Spring Core and Maven

Exercise 1: Configuring a Basic Spring Application

```
<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>

    <dependencies>
        <!-- Spring Core dependency -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.34</version>
        </dependency>
    </dependencies>
</project>
<?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>
package com.library.repository;

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

import com.library.repository.BookRepository;

public class BookService {
   private BookRepository bookRepository;

   // Setter injection
```

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

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

Output:

Adding book: The Great Gatsby

Saving book: The Great Gatsby

Exercise 2: Implementing Dependency Injection

```
<?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">

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

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

</beans>
package com.library.service;

import com.library.repository.BookRepository;
```

```
public class BookService {
    private BookRepository bookRepository;

    // Setter method for DI
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

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

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

```
import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import
org.springframework.context.support.ClassPathXmlApplicationContext;


public class LibraryManagementApplication {

   public static void main(String[] args) {

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


      BookService bookService = context.getBean("bookService",
BookService.class);

      bookService.addBook("To Kill a Mockingbird");

   }

}
```

Output:

```
Adding book: To Kill a Mockingbird

Saving book: To Kill a Mockingbird
```

Exercise 4: Creating and Configuring a Maven Project

```
<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 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>
</dependencies>

<build>
    <plugins>
        <!-- Maven Compiler Plugin -->
        <plugin>
```

```
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
               <source>1.8</source>
               <target>1.8</target>
            </configuration>
         </plugin>
      </plugins>
   </build>

</project>
```

Output:

```
mvn clean install
 Scanning for projects...
 Compiling 1 source file to target/classes
 BUILD SUCCESS
```

Difference between JPA, Hibernate and Spring Data JPA

Feature	JPA (Java Persistence API)	Hibernate	Spring Data JPA
Type	Specification (JSR 338)	Implementation (ORM Tool)	Abstraction layer over JPA (provided by Spring)
Implements	Interface only (no implementation)	Implements JPA + adds more ORM features	Uses JPA provider (like Hibernate) underneath
Boilerplate Code	Requires writing entity manager code	Requires boilerplate session and transaction	Reduces boilerplate with auto CRUD methods
Configuration	Needs manual configuration	Needs more config than Spring	Spring Boot auto-configures with annotations
Transaction Management	Manual or programmatic	Manual / programmatic	Handled declaratively via `@Transactional`
Use in Spring Boot	Indirectly used	Often used as default JPA implementation	Directly used for simplified repository-based access

Spring Data JPA - Quick Example

```
spring.datasource.url=jdbc:mysql://localhost:3306/springdata
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
package com.example.springdatajpa.model;

import jakarta.persistence.*;

@Entity
public class Employee {

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

    private String name;
    private String department;

    // 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 getDepartment() { return department; }
    public void setDepartment(String department) { this.department = department; }
}
package com.example.springdatajpa.repository;

import com.example.springdatajpa.model.Employee;
import org.springframework.data.jpa.repository.JpaRepository;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {
}
package com.example.springdatajpa;

import com.example.springdatajpa.model.Employee;
import com.example.springdatajpa.repository.EmployeeRepository;
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 SpringDataJpaApplication implements CommandLineRunner {

    @Autowired
    private EmployeeRepository employeeRepository;
```

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

    @Override
    public void run(String... args) {
        // Create and save employee
        Employee emp = new Employee();
        emp.setName("John Doe");
        emp.setDepartment("HR");
        employeeRepository.save(emp);

        // Fetch and print
        employeeRepository.findAll().forEach(e ->
            System.out.println("Employee: " + e.getName() + ", Dept: " + e.getDepartment()));
    }
}
```

Output:

Employee: John Doe, Dept: HR

Additional Hands on:

Exercise 5: Configuring the Spring IoC Container

```
<?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">

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

   <!-- Define BookService bean with dependency injected -->
   <bean id="bookService" class="com.library.service.BookService">
      <property name="bookRepository" ref="bookRepository" />
   </bean>

</beans>
package com.library.repository;

public class BookRepository {
   public void saveBook(String title) {
      System.out.println("Book saved: " + title);
   }
}
```

```
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {
    private BookRepository bookRepository;

    // Setter for DI
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

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

Output:

```
Adding book: The Alchemist
Book saved: The Alchemist
```

Exercise 7: Implementing Constructor and Setter Injection

```
package com.library.repository;

public class BookRepository {
    public void save(String title) {
        System.out.println("Book saved: " + title);
    }
}
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {
    private BookRepository bookRepository;
    private String serviceName;

    // Constructor for constructor injection
    public BookService(String serviceName) {
        this.serviceName = serviceName;
    }

    // Setter for setter injection
```

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

    public void addBook(String title) {
        System.out.println("[" + serviceName + "] Adding book: " + title);
        bookRepository.save(title);
    }
}
<?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 for BookRepository -->
    <bean id="bookRepository"
class="com.library.repository.BookRepository" />

    <!-- Bean for BookService with both constructor and setter injection -
->
```

```
    <bean id="bookService" class="com.library.service.BookService">
        <!-- Constructor Injection -->
        <constructor-arg value="LibraryServiceBean" />

        <!-- Setter Injection -->
        <property name="bookRepository" ref="bookRepository" />
    </bean>

</beans>
package com.library;

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

public class LibraryManagementApplication {
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");

        BookService bookService = context.getBean("bookService",
BookService.class);
```

```
        bookService.addBook("Wings of Fire");
    }
}
```

Output:

```
[LibraryServiceBean] Adding book: Wings of Fire
Book saved: Wings of Fire
```

Exercise 9: Creating a Spring Boot Application

```
# H2 DB Config
spring.datasource.url=jdbc:h2:mem:librarydb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# Auto-create tables
spring.jpa.hibernate.ddl-auto=update

# Enable H2 Console
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
package com.library.model;

import jakarta.persistence.*;

@Entity
public class Book {

    @Id
```

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

private String title;
private String author;

// Constructors
public Book() {}
public Book(String title, String author) {
   this.title = title;
   this.author = 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; }
```

```
}
package com.library.repository;

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

public interface BookRepository extends JpaRepository<Book, Long> {
}
package com.library.controller;

import com.library.model.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("/books")
public class BookController {

    @Autowired
```

```
    private BookRepository bookRepository;

    @GetMapping
    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }

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

    @GetMapping("/{id}")
    public Book getBook(@PathVariable Long id) {
        return bookRepository.findById(id).orElse(null);
    }

    @PutMapping("/{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);
    }
}
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:


```
[
  {
    "id": 1,
    "title": "The Alchemist",
    "author": "Paulo Coelho"
  }
]
```