**Exercise 1: Configuring a Basic Spring Application**

Add the following code to pom.xml

<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>6.1.4</version>  
 </dependency>  
</dependencies>

package com.library.repository;  
  
import com.library.model.Book;  
import java.util.List;  
  
public class BookRepository {  
 public List<Book> findAll() {  
 return List.*of*(  
 new Book(1, "The Hobbit"),  
 new Book(2, "Pride and Prejudice"),  
 new Book(3, "Mahashweta")  
 );  
 }  
}

package com.library.model;  
  
public class Book {  
  
 private final int id;  
 private final String title;  
  
 public Book(int id, String title) {  
 this.id = id;  
 this.title = title;  
 }  
  
 public int getId() { return id; }  
 public String getTitle() { return title; }  
  
 @Override  
 public String toString() {  
 return id + " – " + title;  
 }  
}

package com.library.service;  
  
import com.library.model.Book;  
import com.library.repository.BookRepository;  
import java.util.List;  
  
public class BookService {  
 private final BookRepository repo;  
  
 public BookService(BookRepository repo) {  
 this.repo = repo;  
 }  
  
 public List<Book> listBooks() {  
 return repo.findAll();  
 }  
}

package com.library;  
  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class LibraryApp {  
 public static void main(String[] args) {  
 // Load Spring context from XML  
 ApplicationContext ctx =  
 new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 // Get the BookService bean  
 BookService service = ctx.getBean("bookService", BookService.class);  
  
 // Print all books  
 service.listBooks().forEach(System.*out*::println);  
 }  
}

**Output:**

1 – The Hobbit

2 – Pride and Prejudice

3 – Mahashweta

**Exercise 2: Implementing Dependency Injection**

Update applicationContext.xml

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="https://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="  
 https://www.springframework.org/schema/beans  
 https://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <!-- Repository Bean -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository"/>  
  
 <!-- Service Bean using Setter Injection -->  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
</beans>

package com.library.service;  
  
import com.library.model.Book;  
import com.library.repository.BookRepository;  
import java.util.List;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 // 🔹 Setter method for Spring to inject BookRepository  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public List<Book> listBooks() {  
 return bookRepository.findAll();  
 }  
}

package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class LibraryApp {  
 public static void main(String[] args) {  
 ApplicationContext context =  
 new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 BookService bookService = context.getBean("bookService", BookService.class);  
  
 bookService.listBooks().forEach(System.*out*::println);  
 }  
}

**Output:**

1 – The Hobbit

2 – Pride and Prejudice

3 – Mahashweta

**Exercise 4: Creating and Configuring a Maven Project**

Change pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0   
 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 </properties>  
  
 <dependencies>  
 <!-- Spring Core and Context -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.34</version>  
 </dependency>  
  
 <!-- Spring AOP -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.34</version>  
 </dependency>  
  
 <!-- Spring Web MVC -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.34</version>  
 </dependency>  
  
 <!-- Servlet API (for Spring MVC to compile) -->  
 <dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>javax.servlet-api</artifactId>  
 <version>4.0.1</version>  
 <scope>provided</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <!-- Maven Compiler Plugin -->  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.10.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

**Spring Data JPA - Quick Example**

**Spring Data JPA** is a part of the **Spring Data project** that simplifies the development of JPA-based data access layers. It builds on top of **JPA (Java Persistence API)** and provides an **abstraction layer** to reduce boilerplate code when performing CRUD operations on a database.

**Steps in a Basic Spring Data JPA Application:**

1. **Define the Entity Class**:
   * An entity is a POJO (Plain Old Java Object) annotated with @Entity, mapped to a database table.
   * Each entity has an @Id field representing its primary key.
2. **Create a Repository Interface**:
   * An interface that extends JpaRepository or CrudRepository from Spring Data.
   * This automatically provides CRUD methods like save(), findAll(), deleteById(), etc.
   * Custom queries can also be defined using method naming conventions.
3. **Configure the Database**:
   * The application.properties file is used to configure database connection, JPA settings, and logging.
   * An embedded database like H2 can be used for testing without any external setup.
4. **Create a REST Controller (Optional)**:
   * A controller class annotated with RestController can expose API endpoints for CRUD operations.
   * The controller internally uses the repository to interact with the database.
5. **Run the Application**:
   * The SpringBootApplication annotation on the main class enables auto-configuration and component scanning.
   * Once the application starts, Spring Boot sets up the JPA infrastructure, connects to the database, and prepares the repository beans.

**Example Overview:**

* **Entity Class**: Book.java with fields like id, title, and author
* **Repository Interface**: BookRepository extends JpaRepository<Book, Long>
* **Database**: H2 configured using application.properties
* **Controller**: BookController with endpoints like /books for save and retrieve operations

**Advantages of Using Spring Data JPA:**

* Eliminates boilerplate code for common database operations
* Reduces development time and complexity
* Enables rapid development through convention over configuration
* Provides a clean separation of persistence and business logic
* Supports pagination, sorting, and custom query methods with ease

**Conclusion:**

Spring Data JPA is a powerful abstraction that simplifies working with relational databases in Java applications. By combining JPA’s object-relational mapping capabilities with Spring’s dependency injection and repository support, developers can focus more on business logic and less on repetitive database code. This makes it ideal for building modern, maintainable, and scalable enterprise applications.

**Difference between JPA, Hibernate and Spring Data JPA**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| | **Feature / Aspect** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** | | --- | --- | --- | --- | | **Type** | Specification (Interface/API) | Framework (Implementation of JPA) | Abstraction layer over JPA & Hibernate | | **Provided By** | Oracle / Eclipse Foundation | Red Hat | Spring (Pivotal) | | **Implements** | – | JPA | Uses JPA (usually with Hibernate) | | **Requires Implementation?** | Yes (e.g., Hibernate) | No – it's the implementation | Yes – uses Hibernate or other JPA provider | | **Boilerplate Code** | Moderate | Moderate | Minimal (uses generated methods) | | **Ease of Use** | Medium | Medium | Very Easy | | **SQL Writing Needed** | Yes (JPQL/Native queries) | Yes (HQL/Native queries) | Often No (query method names auto-generate queries) | | **Custom Query Support** | JPQL / Native SQL | JPQL, HQL, Native SQL | JPQL + Derived Queries + @Query | | **CRUD Operation Support** | Manual | Manual | Auto with JpaRepository/CrudRepository | | **Pagination Support** | Manual | Manual | Built-in via Pageable, Slice | | **Caching Support** | Limited | First-level + Second-level cache | Inherited via Hibernate | | **Lazy/Eager Loading** | Supported | Supported | Supported | | **Default in Spring Boot** | No | ✅ Yes (Hibernate) | ✅ Yes | | **Used in Spring Boot Projects** | Often | Yes | ✅ Recommended | | **Query Language** | JPQL | HQL (JPQL + Hibernate extensions) | Method Name Queries + JPQL + @Query | | **EntityManager Usage** | Required | Required | Mostly hidden/abstracted | | **Best For** | ORM standardization across vendors | Advanced ORM use cases | Rapid Spring development with minimal code | |