Extraído de "Introduction to Java Programming"
// de Y. Daniel Lang (Prentice Hall, 2001)

Tratamiento de eventos

- Al interactuar con la aplicación, el usuario:
  - Acciona componentes (**ActionEvent**).
    - El usuario pulsa un botón.
    - El usuario termina de introducir un texto en un campo y presiona Intro.
    - El usuario selecciona un elemento de una lista pulsando el preferido (o de un menú).
- Pulsa o suelta botones del ratón (**MouseEvent**).
- Minimiza, cierra o manipula una ventana (**WindowEvent**).
- Escribe con el teclado (**KeyEvent**).
- Descubre porciones de ventanas (**PaintEvent**).

Tratamiento de eventos

- Los eventos se corresponden a las interacciones del usuario
- Los componentes están asociados a distintos tipos de eventos. Ej.:
  - Ventanas(Ej.: Jframe): WindowEvent
  - Menús: ActionEvent
- Se pueden definir objetos (listeners) que saben cómo tratar los eventos
  - Cuando ocurre un evento, estos objetos (listeners) son notificados y llaman al método que tratará el evento

Eventos

Acción	Objeto origen	Tipo de evento
Pulsar un botón	JButton	ActionEvent
Cambio del texto	JTextComponent	TextEvent
Pulsar Intro en un campo de texto	JTextField	ActionEvent
Selección de un nuevo elemento	JComboBox	ItemEvent ActionEvent
Selección de elemento(s)	JList	ListSelection- Event
Pulsar una casilla de verificación	JCheckBox	ItemEvent ActionEvent
Pulsar un botón de radio	JRadioButton	ItemEvent ActionEvent
Selección de una opción de menú	JMenuItem	ActionEvent
Mover la barra de desplazamiento	JScrollBar	AdjustmentEvent
Abrir, cerrar, minimizar, maximizar o cerrar la ventana	JWindow	WindowEvent
. . .		

Tratamiento de eventos

Dos categorías de eventos:

- Eventos de bajo nivel
  - Están relacionados con la interacción física con la interfaz (por ejemplo, ¿qué botón del ratón se ha pulsado?)
  - Ejemplos: MouseInputEvent, WindowEvent y KeyEvent
- Eventos de alto nivel, o semánticos
  - Representan operaciones lógicas realizadas sobre los elementos (por ejemplo, se ha pulsado el botón “Salir” en la interfaz)
  - ActionEvent

Tratamiento de eventos. Listeners

- El objeto listener debe ser de una clase que herede la interfaz correspondiente al tipo de evento que va a tratar

Ej.:

Evento	Lnterfaz	Método
ActionEvent	ActionListener	actionPerformed()

- El objeto listener se añade a una lista de listeners asociados al componente
- Cuando se produce un evento, se notifica a todos los objetos listeners asociados

Eventos de acción: ActionEvent

- Indica que se ha producido un evento sobre un componente de la interfaz
  - JButton, JList, JTextField, JMenuItem, etc.
- El método que se invoca en los listeners está definido en **la interfaz** ActionListener

```
public interface ActionListener extends EventListener {
void actionPerformed(ActionEvent e) ;
}
```

- Luego la clase del objeto oyente debe implementar el método *actionPerformed*
- Los componentes tienen métodos para poder añadir o quitar objetos oyentes
  - **addActionListener**(ActionListener l)
  - **removeActionListener**(ActionListener l)

Tratamiento de eventos. Listeners

- Creamos un manejador de eventos en dos pasos:
- Definimos una clase específica que haga de oyente de eventos y que implemente el método **actionPerformed()**.

```
class MiOyente implements ActionListener {
public void actionPerformed() {
// Aquí se responde el evento
```

- Registramos un ejemplar(instancia) como oyente de componentes:
  - `componente.addActionListener(ejemplar_de_MiOyente);`

Interfaces habituales y sus métodos

ActionListener Escucha eventos de tipo **ActionEvent**	actionPerformed(ActionEvent)
KeyListener Escucha eventos de teclado	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
MouseListener Escucha eventos de acción del ratón (botones)	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener Escucha eventos de movimiento del ratón	mouseDragged(MouseEvent) mouseMoved(MouseEvent)

Listeners

- Lo mejor es definir clases específicas por cada evento
- Veamos cómo hacerlo usando clases anidadas

Ejemplo

```
class MiGui extends JFrame {
                                        ...
public void MiGui() {
    // Registro los componentes interesados
    // en responder a eventos ... componente.addActionListener(new
    MiOyente());


                                            }
                                        ...
    class MiOyente implements ActionListener { public void
    actionPerformed(ActionEvent e) {
    ... // Aqui se responde el evento
                                            }
                                        }
```

Se añade el oyente especificado (ejemplar de **`MiOyente`**) a la lista de oyentes para recibir eventos de acción (**`ActionEvent`**) desde ese componente.

Clases anidadas

- Las definiciones de clase se pueden anidar

```
public class Contenedora
{
...
private class Anidada
{
...
}
}
```

- Las instancias de las clases internas estarán dentro de la clase contenedora
- Las instancias de la clase interna tienen acceso a la parte privada de la clase contenedora
  - Esto es práctico porque el tratamiento de un evento requiere normalmente acceso al estado de la aplicación

Clases anidadas **anónimas**

Se usan para crear objetos específicos en un momento dado
Ejemplo de ActionListener anónimo:
La instancia es referenciada por su supertipo, ya que no tiene nombre el subtipo

```
JMenuItem openItem = new JMenuItem("Open");

openItem.addActionListener(
    new ActionListener()
    {
        public void actionPerformed(ActionEvent e)
        {
            openFile();
        }
    }
);
```

Anonymous object creation

Class definition

Actual parameter

ActionEvent

- **getActionCommand()**
  - devuelve un objeto con la información sobre el origen del evento (el botón en nuestro caso).
  - Con un botón, devuelve la etiqueta del botón.
- **GetSource()**
  - indica el objeto en el que se ha originado el evento

Botones

- Los botones, junto con los menús, son los controles más típicos en una GUI
- Existen diferentes tipos (especializaciones de AbstractButton)
- **JButton**: Botón aislado. Puede pulsarse, pero su estado no cambia
- **JToggleButton** : Botón seleccionable. Cuando se pulsa el botón, su estado pasa a *seleccionado*, hasta que se pulsa de nuevo (entonces se deselecciona)
  - *isSelected()* permite chequear su estado
- **JCheckBox** : Especialización de JToggleButton que implementa una casilla de verificación. Botón con estado interno, que cambia de apariencia de forma adecuada según si está o no está seleccionado
- **JRadioButton**: Especialización de JToggleButton que tiene sentido dentro de un mismo grupo de botones (*ButtonGroup*) que controla que sólamente uno de ellos está seleccionado
  - Nota: ButtonGroup es únicamente un controlador, no un componente)
- El evento semántico más común anunciado por los botones es ActionEvent

Otros componentes de Swing

- JLabel. Etiquetas
- JTextField. Entrada de una línea de texto
- JTextArea. Entradas de varias líneas de texto
- JSlider. Una barra de selección en una escala
- JList. Una lista de elementos seleccionables
- JComboBox. Una lista de elementos desplegable

JButton

- Constructores:
  - **JButton(String *text*)**
  - **JButton(String *text*, Icon *icon*)**
  - **JButton(Icon *icon*)**

JLabel

- Constructor: **JLabel(String *text*)**
- Cambiarle el texto: **setText(String tex)**

JTextField

- **Constructores**
  - **JTextField(int *columns*)**
  - **JTextField(String *text*)**
  - **JTextField(String *text*, int *columns*)**
- **Métodos:**
  - Poner texto: **text1.setText("Adios");**
  - Obtener texto: **String str = text1.getText();**
  - Agregar al Panel: **p1.add(text1);**

JComboBox

Listas de elementos para seleccionar un solo valor:

- Creación:

```
JComboBox ch1 = new JComboBox();
```

- Agregar opciones:

```
ch1.addItem(Object elemento);
```

- Registrar Evento:

```
ch1.addItemListener( objeto);
```

- Obtener selección:

```
val = ch1.getSelectedIndex();
ch1.getSelectedItem()
```

- Implementar la interfaz ItemListener

Implementar el método **itemStateChanged(ItemEvent e)**

```
ch1.addItemListener(new OyenteItem());
. . .
class OyenteItem implements ItemListener {
public void itemStateChanged(ItemEvent e) {
    ...

}}
```

JList

Listas de elementos para seleccionar uno o varios valores:

JList l1 = new JList();

JList l2 = new JList(Object[] *elements*);

String[] cosas = {"Opción 1", "Opción 2", "Opción 3"};

Jlist l2 = new Jlist(cosas);

- Registrar evento: **l2.addListSelectionListener(oyente);**
- Obtener selección:
- **int[] indices = l2.getSelectedIndices();**

Ejemplo List

Implementar la interfaz **ListSelectionListenner**

Implementar el método **valueChanged(ListSelectionEvent e)**

```
l.addListSelectionListener(new OyenteLista());
...
class OyenteLista implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent e) {
        int[] indices = l.getSelectedIndices();
        int i;
        for(i = 0; i < indices.length; i++) {
        ...
        }
    }
}
```

Interacción modal

showMessageDialog: ERROR_MESSAGE, INFORMATION_MESSAGE……

```
JOptionPane.showMessageDialog(this, // La ventana padre.
   "Error deconocido!: Lo llevas muy mal!", //El mensaje.
   "Error",  // El título de la ventana de diálogo.
   JOptionPane.ERROR_MESSAGE // El tipo de mensaje
);
```

```
JOptionPane.showMessageDialog(this,
"Te informo de que lo llevas fatal", "Info",
JOptionPane.INFORMATION_MESSAGE);
```

Iteracción Modal

showMessageDialog: WARNING_MESSAGE, PLAIN_MESSAGE…….

```
JOptionPane.showMessageDialog(this,
 "Te aviso de que lo llevas fatal", "Aviso",
 JOptionPane.WARNING_MESSAGE);
```

```
JOptionPane.showMessageDialog(this,
  Este mensaje es para tí, majete!", "Mensaje",
JOptionPane.PLAIN_MESSAGE);
```

Iteracción Modal

showConfirmDialog

```
int seleccionada =
  JOptionPane.showConfirmDialog(this,
  "Lo aceptas?", "Aviso",
  JOptionPane.YES_NO_OPTION, // Configuración del mensaje
  JOptionPane.INFORMATION_MESSAGE);
switch(seleccionada) {
  case JOptionPane.YES_OPTION: ... // tratar SI
  case JOptionPane.NO_OPTION: .. // tratar NO
  case JOptionPane.CLOSED_OPTION: .. // tratar ventana cerr
  default: ... // esta opción nunca debería alcanzarse
}
```