Clase grabada

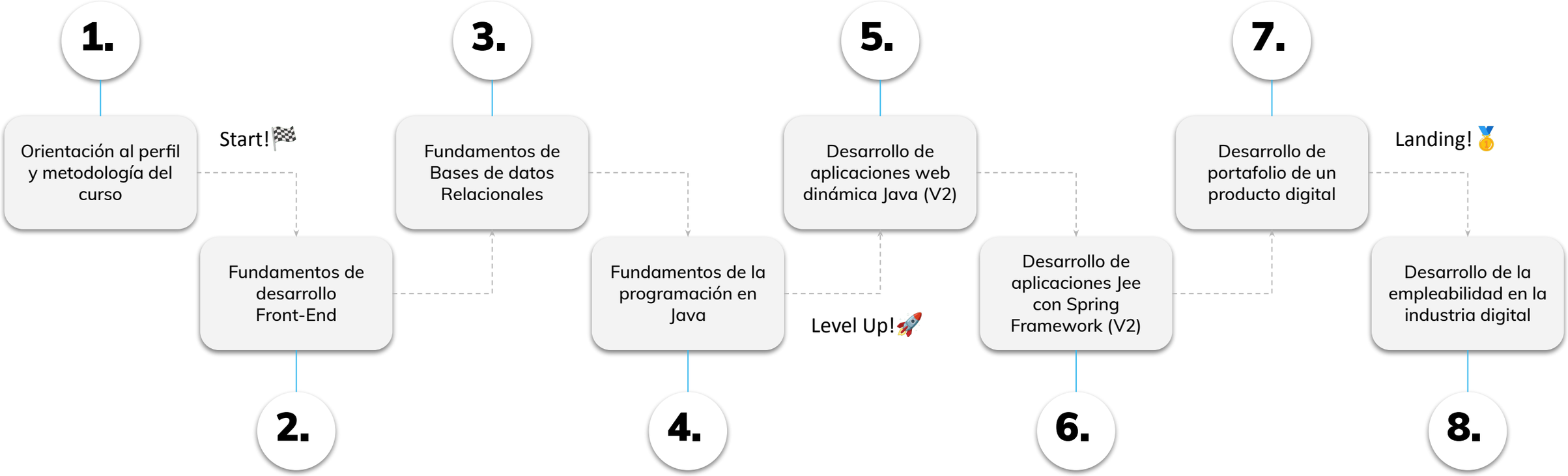
# Acceso a datos en Spring Framework

**Plan formativo:** Desarrollo de Aplicaciones Full Stack Java Trainee V2.0

English Always | alkemy

# HOJA DE RUTA

¿Cuáles **skill** conforman el programa?




English Always | alkemy

# REPASO CLASE ANTERIOR

En la clase anterior trabajamos 📚:

- ✓ *Acceso a datos con JDBC Templates*
- ✓ *Configuración de datasource en Spring*

# LEARNING PATHWAY

¿Sobre qué temas trabajaremos?

**6.3** Start! 🏁

**Acceso a datos en Spring Framework** → JDBCTemplate DAO → Manipulación de datos

Creando un DAO que utiliza JdbcTemplate
Realizando querys que reciben parámetros
Mapeando los resultados de una consulta a objetos
Modificando datos con JdbcTemplate
Invocando un DAO desde un servicio

English Always | alkemy

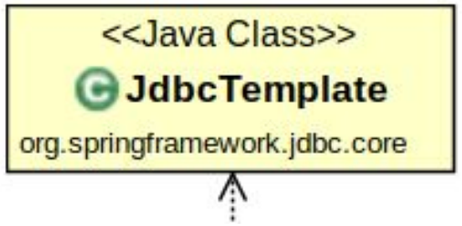
# OBJETIVOS DE APRENDIZAJE

¿Qué aprenderemos?

- *Manipular la capa de acceso a datos de JDBC Template con una clase DAO en Spring*

# DAO con JDBC Template

# DAO con JDBC Template




Crear un **DAO** (Data Access Object) implica **interactuar con una base de datos**. Al integrar **Spring** y su módulo **JdbcTemplate**, los desarrolladores pueden aprovechar una poderosa combinación para simplificar y eficientizar la interacción con bases de datos relacionales en aplicaciones Java.

# DAO con JDBC Template

El DAO actúa como una capa intermedia entre la aplicación y la base de datos. Proporciona una interfaz de alto nivel para acceder y manipular datos, abstrayendo los detalles específicos de la base de datos. La integración de JdbcTemplate con Spring facilita la gestión de la conexión, las transacciones y la ejecución de consultas SQL, eliminando la necesidad de manejar manualmente tareas repetitivas y propensas a errores.

# DAO con JDBC Template

**JdbcTemplate** es una clase proporcionada por el módulo JDBC de Spring que simplifica la ejecución de operaciones de base de datos utilizando JDBC puro. Elimina la necesidad de manejar manualmente la creación y liberación de recursos de conexión, gestionando estos aspectos de manera transparente. Además, simplifica la escritura de código al proporcionar métodos convenientes para ejecutar consultas y actualizar datos.

# DAO con JDBC Template

**Configuración de la Base de Datos:**

En una aplicación Spring con JdbcTemplate, la configuración de la base de datos se realiza generalmente en un archivo de configuración, ya sea mediante anotaciones o mediante un archivo XML. La configuración incluye detalles como la fuente de datos (**DataSource**), la cual puede ser configurada para utilizar un pool de conexiones, y el objeto JdbcTemplate que se basará en esta fuente de datos.

# DAO con JDBC Template

## Configuración de la Base de Datos

En nuestro caso utilizaremos la base de datos MySQL, por lo cual agregaremos la dependencia necesaria en el archivo pom.xml

```
<!-- Dependencia MySQL JDBC Driver -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.23</version> <!-- Ajusta la versión según tu entorno -->
</dependency>
```

# DAO con JDBC Template

**Luego, debemos configurar el DataSource en el archivo application.properties para MySQL.**

```
# Configuración de la base de datos MySQL
spring.datasource.url=jdbc:mysql://localhost:3306/nombre_de_tu_base_de_datos
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.username=tu_usuario
spring.datasource.password=tu_contraseña
spring.jpa.hibernate.ddl-auto=update
```

# DAO con JDBC Template

Los datos del application.properties deben coincidir con la configuración de la clase anotada con @Configuration.

```java
@Configuration
public class ConfiguracionBD {
    @Bean
        public DataSource dataSource() {
            DriverManagerDataSource dataSource = new DriverManagerDataSource();
            dataSource.setDriverClassName("com.mysql.cj.jdbc.Driver");
            dataSource.setUrl("jdbc:mysql://localhost:3306/tu_basedatos");
            dataSource.setUsername("usuario");
            dataSource.setPassword("contraseña");
            return dataSource;
        }
```

# DAO con JDBC Template

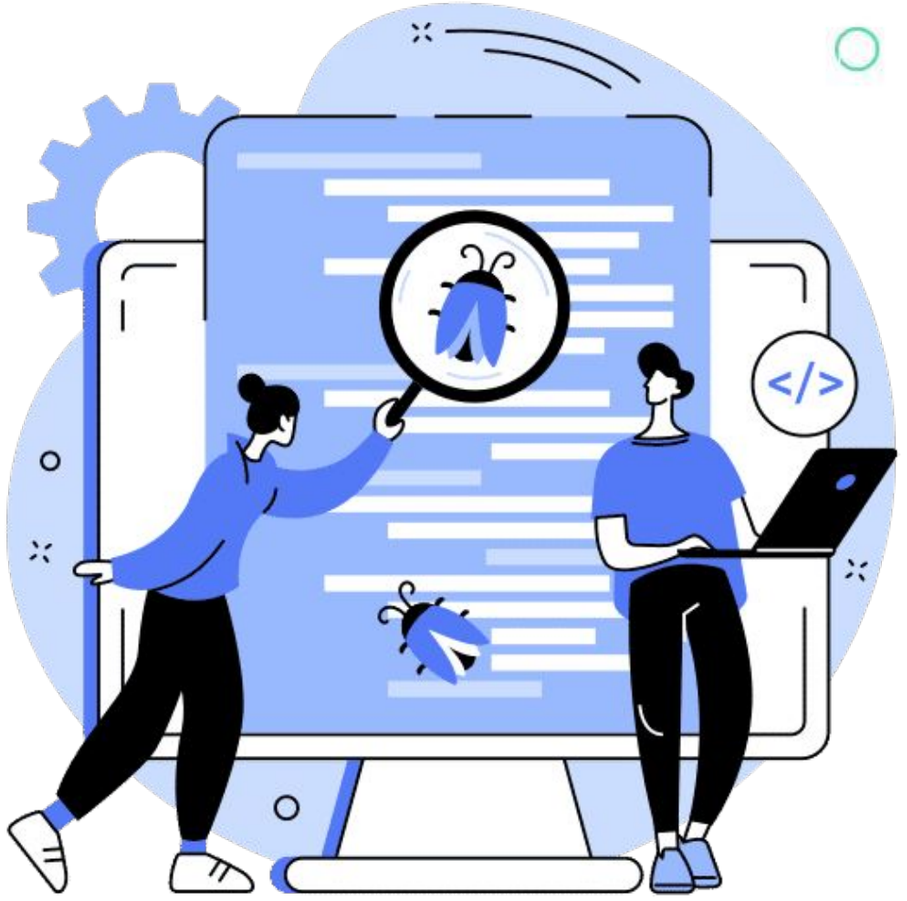
Finalmente creamos una clase DAO para manipular la conexión y consultas a la base de datos. En este ejemplo creamos la clase UsuarioDAO con el atributo jdbcTemplate.

```
public class UsuarioDao {

    @Autowired
    private final JdbcTemplate jdbcTemplate;

    public UsuarioDao(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }
```

# Creando Querys con parámetros

# Creando Querys que reciben parámetros

Realizar consultas que involucran parámetros es una parte esencial de cualquier aplicación de acceso a datos. Con JdbcTemplate, la creación y ejecución de consultas parametrizadas es sencilla y segura. Veamos un ejemplo de cómo realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) utilizando JdbcTemplate.



# Creando Querys que reciben parámetros

**1. Crear Registros con Parámetros:**

Para la operación de creación, supongamos que queremos agregar un método a nuestro DAO para insertar un nuevo usuario en la tabla. Vamos a utilizar un método que acepta un objeto Usuario como parámetro.

```
public void insertarUsuario(Usuario usuario) {
    String sql = "INSERT INTO usuario (id, nombre, email) VALUES (?, ?, ?)";
    jdbcTemplate.update(sql, usuario.getId(), usuario.getNombre(), usuario.getEmail());
}
```

Aquí, el método **insertarUsuario** utiliza el método **update** de JdbcTemplate, que acepta la consulta SQL y los valores de los parámetros.

# Creando Querys que reciben parámetros

**2. Actualizar Registros con Parámetros:**

Si queremos permitir la actualización de la información del usuario, podríamos implementar un método como el siguiente:

```
public void actualizarUsuario(Usuario usuario) {
    String sql = "UPDATE usuario SET nombre = ?, email = ? WHERE id = ?";
    jdbcTemplate.update(sql, usuario.getNombre(), usuario.getEmail(), usuario.getId());
}
```

Este método utiliza una consulta UPDATE parametrizada para **actualizar** la información de un usuario específico.

# Creando Querys que reciben parámetros

**3- Eliminar Registros con Parámetros:**

Finalmente, para la operación de eliminación, podríamos tener un método que elimine un usuario según su ID.

```
public void eliminarUsuarioPorId(String id) {
    String sql = "DELETE FROM usuario WHERE id = ?";
    jdbcTemplate.update(sql, id);
}
```

Este método utiliza una consulta DELETE parametrizada para **eliminar** un usuario **según su ID**.

# Mapeando los resultados de una consulta a objetos

# Mapeando los resultados de una consulta a objetos

Mapear los resultados de una consulta a objetos es una parte clave del proceso de acceso a datos en cualquier aplicación. En el contexto de JdbcTemplate en Spring, este proceso se simplifica mediante el uso de **RowMappers**.

En JdbcTemplate, un **RowMapper** es una interfaz funcional que define el método mapRow, responsable de mapear un conjunto de resultados (un registro de la base de datos) a un objeto Java.

# Mapeando los resultados de una consulta a objetos

Veamos un ejemplo de una clase exclusiva para el mapeo de objetos. En esta implementación, creamos una clase separada llamada **UsuarioRowMapper** que **implementa la interfaz RowMapper.** La interfaz RowMapper requiere la implementación del método mapRow, que realiza el mapeo de la fila del conjunto de resultados a un objeto Usuario.

```
public class UsuarioRowMapper implements RowMapper<Usuario> {

    @Override
    public Usuario mapRow(ResultSet resultSet, int rowNum) throws SQLException {
        Usuario usuario = new Usuario();
        usuario.setId(resultSet.getString("id"));
        usuario.setNombre(resultSet.getString("nombre"));
        usuario.setEmail(resultSet.getString("email"));
        return usuario;
    }
}
```

# Mapeando los resultados de una consulta a objetos

Finalmente, en el DAO, utilizamos la instancia de **UsuarioRowMapper** como **argumento en el método query**. Este enfoque resulta en un código más explícito y es fácil de entender para aquellos que están comenzando con Java y Spring. Además, es útil si deseas realizar operaciones más avanzadas durante el mapeo.

```
public List<Usuario> buscarUsuariosPorNombre(String nombre) {
    String sql = "SELECT * FROM usuario WHERE nombre = ?";
    return jdbcTemplate.query(sql, new Object[]{nombre}, new UsuarioRowMapper());
}
```

# Mapeando los resultados de una consulta a objetos

También es posible realizar el mapeo de resultados **sin una clase RowMapper separada**, especialmente para aplicaciones más simples. Puedes utilizar una instancia anónima de RowMapper directamente en el método query.

```java
public List<Usuario> buscarUsuariosPorNombre(String nombre) {
    String sql = "SELECT * FROM usuario WHERE nombre = ?";

    return jdbcTemplate.query(sql, new Object[]{nombre}, new RowMapper<Usuario>() {
        @Override
        public Usuario mapRow(ResultSet resultSet, int rowNum) throws SQLException {
            Usuario usuario = new Usuario();
            usuario.setId(resultSet.getString("id"));
            usuario.setNombre(resultSet.getString("nombre"));
            usuario.setEmail(resultSet.getString("email"));
            return usuario;
        }
    });
}
```

# Mapeando los resultados de una consulta a objetos

En este ejemplo, hemos creado una instancia anónima de RowMapper directamente en el método query. La implementación de RowMapper está contenida dentro de la declaración de la instancia anónima.

Este enfoque es una alternativa válida cuando se desea mantener una estructura más tradicional. Ambos enfoques (usar clases separadas o instancias anónimas) son aceptados en Java y pueden adaptarse según las preferencias de codificación del equipo y la complejidad del código.

# Modificando datos

Como ya hemos visto, si queremos modificar datos con JdbcTemplate en Spring se realiza comúnmente utilizando el método update. Este método permite ejecutar consultas SQL que realizan operaciones de modificación en la base de datos, como INSERT, UPDATE y DELETE.
En el caso de querer modificar solo un atributo del registro, podemos recibirlo a través de los parámetros. Por ejemplo, este método actualiza el campo email de un usuario identificado por su ID con el nuevo email proporcionado.

```
public void actualizarEmail(String id, String nuevoEmail) {
    String sql = "UPDATE usuario SET email = ? WHERE id = ?";
    jdbcTemplate.update(sql, nuevoEmail, id);
}
```

# Invocando un DAO desde el servicio

# Invocando un DAO desde el Servicio

Invocar un DAO desde un servicio es una práctica común en la arquitectura de aplicaciones en Spring. El servicio actúa como una capa intermedia que encapsula la lógica de negocio, y el DAO se encarga de las operaciones de acceso a datos.

Supongamos que tienes un DAO para la entidad Usuario como se mencionó en ejemplos anteriores. Para poder utilizarlo, es necesario crear una instancia única del DAO en la clase de servicio para así acceder a todos sus métodos,

```java
@Service
public class UsuarioService {

    private final UsuarioDao usuarioDao;

    @Autowired
    public UsuarioService(UsuarioDao usuarioDao) {
        this.usuarioDao = usuarioDao;
    }

    public List<Usuario> obtenerTodosLosUsuarios() {
        return usuarioDao.obtenerTodosLosUsuarios();
    }

    public void guardarUsuario(Usuario usuario) {
        usuarioDao.guardarUsuario(usuario);
    }
```

# Evaluación Integradora ✨

¿Listos para un nuevo desafío? En esta clase comenzamos a construir nuestro....

## Trabajo Integrador del Módulo 💪🏽

Iremos completándolo progresivamente clase a clase.

English Always | alkemy

# LIVE CODING

Ejemplo en vivo

**Completando la Wallet:**

- *Es hora de completar nuestro proyecto de billetera virtual. Para esto, vamos a agregar la clase CuentaDAO y UsuarioDAO.*

- *Además, vamos a crear en los DAO los métodos necesarios para el CRUD de cada correspondiente entidad.*

**Tiempo: 30 minutos**

# ¿Alguna consulta?

## Ejercicio

# Manipulación de datos

# Manipulación de datos

**Contexto:** 🙌

Vamos a continuar con el ejercicio de la clase anterior. En este caso vamos a crear una clase DAO que maneje la interacción con la base de datos configurada en el datasource.

**Consigna:** ✍️

Crear un DAO que realice el CRUD de una entidad determinada,

Puedes continuar con la generación del código de manera asíncrona. Recuerda traer todas las dudas la próxima clase.

**Tiempo🕐:** *15 minutos*

# RESUMEN

¿Qué logramos en esta clase?

- ✓ *Comprender la creación de clases DAO para manipulación de datos*
- ✓ *Aprender a invocar los DAO desde un servicio*

English Always | alkemy

# #WorkingTime

Continuemos ejercitando

**¡Antes de cerrar la clase!** Te invitamos a: 👇👇👇

1. Repasar nuevamente la grabación de esta clase
2. Revisar el material compartido en la plataforma de Moodle (lo que se vio en clase y algún ejercicio adicional)
   a. *Lectura Modulo 6, Lección 3: páginas 5 - 12*
3. Traer al próximo encuentro, todas tus dudas y consultas para verlas antes de iniciar nuevo tema.

# ¡Muchas Gracias!

Nos vemos en la próxima clase 👋🏻

English Always | alkemy