

# Interfaces Gráficas de Usuario (GUIs)

# Índice

- Los paquetes `java.awt` y `javax.swing`
- Diseño de Interfaces Gráficas de Usuario (GUIs)
  - El patrón Modelo-Vista-Controlador (MVC)
- Vistas (Parte I) (no la estudiaremos)
  - Componentes y Contenedores
  - Construcción de Vistas. Un ejemplo sencillo (I)
- Controladores
  - Relación Vista-Controlador
  - El mecanismo de eventos. Un ejemplo sencillo (II)
  - Relación Controlador-Modelo. Un ejemplo sencillo (III)
- Vistas (Parte II) (no la estudiaremos)
  - Gestores de Esquemas
  - Componentes
  - Contenedores intermedios
  - Un ejemplo más elaborado
  - Pintar en Swing

# Los paquetes `java.awt` y `javax.swing`

- Permiten la construcción de Interfaces Gráficas de Usuario (GUIs).
- Inicialmente sólo existe AWT (Abstract Window Toolkit)
- SWING se basa en (y extiende) AWT.
- Se verán las características más importantes para construir GUIs básicos.

# Los paquetes

## `java.awt` y `javax.swing`

- Por cada componente visible de AWT (botón, campo de texto, ...) existe otro en el sistema operativo que es el que realmente realiza la representación.
- Problemas 😞:
  - Hay componentes que algún sistema operativo no tiene, por lo que AWT sólo define lo que tienen en común todos.
  - Los componentes se pueden representar de forma diferente en sistemas operativos distintos, ya que pueden tener características y propiedades diferentes. Además la visualización (“look and feel”) es diferente.

# Los paquetes `java.awt` y `javax.swing`

- Swing elimina estos problemas 😊:
  - Define todos los componentes usuales en GUIs.
  - Se encarga de representar cada componente. Sus características y propiedades son comunes a cualquier sistema operativo. Aunque la visualización (“look and feel”) puede ser diferente.
- Necesita los paquetes (y subpaquetes):  
`java.awt`, `java.awt.event` y `javax.swing`

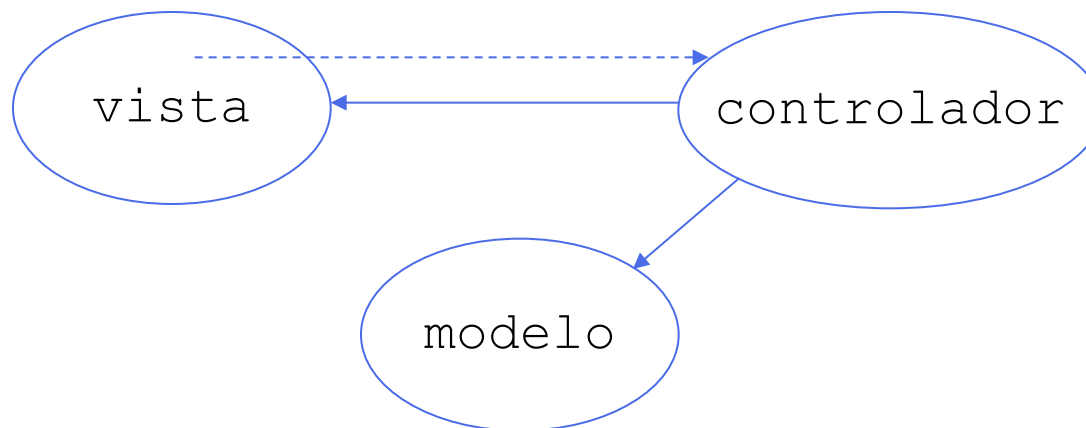
# Índice

- Los paquetes `java.awt` y `javax.swing`
- Diseño de Interfaces Gráficas de Usuario (GUIs)
  - El patrón Modelo-Vista-Controlador (MVC)
- Vistas (Parte I)
  - Componentes y Contenedores
  - Construcción de Vistas. Un ejemplo sencillo (I)
- Controladores
  - Relación Vista-Controlador
  - El mecanismo de eventos. Un ejemplo sencillo (II)
  - Relación Controlador-Modelo. Un ejemplo sencillo (III)
- Vistas (Parte II)
  - Gestores de Esquemas
  - Componentes
  - Contenedores intermedios
  - Un ejemplo más elaborado
  - Pintar en Swing



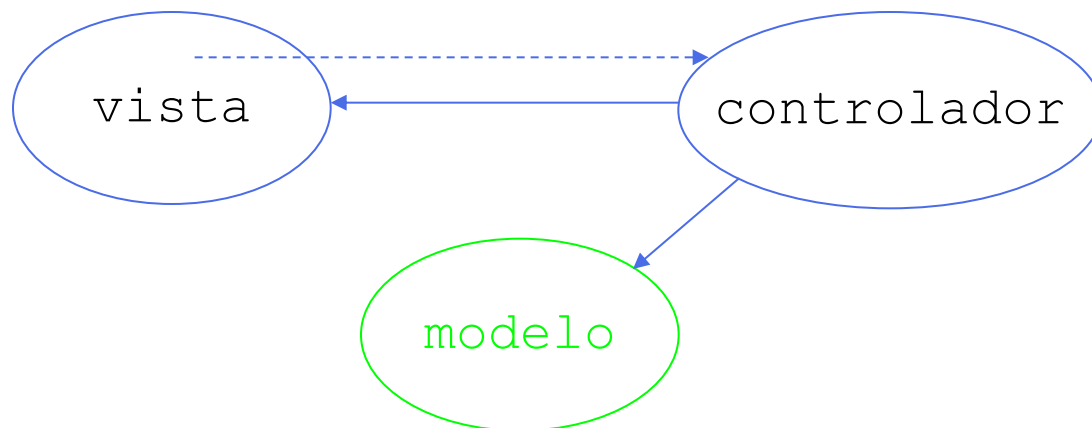
# Diseño de GUIs

- Usaremos el patrón Modelo-Vista-Controlador (MVC)
  - Modelo: los datos a manipular por la aplicación.
  - Vista: la representación de la información.
  - Controlador: la lógica de la aplicación.
    - Está pendiente de las acciones del usuario sobre la vista.
      - Estas acciones provocan una reacción del controlador:
        - » Consultar/actualizar la vista y el modelo.
- Objetivo: Independizar los distintos componentes.



# MVC: Modelo

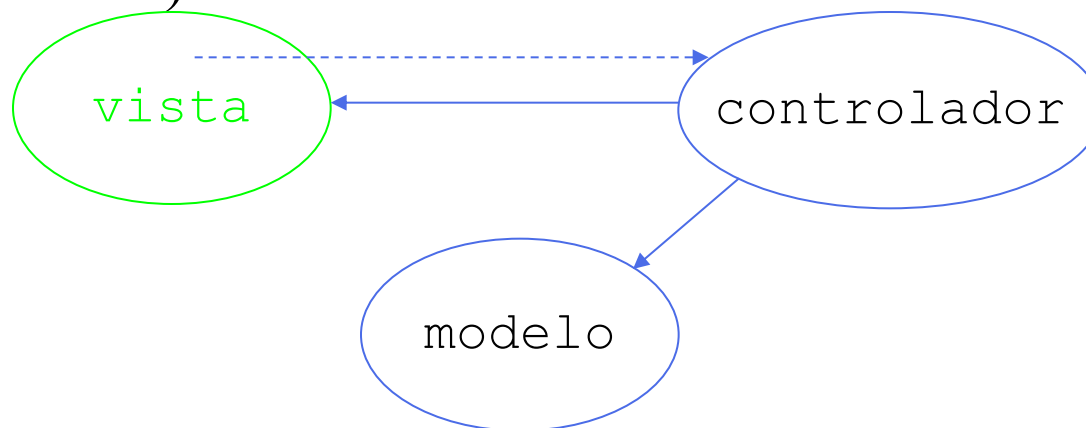
- Información para la que se realiza la interfaz gráfica.
  - Puede ser desde una variable hasta una gran cantidad de objetos.
- Debe ser lo más independiente posible de la vista y del controlador.
  - Existe aunque no tengamos interfaz gráfica.





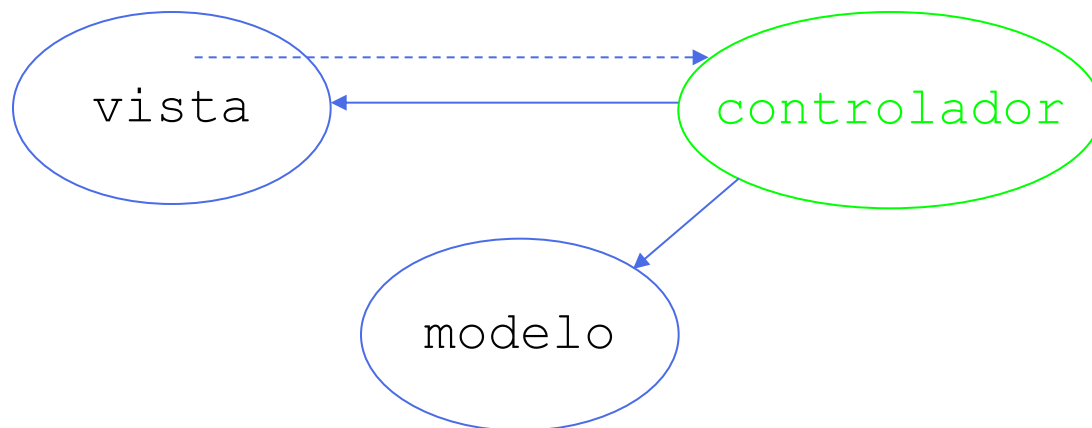
# MVC: Vista

- Representación de la información.
  - Panel que contiene botones, áreas editables de texto, etiquetas, listas desplegables, etc.
- Interactúa con el controlador.
  - En ciertas ocasiones, también con el modelo.
- Para un mismo modelo es posible generar varias vistas distintas. Por ello **es interesante definir una Interfaz** que sea implementada por las distintas vistas (aumenta la independencia en el diseño)



# MVC: Controlador

- La lógica de la aplicación.
- Es avisado cuando el usuario actúa sobre la vista.
  - Para ello, debe registrarse en ciertos elementos activos de la vista.
- En un buen diseño, varias vistas podrían disponer del mismo controlador.
- También es posible disponer de varios controladores especializados, cada uno controlando distintos eventos.



# Ejemplo MVC:

## Gestión de Cuentas Bancarias

- Permite manipular una cuenta bancaria.
- Operaciones:
  - Ingresar en la cuenta  
`void ingresa(double)`
  - Extraer de la cuenta  
`double extrae(double)`
    - Devuelve la cantidad realmente extraída según el saldo.
  - Consultar el saldo  
`double saldo()`

# El Modelo: La clase Cuenta

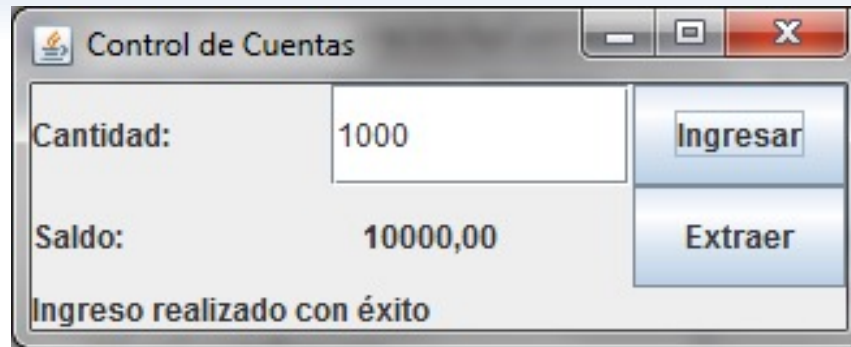
```
public class Cuenta {  
    private double saldo;  
    public Cuenta(double salInic) {  
        // lanzar excepción si saldo inicial negativo  
        saldo = salInic;  
    }  
    public void ingresa(double cant) {  
        // lanzar excepción si ingreso negativo  
        saldo += cant;  
    }  
    public double extrae(double cant) {  
        // lanzar excepción si extracción negativa  
        double realExtrae = cant;  
        if (saldo < cant) {  
            realExtrae = saldo;  
            saldo = 0;  
        } else {  
            saldo -= realExtrae;  
        }  
        return realExtrae;  
    }  
    public double saldo() {  
        return saldo;  
    }  
}
```

# Sin GUI:

```
public class ApCuenta {
    public static void main(String args[]) {
        try (Scanner sc = new Scanner(System.in)) {
            Cuenta cuenta = new Cuenta(Double.parseDouble(args[0]));
            System.out.println("Saldo en la cuenta = " + cuenta.saldo());
            System.out.print("Operación (Ingresar, Extraer, Fin): ");
            String op = sc.next();
            while (!op.equalsIgnoreCase("Fin")) {
                try {
                    switch (op.toLowerCase()) {
                        case "ingresar" :
                            System.out.print("Cantidad a Ingresar: ");
                            cuenta.ingresa(sc.nextDouble());
                            System.out.println("Saldo en la cuenta = " + cuenta.saldo());
                            break;
                        case "extraer" :
                            System.out.print("Cantidad a Extraer: ");
                            double realExt = cuenta.extrae(sc.nextDouble());
                            System.out.println("Extraído= " + realExt);
                            System.out.println("Saldo en la cuenta = " + cuenta.saldo());
                            break;
                    }
                } catch (...) {
                    System.out.print("Operación (Ingresar, Extraer, Fin): ");
                    op = sc.next();
                }
                System.out.println("Programa Finalizado");
            } catch (...) {
            }
        }
    }
}
```

# Con GUI:

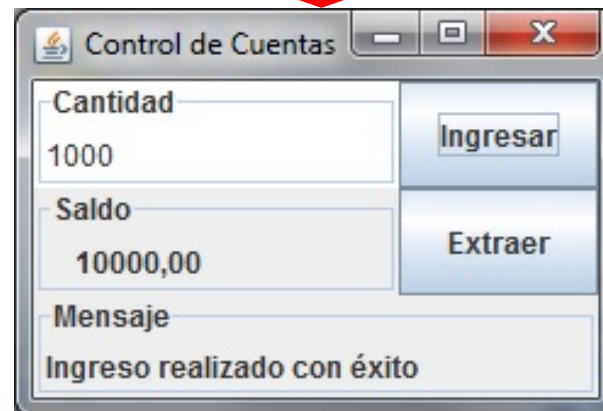
## Posibles vistas para Cuenta



Control de Cuentas

Cantidad:	1000	Ingresar
Saldo:	10000,00	Extraer
Ingreso realizado con éxito		

**Vistas**



Control de Cuentas

Cantidad	Ingresar
1000	
Saldo	Extraer
10000,00	
Mensaje	
Ingreso realizado con éxito	

# Vistas: Interfaz VistaCuenta

- Definimos una interfaz para las vistas:

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

public interface VistaCuenta {
    ...
    double obtenerCantidad();
    void saldo(double saldo);
    void mensaje(String msg);
    void controlador(ActionListener ctr);
}
```

Importante: Solo  
manejaremos  
las vistas a través  
de sus interfaces

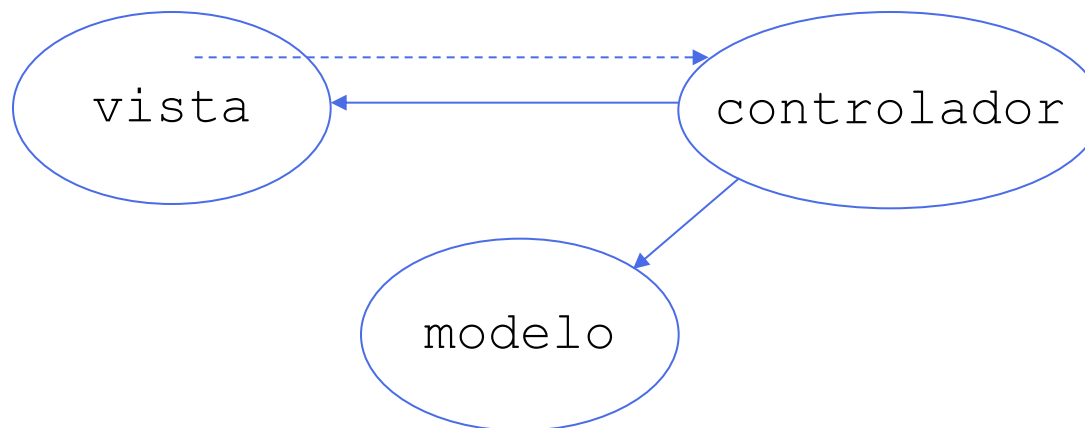
- El método **controlador (ActionListener ctr)**
  - Registra el controlador **ctr** en los componentes adecuados.
- Definimos vistas distintas implementando esta interfaz (VistaCuenta1, VistaCuenta2, ...)

# Controlador: ControlCuenta

- Mantiene dos variables de instancia:
  - El modelo: **cuenta**
  - La vista : **vistaCuenta**

Constructor:

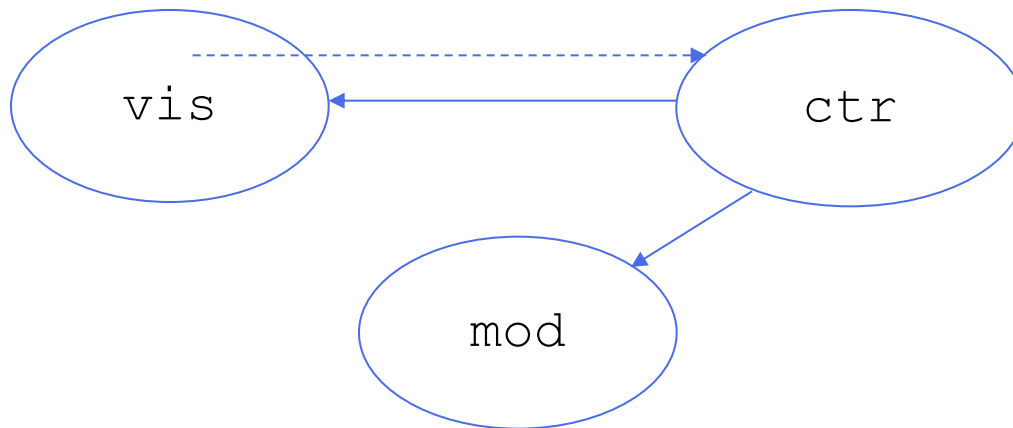
```
public ControlCuenta(VistaCuenta vc, Cuenta c ) {  
    vistaCuenta = vc;           // La vista  
    cuenta = c;                 // El modelo  
}
```





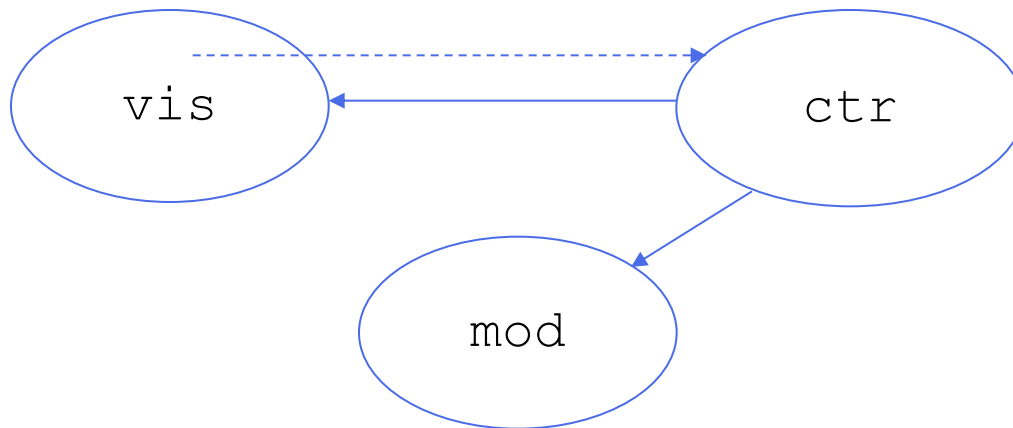
# Aplicación con MVC

```
import javax.swing.*;  
public class CuentaDemo {  
    public static void main(String args[]) {  
        VistaCuenta vistaCuenta = new VistaCuenta1();  
        Cuenta cuenta = new Cuenta(3000);  
        ControlCuenta ctrCuenta =  
            new ControlCuenta(vistaCuenta, cuenta);  
        vistaCuenta.controlador(ctrCuenta);  
        ...  
    }  
}
```

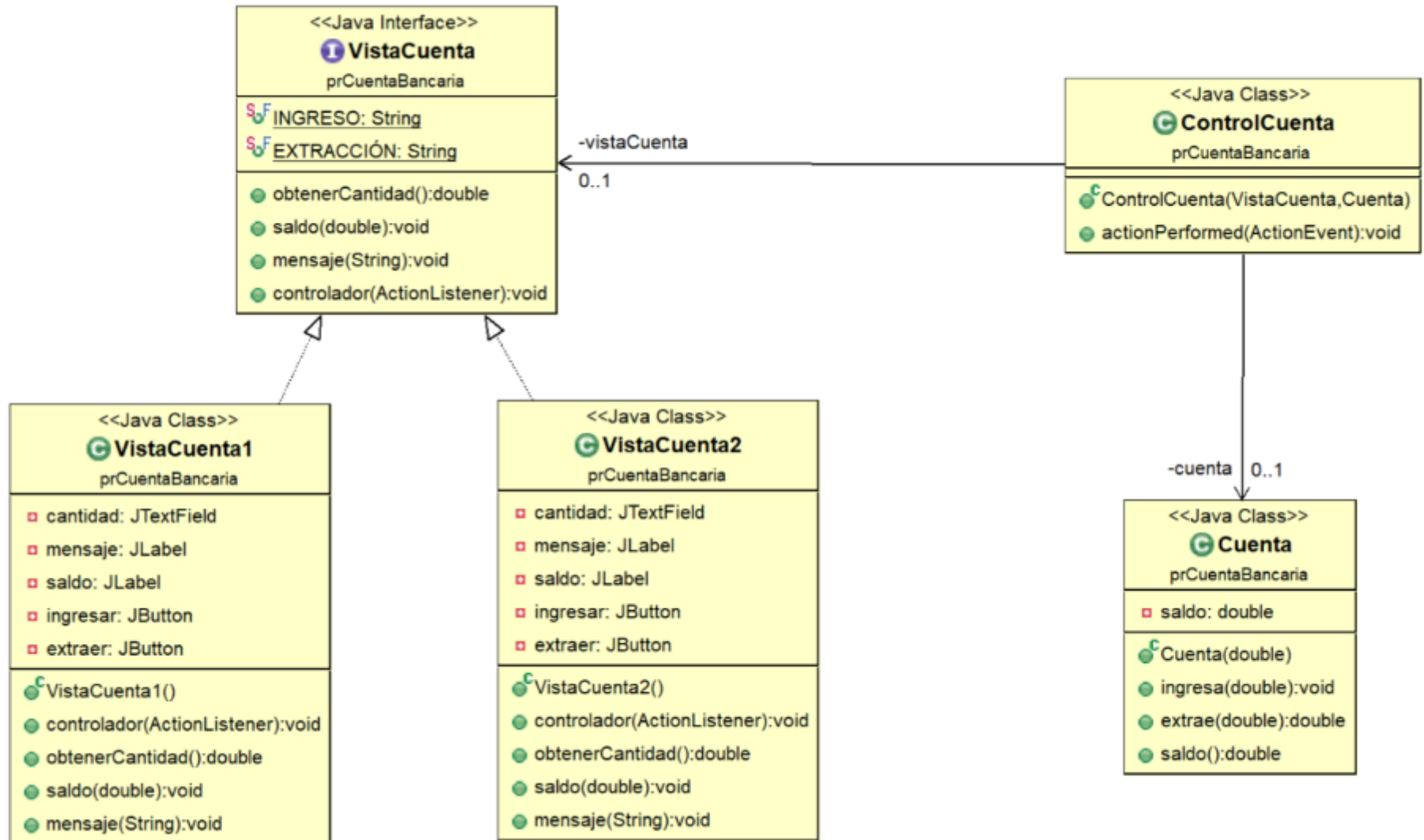


# Aplicación con MVC

```
import javax.swing.*;  
public class CuentaDemo {  
    public static void main(String args[]) {  
        VistaCuenta vistaCuenta = new VistaCuenta2();  
        Cuenta cuenta = new Cuenta(3000);  
        ControlCuenta ctrCuenta =  
            new ControlCuenta(vistaCuenta, cuenta);  
        vistaCuenta.controlador(ctrCuenta);  
        ...  
    }  
}
```

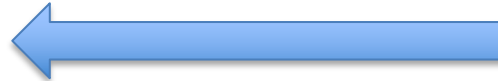


# Aplicación con MVC



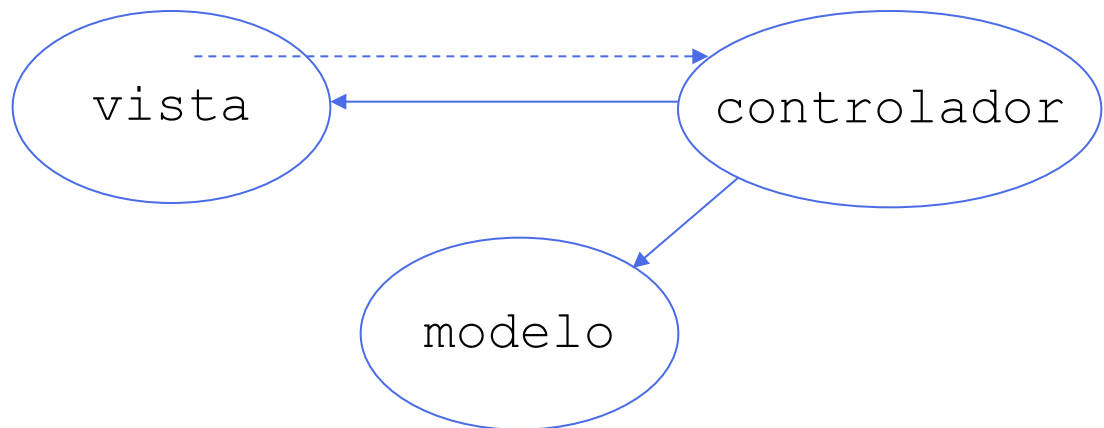
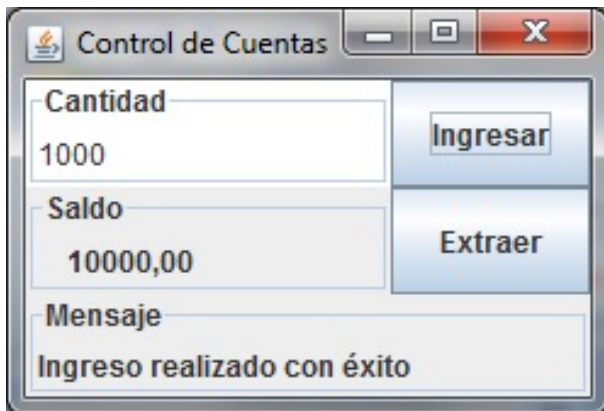
# Índice

- Los paquetes `java.awt` y `javax.swing`
- Diseño de Interfaces Gráficas de Usuario (GUIs)
  - El patrón Modelo-Vista-Controlador (MVC)
- Vistas (Parte I)
  - Componentes y Contenedores
  - Construcción de Vistas. Un ejemplo sencillo (I)
- Controladores
  - Relación Vista-Controlador
  - El mecanismo de eventos. Un ejemplo sencillo (II)
  - Relación Controlador-Modelo. Un ejemplo sencillo (III)
- Vistas (Parte II)
  - Gestores de Esquemas
  - Componentes
  - Contenedores intermedios
  - Un ejemplo más elaborado
  - Pintar en Swing



# Relación Vista-Controlador

- Ya sabemos cómo construir una VISTA (representación de la información que constituye el MODELO)
- **¿Cómo construir un CONTROLADOR (lógica de la aplicación) para esa vista, que reaccione y actúe ante la interacción del usuario con la vista (ej. pulsar un botón)?**
- **¿Cómo se produce la comunicación entre la vista y el controlador? (dejemos por ahora el modelo)**



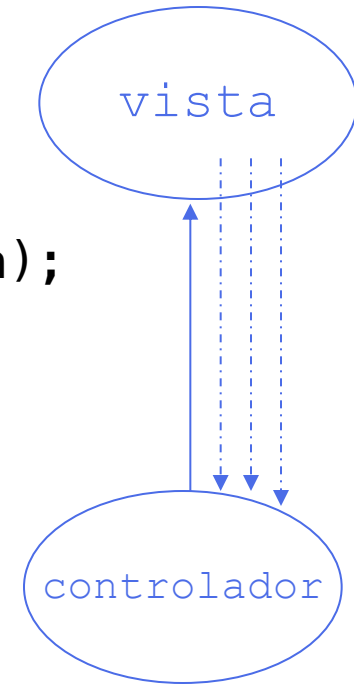
# Relación Vista-Controlador

```
// Se crea la vista, es decir, el
// panel que contiene la GUI
InterfazVista vista = new Vista();

// El controlador conoce la vista
MiControlador crt = new MiControlador(vista);

// La vista debe disponer de un método
// para registrar el controlador.
// Aquí, ctr se debe registrar en cada
// componente que se desee controlar
vista.controlador(ctr);

// Si tenemos controladores diferentes, será
// necesario disponer de distintos métodos "controlador"
```



# El mecanismo de eventos

## (Eventos)

- Un componente puede disparar un *evento* (p. ej. un botón al ser pulsado dispara un evento del tipo **ActionEvent**)
- Los eventos se implementan como subclases de  
`java.util.EventObject`
- Los eventos se encuentran en los paquetes:  
`java.awt.event` y `javax.swing.event`
- El nombre de la clase de un evento tiene el formato  
**XxxxxEvent**
- *Ejemplos:*  
**ActionEvent**      **FocusEvent**      **MouseEvent**      ...

# El mecanismo de eventos

## (Registro de Controladores)

- Cuando un componente dispara un evento, se comunica con cada uno de los objetos *controladores* u *oyentes* (*listeners*) que tenga *registrados* el componente.
- El registro de un controlador a un componente se realiza mediante un método del componente sobre el que se registra:

**addXxxxxListener (XxxxxListener)**

- El argumento será el objeto controlador, cuya clase debe implementar la interfaz **XxxxxListener**.
- Un controlador implementará la interfaz adecuada para poder tratar eventos de un determinado tipo:

- Ejemplos:

Evento	Interfaz
<b>ActionEvent</b>	<b>ActionListener</b>
<b>FocusEvent</b>	<b>FocusListener</b>
<b>MouseEvent</b>	<b>MouseListener</b>

...



# El mecanismo de eventos

## (Tratamiento de Eventos)

- Cuando un componente dispara un evento, envía a cada uno de sus controladores un *mensaje* que lleva como argumento el evento generado.
- El controlador “atrapa” dicho mensaje mediante la implementación del método correspondiente.
- Las diferentes interfaces relacionadas con eventos obligan a implementar distintos métodos por parte de los controladores correspondientes.

• Ejemplos:	Interfaz	Métodos
	<b><i>ActionListener</i></b>	<b>actionPerformed(ActionEvent)</b>
	<b><i>MouseListener</i></b>	<b>mouseClicked(MouseEvent)</b>
		<b>mouseEntered(MouseEvent)</b>
		<b>mouseExited(MouseEvent)</b>
		<b>mousePressed(MouseEvent)</b>
		<b>mouseReleased(MouseEvent)</b>

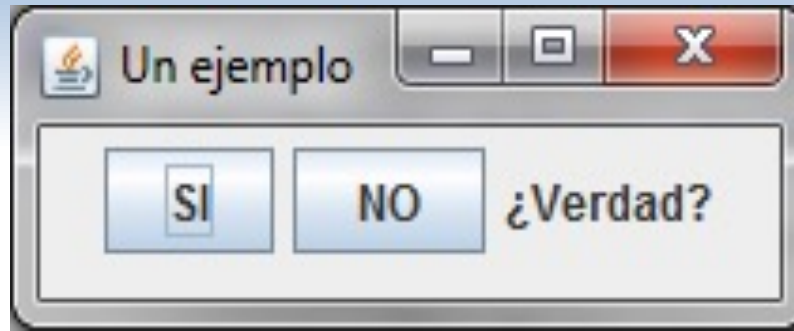
...

# El mecanismo de eventos

## (Distinción de Eventos)

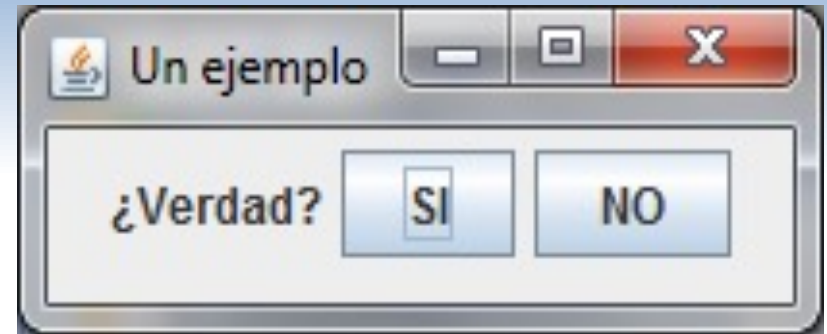
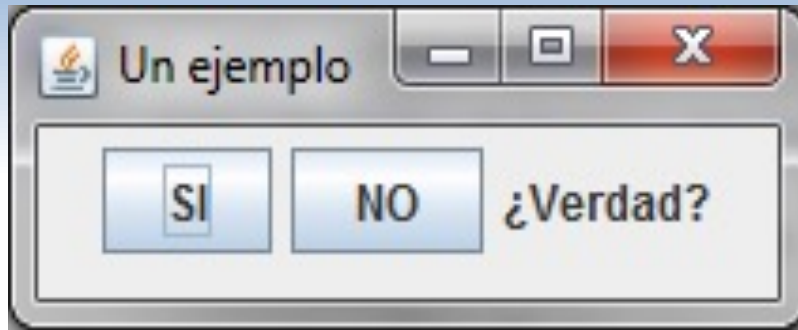
- Fundamentalmente dos mecanismos diferentes:
  - Método de instancia para conocer quién dispara el evento:  
**Object getSource()**
  - Utilizar identificadores de acciones:  
Consulta sobre una acción
    - **String getActionCommand()**  
previamente establecida cuando se registró el controlador
    - **setActionCommand(String)**

# Un ejemplo sencillo (II)



- El evento **ActionEvent** se dispara si:
  - Se pulsa un botón de cualquier tipo.
  - Se selecciona una opción de menú.
  - Se pulsa Enter en un campo de texto.
- Método en la interfaz **ActionListener**:  

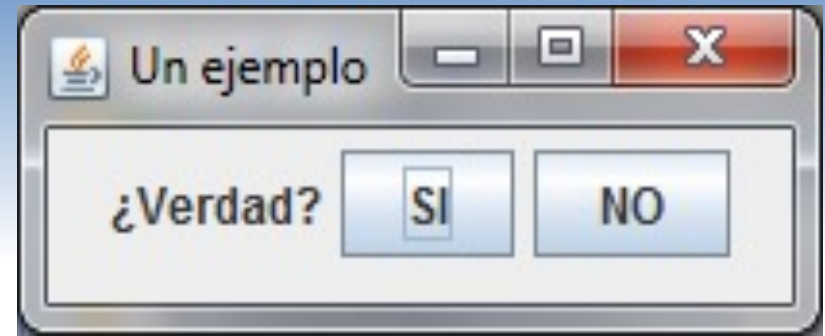
```
void actionPerformed(ActionEvent)
```



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

public interface InterfazVista {
    String SI = "SI";
    String NO = "NO";
    void controlador(ActionListener ctr);
    void cambiaTexto(String s);
}
```

*INTERFAZ  
PARA  
VISTAS*



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

```
public class Controlador implements ActionListener {  
    private InterfazVista vista;
```

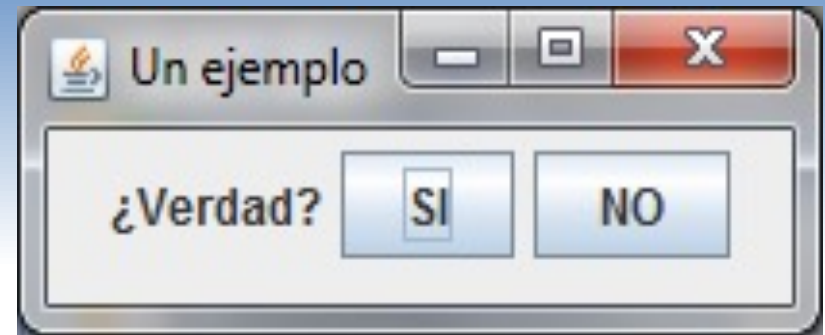
```
    public Controlador(InterfazVista v) {  
        vista = v;  
    }
```

*CONTROLADOR*

```
    public void actionPerformed(ActionEvent e) {  
        String comando = e.getActionCommand();  
        if (comando.equals(InterfazVista.SI)) {  
            vista.cambiaTexto("Sí pulsado");  
        } else {  
            vista.cambiaTexto("No pulsado");  
        }  
    }  
}
```

Tratamiento de  
eventos





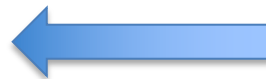
```
import java.awt.event.*;
```

```
public class Controlador implements ActionListener {  
    private InterfazVista vista;
```

```
    public Controlador(InterfazVista v) {  
        vista = v;  
    }
```

*CONTROLADOR*

```
    public void actionPerformed(ActionEvent e) {  
        JButton boton = (JButton) e.getSource();  
        String texto = boton.getText();  
        if (texto.equals(InterfazVista.SI)) {  
            vista.cambiaTexto("Sí pulsado");  
        } else {  
            vista.cambiaTexto("No pulsado");  
        }  
    }
```

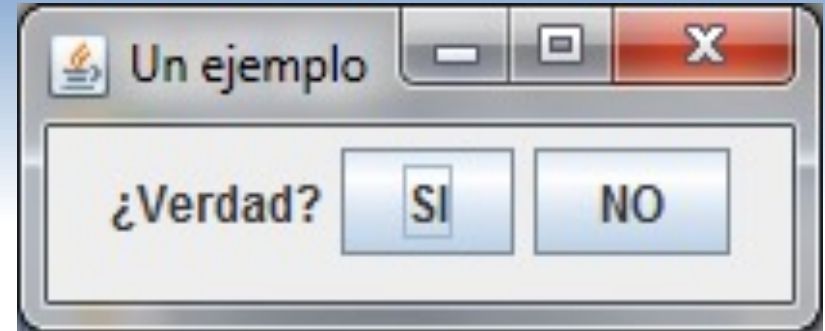


Tratamiento de  
eventos  
(otra forma)

```
}
```

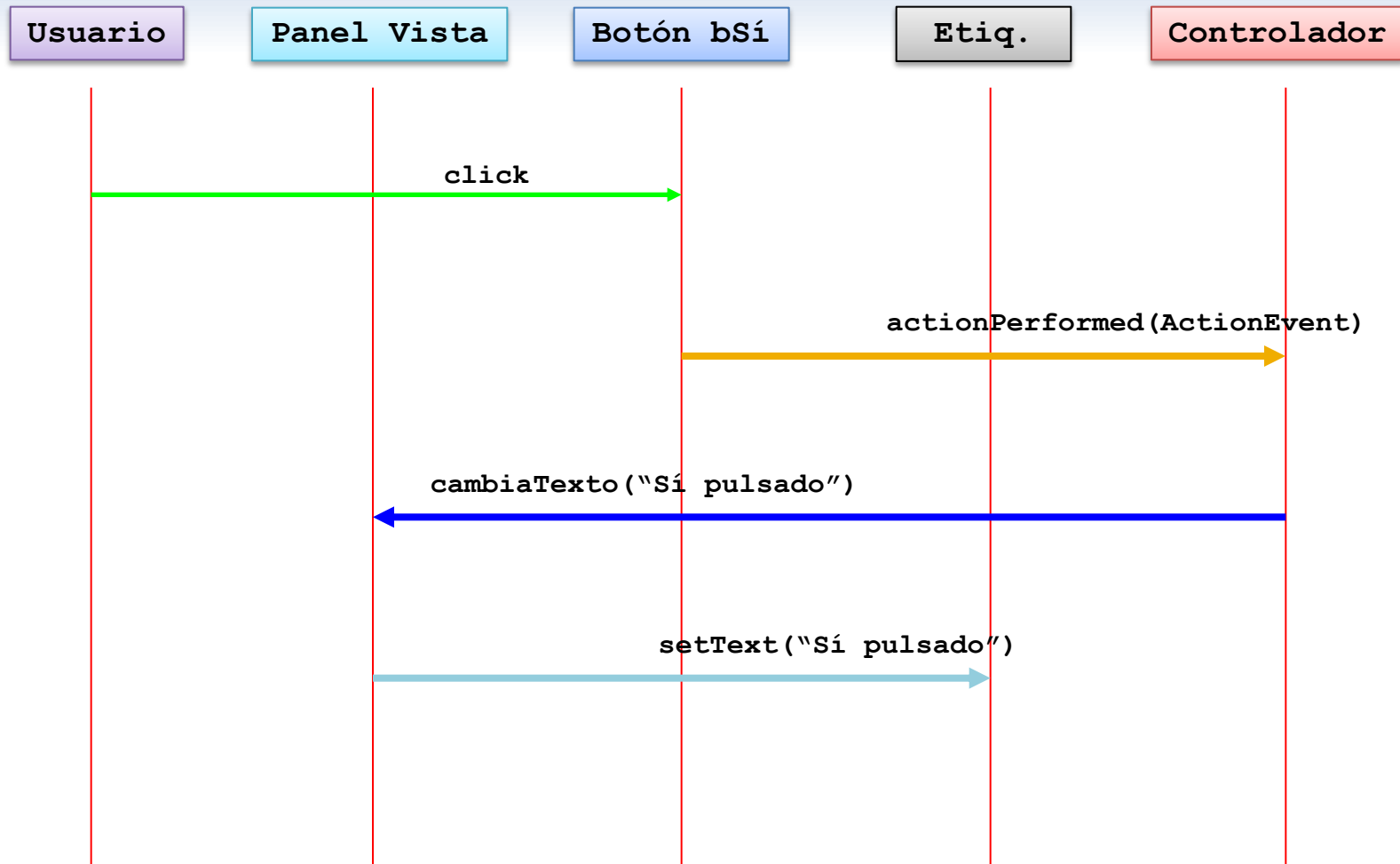
```
import javax.swing.JFrame;
```

```
public class ClasePrincipal {  
    public static void main(String[] args) {  
        InterfazVista vista = new Vista1();  
        Controlador control = new Controlador(vista);  
        vista.controlador(control);  
  
        JFrame ventana = new JFrame("Un ejemplo ");  
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        ventana.setContentPane((JPanel) vista);  
        ventana.pack();  
        ventana.setVisible(true);  
    }  
}
```



*Clase Principal*

# Escenario posible





# Relación Controlador-Modelo

- En nuestro ejemplo anterior, no había modelo. Sólo hemos creado la vista y el controlador y se ha analizado la relación entre ambos.
- Lo normal es que exista un modelo también.
- **El modelo puede crearse dentro del controlador o bien se pasa como parámetro del constructor** (al igual que ha ocurrido con la vista)
- **El controlador tendrá otra/s variable/s de instancia para almacenar el modelo.** De esta forma puede interactuar con él, al igual que lo hace con la vista.

# Esquema genérico de un controlador (ActionListener)

(recibiendo Vista y Modelo)

```
public class Controlador implements ActionListener {  
    private Vista vista;  
    private Modelo modelo; // podrían ser varias variables  
  
    public Controlador(Vista v, Modelo m) {  
        vista = v;  
        modelo = m;  
        // posible interacción inicial con la vista  
    }  
  
    public void actionPerformed(ActionEvent e) {  
        ...  
    }  
}
```

# Esquema genérico de un controlador (ActionListener)

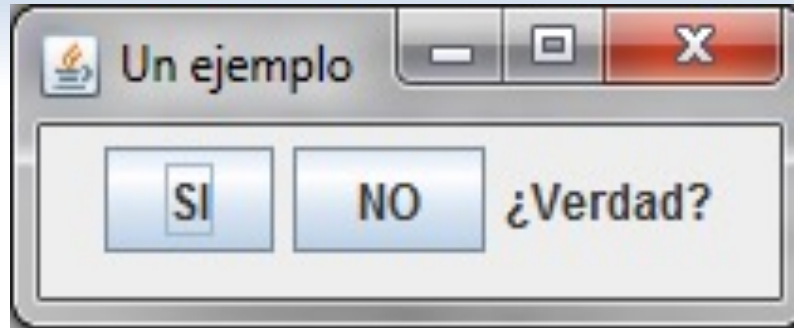
(el Modelo lo crea el controlador)

```
public class Controlador implements ActionListener {  
    private Vista vista;  
    private Modelo modelo; // podrían ser varias variables  
  
    public Controlador(Vista v) {  
        vista = v;  
        // posible interacción inicial con la vista  
    }  
  
    public void actionPerformed(ActionEvent e) {  
        ... // creación del modelo tras recibir evento  
        correspondiente  
    }  
}
```

# Esquema genérico del método actionPerformed

```
public void actionPerformed(ActionEvent e) {  
    // obtener el tipo de botón (comando)  
    try {  
        // sentencia if-else-if (o switch) para tratar cada botón pulsado  
        // (interactuando con la vista y con el modelo)  
    } catch (...) {  
        // mostrar en la vista mensaje de error  
    }  
}
```

# Un ejemplo sencillo (III)



- Añadimos un modelo simple:
  - Almacena el número de veces que se ha pulsado el botón SI menos el número de veces que se ha pulsado el botón NO
- El controlador hará que cada vez que se pulse un botón, se muestre el valor de dicho contador en la etiqueta que inicialmente tiene el valor “¿verdad?”

```
public class Contador {  
    private int cont;  
  
    public Contador() {  
        cont = 0;  
    }  
    public void incrementar() {  
        cont++;  
    }  
    public void decrementar() {  
        cont--;  
    }  
    public int valor() {  
        return cont;  
    }  
}
```

*MODELO*

```
public class Controlador implements ActionListener {
```

```
    private InterfazVista vista;
```

```
    private Contador modelo;
```

# CONTROLADOR

```
    public Controlador(InterfazVista vista, Contador modelo) {
```

```
        this.vista = vista;
```

```
        this.modelo = modelo;
```

```
    }
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        String comando = e.getActionCommand();
```

```
        // try {
```

```
            if (comando.equals(InterfaceVista.SI)) {
```

```
                modelo.incrementar();
```

```
                vista.cambiaTexto(String.valueOf(modelo.valor()));
```

```
            } else {
```

```
                modelo.decrementar();
```

```
                vista.cambiaTexto(String.valueOf(modelo.valor()));
```

```
            }
```

```
        // } catch () {...}
```

```
    }
```

```
}
```

```
import javax.swing.JFrame;
```

```
public class ClasePrincipal {  
    public static void main(String[] args) {  
        InterfazVista vista = new Vista1();  
        Contador modelo = new Contador();  
        Controlador control = new Controlador(vista, modelo);  
        vista.controlador(control);  
  
        JFrame ventana = new JFrame("Un ejemplo ");  
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        ventana.setContentPane((JPanel) vista);  
        ventana.pack();  
        ventana.setVisible(true);  
  
    }  
}
```

*Clase Principal*

```
}
```