**Digital Nurture 4.0 – Week 3**

1.Spring Core And Maven

Hands on exercise

**Exercise 1: Configuring a Basic Spring Application**

**Program:**

This program demonstrates a simple Spring-based application for managing books in a library. The BookService and BookRepository classes are configured as beans in an XML file, showing how dependency injection works in Spring. Finally, a main class loads the Spring context to retrieve and test these beans, simulating the addition of a book.

**CODE:**

*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 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>  
 <packaging>jar</packaging>  
  
 <name>LibraryManagement</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.22</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 = context.getBean("bookService", BookService.class);  
 service.displayBook();  
 }  
}

*BookRepository.java:*

package com.library.repository;  
  
public class BookRepository {  
 public void getBook() {  
 System.*out*.println("Fetching book from repository...");  
 }  
}

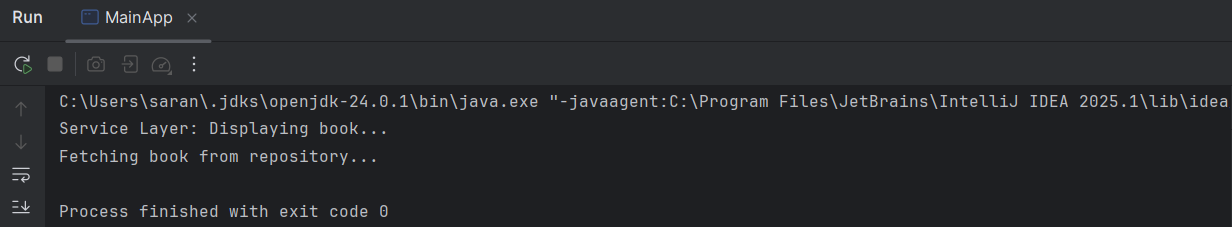
*BookService.java:*

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 displayBook() {  
 System.*out*.println("Service Layer: Displaying book...");  
 bookRepository.getBook();  
 }  
}

*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>

**Output:**

****

**Exercise 2: Implementing Dependency Injection:**

**Program:**

This exercise demonstrates dependency injection in a Spring-based library management system. The BookService class depends on BookRepository, and Spring injects this dependency through XML configuration using setter injection. When the main application runs, it verifies that the BookService can successfully call the BookRepository methods through the injected dependency.

**CODE:**

*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 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>  
 <packaging>jar</packaging>  
  
 <name>LibraryManagement</name>  
 <url>http://maven.apache.org</url>  
  
 <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.22</version>  
 </dependency>  
 </dependencies>  
</project>

*LibraryManagementApplication.xml:*

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 = (BookService) context.getBean("bookService");  
 bookService.addBook("Dependency Injection with Spring");  
 }  
}

*BookRepository.java:*

package com.library.repository;  
  
public class BookRepository {  
 public void saveBook(String name) {  
 System.*out*.println("Book saved: " + name);  
 }  
}

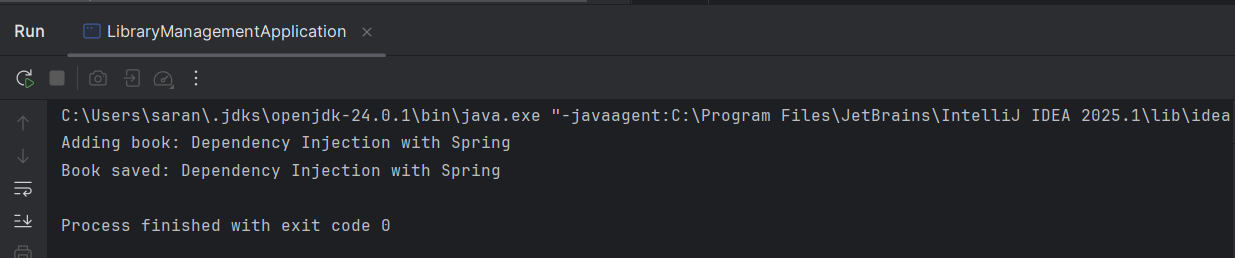
*BookService.java:*

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 name) {  
 System.*out*.println("Adding book: " + name);  
 bookRepository.saveBook(name);  
 }  
}

*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">  
 <property name="bookRepository" ref="bookRepository" />  
 </bean>  
</beans>

**OUTPUT:**



**Exercise 4: Creating and Configuring a Maven Project:**

**Program:**

This Spring program defines a GreetingService bean configured in applicationContext.xml to demonstrate basic dependency injection. The GreetingApp class loads the Spring context and retrieves the bean to print a personalized greeting. It confirms that Spring Core and Maven are correctly configured and working with XML-based bean definitions

**CODE:**

*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  
 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>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.22</version>  
 </dependency>  
   
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.22</version>  
 </dependency>  
   
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.22</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.10.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

*GreetingApp.java:*

package com.library;  
  
import com.library.service.GreetingService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class GreetingApp {  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 GreetingService greetingService = (GreetingService) context.getBean("greetingService");  
 greetingService.greet("Saranya");  
 }  
}

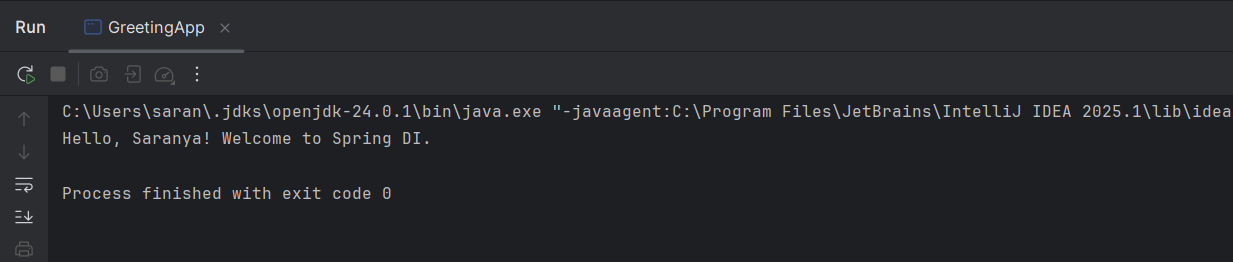
*GreetingService.java:*

package com.library.service;  
  
public class GreetingService {  
 public void greet(String name) {  
 System.*out*.println("Hello, " + name + "! Welcome to Spring DI.");  
 }  
}

*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="greetingService" class="com.library.service.GreetingService" />  
  
</beans>

**OUTPUT:**

****

Additional exercise

**Exercise 5: Configuring the Spring IoC Container:**

**Program:**

This exercise shows how to configure the Spring IoC container for the library management system using an XML configuration file. The BookService and BookRepository beans are defined in applicationContext.xml, with dependency injection handled through setter injection. Finally, a main class loads the Spring context to verify that the beans and dependencies are properly wired and functioning**.**

**CODE:**

*LibraryManagementApp.java:*

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

*Bookrepository.java:*

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

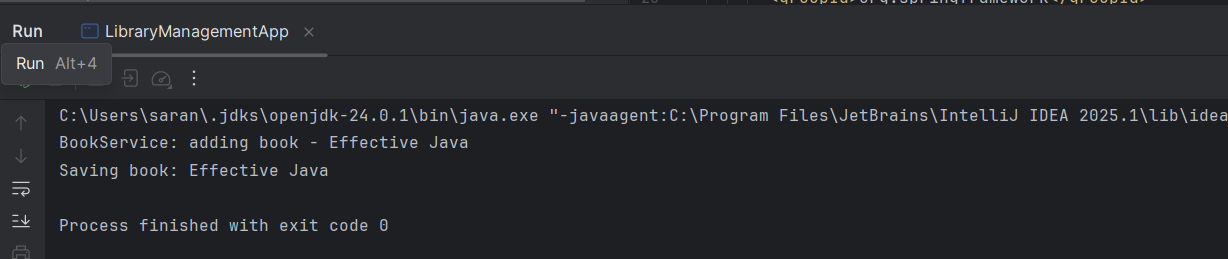
*BookService.java:*

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 name) {  
 System.*out*.println("BookService: adding book - " + name);  
 bookRepository.save(name);  
 }  
}

*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">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
</beans>

**OUTPUT:**



**Exercise 7: Implementing Constructor and Setter Injection:**

**Program:**

This program shows how to use both constructor and setter injection on a single Spring bean. The BookService takes a BookRepository through its constructor and receives a bookCategory through a setter method. Spring XML configuration wires them together, demonstrating mixed injection techniques..

**CODE:**

*BookRepository.java:*

package com.library.repository;

public class BookRepository {

public void save(String bookName) {

System.out.println("Saving book: " + bookName);

}

}

*BookService.java:*

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

private String bookCategory;

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void setBookCategory(String bookCategory) {

this.bookCategory = bookCategory;

}

public void addBook(String name) {

System.out.println("Category: " + bookCategory);

bookRepository.save(name);

}

}

*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">

<!-- constructor injection -->

<constructor-arg ref="bookRepository"/>

<!-- setter injection -->

<property name="bookCategory" value="Programming"/>

</bean>

</beans>

*LibraryManagementApplication.java:*

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");

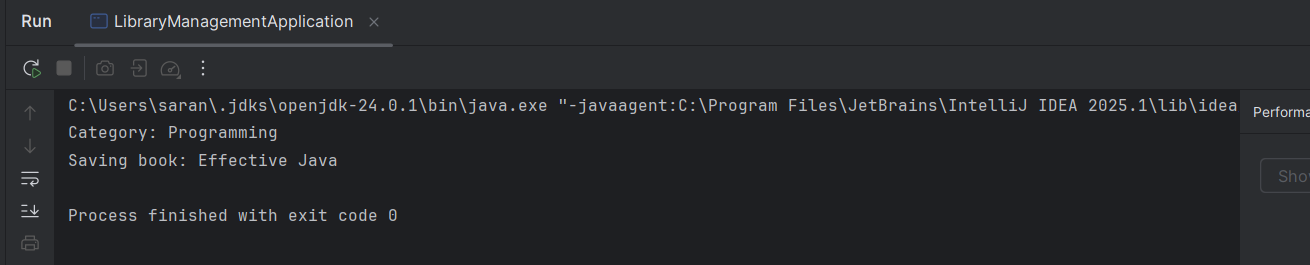
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("Effective Java");

}

}

**OUTPUT:**

****

**Exercise 9: Creating a Spring Boot Application:**

**Program:**

This exercise uses Spring Boot to modernize the library management system, making configuration and deployment simpler. By integrating Spring Web, Spring Data JPA, and H2 in-memory database, you build a REST API to manage books. The Spring Boot application enables quick testing of CRUD operations through a BookController**.**

**CODE:**

*Pom.xml:*

<?xml version="1.0" encoding="UTF-8"?>  
<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>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.2.4</version>  
 <relativePath/>   
 </parent>  
  
 <properties>  
 <java.version>17</java.version> <!-- or 1.8 if you use Java 8 -->  
 </properties>  
  
 <dependencies>  
   
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
  
   
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
  
   
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>

*LibraryManagementApplication.java:*

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);  
 }  
}

*BookRepository.java:*

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

*Book.java:*

package com.library.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 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; }  
}

*BookController.java:*

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 getBookById(@PathVariable Long id) {  
 return bookRepository.findById(id).orElse(null);  
 }  
  
 @PutMapping("/{id}")  
 public Book updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {  
 Book book = bookRepository.findById(id).orElse(null);  
 if (book != null) {  
 book.setTitle(bookDetails.getTitle());  
 book.setAuthor(bookDetails.getAuthor());  
 return bookRepository.save(book);  
 }  
 return null;  
 }  
  
 @DeleteMapping("/{id}")  
 public void deleteBook(@PathVariable Long id) {  
 bookRepository.deleteById(id);  
 }  
}

*application .properties:*

spring.datasource.url=jdbc:h2:mem:cts  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=root  
spring.datasource.password=Saran81  
spring.jpa.hibernate.ddl-auto=update  
spring.h2.console.enabled=true

**SQL QUERY:**

CREATE TABLE book (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

title VARCHAR(255),

author VARCHAR(255)

);

INSERT INTO book (title, author)

VALUES ('Spring Boot in Action', 'Craig Walls');

INSERT into book

values(1,"java", "xx");

SELECT \* FROM book;

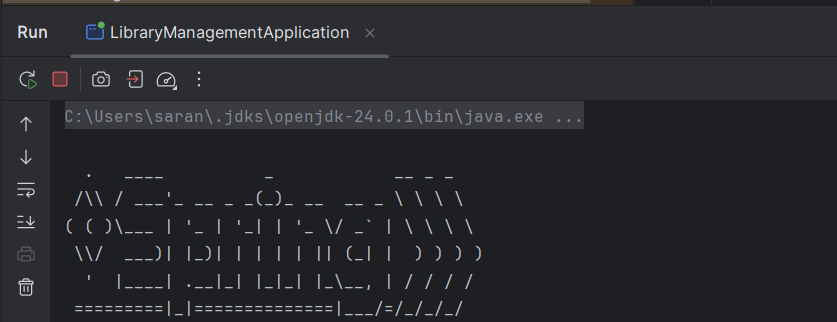
INSERT into book

values(2,"C", "aaa");

SELECT \* FROM book;

SELECT \* FROM book WHERE id = 1;

**OUTPUT:**

****

