

WEEK 3

Taniya G Remula

St. Joseph's Institute of Technology

Superset ID: 6376013

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:

LibraryApp.java

Share

Run

Output

Clear

```

1+ public class LibraryApp {
2
3    // Repository class
4+ static class BookRepository {
5+     public void saveBook(String bookName) {
6         System.out.println("Book '" + bookName + "' saved to the
           database.");
7     }
8 }
9
10 // Service class
11+ static class BookService {
12     private BookRepository bookRepository;
13
14+     public void setBookRepository(BookRepository bookRepository)
15     {
16         this.bookRepository = bookRepository;
17     }
18+
19     public void addBook(String bookName) {
20         System.out.println("Processing the book: " + bookName);
21         bookRepository.saveBook(bookName);
22     }
23 }

```

```

Processing the book: The Harry Potter
Book 'The Harry Potter' saved to the database.

=== Code Execution Successful ===

```

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);

    }

}
```

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;
```

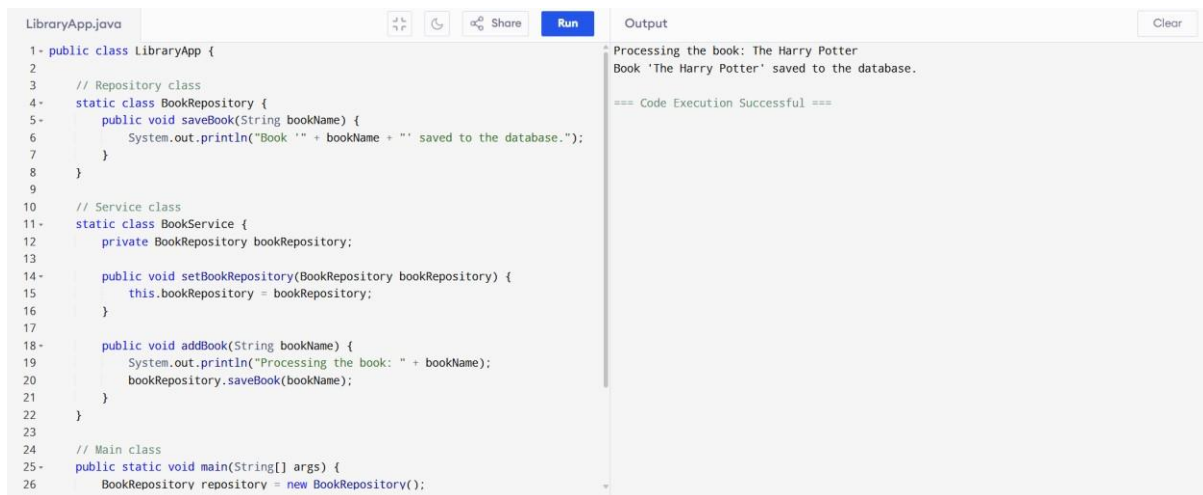
```

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

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

```

Output:



The screenshot shows an IDE window titled 'LibraryApp.java'. The code defines three classes: `LibraryApp`, `BookRepository`, and `BookService`. `BookRepository` has a `saveBook` method that prints the book name and 'saved to the database.'. `BookService` has `setBookRepository` and `addBook` methods. `LibraryApp` has a `main` method that creates a `BookRepository` instance and calls `addBook` with 'The Harry Potter'. The output pane on the right shows the execution results: 'Processing the book: The Harry Potter', 'Book 'The Harry Potter' saved to the database.', and '=== Code Execution Successful ==='.

```

LibraryApp.java
1- public class LibraryApp {
2-
3-     // Repository class
4-     static class BookRepository {
5-         public void saveBook(String bookName) {
6-             System.out.println("Book '" + bookName + "' saved to the database.");
7-         }
8-     }
9-
10-    // Service class
11-    static class BookService {
12-        private BookRepository bookRepository;
13-
14-        public void setBookRepository(BookRepository bookRepository) {
15-            this.bookRepository = bookRepository;
16-        }
17-
18-        public void addBook(String bookName) {
19-            System.out.println("Processing the book: " + bookName);
20-            bookRepository.saveBook(bookName);
21-        }
22-    }
23-
24-    // Main class
25-    public static void main(String[] args) {
26-        BookRepository repository = new BookRepository();
    }
}

```

Output

```

Processing the book: The Harry Potter
Book 'The Harry Potter' saved to the database.

=== Code Execution Successful ===

```

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() {

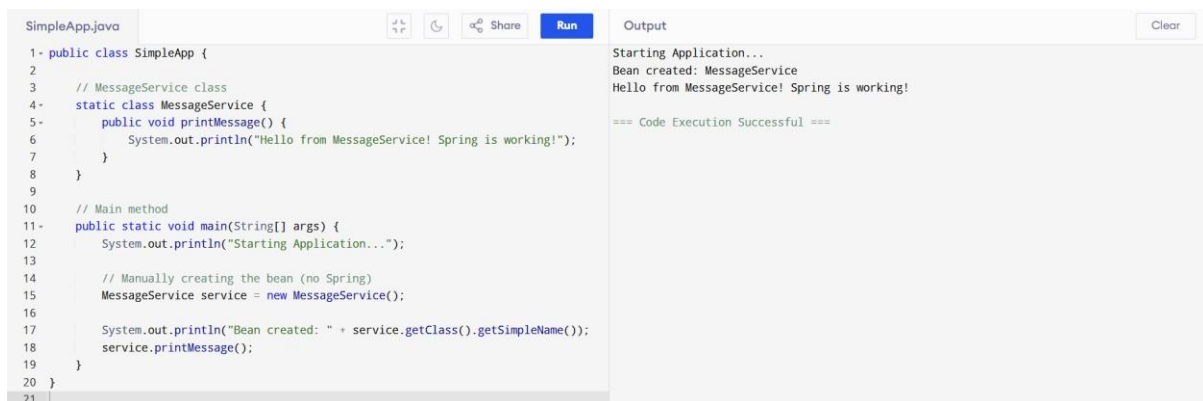
        System.out.println("Hello from MessageService! Spring is working!");

    }

}

```

Output:



```

SimpleApp.java
1- public class SimpleApp {
2
3    // MessageService class
4-    static class MessageService {
5-        public void printMessage() {
6            System.out.println("Hello from MessageService! Spring is working!");
7        }
8    }
9
10   // Main method
11-   public static void main(String[] args) {
12       System.out.println("Starting Application...");
13
14       // Manually creating the bean (no Spring)
15       MessageService service = new MessageService();
16
17       System.out.println("Bean created: " + service.getClass().getSimpleName());
18       service.printMessage();
19   }
20 }
21

```

Output

```

Starting Application...
Bean created: MessageService
Hello from MessageService! Spring is working!

=== Code Execution Successful ===

```

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:
 - o Caching
 - o Lazy loading
 - o 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.

