Exercise 1: Configuring a Basic Spring Application

Coding:

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<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>
 <name>librarymanagement</name>
 <url>http://www.example.com</url>

 <properties>
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
 <maven.compiler.source>1.8</maven.compiler.source>
 <maven.compiler.target>1.8</maven.compiler.target>
</properties>


 <dependencies>
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.11</version>
   <scope>test</scope>
  </dependency>

  <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>7.0.0-M6</version>
</dependency>
 </dependencies>

 <build>
  <plugins>
   <!-- Plugin to run the Java application -->
   <plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>3.1.0</version>
```

```
      <configuration>
        <mainClass>com.library.App</mainClass>
      </configuration>
    </plugin>

    <!-- Compiler Plugin -->
    <plugin>
 <artifactId>maven-compiler-plugin</artifactId>
 <version>3.8.0</version>
 <configuration>
  <source>1.8</source>
  <target>1.8</target>
 </configuration>
</plugin>

  </plugins>
 </build>

</project>
```

BookService.java

```
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {
 private BookRepository bookRepository;

  // Setter for Spring to inject dependency
  public void setBookRepository(BookRepository bookRepository) {
    this.bookRepository = bookRepository;
  }

  public void addBook(String bookName) {
    System.out.println("Adding book: " + bookName);
    bookRepository.saveBook(bookName);
  }

}
```

BookRepository.java

```
package com.library.repository;

public class BookRepository {
  public void saveBook(String bookName) {
    System.out.println("Saving book: " + bookName);
  }
}
```

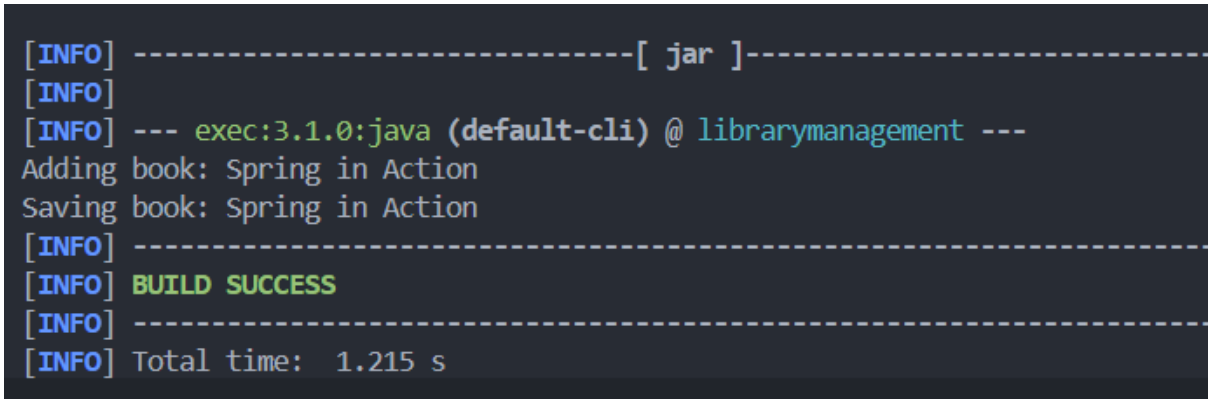
ApplicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
     http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans.xsd">

  <bean id="bookRepository" class="com.library.repository.BookRepository" />

  <bean id="bookService" class="com.library.service.BookService">
    <property name="bookRepository" ref="bookRepository" />
  </bean>

</beans>
```



Exercise 2: Implementing Dependency Injection

ApplicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
     http://www.springframework.org/schema/beans
     http://www.springframework.org/schema/beans/spring-beans.xsd">

  <!-- Repository Bean -->
  <bean id="bookRepository" class="com.library.repository.BookRepository" />

  <!-- Service Bean with setter-based DI -->
  <bean id="bookService" class="com.library.service.BookService">
    <property name="bookRepository" ref="bookRepository" />
  </bean>

</beans>
```

```
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ librarymanagement ---
Adding book: Spring in Action
Saving book: Spring in Action
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
```

Exercise 4: Creating and Configuring a Maven Project

In the Source File:
mvn archetype:generate -DgroupId=com.library -DartifactId=LibraryManagement
DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

Pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<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>
 <name>librarymanagement</name>
 <url>http://www.example.com</url>

 <properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
 </properties>

 <dependencies>
  <!-- JUnit for testing -->
  <dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.11</version>
   <scope>test</scope>
  </dependency>

  <!-- Spring Context -->
```

```
  <dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-context</artifactId>
   <version>7.0.0-M6</version>
  </dependency>

  <!-- Spring AOP -->
  <dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-aop</artifactId>
   <version>7.0.0-M6</version>
  </dependency>

  <!-- Spring WebMVC -->
  <dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-webmvc</artifactId>
   <version>7.0.0-M6</version>
  </dependency>
 </dependencies>

 <build>
  <plugins>
   <!-- Plugin to run the Java application -->
   <plugin>
    <groupId>org.codehaus.mojo</groupId>
    <artifactId>exec-maven-plugin</artifactId>
    <version>3.1.0</version>
    <configuration>
     <mainClass>com.library.App</mainClass>
    </configuration>
   </plugin>

   <!-- Maven Compiler Plugin -->
   <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.0</version>
    <configuration>
     <source>1.8</source>
     <target>1.8</target>
    </configuration>
   </plugin>
  </plugins>
 </build>
</project>
```

OUTPUT

```
rchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------< org.apache.maven:standalone-pom >-------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] --------------------------------[ pom ]---------------------------------
[INFO]
[INFO] >>> archetype:3.4.0:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO]
[INFO] <<< archetype:3.4.0:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO]
[INFO]
[INFO] --- archetype:3.4.0:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Batch mode
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: basedir, Value: C:\Users\Dell\OneDrive\Desktop\Practice2
[INFO] Parameter: package, Value: com.library
[INFO] Parameter: groupId, Value: com.library
[INFO] Parameter: artifactId, Value: LibraryManagement
[INFO] Parameter: packageName, Value: com.library
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\Dell\OneDrive\Desktop\Practice2\LibraryManageme
[INFO] ----------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ----------------------------------------------------------------------------
[INFO] Total time:  3.145 s
[INFO] Finished at: 2025-07-06T17:10:28+05:30
```

```
[INFO] --- exec:3.1.0:java (default-cli) @ librarymanagement ---
Hello World!
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.692 s
[INFO] Finished at: 2025-07-06T17:12:31+05:30
```

Spring Data JPA - Quick Example

Objective

We'll create a minimal Spring Boot application that uses **Spring Data JPA** to perform **CRUD operations** on a Book entity.

Tech Stack

- Spring Boot
- Spring Data JPA
- H2 (in-memory database)
- Maven

Project Structure

```
src/
└── main/
    ├── java/
    │   └── com.example.demo/
    │       ├── DemoApplication.java
    │       ├── model/
    │       │   └── Book.java
    │       ├── repository/
    │       │   └── BookRepository.java
    │       └── controller/
```

```
    └── BookController.java
└── resources/
    └── application.properties
```

Step-by-Step Code

1. 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.example</groupId>
 <artifactId>demo</artifactId>
 <version>0.0.1-SNAPSHOT</version>
 <packaging>jar</packaging>

 <parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.1.4</version>
 </parent>

 <dependencies>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
   <groupId>com.h2database</groupId>
   <artifactId>h2</artifactId>
   <scope>runtime</scope>
  </dependency>
  <dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
 </dependencies>

</project>
```

---

2. Book.java (Entity)

```
package com.example.demo.model;

import jakarta.persistence.*;

@Entity
public class Book {
```

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

    private String title;
    private String author;

    // Getters & Setters
    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; }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }
}
```

---

3. BookRepository.java

```
package com.example.demo.repository;

import com.example.demo.model.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {
    // No implementation needed, Spring generates all CRUD methods
}
```

---

4. BookController.java

```
package com.example.demo.controller;

import com.example.demo.model.Book;
import com.example.demo.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/books")
public class BookController {

    @Autowired
    private BookRepository repo;

    @GetMapping
    public List<Book> getAllBooks() {
```

```
    return repo.findAll();
  }

  @PostMapping
  public Book createBook(@RequestBody Book book) {
    return repo.save(book);
  }
}
```

---

5. application.properties

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update
```

---

6. DemoApplication.java

```
package com.example.demo;

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

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

Run the App

mvn spring-boot:run

Then visit:

- http://localhost:8080/books → GET all books
- POST to http://localhost:8080/books with JSON body:
- {
- "title": "Clean Code",
- "author": "Robert C. Martin"
- }

Difference between JPA, Hibernate and Spring Data JPA

Relationship Overview

Spring Data JPA

⬇

JPA (Java API)

⬇

Hibernate (Implementation of JPA)

---

1. JPA (Java Persistence API)

Feature	Details
Type	Specification (Interface/API)
Owner	Part of Java EE / Jakarta EE
What it does	Defines *how* Java objects should be mapped to relational DBs
Examples	@Entity, @Id, EntityManager, JPQL
Needs Implementation?	Yes (like Hibernate, EclipseLink)

☑ **Think of it as a contract/standard for ORM.**

---

2. Hibernate

Feature	Details
Type	JPA Implementation (ORM framework)
Owner	Red Hat
What it does	Implements JPA specs + adds extra features
Examples	SessionFactory, native queries, @Cascade
Extra Features	Lazy loading tuning, cache support, advanced mappings

---

3. Spring Data JPA

Feature	Details
Type	Abstraction built on top of JPA
Owner	Spring Framework
What it does	Reduces boilerplate (e.g., no need to write DAO implementations)
Examples	JpaRepository, @Query, method-name queries
Benefits	Auto-generated queries, pagination, auditing, etc.

Summary Comparison

Feature	JPA	Hibernate	Spring Data JPA
Type	Specification	Implementation	Spring abstraction
Developed by	Oracle	Red Hat	Spring Team
Boilerplate Reduction	❌ No	❌ Some	☑ Major
Repository Interface	❌ No	❌ No	☑ Yes (JpaRepository)
Custom Query Support	☑ JPQL	☑ JPQL + Native	☑ + Derived Queries
Spring Integration	⛔ Manual	⛔ Manual	☑ Seamless
