Practical 4

AIM: Assignment based Aspect Oriented Programming

THEORY:

Aspect oriented programming(AOP) as the name suggests uses aspects in programming. It can be defined as the breaking of code into different modules, also known as modularisation, where the aspect is the key unit of modularity. Aspects enable the implementation of crosscutting concerns such as- transaction, logging not central to business logic without cluttering the code core to its functionality. It does so by adding additional behaviour that is the advice to the existing code. For example- Security is a crosscutting concern, in many methods in an application security rules can be applied, therefore repeating the code at every method, define the functionality in a common class and control were to apply that functionality in the whole application.

Dominant Frameworks in AOP:

AOP includes programming methods and frameworks on which modularisation of code is supported and implemented. Let's have a look at the three dominant frameworks in AOP:

AspectJ: It is an extension for Java programming created at PARC research centre. It uses Java like syntax and included IDE integrations for displaying crosscutting structure. It has its own compiler and weaver, on using it enables the use of full AspectJ language.

Spring: It uses XML based configuration for implementing AOP, also it uses annotations which are interpreted by using a library supplied by AspectJ for parsing and matching.Currently, AspectJ libraries with Spring framework are dominant in the market, therefore let's have an understanding of how Aspect-oriented programming works with Spring.

How Aspect-Oriented Programming works with Spring:

One may think that invoking a method will automatically implement cross-cutting concerns but that is not the case. Just invocation of the method does not invoke the advice(the job which is meant to be done). Spring uses proxy based mechanism i.e. it creates a proxy Object which will wrap around the original object and will take up the advice which is relevant to the method call. Proxy objects can be created either manually through proxy factory bean or through auto proxy configuration in the XML file and get destroyed when the execution completes. Proxy objects are used to enrich the Original behaviour of the real object.

Common terminologies in AOP:

Aspect: The class which implements the JEE application cross-cutting concerns(transaction, logger etc) is known as the aspect. It can be normal class configured through XML configuration or through regular classes annotated with @Aspect.

Weaving: The process of linking Aspects with an Advised Object. It can be done at load time, compile time or at runtime time. Spring AOP does weaving at runtime.

Advice: The job which is meant to be done by an Aspect or it can be defined as the action taken by the Aspect at a particular point. There are five types of Advice namely: Before, After, Around, AfterThrowing and AfterReturning. Let's have a brief discussion about all the five types.

Types of Advices:

Before: Runs before the advised method is invoked. It is denoted by @Before annotation.

After: Runs after the advised method completes regardless of the outcome, whether successful or not. It is denoted by @After annotation.

AfterReturning: Runs after the advised method successfully completes ie without any runtime exceptions. It is denoted by @AfterReturning annotation.

Around: This is the strongest advice among all the advice since it wraps around and runs before and after the advised method. This type of advice is used where we need frequent access to a method or database like- caching. It is denoted by @Around annotation.

AfterThrowing: Runs after the advised method throws a Runtime Exception. It is denoted by @AfterThrowing annotation.

JoinPoints: An application has thousands of opportunities or points to apply Advice. These points are known as join points. For example – Advice can be applied at every invocation of a method or exception be thrown or at various other points. But Spring AOP currently supports only method execution join points (advising the execution of methods on Spring beans).

Pointcut: Since it is not feasible to apply advice at every point of the code, therefore, the selected join points where advice is finally applied are known as the Pointcut. Often you specify these pointcuts using explicit class and method names or through regular expressions that define a matching class and method name patterns. It helps in reduction of repeating code by writing once and use at multiple points.

Q1) Create class car, bike, airplane, create an aspect Engine, create method drive() in car, ride() in bike, fly in airplane, the engine aspect has method enginestart() and engine stop(). When you run the application the enginestart() method should execute before drive(), ride() and fly() a enginestop() method should run after drive(), ride() and fly().

Code :-

App.java

```
package finny.vesit.spring;

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


public class App
{
    public static void main( String[] args )
    {

        ApplicationContext context = new ClassPathXmlApplicationContext("confi.xml");
        car car= (car) context.getBean("car");
             car.drive();
             bike bike= (bike) context.getBean("bike");
             bike.ride();
             plane airplane= (plane) context.getBean("airplane");
             airplane.fly();

    }
}
```

Engine.java

```
package finny.vesit.spring;


      import org.aspectj.lang.annotation.After;
      import org.aspectj.lang.annotation.Aspect;
      import org.aspectj.lang.annotation.Before;

      @Aspect
      public class Engine {

            @Before("execution(public void *()))")
            public void enginestart() {
                  System.out.println("The engine has started now.");
            }

            @After("execution(public void *())")
            public void enginestop() {
                  System.out.println("The engine has stopped now.\n");
            }
      }
```

Bikejava

```
package finny.vesit.spring;

public class bike {
	public void ride() {
		System.out.println("I am riding my bike.");
	}

}
```

Car.java

```
package finny.vesit.spring;

public class car {
	public void drive() {
		System.out.println("I am driving");
	}

}
```

Plane.java

```
package finny.vesit.spring;

public class plane {
	public void fly() {
		System.out.println("I am in a plane");
	}

}
```

Confi.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:aop="http://www.springframework.org/schema/aop"

   xsi:schemaLocation="http://www.springframework.org/schema/beans

   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

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

   http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">
```

```
<aop:aspectj-autoproxy />

<bean id="car" class=" Spring.aopp4.Car"/>
<bean id="bike" class=" Spring.aopp4.Bike"/>
<bean id="airplane" class=" Spring.aopp4.Airplane"/>
<bean id="engine" class=" Spring.aopp4.Engine"/>

</beans>
```

Output :-

```
The engine has started now.
I am driving
The engine has stopped now.

The engine has started now.
I am riding my bike.
The engine has stopped now.

The engine has started now.
I am in a plane
The engine has stopped now.
```

Q2) Create a class Account Holder with method Account_transaction, create an aspect logging, the logging have methods transaction_begin and Transaction_end method. When you run the application the Account_transaction should be preceded by transaction_begin and followed by transaction_end.

Code :-

App.java

```
package finny.vesit.spring;

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


public class App
{
    public static void main( String[] args )
    {

        ApplicationContext context = new ClassPathXmlApplicationContext("confi.xml");
        AccountHolder accountholder =
(AccountHolder)context.getBean("accountholder");
      int transaction_number = accountholder.accountTransaction();
      System.out.println("Your transaction number: "+transaction_number);
```

```
    }
}
```

AccountHolder.java

```
package finny.vesit.spring;

public class AccountHolder {
	public int transaction_number;
	public int getTransaction_number() {
		return transaction_number;
	}
	public void setTransaction_number(int transaction_number) {
		this.transaction_number = transaction_number;
	}
	public int accountTransaction() {
		System.out.println("Transaction is in the process...");
		return transaction_number;
	}

}
```

Logging.java

```
package finny.vesit.spring;
import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
@Aspect
public class Logging {
	@Before("execution(public int accountTransaction()))")
	public void transactionBegin() {
		System.out.println("Transaction has started now.");
	}
	@AfterReturning("execution(public int accountTransaction()))")
	public void transactionEnd() {
		System.out.println("Transaction has completed successfully.");
	}


}
```

Confi.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:aop="http://www.springframework.org/schema/aop"

   xsi:schemaLocation="http://www.springframework.org/schema/beans

   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

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

   http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">


<aop:aspectj-autoproxy />

<bean id="accountholder" class="finny.vesit.spring.AccountHolder">
<property name="transaction_number" value="7492"></property>
</bean>
<bean id="logging" class="finny.vesit.spring.Logging"/>


</beans>
```

Output:-

```
<terminated> App [Java Application] C:\Users\LENOVO\.
Transaction has started now.
Transaction is in the process...
Transaction has completed successfully.
Your transaction number: 7492
```

Q3) Create a business class multiplier(int a, int b) which returns product of a and b and create an aspect Adder which uses After-returning advice.

Code :-

App.java

```
package finny.vesit.spring;

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


public class App
```

```
{
    public static void main( String[] args )
    {

        ApplicationContext context = new ClassPathXmlApplicationContext("confi.xml");
        Business business = (Business)context.getBean("business");
      int multiple = business.multiplier();
      System.out.println("Product is = "+multiple);



    }
}
```

Business.java

```
package finny.vesit.spring;

public class Business {
      int a,b;

      public int getA() {
            return a;
      }

      public void setA(int a) {
            this.a = a;
      }

      public int getB() {
            return b;
      }
      public void setB(int b) {
            this.b = b;
      }
      public int multiplier() {
            return a*b;
      }

}
```

Adder.java

```
package finny.vesit.spring;

import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;

@Aspect
public class Adder {
```

```
private int a,b;

	public int getA() {
		return a;
	}

	public void setA(int a) {
		this.a = a;
	}

	public int getB() {
		return b;
	}

	public void setB(int b) {
		this.b = b;
	}

	@AfterReturning("execution(public int multiplier())")
	public void Addition() {
		System.out.println("Addition of numbers "+a+" and "+b+" = "+(a+b));
	}


}
```

Confi.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:aop="http://www.springframework.org/schema/aop"

   xsi:schemaLocation="http://www.springframework.org/schema/beans

   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

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

   http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">


<aop:aspectj-autoproxy />


<bean id="business" class="finny.vesit.spring.Business">
<property name="a" value="3"></property>
<property name="b" value="2"></property>
</bean>
```

```
<bean id="adder" class="finny.vesit.spring.Adder">
<property name="a" value="3"></property>
<property name="b" value="2"></property>
</bean>

</beans>
```

Output :-

```
<terminated> App [Java Application] C:\Users\LENOV
Addition of numbers 3 and 2 = 5
Product is = 6
```

Q4) Create a class car, bike, airplane, create an aspect engine, create method drive() in car, ride() in bike, fly in airplane, the engine aspect has method enginestart() and enginestop(). When you run the application the enginestart() method should execute before drive(), ride() and fly() a enginestop() method should run after drive(), ride() and fly() by using pointcuts.

Code :-

App.java

```
package finny.vesit.spring;

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


public class App
{
    public static void main( String[] args )
    {

        ApplicationContext context = new ClassPathXmlApplicationContext("confi.xml");
        car car= (car) context.getBean("car");
             car.drive();
             bike bike= (bike) context.getBean("bike");
             bike.ride();
             plane airplane= (plane) context.getBean("airplane");
             airplane.fly();

    }
}
```

Engine.java

```
package finny.vesit.spring;


	import org.aspectj.lang.annotation.After;
	import org.aspectj.lang.annotation.Aspect;
	import org.aspectj.lang.annotation.Before;
	 import org.aspectj.lang.annotation.Pointcut;

	@Aspect
	public class Engine {

		@Before("myPointcut()")
		public void enginestart() {
			System.out.println("The engine has started now.");
		}

		@After("myPointcut()")
		public void enginestop() {
			System.out.println("The engine has stopped now.\n");
		}
		@Pointcut("execution(public void *())")
		public void myPointcut() {
			//Empty method
		}	}
```

Bikejava

```
package finny.vesit.spring;

public class bike {
	public void ride() {
		System.out.println("I am riding my bike.");
	}

}
```

Car.java

```
package finny.vesit.spring;

public class car {
	public void drive() {
		System.out.println("I am driving");
	}

}
```

Plane.java

```
package finny.vesit.spring;

public class plane {
	public void fly() {
		System.out.println("I am in a plane");
	}
```

```
}
```

Confi.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:aop="http://www.springframework.org/schema/aop"

   xsi:schemaLocation="http://www.springframework.org/schema/beans

   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

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

   http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">


<aop:aspectj-autoproxy />

<bean id="car" class=" Spring.aopp4.Car"/>
<bean id="bike" class=" Spring.aopp4.Bike"/>
<bean id="airplane" class=" Spring.aopp4.Airplane"/>
<bean id="engine" class=" Spring.aopp4.Engine"/>

</beans>
```

Output :-

```
<terminated> App [Java Application] C:\Users\LENO
The engine has started now.
I am driving
The engine has stopped now.

The engine has started now.
I am riding my bike.
The engine has stopped now.

The engine has started now.
I am in a plane
The engine has stopped now.
```

Conclusion :- Successfully Implemented AOP concepts.