Modelo - Vista - Controlador

MVC es un acrónimo que significa "Model-View-Controller" (Modelo-Vista-Controlador en español). Es un patrón de diseño de software ampliamente utilizado en el desarrollo de aplicaciones informáticas, especialmente en el desarrollo de aplicaciones web y de escritorio. El objetivo principal del patrón MVC es separar la lógica de la aplicación en tres componentes distintos, cada uno con una función específica:

- **Modelo** (Model): El Modelo representa los datos y la lógica de negocio de la aplicación. En otras palabras, maneja la información y las operaciones relacionadas con esa información. Por lo general, el Modelo es responsable de la interacción con la base de datos o el almacenamiento de datos, así como de las operaciones que se realizan en esos datos.

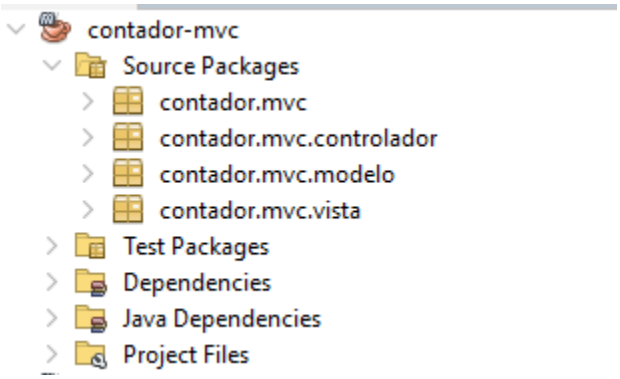
- **Vista** (View): La Vista es la parte de la aplicación que se encarga de la presentación de la información al usuario. Es responsable de cómo se muestran los datos y de la interfaz de usuario. No realiza operaciones en los datos ni procesa la lógica de la aplicación; simplemente muestra la información de manera legible y comprensible para el usuario.

- **Controlador** (Controller): El Controlador actúa como intermediario entre el Modelo y la Vista. Responde a las interacciones del usuario, como clics de botón o entradas de teclado, y realiza las operaciones necesarias en el Modelo o la Vista. El Controlador controla la lógica de flujo de la aplicación y decide cómo se deben manejar las solicitudes del usuario.

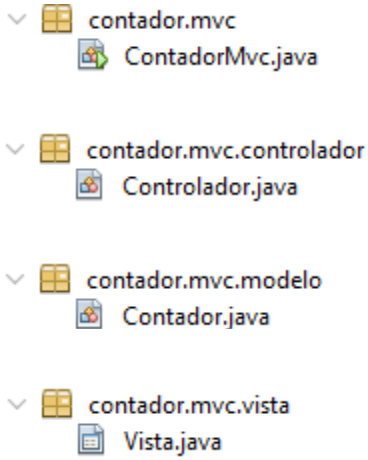
Contador en Java utilizando MVC

El proyecto consiste en realizar un contador, que incrementa o decrementa su valor de acuerdo a los eventos accionados por el usuario, utilizando el patrón de diseño MVC.

Estructura del proyecto



Paquetes del proyecto



“**Contador.mvc**” es el paquete raíz del cual se estructuran los demás paquetes en el proyecto.

Clases del proyecto

ContadorMvc.java

```java
package contador.mvc;

import contador.mvc.controlador.Controlador;
import contador.mvc.modelo.Contador;
import contador.mvc.vista.Vista;

public class ContadorMvc {

    public static void main(String[] args) {

        Vista vista = new Vista();
        Contador contador = new Contador();
        Controlador controlador = new Controlador(vista, contador);

        vista.setActionListener(controlador);

        java.awt.EventQueue.invokeLater(() -> {
            vista.setVisible(true);
        });
    }
}
```

La clase `ContadorMvc` es la entrada principal de una aplicación que sigue el patrón Modelo-Vista-Controlador (MVC). En este código, se crea una instancia de la vista, el modelo y el controlador, luego se conecta la vista con el controlador y se muestra la vista en una interfaz gráfica. La clase facilita la inicialización y coordinación de estos componentes MVC para el contador.

Controlador.java

```java
package contador.mvc.controlador;

import contador.mvc.modelo.Contador;
import contador.mvc.vista.Vista;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Controlador implements ActionListener{

    private final Vista vista;
    private final Contador contador; //modelo

    public Controlador(Vista vista, Contador contador) {
        this.vista = vista;
        this.contador = contador;
    }

    public void iniciarControlador(){
        vista.setActionListener(this);
        vista.setVisible(true);
        mostrarContador();
    }

    public void mostrarContador(){
        int numero = contador.getContador();
        vista.mostrarContador(numero);
    }


    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        if(command.equals("+")){
            contador.incrementar();
        } else if (command.equals("-")){
            contador.decrementar();
        }
```

```
        mostrarContador();
    }

}
```

La clase Controlador es parte del patrón Modelo-Vista-Controlador (MVC) en una aplicación de contador. Su función es controlar la interacción entre el modelo (Contador) y la vista (Vista). Responde a eventos de botón (+/-), actualiza el modelo según las acciones y actualiza la vista para reflejar el contador actual. La interfaz ActionListener se utiliza para manejar eventos de acción.

La interface ActionListener

La interfaz `ActionListener` en Java es parte del paquete `java.awt.event` y se utiliza para manejar eventos de acción, comúnmente asociados a componentes de interfaz de usuario, como botones, menús y otros elementos interactivos. Su función principal es la de "escuchar" o "atender" eventos de acción generados por estos componentes y responder a ellos con código personalizado. Aquí está cómo funciona:

1. **Implementación de la Interfaz:** Para utilizar `ActionListener`, debes crear una clase que implemente esta interfaz. Esto significa que debes proporcionar una implementación para el método `actionPerformed(ActionEvent e)` que se encuentra en la interfaz. Este método se ejecutará cuando ocurra un evento de acción.

2. **Asociación con Componentes:** Después de implementar `ActionListener`, debes asociar una instancia de esta clase con un componente que genere eventos de acción. Comúnmente, esto se hace mediante el método `addActionListener(ActionListener listener)` de los componentes. Por ejemplo, en el código que proporcionaste, se utiliza `vista.setActionListener(this)` para asociar una instancia de `Controlador` a la vista.

3. **Manejo de Eventos:** Cuando ocurre un evento de acción en el componente asociado (por ejemplo, un clic en un botón), se llama al método `actionPerformed` de la instancia de `ActionListener`. El objeto `ActionEvent` proporcionado como argumento contiene información sobre el evento, como el comando asociado (a través de `getActionCommand()`) y la fuente del evento (a través de `getSource()`), lo que permite identificar qué componente generó el evento y qué acción se debe realizar.

4. **Respuesta a Eventos:** Dentro del método `actionPerformed`, puedes escribir código personalizado para responder al evento. Por lo general, esto implica realizar acciones específicas basadas en el tipo de evento y la fuente. Por ejemplo, en tu código, se verifica si el comando es "+" o "-", y se llama a métodos en el modelo (`Contador`) y en la vista (`Vista`) para actualizar el contador y la interfaz gráfica en consecuencia.

En resumen, `ActionListener` proporciona una forma de controlar la interacción del usuario con componentes de la interfaz de usuario, lo que permite crear aplicaciones interactivas y responder a eventos como clics de botón, selecciones de menú, etc.

Contador.java

```java
package contador.mvc.modelo;

public class Contador {

    private int contador;

    public Contador(){
        contador = 0;
    }

    public int getContador() {
        return contador;
    }

    public void setContador(int contador) {
        this.contador = contador;
    }

    public void incrementar(){
        this.contador++;
    }

    public void decrementar(){
        this.contador--;
    }

}
```

La clase `Contador` representa un modelo en el contexto de un patrón Modelo-Vista-Controlador (MVC) en una aplicación de contador. Aquí tienes una breve explicación de esta clase:

1. **Atributos:** La clase contiene un atributo privado llamado `contador` que almacena el valor del contador.

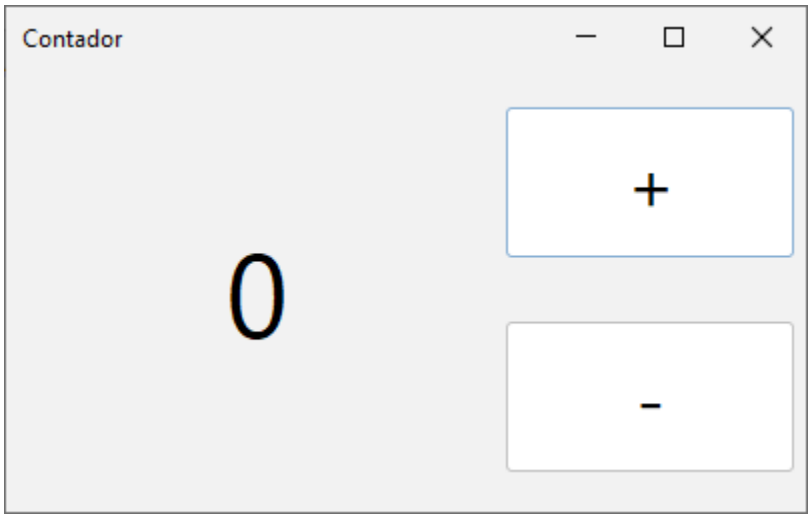
2. **Constructor:** El constructor de la clase inicializa el contador en 0 cuando se crea una instancia de `Contador`.

3. **Métodos:**

- **getContador():** Este método permite obtener el valor actual del contador.
- **setContador(int contador):** Este método permite establecer el valor del contador, lo cual no parece ser utilizado en el código proporcionado.
- **incrementar():** Este método incrementa en 1 el valor del contador.
- **decrementar():** Este método decrementa en 1 el valor del contador.

En resumen, la clase `Contador` encapsula la lógica relacionada con el contador, proporcionando métodos para obtener, establecer, incrementar y decrementar su valor. Es un componente clave en la implementación del modelo dentro del patrón MVC, donde el modelo representa los datos y la lógica de negocio de la aplicación.

Vista.java



La clase `Vista` en tu código representa la vista en el patrón Modelo-Vista-Controlador (MVC) para una aplicación de contador.

```java
package contador.mvc.vista;

import java.awt.event.ActionListener;

public class Vista extends javax.swing.JFrame {

    public Vista() {
        initComponents();
    }

    public void mostrarContador(int numero){
        contadorLb.setText("" + numero);
    }

    public void setActionListener(ActionListener actionListener) {
        incrementarBtn.addActionListener(actionListener);
        decrementarBtn.addActionListener(actionListener);
    }

   ...
}
```

1. **Constructor:** El constructor `Vista()` inicializa la interfaz gráfica de usuario (UI) utilizando el método `initComponents()`, que generalmente se genera automáticamente por el entorno de desarrollo integrado (IDE) NetBeans. Esta función crea y configura los componentes visuales, como botones y etiquetas, que se utilizarán para interactuar con el usuario.

2. **mostrarContador(int numero):** Este método toma un número como argumento y actualiza la etiqueta `contadorLb` en la vista con ese número. Es responsable de mostrar el valor actual del contador en la interfaz gráfica.

3. **setActionListener(ActionListener actionListener)**: Este método permite asociar un objeto `ActionListener` a los botones de incremento y decremento en la vista. Cuando estos botones se presionan, generan eventos de acción, y el `ActionListener` proporcionado se encargará de manejar estos eventos y realizar las acciones apropiadas en función de los clics de los botones.

En resumen, la clase `Vista` se encarga de la representación visual de la aplicación de contador y proporciona métodos para mostrar el contador en la interfaz gráfica y para asociar

un `ActionListener` que maneja los eventos de los botones. Esto cumple con la parte "Vista" del patrón MVC.