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

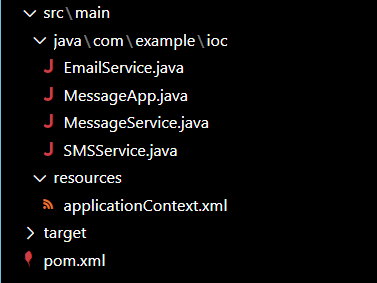
**Scenario:**

The library management application requires a central configuration for beans and dependencies.

**Steps:**

1. **Create Spring Configuration File:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   * Define beans for **BookService** and **BookRepository** in the XML file.
2. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.
3. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**Folder Structure:**

****

**Code:**

**MessageApp.java:**package com.example.ioc;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MessageApp {

    public static void main(String[] args) {

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

        MessageService messageService = context.getBean("emailService", MessageService.class);

        messageService.sendMessage("Hello via Spring IoC!");

    }

}

**SMSService.java**:

package com.example.ioc;

public class SMSService implements MessageService {

    @Override

    public void sendMessage(String message) {

        System.out.println("SMS: " + message);

    }

}

**EmailService.java:**

package com.example.ioc;

public class EmailService implements MessageService {

    @Override

    public void sendMessage(String message) {

        System.out.println("Email: " + message);

    }

}

**MessageService.java:**

package com.example.ioc;

public interface MessageService {

    void sendMessage(String message);

}

**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.example</groupId>

    <artifactId>spring-ioc-demo</artifactId>

    <version>1.0-SNAPSHOT</version>

    <!-- Dependencies go here -->

    <dependencies>

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-context</artifactId>

            <version>5.3.29</version>

        </dependency>

    </dependencies>

    <!-- Build and Plugins go here -->

    <build>

        <plugins>

            <plugin>

                <groupId>org.codehaus.mojo</groupId>

                <artifactId>exec-maven-plugin</artifactId>

                <version>3.1.0</version>

                <configuration>

                    <mainClass>com.example.ioc.MessageApp</mainClass>

                </configuration>

            </plugin>

        </plugins>

    </build>

</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"

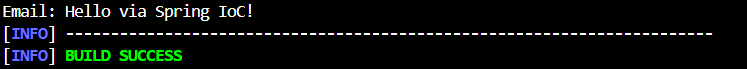
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="emailService" class="com.example.ioc.EmailService" />

    <bean id="smsService" class="com.example.ioc.SMSService" />

</beans>

**Output:**

****

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

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

**Steps:**

1. Configure Constructor Injection:
   * Update applicationContext.xml to configure constructor injection for BookService.
2. Configure Setter Injection:
   * Ensure that the BookService class has a setter method for BookRepository and configure it in applicationContext.xml.
3. Test the Injection:
   * Run the LibraryManagementApplication main class to verify both constructor and setter injection.

**Code:**

**applicationContext.xml (updated):**

<?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">

    <!-- Service -->

    <bean id="emailService" class="com.example.ioc.EmailService" />

    <!-- Setter Injection -->

    <bean id="setterInjectionDemo" class="com.example.ioc.SetterInjectionDemo">

        <property name="messageService" ref="emailService" />

    </bean>

    <!-- Constructor Injection -->

    <bean id="constructorInjectionDemo" class="com.example.ioc.ConstructorInjectionDemo">

        <constructor-arg ref="emailService" />

    </bean>

</beans>

**ConstructorInjectionDemo.java:**

package com.example.ioc;

public class ConstructorInjectionDemo {

    private MessageService messageService;

    public ConstructorInjectionDemo(MessageService messageService) {

        this.messageService = messageService;

    }

    public void send(String msg) {

        messageService.sendMessage(msg);

    }

}

**SetterInjectionDemo.java:**

package com.example.ioc;

public class SetterInjectionDemo {

    private MessageService messageService;

    public void setMessageService(MessageService messageService) {

        this.messageService = messageService;

    }

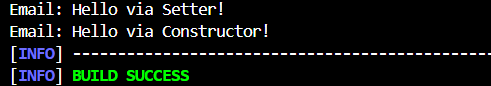
    public void send(String msg) {

        messageService.sendMessage(msg);

    }

}

**Output:**

****

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

**Scenario:**

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

**Steps:**

**1. Create a Spring Boot Project:**

Use Spring Initializr to create a new Spring Boot project named LibraryManagement.

**2. Add Dependencies:**

Include dependencies for Spring Web, Spring Data JPA, and H2 Database.

**3. Create Application Properties:**

Configure database connection properties in application.properties.

**4. Define Entities and Repositories:**

Create Book entity and BookRepository interface.

**5. Create a REST Controller:**

Create a BookController class to handle CRUD operations.

**6. Run the Application:**

Run the Spring Boot application and test the REST endpoints.

**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.example</groupId>

    <artifactId>spring-boot-basic</artifactId>

    <version>1.0-SNAPSHOT</version>

    <parent>

        <groupId>org.springframework.boot</groupId>

        <artifactId>spring-boot-starter-parent</artifactId>

        <version>2.7.18</version>

    </parent>

    <dependencies>

        <dependency>

            <groupId>org.springframework.boot</groupId>

            <artifactId>spring-boot-starter</artifactId>

        </dependency>

    </dependencies>

    <properties>

        <java.version>1.8</java.version>

    </properties>

</project>

**GreetingService.java:**

package com.example.boot;

import org.springframework.stereotype.Component;

@Component

public class GreetingService {

    public void greet() {

        System.out.println("Welcome to Spring Boot Application!");

    }

}

**BootApp.java:**package com.example.boot;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class BootApp implements CommandLineRunner {

    @Autowired

    private GreetingService greetingService;

    public static void main(String[] args) {

        SpringApplication.run(BootApp.class, args);

    }

@Override

    public void run(String... args) throws Exception {

        greetingService.greet();

    }

}

**Output:**

