

WEEK-3

Spring Core and Maven

Exercise 1: Configuring a Basic Spring Application

- Create a new maven project with group Id as com.library and artifact Id as LibraryManagement.
- Add dependencies in the pom.xml file.
- Create a package **com.library.service** and add a class **BookService**.
- Create a package **com.library.repository** and add a class **BookRepository**.
- Create an ApplicationContext.xml in the resources directory.
- Create MainApp.java to test the configurations.

BookRepository.java

```
package com.library.repository;
public class BookRepository {
    public void displayRepository() {
        System.out.println("BookRepository is working.");
    }
}
```

BookService.java

```
package com.library.service;
public class BookService {
    public void displayService() {
        System.out.println("BookService is working.");
    }
}
```

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
        https://www.springframework.org/schema/beans/spring-beans.xsd">
    <bean id="bookRepository" class="com.library.repository.BookRepository" />
    <bean id="bookService" class="com.library.service.BookService" />
</beans>
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.library</groupId>
    <artifactId>LibraryManagement</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

```

<dependencies>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-context</artifactId>
  <version>5.3.32</version>
</dependency>
</dependencies>
</project>

```

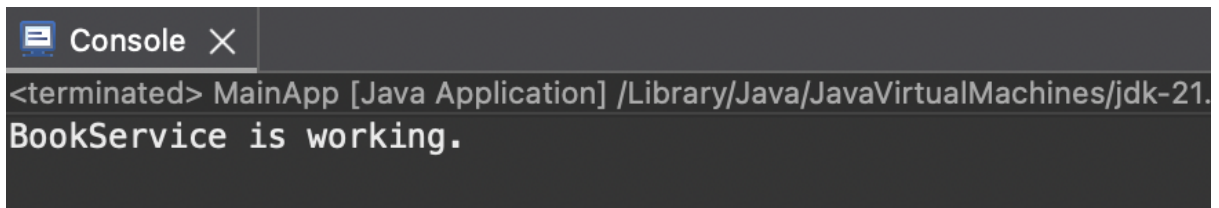
MainApp.java

```

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

```

Output:



The screenshot shows a console window titled "Console" with a close button. The output text is:


```
<terminated> MainApp [Java Application] /Library/Java/JavaVirtualMachines/jdk-21.
BookService is working.
```

Exercise 2: Implementing Dependency Injection

- Update the files of the above maven 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">
  <bean id="bookRepository" class="com.library.repository.BookRepository" />
  <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 displayBooks() {
        System.out.println("BookRepository: Displaying list of books...");
    }
}
```

BookService.java

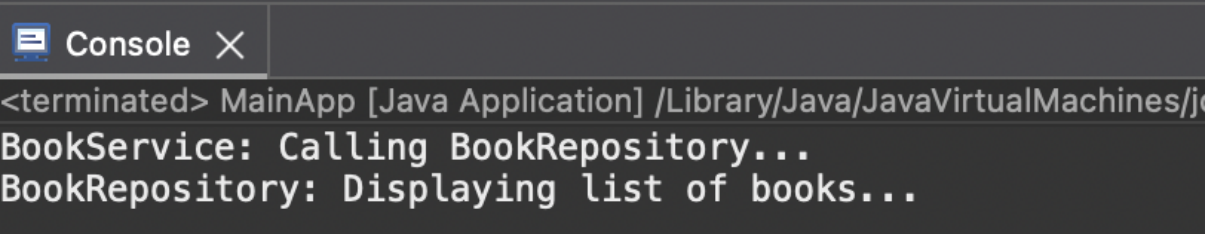
```
package com.library.service;
import com.library.repository.BookRepository;
public class BookService {
    private BookRepository bookRepository;
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
    public void listBooks() {
        System.out.println("BookService: Calling BookRepository...");
        bookRepository.displayBooks();
    }
}
```

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

Output:



The screenshot shows a console window titled "Console" with a close button. The output text is as follows:

```
<terminated> MainApp [Java Application] /Library/Java/JavaVirtualMachines/j
BookService: Calling BookRepository...
BookRepository: Displaying list of books...
```

Exercise 4: Creating and Configuring a Maven Project

- Add dependencies of Spring Context, Spring AOP, and Spring WebMVC.
- Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

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
    https://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 Context -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.22</version>
    </dependency>
    <!-- Spring AOP -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aop</artifactId>
      <version>5.3.22</version>
    </dependency>
    <!-- Spring Web MVC -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.3.22</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>
```

Spring Data JPA - Quick Example

- Create a new maven project and add dependencies in pom.xml file.
- Add controller, repository and model packages in the src/main/java directory.

SpringJpaApplication.java

```
package com.example.springjpa;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class SpringJpaApplication {
    public static void main(String[] args) {
        SpringApplication.run(SpringJpaApplication.class, args);
    }
}
```

Book.java

```
package com.example.springjpa.model;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;
    public Long getId() {
        return id;
    }
    public String getTitle() {
        return title;
    }
    public String getAuthor() {
        return author;
    }
    public void setId(Long id) {
        this.id = id;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public void setAuthor(String author) {
        this.author = author;
    }
}
```

BookRepository.java

```
package com.example.springjpa.repository;
```

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

BookController.java

```
package com.example.springjpa.controller;
import com.example.springjpa.model.Book;
import com.example.springjpa.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;
    @PostMapping
    public Book addBook(@RequestBody Book book) {
        return bookRepository.save(book);
    }
    @GetMapping
    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }
}
```

application.properties

```
server.port=9090
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
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.jpa.hibernate.ddl-auto=update
spring.sql.init.mode=always
spring.jpa.defer-datasource-initialization=true
```

Output:

The output of table “BOOK” can be seen in the h2 console.

The screenshot shows the H2 database console interface. On the left, a tree view displays the database structure, including the 'BOOK' table with columns 'ID', 'AUTHOR', and 'TITLE'. The main area shows the SQL statement 'SELECT * FROM BOOK BOOK;' and its execution result, which is an empty table with 3 columns (ID, AUTHOR, TITLE) and 0 rows, taking 3 ms to execute. The interface includes a toolbar with buttons for 'Run', 'Run Selected', 'Auto complete', and 'Clear', as well as a status bar at the bottom with an 'Edit' button.

Difference between JPA, Hibernate and Spring Data JPA

JPA: JPA, stands for Jakarta Persistence API is a specification (just a set of interfaces and annotations) that defines how Java objects interact with relational databases. It does not provide the actual implementation, it just tells what it does. It needs a provider like Hibernate. JPA uses Object Relational Mapping which provides a standard specification that defines how to map Java objects to database tables and provides a consistent API for database operations. It uses annotations like `@Entity`, `@Id`, and `@Column` to map Java classes to tables. JPA is commonly used in Enterprise Java Beans (EJBs), web applications, Java SE applications, and desktop applications.

Example:

```
@Entity
public class Book {
    @Id
    private Long id;
}
```

Hibernate: Hibernate is an open-source Object/Relational Mapping (ORM) framework for the Java programming language. It simplifies the interaction between Java applications and relational databases by providing a framework for mapping an object-oriented domain model to a relational database. A concrete library that executes JPA instructions. It can be directly or through JPA.

Example:

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("myPU");
EntityManager em = emf.createEntityManager();
em.getTransaction().begin();

Book book = new Book();
book.setTitle("JPA Basics");
em.persist(book);
em.getTransaction().commit();
em.close();
```

Spring Data JPA: A Spring module that makes working with JPA even easier especially when working with repositories. It significantly reduces the amount of boilerplate code typically required for implementing data access objects (DAOs) by providing an abstraction layer over JPA. It cannot work as a standalone framework. It requires Spring and JPA. Developers define repository interfaces by extending Spring Data JPA interfaces like `JpaRepository` or `CrudRepository`. Spring Data JPA then automatically generates the implementation for these interfaces at runtime, handling common CRUD (Create, Read, Update, Delete) operations.

Example:

```
public interface BookRepository extends JpaRepository<Book, Long> {}

@Autowired
private BookRepository bookRepo;

public void saveBook() {
    Book book = new Book();
    book.setTitle("Spring JPA");
    bookRepo.save(book); // No boilerplate code needed
}
```