

Tema 2 – Spring Boot Acceso a datos: CRUD

Grado en Ingeniería Informática en Tecnologías de la Información

Departamento de Ingeniería de Sistemas Informáticos y Telemáticos

Área de Lenguajes y Sistemas Informáticos

Dr. Luis V. Calderita

Objetivos

- Crear una aplicación web con Spring Boot para acceder a una base de datos relacional.
 - Mapear una clase con una tabla en la base de datos.
 - Implementar el manejo de los datos mediante las operaciones CRUD
 - Crear objetos de esa clase y persistirlos en la base de datos
 - Consultar y recuperar objetos de la base de datos...

Recordatorios

- CRUD, Create Read Update and Delete
 - Operaciones típicas para las tablas de una base de datos
- ORM, HIBERNATE, JPA
 - Object-Relational Mapping técnica de mapeo entre objetos y tablas de una BD relacional
 - Java Persistence API, interfaz de persistencia propuesta por Java
 - Hibernate, proporciona una implementación de la JPA
- H2
 - Base de datos relacional que usaremos

Crear el proyecto de Spring Boot

- Crear el proyecto con <https://start.spring.io/> o Sprint Tool Suite
- Dependencias:
 - Spring Web, sirve para crear app-web con Spring
 - Spring Data JPA, sirve para crear repositorios de datos basados en JPA. Usa Hibernate
 - H2 Database, incluye soporte para la base de datos H2

New Spring Starter Project



Service URL	<input type="text" value="https://start.spring.io"/>		
Name	<input type="text" value="spring-data"/>		
<input checked="" type="checkbox"/> Use default location			
Location	<input type="text" value="C:\Users\alumno\spring-boot\spring-data"/>	<input type="button" value="Browse"/>	
Type:	<input type="text" value="Maven Project"/>	Packaging:	<input type="text" value="Jar"/>
Java Version:	<input type="text" value="17"/>	Language:	<input type="text" value="Java"/>
Group	<input type="text" value="com.example.data"/>		
Artifact	<input type="text" value="spring-data"/>		
Version	<input type="text" value="0.0.1-SNAPSHOT"/>		
Description	<input type="text" value="Demo project for Spring Boot DATA"/>		
Package	<input type="text" value="com.example.data"/>		
Working sets			
<input type="checkbox"/> Add project to working sets	<input type="button" value="New..."/>		
Working sets:	<input type="text"/>	<input type="button" value="Select..."/>	

New Spring Starter Project Dependencies



Spring Boot Version: 2.7.4

Frequently Used:

☒ Spring Web

Available:

- ▶ Developer Tools
- ▶ Google Cloud Platform
- ▶ I/O
- ▶ Messaging
- ▶ Microsoft Azure
- ▶ NoSQL
- ▶ Observability
- ▶ Ops
- ▶ SQL
- ▶ Security
- ▶ Spring Cloud
- ▶ Spring Cloud Circuit Breaker
- ▶ Spring Cloud Config
- ▶ Spring Cloud Discovery
- ▶ Spring Cloud Messaging
- ▶ Spring Cloud Routing

Selected:

- X Spring Data JPA
- X H2 Database
- X Spring Web

Make Default

Clear Selection



< Back

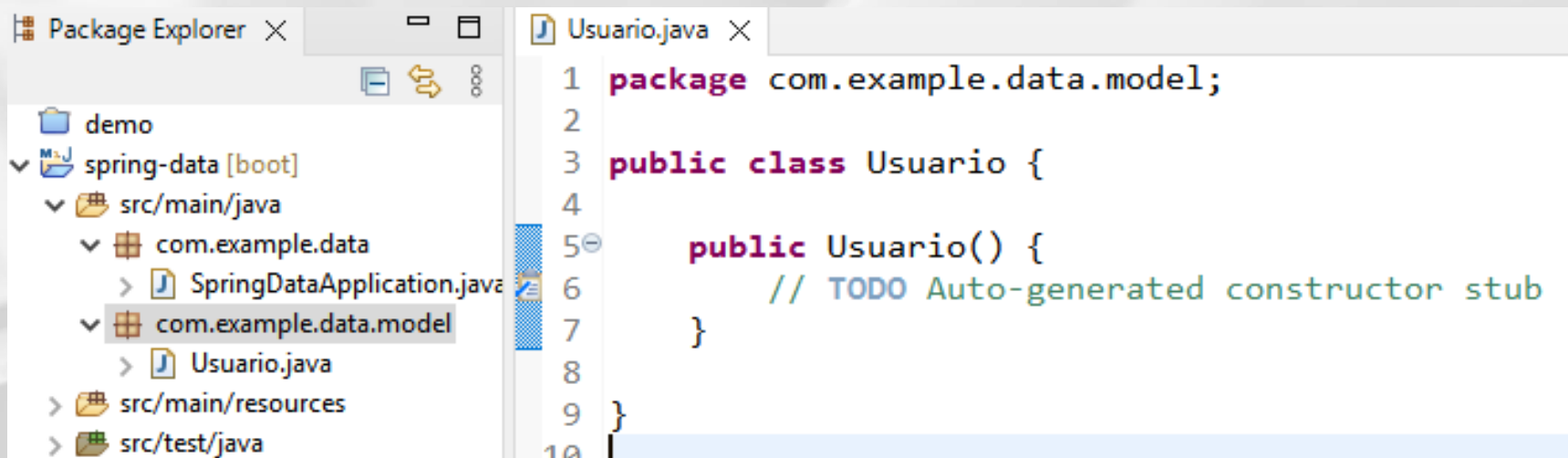
Next >

Finish

Cancel

Model, con clase Usuario

- Crear el subpaquete model, con la clase Usuario



Convirtiendo la clase Usuario en una Entidad

- Para que Spring sepa que ese objeto es una Entidad debemos anotarlo.
- **@Entity**, indica que esa clase es una entidad JPA.
- Una entidad JPA es un Plain Old Java Object que debe ser persistido en una tabla de la Base de Datos.
- Por defecto, Spring Data usa el nombre de la clase como nombre de la Entidad.

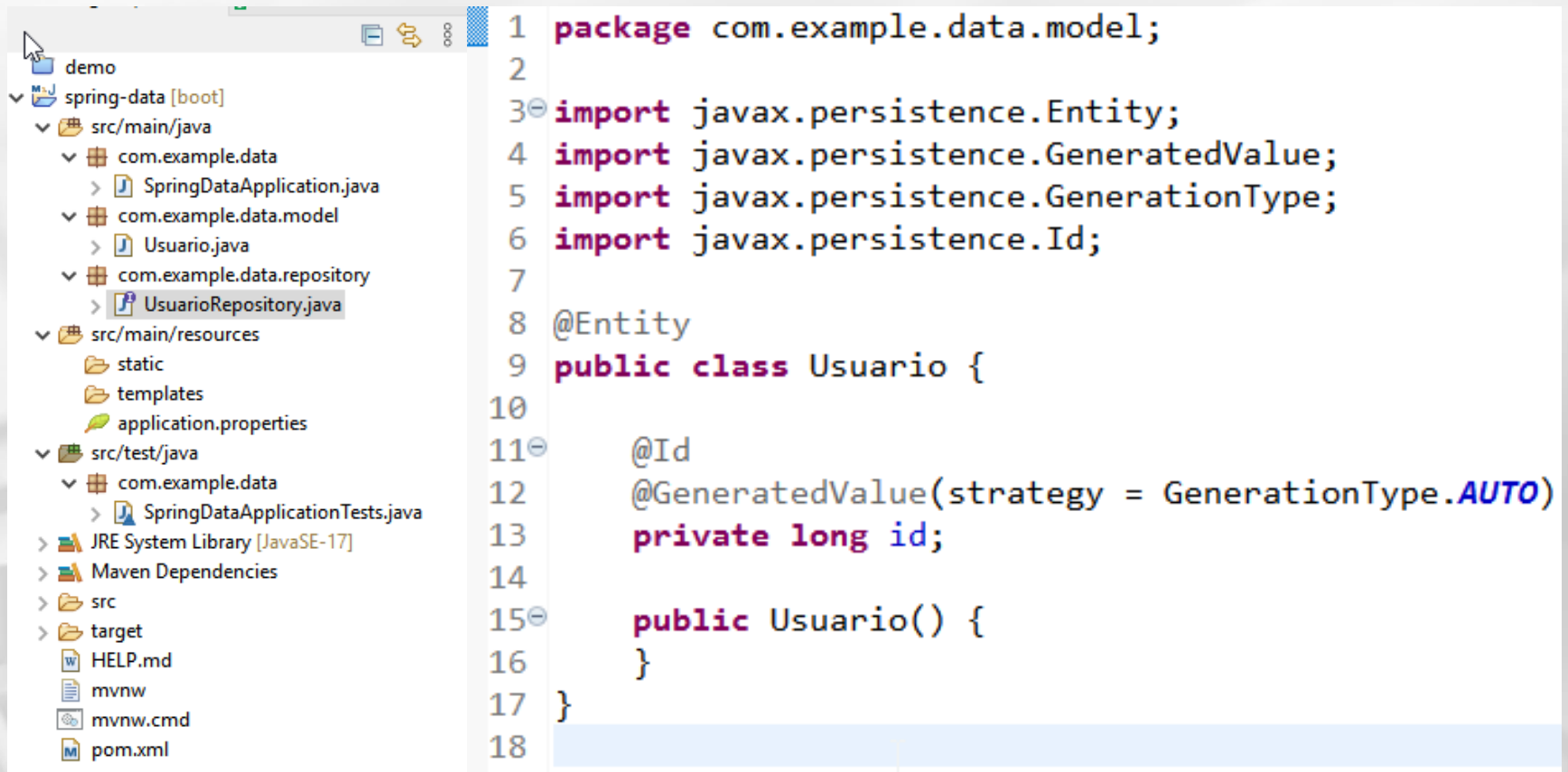
Convirtiendo la clase Usuario en una Entidad

- Una Tabla en una BD normalmente tiene una clave primaria. Su identificador único.
- Una Entidad requiere un *identificador* para identificar cada fila en la tabla de la base de datos subyacente.
- Por tanto, debo crear un atributo y anotarlo como su clave primaria.

Convirtiendo la clase Usuario en una Entidad

- Anotaciones:
 - @Id
 - Identifica la propiedad de la clase como clave primaria
 - @GeneratedValue (strategy = GenerationType.**AUTO**)
 - Define la estrategia para generar el valor de la propiedad
 - Esta opción permite al proveedor de persistencia determinar el esquema de generación del ID
 - El uso de la anotación GeneratedValue sólo es necesario para las claves primarias simples.
 - De momento, usamos el tipo de generación AUTO

Entidad Usuario



The image shows a screenshot of an IDE with a project structure on the left and a Java class definition on the right.

Project Structure (Left):

- demo
 - spring-data [boot]
 - src/main/java
 - com.example.data
 - SpringDataApplication.java
 - com.example.data.model
 - Usuario.java
 - com.example.data.repository
 - UsuarioRepository.java
 - src/main/resources
 - static
 - templates
 - application.properties
 - src/test/java
 - com.example.data
 - SpringDataApplicationTests.java
 - JRE System Library [JavaSE-17]
 - Maven Dependencies
 - src
 - target
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

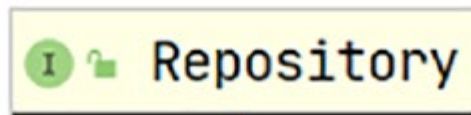
Java Class Definition (Right):

```
1 package com.example.data.model;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.GenerationType;
6 import javax.persistence.Id;
7
8 @Entity
9 public class Usuario {
10
11     @Id
12     @GeneratedValue(strategy = GenerationType.AUTO)
13     private long id;
14
15     public Usuario() {
16     }
17 }
18
```

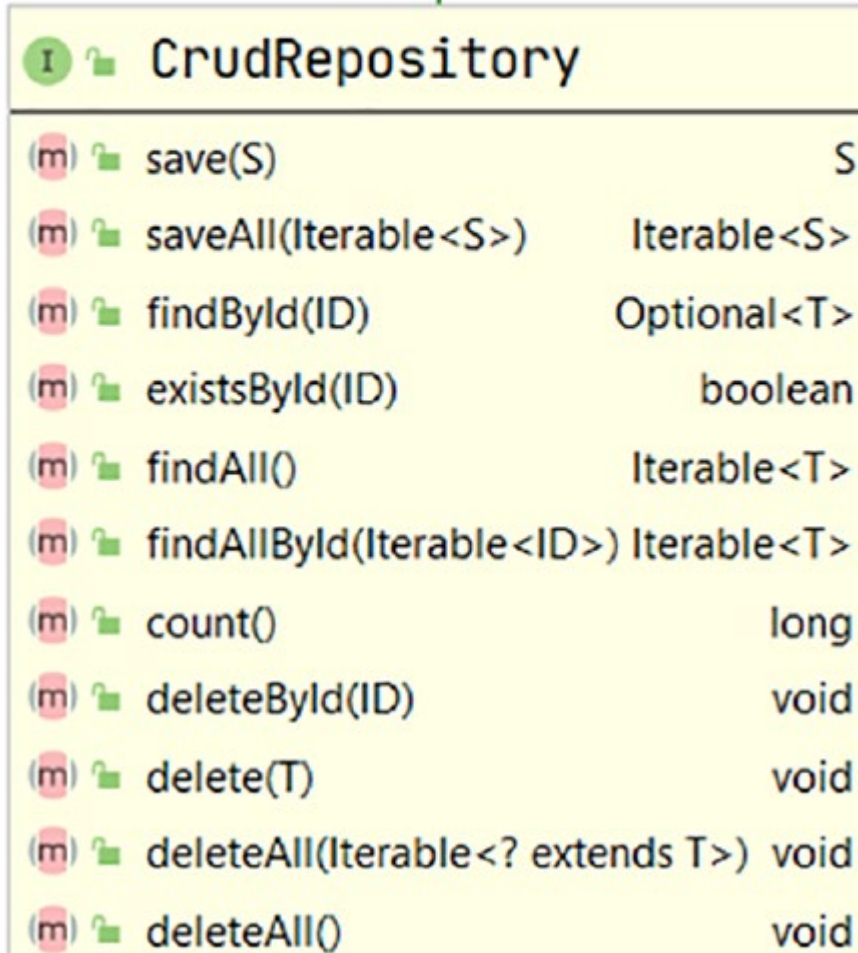
Entendiendo *Repository* y *CrudRepository* Interface

- Spring Data repository usa la interfaz genérica *Repository* como primera abstracción para una fuente de datos.
- *CrudRepository* es una subinterfaz de la interfaz *Repository* y proporciona operaciones CRUD.
- El tipo genérico T representa la CLASE, y el tipo ID representa el TIPO del identificador de la clase

```
public interface CrudRepository<T, ID> extends Repository<T, ID> {
```

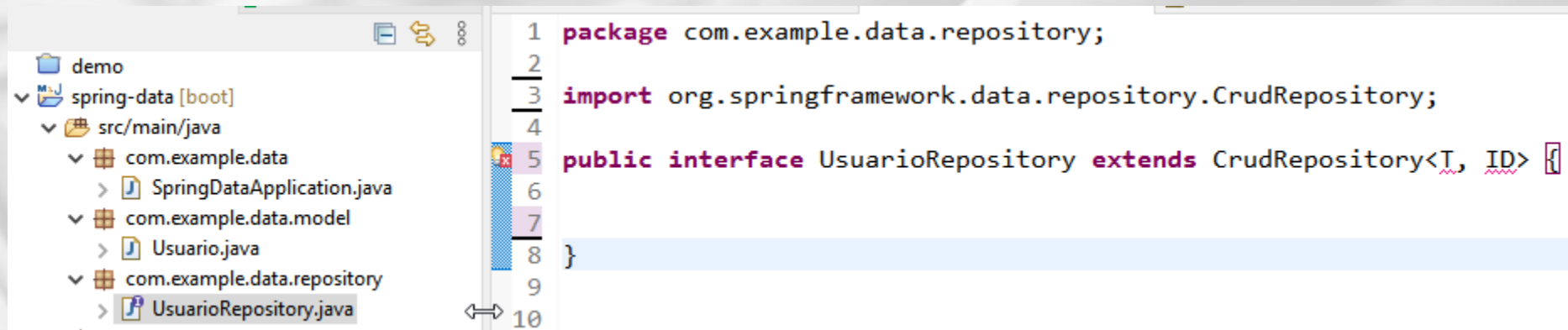


- CRUD operaciones:
- Create
 - save*
- Read
 - find*, exists, count
- Update
 - save*
- Delete
 - delete*



Interface UsuarioRepository

- Crear el subpaquete Repository
 - Dentro, crear la interfaz UsuarioRepository que extiende a *CrudRepository*



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure includes a package `com.example.data.repository` containing `UsuarioRepository.java`. The code editor displays the following Java code:

```
1 package com.example.data.repository;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 public interface UsuarioRepository extends CrudRepository<T, ID> {
6
7 }
8
9
10
```

@Repository

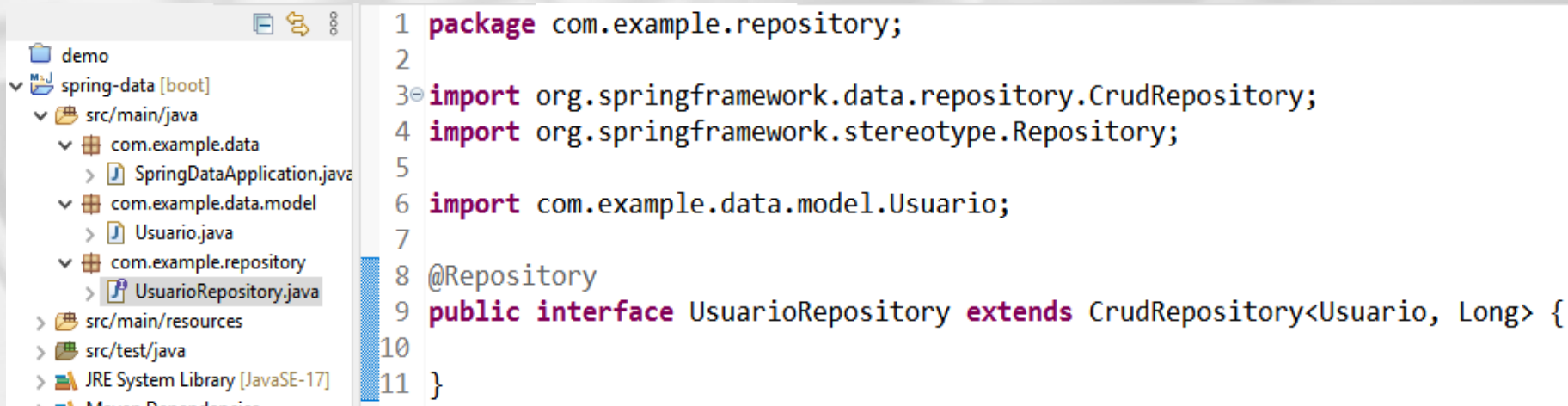
- **@Repository** indica que una clase anotada es un "Repositorio" de datos.
- Se definió originalmente en 2003, como "un mecanismo para encapsular el comportamiento de almacenamiento, recuperación y búsqueda que emula una colección de objetos".
- Desde Spring 2.5, esta anotación es también una especialización de @Component, permitiendo que sea auto detectada.

@Repository

- La anotación @Repository sirve para indicar que se trata de un repositorio de Spring.
- **Aviso:** Aunque parece una interfaz vacía, en tiempo de ejecución la implementación de sus métodos concretos es proporcionada por Spring Data JPA.
- La interfaz CrudRepository cubre las operaciones CRUD.

UsuarioRepository anotado

- @Repository, también ayuda a clarificar el cometido de la clase dentro de la estructura del proyecto

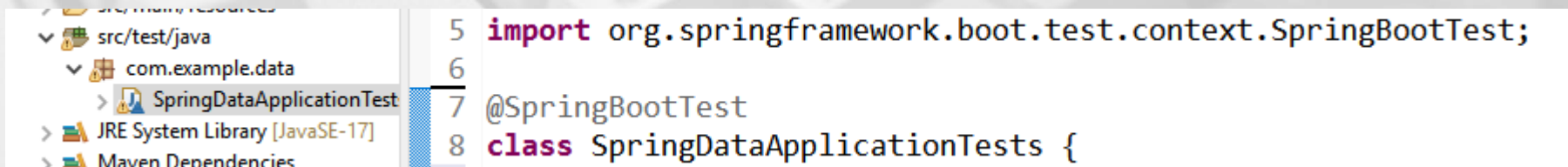


The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure is a Maven project named 'demo' with a 'spring-data' dependency. The source code is organized into packages: 'com.example.data' (containing 'SpringDataApplication.java'), 'com.example.data.model' (containing 'Usuario.java'), and 'com.example.repository' (containing 'UsuarioRepository.java'). The code editor shows the implementation of 'UsuarioRepository' in 'com.example.repository'. It imports 'CrudRepository' and 'Repository' from 'org.springframework.data.repository' and 'Usuario' from 'com.example.data.model'. The class is annotated with '@Repository' and implements 'CrudRepository<Usuario, Long>'.

```
1 package com.example.repository;
2
3 import org.springframework.data.repository.CrudRepository;
4 import org.springframework.stereotype.Repository;
5
6 import com.example.data.model.Usuario;
7
8 @Repository
9 public interface UsuarioRepository extends CrudRepository<Usuario, Long> {
10
11 }
```

SpringDataApplicationTests

- src/test/java/SpringDataApplicationTests.java
- `@SpringBootTest` sirve para ejecutar tests basados en Spring Boot.
 - Para ello realiza una serie de operaciones interna que permiten levantar la aplicación de Spring
- Aquí escribiremos los métodos (tests) para probar la persistencia



The screenshot shows an IDE with a project structure on the left and code on the right. The project structure includes 'src/main/resources', 'src/test/java', 'com.example.data', 'SpringDataApplicationTest', 'JRE System Library [JavaSE-17]', and 'Maven Dependencies'. The code on the right is as follows:

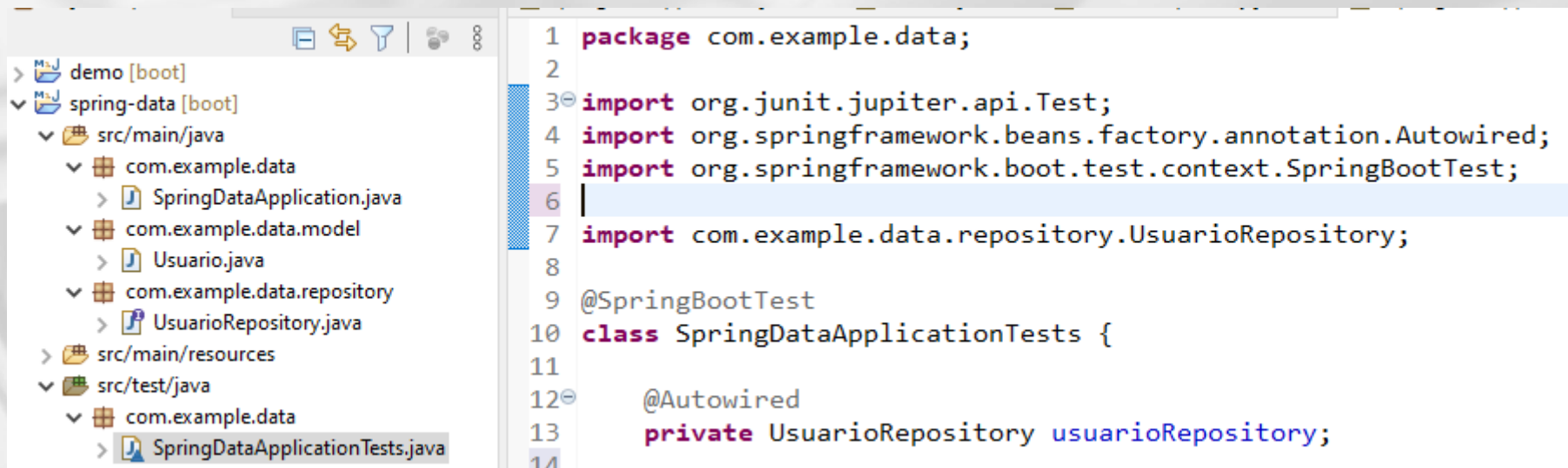
```
5 import org.springframework.boot.test.context.SpringBootTest;  
6  
7 @SpringBootTest  
8 class SpringDataApplicationTests {
```

@Autowired

- @Autowired sirve para auto conectar un constructor, **un campo (atributo, propiedad...)**, un método setter o un método config como auto conectado para la inyección de dependencias de Spring
- De momento, *Autowired Fields*
 - Los campos se inyectan justo después de la construcción de un bean, antes de invocar cualquier método de configuración.
 - Estos campos de configuración no tienen por qué ser públicos.

SpringDataApplicationTests

- En la clase SpringDataApplicationTests
 - Definimos un atributo privado como @Autowired
 - **private** UsuarioRepository **usuarioRepository**;



The screenshot shows an IDE with a project structure on the left and a code editor on the right. The project structure is as follows:

- demo [boot]
 - spring-data [boot]
 - src/main/java
 - com.example.data
 - SpringDataApplication.java
 - com.example.data.model
 - Usuario.java
 - com.example.data.repository
 - UsuarioRepository.java
 - src/main/resources
 - src/test/java
 - com.example.data
 - SpringDataApplicationTests.java

The code editor shows the following code for SpringDataApplicationTests.java:

```
1 package com.example.data;  
2  
3 import org.junit.jupiter.api.Test;  
4 import org.springframework.beans.factory.annotation.Autowired;  
5 import org.springframework.boot.test.context.SpringBootTest;  
6  
7 import com.example.data.repository.UsuarioRepository;  
8  
9 @SpringBootTest  
10 class SpringDataApplicationTests {  
11  
12     @Autowired  
13     private UsuarioRepository usuarioRepository;  
14
```

Practicando: testing Spring Data

- Pruebas a realizar
 - Añade a la entidad Usuario dos atributos: nombre y email
 - @Test para “probar” las operaciones CRUD
 - Crea un test que cree 6 usuarios 3 con el mismo nombre.
 - Crea otro test que consulte los usuarios:
 - Que muestre todos los usuarios
 - Que consulte por Id, el Id= 5 y el 50
 - ¿Cómo se te ocurriría hacer una consulta por nombre?

Posible: Console OutPut

```
Test create user
Consultar Todos Usuarios
Usuario [id=1, name=luiky, email=lvcalderita@unex.es]
Usuario [id=2, name=luiky, email=luiky@unex.es]
Usuario [id=3, name=luiky, email=luiky@gmail.com]
Usuario [id=4, name=lidia, email=lidia@gmail.com]
Usuario [id=5, name=Juan, email=juan@gmail.com]
Usuario [id=6, name=Antonio, email=antonio@unex.es]
-----
Consultar usuarios by ID
Optional[Usuario [id=5, name=Juan, email=juan@gmail.com]]
Optional.empty
-----
Consultar por Nombre
Usuario [id=1, name=luiky, email=lvcalderita@unex.es]
Usuario [id=2, name=luiky, email=luiky@unex.es]
Usuario [id=3, name=luiky, email=luiky@gmail.com]
-----
```

Extras: @Entity

- Normalmente el constructor vacío se declara en Entidades como private o protected
- @Table, permite mapear la clase con un nombre específico de una tabla de la BD
 - @Table(name = "CLIENTES")
- @Column, permite mapear la propiedad con una columna específica de una tabla.
 - @Column(name = "NOMBRE")
 - private String name;

Recursos

- Spring Initializr:
 - <https://start.spring.io>
- Common application properties
 - [application-properties.html](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/CommonApplicationProperties.html)
- Javadoc-api
 - <https://docs.spring.io/spring-framework/docs/current/javadoc-api/>