Spring Core and Maven Hands-On Exercises

# Exercise 1: Configuring a Basic Spring Application

## Introduction to Spring Core

Spring Core is the foundational module of the Spring Framework that provides essential features like Inversion of Control (IoC) and Dependency Injection (DI). It allows developers to build loosely coupled applications that are easier to manage and test.  
  
The key concept in Spring Core is the Bean, which is a Java object managed by the Spring container.

## Objective of Exercise 1

In this exercise, we focus on setting up a basic Spring application using Spring Core. The purpose is to understand the core container, configure beans, and load them using an XML or Java-based configuration.

## Steps

1. Add Spring Core Dependency using Maven

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>6.1.4</version>  
</dependency>

2. Create a Bean Class

public class HelloWorld {  
 public void sayHello() {  
 System.out.println("Hello, Spring Core!");  
 }  
}

3. Configure XML (beans.xml)

<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="helloBean" class="HelloWorld"/>  
</beans>

4. Create Main Application

import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class MainApp {  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("beans.xml");  
 HelloWorld obj = (HelloWorld) context.getBean("helloBean");  
 obj.sayHello();  
 }  
}

## Explanation

- The HelloWorld class is defined as a Spring Bean.  
- beans.xml contains the configuration for the Spring context.  
- MainApp loads the context and retrieves the bean using its ID.

## Benefits

- Minimal configuration to get started.  
- Encourages loose coupling and easier testability.  
- Reduces boilerplate code using Spring-managed beans.

# Exercise 2: Implementing Dependency Injection

## Objective

To understand how Spring manages dependencies and injects them into beans using XML or annotations.

## Example

1. Create Dependent Classes

public class Address {  
 private String city;  
 public void setCity(String city) { this.city = city; }  
 public void display() { System.out.println("City: " + city); }  
}  
  
public class Employee {  
 private Address address;  
 public void setAddress(Address address) { this.address = address; }  
 public void showDetails() { address.display(); }  
}

2. Configure Beans in XML

<bean id="addressBean" class="Address">  
 <property name="city" value="Chennai"/>  
</bean>  
  
<bean id="employeeBean" class="Employee">  
 <property name="address" ref="addressBean"/>  
</bean>

3. Access in Main

Employee emp = (Employee) context.getBean("employeeBean");  
emp.showDetails();

## Explanation

- Dependency Injection allows Employee to access Address without creating it directly.  
- Promotes loose coupling and easy testability.

# Exercise 3: Creating and Configuring a Maven Project

## Objective

To create a Maven-based Java project and integrate it with Spring.

## Steps

1. Create Maven Project  
 - Use any IDE (like Eclipse/IntelliJ) to create a Maven project.

2. Add Spring Dependencies

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>6.1.4</version>  
</dependency>

3. Create Classes and XML Configuration

- Follow Exercise 1 structure.

4. Build the Project

- Use mvn clean install to build.  
 - Run the application with your IDE or java -cp.

## Conclusion

This document guided you through:  
- Setting up a basic Spring Core application.  
- Implementing Dependency Injection using XML.  
- Creating and building a Maven-integrated Spring project.