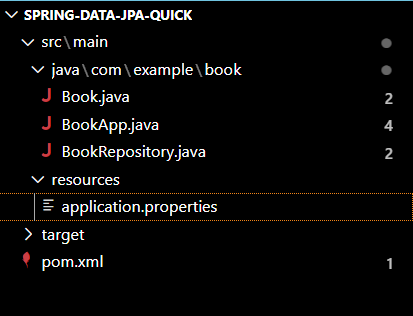
**Spring Data JPA – Quick Example**

**Folder Structure:**



**BookApp.java:**  
package com.example.book;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BookApp implements CommandLineRunner {

    @Autowired

    private BookRepository bookRepository;

    public static void main(String[] args) {

        SpringApplication.run(BookApp.class, args);

    }

    @Override

    public void run(String... args) throws Exception {

        bookRepository.save(new Book(1L, "Java Fundamentals"));

        bookRepository.save(new Book(2L, "Spring Boot in Action"));

        System.out.println("All Books:");

        bookRepository.findAll().forEach(book -> System.out.println(book.getId() + " - " + book.getTitle()));

    }

}

**Book.java:**  
package com.example.book;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class Book {

    @Id

    private Long id;

    private String title;

    public Book() {

    }

    public Book(Long id, String title) {

        this.id = id;

        this.title = title;

    }

    public Long getId() {

        return id;

    }

    public void setId(Long id) {

        this.id = id;

    }

    public String getTitle() {

        return title;

    }

    public void setTitle(String title) {

        this.title = title;

    }

}

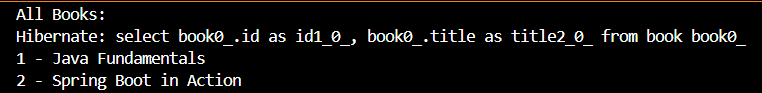
**BookRepository.java:**package com.example.book;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}

**Output:**

****

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

1. **JPA (Java Persistence API)**

**What it is:** A specification for ORM (Object Relational Mapping) in Java offered by Oracle.

**Purpose:** Establishes the procedure of transforming Java objects into relational tables within databases.

**Key Point**: JPA stands as an interface and requires a provider like Hibernate for practical handling.

**Key Features**:

* No implementation itself — it’s just an API.
* Defines annotations like @Entity, @Id, @Table, @OneToMany, etc.
* Promotes portability between different JPA providers (e.g., Hibernate, EclipseLink).

**Common Use Case**:  
If you want a framework-independent persistence layer and don’t want to be locked into Hibernate.

2. **Hibernate**

**What it is**: Widely known for its adherence to the JPA specification, Hibernate is one of its most popular implementations.

**Purpose**: It provides all features of JPA and adds some more features like caching, lazy loading, and HQL (Hibernate Query Language).

**Key Point**: Hibernate has proven to be an effective ORM tool; it takes care of Java object persistence to the database.

**Key Features**:

* Supports JPA and adds its own extensions
* Native query language: HQL (Hibernate Query Language)
* Built-in caching support (first-level and second-level)
* Automatic dirty checking (detecting changes in objects)

**Common Use Case**:  
If you're writing a Java application and want full control and features beyond the JPA spec.

3**. Spring Data JPA**

**What it is**: A framework that extends JPA and Hibernate, it is part of the Spring ecosystem.

**Purpose:** Aids in decreasing the amount of data access boilerplate code in systems underpinned by interfaces and annotations.

**Key Point:** While using Spring Data JPA, in many cases, you will not need to write SQL or even method bodies because the system would generate the queries based on method names.

**Key Features**:

* No need to write implementation classes
* Method name-based query generation (e.g., findByTitle(String title))
* Integration with Spring Boot, H2, PostgreSQL, MySQL, etc.
* Supports paging, sorting, and custom queries via @Query

**Common Use Case**:  
Ideal for Spring Boot applications where you want to rapidly develop clean, maintainable code with minimal configuration.

**JPA (Java Persistence API)** is a specification that defines how Java objects map to database tables, but it doesn’t include any implementation. **Hibernate** is the most popular implementation of JPA, offering advanced features like caching and lazy loading**. Spring Data JPA** builds on JPA and Hibernate to simplify database operations by reducing boilerplate code and auto-generating queries based on method names. In essence, JPA defines the rules, Hibernate implements them, and Spring Data JPA makes using them easier in Spring applications.