Diseño y Desarrollo de Sistemas de Información

Diseño, creación y gestión de interfaces gráficas (vistas) usando la librería Swing

- Introducción
- El contenedor JFrame
- Componentes básicos
- Gestión de los eventos desde el controlador

¿Qué es Swing?

- Swing es una librería de clases que permite crear interfaces gráficas de usuario en Java
- Existen multitud de librerías para diseñar interfaces (por ejemplo, **JavaFX, AWT**, etc.). En este curso usaremos Swing por ser una librería ligera, estable y estar muy extendida en la comunidad de desarrolladores
- Además, la librería Swing está muy bien integrada con el IDE *Netbeans*
- **Esto no es un manual de Swing**. Sólo son unas nociones básicas para poder crear aplicaciones que interactúen, de forma gráfica, con una base de datos

- El número de clases que proporciona la librería Swing es enorme

- Nosotros estudiaremos una pequeña parte que nos permitirá construir interfaces gráficas de usuario muy completas

- Las interfaces (vistas) las crearemos mediante la herramienta de diseño incluida en *NetBeans*, lo que implica que buena parte del código será generado de forma automática

- Existen dos elementos básicos para la creación de interfaces gráficas de usuario usando Swing:
  - **Contenedores.** Elementos capaces de albergar otros elementos.
  - **Componentes.** Elementos que se añaden a los contenedores. También suelen denominarse **controles**

Diseño de la vista

- Para explicar los primeros conceptos de Swing vamos a crear una vista cuya función será el control del acceso a nuestra aplicación

- Deberá tener un aspecto parecido a este

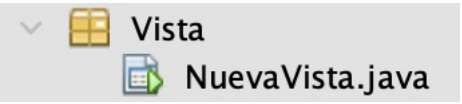
- En el paquete "Vista" seleccionamos un nuevo **"Formulario JFrame" (*JFrame Form*)**

> Los *Frames* se utilizan, generalmente, como ventanas independientes de alto nivel. La mayoría de las aplicaciones Swing se crean a partir de este tipo de contenedor

- Una vez elegido el nombre del fichero y la ubicación, aparecerá un panel "vacío" al que podremos ir añadiendo componentes desde la "Paleta de Componentes"

- En nuestro primer ejemplo, los componentes que tendrá la vista son:
  - **Etiqueta (JLabel)**. Para mostrar texto identificativo
  - **Cuadro de texto (jTextField)**. Para escribir y mostrar texto
  - **Cuadro de edición de password (jPasswordField)**. Para escribir texto "oculto"
  - **Botón (jButton)**. Componente que, al ser pulsado, lanzará eventos
  - **Lista desplegable (JComboBox)**. Para seleccionar un elemento entre varias opciones

- Al crear un nuevo *JFrame*, *NetBeans* le añade un método **main()**

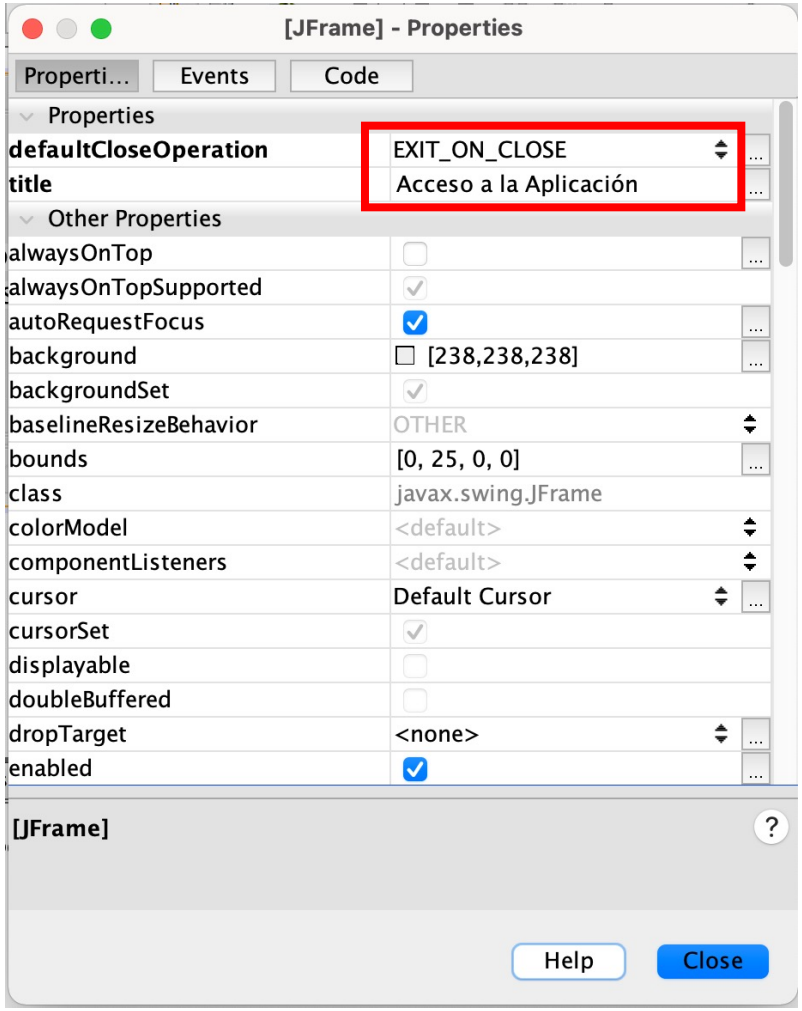


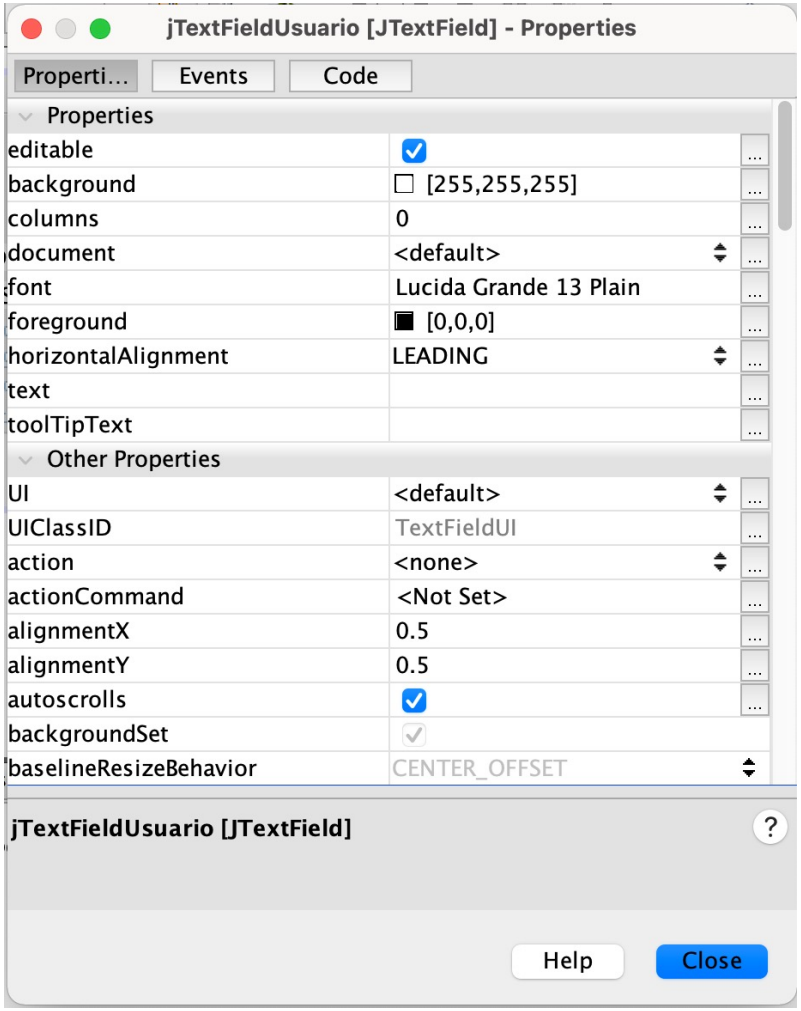

```
/**
 * @param args the command line arguments
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
     Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new NuevaVista().setVisible(true);
        }
    });
}
```

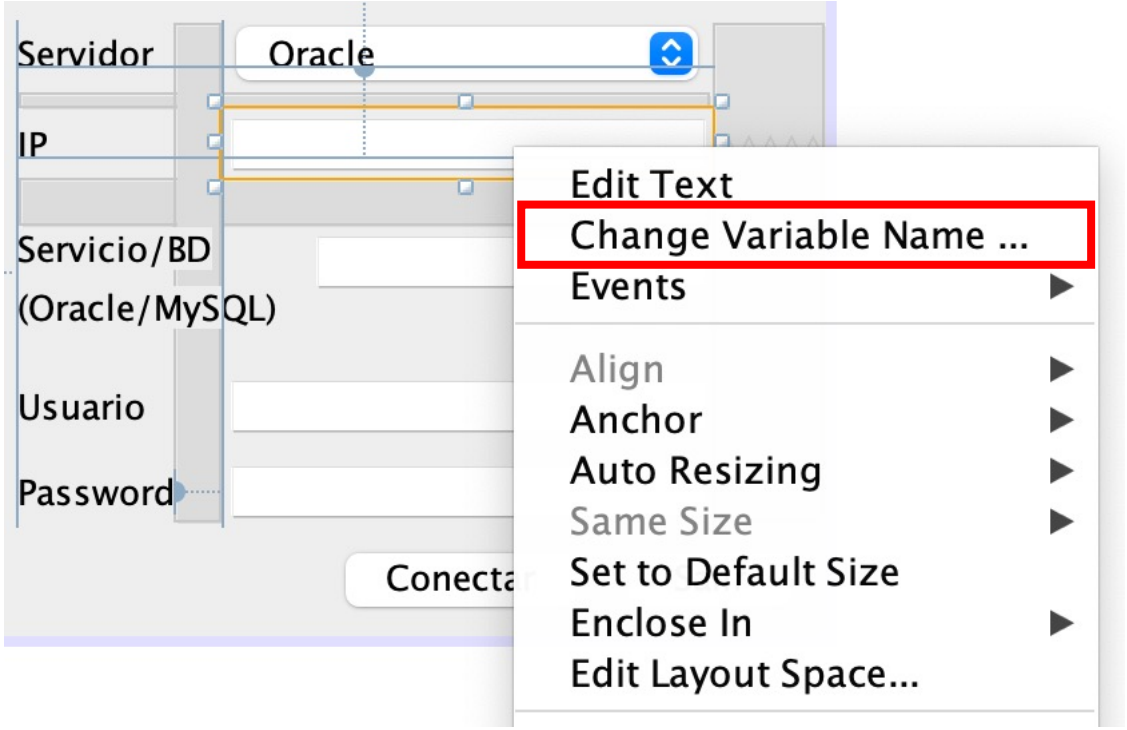
- En los proyectos que desarrollemos en este curso eliminaremos esta sección de código puesto que ya tenemos la **clase principal** en la capa de aplicación

- Una vez situados los componentes, podemos definir y modificar todas sus características y propiedades (que son muchas)

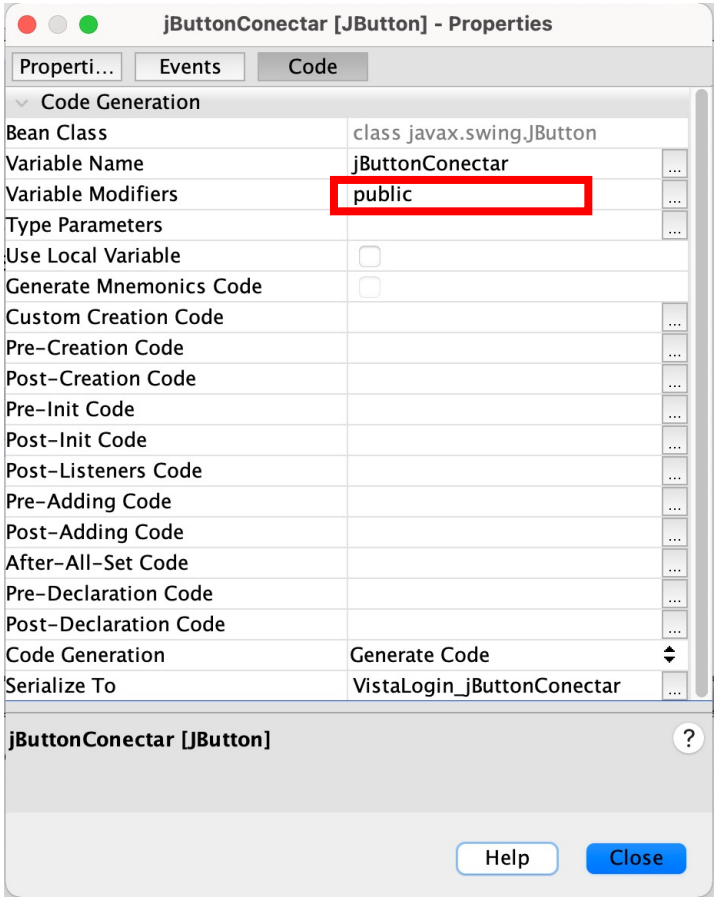




- Una de las principales acciones que debemos hacer es asignar al componente un nombre de variable "entendible" ya que luego lo tendremos que utilizar en el código de los controladores
- Se puede hacer desde el menú de propiedades o desde el menú contextual con el botón derecho



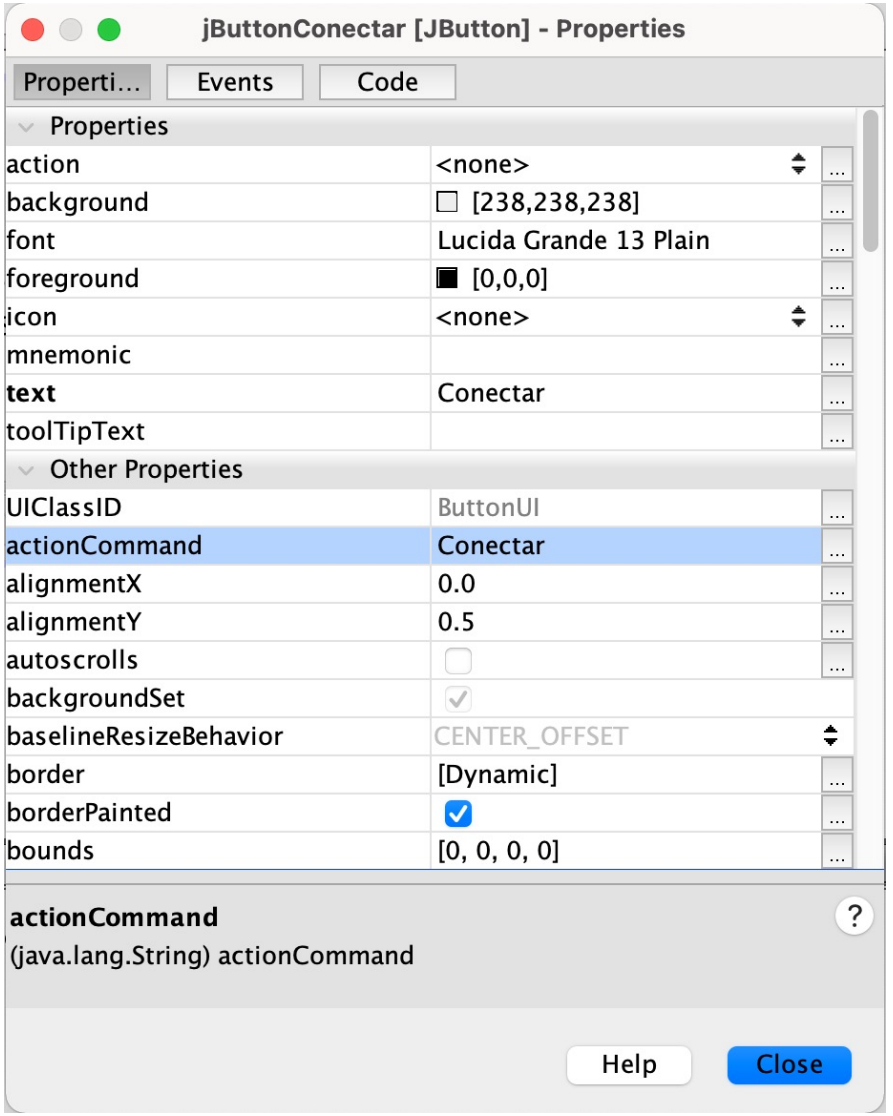
- Para que los componentes sean visibles desde el controlador, debemos modificar su acceso y hacerlos **públicos** (por defecto son privados)



- Una propiedad que también debemos modificar es **actionCommand**, especialmente en aquellos componentes que van a lanzar eventos durante la ejecución de la aplicación

- Esta propiedad se utilizará en el código del controlador para saber el componente que ha lanzado un determinado evento

- Por tanto, debemos ponerle un nombre significativo que lo identifique



- Una vez diseñada la vista, si vemos el código fuente que ha generado *NetBeans* comprobaremos que se han declarado los componentes en una sección cuyo código no se puede modificar

```
// Variables declaration - do not modify
public javax.swing.JButton btnSalirDialogoConexion;
public javax.swing.JButton jButtonConectar;
public javax.swing.JComboBox<String> jComboBoxServidores;
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JLabel jLabel4;
private javax.swing.JLabel jLabel5;
private javax.swing.JLabel jLabel6;
public javax.swing.JPasswordField jPasswordField;
public javax.swing.JTextField jTextFieldIP;
public javax.swing.JTextField jTextFieldService_BD;
public javax.swing.JTextField jTextFieldUsuario;
// End of variables declaration
```

- La otra parte del código que genera *Netbeans*, de forma automática, es la función **initComponents()**, a la que se llama desde el constructor de la vista

```
public VistaLogin() {
    initComponents();
}
```

```
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jButtonSalirDialogoConexion = new javax.swing.JButton();
    jButtonConectar = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jTextFieldUsuario = new javax.swing.JTextField();
    jPasswordField = new javax.swing.JPasswordField();
    jLabel3 = new javax.swing.JLabel();
    jComboBoxServidores = new javax.swing.JComboBox<>();
    jLabel4 = new javax.swing.JLabel();
    jTextFieldIP = new javax.swing.JTextField();
    jLabel5 = new javax.swing.JLabel();
    jTextFieldService_BD = new javax.swing.JTextField();
    jLabel6 = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
    setTitle("Acceso a la Aplicación");

    jButtonSalirDialogoConexion.setText("Salir");
    jButtonSalirDialogoConexion.setActionCommand("SalirDialogoConexion");

    jButtonConectar.setText("Conectar");
```

Gestión de eventos

- La gestión de los eventos que se producen en las vistas la realizaremos desde los controladores
- Existen varias formas de gestionar los eventos. Una de ellas consiste en indicar que la clase "Controlador" implementará las interfaces de gestión de eventos que vayamos a utilizar en nuestra aplicación
- En nuestro ejemplo solo vamos a gestionar eventos de tipo **ActionListener**,

```
public class ControladorLogin implements ActionListener {
```

- En el controlador se asignarán los ***listener*** (escuchadores) a los componentes que lanzan eventos. Suele hacerse con una función de nombre **addListeners()**, y la llamaremos desde el constructor de la clase "Controlador"

```
private void addListeners() {
    vLogin.jButtonSalirDialogoConexion.addActionListener(this);
    vLogin.jButtonConectar.addActionListener(this);
}
```

- Por último, se implementa el método **actionPerformed()**, de la interfaz **actionListener**, en el que gestionamos los eventos usando la propiedad **actionCommand** de los componentes

```
@Override
public void actionPerformed(ActionEvent e) {
    switch (e.getActionCommand()) {
        case "Conectar":
            conexionOK = conectar();
            if (conexionOK) {
                vLogin.dispose();
                // Llamadas a las funciones que se deben
                // realizar si la conexión es correcta
            }
            break;
        case "SalirDialogoConexion":
            vLogin.dispose();
            System.exit(0);
            break;

    }
}
```

Propiedad **actionCommand** del botón "Conectar"

Propiedad **actionCommand** del botón "Salir"

También se pueden gestionar los eventos usando el nombre de la variable del componente. En ese caso habría que utilizar el método **getSource()** y usar sentencias *if*

Por ejemplo: if (e.getSource() == vLogin.jButtonConectar)

- Siguiendo con el ejemplo que estamos desarrollando, el método constructor de la clase **ControladorLogin** sería algo así

```
public ControladorLogin() {
    vLogin = new VistaLogin();
    vMensaje = new VistaMensajes();

    addListeners();

    vLogin.setLocationRelativeTo(null);
    vLogin.setVisible(true);
}
```

Usaremos un objeto de tipo vMensaje para mostrar mensajes

Sitúa la ventana en el centro de la pantalla

Muestra la ventana

- Si la conexión es correcta, eliminamos la vista (ventana) de acceso a la aplicación e instanciamos un objeto de la clase **Controlador**, que será el controlador principal de la aplicación

- Por motivos de eficiencia, el constructor de la clase Controlador recibirá, como parámetro, el objeto "**conexion**" (atributo de la clase ControladorLogin encargado de realizar la conexión a la BD a través de la clase "**Conexion**" creada en el Modelo)

```
case "Conectar":
    conexionOK = conectar();
    if (conexionOK) {
        vLogin.dispose();
        Controlador controlador = new Controlador(conexion);
    }
    break;
```

- La clase Controlador tendrá una implementación similar a esta

```
public class Controlador implements ActionListener {
    private Conexion conexion = null;
    private VistaMensajes vMensaje = null;
    private VistaPrincipal vPrincipal = null;

    public Controlador(Conexion conexion) {
        this.conexion = conexion;
        vMensaje = new VistaMensajes();
        vPrincipal = new VistaPrincipal();

        addListeners();

        vPrincipal.setLocationRelativeTo(null);
        vPrincipal.setVisible(true);
    }

    private void addListeners() {
        vPrincipal.jButtonCerrarVPrincipal.addActionListener(this);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        switch (e.getActionCommand()) {
            case "CerrarVentanaPrincipal":
                vPrincipal.dispose();
                System.exit(0);
                break;
        }
    }
}
```

Se asigna al atributo propio el parámetro recibido

De momento, la vista principal solo tiene un botón que, al pulsarlo, cierra la ventana y sale de la aplicación

- Los objetos componentes tienen un gran conjunto de métodos para gestionarlos
- Por ejemplo, para capturar su valor y asignarlo a una variable, usaremos los métodos **get()**, que tendrán un nombre específico dependiendo del componente

```
String server = (String) (vLogin.jComboBoxServidores.getSelectedItem());
String ip = vLogin.jTextFieldIP.getText();
String service_bd = vLogin.jTextFieldService_BD.getText();
String u = vLogin.jTextFieldUsuario.getText();
String p = new String (vLogin.jPasswordField.getPassword());
```

- De esta forma, con los valores leídos desde la vista podemos llamar al constructor de conexión

```
conexion = new Conexion(server, ip, service_bd, u, p);
```

NOTA: hay que prestar atención a la diferencia que hay entre el contenido del ComboBox y el valor que hay que pasarle al constructor de "Conexión"