GUIA # 4

Taller De Programación

Daniel Sebastián Ramos Moreno
Andrés Rojas Reyes
Manuel Feo

PREGUNTAS ORIENTADORAS

1. ¿Dónde presento mayor dificultad resolviendo la guía? y ¿cómo lo resolvieron? ¿cuáles fueron las estrategias de solución?

¿Dónde presentamos mayor dificultad resolviendo la guía?

La mayor dificultad que enfrentamos al resolver la guía fue entender cómo aplicar el patrón de diseño Modelo-Vista-Controlador (MVC) en la biblioteca Swing de Java. Específicamente la división sobre esta tecnología los puntos en los que nos enfocamos debido a su dificultad fueron:

a. **Separación de Responsabilidades**: Distinguir claramente las responsabilidades del Modelo, la Vista y el Controlador en el contexto de Swing. La integración de estos componentes con los eventos y la actualización de la interfaz de usuario nos resultó especialmente desafiante.
b. **Comunicación entre Componentes**: Implementar la comunicación entre el Modelo, la Vista y el Controlador sin que los componentes se acoplen demasiado. Encontramos dificultades para asegurarnos de que las actualizaciones en la Vista reflejaran correctamente los cambios en el Modelo y viceversa.
c. **Sincronización de Datos**: Manejar la sincronización de datos entre los componentes del MVC y asegurarnos de que la Vista se actualizara correctamente en respuesta a los cambios en el Modelo, además de garantizar que las acciones del usuario se transmitieran adecuadamente al Modelo a través del Controlador.

¿Cómo lo resolvimos?

Para superar estas dificultades, implementamos varias estrategias:

a. **Investigación Adicional**: Buscamos y revisamos recursos adicionales como videos tutoriales y artículos especializados que explicaban el patrón MVC en el contexto de Swing. Estos materiales nos proporcionaron ejemplos prácticos y explicaciones claras sobre cómo dividir las responsabilidades entre los componentes.
b. **Ejemplos Prácticos**: Implementamos ejemplos simples para entender cómo funciona la interacción entre el Modelo, la Vista y el Controlador. Estos ejemplos

nos ayudaron a visualizar cómo cada componente debería comunicarse y actualizarse.

c. **Lectura de Documentación y Foros**: Consultamos la documentación oficial de Swing y participamos en foros en línea donde discutimos nuestras dudas con otros desarrolladores que tenían experiencia con MVC en Swing.

d. **Pruebas y Ajustes**: Realizamos pruebas iterativas y ajustes en nuestro código. Creamos prototipos y ajustamos la implementación según los resultados que obteníamos, lo que nos permitió corregir errores y mejorar la integración de los componentes.

e. **Trabajo en Equipo**: Dividimos las tareas según las fortalezas de cada miembro del grupo. Algunos se enfocaron en la implementación del Modelo y la Vista, mientras que otros se centraron en el Controlador y la sincronización de eventos. Esto nos permitió abordar las dificultades de manera más eficiente y aprender colaborativamente.

2. ¿Cuáles fueron los aprendizajes obtenidos al realizar esta guía?, liste como mínimo aprendizajes y relaciónelos con su futuro que hacer profesional.

- Al realizar una guía utilizando el patrón de diseño MVC (Modelo-Vista-Controlador con la biblioteca Swing en Java, se obtienen varios aprendizajes valiosos que pueden ser aplicados en nuestro futuro profesional.

1.**Separación de responsabilidades y modularidad**:

- **Aprendizaje**: La implementación del patrón MVC permite una clara separación de responsabilidades entre el modelo (datos y lógica de negocio), la vista (interfaz de usuario) y el controlador (gestión de eventos y coordinación entre modelo y vista). Esto facilita la modularidad y el mantenimiento del código.
- **Relación con el futuro profesional**: En cualquier desarrollo de software, la capacidad de diseñar sistemas con una buena separación de responsabilidades es crucial para crear aplicaciones robustas y escalables. Este aprendizaje te ayudará a diseñar sistemas más organizados y fáciles de

mantener en el futuro, lo que es altamente valorado en la industria del software.

2. **Gestión eficiente de eventos y actualización de la interfaz de usuario**:
    - **Aprendizaje**: La utilización de Swing para construir interfaces gráficas y la integración con el patrón MVC enseña cómo manejar eventos de usuario de manera eficiente y cómo actualizar la interfaz de usuario en respuesta a cambios en los datos sin acoplar demasiado la lógica del negocio con la presentación.
    - **Relación con el futuro profesional**: La habilidad para gestionar eventos de manera efectiva y actualizar interfaces de usuario dinámicamente es fundamental en el desarrollo de aplicaciones interactivas. Este conocimiento te permitirá construir aplicaciones con una experiencia de usuario más fluida y reactiva, que es esencial para cualquier desarrollador de front-end o diseñador de interfaces.
3. **Trabajo con bibliotecas y herramientas específicas**:
    - **Aprendizaje**: Trabajar con la biblioteca Swing te familiariza con herramientas y componentes específicos para la creación de interfaces gráficas en Java. Aprendes a usar componentes como botones, paneles, y cuadros de texto, así como a aplicar diseños y manejar layouts.
    - **Relación con el futuro profesional**: La experiencia con bibliotecas y herramientas específicas como Swing es una base importante para trabajar con otros marcos y bibliotecas en el desarrollo de interfaces. En el futuro, podrás transferir habilidades similares a otras tecnologías y frameworks, lo que te permitirá adaptarte a diferentes entornos de desarrollo y tecnologías emergentes.

Actividad de Trabajo Autónomo

1. Diferencia entre ODBC y JDBC

Al buscar las diferencias de ODBC (Open Database Connectivity) y JDBC (Java Database Connectivity), es importante entender que ambos son tecnologías diseñadas para facilitar la conexión y la interacción con bases de datos, pero están orientadas a diferentes lenguajes de programación entornos y metodologías las principales diferencias las mostramos a continuacion.

Lenguaje y Plataforma:

- **ODBC**: Se usa principalmente con lenguajes como C y C++, y es común en sistemas Windows. Permite conectar diferentes tipos de bases de datos usando un controlador ODBC.
- **JDBC**: Está diseñado específicamente para Java. Permite que aplicaciones Java se conecten a bases de datos de manera directa.

Cómo Funciona:

- **ODBC**: Usa un "driver" ODBC que actúa como un traductor entre la aplicación y la base de datos.
- **JDBC**: Usa "drivers JDBC" que están diseñados para Java y permiten a las aplicaciones Java comunicarse con la base de datos.

Uso y Configuración:

- **ODBC**: Puede ser un poco más complejo de configurar y usar, especialmente si trabajas en diferentes sistemas operativos.
- **JDBC**: Es más fácil de usar para desarrolladores Java porque está hecho para integrarse bien con el entorno Java.

2. Diferencias y Similitudes entre base de datos relacional y bases de datos NoSQL

Las bases de datos son fundamentales para la gestión de datos, y existen principalmente dos tipos: relacionales y NoSQL. Las bases de datos relacionales utilizan tablas con un esquema fijo y ofrecen un sólido soporte para transacciones. En contraste, las bases de datos NoSQL son más flexibles, permitiendo diferentes modelos de datos y escalabilidad horizontal para grandes volúmenes de información.

Diferencias:

1. **Estructura de Datos**:
    - **Relacional**: Usa tablas con filas y columnas. Cada dato se organiza en registros con un esquema fijo (definido por las tablas).
    - **NoSQL**: Ofrece varias formas de almacenar datos, como documentos, pares clave-valor, grafos o columnas. No requiere un esquema fijo.
2. **Escalabilidad**:
    - **Relacional**: Generalmente escala verticalmente, lo que significa que se mejora el rendimiento añadiendo más recursos a un solo servidor.
    - **NoSQL**: Diseñado para escalar horizontalmente, agregando más servidores para manejar grandes volúmenes de datos y tráfico.
3. **Transacciones**:
    - **Relacional**: Soporta transacciones ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), lo que garantiza integridad y precisión en operaciones complejas.
    - **NoSQL**: No siempre ofrece el mismo nivel de consistencia o transacciones ACID, priorizando la disponibilidad y la escalabilidad sobre la consistencia estricta.
4. **Consultas**:
    - **Relacional**: Utiliza SQL (Structured Query Language) para consultas estructuradas y complejas.

- **NoSQL**: Utiliza diferentes métodos de consulta según el tipo de base de datos (por ejemplo, JSON en MongoDB, o lenguajes específicos en bases de datos de grafos).

Similitudes:

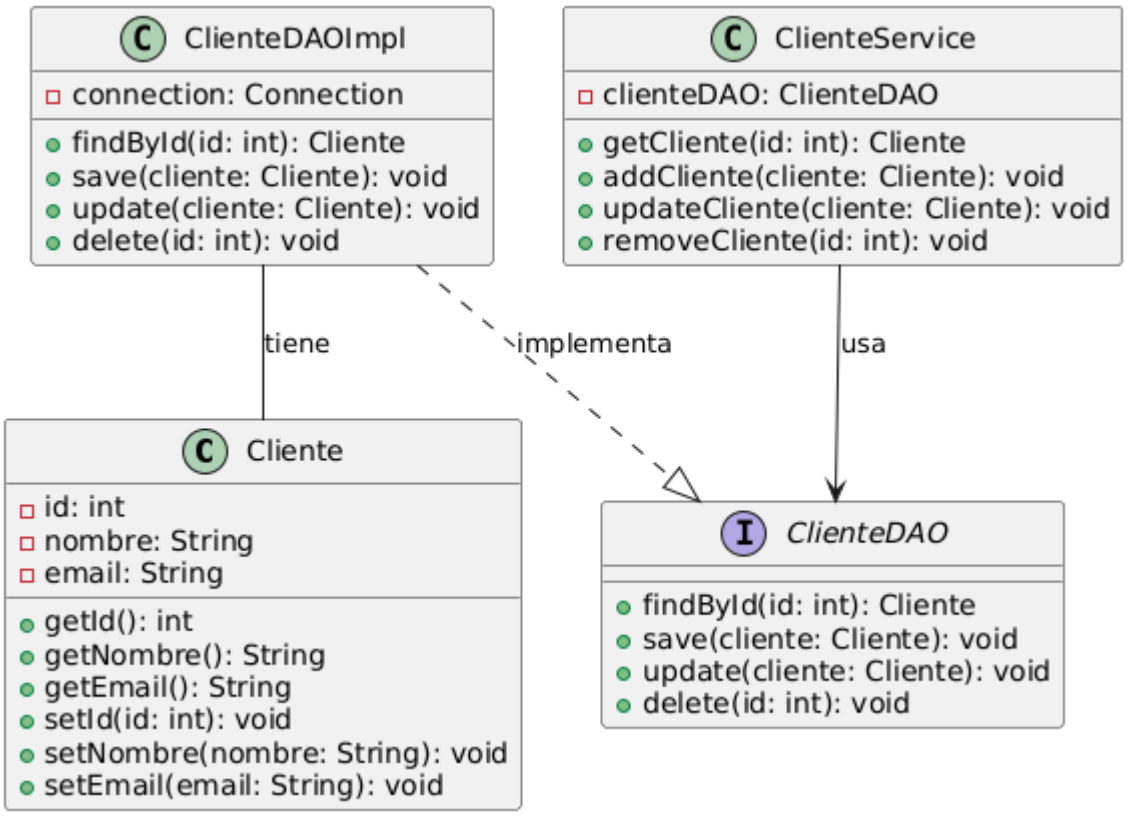
- **Almacenamiento de Datos**:
  - Ambas tipos de bases de datos se utilizan para almacenar y gestionar datos, pero lo hacen de maneras diferentes según sus diseños y necesidades.
- **Indices**:
  - Tanto las bases de datos relacionales como las NoSQL pueden usar índices para acelerar las consultas y mejorar el rendimiento.
- **Persistencia**:
  - Ambas garantizan la persistencia de datos, es decir, los datos se guardan de manera duradera para que no se pierdan tras el cierre de la aplicación.

3. Definir Patrón Data Access Object (DAO), explique mediante un diagrama de UML

Definición:

Patrón Data Access Object (DAO): Es un patrón de diseño que define una interfaz para las operaciones de acceso a datos y proporciona una implementación concreta para estas operaciones. El DAO actúa como un intermediario entre la aplicación y la base de datos, permitiendo que la lógica de negocio interactúe con los datos sin conocer los detalles de cómo están almacenados o recuperados.

Diagrama UML



4. Definir patrones Modelo Vista Controlador (MVC) y Data Transfer Object (DTO)

patrón Modelo Vista Controlador (MVC)

El patrón **Modelo Vista Controlador (MVC)** es un patrón de diseño que separa una aplicación en tres componentes principales:

1. **Modelo**: Representa los datos y la lógica de negocio. El modelo gestiona la información, la lógica de los datos y las reglas de negocio. No sabe nada acerca de la vista ni del controlador.
2. **Vista**: Es la interfaz de usuario. La vista muestra la información del modelo al usuario y envía las acciones del usuario al controlador. La vista se actualiza cuando el modelo cambia.
3. **Controlador**: Actúa como intermediario entre el modelo y la vista. Recibe las entradas del usuario desde la vista, procesa estas entradas (a menudo actualizando el modelo) y luego actualiza la vista según sea necesario.

Patrón Data Transfer Object (DTO)

El patrón **Data Transfer Object (DTO)** es un patrón de diseño que se utiliza para transferir datos entre diferentes capas de una aplicación o entre sistemas de manera eficiente. El DTO es un objeto que solo contiene datos y no tiene lógica de negocio. Su propósito principal es encapsular los datos y transportarlos a través de la red o entre diferentes partes de una aplicación.

Características del DTO:

1. **Sin Lógica de Negocio**: El DTO solo contiene atributos y métodos de acceso a esos atributos (getters y setters). No debe contener lógica de negocio o métodos complejos.
2. **Encapsulación de Datos**: Agrupa los datos relacionados en un solo objeto para simplificar el transporte de información.
3. **Optimización de la Comunicación**: Reduce la cantidad de llamadas entre capas o sistemas al agrupar datos en un solo objeto.

5. Defina que es JSON y para que se utiliza

¿Qué es JSON?

JSON (JavaScript Object Notation) es un formato de texto ligero para intercambiar datos estructurados. Se utiliza para representar datos en forma de pares clave-valor y listas ordenadas, y es fácil de leer y escribir para humanos y máquinas.

¿Para qué se utiliza JSON?

1. **Intercambio de Datos**: Principalmente para enviar datos entre un servidor y una aplicación web.
2. **Configuración**: Para almacenar configuraciones de aplicaciones.

3. **Almacenamiento**: En bases de datos NoSQL como MongoDB.
4. **APIs**: Para comunicar datos en servicios web.