¿Qué es JDBC?

> Estándar de conectividad que nos permite conectar programas en Java con bases de datos.
>
> API de JDBC

Beneficios de JDBC

- Creación de consultas y declaraciones SQL personalizadas
- Soporte a múltiples bases de datos
- Portabilidad

¿Qué nos va a permitir la API JDBC?

- Establecer conexión con la base de datos
- Enviar consultas
- Recibir resultados

Arquitectura de la API JDBC

Tiene la responsabilidad de **comunicarse con los usuarios** y utilizar las funciones proporcionadas por la API de JDBC.

JDBC Driver

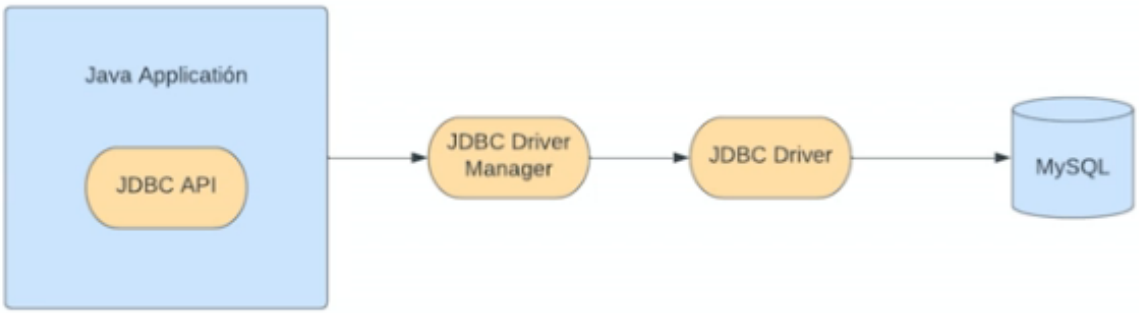
Actúa directamente con la base de datos y es la responsable de **establecer conexión, enviar consultas y recibir resultados.**

JDBC Driver Manager

Responsable de manejar los drivers JDBC disponibles y seleccionar el adecuado para **establecer la conexión con la base de datos.**

Origen de datos

Representa la **base de datos** con la que se comunica la aplicación Java mediante JDBC



Del JDBC driver podrías salir varias bases de datos , como ORACLE, etc.

Componentes:

JDBC Connection

Carga lo necesario para **establecer conexión:**

- URL de la base de datos
- Usuario
- Contraseña

JDBC Statement

Se utiliza para **enviar consultas SQL** simples sin parámetros.

JDBC Prepared Statement

Nos permite **crear consultas SQL** con parámetros variables.

JDBC ResultSet

Para obtener los resultados de una consulta SQL a una base de datos.

execute Query()

Para ejecutar una consulta SQL y obtener los resultados

next()

Avanzar al siguiente registro en un conjunto de resultados

close()

Cerrar una conexión con la base de datos.

executed Update()

Ejecutar sentencia SQL que modifica los datos en la base de datos.

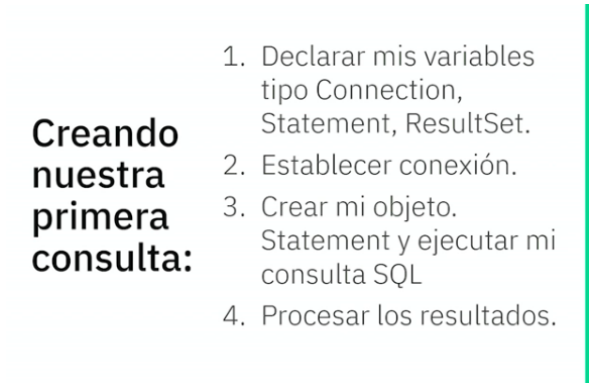
get()

Para obtener valores de los registros recuperados de la base de datos, existen varios métodos.

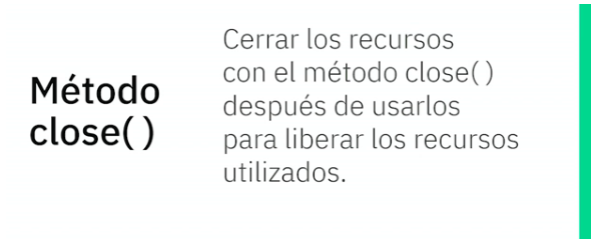
get()

- getInt()
- getString()
- getDouble()
- getDate()
- getTime()
- getTimestamp()

Crear y Actualizar



Manejo de Recursos
Optimización de recursos.



Es una buena práctica cerrar los recursos con el método `close()` después de usarlos para liberar los recursos utilizados. Esto se aplica a la programación JDBC con Java, así como a cualquier otra situación en la que se utilicen recursos del sistema, como archivos o sockets. Una forma común de hacerlo es usando el bloque `finally` para asegurarnos de que los recursos se cierren incluso si se produce una excepción en el código.

Otra forma introducida en Java 7 es usar la característica "Try with resources" que implementan la interfaz `AutoCloseable` o su subinterfaz `Closeable`. Antes de la introducción del "try con recursos", era necesario cerrar manualmente los recursos abiertos utilizando bloques `finally` para asegurarse de que se liberaran correctamente, incluso en caso de excepciones. Sin embargo, esto puede ser propenso a errores y aumenta la cantidad de código necesario. Al utilizar esta estructura, los recursos declarados dentro del bloque `try` se cierran automáticamente al finalizar el bloque, ya sea que se haya producido una excepción o no.

Si tienes una clase que implementa la interfaz `AutoCloseable` o `Closeable`, no es necesario agregar manualmente la interfaz `AutoCloseable` en el código.

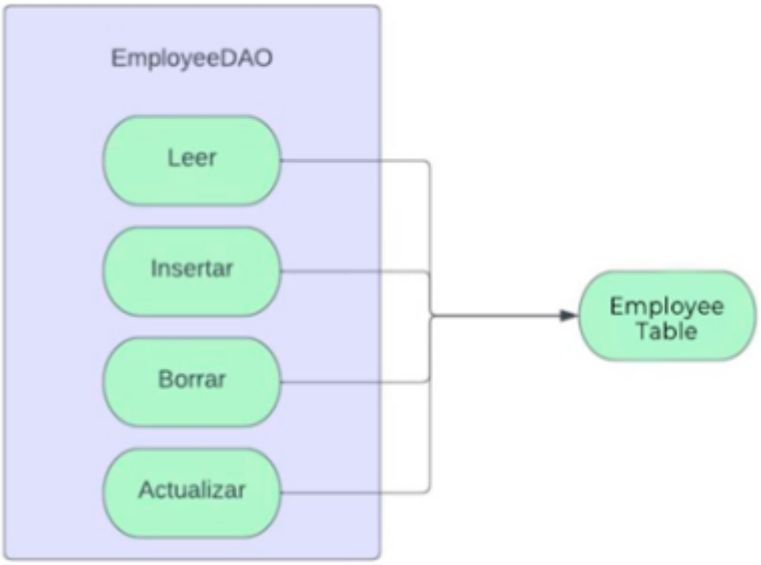
PATRÓN DAO Y REPOSITORY:

Patrón DAO (Data Access Object)

- Separa la lógica de acceso a datos de la lógica de negocios.
- El DAO proporcionará los métodos necesarios para insertar, actualizar, borrar y consultar la información.

Patrón DAO (Data Access Object)

El patrón DAO está más ligado a **guardar datos de una tabla muy concreta** de una base de datos SQL.

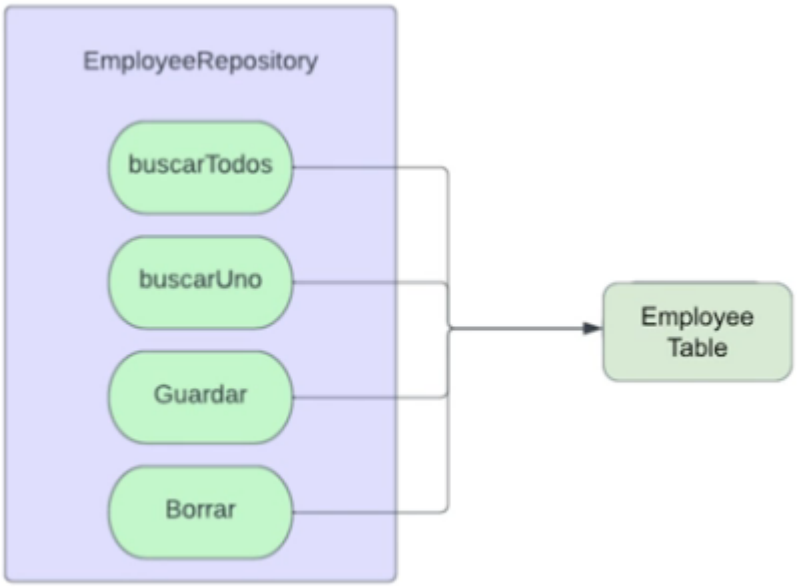


Patrón Repository:

- Abstraer y encapsular el acceso a datos.
- Proporcionar una capa de negocio y la capa de persistencia de datos.

Patrón Repository:

El patrón repositorio está más ligado a guardar una colección de objetos sin importar mucho el sistema de persistencia de destino.



El patrón DAO (Data Access Object) y el patrón Repository son dos patrones de diseño utilizados en el desarrollo de aplicaciones para separar la lógica de acceso a datos de la lógica de negocio.

Patrón DAO:

- Se centra en proporcionar una capa de abstracción entre la capa de negocio y la capa de acceso a datos.
- El DAO define una interfaz o una clase abstracta que encapsula la lógica de acceso a datos.
- Proporciona métodos para crear, leer, actualizar y eliminar (CRUD).
- El DAO oculta los detalles específicos de implementación del almacenamiento de datos y proporciona una interfaz coherente para que la capa de negocio interactúe con los datos.
- Permite centralizar y reutilizar la lógica de acceso a datos en toda la aplicación.

Patrón Repository:

- Se centra en separar la lógica de acceso a datos (capa de persistencia de datos) de la lógica de negocio.
- El Repository define una interfaz o una clase abstracta que proporciona métodos para recuperar y almacenar.
- Proporciona métodos más específicos para leer, guardar y borrar.

- El Repository oculta los detalles específicos de almacenamiento y consulta, y proporciona una abstracción para interactuar con los objetos.
- Permite un enfoque más orientado a objetos para trabajar con los datos, tratando los objetos como colecciones en lugar de registros aislados.

Podemos concluir que ambos patrones tienen como objetivo separar la lógica de acceso a datos de la lógica de negocio, nos proporcionan una abstracción para interactuar con los datos y ocultan los detalles específicos de almacenamiento y consulta, además de que nos facilitan la reutilización y centralización de la lógica de acceso a datos.

TRANSACCIONES:

¿QUE ES UN POOL?

Pool de conexiones

Conjunto de conexiones de base de datos que se crean y administran para su reutilización.

Pool de conexiones

En lugar de establecer una nueva conexión cada vez que se necesita acceder a la base de datos, se utilizan conexiones previamente creadas y almacenadas en el pool.

Beneficios

- Mejora de rendimiento
- Ahorro de recursos
- Tiempo de respuesta más rápido
- Control y gestión de conexiones
- Escalabilidad

IMPLEMENTANDO UN POOL

Métodos de configuración:

1. `setInitialSize(int)`: **Establece el tamaño inicial del pool de conexiones.**
2. `setMaxTotal(int)`: **Establece el número máximo de conexiones totales que el pool puede mantener activas simultáneamente.**
3. `setMaxIdle(int)`: **Establece el número máximo de conexiones inactivas que el pool puede mantener en todo momento.**
4. `setMinIdle(int)`: **Establece el número mínimo de conexiones inactivas que el pool debe mantener en todo momento.**
5. `setMaxWaitMillis(long)`: **Establece el tiempo máximo (en milisegundos) que una solicitud de conexión puede esperar antes de lanzar una excepción por tiempo de espera.**

Métodos de configuración

- setInitialSize(int)
- setMaxTotal(int)
- setMaxIdle(int)
- setMinIdle(int)
- setMaxWaitMillis(long)

¿QUÉ ES JPA Y ORM?

JPA: PERSISTENCIA DE DATOS

> Java Persistence API (JPA) es una especificación estándar de Java que define una interfaz para el manejo de la persistencia de datos en aplicaciones Java.

Mapeo de Objeto - Relacional

- Sustituye sentencias SQL por código Java.
- Podemos definir clases y métodos que actúan como los datos y operaciones.
- Hace el mantenimiento más fácil y eficiente.

@Entity	Marca una clase Java como una entidad que será mapeada a una tabla en la base de datos.
@Table	Especifica el nombre de la tabla en la base de datos que se corresponde con la entidad.
@Id	Marca una propiedad como la clave primaria de la entidad.
@Generated Value	Especifica cómo se generará automáticamente el valor de la clave primaria.
@Column	Permite especificar el mapeo de una propiedad a una columna de la tabla.

@OneTo Many	Establece una relación uno a muchos entre entidades.
@Many ToOne	Establece una relación muchos a uno entre entidades.
@Join Column	Especifica la columna de la clave foránea utilizada para la relación entre entidades.

¿QUE ES HIBERNATE?

Hibernate es una herramienta de mapeo objeto-relacional para Java, que proporciona una capa de abstracción entre una base de datos relacional y las entidades de la aplicación. Facilita el desarrollo de aplicaciones persistentes al eliminar la necesidad de escribir consultas SQL directamente, permitiendo a los desarrolladores interactuar con la base de datos a través de objetos y consultas en lenguaje Java.

Además de las anotaciones estándar de JPA (Java Persistence API), Hibernate ofrece sus propias anotaciones para personalizar el mapeo y el comportamiento de las entidades. Estas anotaciones brindan un mayor control sobre cómo se realiza el mapeo de objetos a tablas de base de datos y permiten definir características específicas de Hibernate.

Algunas de las anotaciones proporcionadas por Hibernate incluyen:

- **@Cascade: Se utiliza para especificar el comportamiento de cascada en las operaciones CRUD, permitiendo que las operaciones realizadas en una entidad se propaguen a las entidades asociadas.**
- **@Fetch: Permite controlar la estrategia de recuperación de datos, especificando cómo se deben cargar y recuperar los datos asociados a una entidad.**
- **@Formula: Permite definir fórmulas SQL personalizadas en atributos calculados, donde el valor del atributo se calcula mediante una expresión SQL.**

Estas anotaciones son solo algunos ejemplos de las muchas opciones que Hibernate proporciona para personalizar el mapeo y el comportamiento de las entidades.

Puedes encontrar más información y documentación detallada sobre Hibernate en el sitio web oficial: https://hibernate.org/

Además, aquí hay algunos enlaces útiles para aprender más sobre Hibernate:

- **Hibernate User Guide: https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html**
- **Hibernate Annotations Reference Guide: https://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single/**

Explorar estos recursos te ayudará a comprender mejor Hibernate y aprovechar al máximo sus capacidades para el desarrollo de aplicaciones persistentes en Java.