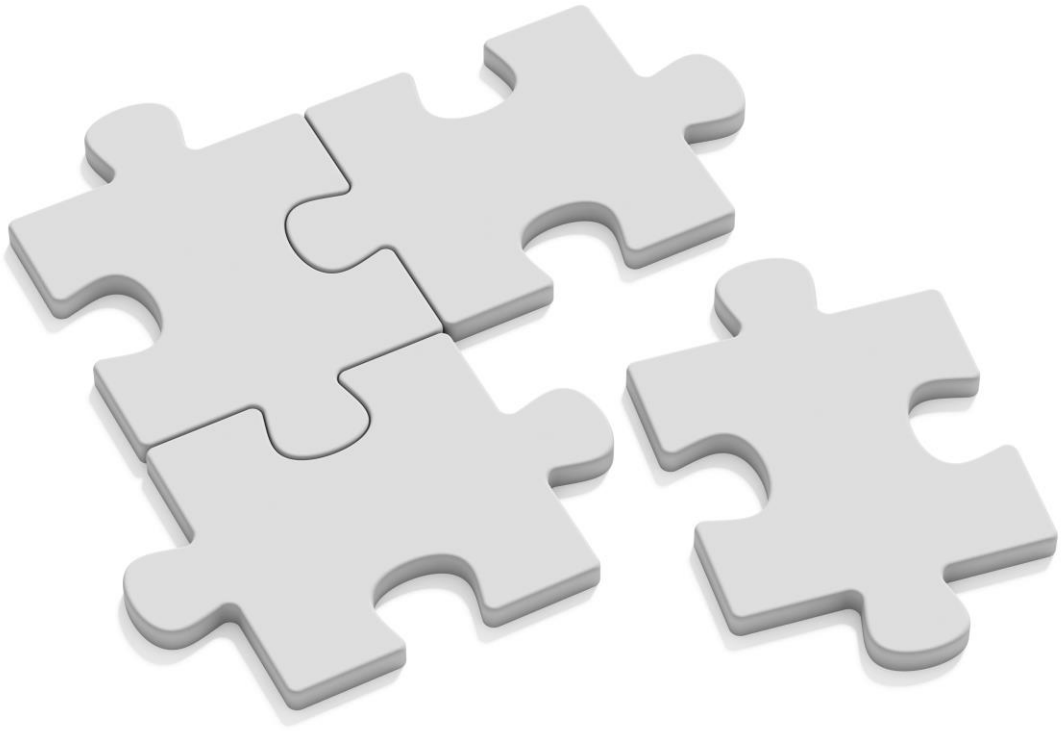
Universidad Tecnológica Nacional



2017

CÁTEDRA DE LENGUAJE DE PROGRAMACIÓN JAVA

Ings. Mario Bressano & Miguel Iwanow

ENVÍO 07/2017

EJERCICIO Nº 58

En este ejercicio se aprecia cómo se obtiene las coordenadas del mouse cuando se acciona y que botón lo hace.

```
// FuncionRaton.java
// Muestra el click del mouse en coordenadas y distingue entre los botones del mismo

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

public class FuncionRaton extends JFrame {
   private int xPos, yPos;

   // Configuramos la barra de estado, registramos la escucha del mouse ; Tamaño y mostrar windows

   public FuncionRaton()
   {
      super( "Click de mouse y botones" );

      addMouseListener( new MouseClickHandler() );

      setSize( 350, 450 );
      setVisible( true );
   }

   // Dibujamos la cadena de caracteres y ubicamos las coordenadas de donde se cliquea el mouse
   public void paint( Graphics g )
   {
      // Llamamos a la superclase del método Paint
      super.paint( g );

      g.drawString( "Cliquea en la posición [" + xPos + ", " + yPos + "]",
         xPos, yPos );
   }

   public static void main( String args[] )
   {

      FuncionRaton application = new FuncionRaton();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }

   // Clase interna manejadora de los eventos del mouse
   private class MouseClickHandler extends MouseAdapter {

      // Manejador del evento click del mouse, se establece con que botón se Cliquea
      public void mouseClicked( MouseEvent event )
```


```
    {
      xPos = event.getX();
      yPos = event.getY();

      String title = "Se cliquea " + event.getClickCount() + " vez(ces) con el botón";

      if ( event.isMetaDown() )  // botón derecho
        title += " derecho";

      else if ( event.isAltDown() )  // botón del centro
        title += " del centro ";

      else  // botón izquierdo
        title += " izquierdo ";

      setTitle( title );  // se configura lo que se va a visualizar en la barra de títulos
      repaint();

    }
  }
}
```

EJERCICIO Nº 59

En este ejercicio se aprecia como al presionar teclas, las mismas modifican la ejecución de la aplicación.

```
// DemoTecla.java
//Funcionamiento de eventos producidos por teclas.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class DemoTecla extends JFrame implements KeyListener {
  private String line1 = "", line2 = "", line3 = "";
  private JTextArea textArea;

public static void main( String args[] )
  {
     DemoTecla application = new DemoTecla();
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }
  // Configuración GUI

        public DemoTecla()
  {
    super( " Funcionamiento de eventos producidos por teclas " );
```


```
    // Configurar JTextArea
    textArea = new JTextArea( 10, 15 );
    textArea.setText( "Presionar alguna tecla del teclado.." );
    textArea.setEnabled( false );
    textArea.setDisabledTextColor( Color.BLACK );
    getContentPane().add( textArea );

    addKeyListener( this );  // En el marco se tratan los eventos de las teclas

    setSize( 350, 100 );
    setVisible( true );

  }

  // Manejador de presionar una tecla...
  public void keyPressed( KeyEvent event )
  {
    line1 = "Tecla presionada..: " + event.getKeyText( event.getKeyCode() );
    setLines2and3( event );
  }

  // Manejador de dejar de presionar una tecla
  public void keyReleased( KeyEvent event )
  {
    line1 = "Tecla dejada de presionar..: " + event.getKeyText( event.getKeyCode() );
    setLines2and3( event );
  }

  // Manejador de tecla presionada y su acción
  public void keyTyped( KeyEvent event )
  {
    line1 = "Tecla tipeada : " + event.getKeyChar();
    setLines2and3( event );
  }

  // configurar segunda y tercera línea de salida
  private void setLines2and3( KeyEvent event )
  {
    line2 = "Esta tecla es " + ( event.isActionKey() ? "" : "no " ) +
      "una tecla de acción";

    String temp = event.getKeyModifiersText( event.getModifiers() );

    line3 = "Modifican las teclas presionadas: " +
      ( temp.equals( "" ) ? "no" : temp );

    textArea.setText( line1 + "\n" + line2 + "\n" + line3 + "\n" );
  }


}
```

EJERCICIO Nº 60

En este ejercicio se aprecia cómo se usa FlowLayout

```
// FlowLayoutDemo.java

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

public class FlowLayoutDemo extends JFrame {
   private JButton BotonIzquierdo, BotonCentral, BotonDerecho;
   private Container container;
   private FlowLayout layout;

   public static void main( String args[] )
   {
      FlowLayoutDemo application = new FlowLayoutDemo();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }

// Configurar GUI y registrar escucha de botones
   public FlowLayoutDemo()
   {
      super( "Funcionamiento de FlowLayout" );

      layout = new FlowLayout();

      // Tomar un  content pane y configurar el  layout
      container = getContentPane();
      container.setLayout( layout );

      // Configurar BotonIzquierdo y registrar la escucha
      BotonIzquierdo = new JButton( "Izquierdo" );
      container.add( BotonIzquierdo );
      BotonIzquierdo.addActionListener(

         new ActionListener() {  // Clase interna anónima

           // Procesar evento BotonIzquierdo
           public void actionPerformed( ActionEvent event )
           {
              layout.setAlignment( FlowLayout.LEFT );

             // Realineación de components atachados
              layout.layoutContainer( container );
           }

         }
```


```
    );

    // Configurar BotonCentral y registrar la escucha
    BotonCentral = new JButton( "Centro" );
    container.add( BotonCentral );
    BotonCentral.addActionListener(

      new ActionListener() {  // Clase interna anónima

        // Procesar evento BotonCentral
        public void actionPerformed( ActionEvent event )
        {
          layout.setAlignment( FlowLayout.CENTER );

          // Realineación de components atachados
          layout.layoutContainer( container );
        }
      }
    );

    // Configurar BotonDerecho y registrar la escucha
    BotonDerecho = new JButton( "Derecho" );
    container.add( BotonDerecho );
    BotonDerecho.addActionListener(

      new ActionListener() {  // Clase interna anónima

        // Procesar evento BotonDerecho
        public void actionPerformed( ActionEvent event )
        {
          layout.setAlignment( FlowLayout.RIGHT );

          // Realineación de components atachados
          layout.layoutContainer( container );
        }
      }
    );

    setSize( 300, 75 );
    setVisible( true );

  } // fin constructor FlowLayoutDemo


}
```

EJERCICIO Nº 61

En este ejercicio se aprecia cómo se usa BorderLayout

Importante: Cuando se añade un componente a un contenedor que usa BorderLayout, se especifica la localización específica del componente como uno de los argumentos del método add. No esperes que un componente sea añadido al centro, por defecto. Si encontramos que un componente ha desaparecido de un contenedor controlador por un BorderLayout, debemos asegurarnos de que hemos especificado la localización del componente y de que no hemos puesto otro componente en la misma localización.

Todos nuestros ejemplos que usan BorderLayout especifican el componente como el primer argumento del método add. Por ejemplo.

add(component, BorderLayout.CENTER)

Sin embargo, podríamos ver el código de otros programas que especifican el componente en segundo lugar. Por ejemplo, esto sería una alternativa al código anterior.

add(BorderLayout.CENTER, component) //valido pero pasado de moda

o

add("Center", component) //valido pero propenso a errores

Por defecto, un BorderLayout no pone espacios entre los componentes que maneja. En el applet anterior, cualquier espacio aparente es el resultado del espacio extra que reserva JButton alrededor de su área. Podemos especificar los bordes (en píxeles) usando el siguiente constructor.

public BorderLayout(int horizontalGap, int verticalGap)

// BorderLayoutDemo.java

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

public class BorderLayoutDemo extends JFrame implements ActionListener {
  private JButton buttons[];
  private final String names[] = { "Esconder Norte", "Esconder Sur",
    "Esconder Este", "Esconder Oeste", "Esconder Centro" };
  private BorderLayout layout;

  public static void main( String args[] )
  {
    BorderLayoutDemo application = new BorderLayoutDemo();
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }

  // Configurar GUI y manejador de eventos
  public BorderLayoutDemo()
  {
```


```
    super( "Funcionamiento de BorderLayout" );

    layout = new BorderLayout( 5, 5 ); // espacios de 5 pixeles

    // Definir el panel contenedor  y asignarle el layout
    Container container = getContentPane();
    container.setLayout( layout );

    // Instanciar los objetos botones
    buttons = new JButton[ names.length ];

    for ( int count = 0; count < names.length; count++ ) {
       buttons[ count ] = new JButton( names[ count ] );
       buttons[ count ].addActionListener( this );
    }

    // Colocar botones en el BorderLayout; el orden no es importante
    container.add( buttons[ 0 ], BorderLayout.NORTH );
    container.add( buttons[ 1 ], BorderLayout.SOUTH );
    container.add( buttons[ 2 ], BorderLayout.EAST );
    container.add( buttons[ 3 ], BorderLayout.WEST );
    container.add( buttons[ 4 ], BorderLayout.CENTER );

    setSize( 300, 200 );
    setVisible( true );

  } // fin del constructor de BorderLayoutDemo

  // Manejador de eventos de botones
  public void actionPerformed( ActionEvent event )
  {
    for ( int count = 0; count < buttons.length; count++ )

       if ( event.getSource() == buttons[ count ] )
          buttons[ count ].setVisible( false );
       else
          buttons[ count ].setVisible( true );

    // Se reconfigura el layout del panel contenedor
    layout.layoutContainer( getContentPane() );
  }


}
```

EJERCICIO Nº 62

En este ejercicio se aprecia cómo se usa GridLayout

Importante: Un GridLayout sitúa los componentes en una parrilla de celdas. Cada componente toma todo el espacio disponible dentro de su celda, y cada celda es exactamente del mismo tamaño. Si redimensionamos la ventana GridLayout, veremos que el GridLayout cambia el tamaño de celda para todas sean lo más grande posible, dando el espacio disponible al contenedor.

Abajo tenemos el código quecrea el GridLayout y los componentes que maneja. (Aquí está el programa completo.

El programa corre como un applet, con la ayuda de AppletButton, o como una aplicación.)

```
Container contentPane = getContentPane();
contentPane.setLayout(new GridLayout(0,2));
contentPane.add(new JButton("Button 1"));
contentPane.add(new JButton("2"));
contentPane.add(new JButton("Button 3"));
contentPane.add(new JButton("Long-Named Button 4"));
contentPane.add(new JButton("Button 5"));
```

El constructor le dice a la clase GridLayout que cree un ejemplar que tenga dos columnas y tantas filas como sean necesarias. Es uno de los dos constructores de GridLayout. Aquí podemos ver los dos juntos.

```
public GridLayout(int rows, int columns)
public GridLayout(int rows, int columns,
                  int horizontalGap, int verticalGap)
```

Al menos uno de los argumentos rows y columns deben ser distintos de cero. Los argumentos horizontalGap y verticalGap del segundo constructor permiten especificar el número de píxeles entre celdas.

// GridLayoutDemo.java

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

public class GridLayoutDemo extends JFrame implements ActionListener {
  private JButton buttons[];
  private final String names[] =
    { "Uno", "Dos", "Tres", "Cuatro", "Cinco", "Seis" };
  private boolean toggle = true;
  private Container container;
  private GridLayout grid1, grid2;
```


```
public static void main( String args[] )
  {
    GridLayoutDemo application = new GridLayoutDemo();
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }

  //Configurar GUI
  public GridLayoutDemo()
  {
    super( "Funcionamiento GridLayout" );

    // Configurar Layout
    grid1 = new GridLayout( 2, 3, 5, 5 );//2 filas, 3 columnas y espaciado de 5 píxeles
    grid2 = new GridLayout( 3, 2 );

  // Definir el panel contenedor  y asignarle el layout
    container = getContentPane();
    container.setLayout( grid1 );

    // Crear y adicionar botones
    buttons = new JButton[ names.length ];

    for ( int count = 0; count < names.length; count++ ) {
      buttons[ count ] = new JButton( names[ count ] );
      buttons[ count ].addActionListener( this );
      container.add( buttons[ count ] );
    }

    setSize( 300, 150 );
    setVisible( true );

  } //Fin del constructor GridLayoutDemo

  // Manejador de eventos  de botones entre layouts
  public void actionPerformed( ActionEvent event )
  {
    if ( toggle )
      container.setLayout( grid2 );
    else
      container.setLayout( grid1 );

    toggle = !toggle;   // Configurar con el valor opuesto
    container.validate();
  }


}
```

EJERCICIO Nº 63

En este ejercicio se aprecia como se usa un panel con botones (barra de botones) con PanelDemo

```
//PanelDemo.java
//Usando un JPanel para ayudar a capas y sus componentes.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PanelDemo extends JFrame {
   private JPanel buttonPanel;
   private JButton buttons[];

   // Configurar GUI
   public PanelDemo()
   {
      super( "Ejemplo de Panel" );

      // Definimos el contenedor content pane
      Container container = getContentPane();

      // Creamos matriz de botones
      buttons = new JButton[ 5 ];

      // Configuramos el  panel y su  layout
      buttonPanel = new JPanel();
      buttonPanel.setLayout( new GridLayout( 1, buttons.length ) );

      // Creamos y adicionamos
      for ( int count = 0; count < buttons.length; count++ ) {
         buttons[ count ] = new JButton( "Button " + ( count + 1 ) );
         buttonPanel.add( buttons[ count ] );
      }

      container.add( buttonPanel, BorderLayout.SOUTH );

      setSize( 425, 150 );
      setVisible( true );

   } // fin del constructor PanelDemo

   public static void main( String args[] )
   {
      PanelDemo application = new PanelDemo();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }

}
```

EJERCICIO Nº 64

En este ejercicio se aprecia cómo se usa TextArea usando la copia de seleccionado de un área a otra.

```
// TextAreaDemo.java

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

public class TextAreaDemo extends JFrame {
  private JTextArea textArea1, textArea2;
  private JButton copyButton;

  // Configurar GUI
  public TextAreaDemo()
  {
    super( "Muestra TextArea" );

    Box box = Box.createHorizontalBox();

    String string = "Esto es una muestra de como\n" +
      "copiar texto\nde un area a \n" +
      "otra area usando un\nevento externo\n";

    //Configurar textArea1
    textArea1 = new JTextArea( string, 10, 15 );
    box.add( new JScrollPane( textArea1 ) );

    // Configurar el Botón de copiado
    copyButton = new JButton( "Copiar ->>" );
    box.add( copyButton );
    copyButton.addActionListener(
```


```
        new ActionListener() {  // clase interna anónima

          // Configurar la  textArea2 para seleccionar el texto desde la textArea1
          public void actionPerformed( ActionEvent event )
          {
            textArea2.setText( textArea1.getSelectedText() );
          }

        } // fin de la clase anónima

      ); // fin de la llamada addActionListener

      // Configurar textArea2
      textArea2 = new JTextArea( 10, 15 );
      textArea2.setEditable( false );
      box.add( new JScrollPane( textArea2 ) );

      // Adicionar una caja en el panel contenedor
      Container container = getContentPane();
      container.add( box );   // colocar en  BorderLayout.CENTER

      setSize( 425, 200 );
      setVisible( true );

    } // fin del constructor TextAreaDemo

    public static void main( String args[] )
    {
      TextAreaDemo application = new TextAreaDemo();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    }

}
```

EJERCICIO Nº 65

En este ejercicio se aprecia cómo se puede confeccionar un Menú.

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

public class MenuTest extends JFrame {
   private final Color colorValues[] =
      { Color.black, Color.blue, Color.red, Color.green };
   private JRadioButtonMenuItem colorItems[], fonts[];
   private JCheckBoxMenuItem styleItems[];
   private JLabel displayLabel;
   private ButtonGroup fontGroup, colorGroup;
   private int style;

   // Configurar GUI
   public MenuTest()
   {
      super( "Usando JMenus" );

      // Configurar el File menu y sus ítems
      JMenu fileMenu = new JMenu( "Archivo" );
      fileMenu.setMnemonic( 'A' );

      // Configurar el Acerca de... (ítem del menú)
      JMenuItem aboutItem = new JMenuItem( "Acerca de..." );
      aboutItem.setMnemonic( 'c' );
      fileMenu.add( aboutItem );
      aboutItem.addActionListener(

         new ActionListener() {  //clase interna anónima
```


```
        // Visualización del mensaje de diálogo que aparece cuando se selecciona
Acerca de..
        public void actionPerformed( ActionEvent event )
        {
          JOptionPane.showMessageDialog( MenuTest.this,
            "Este es un ejemplo\ndel uso de menu",
            "Acerca de..", JOptionPane.PLAIN_MESSAGE );
        }

      } //fin de la clase anónima

    ); // fin de la llamada a addActionListener

    // Confirar el menu Salida
    JMenuItem exitItem = new JMenuItem( "Salir" );
    exitItem.setMnemonic( 'S' );
    fileMenu.add( exitItem );
    exitItem.addActionListener(

      new ActionListener() {  // clase interna anónima

        // termina aplicación cuando se usa el ítem Salir
        public void actionPerformed( ActionEvent event )
        {
          System.exit( 0 );
        }

      } // fin de la clase anónima

    ); //fin de la llamada a addActionListener

    // Crear una barra de menú y adjuntarla en la ventana del MenuTest
    JMenuBar bar = new JMenuBar();
    setJMenuBar( bar );
    bar.add( fileMenu );
```


```
// Crear menú Format y los ítems del sumenú
JMenu formatMenu = new JMenu( "Formato" );
formatMenu.setMnemonic( 'F' );

// Crear submenú Color
String colors[] = { "Negro", "Azul", "Rojo", "Verde" };

JMenu colorMenu = new JMenu( "Colores" );
colorMenu.setMnemonic( 'C' );

colorItems = new JRadioButtonMenuItem[ colors.length ];
colorGroup = new ButtonGroup();
ItemHandler itemHandler = new ItemHandler();

// Crear los radio button de color para los items de menú
for ( int count = 0; count < colors.length; count++ ) {
   colorItems[ count ] =
      new JRadioButtonMenuItem( colors[ count ] );
   colorMenu.add( colorItems[ count ] );
   colorGroup.add( colorItems[ count ] );
   colorItems[ count ].addActionListener( itemHandler );
}

// Seleccionar el primer ítem del menú Color
colorItems[ 0 ].setSelected( true );

// Adicionar el menú format a la barra de menú
formatMenu.add( colorMenu );
formatMenu.addSeparator();

// Crear el submenú Fuentes
String fontNames[] = { "Serif", "Espacio Simple", "SansSerif" };

JMenu fontMenu = new JMenu( "Fuente" );
fontMenu.setMnemonic( 'n' );
```


```
fonts = new JRadioButtonMenuItem[ fontNames.length ];
fontGroup = new ButtonGroup();

// Crear los botones de radio de  Font radio
for ( int count = 0; count < fonts.length; count++ ) {
   fonts[ count ] = new JRadioButtonMenuItem( fontNames[ count ] );
   fontMenu.add( fonts[ count ] );
   fontGroup.add( fonts[ count ] );
   fonts[ count ].addActionListener( itemHandler );
}

// Seleccionar el primer ítem del menú Font
fonts[ 0 ].setSelected( true );

fontMenu.addSeparator();

// Configurar ítem Estilo
String styleNames[] = { "Negrita", "Cursiva" };

styleItems = new JCheckBoxMenuItem[ styleNames.length ];
StyleHandler styleHandler = new StyleHandler();

// Crear checkbox ítems de estilo
for ( int count = 0; count < styleNames.length; count++ ) {
   styleItems[ count ] =
      new JCheckBoxMenuItem( styleNames[ count ] );
   fontMenu.add( styleItems[ count ] );
   styleItems[ count ].addItemListener( styleHandler );
}

// Adicionar el menú Font menú en el menú Format
formatMenu.add( fontMenu );

// Adicionar el menu Format en la barra de menú
bar.add( formatMenu );
```


```
   // Configurar la etiqueta para mostrar texto
   displayLabel = new JLabel( "Texto de Ejemplo", SwingConstants.CENTER );
   displayLabel.setForeground( colorValues[ 0 ] );
   displayLabel.setFont( new Font( "Serif", Font.PLAIN, 72 ) );

   getContentPane().setBackground( Color.CYAN );
   getContentPane().add( displayLabel, BorderLayout.CENTER );

   setSize( 500, 200 );
   setVisible( true );

} // fin del constructor

public static void main( String args[] )
{
   MenuTest application = new MenuTest();
   application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
}

// clase interna para manejar los eventos de los ítems del menú
private class ItemHandler implements ActionListener {

   // procesar colores en las selecciones de la fuente
   public void actionPerformed( ActionEvent event )
   {

      for ( int count = 0; count < colorItems.length; count++ )

         if ( colorItems[ count ].isSelected() ) {
            displayLabel.setForeground( colorValues[ count ] );
            break;
         }

      // procesar selección de fuentes
      for ( int count = 0; count < fonts.length; count++ )
```


```
        if ( event.getSource() == fonts[ count ] ) {
          displayLabel.setFont(
            new Font( fonts[ count ].getText(), style, 72 ) );
          break;
        }

      repaint();

    } // fin del método actionPerformed

} //fin de la clase ItemHandler

// clase interna para manejar los eventos de los check box del menú
private class StyleHandler implements ItemListener {

    // procesar la selección de estilos de fuentes
    public void itemStateChanged( ItemEvent e )
    {
      style = 0;

      // chequeo de la selección de fuente negrita
      if ( styleItems[ 0 ].isSelected() )
        style += Font.BOLD;

      //chequeo de la selección en cursiva
      if ( styleItems[ 1 ].isSelected() )
        style += Font.ITALIC;

      displayLabel.setFont(
        new Font( displayLabel.getFont().getName(), style, 72 ) );

      repaint();
    }

} // fin de la clase StyleHandler
```


```
}
```

EJERCICIO Nº 66

En este ejercicio se aprecia cómo se puede manejar el Look and Feel. **Existen manejos de errores que se explicarán en próximas clases.**

```
// LookAndFeelDemo.java

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

public class LookAndFeelDemo extends JFrame {
   private final String strings[] = { "Metal", "Motif", "Windows" };
   private UIManager.LookAndFeelInfo looks[];
   private JRadioButton radio[];
   private ButtonGroup group;
   private JButton button;
   private JLabel label;
   private JComboBox comboBox;

   // Configurar GUI
   public LookAndFeelDemo()
   {
      super( "Demo del Look and Feel" );

      Container container = getContentPane();

      // Configurar el panel norte del BorderLayout
      JPanel northPanel = new JPanel();
      northPanel.setLayout( new GridLayout( 3, 1, 0, 5 ) );

      // Configurar la etiqueta del panel norte
```


```
label = new JLabel( "Este es el  look-and-feel",
   SwingConstants.CENTER );
northPanel.add( label );

// Configurar el botón para el panel norte
button = new JButton( "Botón" );
northPanel.add( button );

// configurar el combo box para el panel norte
comboBox = new JComboBox( strings );
northPanel.add( comboBox );

// Crear una matriz para los botones de radio
radio = new JRadioButton[ strings.length ];

// Configurar el  panel sur del  BorderLayout
JPanel southPanel = new JPanel();
southPanel.setLayout( new GridLayout( 1, radio.length ) );

// configurar los botones del panel sur
group = new ButtonGroup();
ItemHandler handler = new ItemHandler();

for ( int count = 0; count < radio.length; count++ ) {
   radio[ count ] = new JRadioButton( strings[ count ] );
   radio[ count ].addItemListener( handler );
   group.add( radio[ count ] );
   southPanel.add( radio[ count ] );
}

// adicionar NORTE Y SUR a  content pane
container.add( northPanel, BorderLayout.NORTH );
container.add( southPanel, BorderLayout.SOUTH );

// Tomar la información instalada del look-and-feel
looks = UIManager.getInstalledLookAndFeels();
```


```
   setSize( 300, 200 );
   setVisible( true );

   radio[ 0 ].setSelected( true );

} // fin del constructor LookAndFeelDemo

// Usar UIManager para cambiar el look-and-feel del GUI
private void changeTheLookAndFeel( int value )
{
   // cambiar el look and feel
   try {
      UIManager.setLookAndFeel( looks[ value ].getClassName() );
      SwingUtilities.updateComponentTreeUI( this );
   }

   // Procesar si hay problemas en los cambios del look and feel
   catch ( Exception exception ) {
      exception.printStackTrace();
   }
}

public static void main( String args[] )
{
   LookAndFeelDemo application = new LookAndFeelDemo();
   application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
}

// clase interna para el manejo de eventos de los  radio button
private class ItemHandler implements ItemListener {

   // procesos del uso en la selección del look-and-feel
   public void itemStateChanged( ItemEvent event )
   {
      for ( int count = 0; count < radio.length; count++ )
```


```
            if ( radio[ count ].isSelected() ) {
               label.setText( "Esto es un  " +
                  strings[ count ] + " look-and-feel" );
               comboBox.setSelectedIndex( count );
               changeTheLookAndFeel( count );
            }
         }

      } // fin de la clase interna ItemHandler

}
```

EJERCICIO Nº 67

En este ejercicio se aprecia cómo se puede manejar el JDesktopPane

```
// DesktopTest.java
// Muestra de  JDesktopPane.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class DesktopTest extends JFrame {
   private JDesktopPane theDesktop;

   // Configurar GUI
   public DesktopTest()
   {
      super( "Usando un  JDesktopPane" );

      // Crear barra de menú y los ítems  del menú
      JMenuBar bar = new JMenuBar();
      JMenu addMenu = new JMenu( "Agregar" );
      JMenuItem newFrame = new JMenuItem( "Marco Interno" );
```


```
addMenu.add( newFrame );
bar.add( addMenu );

setJMenuBar( bar );

// Configurar desktop
theDesktop = new JDesktopPane();
getContentPane().add( theDesktop );

// Configurar listener para el item de menú newFrame
newFrame.addActionListener(

   new ActionListener() {  // clase anónima interna

      // Mostrar la nueva ventana interna
      public void actionPerformed( ActionEvent event ) {

         // Crear en marco interno
         JInternalFrame frame = new JInternalFrame(
            "Marco Interno", true, true, true, true );

         //adjuntar el marco interno en el panel contenedor
         Container container = frame.getContentPane();
         MyJPanel panel = new MyJPanel();
         container.add( panel, BorderLayout.CENTER );

         // configurar tamaño del marco interno y de sus contenidos
         frame.pack();

         // adjuntar el marco interno al desktop y mostrarlo
         theDesktop.add( frame );
         frame.setVisible( true );
      }

   } // fin de clase anónima
```


```
      ); // fin de la llamada a addActionListener

      setSize( 600, 460 );
      setVisible( true );

   }

   public static void main( String args[] )
   {
      DesktopTest application = new DesktopTest();
      application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
   }

} // fin clase DesktopTest

// clase para mostrar un ícono de imagen en un panel
class MyJPanel extends JPanel {
   private ImageIcon imageIcon;
   private String[] images = { "yellowflowers.png", "purpleflowers.png",
      "redflowers.png", "redflowers2.png", "lavenderflowers.png" };

   // cargar imagen
   public MyJPanel()
   {
      int randomNumber = ( int ) ( Math.random() * 5 );
      imageIcon = new ImageIcon( images[ randomNumber ] );
   }

   // mostrar el ícono de imagen en el panel

 public void paintComponent( Graphics g )
   {
      // llamada al método de la superclase  paintComponent
      super.paintComponent( g );
```


```
    // mostrar ícono
    imageIcon.paintIcon( this, g, 0, 0 );
  }

  // retomar las dimensiones de la imagen
  public Dimension getPreferredSize()
  {
    return new Dimension( imageIcon.getIconWidth(),
      imageIcon.getIconHeight() );
  }

} //fin de claseMyJPanel
```

EJERCICIO Nº 68

En este ejercicio se aprecia cómo se pueden usar solapas

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

public class JTabbedPaneDemo extends JFrame  {

  // configurar GUI
  public JTabbedPaneDemo()
  {
    super( "Muestra de funcionamiento de JTabbedPane" );

    // Crear JTabbedPane
    JTabbedPane tabbedPane = new JTabbedPane();

    // Configurar panel1 y adicionar dentro de JTabbedPane
```


```
    JLabel label1 = new JLabel( "Panel uno", SwingConstants.CENTER );
    JPanel panel1 = new JPanel();
    panel1.add( label1 );
    tabbedPane.addTab( "Tab Uno", null, panel1, "Primer Panel" );

    // Configurar panel2 y adicionar dentro de JTabbedPane
    JLabel label2 = new JLabel( "Panel dos", SwingConstants.CENTER );
    JPanel panel2 = new JPanel();
    panel2.setBackground( Color.YELLOW );
    panel2.add( label2 );
    tabbedPane.addTab( "Tab Dos", null, panel2, "Segundo Panel" );

    // Configurar panel3 y adicionar dentro de JTabbedPane
    JLabel label3 = new JLabel( "Panel tres" );
    JPanel panel3 = new JPanel();
    panel3.setLayout( new BorderLayout() );
    panel3.add( new JButton( "Norte" ), BorderLayout.NORTH );
    panel3.add( new JButton( "Oeste" ), BorderLayout.WEST );
    panel3.add( new JButton( "Este" ), BorderLayout.EAST );
    panel3.add( new JButton( "Sur" ), BorderLayout.SOUTH );
    panel3.add( label3, BorderLayout.CENTER );
    tabbedPane.addTab( "Tab tres", null, panel3, "Tercer Panel" );

    // Adicionar JTabbedPane n el contenedor
    getContentPane().add( tabbedPane );

    setSize( 450, 400 );
    setVisible( true );

}

public static void main( String args[] )
{
    JTabbedPaneDemo tabbedPaneDemo = new JTabbedPaneDemo();
    tabbedPaneDemo.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
}
```

} **// fin clase JTabbedPaneDemo**

EJERCICIO Nº 69

En este ejercicio se aprecia cómo se usa Box Layout

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

public class BoxLayoutDemo extends JFrame {

  // Configurar GUI
  public BoxLayoutDemo()
  {
    super( "Funcionamiento de BoxLayout" );

    // Crear una caja contenedora con BoxLayout
    Box horizontal1 = Box.createHorizontalBox();
    Box vertical1 = Box.createVerticalBox();
    Box horizontal2 = Box.createHorizontalBox();
    Box vertical2 = Box.createVerticalBox();

    final int SIZE = 3; // número de botones en el  Box

    // adicionar botones en el  Box horizontal1
    for ( int count = 0; count < SIZE; count++ )
      horizontal1.add( new JButton( "Botón " + count ) );

    // Adicionar botones en box vertical1
    for ( int count = 0; count < SIZE; count++ ) {
      vertical1.add( Box.createVerticalStrut( 25 ) );
```


```
   vertical1.add( new JButton( " Botón " + count ) );
}

// Crear horizontal glue y adicionar botones en el Box horizontal2
for ( int count = 0; count < SIZE; count++ ) {
   horizontal2.add( Box.createHorizontalGlue() );
   horizontal2.add( new JButton( " Botón " + count ) );
}

// crear un área rígida y adicionar botones en el Box vertical2
for ( int count = 0; count < SIZE; count++ ) {
   vertical2.add( Box.createRigidArea( new Dimension( 12, 8 ) ) );
   vertical2.add( new JButton( " Botón " + count ) );
}

// Crear vertical glue y adicionar botones al panel
JPanel panel = new JPanel();
panel.setLayout( new BoxLayout( panel, BoxLayout.Y_AXIS ) );

for ( int count = 0; count < SIZE; count++ ) {
   panel.add( Box.createGlue() );
   panel.add( new JButton( " Botón " + count ) );
}

// Crear un JTabbedPane
JTabbedPane tabs = new JTabbedPane(
   JTabbedPane.TOP, JTabbedPane.SCROLL_TAB_LAYOUT );

// Poner el contenedor en la solapa
tabs.addTab( "Caja Horizontal", horizontal1 );
tabs.addTab( "Caja Vertical", vertical1 );
tabs.addTab( "Caja Horizontal 2", horizontal2 );
tabs.addTab( "Caja Vertical 2", vertical2 );
tabs.addTab( "Caja Vertical 3", panel );

getContentPane().add( tabs );  // Colocar la solapa en el panel contenedor
```


```
    setSize( 400, 220 );
    setVisible( true );

  }

  public static void main( String args[] )
  {
    BoxLayoutDemo application = new BoxLayoutDemo();
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }

}
```

EJERCICIO Nº 70

En este ejercicio se aprecia cómo se usa GridBag Layout

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

public class GridBagDemo extends JFrame {
  private Container container;
  private GridBagLayout layout;
  private GridBagConstraints constraints;

  // Configurar GUI
  public GridBagDemo()
  {
    super( "Uso de GridBagLayout" );
```


```
container = getContentPane();
layout = new GridBagLayout();
container.setLayout( layout );

// Instanciar gridbag
constraints = new GridBagConstraints();

// Crear components GUI
JTextArea textArea1 = new JTextArea( "Área de texto 1", 5, 10 );
JTextArea textArea2 = new JTextArea( " Área de texto 2", 2, 2 );

String names[] = { "Acero", "Hierro", "Bronce" };
JComboBox comboBox = new JComboBox( names );

JTextField textField = new JTextField( "Campo de texto" );
JButton button1 = new JButton( "Botón 1" );
JButton button2 = new JButton( " Botón 2" );
JButton button3 = new JButton( " Botón 3" );

// alto y ancho del textArea1 son 0: por defecto
// components centrados: por defecto
constraints.fill = GridBagConstraints.BOTH;
addComponent( textArea1, 0, 0, 1, 3 );

// alto y ancho del boton1 son 0: por defecto
constraints.fill = GridBagConstraints.HORIZONTAL;
addComponent( button1, 0, 1, 2, 1 );

// alto y ancho del combobox son 0: por defecto
// línea HORIZONTAL
addComponent( comboBox, 2, 1, 2, 1 );

// botón
constraints.weightx = 1000;  // puede ser más ancho
constraints.weighty = 1;     // puede ser más alto
//Botón 3
```


```
    constraints.fill = GridBagConstraints.BOTH;
    addComponent( button2, 1, 1, 1, 1 );


    constraints.weightx = 0;
    constraints.weighty = 0;
    addComponent( button3, 1, 2, 1, 1 );

        // alto y ancho del text field son 0: por defecto
    addComponent( textField, 3, 0, 2, 1 );

        // alto y ancho del textarea2 son 0: por defecto
    addComponent( textArea2, 3, 2, 1, 1 );

    setSize( 300, 150 );
    setVisible( true );

} // fin del constructor GridBagDemo

// método de configuración de constraints
private void addComponent( Component component,
    int row, int column, int width, int height )
{
    // configurar grilla
    constraints.gridx = column;
    constraints.gridy = row;

    // configurar ancho y alto de la grilla
    constraints.gridwidth = width;
    constraints.gridheight = height;

    // configurar constraints y adicionar componentes
    layout.setConstraints( component, constraints );
    container.add( component );
}
```


```
  public static void main( String args[] )
  {
    GridBagDemo application = new GridBagDemo();
    application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
  }

}
```

EJERCICIO Nº 71

En este ejercicio se aprecia nuevamente como se usa GridBag Layout

```
//GridBagDemo2.java

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

public class GridBagDemo2 extends JFrame {
  private GridBagLayout layout;
  private GridBagConstraints constraints;
  private Container container;

  // Configurar GUI
  public GridBagDemo2()
  {
    super( "GridBagLayout" );

    container = getContentPane();
    layout = new GridBagLayout();
    container.setLayout( layout );

    // Instanciar gridbag
```


```
constraints = new GridBagConstraints();

// Crear componentes GUI
String metals[] = { "Cobre", "Aluminio", "Plata" };
JComboBox comboBox = new JComboBox( metals );

JTextField textField = new JTextField( "Campo de texto" );

String fonts[] = { "Serif", "Monospaced" };
JList list = new JList( fonts );

String names[] = { "Cero", "Uno", "Dos", "Tres", "Cuatro" };
JButton buttons[] = new JButton[ names.length ];

for ( int count = 0; count < buttons.length; count++ )
   buttons[ count ] = new JButton( names[ count ] );

// Definir componentes GUI para el  textField
constraints.weightx = 1;
constraints.weighty = 1;
constraints.fill = GridBagConstraints.BOTH;
constraints.gridwidth = GridBagConstraints.REMAINDER;
addComponent( textField );

// buttons[0] – el ancho y alto es 1
constraints.gridwidth = 1;
addComponent( buttons[ 0 ] );

// buttons[1] – el ancho y alto es 1

constraints.gridwidth = GridBagConstraints.RELATIVE;
addComponent( buttons[ 1 ] );

// buttons[2] – el ancho y alto es 1

constraints.gridwidth = GridBagConstraints.REMAINDER;
```


```
    addComponent( buttons[ 2 ] );

    // comboBox – ancho es  1
    constraints.weighty = 0;
    constraints.gridwidth = GridBagConstraints.REMAINDER;
    addComponent( comboBox );

    // buttons[3] – ancho es 1
    constraints.weighty = 1;
    constraints.gridwidth = GridBagConstraints.REMAINDER;
    addComponent( buttons[ 3 ] );

    // buttons[4] – el ancho y alto es 1
    constraints.gridwidth = GridBagConstraints.RELATIVE;
    addComponent( buttons[ 4 ] );

    // lista – el ancho y alto es 1

    constraints.gridwidth = GridBagConstraints.REMAINDER;
    addComponent( list );

    setSize( 300, 200 );
    setVisible( true );

}

// adicionar el componente en el container
private void addComponent( Component component )
{
    layout.setConstraints( component, constraints );
    container.add( component );      // adicionar componentes
}

public static void main( String args[] )
{
    GridBagDemo2 application = new GridBagDemo2();
```


```
        application.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    }

}
```

EJERCICIO Nº 72 (PROPUESTO)

Componer un applet libre con JPanel para cualquier aplicación con todos los elementos vistos a este momento