Practical no. 7

Programs based on Assignment based Spring Framework

Program no. 1

AIM:

Write a program to print “Hello World” using spring framework.

Steps to Create a Spring "Hello World" Application

Step 1: Create a New Maven Project

1. Open Eclipse.(2024)
2. Go to **File > New > Other...**.
3. In the wizard, select **Maven > Maven Project** and click **Next**.
4. Select **Create a simple project (skip archetype selection)** and click **Next**.
5. Fill in the **Group Id** (e.g., `com.example`) and **Artifact Id** (e.g., `hello-spring`) and click **Finish**.

Step 2: Update `pom.xml`

Open the `pom.xml` file in your project and add the Spring dependencies. Here’s a basic example:

```
<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>hello-spring</artifactId>
    <version>1.0-SNAPSHOT</version>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.10</version> <!-- Check for the latest
version -->
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
```

```
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Step 3: Create the Application Class

1. Right-click on the `src/main/java` directory, select **New > Package**, and name it `com.example.hellospring`.
2. Right-click on the newly created package, select **New > Class**, and name it `HelloWorldApp`.

Here's a simple example of what the class might look like:

```
package com.example.hellospring;

import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicati
onContext;

public class HelloWorldApp {
    public static void main(String[] args) {
        ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
        HelloWorld helloWorld =
context.getBean(HelloWorld.class);
        helloWorld.sayHello();
    }
}
```

Step 4: Create the Configuration Class

1. Right-click on the same package, select **New > Class**, and name it `AppConfig`.

Here's an example of what the configuration class might look like:

```
package com.example.hellospring;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
public class AppConfig {
    @Bean
    public HelloWorld helloWorld() {
        return new HelloWorld();
    }
}
```

Step 5: Create the HelloWorld Class

1. Right-click on the same package, select **New > Class**, and name it `HelloWorld`.

Here's how it might look:

```
package com.example.hellospring;

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

Step 6: Run the Application

1. Right-click on the `HelloWorldApp.java` file and select **Run As > Java Application**.
2. You should see `Hello, World!` printed in the console.

Program no. 2

AIM
Write a program to demonstrate dependency injection via setter method.
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>SpringDIExample</artifactId>
    <version>1.0-SNAPSHOT</version>
```

```
    <properties>
        <spring.version>5.3.22</spring.version>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
            <version>1.7.30</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>1.7.30</version>
        </dependency>
    </dependencies>
</project>
```

V8Engine.java

```
package di_program;

public class V8Engine implements Engine {
    @Override
    public void start() {
        System.out.println("V8 Engine is starting...");
    }
}
```

Interface

```
package di_program;

public interface Engine {
    void start();
}
```

Car.java

```
package di_program;
```

```
public class Car {
    private Engine engine;

    // Setter method for dependency injection
    public void setEngine(Engine engine) {
        this.engine = engine;
    }

    public void startCar() {
        if (engine != null) {
            engine.start();
            System.out.println("Car is ready to go!");
        } else {
            System.out.println("Engine is not set. Cannot start the
car.");
        }
    }
}
```

Appconfig.java

```
package di_program;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {
    @Bean
    public Engine engine() {
        return new V8Engine(); // Create and return the V8Engine bean
    }

    @Bean
    public Car car() {
        Car car = new Car();
        car.setEngine(engine()); // Inject the engine using the setter
method
        return car;
    }
}
```

Main.java:

```
package di_program;
```

```
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationCont
ext;

public class Main {
    public static void main(String[] args) {
        // Create the application context from the configuration class
        ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);

        // Retrieve the car bean
        Car car = context.getBean(Car.class);

        // Start the car
        car.startCar();
    }
}
```

Program no. 3

AIM : Write a program to demonstrate dependency injection via Constructor.

Step 1: Create a New Maven Project

1. Open Eclipse.(from D drive)
2. Go to **File > New > Spring Legacy Project**
3. Give project name org.viva ,select simple java-> finish.
4. Expand Project->select src->create a package-> org.viva(if name is coming by default just click on finish).
5. Right click on package-> new->Class->Account.java
6. Again-> Right click on package-> new->Class->AccountTest.java

7. Select Src->Right Click->new->other->Bean Configuration File ->give name.
8. Select project-> right click->build path>configure build path->classpath-> add external jar->from d drive->open spring RELEASE->libs->select all jar files->apply->apply and close.
9. Run as JAVA Application-> Account.Test file.
10. Follow same steps for other program too.

Account.java

```
package org.viva;

public class Account {


	int acNo;
	String acName;
	double acbalance;
	/**
	 * @return the acNo
	 */
	public int getAcNo() {
		return acNo;
	}
	/**
	 * @param acNo the acNo to set
	 */
	public void setAcNo(int acNo) {
		this.acNo = acNo;
	}
	/**
	 * @return the acName
	 */
	public String getAcName() {
		return acName;
	}
	/**
	 * @param acName the acName to set
	 */
	public void setAcName(String acName) {
		this.acName = acName;
	}
	/**
	 * @return the acbalance
	 */
	public double getAcbalance() {
```

```
            return acbalance;
      }
      /**
       * @param acbalance the acbalance to set
       */
      public void setAcbalance(double acbalance) {

this.acbalance = acbalance;
      }
      public Account(int acNo, String acName, double acbalance) {
            super();
            this.acNo = acNo;
            this.acName = acName;
            this.acbalance = acbalance;
      }

      public Account() {
            super();
      }
}
```

Appctx.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="Account" class="org.viva.Account">


       <constructor-arg type="int" value="000001" >
                        </constructor-arg>
            <constructor-arg type="String" value="Priya">
                     </constructor-arg>
                <constructor-arg type="double" value="2300">

</constructor-arg>
</bean>

</beans>
```

AcoountTest.java

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


public class AccountTest {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		ApplicationContext ctx=new ClassPathXmlApplicationContext("appctx.xml");

		Account a1=(Account) ctx.getBean("Account");

		System.out.println("Ac NO:"+a1.getAcNo());
		System.out.println("Ac Name:"+a1.getAcName());
		System.out.println("Ac Balance:"+a1.getAcbalance());

	}

}
```

Program no. 4

AIM : Write a program to demonstrate Autowiring.

Engine.java:

```
package myspring.viva;

import org.springframework.stereotype.Component;

@Component
public class Engine {
    public void start() {
        System.out.println("Engine started!");
    }
}
```

Car.java

```
package myspring.viva;

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

@Component
```

```
public class Car {
    private Engine engine;

    // Autowire the Engine class into the Car class
    @Autowired
    public Car(Engine engine) {
        this.engine = engine;
    }

    public void drive() {
        engine.start();
        System.out.println("Car is moving!");
    }
}
```

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

    <!-- Enable component scanning for the 'myspring.viva' package -->
    <context:component-scan base-package="myspring.viva" />
</beans>
```

MainApp.java

```
package myspring.viva;

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

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("spring-config.xml");
```

```
        // Get the Car bean from the Spring container
        Car car1 = context.getBean(Car.class);
        car1.drive();

        ((ClassPathXmlApplicationContext) context).close();
    }
}
```