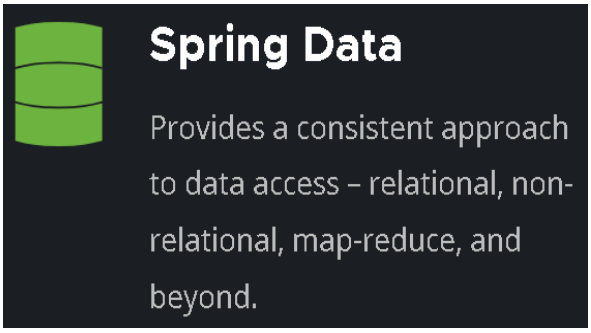
Framework Spring JPA

Índice

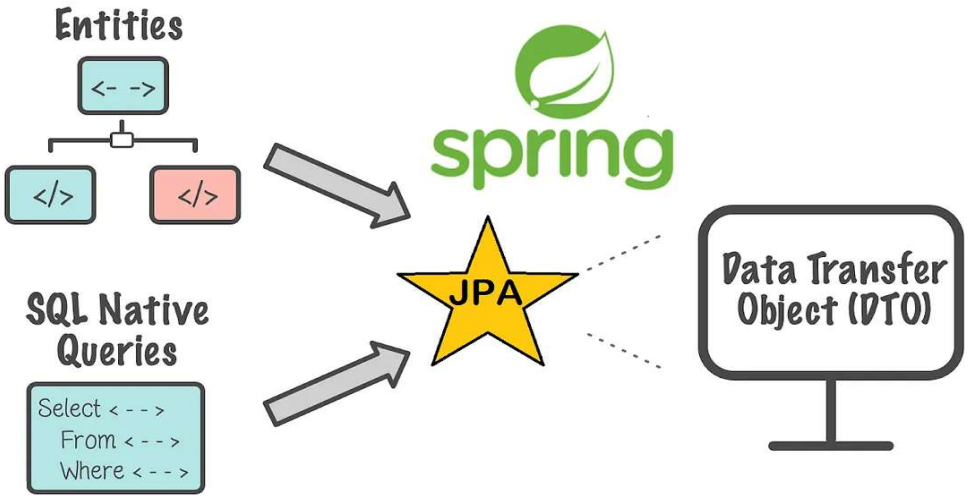
1. Spring JPA: Acceso a datos.

El módulo Spring Data es el encargado de vincular el framework Spring con bases de datos. Existe un gran número de subproyectos dentro del módulo Spring Data:



Spring Data JDBC	Spring Data for Apache Cassandra
Spring Data JPA	Spring Data for Apache Geode
Spring Data LDAP	Spring Data for Apache Solr
Spring Data MongoDB	Spring Data for VMware GemFire
Spring Data Redis	Spring Data Couchbase
Spring Data R2DBC	Spring Data Elasticsearch
Spring Data REST	Spring Data Envers
Spring Data Neo4j	Spring for Apache Hadoop

Spring JPA es la propuesta estándar que ofrece Java para implementar un Framework Object Relational Mapping (ORM), que permite interactuar con la base de datos por medio de objetos.



Spring proporciona básicamente dos ventajas a la hora de dar soporte a nuestros DAOs (Objetos de Acceso a Datos):

• Simplifica las operaciones de acceso a datos en APIs tediosos de utilizar como JDBC, proporcionando una capa de abstracción que reduce la necesidad de código repetitivo. Para ello se usan los denominados templates, que son clases que implementan este código, permitiendo que nos concentremos en la parte "interesante".

• Define una rica jerarquía de excepciones que modelan todos los problemas que nos podemos encontrar al operar con la base de datos, y que son independientes del API empleado.

Un template de acceso a datos en Spring es una clase que encapsula los detalles más tediosos (como por ejemplo la necesidad de abrir y cerrar la conexión con la base de datos en JDBC), permitiendo que nos ocupemos únicamente de la parte de código que hace realmente la tarea (inserción, consulta, …)

En el siguiente vídeo puedes ver las diferencias entre JDBC, JPA, ORMs, Hibernate: https://www.youtube.com/watch?v=z1CBVKexENc

1.1. Creación de un proyecto usando BBDD H2.

H2 es una base de datos relacional que se encuentra escrita en Java y funciona como una base de datos en memoria, cuyo uno de sus puntos fuertes es que puede trabajar como una base de datos embebida en aplicaciones Java o ejecutarse en modo cliente servidor (De manera rápida y sencilla).

Es decir, que añadida en nuestras aplicaciones como una dependencia más y una vez configurada la conexión, nos va a permitir realizar pruebas y trabajar como si fuera una base de datos relacional.

Hay que tener en cuenta que es en memoria, por lo que nunca debería ser usada para entornos productivos.

En el siguiente ejemplo vas a crear una aplicación que almacena los POJO (objetos Java simples) de la clase Cliente (Customer) en una base de datos basada en memoria (H2):

1. Crea un proyecto Spring Starter Project y añade las siguientes dependencias: Spring Data JPA, H2 Database.
2. Crea una tabla (Entity) con 3 campos: id (Primary key autoincrementable), nombre, apellidos.

```
package com.example.accessingdatajpa;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
@Entity                                          //Así se definen las tablas: Annotation
public class Customer {
  @Id                                            //Así se define una clave primaria
  @GeneratedValue(strategy=GenerationType.AUTO) //Así se especifica que el campo es
autoincrementable
  private Long id;
  private String firstName;
  private String lastName;

  protected Customer() {}

  public Customer(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  @Override
  public String toString() {
    return String.format(
        "Customer[id=%d, firstName='%s', lastName='%s']",
        id, firstName, lastName);
  }

  public Long getId() {
    return id;
  }

  public String getFirstName() {
    return firstName;
  }

  public String getLastName() {
    return lastName;
  }
}
```

Habrás podido comprobar que la clase tiene 2 constructores. El constructor predeterminado existe sólo para el contexto de JPA. No lo utilizas directamente, por lo que está designado como protected. El otro constructor es el que vas a usar para crear instancias de Customer que se guardarán en la base de datos.

3. Crea consultas simples: Spring Data JPA se centra en utilizar JPA para almacenar datos en una base de datos relacional. Su característica más atractiva es la capacidad de crear implementaciones de repositorio automáticamente, en tiempo de ejecución, desde una interfaz de repositorio.

   Para ver cómo funciona esto, crea una interfaz de repositorio que funcione con entidades de Customer en la siguiente ruta (src/main/java/com/example/accessingdatajpa/CustomerRepository.java):

```
package com.example.accessingdatajpa;

import java.util.List;
import org.springframework.data.repository.CrudRepository;

public interface CustomerRepository extends CrudRepository<Customer, Long> {

  List<Customer> findByLastName(String lastName);
  Customer findById(long id);
}
```

   CustomerRepository extiende la interfaz de CrudRepository. El tipo de entidad y el ID con el que trabaja, Customer y Long, se especifican en los parámetros genéricos de CrudRepository. Al extender de CrudRepository, CustomerRepository hereda varios métodos para guardar, borrar y buscar Clientes (`findAll()`, `findById(valor)`, `findByLastName()`)

   En una aplicación Java típica, es posible que desee escribir una clase que implemente CustomerRepository. Sin embargo, eso es lo que hace que Spring Data JPA sea tan poderoso: no es necesario escribir una implementación de la interfaz del repositorio. Spring Data JPA crea una implementación cuando ejecuta la aplicación.

4. En la clase Main añadimos el código necesario para usar los métodos asociados a la interface creada en el paso anterior. La clase incluye un método demo() que usa el CustomerRepository para algunas pruebas:

- Primero, recupera CustomerRepository del contexto de la aplicación Spring.
- Crea varios objetos de tipo Customer, mostrando el uso del método save() y configurando algunos datos con los que trabajar.
- A continuación, llama a findAll() para recuperar todos los objetos de la clase Customer de la base de datos.
- Luego llama a findById() para buscar un único Customer por su ID.
- Finalmente, llama a findByLastName() para buscar todos los objetos de la clase Customer cuyo apellido es "Bauer".

  (El método demo() devuelve un bean CommandLineRunner que ejecuta automáticamente el código cuando se inicia la aplicación.)

```
package com.example.demo;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class Tema5EjemploSpringJpAconH2DatabaseApplication {
     private static final Logger log =
LoggerFactory.getLogger(Tema5EjemploSpringJpAconH2DatabaseApplication.class);

      public static void main(String[] args) {

      SpringApplication.run(Tema5EjemploSpringJpAconH2DatabaseApplication.class, args);
      }

            @Bean
            public CommandLineRunner demo(CustomerRepository repository) {
              return (args) -> {
                // save a few customers
                repository.save(new Customer("Jack", "Bauer"));
                repository.save(new Customer("Chloe", "O'Brian"));
                repository.save(new Customer("Kim", "Bauer"));
                repository.save(new Customer("David", "Palmer"));
                repository.save(new Customer("Michelle", "Dessler"));

                // fetch all customers
                log.info("Customers found with findAll():");
                log.info("-------------------------------");
                for (Customer customer : repository.findAll()) {
                  log.info(customer.toString());
                }
                log.info("");

                // fetch an individual customer by ID
```

```
            Customer customer = repository.findById(1L);
            log.info("Customer found with findById(1L):");
            log.info("--------------------------------");
            log.info(customer.toString());
            log.info("");

            // fetch customers by last name
            log.info("Customer found with findByLastName('Bauer'):");
            log.info("--------------------------------------------");
            repository.findByLastName("Bauer").forEach(bauer -> {
              log.info(bauer.toString());
            });
            // for (Customer bauer : repository.findByLastName("Bauer")) {
            //  log.info(bauer.toString());
            // }
            log.info("");
          };
        }

}
```

5. La salida por consola será:

[restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path "

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Started Tema5EjemploSpringJpAconH2DatabaseApplication in 10.337 seconds (process running for 11.945)

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Customers found with findAll():

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : -------------------------------

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Customer[id=1, firstName='Jack', lastName='Bauer']

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Customer[id=2, firstName='Chloe', lastName='O'Brian']

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Customer[id=3, firstName='Kim', lastName='Bauer']

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Customer[id=4, firstName='David', lastName='Palmer']

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Customer[id=5, firstName='Michelle', lastName='Dessler']

[restartedMain] EjemploSpringJpAconH2DatabaseApplication :

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Customer found with findById(1L):

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : -------------------------------

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Customer[id=1, firstName='Jack', lastName='Bauer']

[restartedMain] EjemploSpringJpAconH2DatabaseApplication :

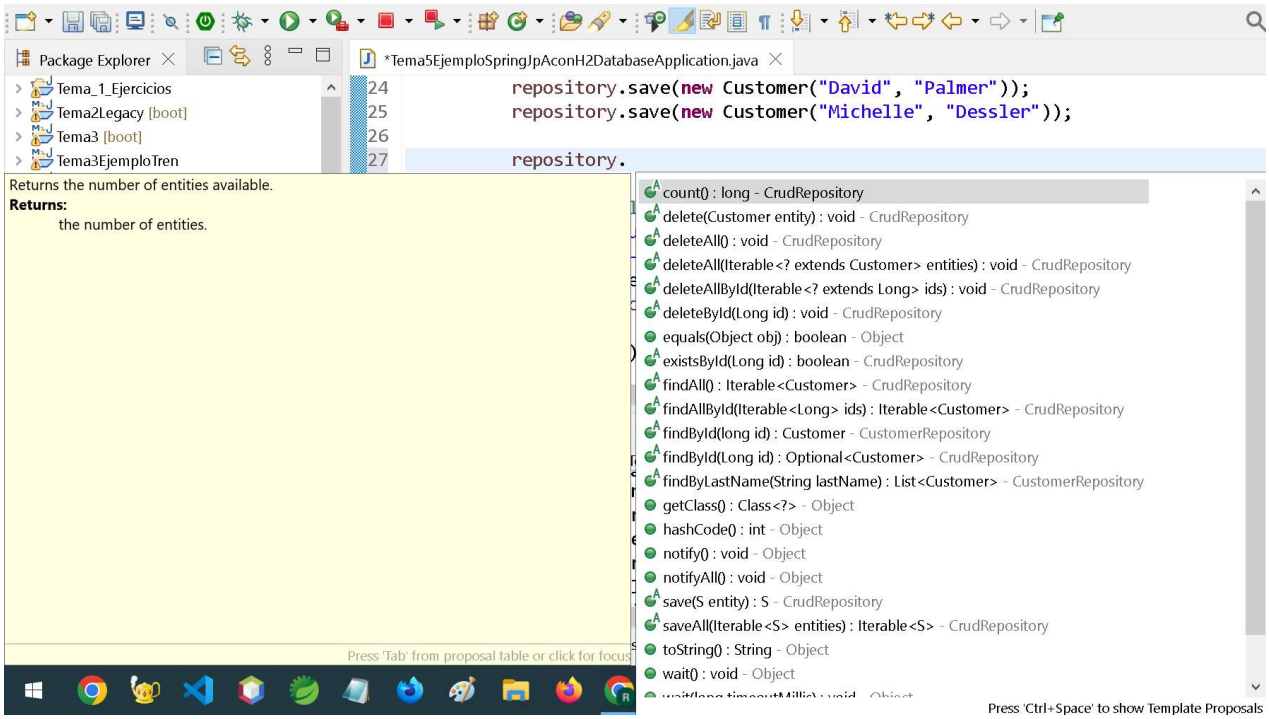
[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Customer found with findByLastName('Bauer'):

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : --

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Customer[id=1, firstName='Jack', lastName='Bauer']

[restartedMain] EjemploSpringJpAconH2DatabaseApplication : Customer[id=3, firstName='Kim', lastName='Bauer']

Repository nos permite usar múltiples métodos como los de la siguiente captura:



En el siguiente enlace puedes encontrar más información acerca del componente Repository:
https://gustavopeiretti.com/spring-boot-componente-repository/

La interface CrudRepository también tiene métodos asociados a bases de datos como:
https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html

1.2. Creación de un proyecto usando BBDD MySQL.

En el 1º curso de DAW has trabajado con BBDD que usan MySQL. El enlace de más abajo explica el proceso de creación de una aplicación Spring conectada a una base de datos MySQL (A diferencia de una base de datos integrada en memoria como H2, utilizada en la mayoría de guías y muchas aplicaciones de muestra).

BBDD MySQL utiliza Spring Data JPA para acceder a la base de datos, pero esta es solo una de muchas opciones posibles (por ejemplo, podría usarse también Spring JDBC). Crea un proyecto con las siguientes dependencias en POM.xml:

```
<dependencies>
                    <dependency>
                              <groupId>org.springframework.boot</groupId>
                              <artifactId>spring-boot-starter-data-jpa</artifactId>
                    </dependency>
                    <dependency>
                              <groupId>org.springframework.boot</groupId>
                              <artifactId>spring-boot-starter-thymeleaf</artifactId>
                    </dependency>
                    <dependency>
                              <groupId>org.springframework.boot</groupId>
                              <artifactId>spring-boot-starter-web</artifactId>
                    </dependency>

                    <dependency>
                              <groupId>org.springframework.boot</groupId>
                              <artifactId>spring-boot-devtools</artifactId>
                              <scope>runtime</scope>
                              <optional>true</optional>
                    </dependency>
                    <dependency>
                              <groupId>com.mysql</groupId>
                              <artifactId>mysql-connector-j</artifactId>
                              <scope>runtime</scope>
                    </dependency>
                    <dependency>
                              <groupId>org.springframework.boot</groupId>
                              <artifactId>spring-boot-starter-test</artifactId>
                              <scope>test</scope>
                    </dependency>
          </dependencies>
```

En el archivo application.properties debes añadir:

```
spring.application.name=Curso24_25_Tema_4BDMySQLApartado1_2
spring.datasource.url=jdbc:mysql://localhost:3306/trabajadores
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Habrás observado que estamos trabajando de la misma forma que el curso anterior. Mi configuración elegida ha sido: jdbc driver usando XAMPP (Módulo MySQL pulsando en opción admin). A partir de ahí, he puesto las credenciales (En mi caso username:root y sin contraseña). He creado una base de datos y una tabla llamadas trabajadores con los campos de la clase que actúa como tabla.

Vamos a realizar una prueba explicando todos los archivos (Clase Controller, 2 Interfaces, Clase que actúa como tabla, templates HTML usando Thymeleaf).

En la Moodle, puedes descargar todos esos archivos.

1.3. Ejercicios a realizar.Mejora el proyecto MVC con MySQL.

Ejercicio 1:

Modifica el ejercicio 4 del tema anterior (Tema 3: Creación de formularios con Spring MVC y Thymeleaf) para que los anuncios se guarden de forma persistente en una base de datos relacional MySQL. Usa Spring Data JPA según hemos visto en clase.

Ejercicio 2:

Mejora tu proyecto del tema anterior usando bases de datos y añadiendo en alguna clase 2 de las siguientes annotations y explica su funcionamiento:

@getter @setter @NoArgsConstructor @AllArgsConstructor