Introduction to Spring

In procedural programming a sequence of instructions that are executed in order to accomplish specific tasks, in which a specific task is done by calling the required libraries by the logic code. Some characteristics of such design pattern is Tight Coupling, Hardcoded Dependencies, Difficulty in testing separate modules.

Spring Framework addresses these issues through Inversion of Control (IoC) and Dependency Injection (DI).

Inversion of Control(IoC): IoC is a design principle in which the control flow of a program is inverted. Instead of the application code controlling the flow of execution, an external framework or container manages it.

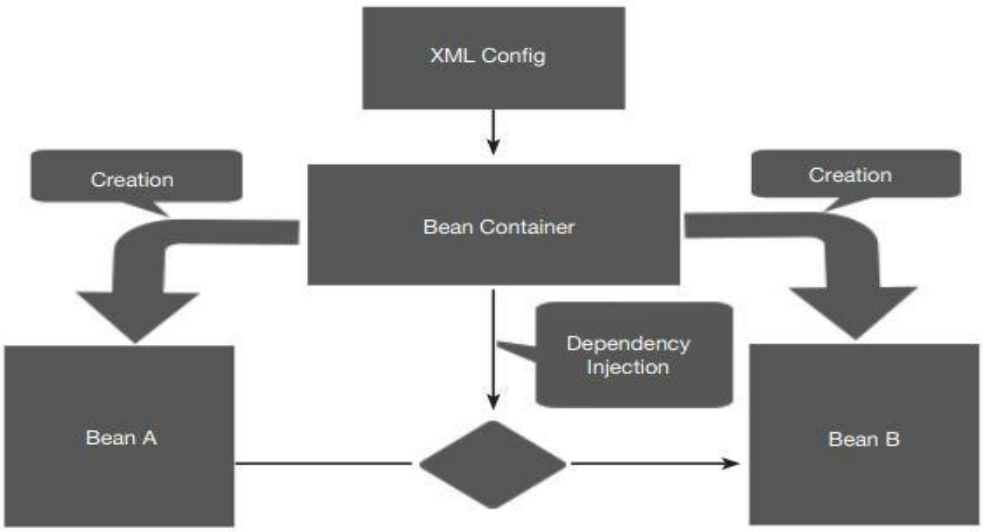
It Decouples components in an application, making the application more modular and easier to manage.

Dependency Injection (DI):

- Dependencies are objects that a class needs to perform its functions.
- Injection refers to the process of providing these dependencies to a class rather than the class creating them itself.

DI is a structural design pattern that eliminates the need for us to initialize these required objects and manage the life cycle by ourselves.

Representation of use of dependency injection



Example 1: An ObjectDepedencyTradiional is dependent on Product Object. In this approach we are creating a Product Object inside a ObjectDepedencyTradiional making it tightly coupled with Product Class.

```
public class ObjectDependencyTraditional {
	private Product product;
	public ObjectDependencyTraditional() {
		product = new Product("My Awesome Product");
	}
}
class Product{
	private String name;
	public Product(String name) {
	this.name = name;
	}
}
```

Example 2:DI approach of above program.

```
public class ObjectDependencyWithDI {
	private ProductDI product;
	public ObjectDependencyWithDI(ProductDI product) {
		this.product = product;
	}
}
class ProductDI{
	private String name;
	public ProductDI(String name) {
		this.name = name;
	}
}
```

In the above program Here IoC container will inject the object of ProductDI while creating ObjectDependencyWithDI bean.

Beans

The IoC container in Spring is responsible for managing the lifecycle and configuration of application objects, known as **beans**. It handles instantiating, configuring, and assembling these beans, as well as managing their dependencies.

Beans can be configured in several ways:

1. Creating a Bean Within an XML Configuration File

```
	<bean id="AnyUniqueId" class="YourClassName">

<!-- Property values will be set using setter injection/ Constructor injection -->
```

</bean>

2. Creating a Bean Using the @Component Annotation

3. Creating a Bean Using the @Bean Annotation

Question 1: An Interface Sim is having two abstract method calling() and data(). Two class Airtel and Voda implements Sim interface. Implement the above where IoC container of Spring framework is used to instantiate and configure beans of both the classes. Hint: Use XML configuration for beans.

Tool: Eclipse IDE

Step 1: Create a java project name **SpringDemo**

Step 2: Create a Package **Demo** in **src.**

Step 3: Create the java files in **Demo** package. Sim.java, Voda.java, Airtel.java, Mobile.java

Sim.java

```
public interface Sim {

	public void calling();
	public void data();
}
```

Airtel.java

```
public class Airtel implements Sim {
	@Override
	public void calling()
	{
		System.out.println("calling using airtel");

	}
	@Override
	public void data()
	{
		System.out.println("data using airtel");
	}
}
```

Voda.java

```
public class Voda implements Sim {
	@Override
	public void calling()
	{
		System.out.println("calling using voda");
```

```
	}
	@Override
	public void data()
	{
		System.out.println("data using voda");
	}
}
```

Mobile.java

```
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Mobile {
	public static void main(String args[])
	{
		ApplicationContext context= new
ClassPathXmlApplicationContext("bean.xml");
		System.out.println("config is loaded");

		Sim sim=(Sim)context.getBean("sim");
		sim.calling();
		sim.data();

		Sim sim1=(Sim)context.getBean("sim1");
		sim1.calling();
		sim1.data();
	}
}
```

Step 4: Configure beans in IoC container using XML configuration file. Add **bean.xml** file in **src**

Bean.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"
    xmlns:p="http://www.springframework.org/schema/p"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd" >

	<bean id="sim" class="Demo.Voda"></bean>
	<bean id="sim1" class="Demo.Airtel"></bean>
</beans>
```

Step 5: Add the Spring Libraries. Right click on src → Build Path → Configure Build Path → Libraries→Click on ClassPath→Add External JARs→ Browse and Select all the required libraries(inside spcorejars file)→Apply and Close.

Step 6: Additional Step for logging purpose. Add **log4j.properties** in **src**

log4j.properties

```
# Root logger option
log4j.rootLogger=DEBUG, console

# Redirect log messages to console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yyyy-MM-dd    HH:mm:ss}    %-5p
%c{1}:%L - %m%n
```

Question 2: Create a student class. Use IoC container of Spring framework to instantiate and configure beans of student class. Use setter method to set the property value for the bean object. Hint: Use XML configuration for beans.

Step 1: Create a java project **SpringDIExample**

Step 2: Create **com.example.model** package in **src**.

Step 3: Create a **Student.java** in **com.example.model.**

Student.java

```
public class Student {
    private String name;
    public void setName(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
}
```

Step 4: Create **com.example** package.

Step 5: Create **StudentApp.java** in **com.example** .

StudentApp.java

```
import com.example.model.Student;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class StudentApp {
    public static void main(String[] args) {
        // Load the Spring context from the XML configuration file
        ApplicationContext context = new
ClassPathXmlApplicationContext("applicationContext.xml");

        // Retrieve the bean from the context
        Student student = (Student) context.getBean("student");

        // Use the bean
        System.out.println("Student's name: " + student.getName());
    }
}
```

StudentApp.java

Step 6: Configure beans in IoC container using XML configuration file. Add **applicationContext.xml** file in **src.**

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">

    <!-- Define a bean for the student -->
    <bean id="student" class="com.example.model.Student">
        <property name="name" value="Jackie Chan"/>
    </bean>

</beans>
```

Step 7: Add the Spring Libraries. Right click on src → Build Path → Configure Build Path → Libraries→Click on ClassPath→Add External JARs→ Browse and Select all the required libraries(inside spcorejars file)→Apply and Close.

Step 8: Additional Step for logging purpose. Add **log4j.properties** in **src**

log4j.properties

```
# Root logger option
log4j.rootLogger=DEBUG, console

# Redirect log messages to console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.ConversionPattern=%d{yyyy-MM-dd    HH:mm:ss}    %-5p
%c{1}:%L - %m%n
```