Universidad Autónoma de Santo Domingo

Facultad de Ciencia

Escuela de Informática

Asignatura:

Laboratorio Lenguaje de Programación III

Tema:

Modelo MVC

Estudiante:

Genesis Batista Mejía - 100572904

Maestro:

Radhames Silverio González

¿Qué es el patrón de diseño MVC?

El patrón de diseño Modelo Vista Controlador (MVC) especifica que una aplicación consta de un modelo de datos, información de presentación e información de control. El patrón requiere que cada uno de estos se separe en objetos diferentes.

- El patrón MVC separa las preocupaciones de una aplicación en tres componentes distintos, cada uno responsable de un aspecto específico de la funcionalidad de la aplicación.
- Esta separación de preocupaciones hace que la aplicación sea más fácil de mantener y ampliar, ya que los cambios en un componente no requieren cambios en los otros componentes.

Componentes del patrón de diseño MVC

1. Modelo

El componente Modelo del patrón de diseño MVC (Modelo-Vista-Controlador) muestra los datos y la lógica de negocio de una aplicación. Es responsable de gestionar los datos de la aplicación, procesar las reglas de negocio y responder a las solicitudes de información de otros componentes, como la Vista y el Controlador.

2. Ver

Muestra los datos del Modelo al usuario y envía las entradas del usuario al Controlador. Es pasivo y no interactúa directamente con el Modelo. En su lugar, recibe datos del Modelo y envía las entradas del usuario al Controlador para su procesamiento.

3. Controlador

El controlador actúa como intermediario entre el modelo y la vista. Gestiona la entrada del usuario y actualiza el modelo según corresponda, así como la vista para reflejar los cambios en el modelo. Contiene la lógica de la aplicación, como la validación de entradas y la transformación de datos.

Comunicación entre los componentes

El siguiente flujo de comunicación garantiza que cada componente sea responsable de un aspecto específico de la funcionalidad de la aplicación, lo que conduce a una arquitectura más escalable y fácil de mantener.

- **Interacción del usuario con la vista:** el usuario interactúa con la vista, como hacer clic en un botón o ingresar texto en un formulario.
- **La vista recibe la entrada del usuario:** la vista recibe la entrada del usuario y la envía al controlador.
- **El controlador procesa la entrada del usuario:** El controlador recibe la entrada del usuario desde la vista. La interpreta, realiza las operaciones necesarias (como actualizar el modelo) y decide cómo responder.
- **El controlador actualiza el modelo:** el controlador actualiza el modelo en función de la entrada del usuario o la lógica de la aplicación.
- **El modelo notifica a la vista sobre los cambios:** si el modelo cambia, notifica a la vista.
- **Ver solicita datos del modelo:** la vista solicita datos del modelo para actualizar su visualización.

- **El controlador actualiza la vista:** el controlador actualiza la vista en función de los cambios en el modelo o en respuesta a la entrada del usuario.
- **La vista representa la IU actualizada:** la vista representa la IU actualizada en función de los cambios realizados por el controlador.

¿Cuándo utilizar el patrón de diseño MVC?

A continuación, se explica cuándo utilizar el patrón de diseño MVC:

- **Aplicaciones complejas**: Use MVC para aplicaciones con muchas funciones e interacciones de usuario, como sitios de comercio electrónico. Ayuda a organizar el código y a gestionar la complejidad.
- **Cambios frecuentes en la interfaz de usuario**: si la interfaz de usuario necesita actualizaciones periódicas, MVC permite realizar cambios en la vista sin afectar la lógica subyacente.
- **Reutilización de componentes**: si desea reutilizar partes de su aplicación en otros proyectos, la estructura modular de MVC lo hace más fácil.
- **Requisitos de prueba**: MVC admite pruebas exhaustivas, lo que le permite probar cada componente por separado para un mejor control de calidad.

Ejemplo del patrón de diseño MVC

1. Modelo (Clase de estudiantes)

Representa los datos (nombre del estudiante y número de matrícula) y proporciona métodos para acceder y modificar estos datos.

```
class Student {
    private String matricula;
    private String nombre;

    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    public String getnombre() {
        return nombre;
    }

    public void setnombre(String nombre) {
        this.nombre = nombre;
    }
}
```

2. Vista (clase StudentView)

Representa cómo se deben mostrar los datos (datos del estudiante) al usuario. Contiene un método ( ) para imprimir el nombre y el número de matrícula del estudiante.printStudentDetails

```
class StudentView {
    public void printStudentDetails(String nombreEstudiante, String
matriculaEstudiante) {
        System.out.println("Estudiante:");
        System.out.println("Nombre: " + nombreEstudiante);
        System.out.println("Matricula: " + matriculaEstudiante);
    }
}
```

3. Controlador (clase StudentController)

Actúa como intermediario entre el Modelo y la Vista. Contiene referencias a los objetos Modelo y Vista. Proporciona métodos para actualizar el Modelo (p. ej., ,) y la Vista ( ).**setStudentNamesetStudentRollNoupdateView**

```
class StudentController {
    private Student model;
    private StudentView view;

    public StudentController(Student model, StudentView view) {
        this.model = model;
        this.view = view;
    }

    public void setnombreEstudiante (String nombre) {
        model.setnombre(nombre);
    }

    public String getnombreEstudiante () {
        return model.getnombre();
    }

    public void setmatriculaEstudiante(String matricula) {
        model.setmatricula(matricula);
    }

    public String getmatriculaEstudiante() {
        return model.getmatriculaEstudiante();
    }

    public void updateView() {
        view.printStudentDetails(model.getnombre(), model.getmatricula());
    }
}
```

```java
public class MVCPattern {
    public static void main(String[] args) {
        Student model = retriveStudentFromDatabase();

        StudentView view = new StudentView();

        StudentController controller = new StudentController(model, view);

        controller.updateView();

        controller.setnombreEstudiante("Natacha Batista");

        controller.updateView();
    }

    private static Student retriveStudentFromDatabase() {
        Student student = new Student();
        student.setnombre("Genesis Batista");
        student.setmatricula("100572904");
        return student;
    }
}
```

```
Estudiante:
Nombre: Genesis Batista
Matricula: 100572904
Estudiante:
Nombre: Natacha Batista
Matricula: 100572904
```