Aspect Oriented programming Exercise

1- Create Maven Project
2- Add relevant dependencies for Spring Core
3- Add the aspects dependency to the **pom.xml**

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aspects</artifactId>
    <version>6.12.1.RELEASE</version>
</dependency>
```

a. run Maven update on the project and wait to be completed.

4- Create a new class Customer under the model package.

a. Set class Scope as a singleton.
b. Set it as a Component
c. Define the following methods for the class

```
public void addCustomer (){
System.out.println("addCustomer () is running ");
    }

public String addCustomerReturnValue (){
System.out.println("addCustomerReturnValue() is
running ");
        return "Hello MST";
    }
```

```
public void addCustomerThrowException () throws
Exception {
System.out.println("addCustomerThrowException() is
running ");
                throw new Exception ("MST Error");
        }

public void addCustomerAround (String name){
 System.out.println("addCustomerAround () is running,
args : " + name);
```

5- Create the Aspect class "**LoggingAspect**" under the package aop.com

a. Implement the logBefore for the method addCustomer

```
System.out.println("@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@");
System.out.println("logBefore() is running!");
System.out.println("MST Spring : " +
joinPoint.getSignature().getName());
System.out.println("@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@@@@@");
```

b. Implement the logAfter for the Customer methods

```
System.out.println("**********************************
**********************");
System.out.println("logAfter() is running!");
System.out.println(" : " +
joinPoint.getSignature().getName());
```

```
System.out.println("**********************************
**********************");
```

c. Implement the logAfterReturning for addCustomerReturnValue.

```
System.out.println("-----------------------------------------
------");
System.out.println("logAfterReturning() is running!");
System.out.println("the  : " +
joinPoint.getSignature().getName());
        System.out.println("Method returned value is : " +
    result);
        System.out.println("-------------------------------------
        -----------");
```

d. Implement the logAfterThrowing for the addCustomerThrowException

```
System.out.println("++++++++++++++++++++++++++++++
++++++++++++++++++++++++");
System.out.println("logAfterThrowing() is running!");
System.out.println("the : " +
joinPoint.getSignature().getName());
System.out.println("Exception : " + error);
System.out.println("++++++++++++++++++++++++++++++
++++++++++++++++++++++++");
```

e. Implement the logAround for addCustomerAround

```
System.out.println("//////////////////////////////////////
//////////");
System.out.println("logAround() is running!");
```

```
System.out.println("MST Around method : " +
joinPoint.getSignature().getName());
System.out.println("MST Around arguments : " +
Arrays.toString(joinPoint.getArgs()));

System.out.println("Around before is running!");
joinPoint.proceed(); //continue on the intercepted
method
System.out.println("Around after is running!");
System.out.println("//////////////////////////////////////////
//////////");
```

6- Open in the root of the project a class SpringMain

a. Give the following dependencies
   i. @Configuration
   ii. @EnableAspectJAutoProxy
   iii. @ComponentScan({"com.mst"})

b. Create the Main method
   i. Create the Spring Core container Context
   ii. Create new Bean from Customer Class
   iii. invoke the below methods :

```
customer.addCustomerReturnValue();

customer.addCustomerThrowException();

customer.addCustomerAround("Hello John");
```

c. Close the context .