Para el diseño de apps se utiliza el patrón de diseño MVC (Modelo Vista Controlador)

Modelo: Se refiere al modelo de datos que utiliza el objeto. Datos y reglas del negocio. Hace el trabajo de ejecutar las cosas que tenga un programa

Vista: Como se muestra el objeto en pantalla.

Controlador: Es lo que define el comportamiento del objeto.



Contenedores: Son un tipo de componentes pensados para almacenar y manejar otros componentes.

Para hacer que un componente forme parte de un contenedor, se utiliza el método **add**. Mientras que el método **remove** es el encargado de eliminar un componente de un contenedor.

**JWindow**: representa un panel de ventana sin bordes ni elementos. Sin embargo, se puede añadir información. Suelen estar dentro de un JFrame.

**JFrame**: Objeto que representa la típica ventana con bordes, botón de cerrar, etc. No permite usar el método ADD. Un ejemplo:

JFrame frame= new JFrame(“IPOO”);

Frame.setSize(50,50);

Frame.setVisible(true);

**JPanel**: Es la clase utilizada como contenedor para agrupar componentes.

**JDialog**: Clase que genera un cuadro de dialogo. No permite usar el método ADD.

**JApplet**: Contenedor que agrupa componentes que serán mostrados en un navegador.

Se utiliza el método getContentPane() que devuelve un objeto Container que representa el área visible de la ventana. A este contenedor se le llama panel contenedor y sí permite método add. Ejemplo: JFrame ventana = new JFrame(“Prueba”)

ventana.setLocation(100,100);

Container c = ventana.getContentPane();

c.add(new JLabel (“Hola”));

ventana.pack();

ventana.setVisible(true);

El “.pack” indica al frame que defina u tamaño para incluir apropiadamente todos los componentes.

**Listeners:**

Un evento es un objeto que es lanzado por un objeto y es enviado a otro objeto llamado **LISTENER.** En la captura de estos eventos hay que tener en cuenta tres objetos implicados:

* El objeto fuente: Es el objeto que lanza los eventos. Dependiendo del tipo de objeto que sea, puede lanzar unos métodos u otros. Permite que un objeto tenga capacidad de enviar eventos. Por ejemplo, un objeto de tipo JLabel, puede lanzar evento de ratón, pero no del teclado.
* El objeto escuchador (Listener): Se trata del objeto que recibe el evento producido. Este objeto captura el evento y ejecuta el código correspondiente.
* El objeto de evento: Se trata del objeto que es enviado desde el objeto fuente al Listener.

Cada tipo de evento tiene asociado una interfaz llamada Listener para manejar ese evento.

El método **actionPerformed** es el encargado de gestionar eventos del tipo **ActionEvent** (se producen, por ejemplo, al hacer clic en un botón). Este método está implementado en la interfaz **ActionListener**. Un objeto que quiera escuchar eventos **ActionEvent**, debe implementar la interfaz **ActionListener**. Esa interfaz obliga a definir el método ya comentado **actionPerformed**. Ejemplo:

Public class miVentana extends JFrame implements ActionListeners(…)

JButton boton1 = new JButton (“prueba”);

Public miVentana(){

……

Boton1.addActionListener(this);}

Public void actionPerformed (ActionEvent e) {

Manejo del evento}

Para eliminar un Listener 🡪 boton1.removeActionListener(this)

Los Listener que escuchan notificaciones lightweigth, implementan la interface ChangeListener. Las notificaciones lightweigthson usadas para las propiedades del modelo que se modifican frecuentemente.

Para facilitar la gestión de eventos en ciertos casos, Java dispone de las llamadas clases adaptadores, que en muchos casos se evita tener que crear clases sólo para escuchar eventos. Estas clases son clases de contenido vacío pero que son muy interesantes para capturas sencillas de eventos. Todas poseen la palabra **adapter** en el nombre de clase. Por ejemplo, esta es la definición de la clase **MouseAdapter**:

Public abstract class MouseAdapter implementes **MouseListener {**

Public void mouseClicked (MouseEvent e)

Librerías a importar: javax.swing.\*;

Java.awt.\*;

 Java.awt.event.\*;



Como implementar un manejador de eventos:

1. Declarar la clase de manejador de eventos:

Public class MiClase implements ActionListener {

1. El código que registra los eventos:

unComponente.addActionListener (instancia de MiClase)

1. Implementar los métodos de la interfaz escuchadora:

Public void actionPerformed (ActionEvent e) { código que reacciona a esa acción}

Ejemplo:

**JOptionPane:**

La clase **JOptionPane** deriva de **JComponent** y es la encargada de crear este tipo ventanas informativas hacia el usuario.

JOptionPane.showMessageDialog(this, “Soy un mensaje normal”, “Cuadro 1”, JOptionPane.INFORMATION\_MESSAGE); es un ejemplo.

Luego, están los cuadros de confirmación, donde se espera una respuesta del usuario para ver si acepta o declina.

Algunas opciones:

* JOptionPane.NO\_OPTION. El usuario no pulsó ningún botón en el cuadro
* JOptionPane.CLOSE\_OPTION. El usuario cerró sin elegir nada
* JOptionPane.OK\_OPTION. El usuario pulsó OK
* JOptionPane.YES\_OPTION. El usuario pulsó el botón Sí
* JOptionPane.CANCEL\_OPTION. El usuario pulsó el botón Cancelar

Y para modificar el cuadro de opciones:

* JOptionPane.OK\_CANCEL\_OPTION. Cuadro con los botones OK y Cancelar
* JOptionPane.YES\_NO\_OPTION. Cuadro con botones Sí y No

Ejemplo:

 Cuadros de dialogo para que rellene el usuario, es decir, para que el usuario ingrese desde teclado alguna información pedida:

Ejemplo:

**Pasos:**

Frame: JFrame – JDialog

Panels: Layout – Boxes

Componentes: JComponent – JLabel – JtextField – JButton – JList

Para crear una app con swing:

1. Crear un JFrame
2. Ponerle los componentes necesarios.
3. Mostrar el JFrame con: setVisible(true)

**JPanel**: es un contenedor de objetos que puede ser desplegados. Un panel puede tener componentes u otro panel.

**Layout:**

La ubicación y tamaño de los componentes en un contenedor están determinados por el **LAYOUT.** Este, toma el control de todos los componentes en el contenedor.

Existen 5 tipos de LAYOUT:

* BorderLayout: Su estructura está compuesta por 5 regiones:
  + - * + BorderLayout.NORTH
        + BorderLayout.SOUTH
        + BorderLayout.EAST
        + BorderLayout.WEST
        + BorderLayout.CENTER
        + Cuando se agrega sin especificar el lugar, se agrega en el centro.
        + El Frame tiene por defecto este BorderLayout.
        + Se puede agregar solo una componente por región
        + Los constructores son:

new BorderLayout()🡪 Por defecto🡪no existe espacio entre las regiones.

new BorderLayout(horiz, vert)🡪Se le indica el espacio entre las componentes

new BorderLayout(botón, región)🡪Se pone el componente en la región especificada.

* CardLayout: Una componente a la vez.
* FlowLayout: Ubica a los componentes en una línea, cada vez que se completa una línea, se comienza una nueva abajo. Los constructores de este son:
  + - * + FlowLayout();
        + FlowLayout(alineación);
        + FlowLayout(alineación, horizontal, vertical)
        + Alineación puede ser: FlowLayout.CENTER/LEF/RIGHT. (Alguna de esas)
        + Horizontal y Vertical se utilizan para asignar el espacio entre los componentes.
        + Ejemplo: contenedor.setLayout(new FlowLayout(FlowLayout.CENTER,5,10)
* GridBagLayout
* GridLayout: Se ubican los componentes en una tabla con filas y columnas, que se especifican al crearla. Para crearla: new GridLayout(filas, columnas). Las celdas se ocupan de izquierda a derecha y de arriba abajo. Otra forma de crear: new GridLayout(filas, columnas, horiz, vert)🡪 donde en esta se indica el espacio entre las componentes. Si se pone un 0 en filas y/o columnas, estas crecerán dinámicamente conforme agreguemos elementos.

Cada tipo de contenedor tiene asociado un Layout por defecto. Y se le puede cambiar con setLayout(nuevo Layout)

**Listener (2):**

Para procesar los eventos, una app debe tener un evento Listener, que es un objeto que se registra con un componente para un evento en particular. Cuando ocurre el evento, el Listener es notificado mediante la invocación de un método.

Para crear un Listener se debe declarar alguna clase que implemente alguna interfaz Listener, algunas de estas son:

* ActionListener
* MouseListener
* KeyListener
* WindowListener

Ejemplo: class EdadKeyListener implements KeyListener {

void keyTyped(KeyEvent e) {

char c = e.getKeyChar();

if (!Character.isDigit(c) {

// error, solo se permiten números} }

void keyPressed(KeyEvent e) { }

void keyReleased(KeyEvent e) { }

}

Pero existe unas clases adapter que implementan la interfaz correspondiente y hace que no tengamos que implementar o definir métodos que no interesan como en el caso anterior.

Ejemplo: class EdadKeyListener extends KeyAdapter {

void keyTyped(KeyEvent e) {

char c = e.getKeyChar();

if (!Character.isDigit(c) {

// error, solo se permiten números }

**Como se usa:** (Con el ejemplo anterior)

TextField edad= new TextField(3);

edad.addKeyListener(new EdadKeyListener());

Las listas y tablas utilizan unos Listener especiales denominados **SelectionListener.**

Cada componente permite agregar un ActionListener, y no es necesario definir KeyEvent o MouseEvent para esto. Un ejemplo de ActionListener:

JButton btnCancelar= new JButton(“Cancelar”);

btnCancelar.addActionListner(new CancelarListener());

class CancelarListener implements ActionListener{

void actionPerformed(ActionEvent e) {

**dialogo**.setVisible(false);

**JDialog:**

Un dialogo es un frame que permite recolectar datos para realizar algún procesamiento. Existe un **JDialog** para esto que permite definir diálogos modales y no modales. Si es modal, cuando se activa, no se puede acceder a ningún otro elemento del programa. Para crear un dialogo: JDialog dialog = new JDialog ( frame, “Titulo”, boolean) 🡪El boolean indica si es modal o no.

setDefaultCloseOperation(EXIT\_ON\_CLOSE) 🡪Se pone para que al darle click de cerrar la ventana, se cierre y deje de ejecutarse.

setTitle(“TITULO”) 🡪Establecemos el titulo de la ventana.

setLocationRelativeTo(NULL) 🡪Aparece la app en el medio de la pantalla.

Dentro del panel, se colocan los componentes, y ese panel, se coloca en el frame/ventana. Ya que no se pueden colocar los componentes directamente en el frame.

Para agregar un panel a un frame 🡪this.getContentPane().add(nombrePanel);

Los JLabel, son etiquetas que pueden ir sobre los JPanels. Para crear: JLabel nombre = new JLabel (“Etiqueta”). Se puede usar el método setBounds() para elegir la posición de la etiqueta. Pero para eso hay que desactivar el Layout del panel.

Para desactivar el Layout por defecto 🡪nombrePanel.setLayout(null);

Para agregar una imagen: Creamos un JLabel 🡪 JLabel nombre = new JLabel (new ImageIcon(NombreFoto.jpg); Luego lo seteamos en el lugar que querramos con: nombre.setBounds(parametros). Y la agregamos al panel 🡪 nombrePanel.add(nombreEtiqueta);