**WEEK 3**

**Naren Karthick A**

**St. Joseph's Institute of Technology**

**Superset ID: 6377326**

# Spring core and maven

**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.**

**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 bookService = (BookService) context.getBean("bookService");
bookService.addBook("The Harry Potter");


                ((ClassPathXmlApplicationContext) context).close();
            }
}
```

**BookRepository.java** package

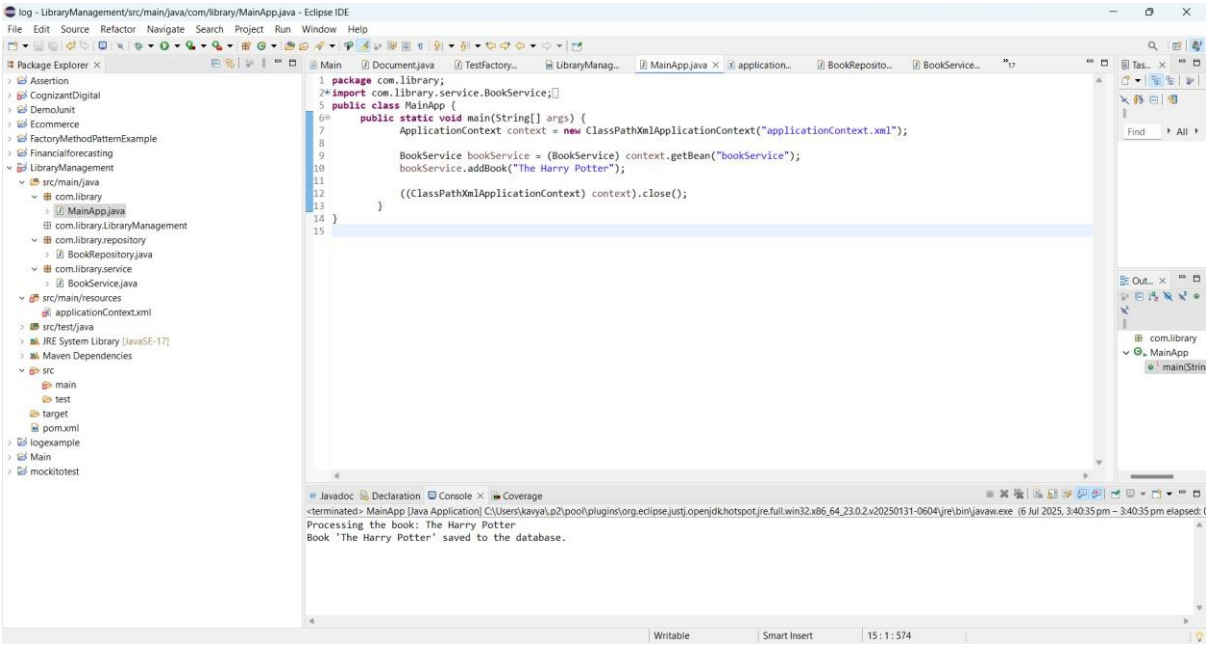
```
com.library.repository; public class BookRepository
{           public void saveBook(String bookName) {
                System.out.println("Book '" + bookName + "' saved to the database.");
            }
}
```

**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("Processing the book: " + bookName);
bookRepository.saveBook(bookName);
        }
}
```

**Output:**




## 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. MainApp.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("Processing the book: " + bookName);
bookRepository.saveBook(bookName);
   }
}
```

**BookRespository.java** package

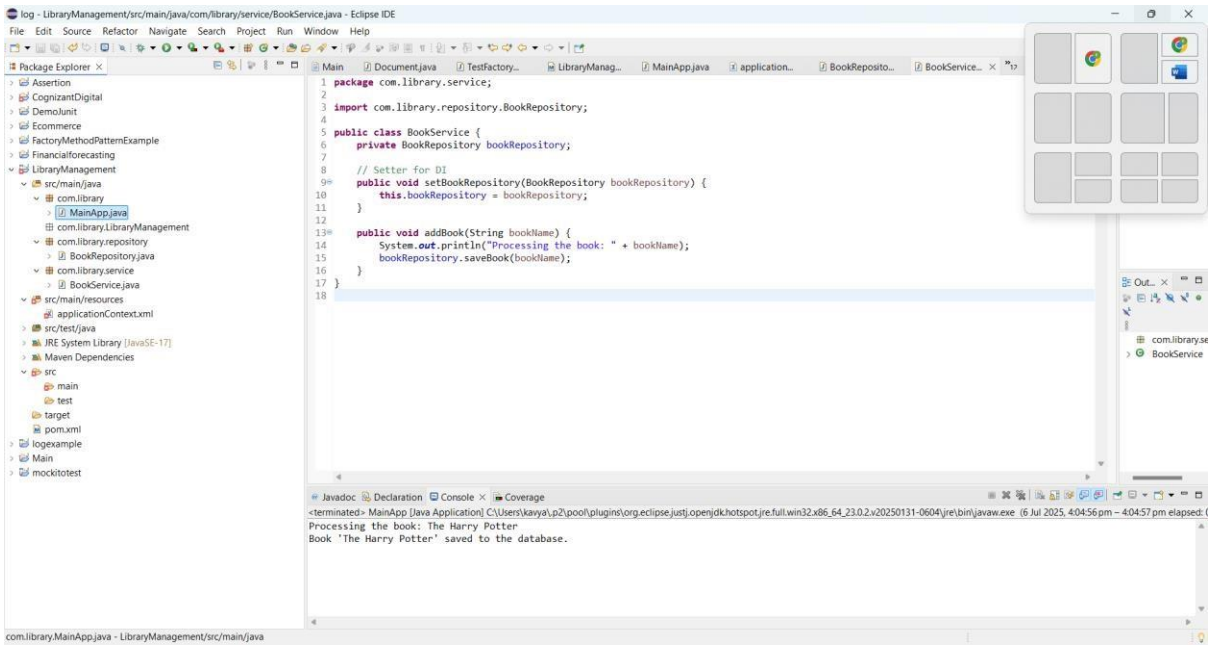
```
com.library.repository; public class
BookRepository {        public void
saveBook(String bookName) {
                System.out.println("Book '" + bookName + "' saved to the database.");
            }
}
```

**BookService.java** package

```
com.library.service; import
com.library.repository.Book
Repository; public class
BookService {     private
BookRepository
bookRepository;     public
void
```

```
setBookRepository(BookRe
pository bookRepository) {
this.bookRepository =
bookRepository;
    }
    public void addBook(String bookName) {
        System.out.println("Processing the book: " + bookName);
bookRepository.saveBook(bookName);
    }
}
```

**Output:**




**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.**

**Code:**

**MainApp.java** package com.example; import

org.springframework.context.ApplicationContext; import

org.springframework.context.support.ClassPathXmlApplicationContext; public

class MainApp { public static void main(String[] args) {

System.*out*.println("Starting Spring Application...");

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

System.*out*.println("Spring context loaded successfully!");

MessageService service = (MessageService) context.getBean("messageService");

System.*out*.println("Retrieved bean: " + service.getClass().getSimpleName());

service.printMessage();

}

}

**MessageService.java** package

com.example; public class

MessageService { public void

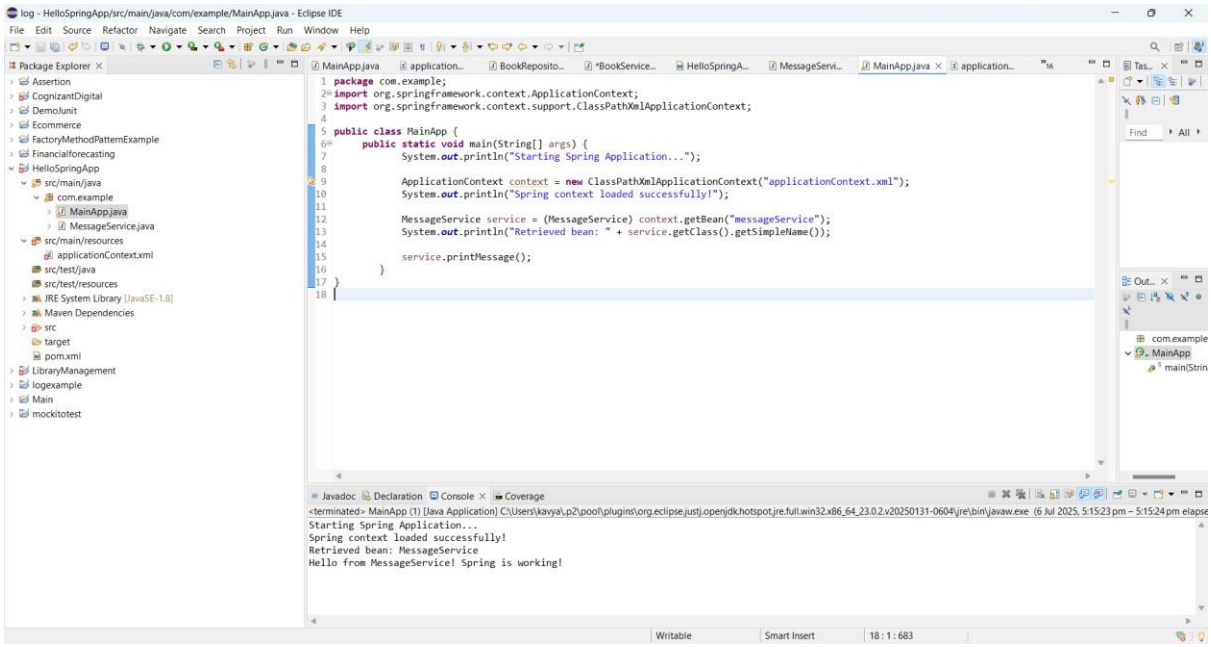
printMessage() {

System.*out*.println("Hello from MessageService! Spring is working!");

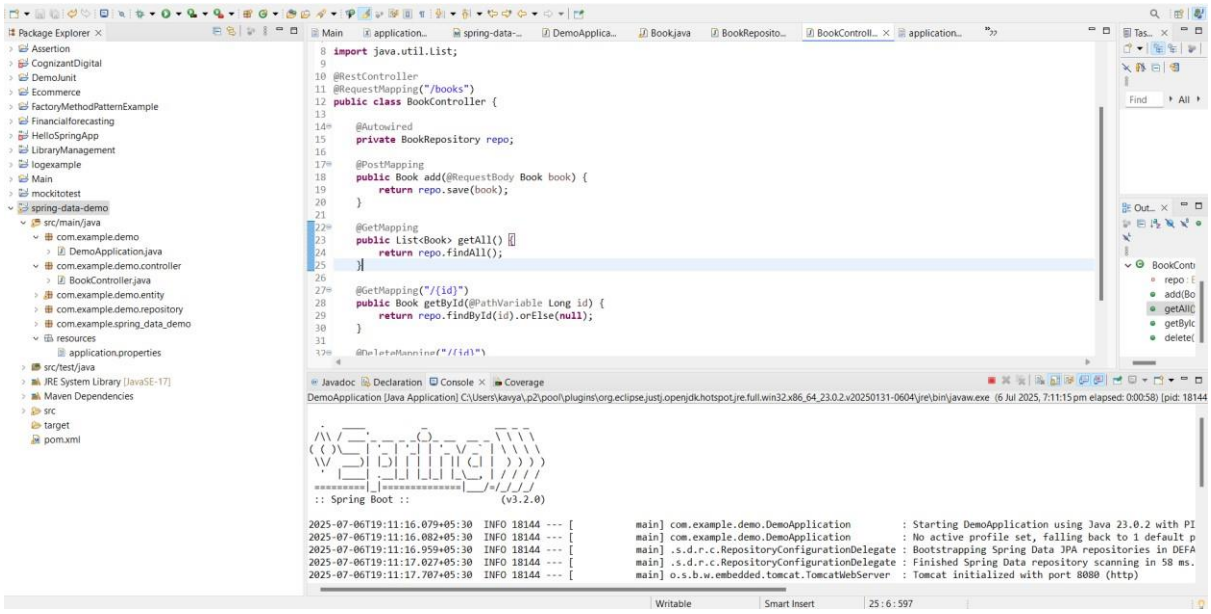
}

}

**Output:**




## Spring Data JPA - Quick Example




## Difference between JPA, Hibernate and Spring Data JPA

### 1. JPA (Java Persistence API)

- It is a specification (interface) provided by Java for ORM (Object-Relational Mapping).
- JPA provides standard APIs for managing relational data in Java applications.
- It does not provide implementation, only guidelines.

- Needs a provider (like Hibernate, EclipseLink) to work.
- Focuses on entity mapping, query language (JPQL), and transactions.
- Example annotation: @Entity, @Id, @GeneratedValue.

**2. Hibernate**

- It is a JPA implementation and a powerful ORM framework.
- It provides all features required by JPA plus extra features like:
    - Caching ○
      Lazy loading ○
      Batch processing
- Supports native Hibernate APIs (like Session) in addition to JPA.
- Can be used with or without Spring.
- Has its own query language called HQL (Hibernate Query Language).

**3. Spring Data JPA**

- It is a Spring project that simplifies the use of JPA in Spring apps.
- It builds on top of JPA and Hibernate.
- Reduces boilerplate by providing pre-built repositories like JpaRepository.
- Supports query method names, custom JPQL, and @Query annotations.
- Automatically implements CRUD operations and supports pagination and sorting.
- Great for rapid development of data access layers.