Universidad Autónoma del Estado de México

UAEM

CENTRO UNIVERSITARIO
UAEM ZUMPANGO
INGENIERÍA EN COMPUTACIÓN

Unidad de Aprendizaje:

Programación Orientada a Objetos

Unidad de Competencia V:

Interfaz Gráfica de Usuario

M. en C. Edith Cristina Herrera Luna

Mayo 2018

PROGRAMACIÓN ORIENTADA A OBJETO

Propósito de la Unidad de Aprendizaje

- El alumno conocerá los conceptos de la programación orientado a objetos y su implementación en un lenguaje apropiado, los cuales servirán de base para unidades de aprendizaje encaminadas al análisis, el diseño y la elaboración de aplicaciones informáticas.

Programación Orientada a Objetos
INTRODUCCIÓN

- El conocimiento en el desarrollo de aplicaciones basadas en el paradigma de programación orientado a objetos, se vuelve un requisito en la formación académica del ingeniero en sus diferentes ramas. Con el manejo de un lenguaje de programación orientada a objetos, el estudiante podrá implementar los conceptos de orientación a objetos y desarrollar, en cursos consecuentes, un número significativo de aplicaciones, en las diferentes plataformas que así lo soliciten.

- Por lo previamente expuesto y como parte de una formación integral para el ingeniero en computación, se propone el siguiente programa que consta de cinco unidades de competencia, en la primera se consideran los fundamentos teóricos sobre un lenguaje de POO y la plataforma en la que ejecuta, dentro de la segunda se abordan conocimientos sobre la sintaxis básica del lenguaje de POO y como parte de la tercera se refiere básicamente a controlar el flujo del programa.

Programación Orientada a Objetos
INTRODUCCIÓN

- En la unidad cuatro se crearan y manipularán estructuras dinámicas de datos utilizando referencias, se conocerán los procedimientos para el manejo de archivos de acceso secuencial y acceso aleatorio. En la cinco se entenderán los principios de diseño de las interfaces graficas de usuario (GUI), clases e interfaces para manejo de eventos y propiedades.

- Para lograr lo anterior se sugieren como estrategias didácticas la revisión bibliográfica y solución de ejercicios por parte del alumno y la explicación por parte del instructor de temas específicos de mayor complejidad. Para consolidar los conocimientos, también es necesario realizar ejercicios que fortalezcan la parte teórica.

- Con lo antes mencionado, el ingeniero en computación como experto en su ramo tendrá las herramientas necesarias para poder interactuar de manera holística con profesionales en otros campos del saber para así solucionar problemas en bases científico - metodológicas congruentes afrontando los retos actuales del desarrollo tecnológico y económico.

PROGRAMACIÓN ORIENTADA A OBJETOS

Unidad de Competencia V

OBJETIVO:

- Instrumentar sistemas de software mediante la aplicación de POO.
- Conocer los principios de diseño de las interfaces graficas de usuario (GUI), clases e interfaces para manejo de eventos y propiedades.

PROGRAMACIÓN ORIENTADA A OBJETOS
CONTENIDO

1. Modelo Vista Controlador (MVC)

2. Interfaz Gráfica de Usuario
   - JFC
   - Swing
   - Contenedores
   - Componentes

PROGRAMACIÓN ORIENTADA A OBJETOS
CONTENIDO

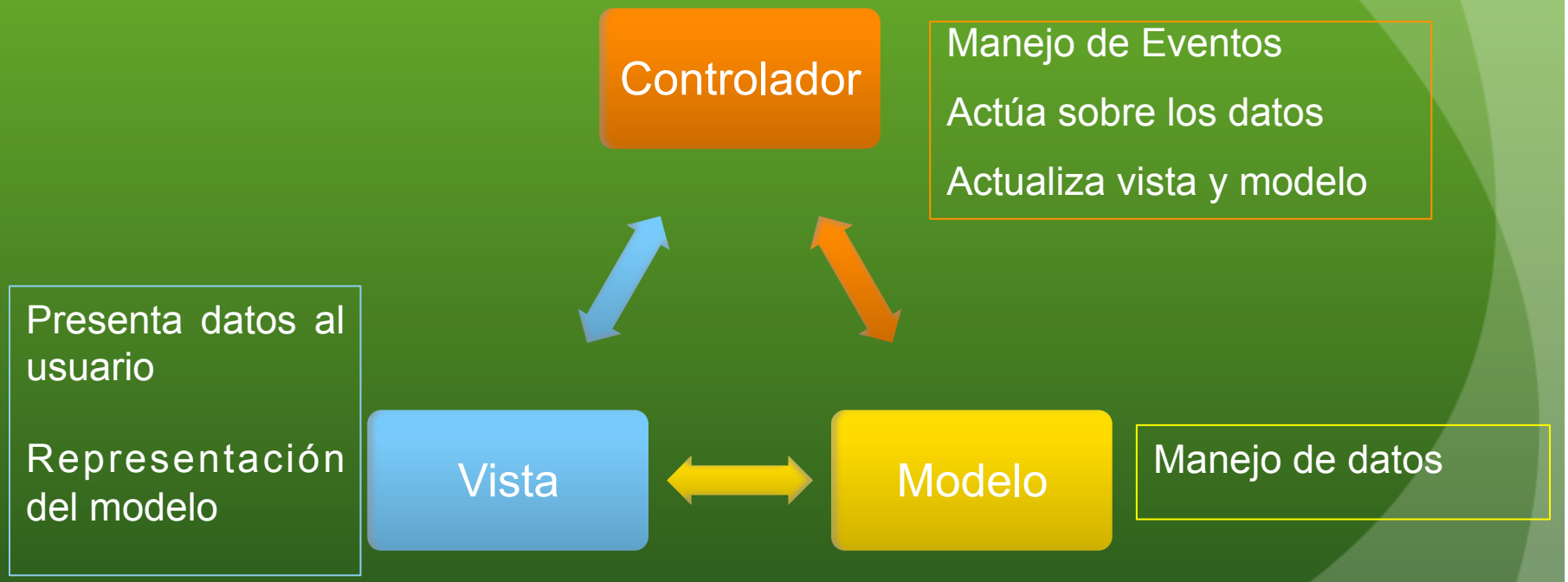
3. Manejo de Eventos
   - Tipos de eventos y componentes
   - Manejo de Eventos en propia clase
   - Manejo de Eventos clase independiente
   - Manejo de Eventos en clase anónima

Modelo Vista Controlador (MVC)

C. U. UAEM ZUMPANGO / ICO / OOP / ECHL

Modelo Vista Controlador

- Es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de su representación y el módulo encargado de gestionar los eventos.



Modelo Vista Controlador

Esta arquitectura se tienen 3 elementos:

- **Modelo**: Se refiere al estado de un componente (definido por sus atributos). Representa la información del sistema, administra el acceso a dicha información. Las peticiones de acceso o manipulación de información llegan al 'modelo' a través del 'controlador'.
- **Vista**: Es la representación del modelo, generalmente es su representación gráfica.
- **Controlador**: Es el encargado del manejo de eventos y responsable de la actualización del modelo. Hace peticiones al modelo cuando se hace solitudes de información y permite hacer cambios en la vista.

Modelo Vista Controlador

Funcionamiento:

La vista y el controlador usualmente son combinados en un objeto denominado *delegado* quien representa gráficamente el modelo (vista) y transfiere la entrada del usuario (controlador).

Los controladores tratan los eventos que se producen en la interfaz gráfica (vista).

Interfaz Gráfica de Usuario

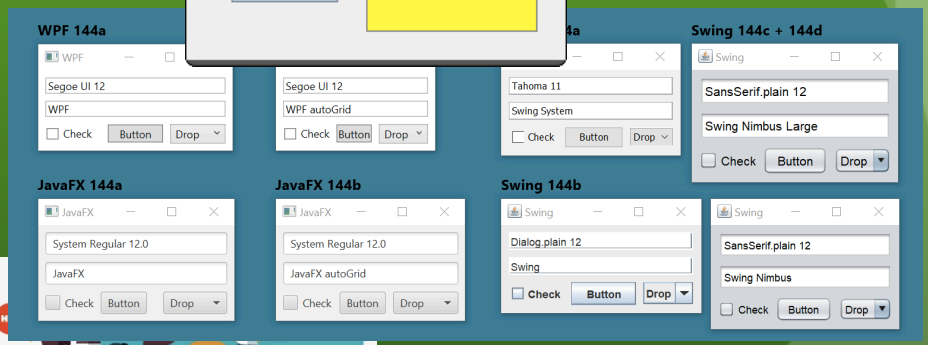
C. U. UAEM ZUMPANGO / ICO / OOP / ECHL

Interfaz Gráfica de Usuario

Para desarrollar aplicaciones con una Interfaz Grafica de Usuario (GUI – *Graphical User Interface*) en Java se tiene la biblioteca JFC *Java Foundation Classes*, conformada por las APIs:

- Swing
- Pluggable Look-and-Feel Support
- Accessibility API
- Java 2D API
- Internationalization

Interfaces para programación de aplicaciones



Swing

- Swing es flexible, poderosa y ofrece amplias posibilidades para generar GUIs. Actualmente cuenta con 18 paquetes:

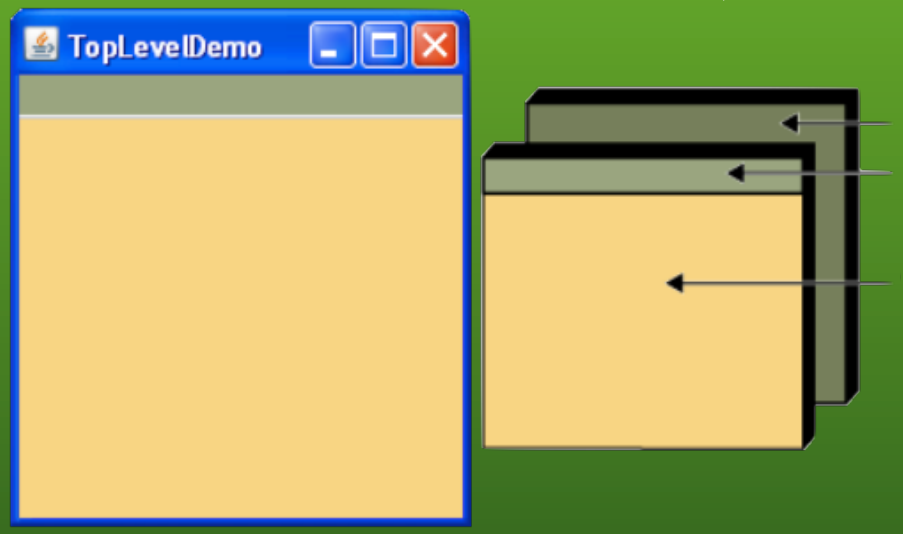
javax.accessibility	javax.swing.plaf	javax.swing.text
javax.swing	javax.swing.plaf.basic	javax.swing.text.html
javax.swing.border	javax.swing.plaf.metal	javax.swing.text.html.parser
javax.swing.colorchooser	javax.swing.plaf.multi	javax.swing.text.rtf
javax.swing.event	javax.swing.plaf.synth	javax.swing.tree
javax.swing.filechooser	javax.swing.table	javax.swing.undo

- Los componentes **swing** son implementados en código **NO NATIVO** ➜ son independientes de la plataforma.

Contenedores

- Son componentes para ubicar/organizar a otros componentes.
- Jerarquía de contenedores ➜ Marco de la Ventana

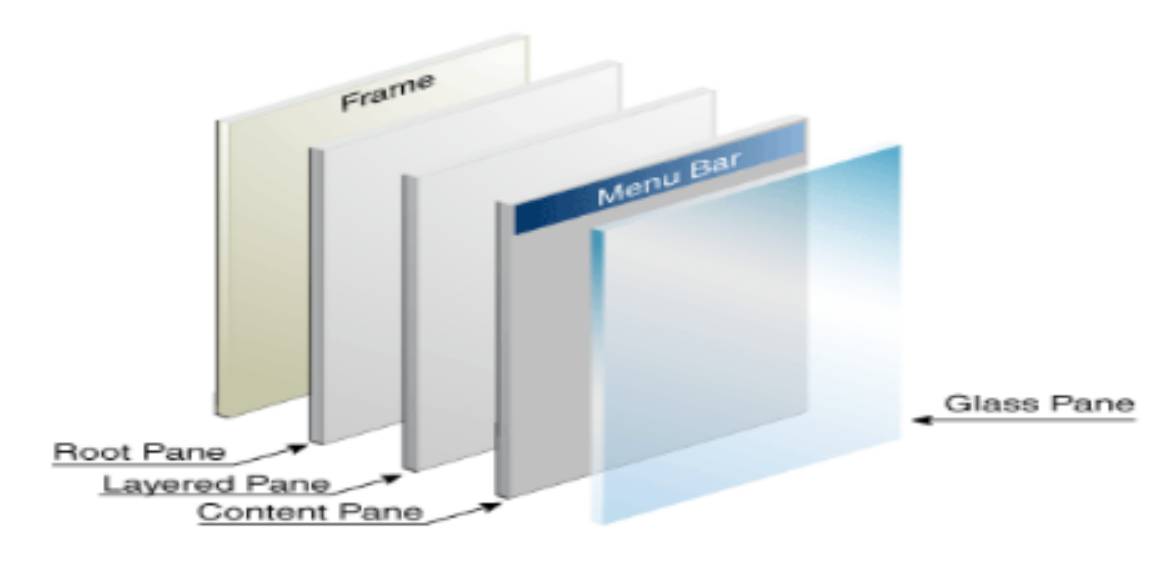
JFrame, JDialog, JApplet



Un JFrame además del marco de ventana tiene un panel principal (ContentPane) y un espacio para una barra de menú.

JFrame

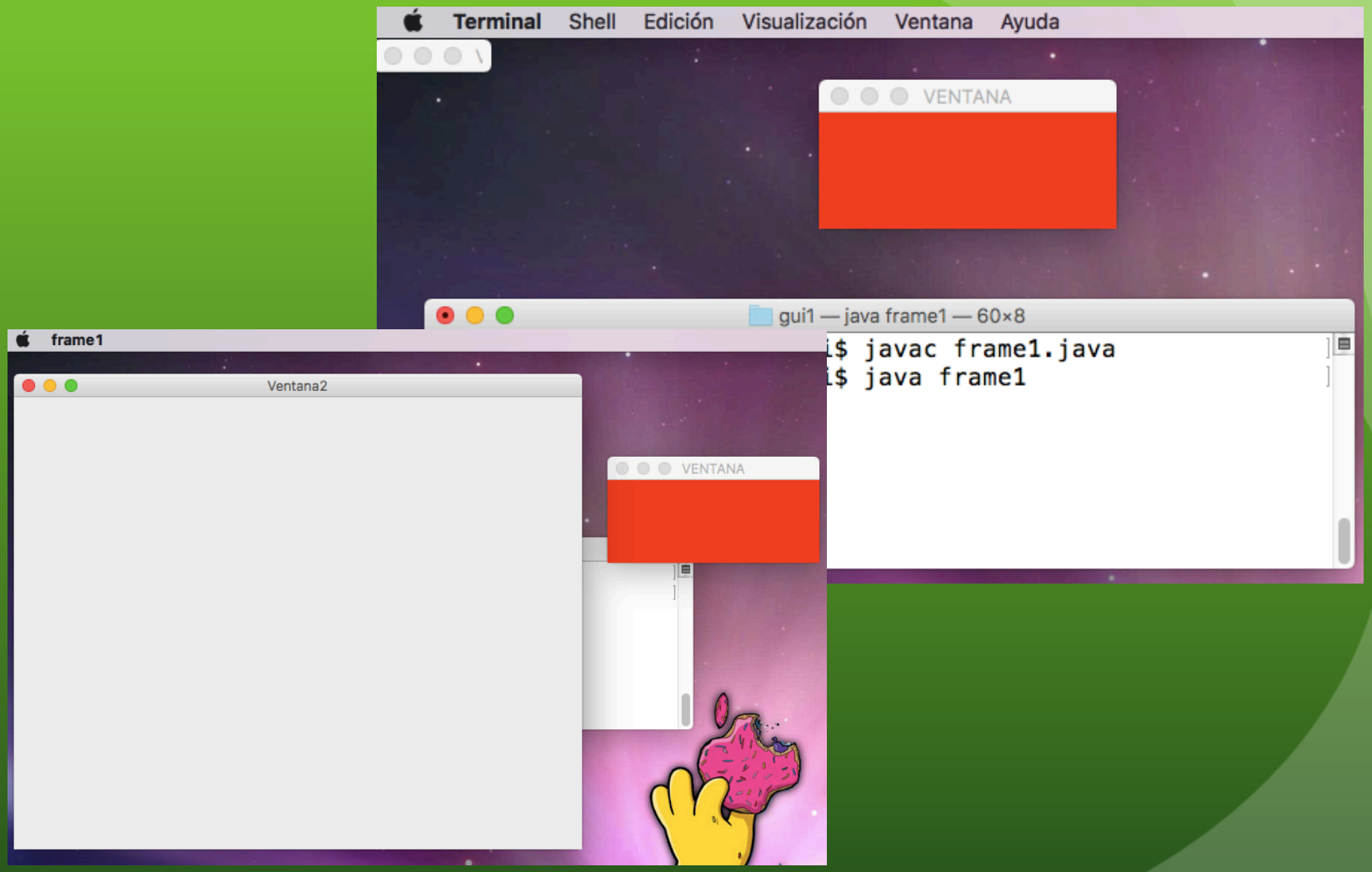
- Cada contenedor tiene un contenedor aislado intermedio llamado panel raíz.
- El panel raíz administra el panel de contenido y la barra de menú, junto con un par de otros contenedores.
  - Panel raíz ➜ ***getContentPane()***
  - Panel de contenido (intermedio) ➜ *JPanel*



JFrame: Ejemplo1


```java
import java.awt.*;
import javax.swing.*;

public class frame1
{
    public static void main(String[] args)
    {
        //Clase identificador = new Constructor();
        JFrame  ventana1 = new JFrame("VENTANA");
        ventana1.setVisible(true);
        ventana1.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);
            //JFrame.EXIT_ON_CLOSE          Termina la aplicacion
            //HIDE_ON_CLOSE                 Oculta la ventana
            //DISPOSE_ON_CLOSE              Libera la ventana
            //DO_NOTHING_ON_CLOSE

        ventana1.setSize(200, 100);             //Tamaño ancho, alto
        ventana1.setResizable(false);           //Cambiar de tamaño
        ventana1.setLocation(300, 50);          //Ubica ventana x, y
        //ventana1.setTitle("Ventana1");            //Título

        JFrame ventana2 = new JFrame("Ventana2");
        ventana2.setVisible(true);
        ventana2.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);

        JPanel panel1 = new JPanel();       //Crear panel
        panel1.setBackground(Color.RED);    //Cambiar color
        ventana1.add(panel1);               //Agregar componente
    }
}
```

JFrame: Ejemplo1



JFrame: Ejemplo2

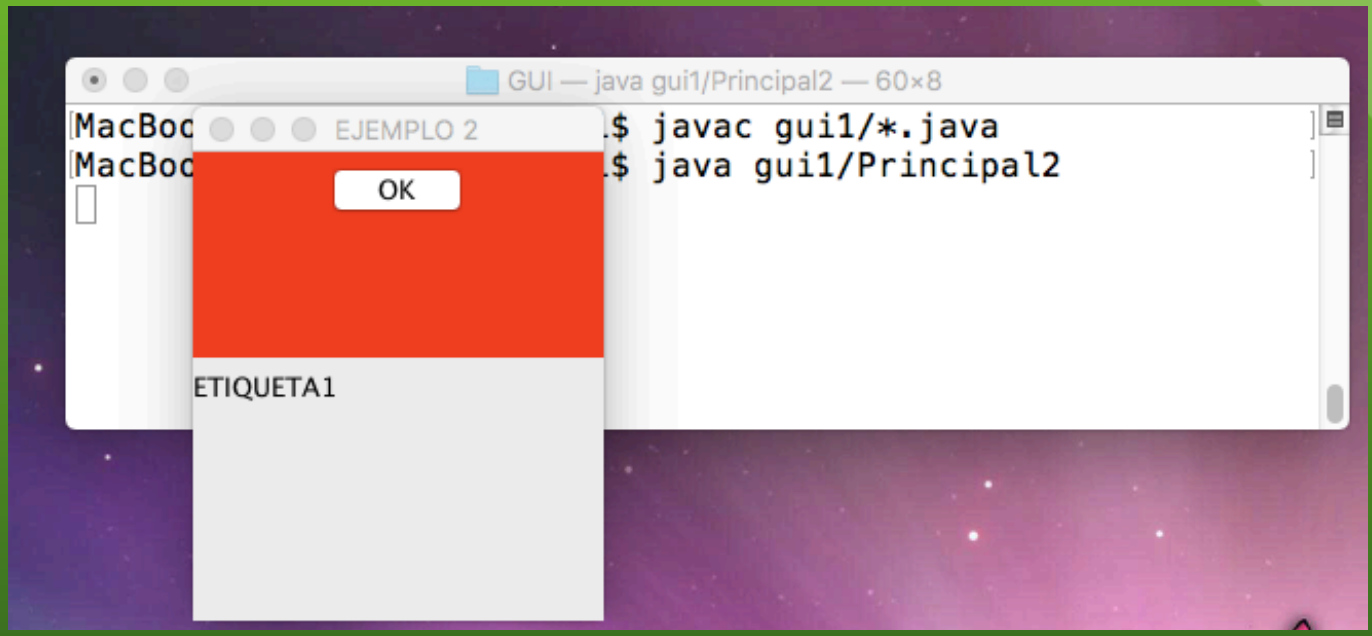
```java
package gui1;
import java.awt.*;
import javax.swing.*;

public class frame2 extends JFrame
{
    JPanel panel1, panel2;
    JLabel eti1;
    JButton btn1;
    JTextField txt1;

    public frame2()
    {
        this.setTitle("EJEMPLO 2");
        setSize(200, 250);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        //Panel
        panel1 = new JPanel();      //crear panel
        panel1.setBackground(Color.RED);    //cambiar color
        panel1.setSize(200,100);
        //panel1.setLayout( new GridLayout(2,2) );
                                    //agregar componente
        panel2 = new JPanel();
        panel2.setBackground( Color.BLUE );
        //panel2.setLayout( new FlowLayout() );
        add(panel1);
        add(panel2);
        initComponents();
        this.setVisible(true);
    }
```

JFrame: Ejemplo2

```
31
32      public void initComponents()
33      {
34          eti1 = new JLabel("ETIQUETA1");
35          btn1 = new JButton("OK");
36          txt1 = new JTextField(" ETIQUETA ");
37          add( eti1 );
38          panel1.add( btn1 );
39          panel2.add( txt1 );
40      }
41
42  }//clase
```

```
1   package gui1;
2   import java.awt.*;
3   import javax.swing.*;
4
5   public class Principal2
6   {
7       public static void main( String[] args)
8       {
9           frame2 obj = new frame2();
10          //frame3 obj2 = new frame3();
11      }
12  }
```

JFrame: Ejemplo2



Componentes

- Con excepción de los contenedores, todos los componentes Swing inician con la letra "**j**" y descienden de la clase **JComponent**. Esta clase provee funcionalidades como:
  - Tool tips.
  - Bordes y pintura.
  - Apoyo para accesibilidad, para el diseño (layout).
  - Apoyo para arrastrar y soltar.
  - Etc.

- Declara el componente *TipoComponente identificador;*
- Crea el componente *identificador = new Constructor( );*
- Modifica los atributos del componente
- Establece la posición del componente en el contenedor

JLabel

JLabel identitificador = new **JLabel**(“Texto Etiqueta”);

ImageIcon icono = createImageIcon (“gui1/imagen.gif ”)

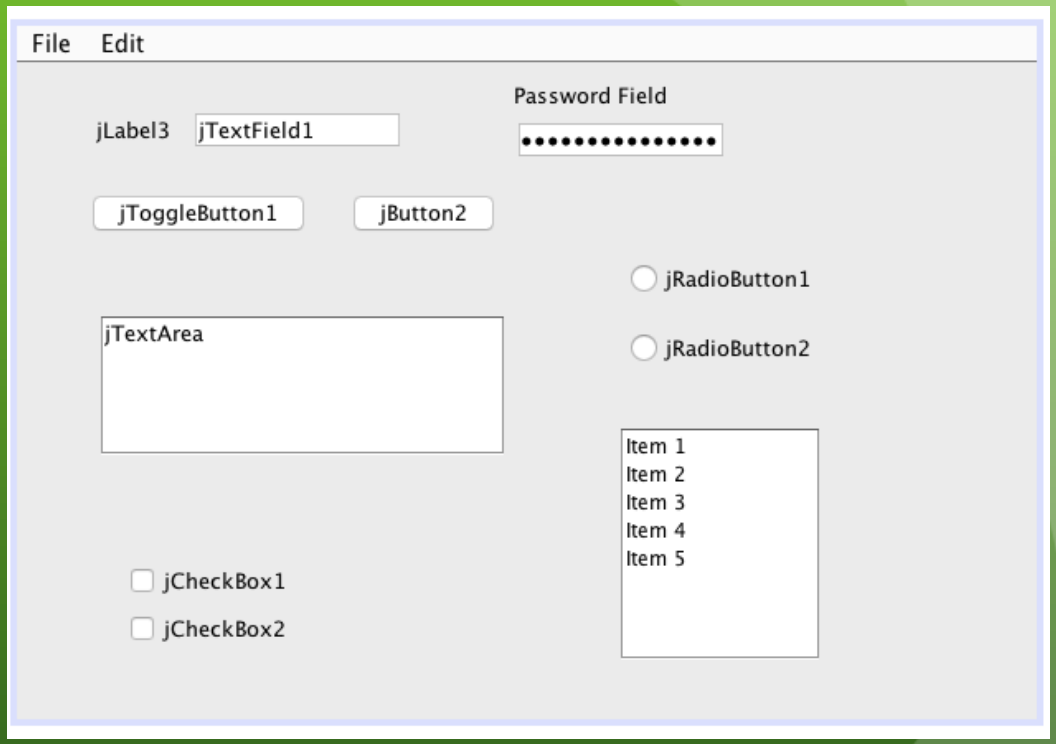
JLabel et1 = new JLabel(“Texto e imagen”, icono, JLabel.CENTER);

//Leading, Trailing

- String ← getText( );
- setText(String) ;
- setIcon( );
- setFont( );
- setForeground( );
- setOpaque( );
- setSize( );

Componentes de Texto

- jTextField
- jFormattedTextField
- jPasswordField
- jTextArea
- jEditorPane
- jTextPane



JTextField

```
JTextField  campoTxt1 = new JTextField ( 30 );

JTextField  campoTxt2 = new JTextField ( “Texto” );
```

Metodo	Descripción
setText	
getText	
setEditable	
setFont	
setColumns	
getColumns	
getColumnWidth	

JTextArea

```
JTextArea  areaTxt1 = new JTextArea ( 5, 20 );

JTextArea  areaTxt2 = new JTextField ( );

JScrollPane scroll = new JScrollPane( areaTxt2 );
```

Metodo	Descripción
setText	
append	
insert	
getLineCount	
getRows	
setRows	
setAutoscrolls	

JButton

```
JButton  boton1 = new JButton ( );

JButton  boton2 = new JButton ( “Texto” );

JButton  boton2 = new JButton ( “Icon” );
```

Metodo	Descripción
setText	
setLabel	
setMnemonic	
setIcon	
setPressedIcon	
setCursor	
setToolTipText	

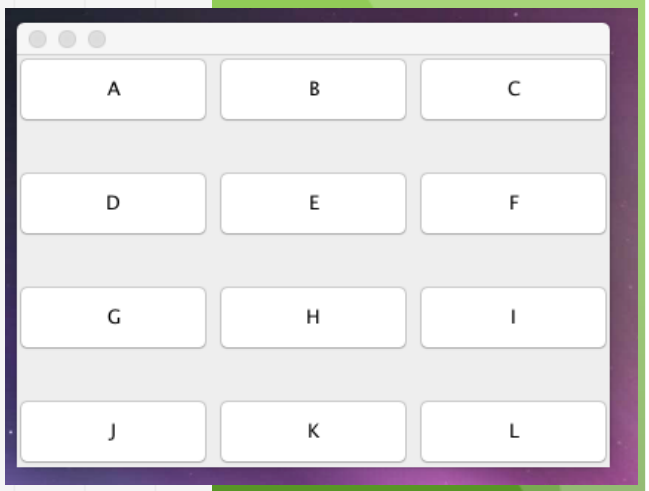
JCheckBox - JRadioButton

```
JCheckBox  check1 = new JCheckBox ( “Texto” );

JRadioButton  radio1 = new JRadioButton ( “Texto” );
```

Metodo	Descripción
isSelected	
setEnable	
setIcon	
setSelected	
getActionCommand	
*** ButtonGroup	

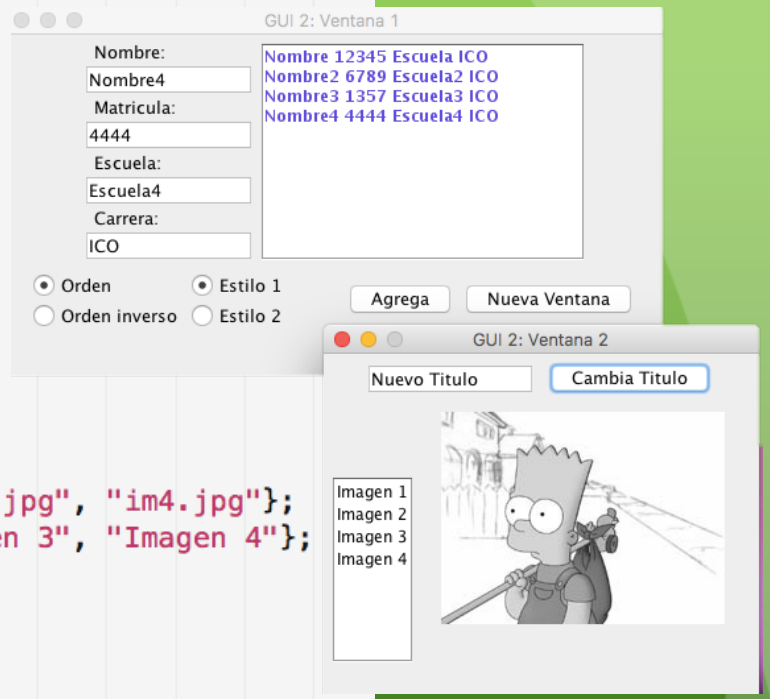
GUI: Ejemplo1

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class duda1 extends JFrame
{
    GridLayout lyt;
    JButton[] btn = new JButton[12];

    public duda1() {
        initComponents();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(400,300);
        setResizable(false);
        setVisible(true);
    }

    public void initComponents(){
        lyt = new GridLayout(4, 3, 5, 30);  // filas, columnas, espacio
        setLayout(lyt);
        for( int i=0; i<12; i++){
            btn[i] = new JButton( String.valueOf((char)(i+65)));
            add(btn[i]);
        }
    }

    public static void main( String[] args){
        duda1 obj = new duda1();
    }
}//clase
```



GUI: Ejemplo2

```
1  package Practica24v;
2  import javax.swing.*;
3  import javax.swing.event.*;
4  import java.awt.*;
5  import java.awt.event.*;
6
7  public class Ventana2 extends JFrame
8  {
9      JButton b1;
10     JTextField t1;
11     JLabel e1;
12     JList lista;
13     JScrollPane scroll;
14     String[] imgRuta = {"im1.jpeg","im2.jpg","im3.jpg", "im4.jpg"};
15     String[] img = {"Imagen 1", "Imagen 2", "Imagen 3", "Imagen 4"};
16     ImageIcon im;
17     JPanel pan1, pan2, pan3;
18
19     public Ventana2()
20     {
21         super("GUI 2: Ventana 2");
22         setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
23         setLayout( new FlowLayout() );
24         setPreferredSize( new Dimension( 330,350 ) );
25         initComponents();
26         setResizable(false);
27         pack();
28         setVisible(true);
29     }//constructor
30
```



GUI: Ejemplo2

```
31      public void initComponents()
32      {
33          b1 = new JButton( "Cambia Titulo");
34          t1 = new JTextField("Nuevo Titulo",10);
35          e1 = new JLabel( new ImageIcon( getClass().getResource( "im1.jpeg" ) ) );
36          lista = new JList(img);
37          scroll = new JScrollPane (lista );
38
39          pan1 = new JPanel();
40  //      pan1.setBackground(Color.RED);
41          pan1.setPreferredSize( new Dimension(250,250) );
42          e1.setSize( new Dimension(240,240) );
43
44          b1.addActionListener( new ActionListener(){
45              public void actionPerformed( ActionEvent e)
46              {
47                  setTitle( t1.getText() );
48              }
49          } );
50
51          ManejoLista obj = new ManejoLista();
52          lista.addListSelectionListener( obj );
53          pan1.add(e1);
54          add(t1);
55          add(b1);
56          add(scroll);
57          add(pan1);
58      }//initC
```


```
59
60      public class ManejoLista implements ListSelectionListener
61      {
62          public void valueChanged( ListSelectionEvent e)
63          {
64              if (e.getValueIsAdjusting() == true)
65              {
66
67                  int op = lista.getSelectedIndex();
68                  if( op == -1)
69                      JOptionPane.showMessageDialog(null,"No se seleccionó imagen","
70                      ", JOptionPane.ERROR_MESSAGE);
71                  else
72                  {
73                      System.out.println(imgRuta[op]);
74                      im = new ImageIcon( getClass().getResource( imgRuta[op] ) );
75                      e1.setIcon(im);
76                  }
77              }
78          }
79      }
80  }//clase
```

GUI: Ejemplo2


```
1  package Practica24v;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.Random;
6
7  public class Ventana1 extends JFrame
8  {
9      JButton b1, b2;
10     JTextField t1, t2, t3, t4;
11     JRadioButton r1, r2, r3, r4;
12     ButtonGroup g1, g2;
13     JTextArea area;
14     JScrollPane scroll;
15     JLabel e1,e2,e3,e4;
16     JPanel pan1, pan2, pan3;
17     Font f1 = new Font("LucidaSans", Font.BOLD, 12);
18     Font f2 = new Font("Arial", Font.ITALIC, 18);
19
20     public Ventana1() {
21         super("GUI 2: Ventana 1");
22         setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
23         setSize(500,280);
24         setLayout( new FlowLayout() );
25         initComponents();
26         setResizable(false);
27         setVisible(true);
28     }//constructor
29
30     public void initComponents(){
31         b1 = new JButton( "Agrega");
32         b2 = new JButton("Nueva Ventana");
33         t1 = new JTextField("",10);
34         t2 = new JTextField("",10);
35         t3 = new JTextField("",10);
36         t4 = new JTextField("",10);
37         e1 = new JLabel("Nombre: ");
38         e2 = new JLabel("Matricula: ");
39         e3 = new JLabel("Escuela: ");
```

GUI: Ejemplo2

```
40          e4 = new JLabel("Carrera: ");
41          r1 = new JRadioButton("Orden");
42          r2 = new JRadioButton("Orden inverso");
43          r3 = new JRadioButton("Estilo 1");
44          r4 = new JRadioButton("Estilo 2");
45          g1 = new ButtonGroup();
46          g2 = new ButtonGroup();
47          area = new JTextArea(10,20);
48          scroll = new JScrollPane(area);
49          pan1 = new JPanel();
50          pan1.setLayout( new BoxLayout(pan1, BoxLayout.Y_AXIS) );
51          //pan1.setBackground( Color.YELLOW );
52          pan2 = new JPanel();
53          pan2.setLayout( new GridLayout(2,2) );
54          //pan2.setBackground(Color.ORANGE);
55          pan3 = new JPanel();
56          pan3.setLayout( new BoxLayout(pan3, BoxLayout.X_AXIS) );
57          //pan3.setBackground(Color.BLUE);
58
59          b1.addActionListener( new ManejoEvento() );
60          b2.addActionListener( new ActionListener(){
61              public void actionPerformed( ActionEvent e ){
62                  Ventana2 obj = new Ventana2();
63              }
64          } );
65
66          pan1.add(e1);       pan1.add(t1);
67          pan1.add(e2);       pan1.add(t2);
68          pan1.add(e3);       pan1.add(t3);
69          pan1.add(e4);       pan1.add(t4);
70          pan2.add(r1);       pan2.add(r3);
71          pan2.add(r2);       pan2.add(r4);
72          g1.add( r1);        g1.add( r2);
73          g2.add( r3);        g2.add( r4);
74          pan3.add( pan2 );   pan3.add( b1 );
75          pan3.add( b2 );     add( pan1 );
76          add( scroll );      add( pan3 );
77      }//initC
78
```

GUI: Ejemplo2

```
79      public static void main(String[] args){
80          Ventana1 obj = new Ventana1();
81      }
82
83      public class ManejoEvento implements ActionListener{
84          public void actionPerformed( ActionEvent e ){
85              boolean ord = r1.isSelected();
86              boolean est1 = r3.isSelected();
87              String txt = "";
88              if( ord ) {
89                  txt += t1.getText() +" ";
90                  txt += t2.getText() +" ";
91                  txt += t3.getText() +" ";
92                  txt += t4.getText() +"\n";
93              } else {
94                  txt += t4.getText() +" ";
95                  txt += t3.getText() +" ";
96                  txt += t2.getText() +" ";
97                  txt += t1.getText() +"\n";
98              }
99
100             if( est1 )
101                 area.setFont( f1 );
102             else
103                 area.setFont( f2 );
104             Random ale = new Random();
105             area.setForeground( new Color( ale.nextInt(256),
106                                 ale.nextInt(256),ale.nextInt(256)) );
107             area.append( txt );
108         }
109     }
110 }//clase
```

Manejo de Eventos

C. U. UAEM ZUMPANGO / ICO / OOP / ECHL

¿Qué es un Evento?

- Un ***evento*** es un mecanismo mediante el cual un objeto puede notificar de la ocurrencia de un suceso.
- Un ***evento*** es un objeto que describe un cambio de estado en otro objeto.

Cuando un evento ocurre en un objeto o fuente especifico es tratado por medio de un “oyente de eventos”, *Listener;* quien tiene información del evento ocurrido y la fuente del mismo.

Existen diferentes tipos de **listeners** de acuerdo al evento que se necesita tratar.

El manejo de eventos consiste en agregar un listener a un componente, y sobreescribir uno o varios métodos dados en una interfaz.

Manejo de eventos

Todos los componentes Swing soportan los listener:

- ComponentListener: Cambios en tamaño, posición, visibilidad.
- FocusListener
- KeyListener
- MouseListener: Clic, presionar, liberar, etc.
- MouseMotionListener: Cambios en la posición del cursor sobre el componente
- HierarchyListener: Cambios en jerarquía de componentes por eventos.

Listener API Table

La siguiente tabla -Tutorial de Oracle-, muestra algunos de los Listener disponibles y los métodos que se deben sobreescribir.

Listener Interface	Adapter Class	Listener Methods
ActionListener	*none*	actionPerformed(ActionEvent)
AncestorListener	*none*	ancestorAdded(AncestorEvent) ancestorMoved(AncestorEvent) ancestorRemoved(AncestorEvent)
CaretListener	*none*	caretUpdate(CaretEvent)
CellEditorListener	*none*	editingStopped(ChangeEvent) editingCanceled(ChangeEvent)
ChangeListener	*none*	stateChanged(ChangeEvent)
ComponentListener	ComponentAdapter	componentHidden(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
ContainerListener	ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
DocumentListener	*none*	changedUpdate(DocumentEvent) insertUpdate(DocumentEvent) removeUpdate(DocumentEvent)
ExceptionListener	*none*	exceptionThrown(Exception)
FocusListener	FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)
HierarchyBoundsListener	HierarchyBoundsAdapter	ancestorMoved(HierarchyEvent) ancestorResized(HierarchyEvent)

Listener API Table

HierarchyListener	*none*	hierarchyChanged(HierarchyEvent)
HyperlinkListener	*none*	hyperlinkUpdate(HyperlinkEvent)
InputMethodListener	*none*	caretPositionChanged(InputMethodEvent) inputMethodTextChanged(InputMethodEvent)
InternalFrameListener	InternalFrameAdapter	internalFrameActivated(InternalFrameEvent) internalFrameClosed(InternalFrameEvent) internalFrameClosing(InternalFrameEvent) internalFrameDeactivated(InternalFrameEvent) internalFrameDeiconified(InternalFrameEvent) internalFrameIconified(InternalFrameEvent) internalFrameOpened(InternalFrameEvent)
ItemListener	*none*	itemStateChanged(ItemEvent)
KeyListener	KeyAdapter	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
ListDataListener	*none*	contentsChanged(ListDataEvent) intervalAdded(ListDataEvent) intervalRemoved(ListDataEvent)
ListSelectionListener	*none*	valueChanged(ListSelectionEvent)
MenuDragMouseListener	*none*	menuDragMouseDragged(MenuDragMouseEvent) menuDragMouseEntered(MenuDragMouseEvent) menuDragMouseExited(MenuDragMouseEvent) menuDragMouseReleased(MenuDragMouseEvent)
MenuKeyListener	*none*	menuKeyPressed(MenuKeyEvent) menuKeyReleased(MenuKeyEvent) menuKeyTyped(MenuKeyEvent)
MenuListener	*none*	menuCanceled(MenuEvent) menuDeselected(MenuEvent) menuSelected(MenuEvent)

Listener API Table

MouseInputListener (extends MouseListener and MouseMotionListener	MouseInputAdapter MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent) mouseDragged(MouseEvent) mouseMoved(MouseEvent) MouseAdapter(MouseEvent)
MouseListener	MouseAdapter, MouseInputAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	MouseMotionAdapter, MouseInputAdapter	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
MouseWheelListener	MouseAdapter	mouseWheelMoved(MouseWheelEvent) MouseAdapter<MouseEvent>
PopupMenuListener	*none*	popupMenuCanceled(PopupMenuEvent) popupMenuWillBecomeInvisible(PopupMenuEvent) popupMenuWillBecomeVisible(PopupMenuEvent)
WindowListener	WindowAdapter	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)
WindowStateListener	WindowAdapter	windowStateChanged(WindowEvent)

Componentes y Listener

Component	Action Listener	Caret Listener	Change Listener	Document Listener, Undoable Edit Listener	Item Listener	List Selection Listener	Window Listener	Other Types of Listeners
button	✔		✔		✔			
check box	✔		✔		✔			
color chooser			✔					
combo box	✔				✔			
dialog							✔	
editor pane		✔		✔				hyperlink
file chooser	✔							
formatted text field	✔	✔		✔				
frame							✔	
internal frame								internal frame
list						✔		list data
menu								menu
menu item	✔		✔		✔			menu key menu drag mouse
option pane								
password field	✔	✔		✔				

Componentes y Listener

Component	Action Listener	Caret Listener	Change Listener	Document Listener, Undoable Edit Listener	Item Listener	List Selection Listener	Window Listener	Other Types of Listeners
popup menu								popup menu
progress bar			✔					
radio button	✔		✔		✔			
slider			✔					
spinner			✔					
tabbed pane			✔					
table						✔		table model table column model cell editor
text area		✔		✔				
text field	✔	✔		✔				
text pane		✔		✔				hyperlink
toggle button	✔		✔		✔			
tree								tree expansion tree will expand tree model tree selection
viewport (used by scrollpane)			✔					

Manejo de Eventos: Clase

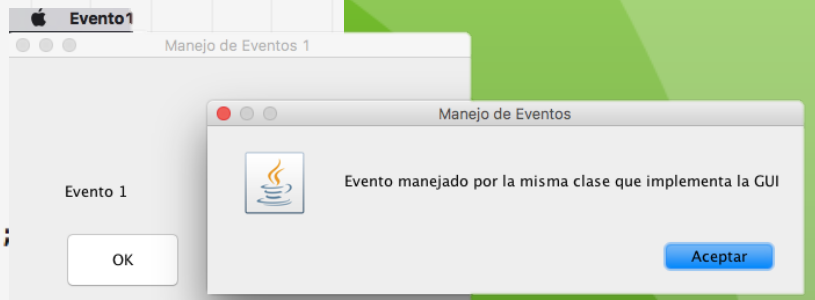
Existen varias formas de manejar eventos en java, mostraremos 3 ejemplos: clase anónima, clase interna y la clase de la GUI maneja el evento.

- **La misma clase que define la GUI puede ser la encargada de manejar los eventos.**

1. Define el tipo de evento que se utilizará e implementa la interfaz.
2. Sobreescribe los métodos de la interfaz.
3. Agrega el oyente o listener al componente.

Manejo de Eventos: Clase

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Evento1 extends JFrame implements ActionListener {

    public JLabel et1;
    public JButton boton1;

    public Evento1(){
        setTitle("Manejo de Eventos 1");
        setSize(400,400);
        setLayout(null);
        initComponents();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    public void initComponents(){
        et1 = new JLabel("Evento 1");
        boton1 = new JButton("OK");
        et1.setBounds(50,100,70,30);
        boton1.setBounds(50,150,100,50);
        boton1.addActionListener( this );
        add(et1);
        add(boton1);
    }

    public void actionPerformed(ActionEvent ev){
        JOptionPane.showMessageDialog(this,"Evento manejado por la misma"
            +" clase que implementa la GUI", "Manejo de Eventos",
             JOptionPane.INFORMATION_MESSAGE);
    }

    public static void main( String[] args){
            Evento1 ventana = new Evento1();
    }//main
}//Frame
```



Manejo de Eventos: Clase interna

Existe una clase que exclusivamente maneja el evento. Es de utilidad cuando se incorpora la misma acción sobre diversos componentes.

1. Crea una clase interna
2. Define el tipo de evento que se utilizará e implementa la interfaz.
3. Sobreescribe los métodos de la interfaz.
4. Crea un oyente o listener y agregalo al componente.

Manejo de Eventos: Clase interna

```
3  import javax.swing.*;
4  public class Evento2 extends JFrame{
5
6      public JLabel et1;
7      public JButton boton1;
8
9      public Evento2(){
10         setTitle("Manejo de Eventos 1");
11         setSize(400,400);
12         setLayout(null);
13         initComponents();
14         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15         setVisible(true);
16     }
17
18     public void initComponents(){
19         et1 = new JLabel("Evento 1");
20         boton1 = new JButton("OK");
21         et1.setBounds(50,100,70,30);
22         boton1.setBounds(50,150,100,50);
23         Oyente listener1 = new Oyente();
24         boton1.addActionListener( listener1 );
25         add(et1);
26         add(boton1);
27     }
28
29     public static void main( String[] args){
30             Evento2 ventana = new Evento2();
31     }//main
32
33     public class Oyente implements ActionListener{
34
35         public void actionPerformed(ActionEvent ev){
36         JOptionPane.showMessageDialog(null,"Evento manejado por la misma"
37             +" clase que implementa la GUI", "Manejo de Eventos",
38              JOptionPane.INFORMATION_MESSAGE);
39         }
40     }//CLASE OYENTE
41 }//Frame
```

Manejo de Eventos: Clase anónima

Existe una clase que no tiene nombre, pero se crea específicamente para el manejo del evento. Es de utilidad cuando se incorporan acciones sencillas a un componente.

1. Define el tipo de evento que se utilizará
2. Agrega un listener al componente: Al pasar el argumento listener, se debe de crear un objeto LISTENER y se sobreescriben los métodos.

Manejo de Eventos: Clase anónima

```java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Evento3 extends JFrame{

    public JLabel et1;
    public JButton boton1;

    public Evento3(){
        setTitle("Manejo de Eventos 3");
        setSize(400,400);
        setLayout(null);
        initComponents();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    public void initComponents(){
        et1 = new JLabel("Evento 3");
        boton1 = new JButton("OK");
        et1.setBounds(50,100,70,30);
        boton1.setBounds(50,150,100,50);

        boton1.addActionListener( new ActionListener(){
            public void actionPerformed(ActionEvent ev){
                JOptionPane.showMessageDialog(null,"Evento manejado por una clase"
                +" anónima", "Manejo de Eventos", JOptionPane.INFORMATION_MESSAGE);
            }
        } );

        add(et1);
        add(boton1);
    }

    public static void main( String[] args){
            Evento3 ventana = new Evento3();
    }//main

}//Frame
```

Referencias

- Wu Thomas. *Introducción a la programación orientada a objetos*. Mc Graw Hill (2008).
- Deytel Harvey M y Paul J. Deytel. *Como programar en Java*, 7a edición, Prentice Hall (2008).
- Drozdek Adam. *Estructuras de datos y algoritmos con Java*, 2a edición. Thomson Learning (2007).
- Goodrich Michel T. y Tamassia Roberto. *Estructuras y datos y algoritmos en Java*, 2a edición, CECSA, (2008).
- López R. Leobardo. *Metodología de la programación Orientada a Objetos*. Alfaomega (2006).
- Ceballos Sierra Fco. Javier. *Java - Interfaces gráficas y aplicaciones para internet.* 4a edición. RA-MA (2015).

Referencias

- Sznajdleder, Pablo. *Java a fondo – Curso de programación*, 3a edición. Alfaomega (2016).
- Pavón Mestras, Juan. *El patrón Modelo-Vista-Controlador*. Departamento de Ingeniería de Software e Inteligencia Artificial. Universidad Complutense Madrid. (2008)
- Steve Burbeck. *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*. (1992)
- *Java Tutorials. Oracle Java Documentation.* https://docs.oracle.com/javase/tutorial/uiswing/components/index.html
- Java Platform, Standard Edition 8 API Specification https://docs.oracle.com/javase/8/docs/api/

GRACIAS

C. U. UAEM ZUMPANGO / ICO / OOP / ECHL

Guía para el Profesor

- Las primeras diapositivas muestran el propósito, justificación y objetivos de la unidad de aprendizaje. Se presentan para que el alumno identifique dichos elementos.

- El contenido, conforme a la unidad de aprendizaje, maneja los temas de un menor a mayor grado de dificultad.

- Se recomienda realizar un investigación sobre otros patrones de diseño, así como pedir una investigación previa de en que consiste el MVC.

- Realizar ejercicios sencillos en los que se involucren uno o dos componentes y varias características de los mismos, para posteriormente ir incrementando el número de componentes y propiedades usadas. Se presentan algunas tablas con la intensión de que el alumno busque el funcionamiento de algunos métodos y se comenten en clase. Se sugiere continuamente consultar documentación de java (API's java SE 8), tutorial Oracle. Los códigos –propios- muestran implementaciones sencillas, que se deben centrar en el tema en cuestión.

Guía para el Profesor

- En esta unidad de aprendizaje se considera conveniente crear GUI sencillas pero desarrollando todo el código. Posteriormente, para complementar crear una GUI usando un IDE.

- Idealmente se desea que el alumno genere la GUI, sin uso de un IDE; creando ejemplos en los que se puedan repasar los conceptos anteriores como arreglos de objetos, generación de múltiples componentes, colecciones, etc.; con el propósito de que al usar un IDE pueda identificar sin problemas todo el código generado automáticamente. Los ejercicios se enfocaran en GUI sencillas pero que muestren la ventaja de identificar estos componentes, por ejemplo: generar 100 botones que respondan a un evento (un juego de batalla naval, buscaminas, memorama, etc.) situación mas laboriosa al crearse por arrastrar y soltar un botón.

- Investigaciones o exposiciones sobre 2 o 3 componentes por grupos pueden ayudar a la compresión de los mismos. Sugiriendo que se presenten 3 códigos ejemplos: creación y uso básico del componente, dificultad media y ejercicio práctico.

- Un proyecto final puede ser la creación de un juego simple o la implementación de algún algoritmo de clasificación, reconocimiento de patrones o IA.