Rutik Bhanushali **Practical 3** **RollNo - 04**

Aim : Develop application using Spring Framework,Lightweight Containers and Dependency Injection with Spring.

Theory :

Dependency Injection (DI) is a design pattern that removes the dependency from the programming code so that it can be easy to manage and test the application. Dependency Injection makes our programming code loosely coupled.

Two ways to perform Dependency Injection in Spring framework.Spring framework provides two ways to inject dependency.

1.By Constructor
2.By Setter method

The <constructor-arg> subelement of <bean> is used for constructor injection.
We can inject the dependency by setter method also.
The <property> subelement of <bean> is used for setter injection.'

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.
Autowiring can't be used to inject primitive and string values. It works with reference only.

Advantage of Autowiring
It requires the less code because we don't need to write the code to inject the dependency explicitly.

Disadvantage of Autowiring
No control of programmer.

The @Autowired annotation marks a Constructor, Setter method, Properties and Config() method as to be autowired that is 'injecting beans'(Objects) at runtime by Spring Dependency Injection mechanism

Spring beans can be declared either by Java configuration or XML configuration. By declaring beans, you provide metadata to the Spring Container to return the required dependency object at runtime. This is called Spring Bean Autowiring. In java based configuration, all the bean methods are defined in the class with @configuration annotation. At runtime, Spring will provide bean definitions by reading those methods. Using @Autowired, the right dependency is assigned by the Spring Container.

1. Write a program to create Student class (name, Department) and Department(departmentName, Deaprtmetnid, Major) class Student Has-A relationship with Deaprtment class. Use constructor injection to invoke object of Department class in Student class.

Code :

Student.java -

```
package com.spring;

public class Student {
	public String name;
	Department department;

	 public Student() {
	 }

	@Override
	public String toString() {
		return "Student [name=" + name + ", department=" + department + "]";
	}

“
	 public Student(String name, Department department) {
		this.name = name;
		this.department = department;
	 }

}
```

Department.java -

```
package com.spring;


public class Department {
	public int departmentId;
	public String departmentName;
	public String major;

	public Department(){
	}
	@Override
	public String toString() {
		return "Department [departmentId=" + departmentId + ", departmentName=" +
departmentName + ", major=" + major
				+ "]";
	}


	public Department(int departmentId, String departmentName, String major) {
		this.departmentId = departmentId;
		this.departmentName = departmentName;
		this.major = major;
	}

}
```

Myprogram.java -

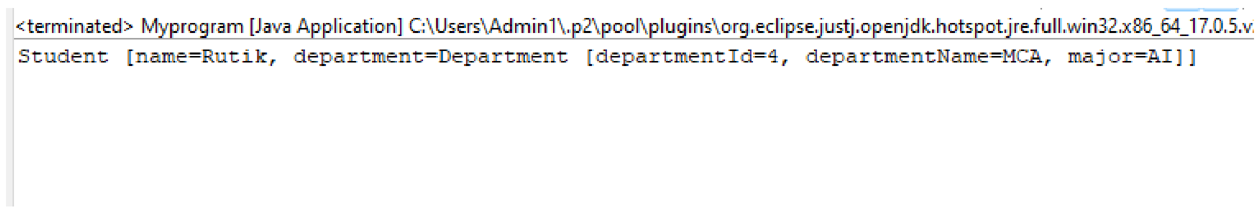
```
package com.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;


public class Myprogram {
	public static void main(String[] args) {
		ApplicationContext context = new
ClassPathXmlApplicationContext("file:src/main/resources/values.xml");
		Student stud1 = (Student) context.getBean("stud1");
		System.out.println(stud1);
	}
}
```

values.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-3.0.xsd">
```

```
<bean id = "dept1" class = "com.spring.Department">
<constructor-arg name = "departmentId" value = "4"/>
<constructor-arg name = "departmentName" value = "MCA"/>
<constructor-arg name = "major" value = "AI"/>
</bean>
<bean id="stud1" class="com.spring.Student">
        <constructor-arg value="Rutik"/>
        <constructor-arg ref="dept1"/>
</bean>

</beans>
```

O/P:



```
<terminated> Myprogram [Java Application] C:\Users\Admin1\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v
Student [name=Rutik, department=Department [departmentId=4, departmentName=MCA, major=AI]]
```

2. Write a program create Student class and Department classStudent class Has-A relationship with Deaprtment class. Use setter injection to invoke object of Department class in Student class

Code :

Student.java -

```
package com.spring;

public class Student {
        public String name;
        Department department;

         public Student() {
         }


        @Override
        public String toString() {
                return "Student [name=" + name + ", department=" + department + "]";
        }
```

```
	public Student(String name, Department department) {
		setName(name);
		setDepartment(department);
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}

	public void setDepartment(Department department) {
		this.department = department;
	}

	public Department getDepartment() {
		return department;
	}
}
```

Department.java -

```
package com.spring;


public class Department {
	public int departmentId;
	public String departmentName;
	public String major;

	public Department(){
	}



	@Override
	public String toString() {
		return "Department [departmentId=" + departmentId + ", departmentName=" +
departmentName + ", major=" + major
					+ "]";
	}
	public Department(int departmentId, String departmentName, String major) {
		setDepartmentId(departmentId);
		setDepartmentName(departmentName);
		setMajor(major);
	}

	public void setDepartmentId(int departmentId) {
		this.departmentId = departmentId;
	}

	public int getDepartmentId() {
		return departmentId;
	}

	public void setDepartmentName(String departmentName) {
		this.departmentName = departmentName;
	}

	public String getDepartmentName() {
		return departmentName;
	}

	public void setMajor(String major) {
		this.major = major;
	}

	public String getMajor() {
		return major;
	}
}
```

Myprogram.java -

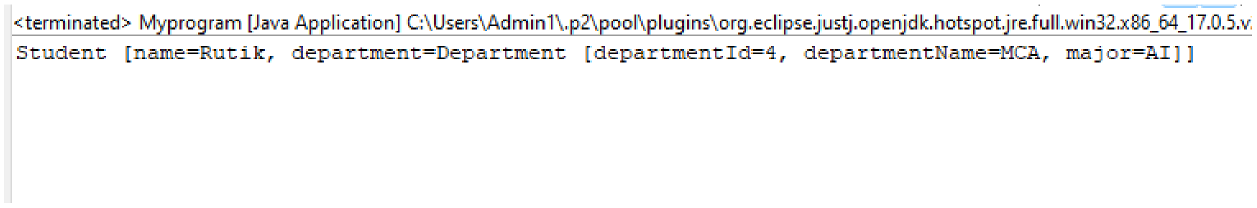
```
package com.spring;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;


public class Myprogram {
	public static void main(String[] args) {
		ApplicationContext context = new
ClassPathXmlApplicationContext("file:src/main/resources/values.xml");
		Student stud1 = (Student) context.getBean("stud1");
```

```
                    System.out.println(stud1);
          }
}
```

values.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-3.0.xsd">

<bean id = "dept1" class = "com.spring.Department">
<property name = "departmentId" value = "4"/>
<property name = "departmentName" value = "MCA"/>
<property name = "major" value = "AI"/>
</bean>

<bean id = "stud1" class = "com.spring.Student">
<property name = "name" value = "Rutik"/>
<property name = "department" ref="dept1"/>
</bean>

</beans>
```

O/P:



```
<terminated> Myprogram [Java Application] C:\Users\Admin1\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v
Student [name=Rutik, department=Department [departmentId=4, departmentName=MCA, major=AI]]
```

3. Write a program to create Car class (CarColor, Brand) and Engine(Chesiss Number, Gears) class class Has-A relationship with Engine class. Use Auto-wiring through XML (any one byname/bytype/by constructor) for dependency injection.

Code:

Web.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-3.0.xsd">


<bean id="engine" class="program3.Engine">
<property name="chesiss_Number" value="123" />
<property name="gears" value="4" />
</bean>
<bean id="car" class="program3.Car" autowire="byName">
<property name="carColor" value="Yellow" />
<property name="brand" value="Audi" />

</bean>
  </beans>
```

Car.java :

```
package program3;

public class Car {
	String carColor;
	String brand;
	Engine engine;
	public String getCarColor() {
		return carColor;
	}
	public void setCarColor(String carColor) {
		this.carColor = carColor;
	}
	public String getBrand() {
		return brand;
	}
	public void setBrand(String brand) {
		this.brand = brand;
	}
	public Engine getEngine() {
		return engine;
	}
	public void setEngine(Engine engine) {
		this.engine = engine;
	}
	@Override
	public String toString() {
		return "Car [carColor=" + carColor + ", brand=" + brand + ", engine=" + engine +
"]";
	}


}
```

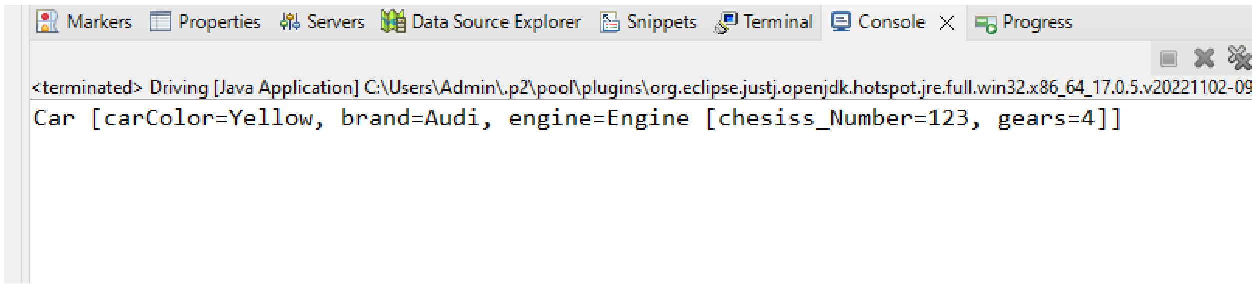
Engine.java :

```
package program3;

public class Engine {
int chesiss_Number;
int gears;
public int getChesiss_Number() {
	return chesiss_Number;
}
public void setChesiss_Number(int chesiss_Number) {
	this.chesiss_Number = chesiss_Number;
}
public int getGears() {
	return gears;
}
public void setGears(int gears) {
	this.gears = gears;
}
@Override
public String toString() {
	return "Engine [chesiss_Number=" + chesiss_Number + ", gears=" + gears + "]";
}


}
```

Driving.java :

```
package program3;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;


public class Driving {
	public static void main(String[] args) {
		ApplicationContext context = new ClassPathXmlApplicationContext("web.xml");
		Car car = (Car) context.getBean("car");
		System.out.println(car);
	}

}
```

O/P:



```
<terminated> Driving [Java Application] C:\Users\Admin\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-09
Car [carColor=Yellow, brand=Audi, engine=Engine [chesiss_Number=123, gears=4]]
```

Q.4. Write a program to create Car class and Engine class class Has-A relationship with Engine class . Use @autowiring to invoke dependency injection (any two method).

Code :

1. **Autowiring by property using annotations.**

DrivingClass:

```
package program4;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class DrivingClass {
	public static void main(String[] args) {
		ApplicationContext context = new
ClassPathXmlApplicationContext("\\program4\\Web.xml");
		Car car = (Car) context.getBean("car");
		System.out.println(car);
	}

}
```

Web.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:context="http://www.springframework.org/schema/context"

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

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.1.xsd">

<context:annotation-config/>
<bean id="engine" class="program4.Engine">
<property name="chesiss_Number" value="123" />
<property name="gears" value="4" />
</bean>
<bean id="car" class="program4.Car">
<property name="carColor" value="Yellow" />
<property name="brand" value="Audi" />

</bean>
  </beans>
```

Car.java :

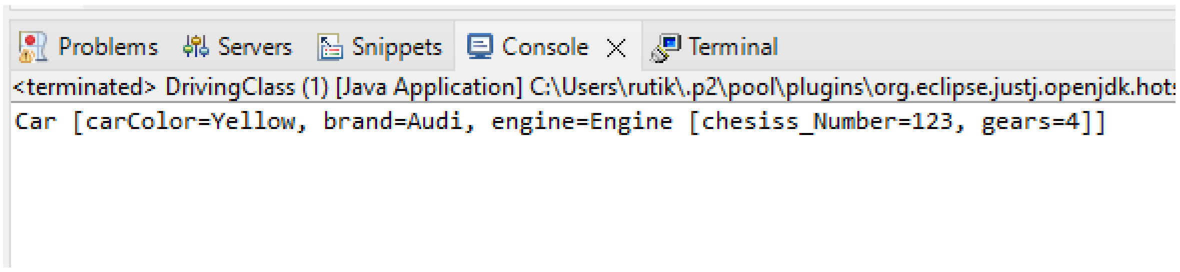
```
package program4;
import org.springframework.beans.factory.annotation.Autowired;
public class Car {
        String carColor;
        String brand;
        @Autowired
        Engine engine;
        public String getCarColor() {
                return carColor;
        }
        public void setCarColor(String carColor) {
                this.carColor = carColor;
        }
```

```
	public String getBrand() {
		return brand;
	}
	public void setBrand(String brand) {
		this.brand = brand;
	}
	public Engine getEngine() {
		return engine;
	}
	public void setEngine(Engine engine) {
		this.engine = engine;
	}
	@Override
	public String toString() {
		return "Car [carColor=" + carColor + ", brand=" + brand + ", engine=" + engine +
"]";
	}

}
```

Engine.java :

```
package program4;

public class Engine {
	int chesiss_Number;
	int gears;
	public int getChesiss_Number() {
		return chesiss_Number;
	}
	public void setChesiss_Number(int chesiss_Number) {
		this.chesiss_Number = chesiss_Number;
	}
	public int getGears() {
		return gears;
	}
	public void setGears(int gears) {
		this.gears = gears;
	}
	@Override
	public String toString() {
		return "Engine [chesiss_Number=" + chesiss_Number + ", gears=" + gears + "]";
	}
}
```

O/P :


Problems Servers Snippets Console Terminal

```
<terminated> DrivingClass (1) [Java Application] C:\Users\rutik\.p2\pool\plugins\org.eclipse.justj.openjdk.hot
Car [carColor=Yellow, brand=Audi, engine=Engine [chesiss_Number=123, gears=4]]
```

2. **Autowiring by constructor using annotations.**

DrivingClass.java :

```
package program4;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class DrivingClass {
        public static void main(String[] args) {
                ApplicationContext context = new
ClassPathXmlApplicationContext("\\program4\\Web.xml");
                Car car = (Car) context.getBean("car");
                System.out.println(car);
        }

}
```

Web.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:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
```

```
http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.1.xsd">

<context:annotation-config/>
<bean id="engine" class="program4.Engine">
<property name="chesiss_Number" value="123" />
<property name="gears" value="4" />
</bean>
<bean id="car" class="program4.Car">
<property name="carColor" value="Yellow" />
<property name="brand" value="Audi" />

</bean>
  </beans>
```

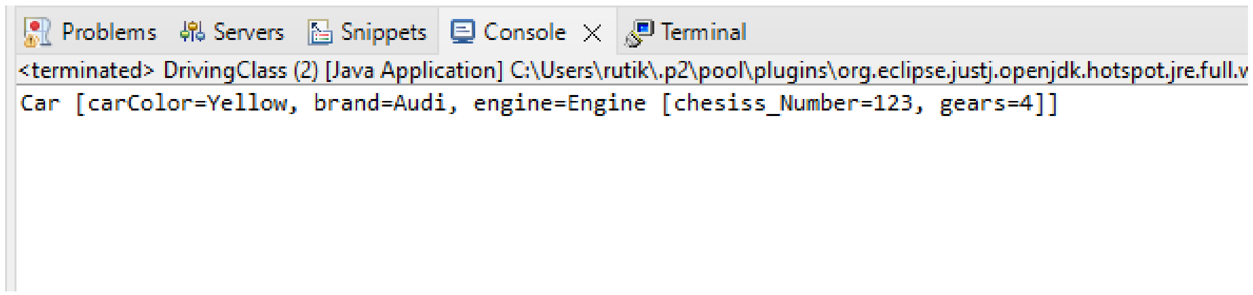
Car.java :

```
package program4b;

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

import program4.Engine;

public class Car {
	String carColor;
	String brand;

	Engine engine;
	@Autowired
	public Car(Engine engine) {
		this.engine=engine;
	}
	public String getCarColor() {
		return carColor;
	}
	public void setCarColor(String carColor) {
		this.carColor = carColor;
	}
	public String getBrand() {
		return brand;
	}
	public void setBrand(String brand) {
		this.brand = brand;
	}
	public Engine getEngine() {
		return engine;
	}
	public void setEngine(Engine engine) {
		this.engine = engine;
	}
	@Override
	public String toString() {
		return "Car [carColor=" + carColor + ", brand=" + brand + ", engine=" + engine +
"]";
	}

}
```

Engine.java :

```
package program4b;

public class Engine {
	int chesiss_Number;
	int gears;
	public int getChesiss_Number() {
		return chesiss_Number;
	}
	public void setChesiss_Number(int chesiss_Number) {
		this.chesiss_Number = chesiss_Number;
	}
	public int getGears() {
		return gears;
	}
	public void setGears(int gears) {
		this.gears = gears;
	}
	@Override
	public String toString() {
		return "Engine [chesiss_Number=" + chesiss_Number + ", gears=" + gears + "]";
	}

}
```

O/P :



```
<terminated> DrivingClass (2) [Java Application] C:\Users\rutik\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.v
Car [carColor=Yellow, brand=Audi, engine=Engine [chesiss_Number=123, gears=4]]
```

Conclusion : I have successfully learned Spring Framework,Lightweight Containers and Dependency Injection with Spring.