INDEX

Spring AOP ----------

Spring AOP

Introduction

AOP: Aspect Oriented Programming

- AOP is not replacement for OOP. It actually compliments for OOP.
- AOP is a methodology of programming to solve the problems related OOP style programming but it is built on the top of OOP Style programming.
- The logics that are minimum and mandatory to complete the task in application development are called primary logics or business logics or main logics.
  e.g.: Transfer money logics, withdraw logics, deposit logics, closing account logics and etc.
- The logics are optional and configurable logics having ability to enable to disable are called secondary logics.
  e.g.: Logging, auditing, Transaction Management, security, Performance Monitoring and etc.
- In OOP style programming the business method of service class/ main class contains both primary and secondary logics together as mixed-up logics having multiple limitations

Logging: Keeps track code flow in the app execution (using log4j, slf4j and etc.)
Auditing: Keeps track user activities flow (operations done by use after entering into Application)

```
public class BankServiceImpl implements BankService{

        public String withdraw (long accno, float amt) {
                //Security logic (Authentication +Authorization)
                …….                                                    Secondary
                //Logging                                             logic
                ……
                //Auditing
                ……
                //primary logic
                balance=balance-amt; // (may use DAO here)
                return "…";
        }
        public String deposit (long accno, float amt) {
                //Security logic (Authentication +Authorization)
                …….
```

```
                //Logging
                ……
                //Auditing
                ……
                //primary logic
                balance=balance+amt; // (may use DAO here)
                return "…";
        }
}
```

Secondary logic

Note: The secondary logics are the Application are also called as Middleware services or aspects or cross cutting concerns.

Limitations of OOP style programming:

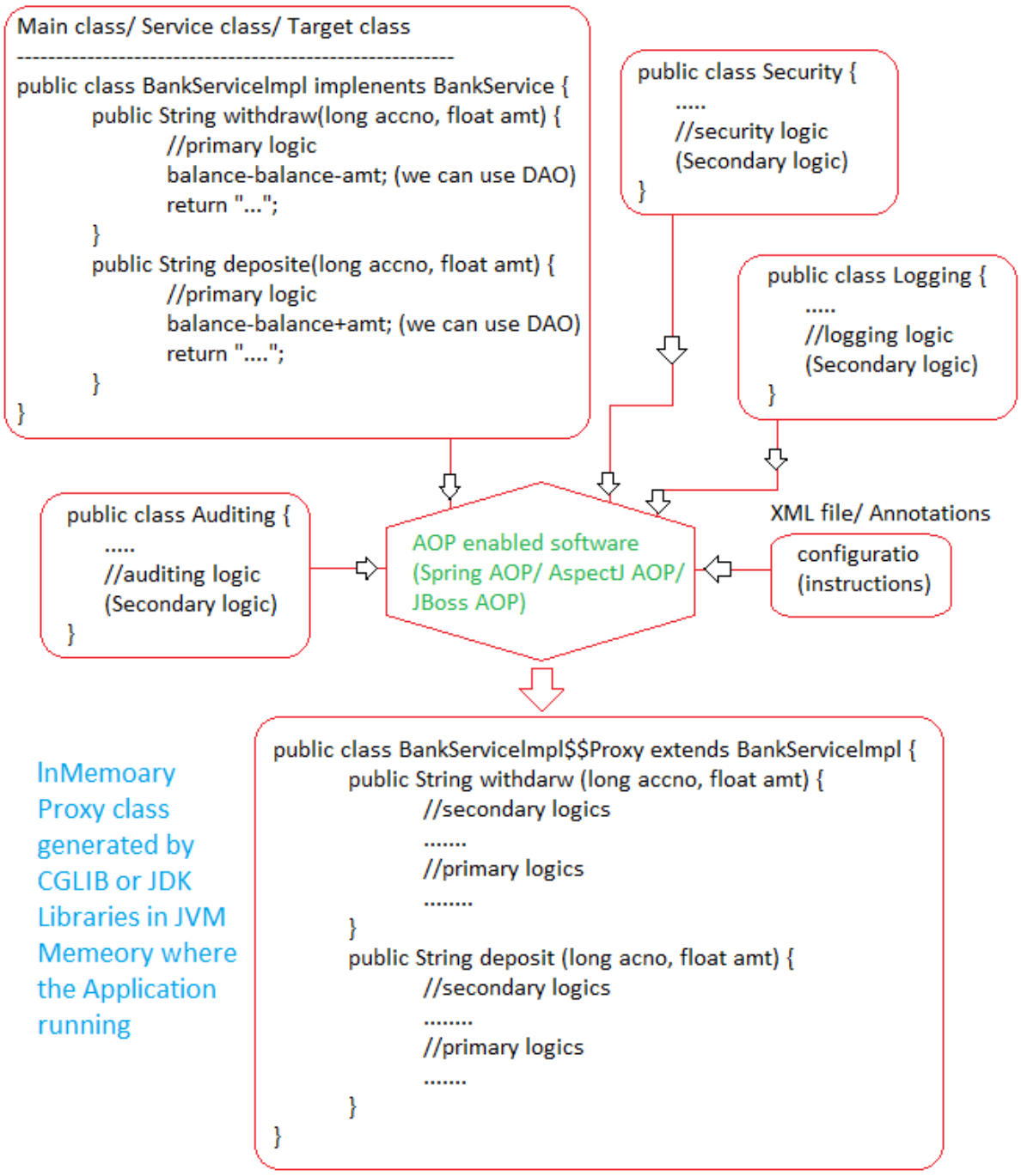
a. Code becomes clumsy because both primary and secondary logics are mixed-up.
b. No reusability for secondary logics (No modularity).
c. Kills the readability of the code.
d. No clean separation between primary and secondary logics.
e. Debugging is complex.
f. We can't enable or disable secondary logics on primary logics without touching the source code.
g. Not suitable for Medium scale and Large-scale projects (It is not industry standard).

Note: To overcome the problems, take the support of AOP style programming which say separate secondary logics from primary logics at the time of development a compilation but mix them dynamically at runtime as needed with the support of AOP enabled s/w for frameworks like Spring AOP/ AspectJ AOP/ JBoss AOP and etc.

Advantages of AOP style programming:

a. Code is not clumsy because the primary and secondary logics are separated.
b. Improves the readability of the code.
c. Improves the modularity of the code (Secondary logics can be developed as reusable logics)
d. Can enable or disable secondary logics on primary logics without effecting the source code/ primary logics.
e. It is industry standard and suitable for all kinds of applications.

f. Debugging becomes easy.



- The languages that are created based on Procedure oriented programming principles are called Procedure Orientated Programming languages (POP languages).
  e.g.: C, Pascal, Basics, Fortran and etc.
- The languages that are created based on Object oriented programming principles are called Object Orientated Programming languages (OOP languages)

e.g.: C++, Java, and etc.
OOP principles are class, object, inheritance, polymorphism, abstraction, encapsulation and etc.

- The Frameworks/ softwares that are given based Aspect oriented programming principles are called AOP enabled frameworks/ softwares (All these are created based on OOP languages)
  e.g.: Spring AOP, AspectJ AOP (best), JBoss AOP and etc.

Note: As on today, the Spring framework is giving support for both Spring AOP and AspectJ AOP.

AOP Principles

These are the following AOP principles or terminologies.

a. Aspect
b. Advice
c. Join Point
d. Pointcut
e. Target class
f. Weaving
g. Proxy class

Aspect (What u want to apply):

- It is the class that represents Secondary logic(s). This is also called as Middleware service or cross cutting concern.

Advice (How u want to apply):

- The action taken place by Aspect is called Advice. For Aspect if specify how it should be executed/ behaved then it is called Advice i.e. Aspect contains secondary logic whereas Advice contains secondary logic with action plan.
- There are 4 types advices
  - Before Advice (Executes before entering into business method/ target method).
  - After Advice (Executes after completing the execution of business method/ target method).
  - Around Advice (Executes before and after target method/ business method execution)
  - Throws Advice (Executes only when exception is raised in the target method/ business method).

Join Point (Where we can apply):

- The possible places in target class where aspects can be advised. These can be fields (variables), constructors, methods. As of now Spring supports only methods as join points.
- Target class with 20 business methods/ target methods then we have 20 join points.

Pointcut (Whey we have applied or we want to apply):

- Collection of join points where the aspects are advised i.e. pointcut holds set of business method/ target method names belonging target class to apply aspects/ advices.
- If target class contains 20 methods then there are 20 join points, in that if we want to apply aspects only 10 methods then pointcut should hold those 10 method names.

Target class:

- The class that contains business methods/ target methods having primary logics nothing but class with join points.
- Pre AOP class with target methods called target class

Weaving:

- The process of applying or mixing-up secondary logics with primary logics and generating lnMemory proxy class is called weaving process. AOP enabled softwares will perform this weaving process.
- The Inputs for weaving process are Target class and Aspect/ advice classes and the outcome of weaving process is Proxy class (lnMemory class).

Proxy class:

- The output/ outcome of weaving process done Spring AOP enabled framework or software is called Prosy class. It is always lnMemory class.
- Post AOP class is called Proxy class

Note:

- ✓ If we call business methods on Target class object only primary logics/ business logics will execute.
- ✓ If we call business methods on Proxy class object the both primary logics/ business logics will execute.

Diagram 1: How to get InMemory Proxy Class

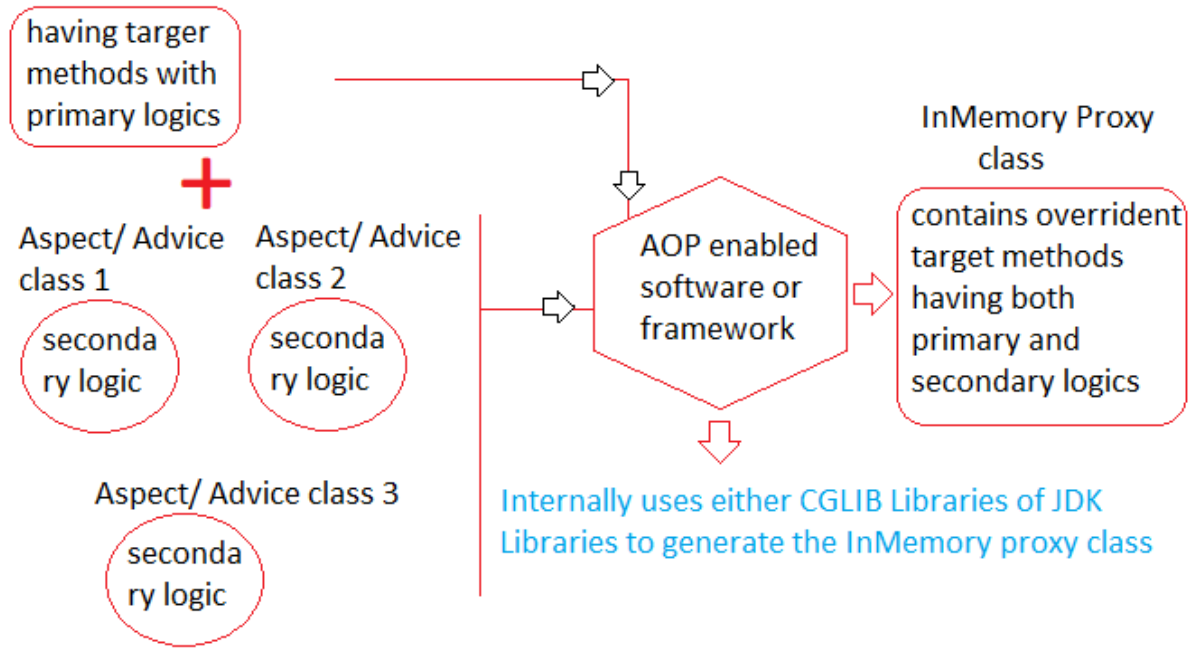
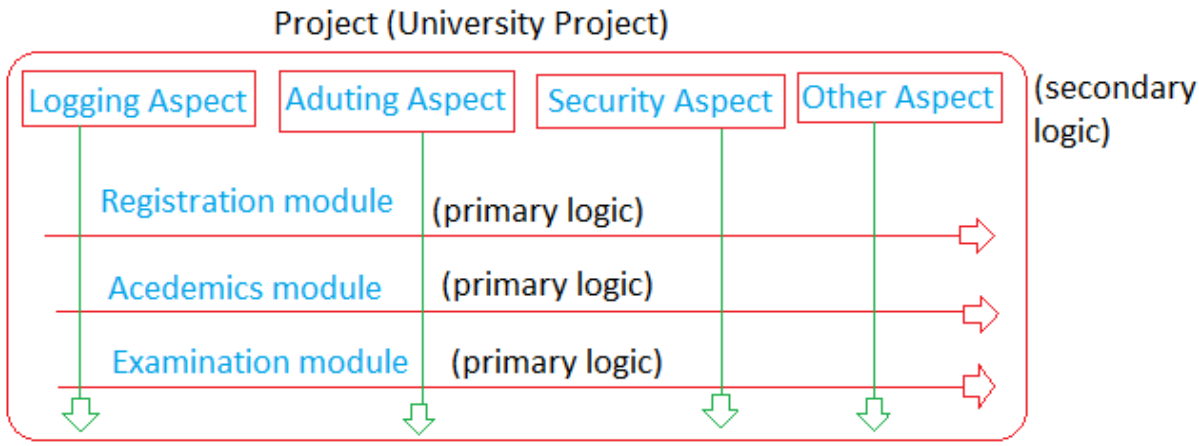


Diagram 2: Clarity about Primary and Secondary logics



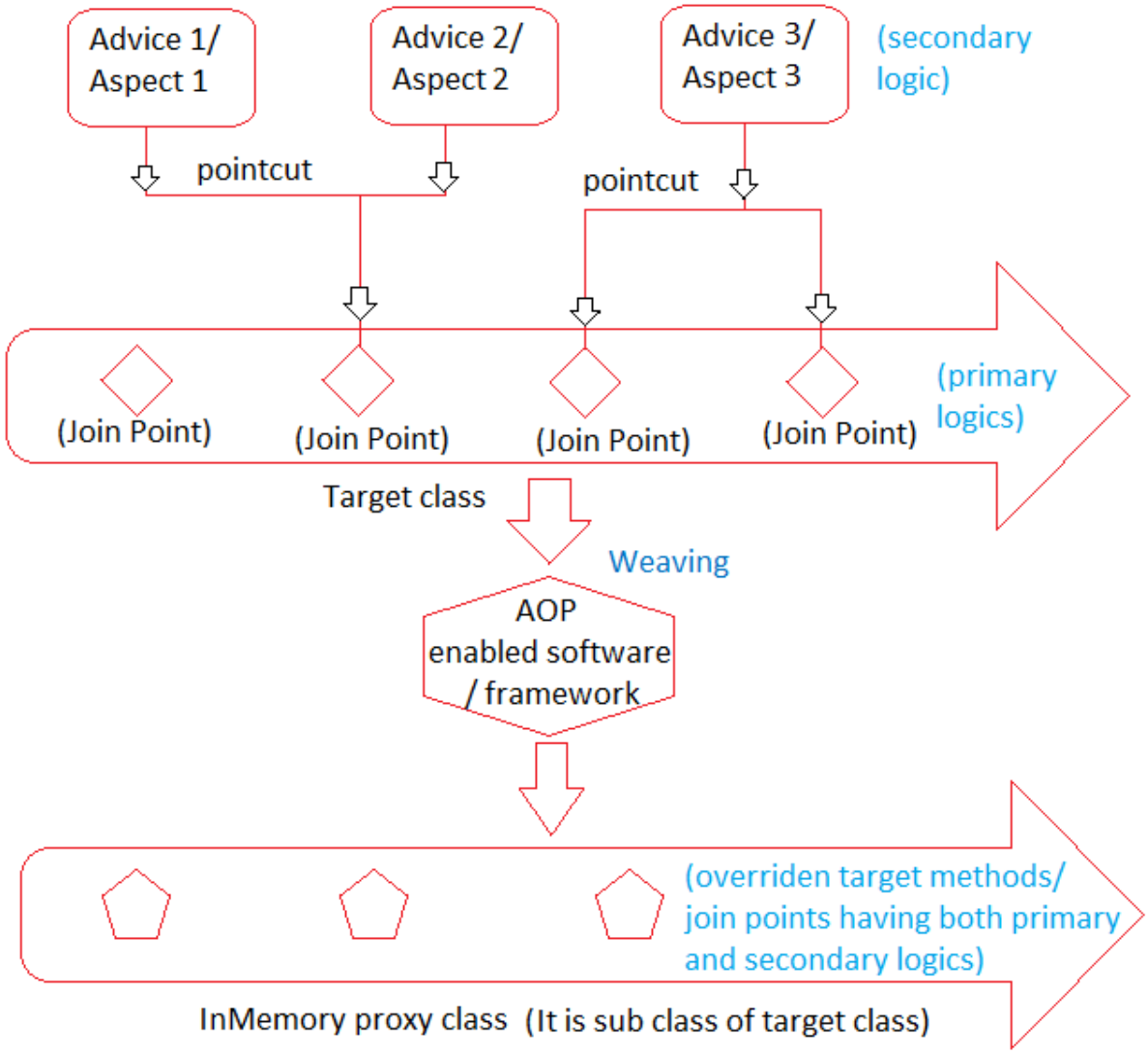
Note: In real project development secondary logics are vertical and primary logics are horizontals more ever the secondary logic will be developed for 1 time and will be applied on multiple primary logics as cross concerns shown in the diagram.

Spring F/W supports two AOP enabled s/w:

a. Spring AOP (old)
b. Spring Integrated AspectJ AOP (Best)

Note: Initially AspectJ AOP is not part of spring framework. Since it is the best AOP enabled s/w the spring framework integrated AspectJ AOP with Spring to avoid the competition from AspectJ AOP.

Diagram 3: Flow of InMemory Proxy class creation



AOP Programming Different Approaches

- We can do AOP programming Spring framework in the following approaches,
    a. Spring AOP Declarative approach (XML configurations)
    b. Spring AOP Programmatic approach (No XML files, No IoC container)
    c. Spring AOP 100% code driven approach
    d. Spring Boot Spring AOP

Note:

- ✓ These are bit old we can see only in maintenance projects.
- ✓ Spring AOP does not support annotation.
- ✓ Spring AOP given from Spring 1.x

- ✓ In c, d approaches we can annotations related to Spring, Spring Boot programming but there are no annotations related to Spring AOP like there are no annotations to make Java classes as aspect/ advice class.
- ✓ In Spring AOP the aspect/ advice classes are invasive, i.e. we should develop them by implementing Spring AOP API interfaces.

e. Spring Integrated AspectJ AOP Declarative Approach (XML configuration)
f. Spring Integrated AspectJ AOP Annotation Approach (For Spring project this is best)
g. Spring Integrated AspectJ AOP 100% Code configuration Approach
h. Spring Boot AspectJ AOP (For Spring Boot projects this is best)

Note:

- ✓ AspectJ AOP Integration with spring is done from spring 2.x.
- ✓ In AspectJ AOP the aspect/ advice classes are non-invasive, i.e. we can develop as POJO classes.

Spring AOP Declarative Approach

- Supports 4 types advices
  a. Around Advice
  b. Before Advice
  c. After Advice
  d. Throws Advice

Spring AOP Declarative Approach Around Advice:

- This advice executes the around the target method (i.e. it executes before entering into target method and after executing target method).
- To develop this advice class, the class must implement org.aopalliance.intercept.MethodInterceptor (I) and should invoke (-) method.
- Use cases: Performance Monitoring, Caching, Transaction Management, Around Logging and etc.

```
public class PerformenceMonitoringAdvice implements MethodInterceptor {
        //MethodInvocation object holds target method details on which
        advice is applied.
        public Object invoke (MethodInvocation invocation) {
```

```
                //pre logics (Executes before entering target method)
                long startTime=System.currentTimeMills();
                // invokes the target method on which this advice is applied
                Object retVal=invocation.proceed();
                //post logics (executes after leaving the target method)
                long endTime=System.currentTimeMillis();
                S.o.p("invocation.getMethod().getName()"+" has taken "+
                (endTime-startTime)+ " ms to complete the execution");
                return retVal;
        } //method
} //class
```

Note:

- ✓ Every AroundAdvice behaves like Servlet Filter component to servlet component/ JSP.
- ✓ Client App gets the dynamically generated InMemory Proxy class object and calls Target method. The overridden target method in Proxy class internally calls invoke (-) of Advice class object in order to apply secondary logic around target method execution.

Important/ control points:

- In advice logic placed in invoke (-) method, we can access and modify target method argument values.
- In advice logic placed in invoke (-) method, we can access and modify target method return value.
- In advice logic placed in invoke (-) method, we can control target method execution.

Note: In real project the AOP advices/ aspects will be applied on service class methods, not on DAO class methods because service class methods internally call DAO class methods. So, applying AOP advices/ aspects on service class methods indirectly applies on DAO class methods.

Procedure to developed Spring AOP Declarative Approach application:

a. Develop target class/ service class target method/ business method with business logic/ primary logic
b. Develop Aspect/ Advice class(es) having secondary logics
c. Develop Spring bean configuration file
    i. Configure target class as Spring bean
    ii. Configure advice/ aspect class as Spring bean

iii. Configure ProxyFactoryBean injecting target, advice class objects
- This internally activates Spring AOP and generates lnMemory class having overridden target methods of target class with both primary and secondary logics and also gives Proxy class object as Factory bean. (Performs weaving process and gives Proxy class object).

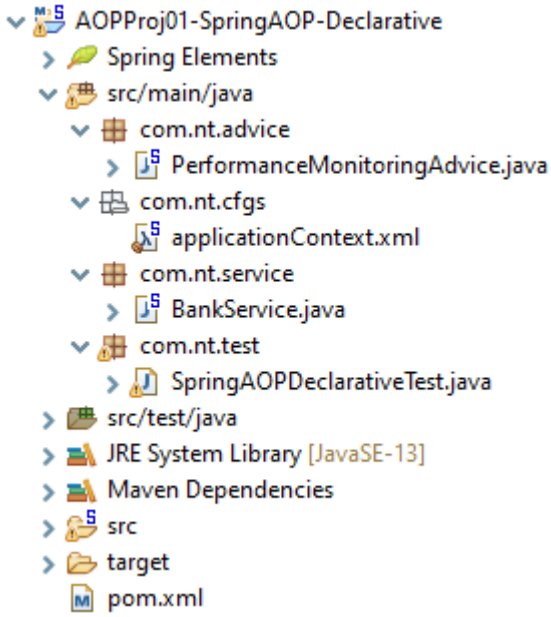
d. Develop Client App
   i. Create IoC container
   ii. Get Proxy class object from IoC container by calling getBean (-, -) with "ProxyFactoryBean class bean id"
   iii. Invoke target methods/ business methods on proxy class object.
   iv. Close IOC container

Note:

✓ ProxyFactoryBean
- Target, Advice classes(inputs)
- lnMemory Proxy class object (output)

✓ If spring-context-support-<version> as dependency through maven or gradle, the spring-aop-<version>.jar file will come automatically dependent jar files.

Directory Structure of AOPProj01-SpringAOP-Declarative:



- Develop the above directory structure and package, class, XML and add the jar dependencies in pom.xml file then use the following code with in their respective file.

pom.xml

```
<dependencies>
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-context-support</artifactId>
		<version>5.2.9.RELEASE</version>
	</dependency>
</dependencies>
```

PerformanceMonitoringAdvice.java

```
package com.nt.advice;

import java.util.Arrays;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
public class PerformanceMonitoringAdvice implements MethodInterceptor
{
	@Override
	public Object invoke(MethodInvocation invocation) throws
Throwable {
		long startTime = System.currentTimeMillis(); //Pre logics
		Object returnVal = invocation.proceed(); //invokes the target M
		long endTime = System.currentTimeMillis(); //Post logics
		System.out.println(invocation.getMethod().getName()+" with
args "+Arrays.toString(invocation.getArguments())+" has taken "+(endTime-
startTime)+"ms.");
		return returnVal;
	}
}
```

BankService.java

```
package com.nt.service;
public class BankService {
	public float calculateSimpleIntrestAmount(float pAmt, float rate,
float time) {
		System.out.println("BankService :
calculateSimpleIntrestAmount()");
		return (pAmt*rate*time)/100;
	}
}
```

applicationContext.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">

	<!-- Configure Target class -->
	<bean id="bankService" class="com.nt.service.BankService"/>

	<!-- Configure Advice class  -->
	<bean id="pmAdvice" class="com.nt.advice.PerformanceMonitoringAdvice"/>

	<!-- Configure ProxyFactoryBean -->
	<bean id="pfb" class="org.springframework.aop.framework.ProxyFactoryBean">
		<property name="target" ref="bankService"/>
		<property name="interceptorNames">
			<array>
				<value>pmAdvice</value>
			</array>
		</property>
	</bean>
</beans>
```

BankService.java

```
package com.nt.test;

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

import com.nt.service.BankService;

public class SpringAOPDeclarativeTest {

	public static void main(String[] args) {
		//Creat IoC container
		BankService proxy = null;
		ApplicationContext ctx = new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
		//get Service bean
```

```
                proxy = ctx.getBean("pfb", BankService.class);
                System.out.println("Interest amount:
 "+proxy.calculateSimpleIntrestAmount(100000, 2, 12));
        }

}
```

Note:

- ✓ From Spring 3.x, CGLIB Libraries that are required to generate proxy class are part spring libraries. So, there is no need of arranging them separately.
- ✓ By Default, Spring AOP uses CGLIB Libraries to generate the proxy class as the lnMemory sub class of Target class.
- ✓ If no pointcut is configured, the configured advices will be applied on all the methods of target class.

The flow of execution for Spring AOP AroundAdvice App:

BankService.java

```
public class BankService {
                                (q)
        public float calculateSimpleIntrestAmount(float pAmt, float rate, float
time) {
                System.out.println("BankService :
calculateSimpleIntrestAmount()");
                return (pAmt*rate*time)/100; (r)
        }
}
```

PerformanceMonitoringAdvice.java

```
public class PerformanceMonitoringAdvice implements MethodInterceptor {

        @Override   (o)
        public Object invoke(MethodInvocation invocation) throws Throwable {
                long startTime = System.currentTimeMillis(); //Pre logics
        (s)     Object returnVal = invocation.proceed(); //invokes the target
method                  (p)
                long endTime = System.currentTimeMillis(); //Post logics
                System.out.println(invocation.getMethod().getName()+" with args
"+Arrays.toString(invocation.getArguments())+" has taken "+(endTime-
startTime)+"ms.");
```

```
            return returnVal; (t)
        }

}
```

applicationContext.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">
                        (d) Pre-instantiation of singleton scope beans
        <!-- Configure Target class -->
        <bean id="bankService" class="com.nt.service.BankService"/> (d1)

        <!-- Configure Advice class  -->
        <bean id="pmAdvice"
class="com.nt.advice.PerformanceMonitoringAdvice"/> (d2)

        <!-- Configure ProxyFactoryBean -->
        <bean id="pfb"
class="org.springframework.aop.framework.ProxyFactoryBean"> (d3)
                <property name="target" ref="bankService"/>
                <property name="interceptorNames">
                        <array>
                                <value>pmAdvice</value>       (e)
                        </array>
                </property>
        </bean>
</beans>
```

Notice that it is factory bean class so, calls geObject() on ProxyFactoryBean class object that contains CGLIB code to generate Proxy class

InMemory Proxy class (f)

```
public class BankService$$EnhancerBySpringCGLIB$$bd2e8a6a extends
BankService implements ApplicationContextAware {
                                For AwareInjection to inject underlying IoC container
        private ApplicationContext ctx;              to Spring bean class object.

        public void setApplicationContext(ApplicationContext ctx){
                this.ctx=ctx;      Aware Injection (g)
        }
```

```
(m)     public float calcSim TeIntrestAmount(float pAmt,float rate, float time){
            //get Target class object
            BankService target =
                        ctx.getBean("bankService",BankSerivce.class);
            //get Advice class object
            PerformenceMonitoringAdvice advice=
            ctx.getBean("pmAdvice",PerformenceMonitoringAdvice.class);
            //get Method Details in the form of Method class object.
            Method method = target.getClass().
                    getDeclaredMethod("calcSimpleIntrestAmount");
            //prepare Methodlnovcation object having target class, target
            method details
            Methodlnovation invocation = new
                        CglibAopProxy$CglibMethodlnvocation();
            invocation.setMethod(method);
            invocation.setTarget(target);
            invocation.setArguments(pAmt, rate, time);
            //call invoke (-) on Advice class object
        (u) Object retVal=advice.invoke(invocation); (n)
            return retVal; (v)
        }
}
```

Internal Cache of IoC container (h) (j?)

bankService	com.nt.service.BankService object
pmAdvice	com.nt.advice.PerformanceMonitoring Advice object
pfb	InMemory Proxy class object BankService$$EnhancerBySpringCGLIB $$bd2e8a6a

Client App

```
public class SpringAOPDeclarativeTest {
                                    (a)
        public static void main(String[] args) {
            //Creat IoC container
```

```
        BankService proxy = null;
        ApplicationContext ctx = new (b)        (c) -> InMemory Meta data
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        //get Service bean
    (k) proxy = ctx.getBean("pfb", BankService.class); (i)
    (w) System.out.println("Interest amount:
"+proxy.calculateSimpleIntrestAmount(100000, 2, 12)); (l)
    }

}
```

- While working with Around advice, we can perform following operations on target method being from advice method
    a. We can access and modify target method argument values.
    b. We can access and modify target method return value.
    c. We can control target method execution.

PerformanceMonitoringAdvice.java

```
public class PerformanceMonitoringAdvice implements MethodInterceptor
{

    @Override
    public Object invoke(MethodInvocation invocation) throws
Throwable {
        System.out.println("PerformanceMonitoringAdvice.invoke()");
        long startTime = System.currentTimeMillis(); //Pre logics
        //To modify the target method arg value
        Object args[] = invocation.getArguments();
        if (((float)args[0])<50000)
            args[1] = ((float)args[1])-0.5f;
        //Controlling target method execution
        if (((float)args[0])<0 || ((float)args[1])<0 || ((float)args[2])<0)
            throw new IllegalArgumentException("Invalid inputs");
        Object returnVal = invocation.proceed(); //invokes the target
method
        long endTime = System.currentTimeMillis(); //Post logics
        System.out.println(invocation.getMethod().getName()+" with
args "+Arrays.toString(invocation.getArguments())+" has taken "+(endTime-
startTime)+"ms.");
        //To modifying the returnVal
```

```
                returnVal=((float)returnVal)+((float)returnVal)*0.5f;
                return returnVal;
        }

}
```

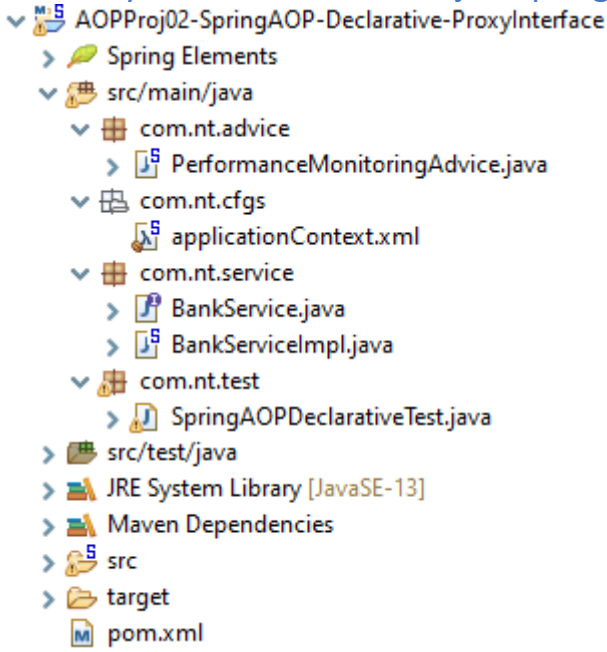
Limitations of using CGLIB Library to generate the Proxy class:

- Since the Proxy class comes as the sub class of target class, so we cannot take target class as final class. If we take the exception will be raised (This against strategy DP rule no:3 Code must be open for extension and must be closed)
  Exception in thread "main" org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'pfb': FactoryBean threw exception on object creation; nested exception is org.springframework.aop.framework.AopConfigException: Could not generate CGLIB subclass of class com.nt.service.BankService: Common causes of this problem include using a final class or a non-visible class; nested exception is java.lang.IllegalArgumentException: Cannot subclass final class com.nt.service.BankService
- Since we cannot override super class final methods in sub classes. So, we cannot take target methods as final methods if we take only target class method logics (primary logics) will execute, not the secondary logics (advice logics).

- To solve this problem, take support Proxy Interface (The interface implemented Target class) and make Spring AOP to use JDK libraries to generate Proxy class by implementing Proxy Interface not by extending from target class.

Converting CGLIB Libraries based application to JDK Libraries based application:
Step 1: Create Proxy interface having declaration of business method and make target class implementing that Proxy interface.
Step 2: Configure Proxy interface as input to ProxyFactoryBean class configuration.
Step 3: Develop client application to get Proxy class object to refer that object by using proxy interface reference variable.

Directory Structure of AOPProj02-SpringAOP-Declarative-ProxyInterface:



- Develop the above directory structure and package, class, XML and add the jar dependencies in pom.xml file then use the following code with in their respective file.
- Rest of code copy from previous project.

BankService.java

```
package com.nt.service;

public interface BankService {
	public float calculateSimpleIntrestAmount(float pAmt, float rate,
float time);
}
```

BankServiceImpl.java

```
package com.nt.service;
public final class BankServiceImpl implements BankService {

	public final float calculateSimpleIntrestAmount(float pAmt, float
rate, float time) {
		System.out.println("BankService :
calculateSimpleIntrestAmount()");
		return (pAmt*rate*time)/100;
	}
}
```

BankServiceImpl.java

```
        <!-- Configure Target class -->
        <bean id="bankService" class="com.nt.service.BankServiceImpl"/>

        <!-- Configure Advice class  -->
        <bean id="pmAdvice"
class="com.nt.advice.PerformanceMonitoringAdvice"/>

        <!-- Configure ProxyFactoryBean -->
        <bean id="pfb"
class="org.springframework.aop.framework.ProxyFactoryBean">
                <property name="target" ref="bankService"/>
                <property name="interceptorNames">
                        <array>
                                <value>pmAdvice</value>
                        </array>
                </property>
                <property name="proxyInterfaces">
                        <array>
                                <value>com.nt.service.BankService</value>
                        </array>
                </property>
        </bean>
```

The Generated Proxy class may look like this:

```
public class BankService$Proxy extends Proxy implements
                                        ApplicationContextAware, BankService {
        private ApplicationContext ctx;
        public void setApplicationContext(ApplicationContext ctx){
                this.ctx=ctx;
        }
        public float calcSimpleIntrestAmount(float pAmt,float rate,float time){
                //get Target class object
                BankServiceImpl target=
                        ctx.getBean("bankService",BankServiceImpl.class);
                //get Advice class object
                PerformenceMointoringAdvice advice=
                ctx.getBean("pmAdvice",PerformenceMonitoringAdvice.class);
                //get target method details
                Method method = target.getClass.getDeclaredMethod(“
```

```
                                    calcSimpleIntrestAmount");
                //prepare MethodInvocation object having target class and
                target method details
                MethodInvocation invocation = new
                ReflectiveMethodInvokation$CglibMethodInvokcation();
                invocation.setTarget(target);
                invocation.setMethod(method);
                invocation.setArgs(pAmt,rate,time);
                //call invoke(-) on advice class object
                Object retVal=advice.invoke(invocation);
                return retVal;
        }
}
```

Note: Spring AOP's ProxyFactoryBean uses CGLIB libraries to generate the proxy as sub class of target class if it notices that there is no interface implementation on target class, otherwise it uses JDK libraries to generate the proxy class the implementation class Proxy Interface.

AspectJ AOP

Limitations of Spring AOP:

a. Aspect/ Advice classes are invasive.
b. No support to work with annotations to make Spring beans as aspect/ advice classes.
c. Not the industry standard.

- To overcome these, we can use AspectJ AOP

Q. Differences between Spring AOP and Plain AspectJ AOP?

Ans.

Spring AOP	Plain AspectJ AOP
a) Aspect/ advice classes are invasive.	a) Aspect/ advice classes non-invasive.
b) No support for annotation driven aspect/ advice classes	b) Supports
c) Performs Runtime weaving i.e. proxy class will be generated dynamically at runtime.	c)Performs compile time weaving by using special AspectJC (AspectJ Compiler).

	.java (target) -> .class -> AspectJC -> Proxy class .java(advice) -> .class -> javac
d) Supports only methods as join points.	d) supports fields, constructors, methods as join points.
e) Supports both static and dynamic pointcuts.	e) Supports only static pointcuts.

Note:

- ✓ Static pointcut: Applying advice based on the method name.
- ✓ Dynamic pointcut: Applying advice based on method names, and their argument values.

Spring Integrated AspectJ AOP:

Instead of competing with AspectJ AOP, the Spring people have integrated AspectJ AOP with Spring framework. Due to this certain change have come in Spring Integrated AspectJ AOP. They are,

a. Performs runtime weaving i.e. Proxy class is generated dynamically at runtime. (No AspectJC compiler is required)
b. Still supports only static pointcuts
c. Supports only methods as join points.

- To work with AspectJ AOP, we need to add the following jar files to the BUILDPATH/ CLASSPATH as additional Jar files along with Spring core module jar files (spring-context-support-<ver>.jar).
  - aspectjweaver-<ver>.jar
  - aspectjrt-<ver>.jar

Different approaches of implementing Spring integrated AspectJ AOP:

a. Declarative approach (XML Configuration)
b. Annotation driven approach *
c. 100% code driven approach
d. Spring boot AspectJ AOP *

Spring Integrated AspectJ AOP Declarative Approach

- We must add aspectjrt-<ver>.jar, aspectjweaver-<ver>.jar file (aspectjrt-1.9.6.jar, aspectjweaver-1.9.6.jar)
- Import AOP namespace to Spring bean configuration file and use the following tags

<aop:config>: To enable AspectJ AOP.
<aop:pointcut >: To define pointcut as OGNL Expression.
OGNL: Object Graph Notation Language
<aop:aspect>: To configure Spring Bean as aspect class.
<aop:before>: To configure java method as before advice method.
<aop:around>: To configure java method as around advice method.
<aop:after-returning>: To configure java method as after advice method.
<aop:after-throwing>: To configure java method as throws advice method.

Spring Integrated AspectJ AOP - Around Advice

- This advice executes around the target method (i.e. before entering into target method and after leaving from target method).
- To develop this in AspectJ AOP we need to class having any method with the following signature

```
public Object <method-name> (ProceedingJoinPoint pjp) throws Throwable
                                    Holds he target class, target method
                                    details and ability to call target method
```

Advice = Aspect + action to perform

```
public class PerformenceMonitoringAspect {      //non-invasive
        //user-defined method with fixed signature.
        public Object monitor(ProceedingJoinPoint pjp)throws Throwable {

                long start=System.currentTimeMillis(); //pre logic
                Object retVal=pjp.proceed(); // invoking target method
                long end=System.currentTimeMillis(); //post logic
                S.o.p(pjp.getSignature()+"with args"+ Arrays.toString
                        (pjp.getArgs()) +" has taken "+(end-start)+" ms");
                return retVal;
        }
}
```

Control/ important points on around advice:

a. We can access and modify target method arguments.
b. We can access and modify target method return value
c. We can control target method execution.

In Spring Integrated AspectJ AOP pointcut will be written as OGNL expression having following syntax:

execution (<return type> <pkgname>.<target class name>.<method name>(..))

com.nt.service (pkg)
|-> BankService
|-> m1() int
|-> m2() String
|-> a1() float
|-> a2() long

Pointcut having all methods of BankService class:

execution (* com.nt.service.BankService.*(..))
any return type all methods

- Pointcut is collection of join points on whom we want to apply advices.

Pointcut having only m1(), m2() of BankService class:

execution (* com.nt.service.BankService.m*(..))

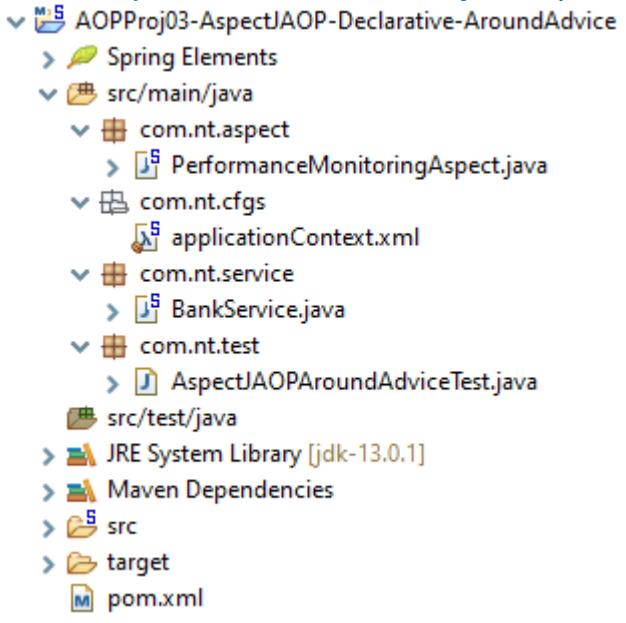
Pointcut having only m1() method of BankService class:

execution (int com.nt.service.BankService.m1(..))

Pointcut having all methods of all classes belonging com.nt.service pkg:

execution (* com.nt.service.*.*(..))

Directory Structure of AOPProj03-AspectJAOP-Declarative-AroundAdvice:



- Develop the above directory structure and package, class, XML and add the jar dependencies in pom.xml file then use the following code with in their respective file.

pom.xml

```
<dependencies>
	<dependency>
		<groupId>org.springframework</groupId>
		<artifactId>spring-context-support</artifactId>
		<version>5.2.9.RELEASE</version>
	</dependency>
	<dependency>
		<groupId>org.aspectj</groupId>
		<artifactId>aspectjweaver</artifactId>
		<version>1.9.6</version>
		<scope>runtime</scope>
	</dependency>
	<dependency>
		<groupId>org.aspectj</groupId>
		<artifactId>aspectjrt</artifactId>
		<version>1.9.6</version>
	</dependency>
</dependencies>
```

PerformanceMonitoringAspect.java

```
package com.nt.aspect;

import java.util.Arrays;
import org.aspectj.lang.ProceedingJoinPoint;
public class PerformanceMonitoringAspect {

	public Object monitor(ProceedingJoinPoint pjp) throws Throwable {
		long start = System.currentTimeMillis(); //Pre logic
		Object returnVal = pjp.proceed();
		long end = System.currentTimeMillis(); //Post logic
		System.out.println(pjp.getSignature()+" with args
"+Arrays.toString(pjp.getArgs())+" has taken "+(end-start)+" ms.");
		return returnVal;
	}
}
```

BankService.java

```
package com.nt.service;
public class BankService {
	public float calculateSimpleIntrestAmount(float pAmt, float rate,
float time) {
		System.out.println("BankService :
calculateSimpleIntrestAmount()");
		return (pAmt*rate*time)/100;
	}
}
```

applicationContext.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/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

	<!-- Configure service/ target class as spring bean -->
	<bean id="bankService" class="com.nt.service.BankService"/>

	<!-- Configure Aspect class as Spring bean -->
	<bean id="pmAspect"
class="com.nt.aspect.PerformanceMonitoringAspect"/>

	<!-- Enable AspectJ AOP -->
	<aop:config>
		<!-- Pointcut expression -->
		<aop:pointcut expression="execution(float
com.nt.service.BankService.calculateSimpleIntrestAmount(..))" id="ptc"/>
		<!-- Configure spring bean as AspectJ AOP advice/ aspect class -
->
		<aop:aspect ref="pmAspect">
			<aop:around method="monitor" pointcut-ref="ptc"/>
		</aop:aspect>
	</aop:config>

</beans>
```

AspectJAOPAroundAdvice.java

```
package com.nt.test;

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

import com.nt.service.BankService;

public class AspectJAOPAroundAdviceTest {

	public static void main(String[] args) {
		//create IoC container
		ApplicationContext ctx = new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
		//get Proxy class object
		BankService proxy = ctx.getBean("bankService",
BankService.class);
		System.out.println("Proxy object class name:
"+proxy.getClass());
		//invoke method on Proxy object
		System.out.println("Intrest amount is :
"+proxy.calculateSimpleIntrestAmount(100000, 2, 12));
		//close container
		((AbstractApplicationContext) ctx).close();
	}

}
```

Note: The <aop:config> tag code enables AspectJ AOP by converting "pmAspect" bean id-based Spring bean class as AspectJ AOP advice/aspect class and by taking its "monitor" method as around advice method and it will be applied on target method "calculateSimpleIntrestAmount" method of target class as specified in given "ptc" id based Pointcut expression.

Q. How Proxy class is generated in Spring Integrated AspectJ AOP Programming?

Ans. The IoC container creates In Memory metadata of Spring bean configuration file performs pre-instantiation of singleton scope beans (target and advice class) before keeping those objects in the internal cache of IoC container, it checks is there any tag in the Spring bean configuration file. Since

found, it collects class name and method names from pointcut expression specified in the and also makes specified class, method as Advice class and advice method (PerformenceMonitoringAspect and monitor () method). Uses CGLIB Libraries to generate Proxy class as the sub class of the class specified in Pointcut expression (BankService class), creates object for proxy class and keeps that object in the Internal cache of IoC container having target class bean id (BankService class bean id).

Internal cache of IoC container

	keys	values
0	bankService	BankService$$CGLIB class obj (Proxy class object)
1	pmAspect	PerformenceMonito Aspect class object
2		

AspectJ AOP Around advice Flow of execution:

BankService.java

```
public class BankService {          (s)
        public float calculateSimpleIntrestAmount(float pAmt, float rate, float
time) {
        System.out.println("BankService.calculateSimpleIntrestAmount()");
                return (pAmt*rate*time)/100.0f;        (t)
        }
}
```

PerformanceMointoringAspect.java

```
public class PerformanceMonitoringAspect {
                        (q)
        public Object monitor(ProceedingJoinPoint pjp) throws Throwable {
                long start = System.currentTimeMillis(); //Pre logic
        (u)     Object returnVal = pjp.proceed();        (r)
                long end = System.currentTimeMillis(); //Post logic
                System.out.println(pjp.getSignature()+" with args
"+Arrays.toString(pjp.getArgs())+" has taken "+(end-start)+" ms.");
                return returnVal;   (v)
        }

}
```

applicationContext.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.xsd
                http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

        <!-- Configure service/ target class as spring bean -->
        <bean id="bankService" class="com.nt.service.BankService"/>

        <!-- Configure Aspect class as Spring bean -->
        <bean id="pmAspect"
class="com.nt.aspect.PerformanceMonitoringAspect"/>

        <!-- Enable AspectJ AOP -->
        <aop:config>
                <!-- Pointcut expression -->
                <aop:pointcut expression="execution(float
com.nt.service.BankService.calculateSimpleIntrestAmount(..))" id="ptc"/>
                <!-- Configure spring bean as AspectJ AOP advice/ aspect class -->
                <aop:aspect ref="pmAspect">
                        <aop:around method="monitor" pointcut-ref="ptc"/>
                </aop:aspect>
        </aop:config>

</beans>
```

(d) Pre-instantiation singleton scope beans

(e) observers' aspect AOP is enabled

(f)

(g)

(j)

Internal cache of IoC container

0	bankService	BankService$$CGLIB class obj (Proxy class object)
1	pmAspect	PerformenceMonito Aspect class object
2		
	keys	values

(I?)

The Generated InMemory Proxy class (h) proxy class generation

```
public class BankService$CGLIBProxy extends BankService implements
ApplicationContextAware{
        private ApplicationContext ctx;
        public void setApplicationContext(ApplicationContext ctx){
                this.ctx=ctx;      (i) aware injection
        }                  (o)
        public void calculateSimpleIntrestAmount(float pAmt,float rate, float
time){
                //get Target class object
                BankService target=
                                ctx.getBean("bankService",BankService.class);
                //get Advice class object
                PerformenceMonitongAspect advice=ctx.getBean("pmAspect",
                                        PerformenceMonitoringAspect.class);
                //get Target Method
                Method method = target.getClass().getDeclaredMethod
                                        ("calculateSimpleIntrestAmount");
                //create ProceedingJoinPoint object
                ProceedingJoinPoint pjp=new JoinPointImpl$StaticPartImpl();
                pjp.setTarget(target);
                pjp.setSignature(method);
                pjp.setArgs(pAmt,rate,time);
                //collect advice method name from <aop:around> tag
                .....
                ..... gets "monitor" as the method name
                //invoke advice method on advice class object
        (w)     Object returnVal = advice.monitor(pjp); (p)
                return retrunVal;          (x)
        }
}
```

Client App

```
public class AspectJAOPAroundAdviceTest {
                        (a)
        public static void main(String[] args) {
                //create IoC container     (b)
                ApplicationContext ctx = new    (c) InMemory metadata
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
                //get Proxy class object
```

```
        (m)     BankService proxy = ctx.getBean("bankService",
BankService.class); (k)
                System.out.println("Proxy object class name: "+proxy.getClass());
                //invoke method on Proxy object
        (y)     System.out.println("Intrest amount is :
"+proxy.calculateSimpleIntrestAmount(100000, 2, 12));   (n)
                //close container
                ((AbstractApplicationContext) ctx).close();
        }
}
```

Take a Logging class:

AspectJAOPAroundAdvice.java

```
package com.nt.aspect;

import java.util.Arrays;

import org.aspectj.lang.ProceedingJoinPoint;

public class AroundLoggingAspect {

        public Object logging(ProceedingJoinPoint pjp) throws Throwable {
                System.out.println("Entering into "+pjp.getSignature()+" with
args "+Arrays.toString(pjp.getArgs()));
                Object returnVal = pjp.proceed();
                System.out.println("Entering into "+pjp.getSignature()+" with
args "+Arrays.toString(pjp.getArgs()));
                return returnVal;
        }

}
```

applicationContext.xml

```
        <!-- Configure Aspect class as Spring bean -->
        <bean id="pmAspect"
class="com.nt.aspect.PerformanceMonitoringAspect"/>
        <bean id="aLoggingAspect"
class="com.nt.aspect.AroundLoggingAspect"/>

        <!-- Enable AspectJ AOP -->
        <aop:config>
                <!-- Pointcut expression -->
```

```
		<aop:pointcut expression="execution(float
com.nt.service.BankService.*(..))" id="ptc"/>
		<!-- Configure spring bean as AspectJ AOP advice/ aspect class -
->
		<aop:aspect ref="pmAspect">
			<aop:around method="monitor" pointcut-ref="ptc"/>
		</aop:aspect>
		<!-- Configure spring bean as AspectJ AOP advice/ aspect class -
->
		<aop:aspect ref="aLoggingAspect">
			<aop:around method="logging" pointcut-ref="ptc"/>
		</aop:aspect>
	</aop:config>
```

Note: If, we apply multiple around advices on the target methods then they will be executed in the order configuration before getting into target method and they will be executed in the reverse order while leaving the target method.

Add another Method in Service class:
BankService.java

```
package com.nt.service;

public class BankService {

	public float calculateSimpleIntrestAmount(float pAmt, float rate,
float time) {

	System.out.println("BankService.calculateSimpleIntrestAmount()");
		return (pAmt * rate * time) / 100.0f;
	}

	public float calculateCompoundIntrestAmount(float pAmt, float rate,
float time) {

	System.out.println("BankService.calculateCompoundIntrestAmount()
");
		return (float) ((pAmt * Math.pow(1 + rate / 100, time)) - pAmt);
	}

}
```

- If you apply multiple around advice/ aspects on target methods. We must apply in proper order for example, it always recommended to apply performance monitoring around advice after around logging advice.
- In order to apply different advices on different target methods of target class, we can take multiple pointcuts and we can link them to different advice method as shown below.

applicationContext.xml

```
        <aop:config>
                <!-- Pointcut expression -->
                <aop:pointcut expression="execution(float
com.nt.service.BankService.*(..))" id="ptc"/>
                <aop:pointcut expression="execution(float
com.nt.service.BankService.calculateCompoundIntrestAmount(..))"
id="ptc1"/>

                <!-- Configure spring bean as AspectJ AOP advice/ aspect class -
->
                <aop:aspect ref="aLoggingAspect">
                        <aop:around method="logging" pointcut-ref="ptc"/>
                </aop:aspect>
                <!-- Configure spring bean as AspectJ AOP advice/ aspect class -
->
                <aop:aspect ref="pmAspect">
                        <aop:around method="monitor" pointcut-ref="ptc1"/>
                </aop:aspect>
        </aop:config>
```

- If, you feel there is reusability for pointcut expression then we can write pointcut expression directly in pointcut attribute of <aop:xxx> tags like <aop:around>, <aop:after>, <aop:before>, <aop:after-returning> and <aop:after-throwning> tags

```
<!-- Configure spring bean as AspectJ AOP advice/ aspect class -->
<aop:aspect ref="pmAspect">
        <aop:around method="monitor" pointcut="execution(float
com.nt.service.BankService.calculateCompoundIntrestAmount(..))"/>
</aop:aspect>
```

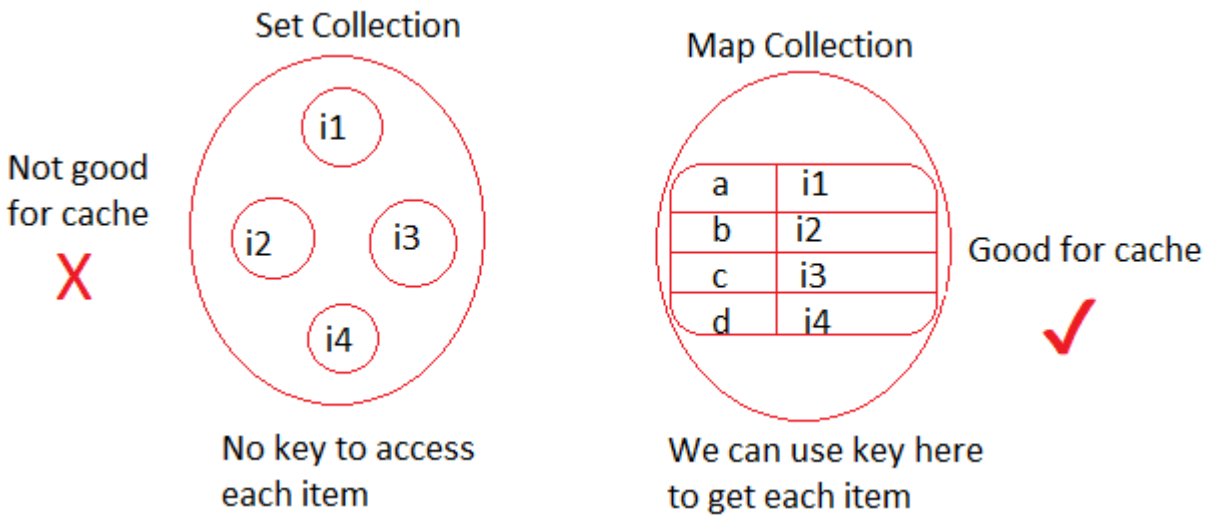
- While applying multiple advices on the target methods we can specify the order of executing advices either by physical configuration order of advices that are placed under tag or by using "order" attribute of <aop:aspect> tag. The "order" attribute priority value specified in tag works like high value indicates low priority and low value indicate high priority.

```
<!-- Configure spring bean as AspectJ AOP advice/ aspect class -->
<aop:aspect ref="pmAspect" order="10">
        <aop:around method="monitor" pointcut="execution(float
com.nt.service.BankService.calculateCompoundIntrestAmount(..))"/>
</aop:aspect>

<!-- Configure spring bean as AspectJ AOP advice/ aspect class -->
<aop:aspect ref="aLoggingAspect" order="5">
        <aop:around method="logging" pointcut-ref="ptc"/>
</aop:aspect>
```

Caching Advice:

- Cache is a temporary memory that gives reusability of different items.
- To implement Cache Map collection is best option because these the multiple items/ values/ objects taken for reusability for can have keys to identity and access.



- We can keep target business methods output/ result in the cache to reuse the output across the multiple calls of same business methods with same argument values.

Client App 1

(a5) proxy.calculateCompoundIntrestAmount(100000,2,12); (a1)

(b3) proxy.calculateCompoundIntrestAmount(100000,2,12); (b1)

Client App 2

(c3) proxy.calculateCompoundIntrestAmount(100000,2,12); (c1)

(d5) proxy.calculateCompoundIntrestAmount(200000,2,12); (d1)

Target class with target method

```
public class BankService {

	public float calculateSimpleIntrestAmount(float pAmt, float rate, float time) {
		System.out.println("BankService.calculateSimpleIntrestAmount()");
			return (pAmt * rate * time) / 100.0f;
		}

	public float calculateCompoundIntrestAmount(float pAmt, float rate, float time) {	(a3)	(d3)
		System.out.println("BankService.calculateCompoundIntrestAmount()");
			return (float) ((pAmt * Math.pow(1 + rate / 100, time)) - pAmt);
		}
```

Cache Advice/ Aspect

cacheMap (Cache as Map collection) (a2?) (b2?) (c2?) (d2?)

calculateCompoundIntrestAmount100000,2,12	2456.66	(a4)
calculateCompoundIntrestAmount200000,2,12	4434.44	(d4)

a1 to a5, b1 to b5, c1 to c5: all are same methods calls with same arguments

- If we are applying caching, logging, performance Monitoring on the target method, it is better to apply on the target method in the following order,
  - caching
  - logging
  - performance monitoring

Take a Caching class:

AspectJAOPAroundAdvice.java

```
package com.nt.aspect;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import org.aspectj.lang.ProceedingJoinPoint;

public class CachingAspect {

	private Map<String, Object> cacheMap = new HashMap<>();
	public Object caching(ProceedingJoinPoint pjp) throws Throwable {
		//Prepare the kry having  methodName and arg value
		String key = pjp.getSignature()+Arrays.toString(pjp.getArgs());
		Object returnVal =null;
		if (!cacheMap.containsKey(key)) {
			//If key is not there in the cacheMap
			System.out.println("From target Method");
			//invoke the target method
			returnVal = pjp.proceed();
			//put the results in cacheMap
			cacheMap.put(key, returnVal);
		}
		else {
			System.out.println("From cache");
			returnVal  = cacheMap.get(key);
		}
		return returnVal;
	}

}
```

applicationContext.xml

```
	<!-- Configure Aspect class as Spring bean -->
	<bean id="pmAspect"
class="com.nt.aspect.PerformanceMonitoringAspect"/>
	<bean id="aLoggingAspect"
class="com.nt.aspect.AroundLoggingAspect"/>
	<bean id="cachingAspect" class="com.nt.aspect.CachingAspect"/>
```

```
        <!-- Enable AspectJ AOP -->
        <aop:config>
                <!-- Pointcut expression -->
                <aop:pointcut expression="execution(float
com.nt.service.BankService.*(..))" id="ptc"/>
                <aop:pointcut expression="execution(float
com.nt.service.BankService.calculateCompoundIntrestAmount(..))"
id="ptc1"/>
                <!-- Configure spring bean as AspectJ AOP advice/ aspect class -
->
                <aop:aspect ref="cachingAspect" order="3">
                        <aop:around method="caching" pointcut="ptc1"/>
                </aop:aspect>
                <!-- Configure spring bean as AspectJ AOP advice/ aspect class -
->
                <aop:aspect ref="pmAspect" order="10">
                        <aop:around method="monitor" pointcut="ptc1"/>
                </aop:aspect
                <!-- Configure spring bean as AspectJ AOP advice/ aspect class -
->
                <aop:aspect ref="aLoggingAspect" order="5">
                        <aop:around method="logging" pointcut-ref="ptc"/>
                </aop:aspect>
        </aop:config>
```

Note: If multiple around advices are configuration with same order value (priority value) then they will be executed in the order of configuration that are placed in spring bean configuration file under <aop:config> tag.

To make AspectJ AOP application working with JDK libraries to generate proxy class:

Step 1: Take Interface having declarative of business methods and make target class implementing that interface.

Step 2: Write Pointcut expressions in applicationContext.xml by specifying Proxy interface instead of target class name.

Step 3: In client app, get Proxy class object and refer that object using Proxy interface reference variable.

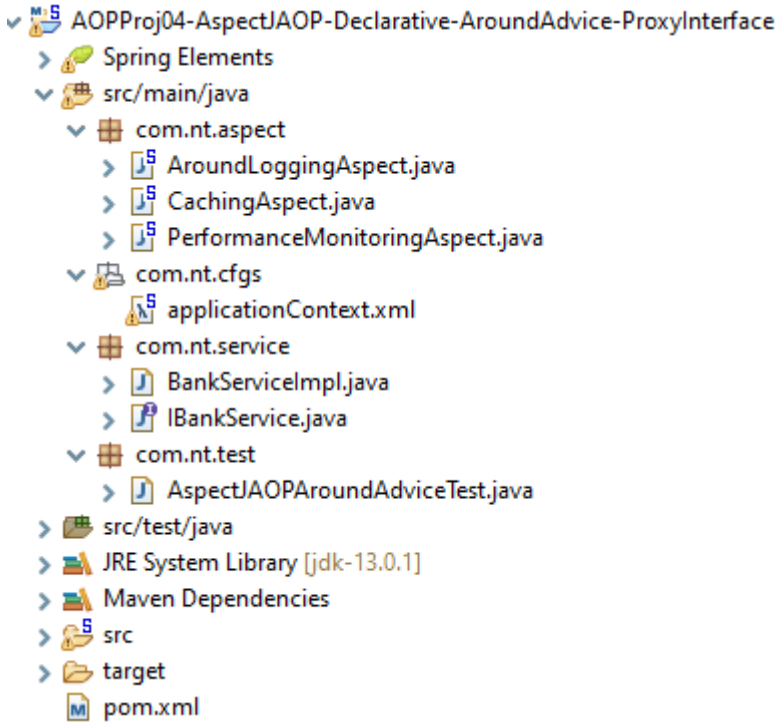
Note:

- ✓ If target class name is specified in the Pointcut OGNL expression, then

the proxy class will be generated as the sub class of target class by using CGLIB Libraries, due to this we cannot take target class as final class and target methods as final methods.

- ✓ If proxy interface name is specified in the Pointcut OGNL expression, then the proxy class will be generated as the implementation class of proxy Interface by using JDK Libraries, due to this we can take target class as final class and target methods as final methods.

Directory Structure of AOPProj04-AspectJAOP-Declarative-AroundAdvice-ProxyInterface:



- Develop the above directory structure and package, class, XML file then use the following code with in their respective file.
- Copy rest of code from the previous project.

IBankService.java

```
package com.nt.service;

public interface IBankService {
	public float calculateSimpleIntrestAmount(float pAmt, float rate,
float time);
	public float calculateCompoundIntrestAmount(float pAmt, float rate,
float time);
}
```

BankServiceImpl.java

```
package com.nt.service;

public class BankServiceImpl implements IBankService {

	public float calculateSimpleIntrestAmount(float pAmt, float rate,
float time) {
	System.out.println("BankService.calculateSimpleIntrestAmount()");
		return (pAmt * rate * time) / 100.0f;
	}

	public float calculateCompoundIntrestAmount(float pAmt, float rate,
float time) {
	System.out.println("BankService.calculateCompoundIntrestAmount()
");
		return (float) ((pAmt * Math.pow(1 + rate / 100, time)) - pAmt);
	}

}
```

applicationContext.xml

```
	<aop:pointcut expression="execution(float
com.nt.service.IBankService.*(..))" id="ptc"/>
		<aop:pointcut expression="execution(float
com.nt.service.IBankService.calculateCompoundIntrestAmount(..))"
id="ptc1"/>
```

Q. Though we are using Proxy Interface in Pointcut expression, how can u make AspectJ AOP Generating Proxy class as sub class of target class using CGLIB Libraries?

Ans. place proxy-target-class="true" in <aop:config> tag as shown below.

```
<aop:config proxy-target-class="true">
	....
	.......
</aop:config>
```

- Entire AOP Programing is designed based on Proxy design pattern which says Proxy is a structural design pattern that lets you provide a substitute or placeholder for another object. A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object.

Spring Integrated AspectJ AOP – Before Advice

- This advice executes before entering to target method.
- After executing this advice, the control automatically goes to target method.
- To develop this advice method, we must take advice method having the following signature in the advice/ aspect class.

public void <method name> (JoinPoint jp) throws Throwable (1)
(or)
public void <method name> (JoinPoint jp, Param1, Param2, ...) throws Throwable (2)
param names to hold target method argument values.
(or)
public void <method name> (Param1, Param2, ...) throws Throwable
param names to hold target method argument value (3)

Q. What is the difference between ProceedingJoinPoint and JoinPoint?

Ans. JoinPoint object allows to gather target method details like signature, argument values and etc. but does not allow to invoke target method by calling proceed () method. ProceedingJoinPoint object allows to gather target method details like signature, argument values and etc. and also allows to invoke target method by calling proceed () method.

org.aspectj.lan .JoinPoint (I)
|extends
org.aspectj.lang.ProceedingJoinPoint(I)

Note: Once the target method execution is over control will not go to advice method it will come to the caller of target method nothing but client App automatically

Use cases: Auditing, Security Check, Validation, Attendance monitoring, Test drive and etc.

Before Advice OGNL pointcut expression signature:

(execution <return type> <Pkg name>.<class name>.<method(s) name>(..))
(Suitable for (1) version of advice method signature)
(or)
(execution <return type> <Pkg name>.<class name>.<method(s) name>(..)) and args (param1,param2,param3 ,....)

target method param names whose names placed in advice method, use this syntax only when before advice method designed by using (2) or (3) version of signatures.

Control points:

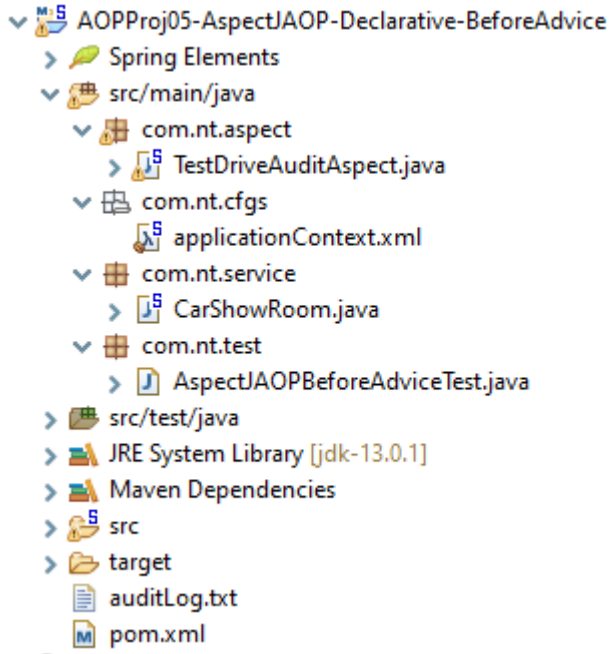
- We can access and modify target method argument values.
- We cannot control target method execution because the control automatically goes to target method once the advice method logics are executed. we can stop control going to target method from advice method by throwing an exception.
- We cannot get target method return value because after executing target method control will not come to advice method.

Before Advice (b)
Target method (c) Clint App
(d) proxy.targetmethod call (a)

Q. What is difference between Logging and auditing?

Ans. Logging keeps track of the code flow i.e. it tells the classes, methods and blocks involved in the code execution. Where auditing keeps user activities i.e. the operations done by end-user in the utilization of the application. Logging info written to log files where auditing info will be written to audit log file.

Directory Structure of AOPProj05-AspectJAOP-Declarative-BeforeAdvice:



- Develop the above directory structure and package, class, XML and add

the jar dependencies in pom.xml file then use the following code with in their respective file.

- Copy the pom.xml file from the previous project, because we are using the same jars.

CarShowRoom.java

```
package com.nt.service;

public class CarShowRoom {

	public String sale(String model, float price, String executive) {
		if (model.equalsIgnoreCase("baleno") ||
model.equalsIgnoreCase("briza"))
			return model+" car is sold out having "+price+" by
executive "+executive;
		else
			return model+" car is not available for sale";
	}

}
```

applicationContext.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/bean
s https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
		http://www.springframework.org/schema/aop
https://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
	<!-- Configure target class -->
	<bean id="showRoom" class="com.nt.service.CarShowRoom"/>

	<!-- Configure aspect class -->
	<bean id="testDriveAspect"
class="com.nt.aspect.TestDriveAuditAspect"/>

	<!-- Enable AOP -->
	<aop:config>
		<!-- Point cut expression -->
		<!-- <aop:pointcut expression="execution(java.lang.String
```

```
com.nt.service.CarShowRoom.sale(..))" id="ptc"/> -->
		<aop:pointcut expression="execution(java.lang.String
com.nt.service.CarShowRoom.sale(..)) and args(model,price,executive)"
id="ptc"/>
		<!-- Configure Spring bean as AspectJ AOP advice class -->
		<aop:aspect ref="testDriveAspect">
			<aop:before method="testDrive" pointcut-ref="ptc"/>
		</aop:aspect>
	</aop:config>

</beans>
```

TestDriveAuditAspect.java

```
package com.nt.aspect;

import java.io.FileWriter;
import java.time.LocalDateTime;

public class TestDriveAuditAspect {

	/*public void testDrive(JoinPoint jp) throws Throwable {
		//get Target method args
		Object args[] = jp.getArgs();
		//write info to audit Log file
		FileWriter writer = new FileWriter("auditLog.txt", true);
		writer.write(args[0]+" model car test drive is taken under the
monitoring of "+args[2]+" executive at "+LocalDateTime.now()+"\n");
		writer.flush();
		writer.close();
	}*/

	/*public void testDrive(JoinPoint jp, String model, float price, String
executive) throws Throwable {
		//write info to audit Log file
		FileWriter writer = new FileWriter("auditLog.txt", true);
		writer.write(model+" model car test drive is taken under the
monitoring of "+executive+" executive at "+LocalDateTime.now()+"\n");
		writer.flush();
		writer.close();
	}*/

	public void testDrive(String model, float price, String executive)
```

```
throws Throwable {
                //write info to audit Log file
                FileWriter writer = new FileWriter("auditLog.txt", true);
                writer.write(model+" model car test drive is taken under the
monitoring of "+executive+" executive at "+LocalDateTime.now()+"\n");
                writer.flush();
                writer.close();
        }

}
```

AspectJAOPBeforeAdviceTest.java

```
package com.nt.test;

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

import com.nt.service.CarShowRoom;

public class AspectJAOPBeforeAdviceTest {

        public static void main(String[] args) {
                //create IoC container
                ApplicationContext ctx = new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
                //get Proxy class object
                CarShowRoom proxy = ctx.getBean("showRoom",
CarShowRoom.class);
                //invoke method
                System.out.println(proxy.sale("briza", 1200000, "Harish"));
                //close container
                ((AbstractApplicationContext) ctx).close();
        }

}
```

Before Advice Flow of Execution:

CarShowRoom.java

```
public class CarShowRoom {      (t)
        public String sale(String model, float price, String executive) {
```

```
            if (model.equalsIgnoreCase("baleno") ||
model.equalsIgnoreCase("briza"))
                  return model+" car is sold out having "+price+" by executive
"+executive;
            else
                  return model+" car is not available for sale";
      } (u)

}
```

TestDriveAuditAspect.java

```
public class TestDriveAuditAspect {
                        (p)
      public void testDrive(String model, float price, String executive) throws
Throwable {
            //write info to audit Log file
            FileWriter writer = new FileWriter("auditLog.txt", true);
            writer.write(model+" model car test drive is taken under the
monitoring of "+executive+" executive at "+LocalDateTime.now()+"\n");
            writer.flush();
            writer.close();       (q)
      }

}
```

applicationContext.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
https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
            http://www.springframework.org/schema/aop
https://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

      <!-- Configure target class -->
      <bean id="showRoom" class="com.nt.service.CarShowRoom"/>
                  (d)  Pre-instantiation of Singleton Spring bean
      <!-- Configure aspect class -->
```

```
        <bean id="testDriveAspect"
class="com.nt.aspect.TestDriveAuditAspect"/>

        <!-- Enable AOP -->
        <aop:config>        (e) <aop:config> is observed
                <!-- Point cut expression -->
                <aop:pointcut expression="execution(java.lang.String
com.nt.service.CarShowRoom.sale(..)) and args(model,price,executive)"
id="ptc"/> (f)
                <!-- Configure Spring bean as AspectJ AOP advice class -->
                <aop:aspect ref="testDriveAspect"> (f)
                        <aop:before method="testDrive" pointcut-ref="ptc"/>
                </aop:aspect>
        </aop:config>

</beans>
```

InMemory Proxy class (g)

```
public class CarShowRoom$Proxy$CGLIB extends CarShowRoom implements
                                                        ApplicationContextAware {
        private ApplicationContext ctx;
        public void setApplicationContext(ApplicationContext ctx){
                this.ctx=ctx;       (h)
        }                       (n)
        public String sale (String model, float price, String executive){
                //get Target class object
                CarShowRoom target=
                        ctx.getBean("showRoom",CarShowRoom.class);
                //get Advice class object
                TestDriveAuditAspect advice=
                ctx.getBean("testDriveAspect",TestDriveAuditAspect.class);
                //get advice method name and signature from <aop:before>
                .....
                //gives testDrive(-,-,-)
                .....
                //invoke the advice method
        (r)     advice.testDrive(model,price,executive);  (o)
                //invoke target method
        (v)     String result = target.sale(model, price, executive); (s)
                return result;     (w)
```

```
        } //method
}//class
```

Internal Cache of IoC container (i)

testDriveAspect	TestDriveAuditAspe class object
showRoom	CarShowRoom$Proxy$CGLIB object

(k?)

AspectJAOPBeforeAdviceTest.java

```
public class AspectJAOPBeforeAdviceTest {
                    (a)
        public static void main(String[] args) {
                //create IoC container
                ApplicationContext ctx = new    (b)             (c)-InMemory
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        (l)     //get Proxy class object   (j)
                CarShowRoom proxy = ctx.getBean("showRoom",
CarShowRoom.class);
                //invoke method   (m)
        (x)     System.out.println(proxy.sale("briza", 1200000, "Harish"));
                //close container
                ((AbstractApplicationContext) ctx).close();
        }

}
```

AspectJ AOP - Security Check Before Advice:

- Security check means Authentication nothing but checking credentials of the users.
- Authentication should be done before getting into target method, so it should be developed as BeforeAdvice.
- For Authentication username, password details are required no business method gives them as the argument values. So, we should take and preserve them before business method calls through Sign-in Activity and use them when business method is called.

Client App

{ signln("raja","rani"); -> takes the username, password preserves them in Temporary place (Here No Authentication place, they are just preserved in temporary place).

withdraw (1001, 5000);

deposit (1002, 6000);

Before entering into these business methods collects the username, password from the place where they are preserved and uses them for authentication (Here by Using Before Advice the username, password preserved in temporary place will be collected and will be used for authentication, if authentication succeeded then business method executes other exception will be thrown).

signOut(); -> removes the username, password from the where they are temporarily preserved.

} In ATM Machine this process will takes place towards validating the PIN number.

If the application is web application:

- singln(-,-) method should place username, password in HttpSession object to preserve them temporarily.
- singOut(-,-) method should remove them from HttpSession object.

If the application is standalone application:

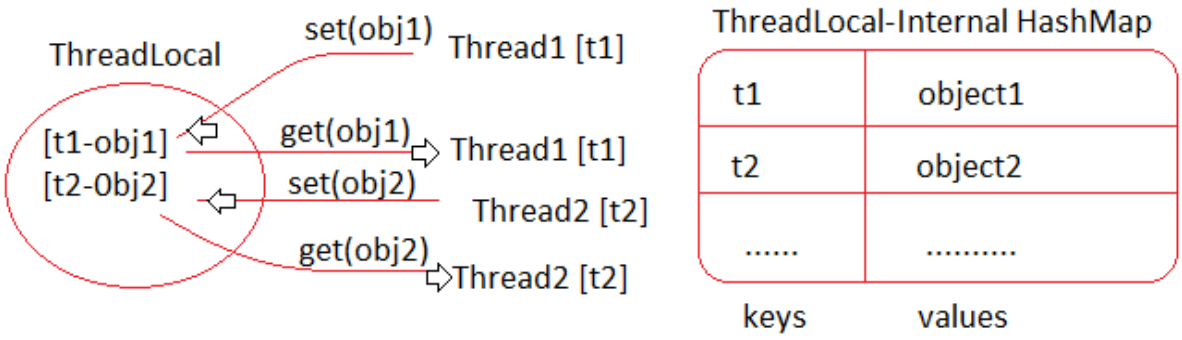
- singln(-,-) method should place username, password in ThreadLocal object to preserve them temporarily.
- singOut(-,-) method should remove them from ThreadLocal object.

Note:

- ✓ HttpSession object cannot be used in Standalone applications. It can be used only in web application
- ✓ ThreadLocal is given to store data specific to one thread and allows only that thread to get from ThreadLocal object. ThreadLocal is given java from 1.4 and generics supported is added from 1.6.
- ✓ Every thread can keep only one object in ThreadLocal. To place multiple values in ThreadLocal place them in single object and put that object in ThreadLocal.

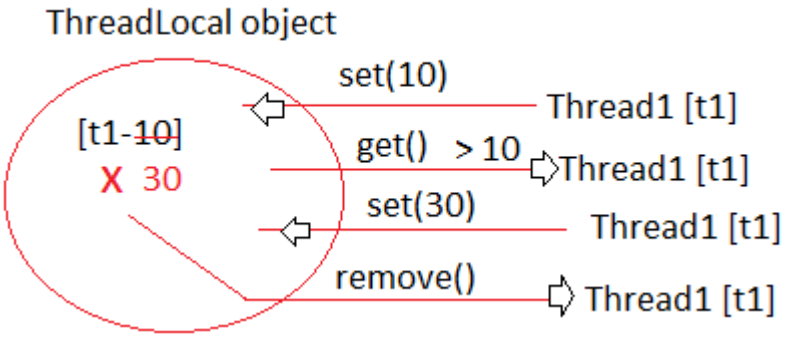
ThreadLocal:

ThreadLocal object is internally HashMap object having thread names as the keys and objects given as the values.



Example code (assume these methods are called by Thread1)

```
ThreadLocal <Integer> t1=new ThreadLocal ();
//To put data in ThreadLocal object (thread1)
t1.set(10);
//To get Data from ThreadLocal object
Integer i1=t1.get();
//To modify data from ThreadLocal obj
t1.set(30);
//To remove data from ThreadLocal obj
t1.remove();
```



Scopes in Standalone application:

a. Local scope (specific to method)
b. Instance scope (specific to object)
c. Class scope (specific to class - static)
d. Thread scope (specific to thread)

Scopes in Web application:

a. page scope
b. request scope
c. session scope
d. application scope

Security check advice as AspectJ AOP Before Advice:

Client App (standalone App)

signln("raja","rani"); keep them in ThreadLocal obj as single UserDetails class object but do not perform Authentication.

withdraw (1001,2000);

deposit (1001, 2000);

Execute SecurityCheckAdvice as BeforeAdvice before getting into to these target methods and perform Authentication if succeeded then execute these target method logics otherwise throw exception (BeforeAdvice/ Security Advice collects the username, password details from ThreadLocal object to perform authentication).

signOut(); remove UserDetails object from ThreadLocal object.

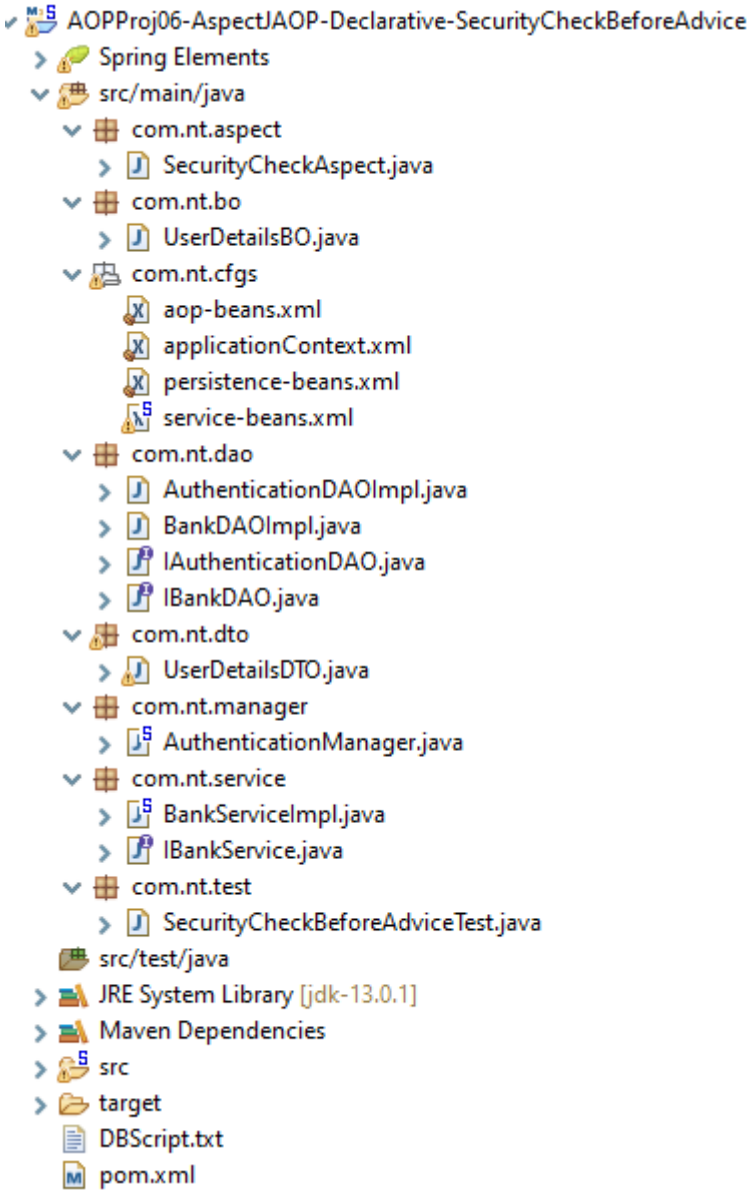
Application development components:

- IBankDAO.java, BankDAOImpl.java
- IBankService.java, BankSEvicelmpl.java persistence
  - withdraw, deposit operations related persistence, business logics
- IAuthenticationDAO.java, AuthenticationDAOImp.java
  - authentication logics
- SecurityCheckAspect.java
  - Performs Authentication Using Authentication Manager as BeforeAdvice
- AuthenticationManager.java
  - Contains multiple utility methods like singnln(-,-), singnOut(-,-) which can be by Client App to place/remove given username, password details to/from ThreadLocal object. Also contains authenticate (-) method to perform authentication to support SecurityCheckAdvice.
- applicationContext.xml
  - service-beans.xml
  - persistence-beans.xml
  - aop-beans.xml
- SecurityCheckBeforeAdviceTest.java

- DB tables
  - BankAccount
    - accno (n) (pk)
    - holdername (vc2)
    - balance (flaot)

- UserList
  - username (vc2) (pk)
  - password (vc2)

Directory Structure of AOPProj06-AspectJAOP-Declarative-SecurityCheckBeforeAdvice:



- Develop the above directory structure and package, class, XML and add the jar dependencies in pom.xml file then use the following code with in their respective file.

DBScript.txt

```
BankAccount
-----------------
```

```
CREATE TABLE "SYSTEM"."BANKACCOUNT"
  ("ACCOUNTNO" NUMBER(*,0) NOT NULL ENABLE,
        "HOLDERNAME" VARCHAR2(20 BYTE),
        "BALANCE" FLOAT(126),
         CONSTRAINT "BANKACCOUNT_PK" PRIMARY KEY ("ACCOUNTNO"));

UsersList
--------------------
 CREATE TABLE "SYSTEM"."USERSLIST"
  ("USERNAME" VARCHAR2(20 BYTE) NOT NULL ENABLE,
        "PASSWORD" VARCHAR2(20 BYTE),
         CONSTRAINT "USERSINFO_PK" PRIMARY KEY ("USERNAME"));
```

pom.xml

```
        <dependencies>
                <!--
https://mvnrepository.com/artifact/org.springframework/spring-context-
support -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-context-support</artifactId>
                        <version>5.2.9.RELEASE</version>
                </dependency>
                <!--
https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
                <dependency>
                        <groupId>org.springframework</groupId>
                        <artifactId>spring-jdbc</artifactId>
                        <version>5.2.9.RELEASE</version>
                </dependency>
                <!--
https://mvnrepository.com/artifact/com.oracle.database.jdbc/ojdbc6 -->
                <dependency>
                        <groupId>com.oracle.database.jdbc</groupId>
                        <artifactId>ojdbc6</artifactId>
                        <version>11.2.0.4</version>
                </dependency>
                <!--
https://mvnrepository.com/artifact/org.aspectj/aspectjweaver -->
                <dependency>
```

```
                <groupId>org.aspectj</groupId>
                <artifactId>aspectjweaver</artifactId>
                <version>1.9.6</version>
                <scope>runtime</scope>
            </dependency>
            <!-- https://mvnrepository.com/artifact/org.aspectj/aspectjrt -
->
            <dependency>
                <groupId>org.aspectj</groupId>
                <artifactId>aspectjrt</artifactId>
                <version>1.9.6</version>
            </dependency>
            <!-- https://mvnrepository.com/artifact/com.zaxxer/HikariCP --
>
            <dependency>
                <groupId>com.zaxxer</groupId>
                <artifactId>HikariCP</artifactId>
                <version>3.4.5</version>
            </dependency>
            <!--
https://mvnrepository.com/artifact/org.projectlombok/lombok -->
            <dependency>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <version>1.18.16</version>
                <scope>provided</scope>
            </dependency>
        </dependencies>
```

SecurityCheckAspect.java

```
package com.nt.aspect;

import org.aspectj.lang.JoinPoint;

import com.nt.manager.AuthenticationManager;

public class SecurityCheckAspect {

        private AuthenticationManager manager;

        public SecurityCheckAspect(AuthenticationManager manager) {
                this.manager = manager;
        }
```

```
	}

	public void check(JoinPoint jp) throws Throwable {
		if (!manager.validation())
			throw new IllegalArgumentException("Bad/ invalid
cradentials");
	}

}
```

UserDetailsBO.java

```
package com.nt.bo;

import lombok.Data;

@Data
public class UserDetailsBO {
	private String username;
	private String password;
}
```

aop-beans.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/bea
ns http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

	<!-- Configure Aspect -->
	<bean id="securityAspect"
class="com.nt.aspect.SecurityCheckAspect">
		<constructor-arg ref="authManager"/>
	</bean>

	<!-- Enable AOP -->
	<aop:config>
		<aop:pointcut expression="execution(java.lang.String
com.nt.service.IBankService.*(..))" id="ptc"/>
		<aop:aspect ref="securityAspect">
```

```
                    <aop:before method="check" pointcut-ref="ptc"/>
                </aop:aspect>
        </aop:config>

</beans>
```

applicationContext.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/bea
ns http://www.springframework.org/schema/beans/spring-beans.xsd">

        <import resource="persistence-beans.xml"/>
        <import resource="service-beans.xml"/>
        <import resource="aop-beans.xml"/>

</beans>
```

persistence-beans.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/bea
ns http://www.springframework.org/schema/beans/spring-beans.xsd">

        <!-- DataSource cofigutaion -->
        <bean id="hkCP" class="com.zaxxer.hikari.HikariDataSource">
                <property name="driverClassName"
value="oracle.jdbc.driver.OracleDriver"/>
                <property name="jdbcUrl"
value="jdbc:oracle:thin:@localhost:1521:xe"/>
                <property name="username" value="system"/>
                <property name="password" value="manager"/>
                <property name="minimumIdle" value="10"/>
                <property name="maximumPoolSize" value="100"/>
        </bean>

        <!-- JdbcTemplate configuration -->
        <bean id="jt" class="org.springframework.jdbc.core.JdbcTemplate">
                <property name="dataSource" ref="hkCP"/>
        </bean>
```

```
        <!-- DAO configuration -->
        <bean id="bankDAO" class="com.nt.dao.BankDAOImpl">
                <constructor-arg ref="jt"/>
        </bean>

        <bean id="authDAO" class="com.nt.dao.AuthenticationDAOImpl">
                <constructor-arg ref="jt"/>
        </bean>

</beans>
```

service-beans.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/bea
ns http://www.springframework.org/schema/beans/spring-beans.xsd">

        <!-- Configure Service class -->
        <bean id="bankService" class="com.nt.service.BankServiceImpl">
                <constructor-arg ref="bankDAO"/>
        </bean>

        <!-- Configure Manager class -->
        <bean id="authManager"
class="com.nt.manager.AuthenticationManager">
                <constructor-arg ref="authDAO"/>
        </bean>

</beans>
```

UserDetailsDTO.java

```
package com.nt.dto;

import java.io.Serializable;

import lombok.Data;

@Data
public class UserDetailsDTO  implements Serializable {
        private String username;
        private String password;
}
```

IBankDAO.java

```
package com.nt.dao;

public interface IBankDAO {
	public int withdraw(long acccountNo, double amount);
	public int deposit(long acccountNo, double amount);
}
```

BankDAOImpl.java

```
package com.nt.dao;
import org.springframework.jdbc.core.JdbcTemplate;
public class BankDAOImpl implements IBankDAO {
	private static final String WITHDRAW_QUERY="UPDATE
BANKACCOUNT SET BALANCE=BALANCE-? WHERE ACCOUNTNO=?";
	private static final String DEPOSIT_QUERY="UPDATE BANKACCOUNT
SET BALANCE=BALANCE+? WHERE ACCOUNTNO=?";
	private JdbcTemplate jt;
	public BankDAOImpl(JdbcTemplate jt) {
		this.jt = jt;
	}
	@Override
	public int withdraw(long acccountNo, double amount) {
		int count=0;
		count = jt.update(WITHDRAW_QUERY, amount, acccountNo);
		return count;
	}
	@Override
	public int deposit(long acccountNo, double amount) {
		int count=0;
		count = jt.update(DEPOSIT_QUERY, amount, acccountNo);
		return count;
	}
}
```

IAuthenticationDAO.java

```
package com.nt.dao;
import com.nt.bo.UserDetailsBO;
public interface IAuthenticationDAO {
	public int authenticate(UserDetailsBO bo);
}
```

AuthenticationDAOImpl.java

```
package com.nt.dao;

import org.springframework.jdbc.core.JdbcTemplate;

import com.nt.bo.UserDetailsBO;

public class AuthenticationDAOImpl implements IAuthenticationDAO {

	private static final String AUTH_QUERY = "SELECT COUNT(*) FROM USERSLIST WHERE USERNAME=? AND PASSWORD=?";

	private JdbcTemplate jt;

	public AuthenticationDAOImpl(JdbcTemplate jt) {
		this.jt = jt;
	}

	@Override
	public int authenticate(UserDetailsBO bo) {
		int count = jt.queryForObject(AUTH_QUERY, Integer.class, bo.getUsername(), bo.getPassword());
		return count;
	}

}
```

AuthenticationManager.java

```
package com.nt.manager;

import org.springframework.beans.BeanUtils;

import com.nt.bo.UserDetailsBO;
import com.nt.dao.IAuthenticationDAO;
import com.nt.dto.UserDetailsDTO;

public class AuthenticationManager {

	private ThreadLocal<UserDetailsDTO> threadLocal = new ThreadLocal<>();

	private IAuthenticationDAO dao;

	public AuthenticationManager(IAuthenticationDAO dao) {
		this.dao = dao;
	}
```

```
	public void signIn(String username, String password) {
		//convert username, password into userDetails object
		UserDetailsDTO dto = new UserDetailsDTO();
		dto.setUsername(username);
		dto.setPassword(password);
		//keep in ThreadLocal object
		threadLocal.set(dto);
	}

	public void singOut() {
		threadLocal.remove();
	}

	public boolean validation() {
		//get current Thread UserDetailsDTO object from ThreadLocal
		UserDetailsDTO dto = threadLocal.get();
		//convert dto to bo
		UserDetailsBO bo = new UserDetailsBO();
		BeanUtils.copyProperties(dto, bo);
		//use DAO
		int count = dao.authenticate(bo);
		return count==1?true:false;
	}

}
```

IBankService.java

```
package com.nt.service;
public interface IBankService {
	public String withdrawMoney(long acccountNo, double amount);
	public String depositMoney(long acccountNo, double amount);
}
```

BankServiceImpl.java

```
package com.nt.service;

import com.nt.dao.IBankDAO;

public class BankServiceImpl implements IBankService {

	private IBankDAO dao;

	public BankServiceImpl(IBankDAO dao) {
```

```
                this.dao = dao;
        }

        @Override
        public String withdrawMoney(long acccountNo, double amount) {
                int count = dao.withdraw(acccountNo, amount);
                return count==0?"Money not withdrawn":amount+" money is
withdrawn from the account: "+acccountNo;
        }

        @Override
        public String depositMoney(long acccountNo, double amount) {
                int count = dao.deposit(acccountNo, amount);
                return count==0?"Money not deposited":amount+" money is
desposited in the account: "+acccountNo;
        }

}
```

SecurityCheckBeforeAdviceTest.java

```
package com.nt.test;

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

import com.nt.manager.AuthenticationManager;
import com.nt.service.IBankService;

public class SecurityCheckBeforeAdviceTest {

        public static void main(String[] args) {
                //create IoC container
                ApplicationContext ctx = new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
                //get Proxy class object
                IBankService service = ctx.getBean("bankService",
IBankService.class);
                //get AuthenticationManager obejct
                AuthenticationManager manager =
ctx.getBean("authManager", AuthenticationManager.class);
                //invoke method
```

```
                try {
                        manager.signIn("nimu", "nimu@123");
                        System.out.println("Deposit operation:
"+service.depositMoney(1001, 10000));
                        System.out.println("Withdraw operation:
"+service.withdrawMoney(1001, 2000));
                } catch (Exception e) {
                        e.printStackTrace();
                }
                //close container
                ((AbstractApplicationContext) ctx).close();
        }
}
```

Spring Integrated AspectJ AOP – After Advice

- After Advice or also known as After Returning Advice.
- This advice executes after executing the target method logic and before it returns the return value to the caller.

```
public String bm1() {
        ..............
        ..... //business logic
        return "……";
}       At this place the AfterAdvice/ After Returning Advice will
        execute
```

Client app

```
String result=bm1();
```

- To develop this advice in AspectJ AOP environment we need to take advice method signature as shown below.

```
public void <method> (JoinPoint jp, <param to hold return value>) {
        ……      #1                                      #2
}
        (or)
public void <method> (<param to hold return value>) {
        .....
}
```

- The param name in the advice method to hold return value should reflect <aop:after-returning> tag as shown here.

```
<aop:aspect ref="....">
        <aop:after-returning method="...." returning="<param name>"
                                                #1              #2
                                                poincut-ref= "...." />
</aop:aspect>
```

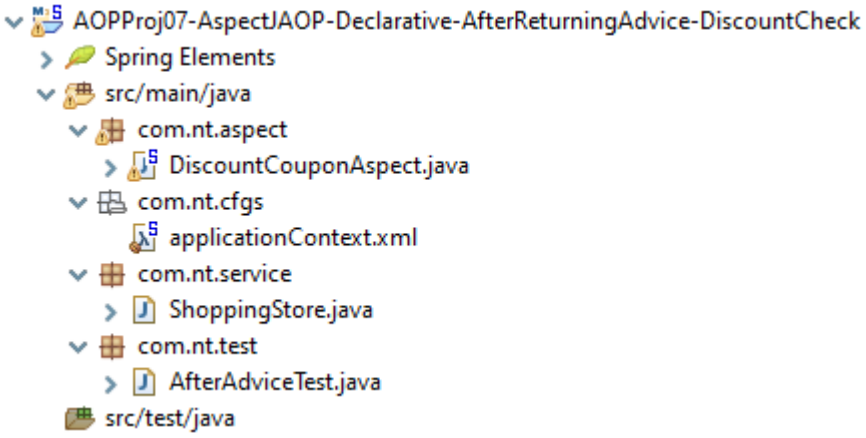
Important points/ Control points:

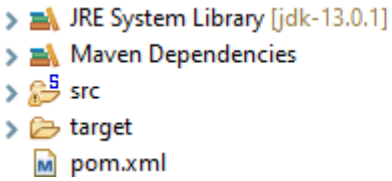
a. We can access target and modify target method argument values in advice method but there is no use because advice method logics execute after executing the entire business logic of target method.
b. We can access target method return value but we cannot modify (i.e. the return value modified in the advice method will not reflect to the caller).
c. We cannot control target method execution being from advice method, but by throwing exception in advice method we can stop return value of target method going to caller.

Realtime use cases:

1. Discount coupon generation for next purchase based on the generated amount.
2. Collecting feedback after Business Transaction.
3. Adding reward points to membership based on bill amount that is generated.
4. Providing scratch cards based on the bill amount that is generated.
5. Checking the weather generated ATM pin number is weak or strong. and etc.

Directory Structure of AOPProj07-AspectJAOP-Declarative-AfterReturningAdvice-DiscountCheck:





- Develop the above directory structure and package, class, XML file then use the following code with in their respective file.
- Copy pom.xml file from any previous project.

DiscountCouponAspect.xml

```
package com.nt.aspect;

import java.io.FileWriter;

import org.aspectj.lang.JoinPoint;

public class DiscountCouponAspect {

	/*public void coupon(JoinPoint jp, float billAmount) throws
Throwable {
		String couponMsg=null;
		//generate Coupon message
		if (billAmount<10000)
			couponMsg="Available 5% discount on the next
purchase";
		else if (billAmount<20000)
			couponMsg="Available 10% discount on the next
purchase";
		else if (billAmount<30000)
			couponMsg="Available 15% discount on the next
purchase";
		else
			couponMsg="Available 20% discount on the next
purchase";
		//Generate cupon
		FileWriter writer = new FileWriter("coupon.txt");
		writer.write(couponMsg);
		writer.flush();
		writer.close();
	}*/

	public void coupon(float billAmount) throws Throwable {
```

```
                String couponMsg=null;
                //generate Coupon message
                if (billAmount<10000)
                        couponMsg="Available 5% discount on the next
purchase";
                else if (billAmount<20000)
                        couponMsg="Available 10% discount on the next
purchase";
                else if (billAmount<30000)
                        couponMsg="Available 15% discount on the next
purchase";
                else
                        couponMsg="Available 20% discount on the next
purchase";
                //Generate coupon
                FileWriter writer = new FileWriter("coupon.txt");
                writer.write(couponMsg);
                writer.flush();
                writer.close();
        }

}
```

applicationContext.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/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd
                http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

        <!-- Target bean class configuration -->
        <bean id="shopping" class="com.nt.service.ShoppingStore"/>

        <!-- Aspect class configuration -->
        <bean id="couponAspect"
class="com.nt.aspect.DiscountCouponAspect"/>
```

```
        <!-- Enable AOP -->
        <aop:config>
                <aop:pointcut expression="execution(float
com.nt.service.ShoppingStore.shopping(..))" id="ptc"/>
                <aop:aspect ref="couponAspect">
                        <aop:after-returning method="coupon"
returning="billAmount" pointcut-ref="ptc"/>
                </aop:aspect>
        </aop:config>

</beans>
```

ShoppingStore.java

```
package com.nt.service;

public class ShoppingStore {

        public float shopping(String items[], float prices[]) {
                float billAmount = 0.0f;
                for (float p : prices)
                        billAmount=billAmount+p;
                return billAmount;
        }

}
```

AfterAdviceTest.java

```
package com.nt.test;

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

import com.nt.service.ShoppingStore;

public class AfterAdviceTest {

        public static void main(String[] args) {
                //create IoC container
                ApplicationContext ctx = new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
                //get Proxy object
```

```
                ShoppingStore storeProxy = ctx.getBean("shopping",
ShoppingStore.class);
                //invoke method
                System.out.println("Bill Aount: "+storeProxy.shopping(new
String[] {"shirt", "Trouser", "Belt" }, new float[] {5000, 3000, 1000}));

                //close container
                ((AbstractApplicationContext) ctx).close();
        }

}
```

Example application and code flow:

ShoppingStore.java

```
public class ShoppingStore {
                        (p)
        public float shopping(String items[], float prices[]) {
                float billAmount = 0.0f;
                for (float p : prices)
                        billAmount=billAmount+p;
                return billAmount;(q)
        }
}
```

DiscountCouponAspect.java

```
public class DiscountCouponAspect {
                        (t)
        public void coupon(float billAmount) throws Throwable {
                String couponMsg=null;
                //generate Coupon message
                if (billAmount<10000)
                        couponMsg="Available 5% discount on the next purchase";
                else if (billAmount<20000)
                        couponMsg="Available 10% discount on the next
purchase";
                else if (billAmount<30000)
                        couponMsg="Available 15% discount on the next
purchase";
                else
```

```
                        couponMsg="Available 20% discount on the next
purchase";
                //Generate cupon
                FileWriter writer = new FileWriter("coupon.txt");
                writer.write(couponMsg);
                writer.flush();
                writer.close();
        }
}
```

applicationContext.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.xsd
                http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
        <!-- Target bean class configuration -->
        <bean id="shopping" class="com.nt.service.ShoppingStore"/>
                                (d) pre-instantiation of singleton scope beans
        <!-- Aspect class configuration -->
        <bean id="couponAspect"
class="com.nt.aspect.DiscountCouponAspect"/>

        <!-- Enable AOP -->
        <aop:config>        (e) – notices this tag
                <aop:pointcut expression="execution(float
com.nt.service.ShoppingStore.shopping(..))" id="ptc"/>    (f)
                <aop:aspect ref="couponAspect"> (f)
                        <aop:after-returning method="coupon"
returning="billAmount" pointcut-ref="ptc"/>
                </aop:aspect>
        </aop:config>
</beans>
```

InMemory Proxy class (Generated by CGLIB Libraries) (g)

```
public class ShoppingStore$ProxyCGLIB extends ShoppingStore implements
ApplicationContextAware {        (h) aware injection
```

```
        private ApplicationContext ctx;

        public void setApplicationContext(ApplicationContext ctx){
                this.ctx=ctx;
        }
                        (n)
        public float shopping(String[] items, float[] prices)  {
                //get Target class object
                ShoppingStore target=
                        ctx.getBean("shopping",ShoppingStore.class);
                //get Advice class object
                DiscountCoupon advice=ctx.getBean("couponAspect",
                                                DiscountCouponAdvice.class);
                //get advice method name from <aop:after-returning> tag
                ……. gets coupon
                //invoke target method            (o)
        (r)     float billAmount =  target.shooping(items, prices);
                //invoke advice method
        (u)     advice.coupon(billAmt);   (s)
                return billAmt;  (v)
        }
}
```

Internal cache of IoC container (i)

shopping	Proxy class object ref ShoppingStore$ProxyCGLIB	(k?)
couponAdvice	DiscountCouponAdvice class obj f	

AfterAdviceTest.java

```
public class AfterAdviceTest {
                        (a)
        public static void main(String[] args) {
                //create IoC container (b) IoC container creation
                ApplicationContext ctx = new (c) InMemory meta data
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
                //get Proxy object          (j)
        (l)     ShoppingStore storeProxy = ctx.getBean("shopping",
ShoppingStore.class);
```

```
            //invoke method                    (m)
        (w) System.out.println("Bill Aount: "+storeProxy.shopping(new
String[] {"shirt", "Trouser", "Belt" }, new float[] {5000, 3000, 1000}));
            //close container
            ((AbstractApplicationContext) ctx).close();
        }
}
```

Spring Integrated AspectJ AOP - Throws Advice

- This advice executes when exception is raised in the target method execution.
- To develop this advice, we must take java class having following signature for the advice method.

```
public void <method name> (JoinPoint jp, <Exception/Throwable> type
                                                        param) {
        ……..    (best)
}
        (or)
public void <method name>(<Exception/Throwable> type param) {
        ……..
}
```

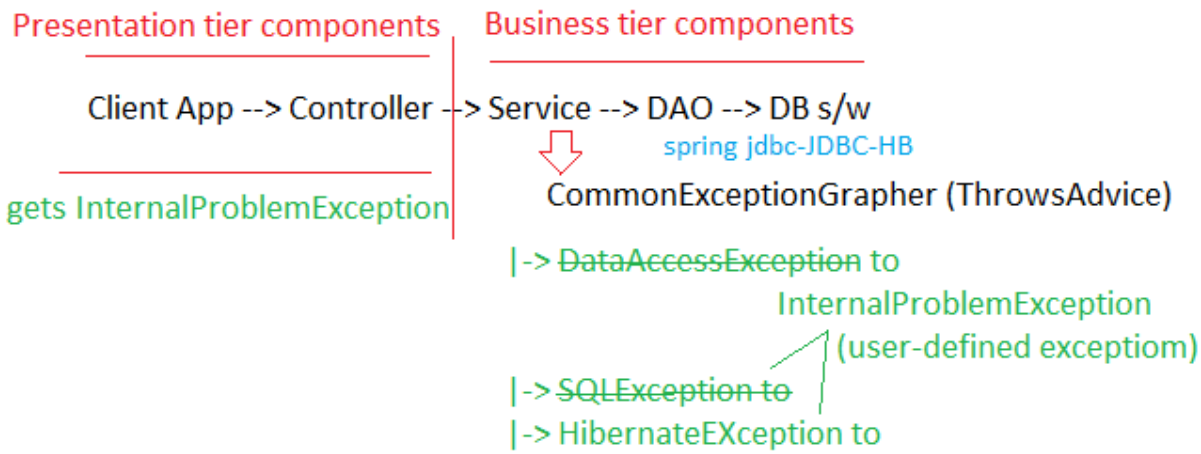
type param: To hold the raised exception in the target method

- Throws advice is not given to catch and handle the exception it is given to gather details of the exception that is raised in target method being from advice method.

Use cases:

a. CommonExceptionLogger (Writes only exception related log messages to separate log file. So, the Maintenance can understand Problem to fix the problem very firstly when the Production mode Application goes down).
b. CommonExceptionGrapher (It will translate the Project raised technology specific exception to Project specific user-defined exception).

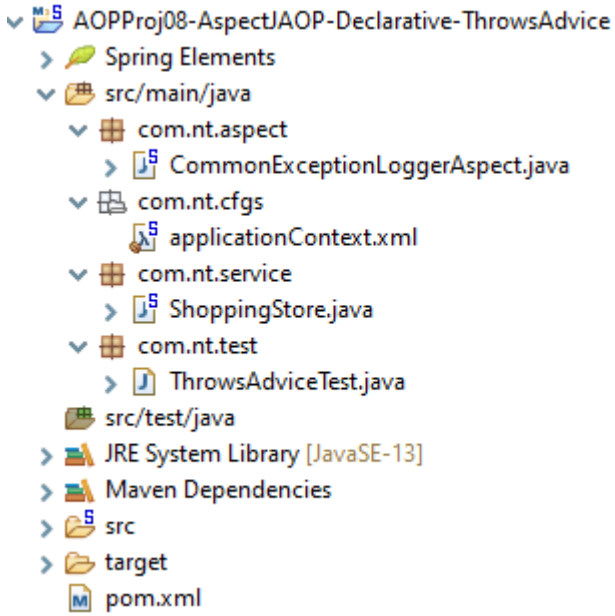
- CommonExceptionGrapher advice helps to achieve loose coupling between Business tier and presentation tier components i.e. the degree of dependency between business tier and presentation tier comps will be very less.



Control/ important points:

a. We can access and modify target method arguments but there is no use because control comes to advice method from target method only when exception is raised in the target method and does not go back target method.
b. We cannot access and modify the target method return value because the control comes to advice method from the middle of target method for exception that is raised.
c. We cannot control target method execution because control comes to advice method from target method itself for the exception raised in target method execution.
d. Throws Advice method/ logic not only executes for the exception raised in target method. It also executes for the exception raised in the other advice methods of target class.

Directory Structure of AOPProj08-AspectJAOP-Declarative-ThrowsAdvice:



- Develop the above directory structure and package, class, XML file then use the following code with in their respective file.
- Copy pom.xml file from any previous project.

ShoppingStore.java

```
package com.nt.service;

public class ShoppingStore {

	public float generateBill(String[] items, float[] prices) {
		boolean problem=false;
		if (items==null||prices==null)
			problem=true;
		for (float p : prices) {
			if (p<=0.0f) {
				problem=true;
				break;
			}
		}
		if (problem)
			throw new IllegalArgumentException("Invalid inputs");
		float billAmount=0.0f;
		for (float p : prices) {
			billAmount=billAmount+p;
		}
		return billAmount;
	}

}
```

CommonExceptionLoggerAspect.java

```
package com.nt.aspect;
import java.util.Arrays;
import org.aspectj.lang.JoinPoint;
public class CommonExceptionLoggerAspect {

	public void exceptionLogger(JoinPoint jp, Exception ex) {
		System.out.println(ex+" exception is rised in
"+jp.getSignature()+" with args "+Arrays.deepToString(jp.getArgs()));
	}

}
```

applicationContext.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 https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
		http://www.springframework.org/schema/aop https://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

	<!-- Target Service class -->
	<bean id="shopping" class="com.nt.service.ShoppingStore"/>

	<!-- Advice/ aspect class configuration -->
	<bean id="celAspect" class="com.nt.aspect.CommonExceptionLoggerAspect"/>

	<!-- Enable aop -->
	<aop:config>
		<aop:pointcut expression="execution(float com.nt.service.ShoppingStore.generateBill(..))" id="ptc"/>
		<aop:aspect ref="celAspect">
			<aop:after-throwing method="exceptionLogger" pointcut-ref="ptc" throwing="ex"/>
		</aop:aspect>
	</aop:config>

</beans>
```

ThrowsAdviceTest.java

```
package com.nt.test;

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

import com.nt.service.ShoppingStore;

public class ThrowsAdviceTest {


	public static void main(String[] args) {
		//create IoC container
```

```
		ApplicationContext ctx = new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
		//get Proxy object
		ShoppingStore storeProxy = ctx.getBean("shopping",
ShoppingStore.class);
		try {
			//invoke method
			System.out.println("Bill Aount:
"+storeProxy.generateBill(new String[] {"shirt", "Trouser", "Belt" }, new
float[] {5000, 3000, 1000}));
		} catch (IllegalArgumentException iae) {
			iae.printStackTrace();
		}
		//close container
		((AbstractApplicationContext) ctx).close();
	}

}
```

Code flow of execution:

ShoppingStore.java

```
public class ShoppingStore {
                        (p)
	public float generateBill(String[] items, float[] prices) {
		boolean problem=false;
		if (items==null||prices==null)
			problem=true;
		for (float p : prices) {
			if (p<=0.0f) {
				problem=true;
				break;
			}
		}
		if (problem)		(q)
			throw new IllegalArgumentException("Invalid inputs");
		float billAmount=0.0f;

		for (float p : prices) {
			billAmount=billAmount+p;
		}
```

```
            return billAmount;
        }

}
```

CommonExecptionLoggerAspect.java

```
public class CommonExceptionLoggerAspect {
                                    (s)
        public void exceptionLogger(JoinPoint jp, Exception ex) {
            System.out.println(ex+" exception is rised in "+jp.getSignature()+"
with args "+Arrays.deepToString(jp.getArgs()));
        }

}
```

applicationContext.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
https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
            http://www.springframework.org/schema/aop
https://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
        <!-- Target Service class -->
        <bean id="shopping" class="com.nt.service.ShoppingStore"/>
                            (d) pre-instantiation
        <!-- Advice/ aspect class configuration -->
        <bean id="celAspect"
class="com.nt.aspect.CommonExceptionLoggerAspect"/>
        <!-- Enable aop -->
        <aop:config>        (e) observers this
            <aop:pointcut expression="execution(float
com.nt.service.ShoppingStore.generateBill(..))" id="ptc"/> (f)
            <aop:aspect ref="celAspect">    (f)
                <aop:after-throwing method="exceptionLogger" pointcut-
ref="ptc" throwing="ex"/>
            </aop:aspect>
        </aop:config>
</beans>
```

InMemory Proxy class (g)

```
public class ShoppingStore$Proxy$CGLIB extends ShoppingStore implements
ApplicationContextAware {
        private ApplicationContext ctx;
        public void setApplicaitonContext(ApplicaitonContext ctx){
                this.ctx=ctx;      (h)
        }
                        (n)
        public float generateBill(String[] items, float[] prices){
                //get Target class object
                ShoppingStore target=ctx.getBean
                                        ("shopping",ShoppingStore.class);
                //get Advice class object
                CommonExceptionLogger advice=ctx.getBean
                        ("celAspect",CommonExceptionLogger.class);
                //get target method details
                Method method = target.getClass().
                                        getDeclaredMethod("generateBill");
                // create JoinPoint object
                JoinPoint jp=new MethodInvocationProceedingJoinPoint()
                jp.setTarget(target);
                jp.setSignature(method);
                jp.setArgs(new Object[]{items,prices});
                //invoke target method
                float billAmt=0.0f;
                try {                (o)
                        billAmt=target.generateBill(items,prices);
                } catch (IllegalArgumentException ex) {
                (t)     advice.exceptionLogger(jp, ex);  (r)
                }
                retunr billAmt;
        }
}
```

(k?) Internal Cache of IoC container (i)

shopping	Proxy class object (ShoppingStore object)
celAspect	CommonExceptionLoggerAspect object reference

ThrowsAdviceTest.java

```
public class ThrowsAdviceTest {
                                (a)
        public static void main(String[] args) {
                //create IoC container     (b)
                ApplicationContext ctx = new            (c)
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
                //get Proxy object            (j)
        (l)     ShoppingStore storeProxy = ctx.getBean("shopping",
ShoppingStore.class);
                try {
                        //invoke method (m)
                        System.out.println("Bill Aount:
"+storeProxy.generateBill(new String[] {"shirt", "Trouser", "Belt" }, new float[]
{5000, 3000, 1000}));
                } catch (IllegalArgumentException iae) {
                        iae.printStackTrace();      (s)
                }
                //close container
                ((AbstractApplicationContext) ctx).close();
        }
}
```

Note:

- ✓ If Exception class is immediate sub class of java.lang.Exception then it is called Checked Exception class.
- ✓ If Exception class is sub class of java.lang.RuntimeException then it is called Unchecked Exception class.
- ✓ CommonExceptionGrapher is capable converting one form of exception to another form of exception.
- ✓ It is generally converts the technology specific exception to the project specific user-defined exception.

Working with CommonExceptionGrapher:

CommonExceptionGrpaher.java

```
package com.nt.aspect;

import org.aspectj.lang.JoinPoint;

import com.nt.exception.InvalidInputsException;
```

```java
public class CommonExceptionGrapher {

	public void exceptionGrapher(JoinPoint jp, Exception ex) {
		throw new InvalidInputsException(ex.getMessage());
	}

}
```

InvalidInputsException.java

```java
package com.nt.exception;

public class InvalidInputsException extends RuntimeException {

	public InvalidInputsException(String msg) {
		super(msg);
	}

}
```

applicationContext.xml

```xml
	<!-- Advice/ aspect class configuration -->
	<bean id="celAspect"
class="com.nt.aspect.CommonExceptionLoggerAspect"/>

	<bean id="ceGrapher"
class="com.nt.aspect.CommonExceptionGrapher"/>

	<!-- Enable aop -->
	<aop:config>
		<aop:pointcut expression="execution(float
com.nt.service.ShoppingStore.generateBill(..))" id="ptc"/>
		<aop:aspect ref="celAspect">
			<aop:after-throwing method="exceptionLogger"
pointcut-ref="ptc" throwing="ex"/>
		</aop:aspect>

		<aop:aspect ref="ceGrapher">
			<aop:after-throwing method="exceptionGrapher"
pointcut-ref="ptc" throwing="ex"/>
		</aop:aspect>
	</aop:config>
```

Q. What happens if multiple throws Advice methods are given as overloaded methods in the same advice/ aspect class?

Ans. The method that is having a smaller number of parameters will get priority to execute. If the multiple overloaded methods are having same least number of parameters the exception will come. like this

Exception in thread "main" org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'shopping' defined in class path resource [com/nt/cfgs/applicationContext.xml]: BeanPostProcessor before instantiation of bean failed; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'org.springframework.aop.aspectj.AspectJPointcutAdvisor#0': Cannot create inner bean '(inner bean)#703580bf' of type [org.springframework.aop.aspectj.AspectJAfterThrowingAdvice] while setting constructor argument; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name '(inner bean)#703580bf': Cannot create inner bean '(inner bean)#67a20f67' of type [org.springframework.aop.config.MethodLocatingFactoryBean] while setting constructor argument; nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name '(inner bean)#67a20f67': Initialization of bean failed; nested exception is **java.lang.IllegalArgumentException: Cannot resolve method 'exceptionLogger' to a unique method. Attempted to resolve to overloaded method with the least number of parameters but there were 2 candidates.**

Summary table on 4 types of advices:

advice type	Ability to access and modify target method argument values	Ability to access and modify target method return value	Ability to control the target method execution	Real-time uses cases
Around Advice	Yes	Yes	Yes	Around Logging, caching, performance monitoring, etc.

Before Advice	Yes	NA	No (but) we can stop control going to target method from advice method by throwing exception	Auditing advice, Security advice, and Validation, etc.
AfterAdvice/ After Returning Advice	NA	We can access but not modify	NA	Discount coupon generation, ATM pin verifier, Key verifier, etc.
ThrowsAdvice	NA	NA	NA	Common Exception logger, Common exception grapher

Spring Integrated AspectJ AOP Annotation Driven Approach

Annotation driven AspectJ AOP programming also known as @Aspectj Programming.

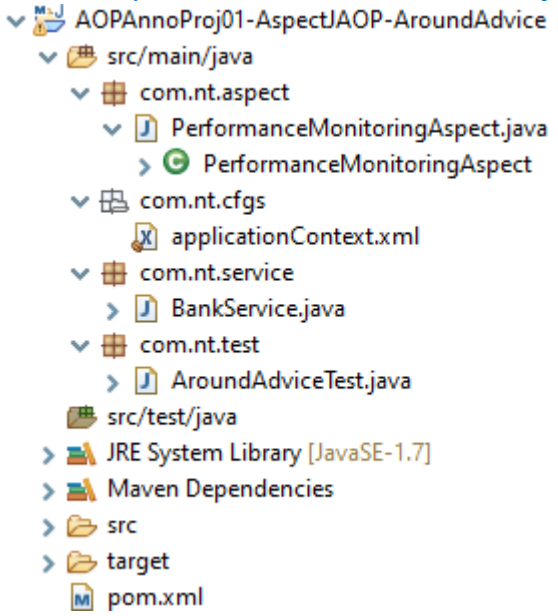
The list of annotations is:

- @Aspect: To make spring bean class AspectJ AOP aspect/advice class (like <aop:aspect>).
- @Pointcut: To define pointcut expression for reusability (like <aop:pointcut>).
- @Around: To make method as Around Advice method (like <aop:around>).
- @Before: To make method as Before Advice method (like <aop:before>).
- @AfterReturning: To make method as After Returning Advice method (like <aop:after-returning>).
- @AfterThrowing: To make method as Throws Advice method (like <aop:after-throwing>).
- @Order: To specify priority Order when multiple same type of advices applied on the same target methods we take @Order with priority value.

Here high value indicates low priority and low value indicates high priority (This will be decided among the given values).

Note: We must write/ place <aop:aspectj-autoproxy/> to enable AspectJ AOP in our application it is like placing <aop:config> tag in xml file.

Directory Structure of AOPAnnoProj01-AspectJAOP-AroundAdvice:



- Develop the above directory structure and package, class, XML file then use the following code with in their respective file.
- Copy pom.xml file from any previous project.

PerformanceMonitoringAspect.java

```
package com.nt.aspect;

import java.util.Arrays;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;

@Component("pmAspect")
@Aspect
public class PerformanceMonitoringAspect {

	@Around("execution(float com.nt.service.BankService.*(..))")
	public Object monitor(ProceedingJoinPoint pjp) throws Throwable {
```

```
		long start = System.currentTimeMillis();
		Object retVal = pjp.proceed();
		long end = System.currentTimeMillis();
		System.out.println(pjp.getSignature()+" with args
"+Arrays.deepToString(pjp.getArgs())+" has taken "+(end-start)+" ms.");
		return retVal;
	}

}
```

applicationContext.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"
	xmlns:context="http://www.springframework.org/schema/context"
	xsi:schemaLocation="http://www.springframework.org/schema/bean
s http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-4.3.xsd
		http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">

	<context:component-scan base-package="com.nt"/>

	<!-- Enables the use of the @AspectJ style of Spring AOP -->
	<aop:aspectj-autoproxy/>

</beans>
```

BankService.java

```
package com.nt.service;

import org.springframework.stereotype.Component;

@Component("bankService")
public class BankService {

	public float calculateSimpleIntrestAmount(float pAmt, float rate,
float time) {
	System.out.println("BankService.calculateSimpleIntrestAmount()");
		return (pAmt * rate * time) / 100.0f;
	}
```

```
	public float calculateCompoundIntrestAmount(float pAmt, float rate,
float time) {
		System.out.println("BankService.calculateCompoundIntrestAmount()
");
			return (float) ((pAmt * Math.pow(1 + rate / 100, time)) - pAmt);
		}

}
```

AroundAdviceTest.java

```
package com.nt.test;

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

import com.nt.service.BankService;

public class AroundAdviceTest {

	public static void main(String[] args) {
		//create IoC container
		ApplicationContext ctx = new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
		//get Proxy class object
		BankService proxy = ctx.getBean("bankService",
BankService.class);
		//invoke method on Proxy object
		System.out.println("Simple Intrest amount is :
"+proxy.calculateSimpleIntrestAmount(100000, 2, 12));
		System.out.println("----------------------------");
		System.out.println("Compound Intrest amount is :
"+proxy.calculateCompoundIntrestAmount(100000, 2, 12));

		//close container
		((AbstractApplicationContext) ctx).close();
	}

}
```

- Mention Order for that take more 2-aspect class and apply the @Order on the top of the Aspect/ advice class with different value.

AroundLoggingAspect.java

```
package com.nt.aspect;

import java.util.Arrays;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

@Component("logAspect")
@Aspect
@Order(5)
public class AroundLoggingAspect {

	@Around("execution(float com.nt.service.BankService.*(..))")
	public Object logging(ProceedingJoinPoint pjp) throws Throwable {
		System.out.println("Entering into "+pjp.getSignature()+" with
args "+Arrays.toString(pjp.getArgs()));
		Object returnVal = pjp.proceed();
		System.out.println("Entering into "+pjp.getSignature()+" with
args "+Arrays.toString(pjp.getArgs()));
		return returnVal;
	}

}
```

CachingAspect.java

```
package com.nt.aspect;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;
```

```java
@Component("cachingAspect")
@Aspect
@Order(3)
public class CachingAspect {

	private Map<String, Object> cacheMap = new HashMap<>();

	@Around("execution(float com.nt.service.BankService.*(..))")
	public Object caching(ProceedingJoinPoint pjp) throws Throwable {
		//Prepare the kry having  methodName and arg value
		String key = pjp.getSignature()+Arrays.toString(pjp.getArgs());
		Object returnVal =null;
		if (!cacheMap.containsKey(key)) {
			//If key is not there in the cacheMap
			System.out.println("From target Method");
			//invoke the target method
			returnVal = pjp.proceed();
			//put the results in cacheMap
			cacheMap.put(key, returnVal);
		}
		else {
			System.out.println("From cache");
			returnVal  = cacheMap.get(key);
		}
		return returnVal;
	}

}
```

PerformanceMonitoringAspect.java

```java
@Component("pmAspect")
@Aspect
@Order(10)
public class PerformanceMonitoringAspect {
```

Implementation of Proxy Interface:

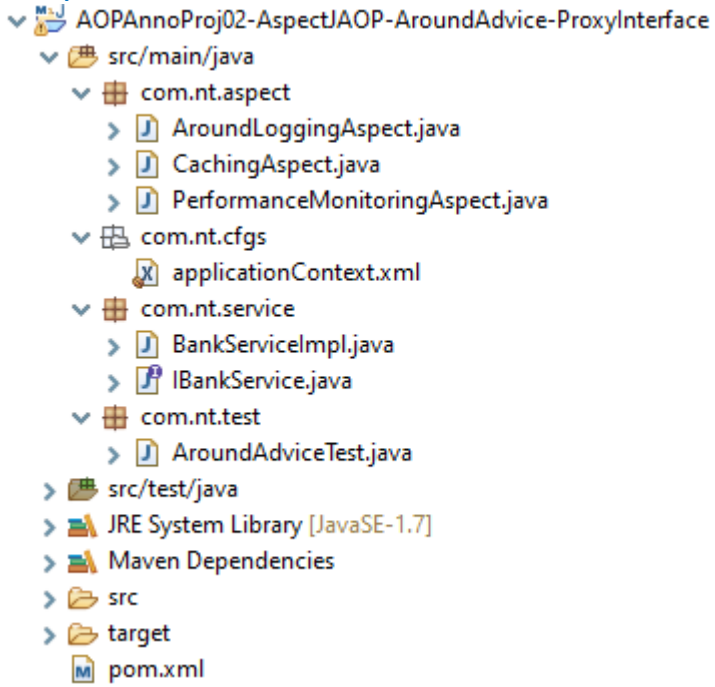
Step 1: Keep ready a project.

Step 2: Take an interface and implementing that interface take a class.

Step 3: Take interface name in pointcut expression.

Step 4: Run the application.

Directory Structure of AOPAnnoProj02-AspectJAOP-AroundAdvice-ProxyInterface:



- Develop the above directory structure and package, class, XML file then use the following code with in their respective file.
- Copy pom.xml file from any previous project.
- Copy rest of code from previous project.

PerformanceMonitoringAspect.java

```
public class PerformanceMonitoringAspect {

        @Around("execution(float com.nt.service.IBankService.*(..))")
        public Object monitor(ProceedingJoinPoint pjp) throws Throwable {
```

AroundLoggingAspect.java

```
public class AroundLoggingAspect {

        @Around("execution(float com.nt.service.IBankService.*(..))")
        public Object logging(ProceedingJoinPoint pjp) throws Throwable {
```

CachingAspect.java

```
public class CachingAspect {
        private Map<String, Object> cacheMap = new HashMap<>();
        @Around("execution(float com.nt.service.IBankService.*(..))")
        public Object caching(ProceedingJoinPoint pjp) throws Throwable {
```

IBankService.java

```
package com.nt.service;

public interface IBankService {

	public float calculateSimpleIntrestAmount(float pAmt, float rate,
float time);
	public float calculateCompoundIntrestAmount(float pAmt, float rate,
float time);

}
```

BankServiceImpl.java

```
package com.nt.service;

import org.springframework.stereotype.Component;

@Component("bankService")
public class BankServiceImpl implements IBankService{
	public float calculateSimpleIntrestAmount(float pAmt, float rate,
float time) {
	System.out.println("BankService.calculateSimpleIntrestAmount()");
		return (pAmt * rate * time) / 100.0f;
	}

	public float calculateCompoundIntrestAmount(float pAmt, float rate,
float time) {
	System.out.println("BankService.calculateCompoundIntrestAmount()
");
		return (float) ((pAmt * Math.pow(1 + rate / 100, time)) - pAmt);
	}
}
```

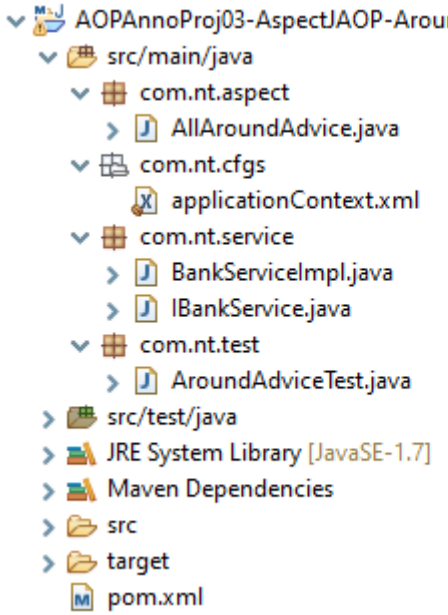
AroundAdviceTest.java

```
public class AroundAdviceTest {
	public static void main(String[] args) {
		//create IoC container
		ApplicationContext ctx = new
ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
		//get Proxy class object
		IBankService proxy = ctx.getBean("bankService",
IBankService.class);
```

Note:

- ✓ If we take Proxy Interface name in the pointcut expression, then we can generate proxy class as the sub class of target class using CGLIB or as the implementation class of Proxy Interface using JDK libraries.
- ✓ <aop:aspectj-autoproxy proxy-target-class="true”/> uses CGLIB Libraries to generate the proxy class as the sub class of target class (irrespective of whether Interface name is specified or not in the Pointcut expression).
- ✓ <aop:aspectj-autoproxy proxy-target-class="false"/> uses JDK Libraries to generate the proxy class as implementation class of the proxy interface that is specified in the Pointcut expression (we must specify interface name in the pointcut expression).

Directory Structure of AOPAnnoProj03-AspectJAOP-AroundAdvice-@Pointcut:



- Develop the above directory structure and package, class, XML file then use the following code with in their respective file.
- Copy pom.xml file from any previous project.
- Copy rest of code from previous project.

AllAroundAdvice.java

```
package com.nt.aspect;

import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
```

```
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Pointcut;
import org.springframework.core.annotation.Order;
import org.springframework.stereotype.Component;

@Component("allAround")
@Aspect
public class AllAroundAdvice {

	@Pointcut("execution(float com.nt.service.IBankService.*(..))")
	public void myPtc() {

	}

	@Around("myPtc()")
	@Order(100)
	public Object logging(ProceedingJoinPoint pjp) throws Throwable {
		System.out.println("Entering into "+pjp.getSignature()+" with
args "+Arrays.toString(pjp.getArgs()));
		Object returnVal = pjp.proceed();
		System.out.println("Entering into "+pjp.getSignature()+" with
args "+Arrays.toString(pjp.getArgs()));
		return returnVal;
	}

	private Map<String, Object> cacheMap = new HashMap<>();

	@Around("myPtc()")
	@Order(20)
	public Object caching(ProceedingJoinPoint pjp) throws Throwable {
		//Prepare the kry having  methodName and arg value
		String key = pjp.getSignature()+Arrays.toString(pjp.getArgs());
		Object returnVal =null;
		if (!cacheMap.containsKey(key)) {
			//If key is not there in the cacheMap
			System.out.println("From target Method");
			//invoke the target method
			returnVal = pjp.proceed();
			//put the results in cacheMap
			cacheMap.put(key, returnVal);
		}
```

```
                else {
                        System.out.println("From cache");
                        returnVal  = cacheMap.get(key);
                }
                return returnVal;
        }

        @Around("myPtc()")
        @Order(200)
        public Object monitor(ProceedingJoinPoint pjp) throws Throwable {
                long start = System.currentTimeMillis();
                Object retVal = pjp.proceed();
                long end = System.currentTimeMillis();
                System.out.println(pjp.getSignature()+" with args
"+Arrays.deepToString(pjp.getArgs())+" has taken "+(end-start)+" ms.");
                return retVal;
        }

}
```

Note: We can place multiple aspects/ advices in the same class as different methods, if all the aspect methods are using same pointcut expression then we can write it only on time on dummy method by using @Pointcut and specify that dummy method name as pointcut expression on all other advice methods.

While developing AspectJ AOP Annotation driven applications:

- Configure user-defined classes using stereo type annotations and link them with Spring bean configuration file using <context:component-scan> tag.
- Configure pre-defined classes as spring beans in Spring bean configuration file using <bean> tags.
- Use AspectJ AOP annotations in aspect/ advice class to make spring bean as AspectJ AOP advice class and place <aop:aspectj-autoproxy/> in Spring bean configuration file to enable the AspectJ AOP.

Directory Structure of AOPAnnoProj04-AspectJAOP-SecurityCheckBeforeAdvice:

- Copy paste the AOPProj06-AspectJAOP-Declarative-SecurityCheckBeforeAdvice application.
- Change the following things accordingly.

SecurityCheckAspect.java

```
@Component("securityAspect")
@Aspect
public class SecurityCheckAspect {

	@Autowired
	private AuthenticationManager manager;

	@Before("execution(java.lang.String
com.nt.service.IBankService.*(..))")
	public void check(JoinPoint jp) throws Throwable {
```

aop-beans.xml

```
	<context:component-scan base-package="com.nt.aspect"/>

	<aop:aspectj-autoproxy/>
```

persistence-beans.xml

```
	<context:component-scan base-package="com.nt.dao"/>
```

service-beans.xml

```
	<context:component-scan base-package="com.nt.service,
com.nt.manager"/>
```

BankDAOImpl.java

```
@Repository("bankDAO")
public class BankDAOImpl implements IBankDAO {


	@Autowired
	private JdbcTemplate jt;
```

AuthenticationDAOImpl.java

```
@Repository("authDAO")
public class AuthenticationDAOImpl implements IAuthenticationDAO {

	private static final String AUTH_QUERY = "SELECT COUNT(*) FROM
USERSLIST WHERE USERNAME=? AND PASSWORD=?";
	@Autowired
	private JdbcTemplate jt;
```

AuthenticationManager.java

```
@Component("authManager")
public class AuthenticationManager {

	@Autowired
	private IAuthenticationDAO dao;
```

BankServiceImpl.java

```
@Service("bankService")
public class BankServiceImpl implements IBankService {

	@Autowired
	private IBankDAO dao;
```

Spring Integrated AspectJ AOP 100% code Driven Approach

- Configure user-defined classes using stereo type annotations and link them with @Configuration class using @ComponentScan annotation.
- Configure pre-defined classes as spring beans using @Bean methods of @Configuration classes.
- Use AspectJ AOP annotations in aspect/ advice class to make spring bean as AspectJ AOP advice class and place @EnableAspectjAutoProxy (alternate to <aop:config> or <aop:aspectj-autoproxy/>) on the top of configuration class to enable AspectJ AOP.

Directory Structure of AOP100pProj01-AspectJAOP-AfterReturningAdvice-DiscountCheck:

- Copy paste the AOPProj07-AspectJAOP-Declarative-AfterReturningAdvice-DiscountCheck application.
- Change the following things accordingly.
- Remove all xml file and AppConfig class in com.nt.config pakage.

DiscountCouponAspect.java

```
@Component("couponAspect")
public class DiscountCouponAspect {

	@AfterReturning(pointcut = "execution(float
com.nt.service.ShoppingStore.shopping(..))", returning = "billAmount")
	public void coupon(float billAmount) throws Throwable {
```

AppConfig.java

```
package com.nt.config;

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

@Configuration
@ComponentScan({"com.nt.service", "com.nt.aspect"})
public class AppConfig {

}
```

ShoppingStore.java

```
@Service("shopping")
public class ShoppingStore {
```

AfterAdviceTest.java

```
public class AfterAdviceTest {

	public static void main(String[] args) {
		//create IoC container
		ApplicationContext ctx = new
AnnotationConfigApplicationContext(AppConfig.class);
```

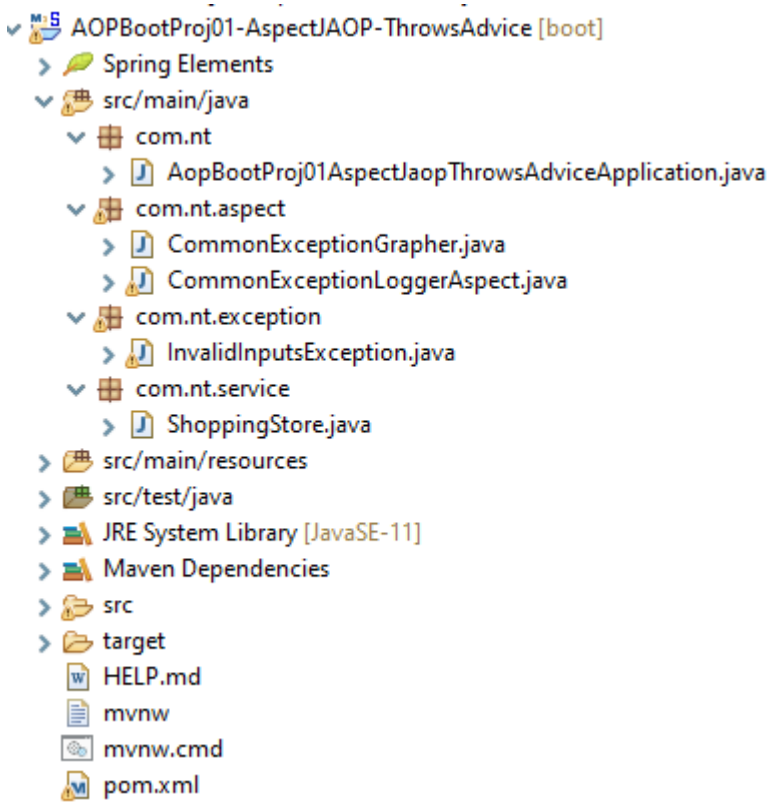
Spring Integrated AspectJ AOP Spring boot Approach

- Configure user-defined classes using stereo type annotations
- Configure pre-defined classes as Spring beans using @Bean methods of @Configuration classes, if they are not coming through AutoConfiguration.
- Use AspectJ AOP annotations in aspect/advice class to make spring bean as AspectJ AOP advice class

Note: No need of placing @EnableAspectjAutoProxy because that will be enabled automatically based on the jar files that are added.

Directory Structure of AOPBootProj01-AspectJAOP-ThrowsAdvice:

- Develop the below project as spring boot project.
- No need to add any Spring boot starter project add the regular jars.
- Copy the most of code form AOPProj08-AspectJAOP-Declarative-ThrowsAdvice project and change as following.



CommonExceptionGrapher.java

```
@Component("ceGrapher")
@Aspect
public class CommonExceptionGrapher {

	@AfterThrowing(pointcut = "execution(float
com.nt.service.ShoppingStore.generateBill(..))", throwing = "ex")
	public void exceptionGrapher(JoinPoint jp, Exception ex) {
```

CommonExceptionLoggerAspect.java

```
@Component("celAspect")
@Aspect
public class CommonExceptionLoggerAspect {

	@AfterThrowing(pointcut = "execution(float
com.nt.service.ShoppingStore.generateBill(..))", throwing = "ex")
	public void exceptionLogger(JoinPoint jp, Exception ex) {
```

ShoppingStore.java

```
@Service("shopping")
public class ShoppingStore {
```

AopBootProj01AspectJaopThrowsAdviceApplication.java

```
package com.nt;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.AbstractApplicationContext;

import com.nt.service.ShoppingStore;

@SpringBootApplication
public class AopBootProj01AspectJaopThrowsAdviceApplication {

	public static void main(String[] args) {
		// get container
		ApplicationContext ctx = SpringApplication.run(AopBootProj01AspectJaopThrowsAdviceApplication.class, args);
		// get Proxy object
		ShoppingStore storeProxy = ctx.getBean("shopping", ShoppingStore.class);
		try {
			// invoke method
			System.out.println("Bill Aount: " + storeProxy.generateBill(new String[] { "shirt", "Trouser", "Belt" },
					new float[] { 5000, 3000, 0 }));
		} catch (IllegalArgumentException iae) {
			iae.printStackTrace();
		} catch (Exception e) {
			e.printStackTrace();
		}
		// close container
		((AbstractApplicationContext) ctx).close();
	}

}
```

---------- The END ----------