THEORY:

Aspect-Oriented Programming (AOP):

Aspect-Oriented Programming (AOP) complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Aspects enable the modularization of concerns such as transaction management that cut across multiple types and objects. One of the key components of Spring is the AOP framework. While the Spring IoC container does not depend on AOP, meaning you do not need to use AOP if you don't want to, AOP complements Spring IoC to provide a very capable middleware solution.

AOP Terminologies:

Sr.No	Terms & Description
1	**Aspect** This is a module which has a set of APIs providing cross-cutting requirements. For example, a logging module would be called AOP aspect for logging. An application can have any number of aspects depending on the requirement.
2	**Join point** This represents a point in your application where you can plug-in the AOP aspect. You can also say, it is the actual place in the application where an action will be taken using Spring AOP framework.
3	**Advice** This is the actual action to be taken either before or after the method execution. This is an actual piece of code that is invoked during the program execution by Spring AOP framework.
4	**Pointcut** This is a set of one or more join points where an advice should be executed. You can specify pointcuts using expressions or patterns as we will see in our AOP examples.
5	**Introduction** An introduction allows you to add new methods or attributes to the existing classes.
6	**Target object** The object being advised by one or more aspects. This object will always be a proxied object, also referred to as the advised object.

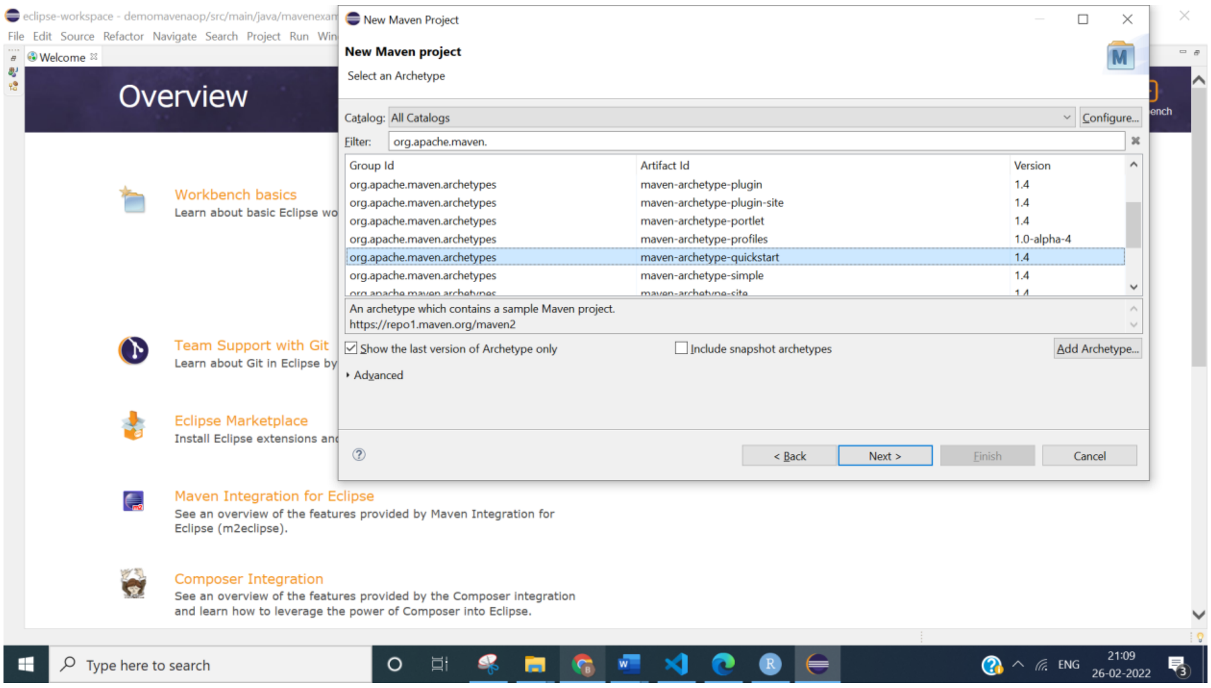
7	**Weaving** Weaving is the process of linking aspects with other application types or objects to create an advised object. This can be done at compile time, load time, or at runtime.

Types of Advice:

Spring aspects can work with five kinds of advice mentioned as follows −

Sr.No	Advice & Description
1	**before** Run advice before the a method execution.
2	**after** Run advice after the method execution, regardless of its outcome.
3	**after-returning** Run advice after the a method execution only if method completes successfully.
4	**after-throwing** Run advice after the a method execution only if method exits by throwing an exception.
5	**around** Run advice before and after the advised method is invoked.

SETUP OF MAVEN PROJECT:

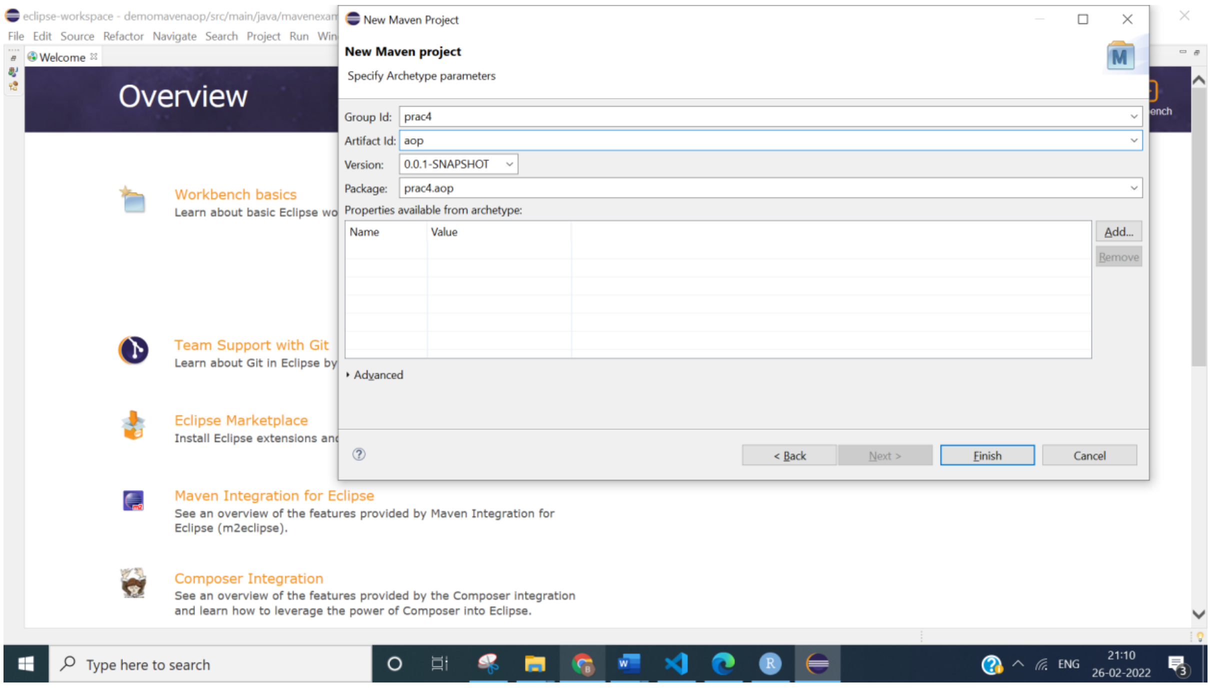

New Maven project
Select an Archetype

Catalog: All Catalogs
Filter: org.apache.maven.

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-plugin	1.4
org.apache.maven.archetypes	maven-archetype-plugin-site	1.4
org.apache.maven.archetypes	maven-archetype-portlet	1.4
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.4
org.apache.maven.archetypes	maven-archetype-simple	1.4

An archetype which contains a sample Maven project.
https://repo1.maven.org/maven2

☑ Show the last version of Archetype only ☐ Include snapshot archetypes


New Maven project
Specify Archetype parameters

Group Id: prac4
Artifact Id: aop
Version: 0.0.1-SNAPSHOT
Package: prac4.aop
Properties available from archetype:

Name	Value

pom.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<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>prac4</groupId>
  <artifactId>aop</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <name>aop</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>

        <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
<dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.8</version>
</dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.16</version>
</dependency>

  </dependencies>

  <build>
    <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be
moved to parent pom) -->
      <plugins>
```

```
        <!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-
core/lifecycles.html#clean_Lifecycle -->
        <plugin>
          <artifactId>maven-clean-plugin</artifactId>
          <version>3.1.0</version>
        </plugin>
        <!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-
core/default-bindings.html#Plugin_bindings_for_jar_packaging -->
        <plugin>
          <artifactId>maven-resources-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-compiler-plugin</artifactId>
          <version>3.8.0</version>
        </plugin>
        <plugin>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.22.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-jar-plugin</artifactId>
          <version>3.0.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-install-plugin</artifactId>
          <version>2.5.2</version>
        </plugin>
        <plugin>
          <artifactId>maven-deploy-plugin</artifactId>
          <version>2.8.2</version>
        </plugin>
        <!-- site lifecycle, see https://maven.apache.org/ref/current/maven-
core/lifecycles.html#site_Lifecycle -->
        <plugin>
          <artifactId>maven-site-plugin</artifactId>
          <version>3.7.1</version>
        </plugin>
        <plugin>
          <artifactId>maven-project-info-reports-plugin</artifactId>
          <version>3.0.0</version>
        </plugin>
      </plugins>
    </pluginManagement>
  </build>
</project>
```

A) **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 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().**

SOURCE CODE:

Bike.java:

```
package prac4.aop.q1;

import org.springframework.stereotype.Component;

@Component
public class Bike {

public void ride()
{
	System.out.println("Bike: I am Riding");
}
}
```

Car.java:

```
package prac4.aop.q1;

import org.springframework.stereotype.Component;

@Component
public class Car {

public void drive()
{
	System.out.println("Car: I am Driving");
}
}
```

Airplane.java:

```
package prac4.aop.q1;

import org.springframework.stereotype.Component;

@Component
public class Airplane {

public void fly()
{
	System.out.println("Airplane: I am Flying");
}
}
```

Engine.java:

```
package prac4.aop.q1;

import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.context.annotation.EnableAspectJAutoProxy;
import org.springframework.stereotype.Component;

@Component
@Aspect
@EnableAspectJAutoProxy
public class Engine {

@Before("execution( public void *())")
public void engineStart()
{
                System.out.println("Engine: Starting Engine");
}

@After("execution( public void *())")
public void engineStop()
{
                System.out.println("Engine: Stopping Engine");
}
}
```

AppConfig.java:

```
package prac4.aop.q1;

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

@Configuration
@ComponentScan(basePackages="prac4.aop.q1")
public class AppConfig {

}
```

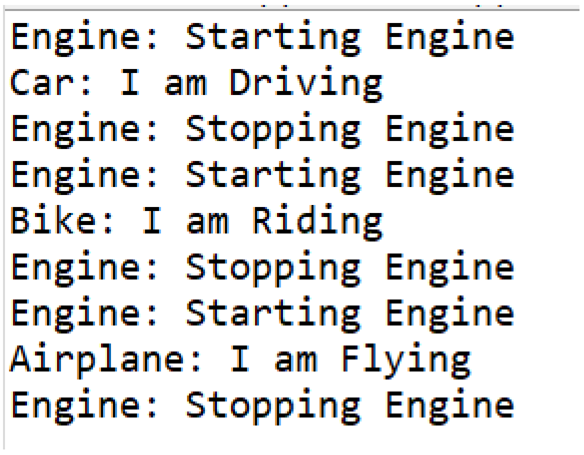
App.java:

```
package prac4.aop.q1;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
```

```
public class App
{
   public static void main( String[] args )
   {
      AbstractApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
      Car c = context.getBean("car", Car.class);
      c.drive();
      Bike b = context.getBean("bike", Bike.class);
      b.ride();
      Airplane a = context.getBean("airplane", Airplane.class);
      a.fly();
   }
}
```

OUTPUT:



```
Engine: Starting Engine
Car: I am Driving
Engine: Stopping Engine
Engine: Starting Engine
Bike: I am Riding
Engine: Stopping Engine
Engine: Starting Engine
Airplane: I am Flying
Engine: Stopping Engine
```

B) Create a business class multiplier(int a, int b) which returns product of a and create an aspect AdderAfterReturnAspect use After-returning advice

SOURCE CODE:

Multiplier.java:

```
package prac4.aop.q2;

import org.springframework.stereotype.Component;

@Component
public class Multiplier {

public int prod(int a, int b)
{
        return a*b;
}
}
```

AdderAfterRequestAspect.java:

```
package prac4.aop.q2;

import org.aspectj.lang.annotation.AfterReturning;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.context.annotation.EnableAspectJAutoProxy;
import org.springframework.stereotype.Component;

@Component
@Aspect
@EnableAspectJAutoProxy
public class AdderAfterReturnAspect {
@AfterReturning(pointcut="execution(public int prod(..))", returning="returnvalue")
public void AdderAfterReturnAspect(int returnvalue)
{
        System.out.println("From After-returning advice");
        System.out.println("Multipication is " + returnvalue);
}
}
```

AppConfig.java:

```
package prac4.aop.q2;

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

@Configuration
@ComponentScan(basePackages="prac4.aop.q2")
```

```
public class AppConfig {

}
```

App.java:

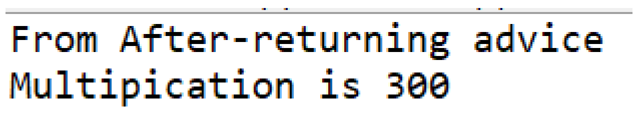
```
package prac4.aop.q2;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;


public class App {

   public static void main( String[] args )
   {
  AbstractApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
     Multiplier m = context.getBean("multiplier", Multiplier.class);
     m.prod(10, 30);
   }
}
```

OUTPUT:



```
From After-returning advice
Multipication is 300
```

C) 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 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.

SOURCE CODE:

Airplane.java:

```
package prac4.aop.q3;

import org.springframework.stereotype.Component;

@Component
public class Airplane {

public void fly()
{
        System.out.println("Airplane: I am Flying");
}
}
```

Bike.java:

```
package prac4.aop.q3;

import org.springframework.stereotype.Component;

@Component
public class Bike {

public void ride()
{
        System.out.println("Bike: I am Riding");
}
}
```

Car.java:

```
package prac4.aop.q3;

import org.springframework.stereotype.Component;

@Component
public class Car {

public void drive()
{
        System.out.println("Car: I am Driving");
}
}
```

Engine.java:

```
package prac4.aop.q3;

import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.context.annotation.EnableAspectJAutoProxy;
import org.springframework.stereotype.Component;

@Component
@Aspect
@EnableAspectJAutoProxy
public class Engine {
@Pointcut("execution( public void *())")
public void getNamePointcut() {}
@Before("getNamePointcut()")
public void engineStart()
{
		System.out.println("Engine: Starting Engine using pointcut");
}

@After("getNamePointcut()")
public void engineStop()
{
		System.out.println("Engine: Stopping Engine using pointcut");
}
}
```

AppConfig.java:

```
package prac4.aop.q3;

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

@Configuration
@ComponentScan(basePackages="prac4.aop.q3")
public class AppConfig {

}
```

App.java:

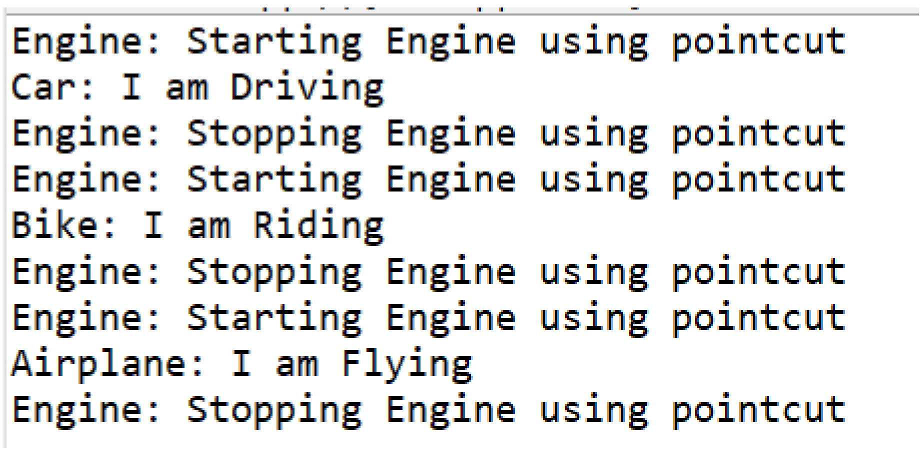
```
package prac4.aop.q3;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;
```

```
public class App {

public static void main( String[] args )
   {
      AbstractApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class);
      Car c = context.getBean("car", Car.class);
      c.drive();
      Bike b = context.getBean("bike", Bike.class);
      b.ride();
      Airplane a = context.getBean("airplane", Airplane.class);
      a.fly();
   }
}
```

OUTPUT:



```
Engine: Starting Engine using pointcut
Car: I am Driving
Engine: Stopping Engine using pointcut
Engine: Starting Engine using pointcut
Bike: I am Riding
Engine: Stopping Engine using pointcut
Engine: Starting Engine using pointcut
Airplane: I am Flying
Engine: Stopping Engine using pointcut
```

CONCLUSION:

From this practical, I have learned about Aspect Oriented Programming (AOP).