

NAME : ANITA.P

SUPERSET ID : 6419735

WEEK – 3

MANDATORY - EXERCISES

1) Exercise 1: Configuring a Basic Spring Application

Scenario:

Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

Steps:

1. Set Up a Spring Project:

- Create a Maven project named **LibraryManagement**.
- Add Spring Core dependencies in the **pom.xml** file.

2. Configure the Application Context:

- Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
- Define beans for **BookService** and **BookRepository** in the XML file.

3. Define Service and Repository Classes:

- Create a package **com.library.service** and add a class **BookService**.
- Create a package **com.library.repository** and add a class **BookRepository**.

4. Run the Application:

- Create a main class to load the Spring context and test the configuration.

ANSWER :

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</version>
  <dependencies>
    <!-- Spring Context (includes Spring Core and Beans) -->
```

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.33</version>
</dependency>
</dependencies>
</project>
```

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

BookRepository.java

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

BookService.java

```
package com.library.service;
import com.library.repository.BookRepository;
public class BookService {
    private BookRepository bookRepository;
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
    public void addBook(String bookName) {
```

```

        System.out.println("Adding book: " + bookName);
        bookRepository.saveBook(bookName);
    }
}

```

MainApp.java

```

package com.library;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.library.service.BookService;

public class MainApp {

    public static void main(String[] args) {

        ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");

        BookService bookService = (BookService) context.getBean("bookService");
        bookService.addBook("Spring in Action");

    }
}

```

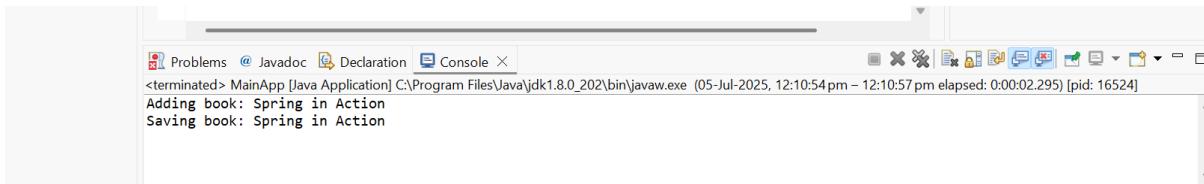
OUTPUT

The screenshot shows the Eclipse IDE interface with the MainApp.java file open in the editor. The code is identical to the one shown above. Below the editor, the Console view displays the application's output:

```

Adding book: Spring in Action
Saving book: Spring in Action

```



2) Exercise 2: Implementing Dependency Injection

Scenario:

In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.

Steps:

1. **Modify the XML Configuration:**
 - o Update **applicationContext.xml** to wire **BookRepository** into **BookService**.
2. **Update the BookService Class:**
 - o Ensure that **BookService** class has a setter method for **BookRepository**.
3. **Test the Configuration:**
 - o Run the **LibraryManagementApplication** main class to verify the dependency injection.

ANSWER :

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>
```

BookService.java

```
package com.library.service;
```

```

import com.library.repository.BookRepository;
public class BookService {
    private BookRepository bookRepository;
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
    public void addBook(String bookName) {
        System.out.println("Adding book: " + bookName);
        bookRepository.saveBook(bookName);
    }
}

```

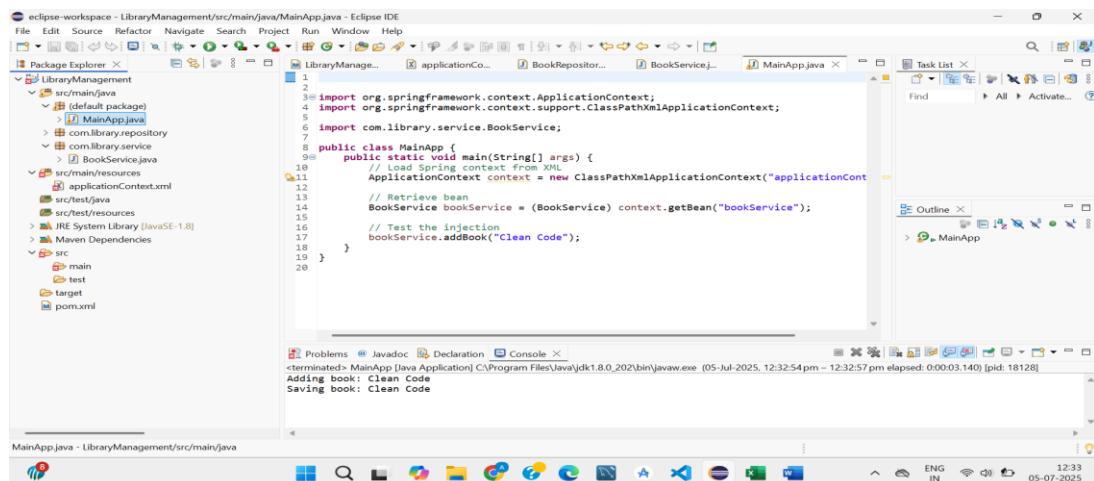
MainApp.java

```

package com.library;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.support.ClassPathXmlApplicationContext;
import com.library.service.BookService;
public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
        BookService bookService = (BookService)
context.getBean("bookService");
        bookService.addBook("Clean Code");
    }
}

```

OUTPUT :





3) Exercise 4: Creating and Configuring a Maven Project

Scenario:

You need to set up a new Maven project for the library management application and add Spring dependencies.

Steps:

1. Create a New Maven Project:

- Create a new Maven project named **LibraryManagement**.

2. Add Spring Dependencies in pom.xml:

- Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.

3. Configure Maven Plugins:

- Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

ANSWER :

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</version>
```

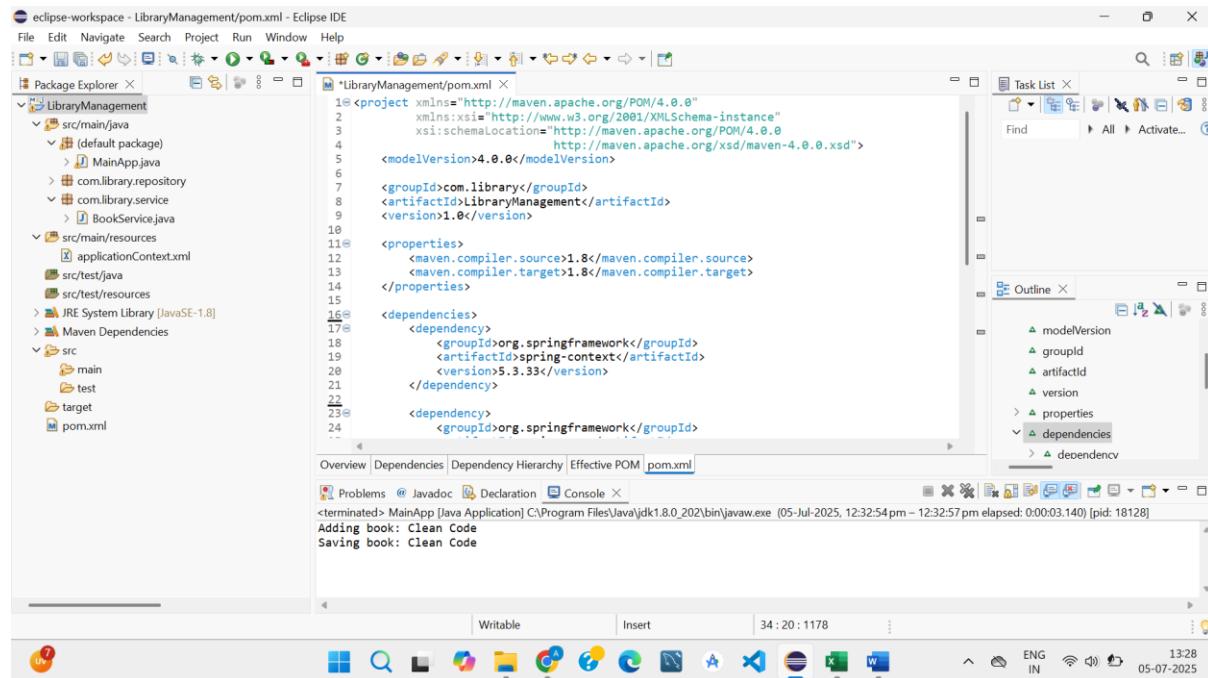
```
<properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>
<dependencies>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.33</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-aop</artifactId>
        <version>5.3.33</version>
    </dependency>
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>5.3.33</version>
    </dependency>
</dependencies>
<build>
    <plugins>
        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.11.0</version>
        
```

```

<configuration>
    <source>1.8</source>
    <target>1.8</target>
</configuration>
</plugin>
</plugins>
</build>
</project>

```

OUTPUT :



4) Spring Data JPA - Quick Example

User.java

```

package com.example.entity;
import jakarta.persistence.*;
@Entity
public class User {

```

```
@Id  
@GeneratedValue(strategy = GenerationType.IDENTITY)  
private Long id;  
private String name;  
private String email;  
public Long getId() { return id; }  
public void setId(Long id) { this.id = id; }  
public String getName() { return name; }  
public void setName(String name) { this.name = name; }  
public String getEmail() { return email; }  
public void setEmail(String email) { this.email = email; }}
```

UserRepository.java

```
package com.example.repository;  
import com.example.entity.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
public interface UserRepository extends JpaRepository<User, Long> {  
}
```

JpaQuickExampleApplication.java

```
package com.example;  
import com.example.entity.User;  
import com.example.repository.UserRepository;  
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 JpaQuickExampleApplication implements  
CommandLineRunner{  
    @Autowired  
    private UserRepository userRepository;  
    public static void main(String[] args) {  
        SpringApplication.run(JpaQuickExampleApplication.class, args);  
    }  
    @Override  
    public void run(String... args) {  
        User user = new User();  
        user.setName("Anita");
```

```

        user.setEmail("anita@example.com");
        userRepository.save(user);
        userRepository.findAll().forEach(u ->
            System.out.println(u.getId() + " - " + u.getName() + " - " +
            u.getEmail())
        );
    }
}

```

OUTPUT:

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure for "jpa-quick-example".
- User.java Content:**

```

1 package com.example.entity;
2
3 import jakarta.persistence.*;
4
5 @Entity
6 public class User {
7
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    private Long id;
11
12    private String name;
13    private String email;
14
15    // Getters and Setters
16    public Long getId() { return id; }
17    public void setId(Long id) { this.id = id; }
18
19    public String getName() { return name; }
20    public void setName(String name) { this.name = name; }
21
22    ...
23}

```
- Java Application Console Output:**

```

JpaQuickExampleApplication (1) [Java Application] C:\Users\anita\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\java.exe (05-
2025-07-05T16:41:50.801+05:30) INFO 8364 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2025-07-05T16:41:51.128+05:30  WARN 8364 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view
2025-07-05T16:41:51.586+05:30  INFO 8364 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080
2025-07-05T16:41:51.599+05:30  INFO 8364 --- [main] com.example.JpaQuickExampleApplication : Started JpaQuickExampleApplication in 0.012 seconds (JVM running for 0.021)
Hibernate: insert into user (email,name) values (?,?)
Hibernate: select u1_0.id,u1_0.email,u1_0.name from user u1_0
1 - Anita - anita@example.com
2025-07-05T17:17:54.955+05:30  WARN 8364 --- [1-1-housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread started
2025-07-05T22:58:25.601+05:30  WARN 8364 --- [1-1-housekeeper] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Thread stopped

```

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure for "jpa-quick-example".
- User.java Content:**

```

1 package com.example.entity;
2
3 import jakarta.persistence.*;
4
5 @Entity
6 public class User {
7
8     @Id
9     @GeneratedValue(strategy = GenerationType.IDENTITY)
10    private Long id;
11
12    private String name;
13    private String email;
14
15    // Getters and Setters
16    public Long getId() { return id; }
17    public void setId(Long id) { this.id = id; }
18
19    public String getName() { return name; }
20    public void setName(String name) { this.name = name; }
21
22    ...
23}

```
- Java Application Console Output:**

```

JpaQuickExampleApplication (1) [Java Application] C:\Users\anita\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.7.v20250502-0916\jre\bin\java.exe (05-
Hibernate: create table user (id bigint not null auto_increment, email varchar(255), name varchar(255), primary key (id)) engine=InnoDB
2025-07-05T16:41:50.801+05:30  INFO 8364 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2025-07-05T16:41:51.128+05:30  WARN 8364 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view
2025-07-05T16:41:51.586+05:30  INFO 8364 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080
2025-07-05T16:41:51.599+05:30  INFO 8364 --- [main] com.example.JpaQuickExampleApplication : Started JpaQuickExampleApplication in 0.012 seconds (JVM running for 0.021)
Hibernate: insert into user (email,name) values (?,?)
Hibernate: select u1_0.id,u1_0.email,u1_0.name from user u1_0
1 - Anita - anita@example.com

```

The screenshot shows the MySQL Workbench interface. In the top-left corner, it says "Local instance MySQL80 x". The menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, Help. The Navigator pane on the left lists various schemas such as ani, anili, anit, anita, college, cust_omer, customer, dat, database1, database2, database3, edu, educt, em. The main area, "Query 1", contains the following SQL code:

```

55 • use star;
56 • show tables;
57 • select *from tasks;
58
59 • select *from project;
60 • select *from project_members;
61 • select *from contact_form;
62
63
64 • CREATE DATABASE your_database;
65 • use your_database;
66 • show tables;
67 • SELECT * FROM user;

```

The "Output" pane below shows the execution log:

#	Action	Time	Message	Duration / Fetch
2	use your_database	16:37:25	0 row(s) affected	0.031 sec
3	show tables	17:19:02	1 row(s) returned	0.125 sec / 0.000 sec
4	SELECT * FROM users LIMIT 0, 1000	17:19:40	Error Code: 1146. Table 'your_database.users' doesn't exist	0.015 sec
5	use your_database	17:20:01	0 row(s) affected	0.000 sec
6	show tables	17:20:07	1 row(s) returned	0.000 sec / 0.000 sec
7	SELECT * FROM user LIMIT 0, 1000	17:20:17	1 row(s) returned	0.032 sec / 0.000 sec

The status bar at the bottom right shows "17:21", "ENG IN", "05-07-2025", and a battery icon.

5) Difference between JPA, Hibernate and Spring Data JPA

Feature/Aspect	JPA(Java Persistence API)	Hibernate	Spring Data JPA
Type	Specification (API only)	Implementation of JPA + extras	Abstraction layer over JPA + Spring support
ORM Support	Defines ORM standards	Full ORM implementation	Uses JPA + underlying ORM like Hibernate
Provider Required	Yes (e.g., Hibernate, EclipseLink)	No, it <i>is</i> a provider	Yes (requires JPA provider like Hibernate)
Query Language	JPQL (Java Persistence Query Lang)	HQL (Hibernate Query Lang)	Derived query methods or @Query annotation
Easy of Use	Moderate	Requires deeper understanding	Very easy (simplified API with Spring Boot)

CRUD Operations	Manually implemented	Manually or using Hibernate tools	Auto-generated from JpaRepository
Use case	When following JPA spec is critical	When low-level control is needed	For rapid Spring-based CRUD apps
Code Example	@Entity + EntityManager	SessionFactory + HQL	JpaRepository + Auto Query Methods
Maintenance Effort	Higher	Higher	Low (less code to maintain)
Batch & Lazy Loading	Supported via annotations	Fully Supported + Tunable	Supported (via JPA provider)
Tooling & Documentation	Extensive (Java EE / Jakarta docs)	Extensive (hibernate.org)	Extensive (Spring docs + community support)

ADDITIONAL EXERCISES

1) Exercise 5: Configuring the Spring IoC Container

Scenario:

The library management application requires a central configuration for beans and dependencies.

Steps:

1. Create Spring Configuration File:

- Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
- Define beans for **BookService** and **BookRepository** in the XML file.

2. Update the BookService Class:

- Ensure that the **BookService** class has a setter method for **BookRepository**.

3. Run the Application:

- o Create a main class to load the Spring context and test the configuration.

ANSWER :

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>
```

BookService.java

```
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

    private BookRepository bookRepository;

    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

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

```
}
```

MainApp

```
package com.library;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.library.service.BookService;
public class MainApp {

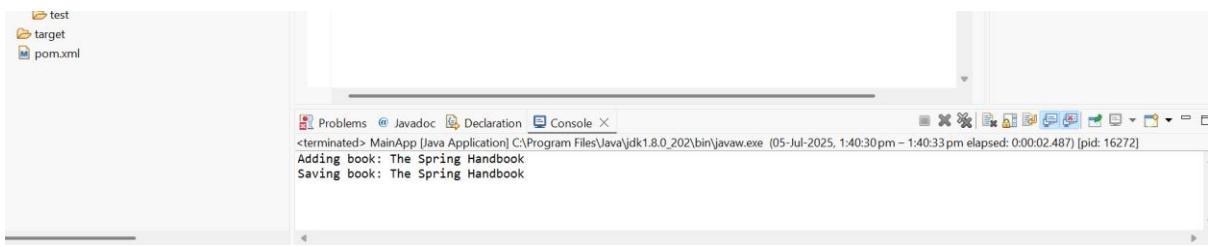
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
        BookService bookService = (BookService) context.getBean("bookService");
        bookService.addBook("The Spring Handbook");
    }
}
```

OUTPUT :

The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows the project structure with packages like src/main/java, src/main/resources, and src/test/java.
- Code Editor:** Displays the MainApp.java code.
- Console:** Shows the terminal output of the application's execution:

```
<terminated> MainApp [Java Application] C:\Program Files\Java\jdk1.8.0_202\bin\javaw.exe (05-Jul-2025, 1:40:30 pm - 1:40:33 pm elapsed: 0:00:02.487) [pid: 16272]
Adding book: The Spring Handbook
Saving book: The Spring Handbook
```
- Bottom Status Bar:** Shows system information including battery level, network, and date/time (05-07-2025, 13:40).



2) Exercise 7: Implementing Constructor and Setter Injection

Scenario:

The library management application requires both constructor and setter injection for better control over bean initialization.

Steps:

1. Configure Constructor Injection:

- Update **applicationContext.xml** to configure constructor injection for **BookService**.

2. Configure Setter Injection:

- Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.

3. Test the Injection:

- Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

ANSWER:

BookService.java

```
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

    private BookRepository bookRepository;

    private String serviceName;

    public BookService(String serviceName) {
```

```
    this.serviceName = serviceName;  
}  
  
public void setBookRepository(BookRepository bookRepository) {  
    this.bookRepository = bookRepository;  
}  
  
public void addBook(String bookName) {  
    System.out.println("[ " + serviceName + " ] 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">  
    <constructor-arg value="Library Service Bean"/>  
    <property name="bookRepository" ref="bookRepository" />
```

```
</bean>
```

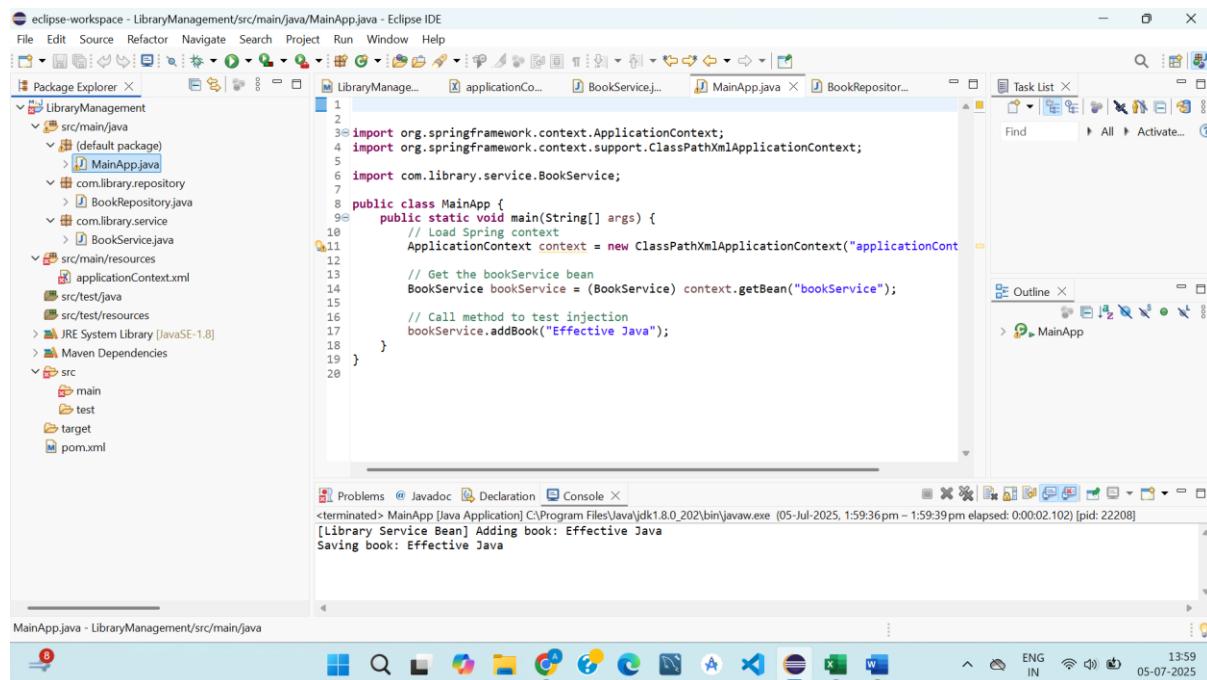
```
</beans>
```

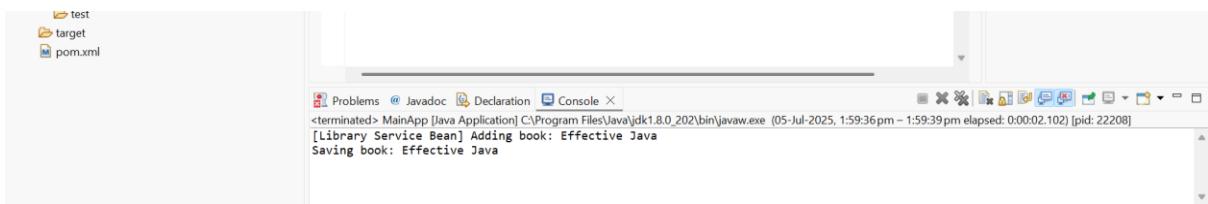
MainApp.java

```
package com.library;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.library.service.BookService;
public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");
        BookService bookService = (BookService) context.getBean("bookService");
        bookService.addBook("Effective Java");
    }
}
```

OUTPUT :





3) Exercise 9: Creating a Spring Boot Application

Scenario:

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

Steps:

1. Create a Spring Boot Project:

- Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.

2. Add Dependencies:

- Include dependencies for **Spring Web**, **Spring Data JPA**, and **H2 Database**.

3. Create Application Properties:

- Configure database connection properties in **application.properties**.

4. Define Entities and Repositories:

- Create **Book** entity and **BookRepository** interface.

5. Create a REST Controller:

- Create a **BookController** class to handle CRUD operations.

6. Run the Application:

- Run the Spring Boot application and test the REST endpoints.

ANSWER :

pom.xml

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <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>
</dependencies>

application.properties
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
spring.datasource.url=jdbc:h2:mem:librarydb
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Book.java

```
package com.library.entity;
import jakarta.persistence.*;
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;
    public Book() {}
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }
    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; }
}
BookRepository.java
package com.library.repository;
```

```
import com.library.entity.Book;  
import org.springframework.data.jpa.repository.JpaRepository;  
public interface BookRepository extends JpaRepository<Book, Long> {  
}
```

BookController.java

```
package com.library.controller;  
import com.library.entity.Book;  
import com.library.repository.BookRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.*;  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/books")  
public class BookController {  
  
    @Autowired  
    private BookRepository bookRepository;  
  
    @GetMapping  
    public List<Book> getAllBooks() {  
        return bookRepository.findAll();  
    }  
  
    @GetMapping("/{id}")  
    public Book getBookById(@PathVariable Long id) {  
        return bookRepository.findById(id).orElse(null);  
    }  
}
```

```

@PostMapping
public Book addBook(@RequestBody Book book) {
    return bookRepository.save(book);
}

@GetMapping("/{id}")
public Book updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {
    return bookRepository.findById(id).map(book -> {
        book.setTitle(updatedBook.getTitle());
        book.setAuthor(updatedBook.getAuthor());
        return bookRepository.save(book);
    }).orElse(null);
}

@DeleteMapping("/{id}")
public void deleteBook(@PathVariable Long id) {
    bookRepository.deleteById(id);
}

```

LibraryManagementApplication.java

```

package com.library;

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

@SpringBootApplication
public class LibraryManagementApplication {

    public static void main(String[] args) {
        SpringApplication.run(LibraryManagementApplication.class, args);
    }
}

```

OUTPUT :

```
'kspace -colibaryManagementApplication.java
```

The screenshot shows a Java IDE interface with the following details:

- Project Explorer:** Shows the project structure under 'src/main/java':
 - (default package)
 - com.library.Controller.java
 - BookController.java
 - com.library.controller
 - Book.java
 - com.library.repository
 - BookRepository.java

Also listed are: ./main/resources, application.properties, c/test/java, c/test/resources, RE System Library [JavaSE-17], Maven Dependencies, and c.
- Console Tab:** Displays the following log output:

```
Console = LibraryMagalApplicationJava
LuraringManagementApplication"-
Boot] 3.2.S: SpringBootApplicat
[2024-04-25T01:41:31.161+0S:R
[2024-04-25T01:41:31.161worgExe
[2024-04-25T01:41:31.161worgStai
[2024-04-25T01:41:31.161dpbcStai
[2024-04-25T01:41:31.161Preparai
[2024-04-25T01:41:31.161Rootweb
[2024-04-25T01:41:31.161Reinitst
[2024-04-25T01:41:31.161webServ
[2024-04-25T01:41:31.801Tomcat v
[2024-04-25T01:41:31.8080(http)
[2024-04-25T01:41:31.801Started
    Tomcat started on port(s): 8080
Run LibraryManitumpliation star
Console initialized in 2.128 se
```

```
'se boot eodrspace LibraryManagementlr
```

The screenshot shows a Java IDE interface with the following details:

- Package Explorer:** Shows the project structure under 'src/main/java':
 - (default package)
 - com.library.Controller.java
 - com.library.repository
 - com.library.entity

Also listed are: src/main/java, application.properties, src/test/java, src/test/resources, Help.md, and pom.xml.
- Console Tab:** Displays the following log output:

```
minal.libraryManagementApplication.java)acaus'L.eibaryManagementApplication(13:0.109.737
[30:01.09.737]      o.s.b.w.w.embedded.tomcat.TomcatWebServer Started tomca
[18:01.09.742]      o.a.c.c.c.StandardService Stanted Cor      start http-@s
[13:01.09.745]      j.LocalContainerEntityFactoryBean Started tom
[13:01.09.760]      o.s.b.a.e.web.EndpointLinksResolver Repf tnd Commactio
[13:01:19.761]      s.LibraryManagementApplication Started LibraryManageme
[13:01:10.161]      Started LibraryManagementAppiication in 1.279 seconds (pl
```

4) Implement services for managing Country

ANSWER:

Country.java

```
package com.example.countryservice.model;
import jakarta.persistence.*;
@Entity
public class Country {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String capital;
    private long population;
    public Country() {}
    public Country(String name, String capital, long population) {
        this.name = name;
        this.capital = capital;
        this.population = population;
    }

    // Getters and setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```
public String getCapital() {
    return capital;
}

public void setCapital(String capital) {
    this.capital = capital;
}

public long getPopulation() {
    return population;
}

public void setPopulation(long population) {
    this.population = population;
}
}
```

CountryRepository

```
package com.example.countryservice.repository;

import com.example.countryservice.model.Country;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CountryRepository extends JpaRepository<Country, Long> {
    // No code required, JpaRepository provides CRUD by default
}
```

CountryService

```
package com.example.countryservice.service;

import com.example.countryservice.model.Country;
import com.example.countryservice.repository.CountryRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;
```

```

@Service
public class CountryService {

    @Autowired
    private CountryRepository countryRepository;

    public List<Country> getAllCountries() {
        return countryRepository.findAll();
    }

    public Optional<Country> getCountryById(Long id) {
        return countryRepository.findById(id);
    }

    public Country addCountry(Country country) {
        return countryRepository.save(country);
    }

    public Country updateCountry(Long id, Country updatedCountry) {
        return countryRepository.findById(id).map(country -> {
            country.setName(updatedCountry.getName());
            country.setCapital(updatedCountry.getCapital());
            country.setPopulation(updatedCountry.getPopulation());
            return countryRepository.save(country);
        }).orElse(null);
    }

    public void deleteCountry(Long id) {
        countryRepository.deleteById(id);
    }
}

```

CountryController

```

package com.example.countryservice.controller;

import com.example.countryservice.model.Country;
import com.example.countryservice.service.CountryService;
import org.springframework.beans.factory.annotation.Autowired;

```

```
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/countries")
@CrossOrigin(origins = "*")
public class CountryController {

    @Autowired
    private CountryService countryService;

    @GetMapping
    public List<Country> getAllCountries() {
        return countryService.getAllCountries();
    }

    @GetMapping("/{id}")
    public Optional<Country> getCountryById(@PathVariable Long id) {
        return countryService.getCountryById(id);
    }

    @PostMapping
    public Country addCountry(@RequestBody Country country) {
        return countryService.addCountry(country);
    }

    @PutMapping("/{id}")
    public Country updateCountry(@PathVariable Long id, @RequestBody
Country country) {
        return countryService.updateCountry(id, country);
    }

    @DeleteMapping("/{id}")
    public void deleteCountry(@PathVariable Long id) {
        countryService.deleteCountry(id);
    }
}
```

Postman Operations

POST

URL : <http://localhost:8081/countries>

```
{  
  "name": "India",  
  "capital": "New Delhi",  
  "population": 1400000000  
}
```

GET

URL : <http://localhost:8081/countries>

OUTPUT :

```
{  
  "name": "India",  
  "capital": "New Delhi",  
  "population": 1400000000  
}
```

GET (By Country Id)

URL : <http://localhost:8081/countries/1>

```
{  
  "name": "India",  
  "capital": "New Delhi",  
  "population": 1400000000  
}
```

PUT (Update the following)

URL : <http://localhost:8081/countries/1>

Input:

```
{  
  "name": "Tamilnadu",  
  "capital": "chennai",  
  "population": 1450000000  
}
```

Where in the above name is changed to Tamilnadu from India and Capital as Chennai from Delhi

OUTPUT:

The screenshot shows the Postman interface. A POST request is made to `http://localhost:8081/countries`. The request body is a JSON object with fields: `"name": "India"`, `"capital": "New Delhi"`, and `"population": 1400000000`. The response status is 200 OK, time 191 ms, size 322 B.

The screenshot shows the Postman interface. A GET request is made to `http://localhost:8081/countries`. The request body is a JSON object with fields: `"id": 1`, `"name": "India"`, `"capital": "New Delhi"`, and `"population": 1400000000`. The response status is 200 OK, time 232 ms, size 324 B.

Postman screenshot showing a successful GET request to `http://localhost:8081/countries/1`. The response status is 200 OK, time 80 ms, size 322 B. The JSON response body is:

```

1 {
2   "id": 1,
3   "name": "India",
4   "capital": "New Delhi",
5   "population": 1400000000
6 }

```

Postman screenshot showing a successful PUT request to `http://localhost:8081/countries/1`. The response status is 200 OK, time 17 ms, size 324 B. The JSON response body is:

```

1 {
2   "id": 1,
3   "name": "Tamilnadu",
4   "capital": "chennai",
5   "population": 1450000000
6 }

```

Eclipse IDE screenshot showing the Java code for `CountryService.java` and the Spring Boot application log.

`CountryService.java` code:

```

1 package com.example.countryservice.service;
2
3 import com.example.countryservice.model.Country;
4 import com.example.countryservice.repository.CountryRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import java.util.List;
9 import java.util.Optional;

```

Spring Boot application log:

```

:: Spring Boot ::          (v3.5.3)
2025-07-06T17:36:33.773+05:30 INFO 18024 --- [           main] c.e.c.CountryServiceApplication           : Starting Count
2025-07-06T17:36:33.778+05:30 INFO 18024 --- [           main] c.e.c.CountryServiceApplication           : No active prof
2025-07-06T17:36:34.964+05:30 INFO 18024 --- [           main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping
2025-07-06T17:36:35.054+05:30 INFO 18024 --- [           main] s.d.r.c.RepositoryConfigurationDelegate : Finished Sprin
2025-07-06T17:36:35.696+05:30 INFO 18024 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initiali
2025-07-06T17:36:35.701+05:30 INFO 18024 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : StandardEngine
2025-07-06T17:36:35.721+05:30 INFO 18024 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Serv
2025-07-06T17:36:35.792+05:30 INFO 18024 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[]         : Initializing S
2025-07-06T17:36:35.793+05:30 INFO 18024 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplic
2025-07-06T17:36:36.055+05:30 INFO 18024 --- [           main] o.hibernate.jpa.internal.util.LogHelper   : HHH000412: H
2025-07-06T17:36:36.164+05:30 INFO 18024 --- [           main] o.h.c.internal.RegionFactoryInitiator     : HHH000026: Sec

```

5) Find a country based on country code

ANSWER :

Country.java

```
package com.example.countryservice.model;

import jakarta.persistence.*;

@Entity
public class Country {

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

    private String name;
    private String capital;
    private long population;

    @Column(unique = true)
    private String countryCode;

    public Country() {}

    public Country(String name, String capital, long population, String
countryCode) {
        this.name = name;
        this.capital = capital;
        this.population = population;
        this.countryCode = countryCode;
    }

    // Getters and Setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
```

```
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getCapital() {
    return capital;
}

public void setCapital(String capital) {
    this.capital = capital;
}

public long getPopulation() {
    return population;
}

public void setPopulation(long population) {
    this.population = population;
}

public String getCountryCode() {
    return countryCode;
}

public void setCountryCode(String countryCode) {
    this.countryCode = countryCode;
}
}
```

CountryRepository.java

```
package com.example.countryservice.repository;
```

```
import com.example.countryservice.model.Country;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface CountryRepository extends JpaRepository<Country, Long> {
    Optional<Country> findByCountryCode(String countryCode);
}
```

CountryService.java

```
package com.example.countryservice.service;

import com.example.countryservice.model.Country;
import com.example.countryservice.repository.CountryRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class CountryService {

    @Autowired
    private CountryRepository countryRepository;

    public List<Country> getAllCountries() {
        return countryRepository.findAll();
    }

    public Optional<Country> getCountryById(Long id) {
        return countryRepository.findById(id);
    }

    public Country addCountry(Country country) {
        return countryRepository.save(country);
    }

    public Country updateCountry(Long id, Country updatedCountry) {
```

```
        return countryRepository.findById(id).map(country -> {
            country.setName(updatedCountry.getName());
            country.setCapital(updatedCountry.getCapital());
            country.setPopulation(updatedCountry.getPopulation());
            country.setCountryCode(updatedCountry.getCountryCode());
            return countryRepository.save(country);
        }).orElse(null);
    }

    public void deleteCountry(Long id) {
        countryRepository.deleteById(id);
    }

    public Optional<Country> getCountryByCode(String countryCode) {
        return countryRepository.findByCountryCode(countryCode);
    }
}
```

CountryController.java

```
package com.example.countryservice.controller;

import com.example.countryservice.model.Country;
import com.example.countryservice.service.CountryService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Optional;

@RestController
@RequestMapping("/countries")
@CrossOrigin(origins = "*")
public class CountryController {

    @Autowired
    private CountryService countryService;

    @GetMapping
    public List<Country> getAllCountries() {
```

```

        return countryService.getAllCountries();
    }

    @GetMapping("/{id}")
    public Optional<Country> getCountryById(@PathVariable Long id) {
        return countryService.getCountryById(id);
    }

    @PostMapping
    public Country addCountry(@RequestBody Country country) {
        return countryService.addCountry(country);
    }

    @PutMapping("/{id}")
    public Country updateCountry(@PathVariable Long id, @RequestBody
Country country) {
        return countryService.updateCountry(id, country);
    }

    @DeleteMapping("/{id}")
    public void deleteCountry(@PathVariable Long id) {
        countryService.deleteCountry(id);
    }
}

```

Postman Operations

POST

URL : <http://localhost:8080/countries>

```
{
  "name": "India",
  "capital": "New Delhi",
  "population": 1400000000,
  "countryCode": "IN"}
```

GET

URL : <http://localhost:8080/countries/code/IN>

```
{  
  "id": 1,  
  "name": "India",  
  "capital": "New Delhi",  
  "population": 1400000000,  
  "countryCode": "IN"  
}
```

OUTPUT

The screenshot shows the Postman application interface. A POST request is made to `http://localhost:8081/countries`. The request body is a JSON object:

```
1 {  
2   "id": 1,  
3   "name": "India",  
4   "capital": "New Delhi",  
5   "population": 1400000000,  
6   "countryCode": "IN"  
7 }
```

The response status is 200 OK, with a time of 21 ms and a size of 322 B. The response body is identical to the request body.

The screenshot shows the Postman application interface. A GET request is made to `http://localhost:8082/countries/code/IN`. The request body is a JSON object:

```
1 {  
2   "id": 1,  
3   "name": "India",  
4   "capital": "New Delhi",  
5   "population": 1400000000,  
6   "countryCode": "IN"  
7 }
```

The response status is 200 OK, with a time of 368 ms and a size of 341 B. The response body is identical to the request body.

eclipse-workspace - country-service/src/main/java/com/example/countryservice/model/Country.java - Eclipse IDE

```

1 package com.example.countryservice.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 Country {
10     @Id
11     @GeneratedValue(strategy = GenerationType.IDENTITY)
12     private Long id;
13
14     private String name;
15
16     private String capital;
17
18     private Long population;
19
20     private String countryCode;
21
22     public Long getId() {
23         return id;
24     }
25
26     public void setId(Long id) {
27         this.id = id;
28     }
29
30     public String getName() {
31         return name;
32     }
33
34     public void setName(String name) {
35         this.name = name;
36     }
37
38     public String getCapital() {
39         return capital;
40     }
41
42     public void setCapital(String capital) {
43         this.capital = capital;
44     }
45
46     public Long getPopulation() {
47         return population;
48     }
49
50     public void setPopulation(Long population) {
51         this.population = population;
52     }
53
54     public String getCountryCode() {
55         return countryCode;
56     }
57
58     public void setCountryCode(String countryCode) {
59         this.countryCode = countryCode;
60     }
61 }

```

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer X Country.java CountryRepos... CountryContr... CountryServ... application... Task List X Find All Activate... Outline X com.example.countryservice.model.Country

Problems Declaration Console X CountryServiceApplication [Java Application] C:\Users\anita.p2\pool\plugins\org.eclipse.jdt.core\hotspot\jre\full\win32\x86_64_21.0.7.v20250502-0916\ve\bin\javaw.exe 2025-07-06T18:22:04.836+05:30 INFO 26184 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JF 2025-07-06T18:22:05.305+05:30 WARN 26184 --- [main] JpaBaseContainerEntityManagerFactoryBean : spring.jpa.openInNewTransaction = true, setting bound value to false (was null). This is likely to lead to memory leaks. 2025-07-06T18:22:05.665+05:30 INFO 26184 --- [main] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at http://localhost:8082/h2-console. 2025-07-06T18:22:05.803+05:30 INFO 26184 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8082 (http) 2025-07-06T18:22:05.813+05:30 INFO 26184 --- [main] c.e.c.CountryServiceApplication : Started CountryServiceApplication in 0.894 seconds (JVM running for 1.012) 2025-07-06T18:22:15.580+05:30 INFO 26184 --- [nio-8082-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring root WebApplicationContext 2025-07-06T18:22:15.589+05:30 INFO 26184 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Spring DispatcherServlet 'dispatcherServlet' 2025-07-06T18:22:15.591+05:30 INFO 26184 --- [nio-8082-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms. Hibernate: insert into country (capital,country_code,name,population,id) values (?, ?, ?, ?, ?) default 2025-07-06T18:22:15.591+05:30 INFO 26184 --- [main] o.s.orm.hibernate.SessionFactory : select c1_.id,c1_.capital,c1_.name,c1_.population from country c1_0 where c1_.country_code = ?

Writable Smart Insert 69:1:1412 ... ENG IN 18:24 06-07-2025

6) Add a new country

≡ ← → Home Workspaces API Network Search Postman Invite Upgrade

HTTP http://localhost:8082/countries POST http://localhost:8082/countries Send

Params Authorization Headers (G) Body Scripts Tests Settings Cookies Beautify

None form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2     "name": "India",
3     "capital": "New Delhi",
4     "population": 14000000000,
5     "countryCode": "IN"
6 }
7

```

Status: 200 OK Time: 814 ms Size: 341 B Save as example

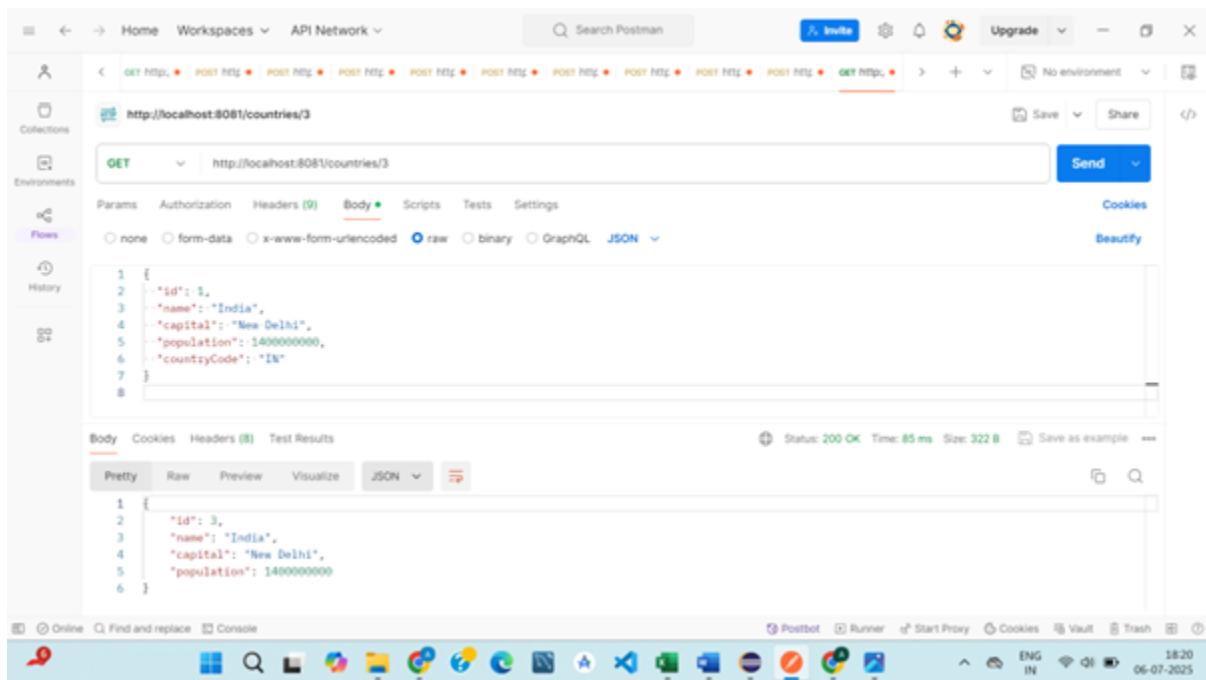
Pretty Raw Preview Visualize JSON

```

1 {
2     "id": 1,
3     "name": "India",
4     "capital": "New Delhi",
5     "population": 14000000000,
6     "countryCode": "IN"
7 }

```

Online Find and replace Console Postbot Runner Start Proxy Cookies Vault Trash ENG IN 18:30 06-07-2025



7) Demonstrate implementation of Query Methods feature of Spring Data JPA

ANSWER :

1. Entity Class – Country.java

```

@Entity
public class Country {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String capital;
    private long population;

    @Column(unique = true)
    private String countryCode;

    // Constructors, Getters, Setters
}

```

2. Repository Interface – CountryRepository.java

```

@Repository
public interface CountryRepository extends JpaRepository<Country, Long> {

```

```
Optional<Country> findByName(String name);

List<Country> findByCapital(String capital);

List<Country> findByPopulationGreaterThan(long population);

Optional<Country> findByCountryCodeIgnoreCase(String countryCode);

List<Country> findByNameContaining(String keyword);

}
```

3. Service Layer – CountryService.java

```
@Service
public class CountryService {

    @Autowired
    private CountryRepository countryRepository;

    public Optional<Country> getByName(String name) {
        return countryRepository.findByName(name);
    }

    public List<Country> getByCapital(String capital) {
        return countryRepository.findByCapital(capital);
    }

    public List<Country> getByPopulationGreaterThan(long value) {
        return countryRepository.findByPopulationGreaterThan(value);
    }

    public Optional<Country> getByCountryCode(String code) {
        return countryRepository.findByCountryCodeIgnoreCase(code);
    }

    public List<Country> findByNameContaining(String keyword) {
        return countryRepository.findByNameContaining(keyword);
    }
}
```

4. Controller Layer – CountryController.java

```
@RestController
```

```

@RequestMapping("/countries")
public class CountryController {

    @Autowired
    private CountryService service;

    @GetMapping("/search/name/{name}")
    public Optional<Country> getByName(@PathVariable String name) {
        return service.getByName(name);
    }

    @GetMapping("/search/capital/{capital}")
    public List<Country> getByCapital(@PathVariable String capital) {
        return service.getByCapital(capital);
    }

    @GetMapping("/search/population/{value}")
    public List<Country> getByPopulationGreaterThan(@PathVariable long value) {
        return service.getByPopulationGreaterThan(value);
    }

    @GetMapping("/search/code/{code}")
    public Optional<Country> getByCountryCode(@PathVariable String code) {
        return service.getByCountryCode(code);
    }

    @GetMapping("/search/name/contains/{keyword}")
    public List<Country> getByNameContains(@PathVariable String keyword) {
        return service.getByNameContaining(keyword);
    }
}

```

8) Demonstrate implementation of O/R Mapping

- Automatically maps Java classes to database tables.
- Helps perform CRUD operations on database using Java objects — without writing SQL.
- Simplifies development and improves maintainability.

1. Create an Entity Class – Country.java

```
package com.example.ormdemo.model;

import jakarta.persistence.*;

@Entity
@Table(name = "countries")
public class Country {

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

    @Column(nullable = false)
    private String name;

    private String capital;

    private long population;

    @Column(unique = true)
    private String countryCode;

    public Country() {}

    public Country(String name, String capital, long population, String countryCode) {
        this.name = name;
        this.capital = capital;
        this.population = population;
        this.countryCode = countryCode;
    }
}
```

2. Spring Data JPA Repository – CountryRepository.java

```
package com.example.ormdemo.repository;
```

```
import com.example.ormdemo.model.Country;
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface CountryRepository extends JpaRepository<Country, Long> {  
}
```

3. Service Class – CountryService.java

```
package com.example.ormdemo.service;  
  
import com.example.ormdemo.model.Country;  
import com.example.ormdemo.repository.CountryRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class CountryService {  
  
    @Autowired  
    private CountryRepository countryRepository;  
  
    public Country addCountry(Country country) {  
        return countryRepository.save(country);  
    }  
  
    public List<Country> getAllCountries() {  
        return countryRepository.findAll();  
    }  
}
```

4. Controller Class – CountryController.java

```
package com.example.ormdemo.controller;  
  
import com.example.ormdemo.model.Country;  
import com.example.ormdemo.service.CountryService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.*;  
  
import java.util.List;  
  
@RestController
```

```

@RequestMapping("/countries")
public class CountryController {

    @Autowired
    private CountryService countryService;

    @PostMapping
    public Country addCountry(@RequestBody Country country) {
        return countryService.addCountry(country);
    }

    @GetMapping
    public List<Country> getAllCountries() {
        return countryService.getAllCountries();
    }
}

```

9) Demonstrate writing Hibernate Query Language and Native Query

ANSWER :

1. Entity: Country.java

```

@Entity
public class Country {

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

    private String name;
    private String capital;
    private long population;

    @Column(unique = true)
    private String countryCode;

    // Constructors, Getters, Setters
}

```

◆ 2. Repository: CountryRepository.java

```
@Repository
public interface CountryRepository extends JpaRepository<Country, Long> {

    @Query("SELECT c FROM Country c WHERE c.population > :pop")
    List<Country> findCountriesWithPopulationGreaterThan(@Param("pop")
    long population);
```

```
    @Query(value = "SELECT * FROM country WHERE capital = :capital",
    nativeQuery = true)
    List<Country> findCountriesByCapital(@Param("capital") String capital);
}
```

3. Service: CountryService.java

```
@Service
public class CountryService {

    @Autowired
    private CountryRepository repository;

    public List<Country> getByPopulationGreaterThan(long population) {
        return repository.findCountriesWithPopulationGreaterThan(population);
    }

    public List<Country> getByCapital(String capital) {
        return repository.findCountriesByCapital(capital);
    }
```

4. Controller: CountryController.java

```
@RestController
@RequestMapping("/countries")
public class CountryController {

    @Autowired
    private CountryService service;

    @GetMapping("/hql/population/{value}")
    public List<Country> getByHQLPopulation(@PathVariable long value) {
        return service.getByPopulationGreaterThan(value);
    }
```

```
@GetMapping("/native/capital/{capital}")
public List<Country> getByNativeCapital(@PathVariable String capital) {
    return service.getByCapital(capital);
}
```