

**Universidad de las Fuerzas Armadas - ESPE**

Departamento de Ciencias de la Computación

Carrera de Ingeniería de Software

Análisis y Diseño de Software - NRC:22426

**Tema:**

**Grupo: 4**

**Integrantes:**

Diego Casignia

Anthony Villarreal

Javier Ramos

**Profesora**: Ing. Jenny Ruiz

***Taller 1 Patron***

***Link:***

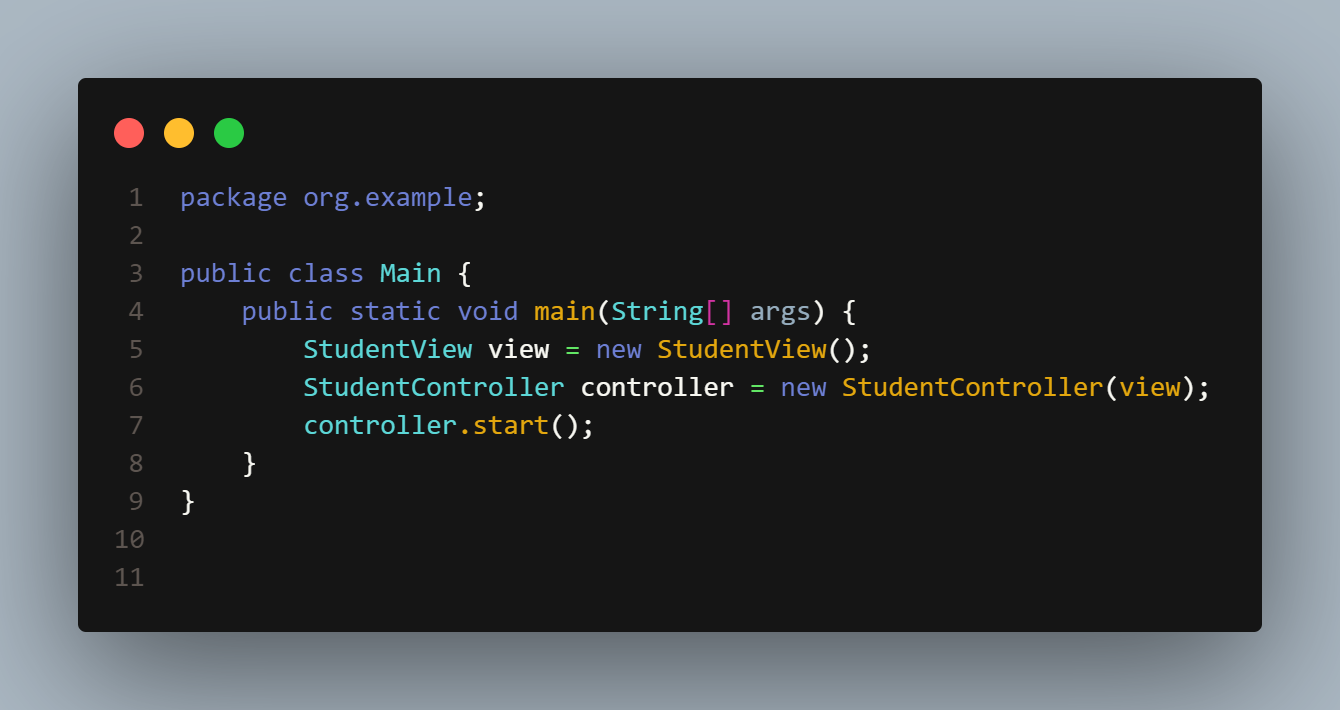
[***https://onlinegdb.com/dt7Nbq3E8***](https://onlinegdb.com/dt7Nbq3E8)

| ***RÚBRICA PARA EVALUACIÒN*** | | |  |
| --- | --- | --- | --- |
|  |  |  |  |
| ***Preguntas*** | ***Puntos*** | ***Calificación*** | ***Observación*** |
| 1. La clase main, llama a la vista (View), y al controlador (Controler) | 1 |  |  |
| 2. Elaborar el modelo en base de datos (BD) al dado, únicamente se adaptando, añadiendo tres nuevos Constructores | 1 |  |  |
| 3. Para la Vista view crear un método de inserción, el cual simula cómo sería la inserción tradicional | 1 |  |  |
| 4. El controlador se cambió en su mayoría, haciendo uso únicamente de los métodos que se requieren para hacer un intermediario entre el modelo, y la vista. | 1 |  |  |
| 5. Crear una clase que simula una base de datos, la cual brinda apoyo en la administración de los datos quemados. (05 estudiantes). | 1 |  |  |
| EJECUCIÓN |  |  |  |
| **TOTAL** | **5** | **/5** |  |

**REQUISITOS:**

Para cada RF realizar la revisión de código y explicar a través de la ejecución el funcionamiento de MVC

1. **La clase main, llama al View, y al controlador**

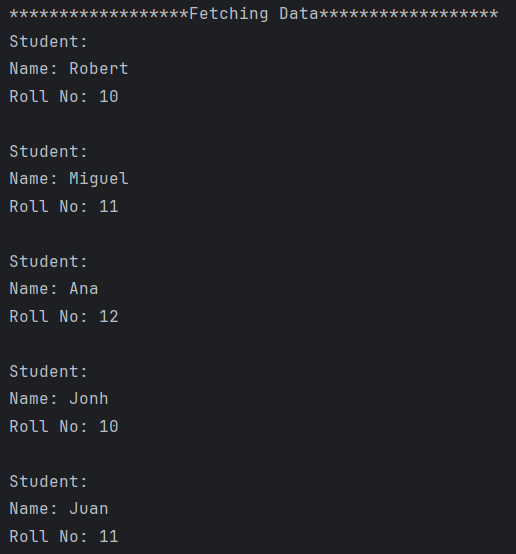


La clase Main crea una instancia de StudentView (la vista) y una instancia de StudentController (el controlador), pasando la vista como parámetro al controlador.

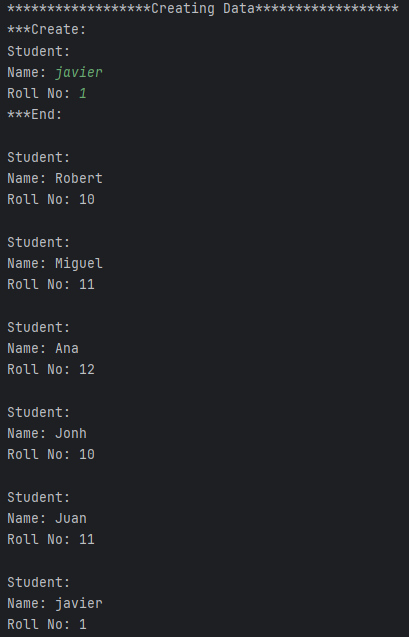
Se llama al método start() del controlador, iniciando la lógica del programa.

Esto sigue el patrón MVC, donde el Main actúa como el inicializador que conecta la vista y el controlador.

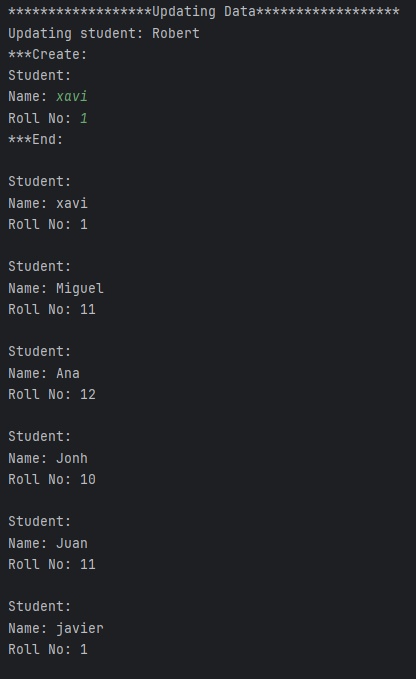
En ejecución del Main se llama a la vista y al controlador, mostrando los datos quemados inicialmente:



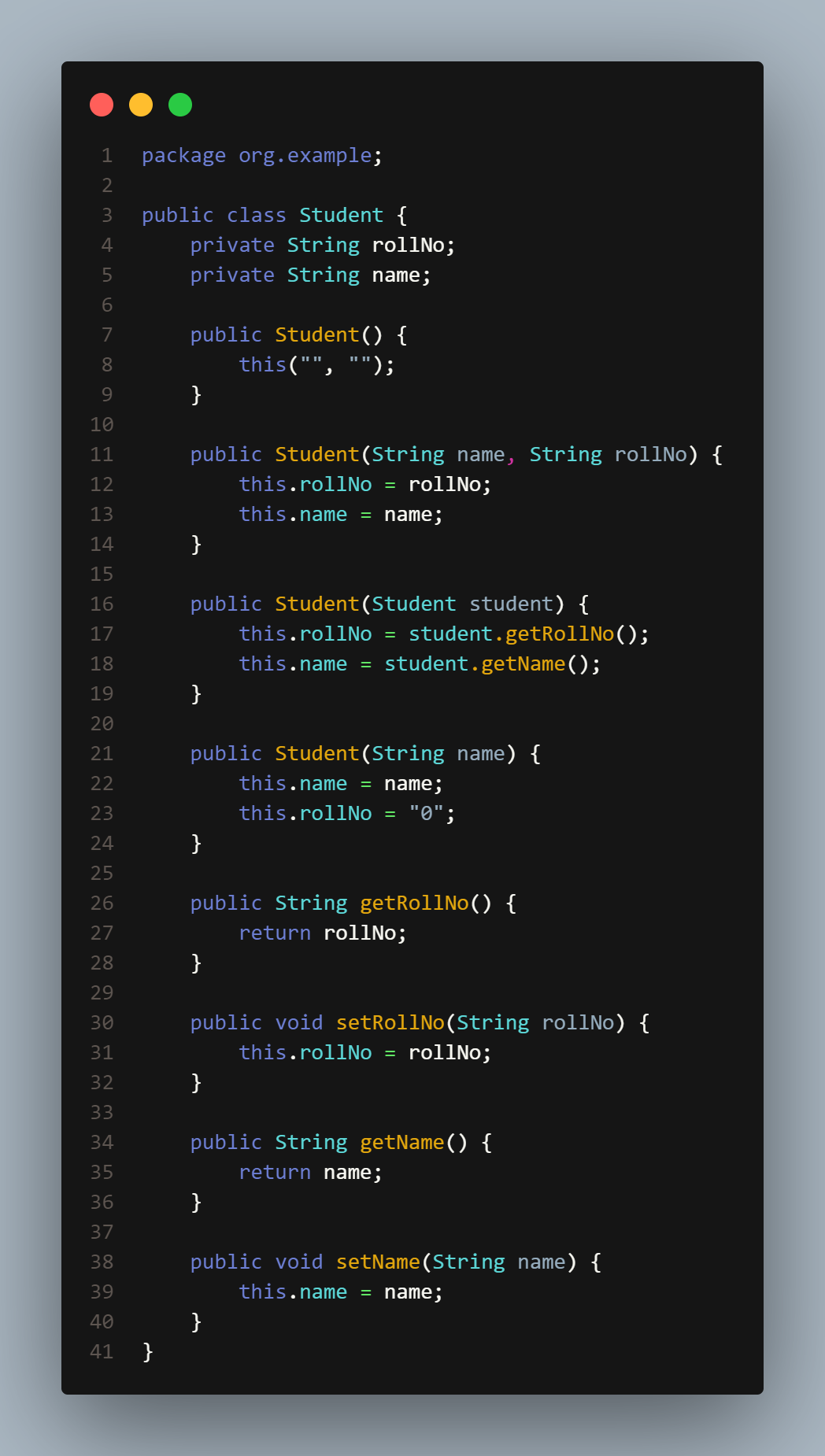
Luego se muestra la parte donde se añade un nuevo dato:



Y luego donde se actualiza el primer dato que está en la lista:



1. **Se hizo el modelo en base al dado, únicamente se adaptando, añadiendo tres nuevos Constructores**



**El modelo Student tiene dos atributos:** rollNo (número de matrícula) y name (nombre), con getters y setters correspondientes.

**Incluye tres constructores:**

* Constructor por defecto: Student().
* Constructor parametrizado: Student(String name, String rollNo).
* Constructor de copia: Student(Student student).

**Problema detectado:** En el constructor de copia, hay un error donde this.name = student.getRollNo() debería ser this.name = student.getName(). Esto causa que el nombre del estudiante se establezca incorrectamente como el número de matrícula.

**En ejecución,** el Controlador usa el Modelo para crear/actualizar estudiantes, y la Vista los muestra.

1. **Para el view se creó un método de inserción, el cual simula cómo sería la inserción tradicional**



El método inputStudent() está diseñado para simular la inserción de un estudiante, pero utiliza valores quemados en lugar de solicitar entrada del usuario.

Una "inserción tradicional" implica típicamente que el usuario ingrese datos interactivamente, por ejemplo, a través de la consola usando Scanner.

El método imprime los valores como si fueran ingresados, pero no permite interacción real, lo cual no cumple completamente con el espíritu de una inserción tradicional.

**En ejecución,** el Controlador llama a este método para añadir un estudiante al Modelo y actualizar la Vista, manteniendo la separación entre presentación y datos.

1. **El controlador se cambió en su mayoría, haciendo uso únicamente de los métodos que se requieren para hacer un intermediario entre el modelo, y la vista**



El controlador actúa como intermediario entre el modelo (StudentDatabase) y la vista (StudentView).

**Métodos como fetchStudents(), createStudent(), y updateStudent() coordinan las operaciones:**

* fetchStudents(): Obtiene los datos del modelo y los muestra en la vista.
* createStudent(): Solicita datos a la vista, los guarda en el modelo y actualiza la vista.
* updateStudent(): Actualiza un estudiante en el modelo y muestra los cambios en la vista.

El controlador no realiza lógica de almacenamiento ni de presentación directamente, cumpliendo con su rol de intermediario.

**En ejecución,** muestra la lista inicial, añade un estudiante y actualiza el primero, coordinando sin almacenar ni mostrar directamente.

1. **Se creó una clase que simula una base de datos, la cual brinda apoyo en la administración de los datos quemados.**



La clase utiliza el patrón singleton para garantizar una única instancia de la base de datos.

Inicializa una lista con 5 estudiantes predefinidos, cumpliendo con el requisito de "datos quemados".

Proporciona métodos CRUD (getStudents(), postStudent(), putStudent(), deleteStudent()) para administrar los datos.

La implementación es adecuada para simular una base de datos simple en memoria.

**En ejecución,** el Controlador usa estos métodos para obtener, añadir o actualizar estudiantes, y la Vista muestra los cambios, manteniendo el Modelo aislado de la presentación.

**Ejecución del código/ Pantallas**

