WEEK 3

Kavya V.R

St. Joseph’s Institute of Technology

Superset ID: 6377320

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();

        }

}
```

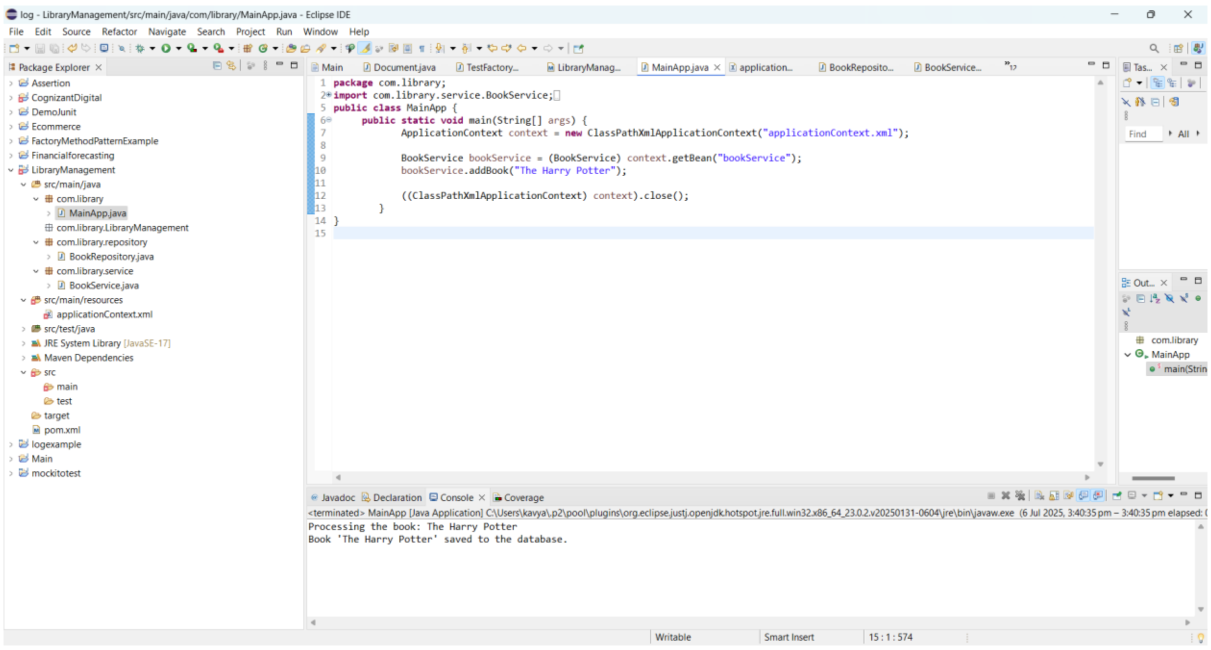
BookRepository.java

```
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:



```
package com.library;
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 Harry Potter");

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

```
<terminated> MainApp [Java Application] C:\Users\kavya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe  (6 Jul 2025, 3:40:35 pm – 3:40:35 pm elapsed:
Processing the book: The Harry Potter
Book 'The Harry Potter' saved to the database.
```

}

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

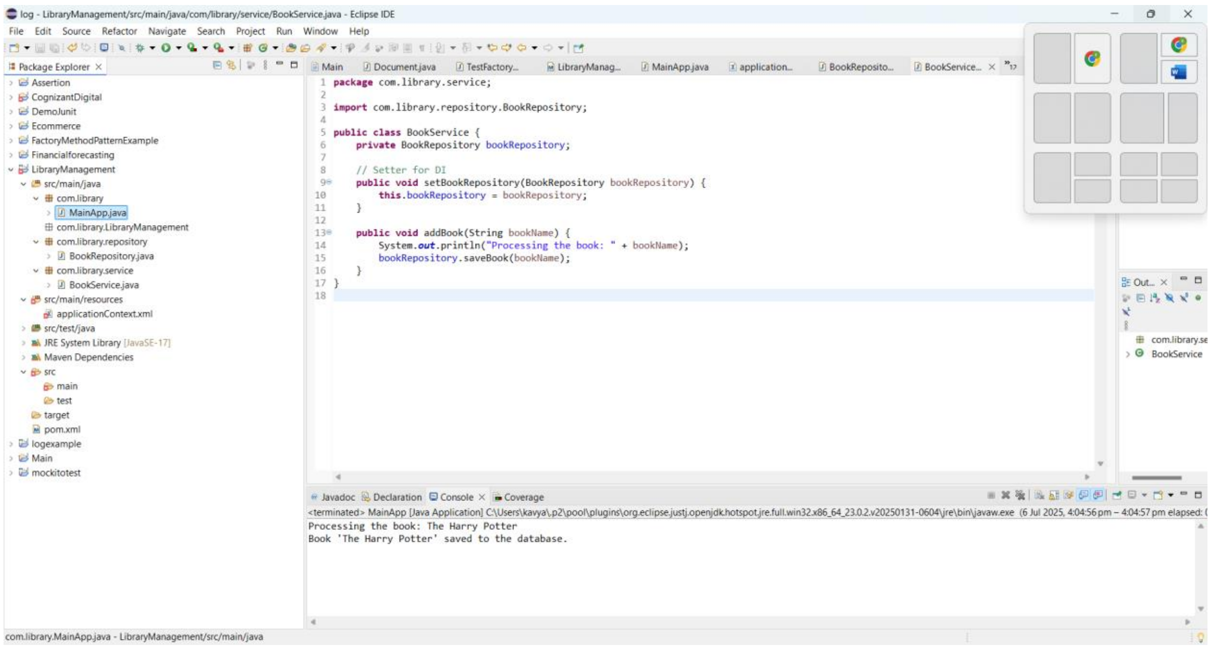
```
package com.library.repository;

public class BookRepository {

            public void saveBook(String bookName) {

                System.out.println("Book '" + bookName + "' saved to the database.");

              }

}
```

BookService.java

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

```java
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:



```java
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 bookName) {
        System.out.println("Processing the book: " + bookName);
        bookRepository.saveBook(bookName);
    }
}
```

```
<terminated> MainApp [Java Application] C:\Users\kavya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe  (6 Jul 2025, 4:04:56 pm – 4:04:57 pm elapsed:
Processing the book: The Harry Potter
Book 'The Harry Potter' saved to the database.
```

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:

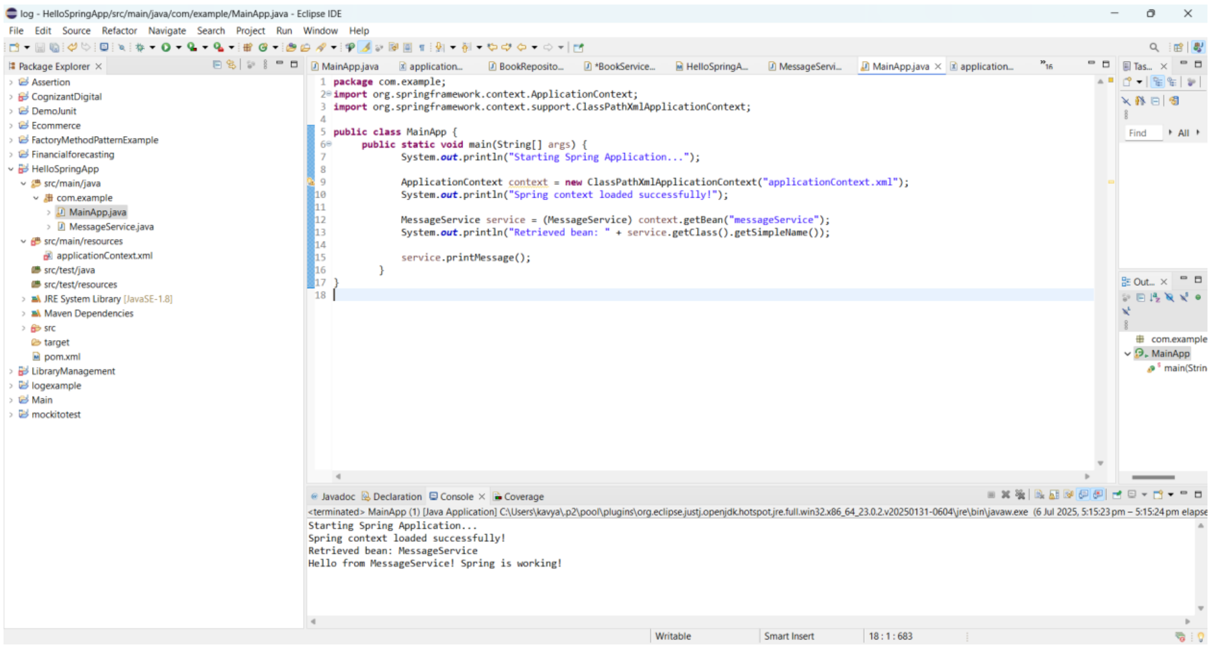
MainApp.java

```
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!");

          }

}
```

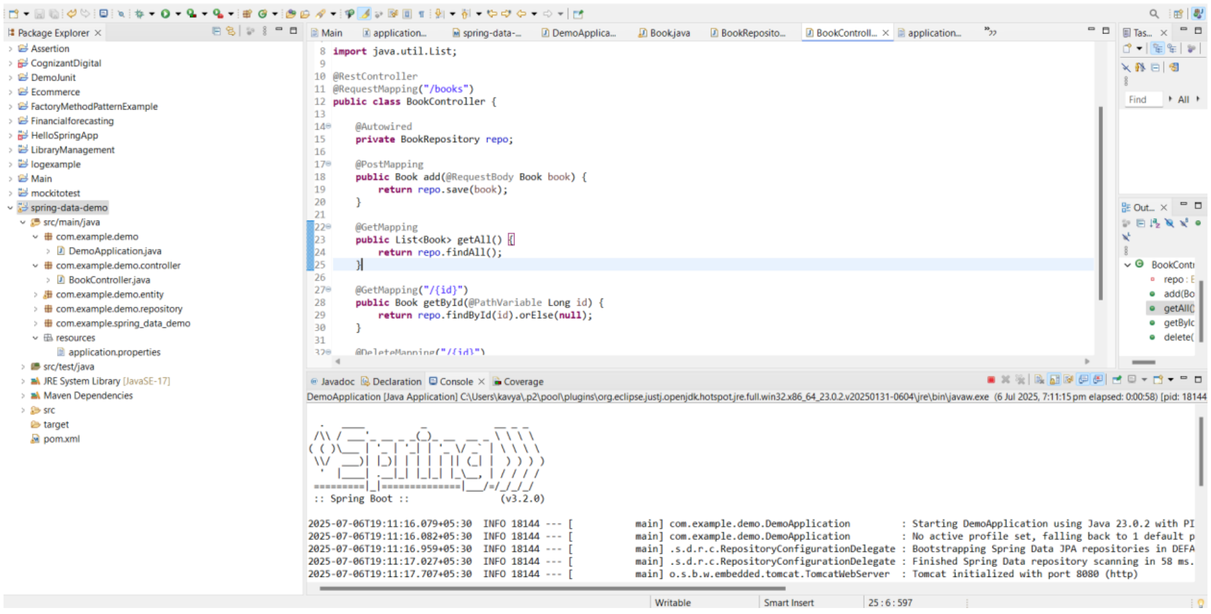
Output:



```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();
        }
}
```

```
<terminated> MainApp (1) [Java Application] C:\Users\kavya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe (6 Jul 2025, 5:15:23 pm – 5:15:24 pm elapse
Starting Spring Application...
Spring context loaded successfully!
Retrieved bean: MessageService
Hello from MessageService! Spring is working!
```

Spring Data JPA - Quick Example



```java
import java.util.List;

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

    @Autowired
    private BookRepository repo;

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

    @GetMapping
    public List<Book> getAll() {
        return repo.findAll();
    }

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

    @DeleteMapping("/{id}")
```

```
DemoApplication [Java Application] C:\Users\kavya\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe (6 Jul 2025, 7:11:15 pm elapsed: 0:00:58) [pid: 18144

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.0)

2025-07-06T19:11:16.079+05:30  INFO 18144 --- [           main] com.example.demo.DemoApplication         : Starting DemoApplication using Java 23.0.2 with PI
2025-07-06T19:11:16.082+05:30  INFO 18144 --- [           main] com.example.demo.DemoApplication         : No active profile set, falling back to 1 default p
2025-07-06T19:11:16.959+05:30  INFO 18144 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFA
2025-07-06T19:11:17.027+05:30  INFO 18144 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 58 ms.
2025-07-06T19:11:17.707+05:30  INFO 18144 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
```

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.