Práctica: Spring Boot + Hibernate/JPA + MySQL (Apache NetBeans)

0. Objetivo y resultado esperado

El alumno creará un proyecto Spring Boot que:

- Se conecta a MySQL (`pruebaspring`).
- Mapea la entidad `Empleado` a la tabla `empleados`.
- Inserta datos automáticamente al arrancar (DataLoader).
- Ejecuta consultas y muestra resultados en consola (QueryRunner).

Salida en consola esperada: listados de empleados y filtros por departamento/salario.

1. Preparación de la base de datos en MySQL

1.1 Crear BD y usuario (si no existen)

En MySQL Workbench o consola:

```
CREATE DATABASE IF NOT EXISTS pruebaspring
  CHARACTER SET utf8mb4
  COLLATE utf8mb4_0900_ai_ci;

CREATE USER IF NOT EXISTS 'alumno'@'%' IDENTIFIED BY 'alumno123';
GRANT ALL PRIVILEGES ON pruebaspring.* TO 'alumno'@'%';
FLUSH PRIVILEGES;

USE pruebaspring;
SHOW TABLES;   -- debe salir vac o inicialmente
```

1.2 Si la BD ya existía y no es utf8mb4 (opcional)

```
ALTER DATABASE pruebaspring
  CHARACTER SET = utf8mb4
  COLLATE = utf8mb4_0900_ai_ci;
```

Para convertir una tabla existente (p.ej., `empleados`):

```
ALTER TABLE empleados
  CONVERT TO CHARACTER SET utf8mb4
  COLLATE utf8mb4_0900_ai_ci;
```

2. Crear proyecto Spring Boot en NetBeans (con Maven)

2.1 Pasos exactos en NetBeans

1. **File** → **New Project...**
2. Categoría: **Java with Maven** → **Spring Boot Initializr Project** → **Next**.
3. En *Project Metadata*:
   - **Group**: `edu.dam.acceso`
   - **Artifact**: `tema7hibernate`
   - **Name**: `tema7hibernate`
4. En *Dependencies*, añade:
   - **Spring Data JPA**
   - **MySQL Driver**
   - (Opcional) **Spring Web** y **Lombok**
5. **Finish**. Espera a que Maven descargue dependencias.

3. Configurar `application.properties`

3.1 Ruta del archivo

Proyecto → `src/main/resources` → `application.properties`

3.2 Contenido (copiar tal cual)

```
# Puerto fijo recomendado (evitar conflictos con 8080)
server.port=8090
# Alternativa: puerto libre aleatorio
# server.port=0

# --- MySQL ---
spring.datasource.url=jdbc:mysql://localhost:3306/pruebaspring?
   useSSL=false&serverTimezone=UTC&useUnicode=true&characterEncoding
   =utf8
spring.datasource.username=alumno
spring.datasource.password=alumno123
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# --- JPA/Hibernate ---
spring.jpa.hibernate.ddl-auto=update
# Hibernate puede inferir el dialecto:
# spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.use_sql_comments=true

# Logs del SQL ejecutado y de los par metros
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

Nota (acentos/): en Windows, si la consola muestra , abrir **Project Properties** → **Run** → **VM Options** y añadir:

```
-Dfile.encoding=UTF-8
```

4. Crear el package de entidades y la clase `Empleado`

4.1 Crear package `domain`

En el árbol del proyecto:

1. **Source Packages** → **(clic derecho sobre)** `edu.dam.acceso.tema7hibernate`
2. **New** → **Java Package**
3. **Name**: `edu.dam.acceso.tema7hibernate.domain` → **Finish**

4.2 Crear clase `Empleado`

1. **Clic derecho** sobre `edu.dam.acceso.tema7hibernate.domain`
2. **New** → **Java Class**
3. **Class Name**: `Empleado` → **Finish**

4.3 Código de `Empleado.java` (con getters/setters)

```java
package edu.dam.acceso.tema7hibernate.domain;

import jakarta.persistence.*;

@Entity
@Table(name = "empleados")
public class Empleado {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable=false, length=60)
    private String nombre;

    @Column(nullable=false, length=40)
    private String departamento;

    @Column(nullable=false)
    private Double salario;

    public Empleado() { }

```

```
24      public Empleado(String nombre, String departamento, Double
            salario) {
25          this.nombre = nombre;
26          this.departamento = departamento;
27          this.salario = salario;
28      }
29
30      public Long getId() { return id; }
31      public String getNombre() { return nombre; }
32      public String getDepartamento() { return departamento; }
33      public Double getSalario() { return salario; }
34
35      public void setId(Long id) { this.id = id; }
36      public void setNombre(String nombre) { this.nombre = nombre; }
37      public void setDepartamento(String departamento) { this.
            departamento = departamento; }
38      public void setSalario(Double salario) { this.salario = salario;
            }
39  }
```

4.4 Verificación

- **Run** del proyecto (ver §8). En consola, con `ddl-auto=update`, debe aparecer el `CREATE TABLE empleados`.
- En MySQL:

```
1  USE pruebaspring;
2  SHOW TABLES;
3  DESCRIBE empleados;
```

5. Crear el package de repositorios y `EmpleadoRepository`

5.1 Crear package `repository`

1. **Clic derecho** sobre `edu.dam.acceso.tema7hibernate`
2. **New** → **Java Package**
3. **Name**: `edu.dam.acceso.tema7hibernate.repository` → **Finish**

5.2 Crear interfaz `EmpleadoRepository`

1. **Clic derecho** sobre `edu.dam.acceso.tema7hibernate.repository`
2. **New** → **Java Interface**
3. **Name**: `EmpleadoRepository` → **Finish**

5.3 Código de `EmpleadoRepository.java`

```
1  package edu.dam.acceso.tema7hibernate.repository;
2
```

```
import edu.dam.acceso.tema7hibernate.domain.Empleado;
import org.springframework.data.jpa.repository.JpaRepository;
import java.util.List;

public interface EmpleadoRepository extends JpaRepository<Empleado,
   Long> {
    List<Empleado> findByDepartamento(String departamento);
    List<Empleado> findBySalarioGreaterThan(Double salarioMin);
    List<Empleado> findByDepartamentoAndSalarioGreaterThan(String
       departamento, Double salarioMin);
}
```

5.4 Verificación

Al arrancar, la consola debe mostrar:

```
Bootstrapping Spring Data JPA repositories...
Finished Spring Data repository scanning in X ms. Found 1 JPA
   repository interface.
```

6. Crear package `bootstrap` y `DataLoader` (insertar datos)

6.1 Crear package `bootstrap`

1. **Clic derecho** sobre `edu.dam.acceso.tema7hibernate`
2. **New** → **Java Package**
3. **Name**: `edu.dam.acceso.tema7hibernate.bootstrap` → **Finish**

6.2 Crear clase `DataLoader`

1. **Clic derecho** sobre `edu.dam.acceso.tema7hibernate.bootstrap`
2. **New** → **Java Class**
3. **Name**: `DataLoader` → **Finish**

6.3 Código de `DataLoader.java`

```
package edu.dam.acceso.tema7hibernate.bootstrap;

import edu.dam.acceso.tema7hibernate.domain.Empleado;
import edu.dam.acceso.tema7hibernate.repository.EmpleadoRepository;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class DataLoader implements CommandLineRunner {

    private final EmpleadoRepository repo;
```

```
12
13      public DataLoader(EmpleadoRepository repo) {
14          this.repo = repo;
15      }
16
17      @Override
18      public void run(String... args) {
19          if (repo.count() == 0) {
20              repo.save(new Empleado("Ana Torres", "Ventas", 2200.0));
21              repo.save(new Empleado("Luis P rez", "IT", 2850.5));
22              repo.save(new Empleado("Marta D az", "IT", 3100.0));
23              repo.save(new Empleado("Sergio Le n", "Marketing",
                    1950.0));
24              System.out.println("    DataLoader: datos iniciales
                    insertados");
25          } else {
26              System.out.println("       DataLoader: ya hay datos, no
                    se insertan duplicados");
27          }
28      }
29  }
```

6.4 Verificación

- En consola: mensaje de `DataLoader` y `INSERT INTO empleados`.
- En MySQL:

```
1  USE pruebaspring;
2  SELECT * FROM empleados;
```

7. Crear `QueryRunner` (consultas y salida en consola)

7.1 Crear clase `QueryRunner`

1. **Clic derecho** sobre `edu.dam.acceso.tema7hibernate.bootstrap`
2. **New** → **Java Class**
3. **Name**: `QueryRunner` → **Finish**

7.2 Código de `QueryRunner.java`

```
1  package edu.dam.acceso.tema7hibernate.bootstrap;
2
3  import edu.dam.acceso.tema7hibernate.domain.Empleado;
4  import edu.dam.acceso.tema7hibernate.repository.EmpleadoRepository;
5  import org.springframework.boot.CommandLineRunner;
6  import org.springframework.stereotype.Component;
7
8  import java.util.List;
```

```
9
10 @Component
11 public class QueryRunner implements CommandLineRunner {
12
13     private final EmpleadoRepository repo;
14
15     public QueryRunner(EmpleadoRepository repo) {
16         this.repo = repo;
17     }
18
19     @Override
20     public void run(String... args) {
21
22         System.out.println("\n=== LISTA COMPLETA (findAll) ===");
23         List<Empleado> todos = repo.findAll();
24         todos.forEach(e ->
25             System.out.printf("%d | %s | %s | %.2f    %n",
26                 e.getId(), e.getNombre(), e.getDepartamento(), e.
                    getSalario())
27         );
28
29         System.out.println("\n=== POR DEPARTAMENTO: IT (
               findByDepartamento) ===");
30         repo.findByDepartamento("IT").forEach(e ->
31             System.out.printf("%s - %.2f    %n", e.getNombre(), e.
                   getSalario())
32         );
33
34         System.out.println("\n=== SALARIO > 2500 (
               findBySalarioGreaterThan) ===");
35         repo.findBySalarioGreaterThan(2500.0).forEach(e ->
36             System.out.printf("%s - %.2f    %n", e.getNombre(), e.
                   getSalario())
37         );
38
39         System.out.println("\n=== IT y SALARIO > 2500 (
               findByDepartamentoAndSalarioGreaterThan) ===");
40         repo.findByDepartamentoAndSalarioGreaterThan("IT", 2500.0).
               forEach(e ->
41             System.out.printf("%s - %.2f    %n", e.getNombre(), e.
                   getSalario())
42         );
43     }
44 }
```

7.3 Verificación (consola)

- Deben verse 4 bloques con resultados.
- Si aparecen en acentos/, aplicar UTF-8 (ver §3.2 nota).

8. Clase principal y ejecución

8.1 Ruta y nombre

Proyecto → `Source Packages` → `edu.dam.acceso.tema7hibernate` → `item`

`Tema7hibernateApplication.java`

8.2 Código (si no existe, crearlo)

```
package edu.dam.acceso.tema7hibernate;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Tema7hibernateApplication {
    public static void main(String[] args) {
        SpringApplication.run(Tema7hibernateApplication.class, args);
    }
}
```

8.3 Ejecutar

- **NetBeans:** botón **Run** del proyecto, o `Shift+F6` sobre la clase principal.
- **Maven (CMD):**

```
mvn spring-boot:run
# o fijando el puerto sin tocar properties:
mvn -Dspring-boot.run.arguments=--server.port=8090 spring-boot:run
```

9. Comprobaciones finales

- Consola: `Found 1 JPA repository interface`.
- Consola: `Tomcat started on port 8090 (http)` o el puerto configurado.
- Consola: mensajes de `DataLoader` y listados de `QueryRunner`.
- MySQL:

```
USE pruebaspring;
SELECT * FROM empleados;
```

10. Problemas frecuentes y solución

- **Puerto 8080 en uso:** en `application.properties` usar `server.port=8090` o liberar el 8080:

```
netstat -aon | findstr :8080
taskkill /PID <PID> /F
```

- **No detecta repositorios (0 interfaces):** el package `repository` debe colgar de `edu.dam.acceso.tema7hibernate`. Si no, añadir en la clase principal:

```
@org.springframework.data.jpa.repository.config.
   EnableJpaRepositories("edu.dam.acceso.tema7hibernate.
   repository")
@org.springframework.boot.autoconfigure.domain.EntityScan("edu.
   dam.acceso.tema7hibernate.domain")
```

- **Faltan getters en `Empleado`:** añadir `getId()`, `getNombre()`, `getDepartamento()`, `getSalario()`.
- **Acentos/ mal en consola:** VM Option `-Dfile.encoding=UTF-8` (ver §3.2).
- **No crea tabla:** revisar `@Entity`, `@Id`, y `spring.jpa.hibernate.ddl-auto=update`.