AOP (Aspect Oriented Programming)
==========

=>AOP is not replacement OOP. In fact it compliments OOP i.e it is built on the top of OOP

=>The logics that are minimum and mandatory to complete the App are called primary logics.
eg:: Withdraw money ,deposite money , transfer money

=> The logics that are additional ,optional anc configurable are called secondary logics
eg: Auditing , logging ,Performence Monitoring and etc..

=>The secondary logics are also called as Middlware services, Aspects, CrossCutting concerns

AOP style Programming
==========

It talks about seperating Secondary logics from primary logics at development , But will be mixedup dynamically at run time.. with Support AOP enable softwares like AspectjAOP, JbossAOP, Spring AOP and etc..

Main class /Target class
==========

```
public class StockMarket{

  public String openDMATAccount(){
    //primary logic
     -->logic to open the account
  }
  public String purchaseShares(String[] shareNames){
     //primary logic
       --> logic to purchase the shares
  }
}
```

Service/B.Methods having B.logic/service logic

```
public class Logging{
   ....
   ... //logging logic
   ....
}

public class Auditing{
   ....
   .... //Auditing logic
   ....
}

public class Security{
   ....
   ... //security logic
   ...
}
```

AOP enabled software (Spring AOP/AspectjAOP) — xml file or annotations

Generates the InMemory Proxy class

```
public class   StockMarket$Proxy extends  StockMarket{

   public String openDMATAccount(){
      ....
      ... //both primary logic and secondary logics
      ...
   }
   public  String purchaseShares(String [] shares){
      ....
      ... both primary and secondary logics will be mixdup
      ....
   }
}
```

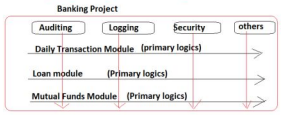
This class called InMemory proxy class becoz it will be generated ,compiled ,loaded and instantiated dynamically at runtime in the Memory where App is executing.. (that is JVM memory of the RAM)

AOP Advantages
==========

=>Secondary logics are seperated from primary logics, Code is not clumzy
=> parellel development is possible
=> reusability of secondary logics
=> we can enable or disable secondary logics on primary logics with out touching the source code target class .. we can control this by giving instructions to AOP enabled software by using xml files or annotations.
=> It is industry standard
=> we can add more primary logics and more secondary logics andwe can mix them dyamically at run time...
and etc..

OOP style programming is like :: Consuming readymade Tea
AOP style programming is like :: Consuming FIVE star hotel tea

Banking Project



In any Project Primary logics are horizontals ... and secondary logics are verticals

So these are also called Cross cutting concerns

=>The languages that are given based on procedure oriented programming principles are called Procedure Orienter Programming languages (POP languages)
eg: c, basic, cobol and etc..

=>The languages that are given based on object oriented programming principles are called Object Orienter Programming languages (OOP languages)
eg: c++,java,c# and etc..

oop priciples :: class,object, Encapsulation, abstraction,inheritence,polymorphism and etc.. are called OOP languages...

=>The frameworks or softwares that are given based AOP principles are called AOP enabled frameworks or softwares
eg: Spring AOP , AspectjAOP, JbossAOP and etc..

AOP Priciples/AOP Terminologies
==========

Aspect
Advice
JoinPoint
Pointcut
weaving
Target calss
Proxy class

Aspect (What u want to apply?)
======

=> The classes/comps that are having secondary logics are called Aspects /corss cutting concerns /middleware services..
eg: Logging -->keeps track of code flow
Auditing -> keeps track of user's flow
Performence Monitoring --> evaluates speed of the logic execution
and etc..

Advice (How u want to apply the aspect)
======

The action specified for Aspect is called Advice..i.e advice tells when /how to apply the aspect
like appy aspect before/after entering /leaving to/from method
AOP supports 4 types advices
==========
Around Advice (Aspect logics should execute around the target method nothing before and after)
BeforeAdvice
AfterReturningAdvice/AfterAdvice
ThrowsAdvice (Advice should execute when the exception is raised in the target method)

JoinPoint
=======

The possible places in target class where aspects can be adviced (applied)..
like target class methods, constructors , fields
note : spring supports only methods as joinpoints..

(Talks about where we can apply)

```
class BankService{
   public void m1(){
     ..
   }
   public void m2(){
     ..
   }
   public void m3(){
     ..
   }
}
```

join points

becoz max times methods only maintain primary logics..

Pointcut
=======

Collection of Join point where the aspects are advised i.e indicates out of multiple joinpoints of a target class the aspects are applied on certain joinpoints that are represented by pointcut

Talks about (Where we have applied)

if target class having 10 methods we can say there are 10 join points.. In that if we want apply advices/apsects only 3 methods .. then their names placed in pointcut..

Target class ::
==========

The main class that contains target methods/B.methods having primary logics

=>pre AOP Main class is called target class..

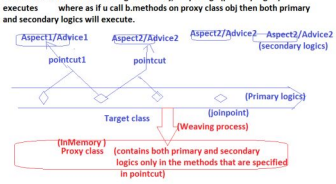
Weaving ::
=========

The process mixing up seconary logics with primary logics dynamically at runtime and generating Proxy classs is called weaving process..

Proxy class ::
==========

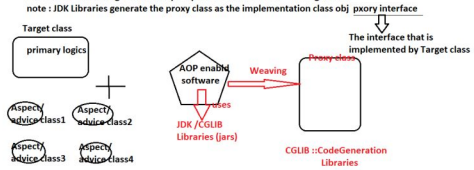
The Outcome of weaving process..
=>POST AOP class is called Proxy class..

note:: if u call b.methods on target class obj, only b.logic(primary logics) executes where as if u call b.methods on proxy class obj then both primary and secondary logics will execute.



The AOP enable software internally use different libraries to generate InMemory Proxy classes like CGLIB libraries (part of spring from 4.x onwods), and Jdk libraries (part of java)

note: CGLIB libraries generate the proxy class as the sub class of target class
note : JDK Libraries generate the proxy class as the implementation class obj pxory interface



The interface that is implemented by Target class

t

=> Spring framework supports Spring AOP, AspectjAOP model AOP style programming
=>Instead of competing with Aspectj AOP , the spring framworks integrrated with Aspectj AOP with spring framework from 2.x version onwords..

In spring we can AOP style app in 8 ways
==========

a) Spring AOP Programatic approach(core java style)
b) Spring AOP Declarative Approach (xml driven)
c) spring AOP in 100% code or Java config Approach
d) Spring AOP in spring BootDriven Approach

In spring AOP there are no annotations...

e) Spring Integrated AspectjAOP Declarative Approach
f) spring integrated AspectjAOP Annotation driven Approach
(g) spring integrated AspectjAOP 100% code driven /Java Config Approach
(h) spring integrated AspectjAOP spring Boot Approach

Different types of Advices
==========

a) Around Advice
==========
==> The apsect logics will execute around the target method (before and after)
usecase:: Performence Monitoring

b) Before Advice
==========
==>==> The apsect logics will execute before entering to target method
usecase :: secuirty

c) After Advice
==========
==> The apsect logics will execute after execution of the target method..
usecase :: generating discount cupon for next purchase based current bill amount

d) Throws advice
==> The apsect logics will execute when exception is raised in the target method.
usecase :: Exception Logger for incident Management..

Around Advice using Spring AOP Declarative Approach (xml driven cfgs..)
==========

--> This Advice/aspect logics will execute around the target method...
--> To develop this in spring , Take a class implementing pkg.MethodInterceptor(I)

```
public class PerformenceMonitoringAdvice implements MethodInterceptor{
                                                    makes class as aroung advice class..
 @Override
 public Object invoke(MethodInvocation invocation)throws Exception{
   long start=0,end=0;          Holds the Target method details..
   Object retVal=null;
   start=System.currentTimeMillis(); //pre- logic

    retVal=invocation.proceed();  //calls target method

   end=System.currentTimeMills();  //post logc
   S.o.p(invocation.getMethod().getName()+" has taken "+ (end-start)+" ms  for execution");
 }
        =>spring aop advice /aspect classes are invasive classes becoz they are
        implementing spring api interfaces...
```

usecases:: PErfromence Monitoring , Around Logging , Caching , Transaction Management

=>Entire AOP is designed around Proxy Design Pattern..
(spring AOP/AspectjAOP) ->Which says add new functionalaties dynamically at runtime...having ability to enable or disable.

imp points /control points of around advice
---
modify
-> In Advice method we can access and target method arguments
-> In advice method we can access and modify method return value.
->In advice method we can control target method execution..

Procedure to develop spring AOP Declarative example Application
==========

step1) develop target class having b.methods/target methods with b.logics(primary logics)

step2) Develop one or more advice /aspect classes having secondary logics..

step3) develop spring bean cfg file..
-->Cfg Target class as spring bean
-->cfg Advice class /classes
->Cfg ProxyFactoryBean class as spring bean giving the following injections
-->taget class obj
--> Advice class/classes objs

This class internally activates spring AOP ,takes given target,advice calsses objs as inputs and generates the InMemory proxy class by mixing up both primary and secondary logics..
and also returns the generated proxy calss obj as resultant object...

less
Factory bean classe are self beans i.e they do not their objects.. they always return one or another resultant object.
give

step4) Develop the Client App for Testing
a) create IOC container (ApplicationContext)
b) get Proxy object by calling container.getBean(-) having ProxyFactoyBean id
c) Invoke b.methods on Proxy object ...

=> Target class :: com.nt.service.BankService
(target methods) |----> float calcSimpleIntrestAmount(float pAmt,float rate,float time)
|----> float calcCompoundIntrestAmount(float pAmt,float rate,float time)
=> Advice :: com.nt.advice.PErformenceMonitoringAdvice

=>spring beancfg file :: com.nt.cfgs.applicationContext.xml

=> client App :: com.nt.test.AroundAdviceTest

AOPProj1-SpringAOPDecl-AroundAdvice
|---->src/main/java
|---->com.nt.service
|-----> BankService.java
|---->com.nt.advice
|-----> PerformenceMonitoringAdvice.java
|---->com.nt.test
|-----> AroundAdvice.java
|---->com.nt.cfgs
|----->pom.xml |--->applicationContext.xml

(add dependencies collected from mvnrepository.com
spring context support
(In Maven/Gradle if u add main jar files.. the relavent dependent jar files will come automatically)

To create maven Project in eclipse
==========

File ---> mavenProject --->next ---> select maven-archetype-quickstart(Project Template for standalone Apps) --> fill details
group Id :: nit (company name)
ArtifactId :: AOPProj1-SpringAOPDecl-AroundAdvice
version :: ... (default value)
package :: com.nt.test (default package name)

-->chanage jre/jdk version:: right click JRE -->properties --->select latest.. (java 9)
->change java compiler version :: right clikc on project --->properties
change java compiler to 9
In Project facects change java version to 9

-> add <depedency> of spring-context-support collected from mvnrepository.com to pom.xml to all main dependent jar files.. including spring-aop-<ver>.jar file..
->create packages in src/main/java as shown above and develop the Applications...

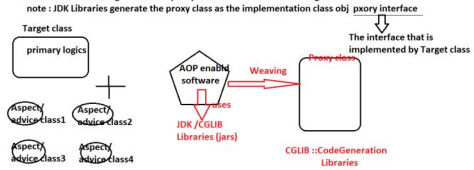
note: We always apply advices (seconary logics) on service class methods (target class b.methods) not on DAO methods... becoz applying on service class methods is equal to applying on DAO class methods indirectly as we know service class methods internally calls DAO class methods..

We should cfg ProxyFactoryBean classes on 1 per each target to get seperate Proxy class for each Target class..but advices can reuses across the multiple ProxyFactoryBean classes cfg..

if u have not taken any pointcuts ..then all advices will be applied on all target methods (public methods) of target class..

The AOP enable software internally use different libraries to generate InMemory Proxy classes like CGLIB libraries (part of spring from 4.x onwods), and Jdk libraries (part of java)

note: CGLIB libraries generate the proxy class as the sub class of target class
note : JDK Libraries generate the proxy class as the implementation class obj pxory interface



=> Spring framework supports Spring AOP, AspectjAOP model AOP style programming
=>Instead of competing with Aspectj AOP , the spring framworks integrrated with Aspectj AOP with spring framework from 2.x version onwords..

In spring we can AOP style app in 8 ways
==========

a) Spring AOP Programatic approach(core java style)
b) Spring AOP Declarative Approach (xml driven)
c) spring AOP in 100% code or Java config Approach
d) Spring AOP in spring BootDriven Approach

In spring AOP there are no annotations...

e) Spring Integrated AspectjAOP Declarative Approach
f) spring integrated AspectjAOP Annotation driven Approach
(g) spring integrated AspectjAOP 100% code driven /Java Config Approach
(h) spring integrated AspectjAOP spring Boot Approach

Different types of Advices
==========

a) Around Advice
==========
==> The apsect logics will execute around the target method (before and after)
usecase:: Performence Monitoring

b) Before Advice
==========
==>==> The apsect logics will execute before entering to target method
usecase :: secuirty

c) After Advice
==========
==> The apsect logics will execute after execution of the target method..
usecase :: generating discount cupon for next purchase based current bill amount

d) Throws advice
==> The apsect logics will execute when exception is raised in the target method.
usecase :: Exception Logger for incident Management..

Around Advice using Spring AOP Declarative Approach

==========

--> This Advice/aspect logics will execute around the target method...\
--> To develop this in spring , Take a class implementing pkg.MethodInterceptor(I)

```
public class PerformenceMonitoringAdvice implements MethodInterceptor{
                                                    makes class as aroung advice class..
  @Override
  public Object invoke(MethodInvocation invocation)throws Exception{
    long start=0,end=0;          Holds the Target method details..
    Object retVal=null;
    start=System.currentTimeMillis(); //pre- logic

    retVal=invocation.proceed();  //calls target method

    end=System.currentTimeMills();  //post logc

    S.o.p(invocation.getMethod().getName()+" has taken "+ (end-start)+" ms  for execution");
  }
        =>spring aop advice /aspect classes are invasive classes becoz they are
        implementing spring api interfaces...
```

usecases:: PErfromence Monitoring , Around Logging , Caching , Transaction Management

=>Entire AOP is designed around Proxy Design Pattern..
(spring AOP/AspectjAOP)
->Which says add new functionalaties dynamically at runtime...having ability to enable or disable.

imp points /control points of around advice

-> In Advice method we can access and modify target method arguments
-> In advice method we can access and modify method return value.
->In advice method we can control target method execution..

Procedure to develop spring AOP Declarative example Application

==========

step1) develop target class having b.methods/target methods with b.logics(primary logics)

step2) Develop one or more advice /aspect classes having secondary logics..

step3) develop spring bean cfg file..

-->Cfg Target class as spring bean
-->cfg Advice class /classes
->Cfg ProxyFactoryBean class as spring bean giving the following injections
-->taget class obj
--> Advice class/classes objs

This class internally activates spring AOP ,takes given target,advice calsses objs as inputs and generates the InMemory proxy class mixing up both primary and secondary logics..
and also returns the generated proxy calss obj as resultant object...

Factory bean classe are self beans i.e they do not their objects.. they always return one or another resultant object.

step4) Develop the Client App for Testing

a) create IOC container (ApplicationContext)
b) get Proxy object by calling container.getBean(-) having ProxyFactoyBean id
c) Invoke b.methods on Proxy object ...

AspectJAOP

Another AOP enabled software/framework .. better than spring AOP

Spring AOP	plain AspectjAOP
a) Here Advice/Aspect classes are tightly coupled with spring aop api i.e these are invasive	a) Here Advice /Apsect classes are loosely coupled with spring api i.e these are non-invasive
b) Does not support Annotations based cfg related to AOP programming	b) Supports
c) Allows only methods as join points	c) Allows Fields,constructors, methods as joinpoints
d) Supports both static and dynamic pointcts	d) Supports only static pointcuts.
e) performs runtime weaving . i.e proxy class is generated dynamically at runtime	e) Performs compile-time weaving and we need AspectjC (aspjectj compiler) to generate proxy class at build time.. and that proxy will be used at runtime
f) Proxy class is InMemory Dyanamic class	f) Here Proxy class Physical class allocating in harddisk
g) Supports only xml configurations with repsect to spring aop operations	g) Supports xml cfgs, annoation cfgs, 100% code cfgs(java config cfgs) , spring boot cfgs

=>Instead of competing with aspectj AOP , the Spring People integrated aspectjAOP with spring from spring 2.x version, Due to this some limitations have come on spring integrated aspectj AOP . they are

(a) Performs Runtime Weaving to generate proxy class.. So there is no need Aspectj compiler (aspectjc)
(b) Supports only methods as joinpoints
(c) Still supports only static pointcuts

note:: In Industry Spring Integrated AspectjAOP bit popular in latest projects ... spring aop was used only in old projets.. which are now in maintence mode..

Different ways of working spring intergrated aspectj AOP

a) Using xml configurations (Declarative approach)
b) Using xml + Annotations configurations
c) Using 100%Code Configurations (JAva Config Cfgs)
d) Using Spring Boot Configurations..

spring integrated AspectjAOP using Declarative Approach (xml driven configuration)

We need aop namespace..

tags are

<aop:aspect>--> To make spring bean aspectj aop advice/aspect class
<aop:config> --> To enable spring AOP configuration on our App
<aop:pointcut > ---> To specify pointcut as OGNL Expression (Object Graph Navigation Language)
<aop:around> --> To cfg aspect class method as around advice method
<aop:before> ---> To cfg aspect class method as before advice method
<aop:after-returning> -->To cfg aspect class method as after returning advice method
<aop:after-throwing> --> To cfg aspect class method as throws advice method.
and etc..

=> Pointcut is OGNL Expression in aspectjAOP and the syntax is

execution(<retrun type> <pkg>.<classname>.<method name>(..)) [and <extra expressions>] (optional)

eg: execution(float com.nt.service.BankService.calcCompoundIntrestAmount(..)) pointcut exp having only calcCompoundIntrestAmount() method)

eg:: execution(* com.nt.service.BankService.*(..)) refers to all methods of BankService class

eg: execution(* com.nt.service.BankService.calc*(..)) refers to those methods of BankService class whose name starts with calc..

BankService
|--->calcSimpleIntrestAmount(-,-,-)
|--->calcCompoundIntrestAmount(-,-,-)
|---> getBalnace(-)
|--->withdraw(-,-)

note:: if we write pointcut OGNL Expression having target class name then it uses CGLIB Libraries to generate the proxy as the sub class of target class.

note:: if we write pointcut OGNL Expression having ProxyInterface name then it use Jdk Libraries to generate the proxy class as the impl class of Proxy Interface..

Around Advice in Spring Integrated AspectjAOP

|->executes around target method
|--> being from advice method we can
-->access and modify target method args
--> access and modify target method return value
-->control target method execution..
|-->usecases :: Performence Monitoring , Around Logging , Caching , Tx Mgmt
=>Checking weight before entering gym and checking once going out from gym...to see calorie burn out
=>submitting belongings when entering into shopping centres and collecting them while coming out..

=>To develop aspectjaop Around advice class we take any class name having having any method but that method must have "ProceedingJoinpoint" as the parameter...

```
package com.nt.aspect;
public class PerformenceMonitoringAspect {

    public  Object  monitor(ProceedingJoinPoint pjp)throws Throwable{
       long start=0 ,end=0;
       Object retVal=null;

       start=System.currentTimeMillis();   //pre logic
       retVal=pjp.proceed();  // calls target method
       end=System.currentTimeMillis();  //post logic
       S.o.p(pjp.getSignature()+" has taken"+ (end-start)+" ms  to complete the execution")
       return  retVal;
       }
    }
```

in applicationContext.xml

```
<beans .....>  //import aop , beans name spaces
   <!--cfg target class-->
     <bean id="bankService"  class="com.nt.service.BankService"/>
   <!--cfg  aspect       class -->
     <bean id="pmAdvice"  class="com.nt.aspect.PerformenceMonitoringAspect*"/>

   <aop:config> <!-- enables and actives spring intergated aspectj aop-->
       <!-- write pointcut expersion-->
         <aop:pointct id="ptc1"  expression="execution( float com.nt.service.BankService.calc*(..))"/>

       <!-- make spring bean aspect aop advice -->
         <aop:aspect  ref="pmAdvice">
           <aop:around  method="monitor"  pointcut-ref="ptc1"/>
         </aop:aspect>
   </aop:config>
</beans>
```

This code makes "pmAdvice" bean id based spring bean class as aspectjAOP advice class by taking "monitor" method around advice method to apply those target class that refered "ptc1" bean id based pointcut expression (they are calc* methods of BankService class)

After Advice flow of execution

ShoppingStore.java (target clas)

```
public class ShoppingStore {
                                                (o)
      public double  shopping(String[] items, double prices[]) {
            double billAmt=0.0f;
            billAmt=DoubleStream.of(prices).sum();
            return billAmt;  (p)
      }
}
```

DiscountCuponAdvice.java

```
public class DiscountCuponAdvice {
                                          (r+s)
      public   void    cuponGenerator(JoinPoint jp,double billAmt)throws Throwable {
            String  cuponMsg=null;
            Writer writer=null;
            System.out.println(jp.getSignature()+" with
args"+Arrays.deepToString(jp.getArgs()));

            if(billAmt<5000)
                  cuponMsg="Avail 5% discount on next purchase";
            else if(billAmt<10000)
                  cuponMsg="Avail 10% discount on next purchase";
            else if(billAmt<20000)
                  cuponMsg="Avail 20% discount on next purchase";
            else
                  cuponMsg="Avail 30% discount on next purchase";

            writer=new FileWriter("E:/cupon.txt");
            writer.write(cuponMsg);
            writer.flush();
            writer.close();
                        (s)
      }
}
```

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 http://www.springframework.org/schema/aop/spring-aop-4.3.xsd">
  <!-- Cfg target class -->
  <bean id="store" class="com.nt.service.ShoppingStore"/> (c)          (c) Preinstantation of
                                                                        singleton scope beans..

  <!-- Cfg advice class.. -->
  <bean id="discountAspect" class="com.nt.aspect.DiscountCuponAdvice"/> (c)

  <!-- Enable aop -->
  <aop:config>  (d)
    <aop:aspect ref="discountAspect">
      <aop:after-returning method="cuponGenerator"
          pointcut="execution(double com.nt.service.ShoppingStore.shopping(..))"
          returning="billAmt"                    (e)
          />
    </aop:aspect>

  </aop:config>

</beans>
```

Inmemory Proxy class

```
                                    (f)
public class ShoppingStore$Proxy extends   ShoppingStore  impelements  ApplicationContextAware{
  private ApplicationContext ctx;

  public void setApplicationContext(ApplicationContext ctx){
      this.ctx=ctx;          (g)
  }
                  (m)
  public  float  shopping(String[] items , float [] prices){

   //get target class object
   ShoppingStore target=ctx.getBean("store",ShoppingStore.class);
   // Get Advice class  object
   DiscountCuponAspect  advice=ctx.getBean("cuponAspect",DiscountCuponAspect.class);
   //get Target  Method method
   Method method=target.getDeclaredMethod("shopping");
   // prepare Object[] having arg values
   Object args[]=new Object[]{items,prices};
   //create JoinPoint Object
   JoinPoint jp=new MethodInvoationJoinPoint();
   jp.setSignature(method);
   jp.setTarget (target);
   jp.setArgs(args);
   // invoke targt method          (n)
(q) float billAmt= target.shopping(items, prices);
   //invoke advice method          (r)
   advice.cuponGenerator (jp,billAmt);
   return billAmt; (t)
  }
}
```

Internal Cache of IOC container (h)

discountAspect	DiscountCuponAspect obj ref
store	ShoppingStore$Proxy obj (s?)

ClientApp

```
public class AfterAdviceTest {
      public static void main(String[] args) {
            ApplicationContext ctx=null;
            ShoppingStore proxy=null;
            //create IOC container      (a)          (b) In Memory Metdata of xml file
            ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
            //get Proxy  object
(k)         proxy=ctx.getBean("store",ShoppingStore.class); (i)
            //invoke method
            try {       (u)          (i)
              System.out.println("Bill Amount::"+proxy.shopping(new String[] {"shirt","trouser","table"},
                                                      new double[] {300,400,600}));

            }
            catch(Exception e) {
                  e.printStackTrace();
            }

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

After Advice /After returning advice

=> This advice executes after completion of target method before it returns the return value to the caller.

=> To develop this advice in aspectjaop , take a class having advice method.. with following signature...

```
public class <Classname>{

   public  void  <method>(JoinPoint jp, <param to hold the return value of target method>){
       ...
       ...
       ...
    }
}
      (or)

public class <Classname>{

   public  void  <method>( <param to hold the return value of target method>){
      ...
      ...
      ...
   }          <aop:after-returning  method="...."  pointcut-ref="..."  returning=" <param name>"/>
}
```

usecases :: Generating discount cupon for next purchase based on current billAmout
Generating reward points based current billAmount
Giving Gpay/PayTM cash back based on the curent transacion..
Giving lucky draw cupon based on current bill amount..

import control points

(a) Being from advice method we can access target arguments ,, but we can not modify
(b) We can access target method return value.. but we can not modify.
(c) we can not control target method execution being from advice method becoz the control comes to advice method after executing target method.. we can stop target method return value going to client by throwing exception in advice method..

```
AOPProj9-AspectjAOP-AfterAdvice-DiscountCupon
   |-->src/main/java
         |---->com.nt.service
                  |------>ShoppingStore.java
         |---->com.nt.aspect
                  |---->DiscountCuponAdvice.java
         |------>com.nt.cfgs
                  |--->applicationContext.xml
         |------>com.nt.test
                  |------>AfterAdviceTest.java

   |-->pom.xml
```

CaseStudy1 :: if Project is released with out enabling logging... Later we can eanble logging by developing just one Login Advice and by limking that advice class with multiple service classes and their methods using spring based AOP configurations..

=>Logging is all about keeping track of code Flow..
Lommitations of S.o.p(-) for Logging
==========
(a) S.o.p does not allow to write log messages to diff destinations..
It allows us to write only console monitor which will be disappered after some time
(b) S.o.p Does not allow to catorize the log messages..
(c) S.o.p does not allow to filter the log messages..
(d) S.o.p does not allow to format the messages..
(e) S.o.p messages are written as single Threaded process...

To overcome these problem use.. other logging tools .. log4j , logback , commons logging and etc...

SLF4J is build on the top multiple logging apis and tools ... instead using different apis to work with diff logging tools .. using single api of slf4j to work diff logging tools..

3 imp objects of log4j
==========
log4j-<ver>.jar file...

(a) Logger object
--> To enable logging on certain class and to write log messages having diff caterories and priorities like
DEBUG<INFO<WARN<ERROR<FATAL

DEBUG --> To indicate flow of execution statements
INFO ---> For important Opertion confirmation.
WARN ----> When Poor or deprecated are used..
ERROR ----> For known exceptions
FATAL ---> for known exceptions..

This -> object is also to filter the log messages while rendering/retrieving..
if we take one logger level like ERROR for retrieving log messages then it write only ERROR ,FATAL message to log file...

ALL<DEBUG<INFO<WARN<ERROR<FATAL<OFF

ALL --> Writes all caterory messages to log file
OFF -->Disables logging..

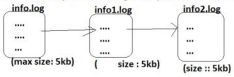
```
Logger logger =Logger.getLogger(BankService.class);

logger.debug("entered into calcSimpleIntrest method");
logger.info("....");
logger.warn(".....");
logger.error("....");
logger.fatal(".....");
```

b) Appender object
==========
=>This object is useful to specify the destination where the log messages can be recorded.. like console, file, db s/w , mail server and etc...

ConsoleAppender , FileAppender , JdbcAppender , RollingFileAppender , DailyRollingFileAppender , SMTPAppender and etc..

RollingFileAppender

=> Creates backup file .. once log file's specified max size is reached..



DailyRollingFileAppender

--> creates the backup file on daily basis, weekly basis , montly basis, hourly basisc , menutely basis , for every 12 hours basis and etc...

Tomcat server generates .. log files based on DailyRollingFileAppender..

c) Layout object
==========
=> Allows to format the log messages... specifies what should be there and what should not be there in the log messages...

eg:: SimpleLayout , HtmlLayout , XmlLayout, PatternLayout and etc...

=>we can give log4j configurations either through properties file (best) or through xml file
In the properties file the keys are pre-defined but value can be given according our comfort/ requirement..

To enable Log4 based logging in our spring AOP Application
==========
step1) Add log4j dependency in pom.xml

```
<!--
https://mvnrepository.com/artifact/log4j/log4j -->
<dependency>
    <groupId>log4j</groupId>
    <artifactId>log4j</artifactId>
    <version>1.2.17</version>
</dependency>
```

step2) Write log4 based logging code .. in LogginAdvice class...

```
package com.nt.aspect;
import java.util.Arrays;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import org.aspectj.lang.ProceedingJoinPoint;
public class AroundLoggingAspect {
    private static Logger logger=Logger.getLogger(AroundLoggingAspect.class);
    static {
        try {
         PropertyConfigurator.configure("src/main/java/com/nt/commons/log4j.properties");
         }
         catch(Exception e) {
               e.printStackTrace();
         }
    }
         public Object logging(ProceedingJoinPoint pjp)throws Throwable{
         Object retVale=0;
         logger.debug("Entering into "+pjp.getSignature()+" with "+Arrays.deepToString(pjp.getArgs()));
         retVal=pjp.proceed();
         logger.debug("Exiting from "+pjp.getSignature()+" with "+Arrays.deepToString(pjp.getArgs()));
         return retVal;
    }
}
```

step3 ::Develope log4.proeprties file

log4j.properties
==========

```
log4j.rootLogger=ALL,R
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
log4j.appender.R.File=info.html
log4j.appender.R.DatePattern='.'yyyy-MM-dd-HH-mm
log4j.appender.R.layout=org.apache.log4j.HTMLLayout
```

In <aop:aspect> tag we can use "order" to specify the order applying aspects when multiple aspects are configured on the same target method.. here

High value indicates low priority
low value indicates high priority

In aspectj aop around advice . after modifying target method arg values in advice method.. we need to call pjp.proceed(-) having those args as the argument value..

```
Object[] args=pjp.getArgs();
if(pjp.getSignature().getName().equalsIgnoreCase("calcBillAmount")) {
    args[0]=(Float)args[0]+1;
}
retVal=pjp.proceed(args);
```

=> if use target class name in Pointcut expressions then the aspectj AOP internally uses CGLIB library to generate the proxy class...as the sub class for target class .. So we can not take target class as final class here..
=> if use proxy interface name in Pointcut expressions then the aspectj AOP internally uses JDK library to generate the proxy class...as the implementation class of Proxy interface .. So we can take target class as final class here..
here..

if our target class is implementing proxy interface .. then we can instruct aspectj aop to use either CGLIB or JDK Libraries to generate proxy class irreseptive of wheater target class or proxy interface name is used in Pointcut expression or not...

<aop:config proxy-target-class="true"/> generate
uses CGLIB Libraries to proxy class as the sub class of target class

<aop:config proxy-target-class="false"/> (default is false)
uses JDK Libraries to generate proxy class as the impl class proxy interface..

CaseStudy1 :: if Project is released with out enabling logging... Later we can eanble logging by developing just one Login Advice and by limking that advice class with multiple service classes and their methods using spring based AOP configurations..

=>Logging is all about keeping track of code Flow..
Lommitations of S.o.p(-) for Logging
==========
(a) S.o.p does not allow to write log messages to diff destinations..
It allows us to write only console monitor which will be disappered after some time
(b) S.o.p Does not allow to catorize the log messages..
(c) S.o.p does not allow to filter the log messages..
(d) S.o.p does not allow to format the messages..
(e) S.o.p messages are written as single Threaded process...

To overcome these problems use.. other logging tools .. log4j , logback , commons logging and etc...

SLF4J is build on the top multiple logging apis and tools ... instead using different apis to work with diff logging tools .. using single api of slf4j to work diff logging tools..

3 imp objects of log4j
==========
log4j-<ver>.jar file...

(a) Logger object
--> To enable logging on certain class and to write log messages having diff caterories and priorities like
DEBUG<INFO<WARN<ERROR<FATAL

DEBUG --> To indicate flow of execution statements
INFO ---> For important Opertion confirmation.
WARN ----> When Poor or deprecated are used..
ERROR ----> For known exceptions
FATAL ---> for known exceptions..

This
-> object is also to filter the log messages while rendering/retrieving..
if we take one logger level like ERROR for retrieving log messages then it write only ERROR ,FATAL message to log file...

ALL<DEBUG<INFO<WARN<ERROR<FATAL<OFF

ALL --> Writes all caterory messages to log file
OFF -->Disables logging..

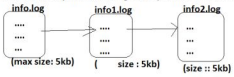
```
Logger logger =Logger.getLogger(BankService.class);

logger.debug("entered into calcSimpleIntrest method");
logger.info("...");
logger.warn(".....");
logger.error("....");
logger.fatal("....");
```

b) Appender object
==========
=>This object is useful to specify the destination where the log messages can be recorded.. like console, file, db s/w , mail server and etc...

ConsoleAppender , FileAppender , JdbcAppender , RollingFileAppender , DailyRollingFileAppender , SMTPAppender and etc..

RollingFileAppender

=> Creates backup file .. once log file's specified max size is reached..



DailyRollingFileAppender

--> creates the backup file on daily basis, weekly basis , montly basis, hourly basisc , menutely basis , for every 12 hours basis and etc...

Tomcat server generates .. log files based on DailyRollingFileAppender..

c) Layout object
==========
=> Allows to format the log messages... specifies what should be there and what should not be there in the log messages...

eg:: SimpleLayout , HtmlLayout , XmlLayout, PatternLayout and etc...

=>we can give log4j configurations either through properties file (best) or through xml file
In the properties file the keys are pre-defined but value can be given according our comfort/ requirement..

To enable Log4 based logging in our spring AOP Application
==========

step1) Add log4j dependency in pom.xml

```
<!--
https://mvnrepository.com/artifact/log4j/log4j -->
<dependency>
<groupId>log4j</groupId>
<artifactId>log4j</artifactId>
<version>1.2.17</version>
</dependency>
```

step2) Write log4 based logging code .. in LogginAdvice class...

```
package com.nt.aspect;
import java.util.Arrays;
import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;
import org.aspectj.lang.ProceedingJoinPoint;
public class AroundLoggingAspect {
private static Logger logger=Logger.getLogger(AroundLoggingAspect.class);
static {
try {
PropertyConfigurator.configure("src/main/java/com/nt/commons/log4j.properties");
}
catch(Exception e) {
e.printStackTrace();
}
}
public Object logging(ProceedingJoinPoint pjp)throws Throwable{
Object retVale=0;
logger.debug("Entering into "+pjp.getSignature()+" with "+Arrays.deepToString(pjp.getArgs()));
retVale=pjp.proceed();
logger.debug("Exiting from "+pjp.getSignature()+" with "+Arrays.deepToString(pjp.getArgs()));
return retVal;
}
}
```

step3 ::Develope log4.proeprties file

log4j.properties
==========

```
log4j.rootLogger=ALL,R
log4j.appender.R=org.apache.log4j.DailyRollingFileAppender
log4j.appender.R.File=info.html
log4j.appender.R.DatePattern='.'yyyy-MM-dd-HH-mm
log4j.appender.R.layout=org.apache.log4j.HTMLLayout
```

In <aop:aspect> tag we can use "order" to specify the order applying aspects when multiple aspects are configured on the same target method.. here

High value indicates low priority
low value indicates high priority

CommonExceptionGrapher

==========

=>It is capable of translating /graphing Target class/service class generated/gathered Exceptions into Project specific user-defined Exceptions...

```
public class CommonExceptionGrapherAspect {

    public void exceptionGraphing(JoinPoint jp,IllegalAccessException iae) {
        throw new InternalProblemException(iae.getMessage());
    }
}
```

If target class is throwing CheckedException and it is converted to project specified User-defined Exception then the ClientApp should catch handle both ogical CheckedException and project specific user-defined exception...to satisfy compiler in all angels.

If target class is throwing UbCheckedException and it is converted to project specified User-defined Exception then the ClientApp should catch handle only project specific user-defined exception...to satisfy compiler in all angels.

Summary table

==========

Advice type	when it executes	accessing and modifying taget method args	accessing and modifying target method return value	controlling target method excution	usecases
around advice	around the target method	yes	yes	yes	caching, Performance Monitoring, Around Logging , Transaction Mgmt
Before Advice	Before entering into target method	accessing is possible modification is not possible	NA	no (By throwing exception in advice method we can stop control going to target method)	Auditing ,Security Check ,CoronaTest before entering into hospital
After Advice	After executing target method	Accessing is possible ,but modification is not possible	Accessing is possible but modification is not possible	No (But by throwing exception we can stop return value being returned to caller)	discount coupon, Gpay-cashback , lucky draw coupon
ThrowsAdvice	if exception ir raised in target method	same as above	NA (not applicable)	NO	Common Exception Logger , Common Exception Grapher

Annotation driven AspectjAOP (@Aspectj Approach)

note:: There is no annotation called @Aspectj ... This whole Approach is called @Aspectj Approach

@Aspect :: To make Sprign Bean as Aspectj AOP Aspect class like <aop:aspect> tag.. (but this should be made as spring bean using @Component annotation)

@Pointcut :: To write Pointcut expression having resuability like <aop:pointcut>

@Around :: To make java method as around advice method like <aop:around>

@Before :: To make java method as before advice method like <aop:before>

@AfterReturning :: To make java method as after advice method like <aop:after-returning>

@AfterThrowing :: To make java method as throws advice method like <aop:after-throwing>

Two xml tags in Spring bean cfgs file to work with above annotations...
<context:component-scan> -->To scan and recognize classes as Spring beans that are having annotaions
<aop:aspectj-autoproxy/> --> It is like <aop:config>

Thum rule while working annotation driven spring programming
->configure pre-defiend classes as spring beans using <bean> tags
->configure user-defined classes as spring beans using stereo type annotations
@Component --> makes java class as spring bean (creates java class object)
@Service ---> makes java class as spring bean + allows b.logics + Transaction magement
@Repository -->makes java class as sprign bean + allows persistence logics + supports exception translation and etc.. @Controller ,@RestController , @Configuration, @ControllerAdvice

note:: We can use @Order(<value>) to specify priority order for advice classes.. here high value indicates low priority and low value indicates high priority..
if @Order is not used, the Priority will be decided based on Alphabetic order of advice/Aspect class name..

note: if can @Order either on the top advice/aspect classes or on the top of advice methods.. (place at same level either at classes level or at advice methods level

if we are placing multiple advice methods in a single advice class/aspect class having same pointcut expression then instead of writing same pointcut expression on multiple advice methods , we can write only for 1 time on the top dummy method using @Poincut.. and we can reuse that pointcut expression on multiple advice methods by specifying dummy method call as the pointcut id.

```
@Component("allAspects")
@Aspect
public class AllAroundAspects{

  @Pointcut("execution(* com.nt.service.ECommerceStore.*(..))")
  public void myPtc(){
  }

  @Around(value = "myPtc()")
  @Order(5)
  public Object logging(ProceedingJoinPoint pjp)throws Throwable{
    ...
    ...
  }

  @Around(value = "myPtc()")
  @Order(10)
  public   Object monitor(ProceedingJoinPoint pjp)throws Throwable{
    ...
    ...
  }

  @Around("myPtc()")
  @Order(2)
  public  Object caching (ProceedingJoinPoint pjp)throws Throwable{
    ...
    ...
    ...
  }

}
```

CommonExceptionGrapher

==========

=>It is capable of translating /graphing Target class/service class generated/gathered Exceptions into Project specific user-defined Exceptions...

```
public class CommonExceptionGrapherAspect {

    public void exceptionGraphing(JoinPoint jp,IllegalAccessException iae) {
        throw new InternalProblemException(iae.getMessage());
    }
}
```

If target class is throwing CheckedException and it is converted to project specified User-defined Exception then the ClientApp should catch handle both ogical CheckedException and project specific user-defined exception...to satisfy compiler in all angels.

If target class is throwing UbCheckedException and it is converted to project specified User-defined Exception then the ClientApp should catch handle only project specific user-defined exception...to satisfy compiler in all angels.

Summary table

==========

Advice type	when it executes	accessing and modifying taget method args	accessing and modifying target method return value	controlling target method excution	usecases
around advice	around the target method	yes	yes	yes	caching, Performance Monitoring, Around Logging , Transaction Mgmt
Before Advice	Before entering into target method	accessing is possible modification is not possible	NA	no (By throwing exception in advice method we can stop control going to target method)	Auditing ,Security Check ,CoronaTest before entering into hospital
After Advice	After executing target method	Accessing is possible ,but modification is not possible	Accessing is possible but modification is not possible	No (But by throwing exception we can stop return value being returned to caller)	discount coupon, Gpay-cashback , lucky draw coupon
ThrowsAdvice	if exception ir raised in target method	same as above	NA (not applicable)	NO	Common Exception Logger , Common Exception Grapher

Annotation driven AspectjAOP (@Aspectj Approach)

note:: There is no annotation called @Aspectj ... This whole Approach is called @Aspectj Approach

@Aspect :: To make Sprign Bean as Aspectj AOP Aspect class like <aop:aspect> tag.. (but this should be made as spring bean using @Component annotation)

@Pointcut :: To write Pointcut expression having resuability like <aop:pointcut>

@Around :: To make java method as around advice method like <aop:around>

@Before :: To make java method as before advice method like <aop:before>

@AfterReturning :: To make java method as after advice method like <aop:after-returning>

@AfterThrowing :: To make java method as throws advice method like <aop:after-throwing>

Two xml tags in Spring bean cfgs file to work with above annotations...
<context:component-scan> -->To scan and recognize classes as Spring beans that are having annotaions
<aop:aspectj-autoproxy/> --> It is like <aop:config>

Thum rule while working annotation driven spring programming
->configure pre-defiend classes as spring beans using <bean> tags
->configure user-defined classes as spring beans using stereo type annotations
@Component --> makes java class as spring bean (creates java class object)
@Service ---> makes java class as spring bean + allows b.logics + Transaction magement
@Repository -->makes java class as sprign bean + allows persistence logics + supports exception translation and etc.. @Controller ,@RestController , @Configuration, @ControllerAdvice

note:: We can use @Order(<value>) to specify priority order for advice classes.. here high value indicates low priority and low value indicates high priority..
if @Order is not used, the Priority will be decided based on Alphabetic order of advice/Aspect class name..

note: if can @Order either on the top advice/aspect classes or on the top of advice methods.. (place at same level either at classes level or at advice methods level

if we are placing multiple advice methods in a single advice class/aspect class having same pointcut expression then instead of writing same pointcut expression on multiple advice methods , we can write only for 1 time on the top dummy method using @Poincut.. and we can reuse that pointcut expression on multiple advice methods by specifying dummy method call as the pointcut id.

```
@Component("allAspects")
@Aspect
public class AllAroundAspects{

  @Pointcut("execution(* com.nt.service.ECommerceStore.*(..))")
  public void myPtc(){
  }

  @Around(value = "myPtc()")
  @Order(5)
  public Object logging(ProceedingJoinPoint pjp)throws Throwable{
    ...
    ...
  }

  @Around(value = "myPtc()")
  @Order(10)
  public   Object monitor(ProceedingJoinPoint pjp)throws Throwable{
    ...
    ...
  }

  @Around("myPtc()")
  @Order(2)
  public  Object caching (ProceedingJoinPoint pjp)throws Throwable{
    ...
    ...
    ...
  }

}
```

CommonExceptionGrapher

==========

=>It is capable of translating /graphing Target class/service class generated/gathered Exceptions into Project specific user-defined Exceptions...

```
public class CommonExceptionGrapherAspect {

    public void exceptionGraphing(JoinPoint jp,IllegalAccessException ex) {
        throw new InternalProblemException(ex.getMessage());
    }

}
```

If target class is throwing CheckedException and it is converted to project specified User-defined Exception then the ClientApp should catch handle both ogical CheckedException and project specific user-defined exception...to satisfy compiler in all angels.

If target class is throwing UbCheckedException and it is converted to project specified User-defined Exception then the ClientApp should catch handle only project specific user-defined exception...to satisfy compiler in all angels.

Summary table

==========

Advice type	when it executes	accessing and modifying taget method args	accessing and modifying target method return value	controlling target method excution	usecases
around advice	around the target method	yes	yes	yes	caching, Performence Monitoring, Around Logging , Transaction Mgmt
Before Advice	Before entering into target method	accessing is possible modification is not possible	NA	no (By throwing exception in advice method we can stop control going to target method)	Auditing ,Security Check ,CoronaTest before entering into hospital
After Advice	After executing target method	Accessing is possible ,but modification is not possible	Accessing is possible but modification is not possible	No (But by throwing exception we can stop return value being returned to caller)	discount coupon, Gpay-cashback , lucky draw coupon
ThrowsAdvice	if exception ir raised in target method	same as above	NA (not applicable)	NO	Common Exception Logger , Common Exception Grapher

Annotation driven AspectjAOP (@Aspectj Approach)

note:: There is no annotation called @Aspectj ... This whole Approach is called @Aspectj Approach

@Aspect :: To make Sprign Bean as Aspectj AOP Aspect class like <aop:aspect> tag..
(but this should be made as spring bean using @Component annotation)

@Pointcut :: To write Pointcut expression having resuability like <aop:pointcut>

@Around :: To make java method as around advice method like <aop:around>

@Before :: To make java method as before advice method like <aop:before>

@AfterReturning :: To make java method as after advice method like <aop:after-returning>

@AfterThrowing :: To make java method as throws advice method like <aop:after-throwing>

Two xml tags in Spring bean cfgs file to work with above annotations...
<context:component-scan> -->To scan and recognize classes as Spring beans that are having annotaions
<aop:aspectj-autoproxy/> --> It is like <aop:config>

Thum rule while working annotation driven spring programming

==========

->configure pre-definrd classes as spring beans using <bean> tags
->configure user-defined classes as spring beans using stereo type annotations
@Component --> makes java class as spring bean (creates java class object)
@Service ---> makes java class as spring bean + allows b.logics + Transaction magement
@Repository -->makes java class as sprign bean + allows persistence logics + supports exception translation and etc.. @Controller

Spring Boot driven Aspectj AOP Appliation development (extension of 100%Code Approach)
==========

Thumb rule to develop spring Boot App
==========

==>Cfg user-defined classes using streo type anntations
(these will be linked with configuration class automtically becoz of
@ComponentScan Annotation that is there in @SpringbootApplication annotation)

==> cfg pre-defined classes as spring beans using @Bean methods only when they are not comming through AutoConfiguration
(Certain pre-defined java classes will come spring beans automatically based on th jar files that we have added to built path/classpath , for this we give inputs from application.properties) yml

if add spring-boot-starter-jdbc-<ver>.jar file to SpringBoot project we will get the following beans througn autoConfiguration..
-> HikariDataSource object pointting to jdbc con pool (based application .properties file supplied jdbc properties the con objs in jdbc con pool will be created (DB s/w is decided))
-> JdbcTemplate class obj injected with DataSource object
-> DataSourceTransactionManager injected with DataSoruce object

=>Write Client App code.. main class where @SpringbootApplication annotation is placed..by accessing the internally created container from SpringApplication.run(...) method.

Procedure to create Spring Boot Project
==========

a) make sure that STS plugin is installed in eclipse IDE

b) create Spring boot Starter Project.. giving following details..

name:: AopProj16-AspectjAOPBoot -BeforeAdvice-SecurityCheck
choolse gradle , choose java version:14 , language :: java , packing :jar
grouid Id: nit.com
artifactId:: AopProj16-AspectjAOPBoot -BeforeAdvice-SecurityCheck
version:: ....
package name:: com.nt
--> next ---> choose starter
a) jdbc api
b) oracle driver
-->next ---> ... ->finish

c) add aspectj jar files in build.gradle by collecting info mvnrepository.com

```
// https://mvnrepository.com/artifact/org.aspectj/aspectjrt
implementation group: 'org.aspectj', name: 'aspectjrt', version: '1.9.6'
// https://mvnrepository.com/artifact/org.aspectj/aspectjweaver
implementation group: 'org.aspectj', name: 'aspectjweaver', version: '1.9.6'
```

d) Add the following entries in application.properties file.. to give inputs to HikartDataSoruce class towards creating jdbc con obejcts for oracle Dbs/w..

application.properties

```
# connection properties
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
```

e) develop the Application reosurces in reqular mannner.. But do not take any Configuration classes... (makesure that all packages are created as the subpackges of "com.nt" pgk.. which is the pkg of main/starter/root app class..

d) get the IOC Container from the main(-) method spring boot main Applicaiton class.. write Client App code...

e) Run the Client App using Run As---> Java App /Spring boot App..

Spring Boot driven Aspectj AOP Appliation development (extension of 100%Code Approach)
==========

Thumb rule to develop spring Boot App
==========

==>Cfg user-defined classes using streo type anntations
(these will be linked with configuration class automtically becoz of
@ComponentScan Annotation that is there in @SpringbootApplication annotation)

==> cfg pre-defined classes as spring beans using @Bean methods only when they are not comming through AutoConfiguration
(Certain pre-defined java classes will come spring beans automatically based on th jar files that we have added to built path/classpath , for this we give inputs from application.properties) yml

if add spring-boot-starter-jdbc-<ver>.jar file to SpringBoot project we will get the following beans througn autoConfiguration..
-> HikariDataSource object pointting to jdbc con pool (based application .properties file supplied jdbc properties the con objs in jdbc con pool will be created (DB s/w is decided))
-> JdbcTemplate class obj injected with DataSource object
-> DataSourceTransactionManager injected with DataSoruce object

=>Write Client App code.. main class where @SpringbootApplication annotation is placed..by accessing the internally created container from SpringApplication.run(...) method.

Procedure to create Spring Boot Project
==========

a) make sure that STS plugin is installed in eclipse IDE

b) create Spring boot Starter Project.. giving following details..

name:: AopProj16-AspectjAOPBoot -BeforeAdvice-SecurityCheck
choolse gradle , choose java version:14 , language :: java , packing :jar
grouid Id: nit.com
artifactId:: AopProj16-AspectjAOPBoot -BeforeAdvice-SecurityCheck
version:: ....
package name:: com.nt
--> next ---> choose starter
a) jdbc api
b) oracle driver
-->next ---> ... ->finish

c) add aspectj jar files in build.gradle by collecting info mvnrepository.com

```
// https://mvnrepository.com/artifact/org.aspectj/aspectjrt
implementation group: 'org.aspectj', name: 'aspectjrt', version: '1.9.6'
// https://mvnrepository.com/artifact/org.aspectj/aspectjweaver
implementation group: 'org.aspectj', name: 'aspectjweaver', version: '1.9.6'
```

d) Add the following entries in application.properties file.. to give inputs to HikartDataSoruce class towards creating jdbc con obejcts for oracle Dbs/w..

application.properties

```
# connection properties
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
```

e) develop the Application reosurces in reqular mannner.. But do not take any Configuration classes... (makesure that all packages are created as the subpackges of "com.nt" pgk.. which is the pkg of main/starter/root app class..

d) get the IOC Container from the main(-) method spring boot main Applicaiton class.. write Client App code...

e) Run the Client App using Run As---> Java App /Spring boot App..

AspectjAOP Before Advice Flow of execution

CarShowRoom.java

```
package com.nt.service;
public class CarShowRoom {                                    (c)
    public String purchaseCar(String modelname,float price,String color,String executive ) {

        if(price<1000000 && modelname.equalsIgnoreCase("baleno"))
            return modelname+" car having price"+price +" with color"+color+" sold to customer by"+executive;
        else                                                  (x)
            return modelname+" car having price"+price +" with color"+color+" not available to sell to customer by"+executive;
    }//method
}//class
```

TestDriveAdvice.java

```
public class TestDriveAdvice {                                (a)
    public  void  testDriving(JoinPoint jp)throws Throwable{
        Object args[]=null;
        Writer writer=null;
        //get target method arguments
        args=jp.getArgs();
        try {
           writer=new FileWriter("E:/auditLog.txt",true);
           writer.write(args[0]+" model car purchase having price"+args[1]+ "with color:"+args[2]+" through excecutive ::"+args[3]+" has come for test driving at ::"+new
Date()+"\n");
           writer.flush();
           writer.close();
          }
          catch(Exception e) {
              e.printStackTrace();
          }
    }//method
}
```

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">
<!-- cfg taget class -->
 <bean id="showRoom"  class="com.nt.service.CarShowRoom"/>  (e)

 <!-- Cfg advice class -->
 <bean id="drivеAdvice"  class="com.nt.advice.TestDriveAdvice"/> (e)

 <!-- enable aspectjAOP -->
 <aop:config >        (f)     (notices that <aop:config> is enabled)
    <!-- Pointcut expression -->                                  (f)
   <aop:pointcut expression="execution(String com.nt.service.CarShowRoom.purchaseCar(..))" id="ptc"/>

   <!-- Cfg  spring bean as aspectjAOP advice -->
   <aop:aspect ref="driveAdvice">
     <aop:before method="testDriving" pointcut-ref="ptc"/>
   </aop:aspect>
 </aop:config>
</beans>
```

Dynamically getnerted in Memory proxy class using AspectjAOP and CGLIB Libraries

```
                                                              (g)
public   class  CarShowRoom$Proxy extends   CarShowRoom  implements  ApplicationContextAware {
    private  ApplicationContext ctx;
     public  void setApplicationContext(ApplicationContext ctx){
        this.ctx=ctx;                (h)  (unerlying  IOC container object injection)
     }
                                  (n)
     public String  purchaseCar (String modelName, float price  , String colour , String executive){
       //get Target  class object
       CarShowRoom target=ctx.getBean("showRoom",CarShowRoom.class);
       //get Advice class object
       TestDrivingAdvice advice=ctx.getBean("drivingAdvice",TestDrivingAdvice.class);
       //get Target  method details
       Method method=target.getClass().getDeclaredMethod("purchaseCar");
       //prepare  JoinPoin object
       JoinPoint jp=new MethodInvocationPrcoeedingJoinPoint();
       jp.setSignature(method);
       jp.setTarget(target);
       jp.setArgs(new Object[]{modelName,price,colour,executive});

      //invoke advice  method
       advice.testingDriving(jp);  (o)
      //invoke target method        (p)
(t)   String result=target.purchaseCar(modelName,price, colour, executive);
      return result;  (u)
     }
}
```

Internal Cache of IOC container

showRoom	CarshowRoom$Proxy obje(t ref
drivingAdvice	TestDrivingAdvice obj ref
	(k?)

(i)

Client App

```
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;
                                                    (e) Pre-instantiation of singleton scope beans
public class AspectjAOPBeforeAviceTest {
      public static void main(String[] args) {  (a)
            ApplicationContext ctx=null;                    (d) preparering InMemory MetaData
            CarShowRoom proxy=null;                             of  applicationContext.xml file
            //create IOC container    (b)                (c)  checking wellformed ,valid or not
            ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
            //get Proxy class object
(l)         proxy=ctx.getBean("showRoom",CarShowRoom.class);  (j)
            //invoke the b.method
            try {                 (m)
                  System.out.println(proxy.purchaseCar("swift",900000, "red","karan"));
            }
            catch(Exception e) {
                  e.printStackTrace();
            }

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

      }//main
}//class
```

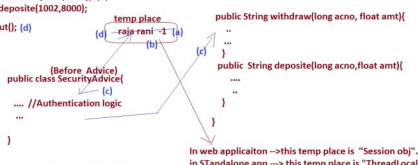
Before Advice --Security Check use-Case

client App 1

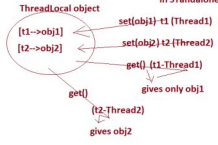
```
signIn("raja","rani");  (a)
proxy.withdraw(1001,7000);  (b)
proxy.deposite(1002,8000);
signout(); (d)
```

taget class

```
public class  BankSErvice{

  public String withdraw(long acno, float amt){
      ..
      ...
  }
  public  String deposite(long acno,float amt){
      ....
      ..
  }
}
```



In web applicaiton -->this temp place is "Session obj".
in STandalone app ---> this temp place is "ThreadLocal".



ThreadLocal allows to keep threads their object.. but 1 thread at a time can keep only object.. More over every thread can get only its data.. i.e can not get other threads data.. this makes the data as thread safe data..

ThreadLocal object is internally HashMap having thread name as the key and object as the value..

Hash Map

t1	obj1
t2	obj2
t3	obj3
keys	values..

ThreadLocal is introduced in java 1.4 and generics support is added in java 6 onwords..

```
ThreadLocal<Integer> local=new ThreadLocal();
   form Thread1 (t1)
   ----------------------        data specific to thread
    local.set(10);    //to keep in ThreadLocal
                  Integer Wrapper class object

    local.set(20);  // to modify Thread data in ThreadLocal

    Integer i1=local.get() ;  // to read thread data from ThreadLocal

    local.remove() ; // to remove  thread data from ThreadLocal
```

Scopes of data in standalone App

```
block ---> specific to a block
Local -->specific to a method
instance --> specific to a object
class  ---> specific to a class
thread --> specific to a thread..
```

Scopes of data in web application

```
request scope -->specific to each request
session scope -->specific to each browser s/w
                    of a client machine
application scope ---> specific to each  web applicaiton
page scope --> speciifc to each servlet/jsp comp
```

The data keep cache is not specific one client or thread.. it is common for all the threads.. i.e the data kept in cache by one thread can be used by other threads.... where as ThreadLocal data is specific to each thread.

AspectjAOP Before Advice Flow of execution

CarShowRoom.java

```
package com.nt.service;
public class CarShowRoom {                                  (c)
      public String purchaseCar(String modelname,float price,String color,String executive ) {

            if(price<1000000 && modelname.equalsIgnoreCase("baleno"))
                  return modelname+" car having price"+price +" with color"+color+" sold to customer by"+executive;
            else  (c)
                  return modelname+" car having price"+price +" with color"+color+" not available to sell to customer by"+executive;
      }//method
}//class
```

TestDriveAdvice.java

```
public class TestDriveAdvice {  (a)
      public  void  testDriving(JoinPoint jp)throws Throwable{
            Object args[]=null;
            Writer writer=null;
            //get target method arguments
            args=jp.getArgs();
            try {
                writer=new FileWriter("E:/auditLog.txt",true);
                writer.write(args[0]+" model car purchase having price"+args[1]+ "with color:"+args[2]+" through excecutive ::"+args[3]+" has come for test driving at ::"+new
Date()+"\n");
                writer.flush();
                writer.close();
            }
            catch(Exception e) {
                e.printStackTrace();
            }
      }//method
}
```

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">
<!-- cfg target class -->
 <bean id="showRoom"  class="com.nt.service.CarShowRoom"/>  (e)

  <!-- Cfg advice class -->
  <bean id="driveAdvice" class="com.nt.advice.TestDriveAdvice"/> (e)

  <!-- enable aspectjAOP -->
 <aop:config >      (f)      (notices that <aop:config> is enabled)
    <!-- Pointcut expression -->                                  (g)
    <aop:pointcut expression="execution(String com.nt.service.CarShowRoom.purchaseCar(..))" id="ptc"/>

    <!-- Cfg  spring bean as aspectjAOP advice -->
    <aop:aspect ref="driveAdvice">
      <aop:before method="testDriving" pointcut-ref="ptc"/>
    </aop:aspect>
  </aop:config>
</beans>
```

Dynamically getnerted In Memory proxy class using AspectjAOP and CGLIB Libraries

```
                                    (g)
public   class   CarShowRoom$Proxy extends    CarShowRoom   implements   ApplicationContextAware {
    private  ApplicationContext ctx;
    public   void setApplicationContext(ApplicationContext ctx){
        this.ctx=ctx;                          (h)   (overlying  IOC container object injection)
    }
                                    (c)
    public String  purchaseCar (String modelName, float price  , String colour , String executive){
      //get Target  class object
      CarShowRoom target=ctx.getBean("showRoom",CarShowRoom.class);
      //get Advice class object
      TestDrivingAdvice advice=ctx.getBean("drivingAdvice",TestDrivingAdvice.class);
      //get  Target  method details
      Method method=target.getClass().getDeclaredMethod("purchaseCar");
      //prepare  JoinPoin object
      JoinPoint jp=new MethodInvocationProceedingJoinPoint();
       jp.setSignature(method);
       jp.setTarget(target);
       jp.setArgs(new Object[]{modelName,price,colour,executive});

      //invoke advice  method
        advice.testingDriving(jp);  (n)
      //invoke target method                  (o)
(m)   String result=target.purchaseCar(modelName,price, colour, executive);
      return result;   (o)
    }
}
```

Internal Cache of IOC container

showRoom	CarshowRoom$Proxy object ref (i)
drivingAdvice	TestDrivingAdvice obj ref (k?)

Client App

```
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;                                  (e) Pre-instantiation of singleton scope beans

public class AspectjAOPBeforeAdviceTest {
      public static void main(String[] args) {  (e)                 (d) prepareing InMemory MetaData
            ApplicationContext ctx=null;                                of  applicationContext.xml file
            CarShowRoom proxy=null;
            //create IOC container   (g)                        (c)  checking wellformed ,valid or not
            ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
            //get Proxy class object
(i)         proxy=ctx.getBean("showRoom",CarShowRoom.class);  (j)
            //invoke the b.method
            try {                   (m)
                  (o)               System.out.println(proxy.purchaseCar("swift",800000, "red","karan"));
            }
            catch(Exception e) {
                  e.printStackTrace();
            }

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

      }//main
}//class
```

Before Advice --Security Check use-Case

```
client App 1
==========
  signIn("raja","rani");  (a)
proxy.withdraw(1001,7000);  (b)
proxy.deposite(1002,8000);
  signout(); (d)
```

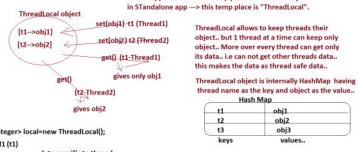
taget class
==========

```
public class  BankSErvice{

  public String withdraw(long acno, float amt){
      ..
      ...
  }
  public  String deposite(long acno,float amt){
      ....
      ..
  }
}
```

temp place
raja rani -1 (a)
(d) (b) (c)

```
{Before  Advice}
public class SecurityAdvice{
                    (c)
  ....  //Authentication logic
  ...

}
```

In web applicaiton -->this temp place is "Session obj".
in STandalone app --> this temp place is "ThreadLocal".



ThreadLocal allows to keep threads their object.. but 1 thread at a time can keep only object.. More over every thread can get only its data.. i.e can not get other threads data.. this makes the data as thread safe data..

ThreadLocal object is internally HashMap having thread name as the key and object as the value..

Hash Map

keys	values..
t1	obj1
t2	obj2
t3	obj3

```
ThreadLocal<Integer> local=new ThreadLocal();
   form Thread1 (t1)
   ----------------          data specific to thread
   local.set(10);    //to keep in ThreadLocal
                     Integer Wrapper class object

   local.set(20);  // to modify Thread data in ThreadLocal

   Integer i1=local.get() ;  // to read thread data from ThreadLocal

   local.remove() ; // to remove  thread data from ThreadLocal
```

ThreadLocal is introduced in java 1.4 and generics support is added in java 6 onwords..

Scopes of data in standalone App

block --> specific to a block
Local -->specific to a method
instance --> specific to a object
class --> specific to a class
thread --> specific to a thread..

Scopes of data in web application

request scope -->specific to each request
session scope -->specific to each browser s/w of a client machine
application scope --> specific to each web applicaiton
page scope --> speciifc to each servlet/jsp comp

The data keep cache is not specific one client or thread.. it is common for all the threads.. i.e the data kept in cache by one thread can be used by other threads.... where as ThreadLocal data is specific to each thread.

```
AOPProj8-AspectAOPDecl-BeforeAdvice-SecurityCheck
   |--->src/main/java
      |-->com.nt.dao
            |----> AuthenticationDAO.java
            |----> AuthenticationDAOImpl.java
            |----> BankDAO.java
            |----.>BankDAOImpl.java
      |-->com.nt.dto
            |----.>UserDetailsDTO.java
      |-->com.nt.bo
            |----.>UserDetailsBO.java
      |-->com.nt.manager
            |----.>AuthenticationManager.java
      |-->com.nt.service
            |----.>BankMgmtSevice.java
            |----.>BankMgmtServiceImpl.java
      |-->com.nt.cfgs
            |----> persistence-beans.xml
            |----.>service-beans.xml
            |----.>aop-beans.xml
            |----.>applicationContext.xml
      |--->com.nt.test
            |------>BeforeAdviceSecurityTest.java
      |--->com.nt.aspect
            |----.>SecurityCheckAspect.java
      |-->pom.xml
            |-->dependecies ::  spring-context-<ver>.jar , ojdbc6.jar , spring-jdbc-<ver>.jar
                              aspectj-rt-<ver>.jar , aspectjweaver-<ver>.jar
                              hikaricp-<ver>.jar
```

Before Advice / Method Before Advice

=>This advice executes before executing target method ... After executing advice method automatically control goes to taget method

=>To develope this advice in AspectjAOP take method having the following signature in the java class...

```
public class <class name>{

  public void <methodname>(Joinpoint jp)throws Throwable{
     ....
     ....
  }
}
```

pointcut exp :: execution(<return type> <pkg>.<classname>. <method name>(..))

What the diff b/w JoinPoint and ProceedingJoinPoint?

JionPoint object gives target method details being from advice method.. but we not invoke/call target method using this object

PreceedingJionPoint object gives target method details being from advice method.. and also allows to invoke/call target method using this object

(or)

(Best)

```
public class <Classname>{
  public void <method-name>(param1, param2,param2 ,...){
     ....
     ...
  }
}//class
```

poincut expr :: execution(<return type> <pkg>.<classname>.<method>(..)) and args param1,param2,param3

(or)

```
public class <classname>{

  public void <method> (JoinPoint jp,param1,param2,param3 ,..)throws Throwable{
     ...
     ...
     ...
  }
}
```

poinct expr:: execution(<return type> <pkg>.<classname>.<method>(..)) and args param1,param2,param3

=> Before enterting target method ,the before advice method executes ..after the control automatically goes to target method .. once the target method execution is over .. the control did not come back advice method.. So there no need catching target method return value.. in advice method..

usecases:: Testing drivng before purchasing car..
Auditing Advice (keeps track who user is calling which method)
Security check Advice (perform authentication check)
reading terms and conditions before accepting the offer
reding the rating of the product before buying the product..

logging keeps track of code flow...
auditing keeps track of user activities in the App
=> we write auditing info to audit log file... similarly logging info log files..

important points /control points

=>1.an access and modify target method arguments values being advice method (possible in spring AOP, not possible in aspectjAOP)

=> can not control target method execution.. becoz from advice method the control automatically goes to taget method..
note:: By throwing exception in advice method.. we can stop control going to target method..

=> we can not access and modify target method retrun value.. becoz after executing target method control does not come back to advice method..

The manager who is Loan approver checks cbil score/credit score and others to sanction or reject.. In that Process i wan to keep track of manager and Applicaiton activities through auditing with the supporrt audit log file..

Throws Advice flow of execution

ShoppingStore.java (Taget class)

```
public class ShoppingStore {                                    (p)
    public float calculateSingleProductBillAmt(float price ,float qty ) {
        if(qty<=0.0f || price<=0.0f)
            throw new IllegalArgumentException("Invalid inputs");

        return  price*qty; (q)
    }
}
```

Advice class

CommonExceptionLogger.java

```
public class CommonExceptionLoggerAspect {
    private static Logger logger=Logger.getLogger(CommonExceptionLoggerAspect.class);
    static {
        try {
            PropertyConfigurator.configure("src/main/java/com/nt/commons/log4j.properties");
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
                                (t)
    public void    exceptionLogger(JoinPoint jp,IllegalArgumentException ex) {
        logger.error(ex+ " exception is raised in  "+jp.getSignature()+ " method 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"
    xmlns:context="http://www.springframework.org/schema/context"
    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-4.2.xsd
        http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context-4.3.xsd">
    <!-- cfg taget class -->
    <bean id="shopping" class="com.nt.service.ShoppingStore"/> (d)              (d) Pre-instantation of
    <!-- Cfg ASpect class -->                                                     singleton scope beans..
    <bean id="expAspect" class="com.nt.aspect.CommonExceptionLoggerAspect"/>  (d)

    <!-- enable aspectjAOP -->
    <aop:config> (e)
        <aop:pointcut expression="execution(float com.nt.service.ShoppingStore.calculateSingleProductBillAmt(..))" id="ptc"/>
                                                                                (f)
        <aop:aspect ref="expAspect">
            <aop:after-throwing method="exceptionLogger" pointcut-ref="ptc" throwing="ex" />
        </aop:aspect>

    </aop:config>


</beans>
```

Generated Proxy class Using CGLIB Libraries

```
                                                (g)
public class  ShoppingStore$Proxy extends ShoppingStore  implements ApplicationContextAware{
   private ApplicationContext ctx;
   public  void setApplicationContext(ApplicationContext ctx){
     this.ctx=ctx;                              (h)
   }
                        (n)
   public  float  calculateSingleProductBillAmt(float price, float qty){

     //get Target  class object
      ShoppingStore target=ctx.getBean("shopping",ShoppingStore.class);
      //get Advice class object
      CommonExceptionLogger advice=ctx.getBean("expLogger",CommonExceptionLogger.class);
      //get Target  method
     Method method=target.getDeclaredMethod("calculateSingleProductBillAmt");
     // prepare Array having args
      Object args[]=new Object[]{price,qty};
     //create JoinPoint object
      JoinPoint jp=new MethodInvocationJoinPoint();
      jp.setSignature (method);
      jp.setTarget(target);
      jp.setArgs(args);
     //invoke taget method method
     float billAmt=0.0f;
     try{                               (o)
(r)   billAmt=target.calculateSingleProductBillAmt(price,qty);
      }
     catch(IllegalArgumentException e){

       //invoke advice method          (s)
(u)    advice.exceptionLogger(jp,e);
       return e; (u)
      }
       return  billAmt;
    }
}
```

Internal Cache of IOC container (i)

keys	values
expLogger	CommonExceptionLoggerAspect obj ref
shopping	ShoppingStore$Proxy object ref
	(k?)

Client App code

```
public class ThrowsAdviceTest {     (a)
    public static void main(String[] args) {
        ApplicationContext ctx=null;
        ShoppingStore proxy=null;
     //create IOC container          (b)                    (c)-->InMemory MetaData
        ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
        //get Proxy object            (j)
(l)     proxy=ctx.getBean("shopping",ShoppingStore.class);
        try {                                                (m)
            System.out.println("Bill Amount is ::"+proxy.calculateSingleProductBillAmt(10000, 0));
            }
            catch(IllegalArgumentException iae) {
                iae.printStackTrace();   (v)
            }

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

    }//main
}//class
```

Q) if multiple throws advice methods are having same name with same no.of parameters then what happens?

Exception will be raised...

```
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 3 candidates.
```

note:: In the above scenario, if any advice method is having less no.of parameters then it will get priority to execute..

Throws Advice

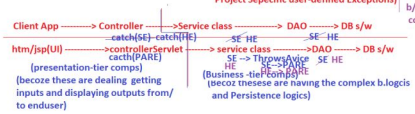
=>This advice executes only when exception is raised in target method..
=> This advice executes even for the exceptions that are raised in the advice methods that are applied on the target method.
=> To develop this...

```
public class <Classname> {

                                        Either Throwable or Exception type
  public void <method> (JoinPoint jp, parameter to get and hold the raised exception)throws Throwable{
      ....
      ....
      ....
   }//method
}//class
```

(or)

```
public class <Classname>{      Either Throwable or Exception type
   public void <method>(param to hold the obj of raised exception){
      ...                                  exp
      ...
      ...
   }
}
```

usecases:: (a) Common Exception Logger (special log file only having exception related messages) | Useful for incident Management
(b) Common Exception Grapher (To translate Technology exceptions to Project Sepecific user-defined Exceptions) | Very useful to achieve loose coupling b/w presentation tier and business tier comps.



note:: we should always develop both business tier and presentation tier comps having loose coupling... i.e if change the techology or code of business tier compps.. that should effect the code of the code of presentation tier comps..

imptorant points /control points

(a) We can access taget method argument values.. but we can not modify
(b) We can not access target method return value..
(c) We can not control target method excution , becoz control comes advice method from target method itself...

In spring bean cfg file

```
<aop:aspect ref="<aspect spring bean id">
   <aop:after-throwing name="<advice method name>"
                       point-cut-ref="<ptc id>"
                       throwing=" param name in advice method that is ready to hold
                                  the raised exception"/>
                                                    exp
</aop:aspect>
```

Procedure to develop AspectjAOP application using gradle

step1) Make sure that Build ship plugin in Eclipse IDE..

step2) Create Gradle Project in Eclipse having name :: AOProj11-AspectjAOPDecl-ThrowsAdvice
File -> project ---> gradle -->GradleProject -> Project name:: AOProj11-AspectjAOPDecl-ThrowsAdvice
select gradlewrapper -----> select auto sync checkbox --->finish

step3) add the following code in build.gradle

build.gradle

```
plugins {
    // Apply the java-library plugin to add support for Java Library
    id 'application'
}

ext{
  ajver="1.9.4"
}

repositories {
    mavenCentral()
}

dependencies {
    // https://mvnrepository.com/artifact/org.springframework/spring-context-support
implementation group: 'org.springframework', name: 'spring-context-support', version:
'5.2.8.RELEASE'
    // https://mvnrepository.com/artifact/org.aspectj/aspectjrt
implementation group: 'org.aspectj', name: 'aspectjrt', version: "$ajver"
    // https://mvnrepository.com/artifact/org.aspectj/aspectjweaver
implementation group: 'org.aspectj', name: 'aspectjweaver', version: "$ajver"

}
```

spring AOP Around Advice Flow of execution

BankService.java (Taget class)

```
package com.nt.service;
public class BankService {
    // SI/100                                    (a)
    public float calcSimpleIntrestAmount(float pAmt,float rate,float time) {
        System.out.println("BankService.calcSimpleIntrestAmount() Target Method1 ");
        return (pAmt*rate*time)/100.0f; (y)
    }//method
    //p(1+(r/100)^n -p
    public  float calcCompoundIntrestAmount(float pAmt,float rate,float time) {
        System.out.println("BankService.calcCompoundIntrestAmount();->Target method2");
        return (float) ((pAmt*Math.pow((1+rate/100),time)))-pAmt;
    }//method
}//class
```

PerformenceMonitoringAdvice.java

```
package com.nt.advice;
import java.util.Arrays;
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;
public class PerformenceMonitoringAdvice implements MethodInterceptor {
    @Override                                   (a)
    public Object invoke(MethodInvocation invocation) throws Throwable {
          System.out.println("PerformenceMonitoringAdvice.invoke()"+ invocation.getClass());
          long start=0,end=0;
          Object retVal=null;
          System.out.println("pre");
          start=System.currentTimeMillis(); //pre-logic
          retVal=invocation.proceed(); //invokes the method  (p)
          System.out.println("post");
          end=System.currentTimeMillis(); //post logic
          System.out.println(invocation.getMethod().getName()+" with args
"+Arrays.toString(invocation.getArguments())+" has taken "+(end-start)+" ms");
          return retVal; (t)
    }//invoke
}//class
```

applicationContext.xml (d) pre-instantiation of singleton scope beans

```
<?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-4.3.xsd">
  <!-- configure advice class -->                                    (d)
  <bean id="pmAdvice"  class="com.nt.advice.PerformenceMonitoringAdvice"/>

  <!-- Configure target class -->                                    (d)
  <bean id="bankService" class="com.nt.service.BankService"/>
   <!-- Cfg ProxyFactoryBean to generate Proxy class  and to return its object -->
  <bean id="pfb" class="org.springframework.aop.framework.ProxyFactoryBean">   (d)
    <property name="target" ref="bankService"/>
    <property name="interceptorNames">
      <array>                                                       (d)
        <value>pmAdvice</value>                                          (e) activates
      </array>                                                           spring AOP
    </property>
  </bean>
```

CGLIB Generated

InMemory Proxy Class (Generated in the JVM Memory of RAM) (f)

```
public class BankService$Proxy extends   BankService implements ApplicationContextAware {
    private  ApplicationContext ctx;                          To perform Awareinjection

    public  void setApplicationContext(ApplicationContext ctx){    //aware  injection..
      this.ctx=ctx;       (g)                                  To inject underlying ApplcationContext obj
    }
                          (m)
    public  float calcSimpleIntrestAmount(float pAmt,float rate, float time){
      //get access to target  class object
      BankService service=ctx.getBean("bankService",BankService.class);
      // get Advice class object
      PerformenceMonitoringAdvice advice=ctx.getBean("pmAdvice",PerformenceAdvice.class);
      Method method=service.getClass().getDeclaredMethod("calcSimpleIntrestAmount");
      //prepare MethodInvocation object having target method details
      MethodInvocation invocation=new CGLIBAOPProxy$CGLIBMethodInvocation();
      invocation.setTarget(service);
      invocation.setMethod(method);
      invocation.setArgs(new Object[]{pAmt,rate,time});
      // call invoke() method on Advice class obj
  (n) Object retVal=advice.invoke(invocation)   (n)
      return retVal;  (q)
    }

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

      ...
      ...
      ...
      ...
      ...
      ...
      ...
      ...

     }//method

} // class
```

Internal Cache of IOC container (h)

bankSEvice	BankService class object
pmAdvice	PerformenceMonitoringAdvice obj
pfb	Proxy obj (BankService$Proxy obj) (j?)
(beanids)	Bean class objects

Client app

```
public class AroundAdviseTest {    (a)  start
    public static void main(String[] args) {
          ApplicationContext ctx=null;
          BankService proxy=null;

          //create IOC container     (b)  Container creation        (c) creating InMemory MetaData
          ctx=new ClassPathXmlApplicationContext("com/nt/cfgs/applicationContext.xml");
          //call getBean() to Proxy class object                                  checking wellformed ,valid or not
  (k)     proxy=(BankService)ctx.getBean("pfb",BankService.class);  (i)
          try {                                                                  (l)
            System.out.println("Simple Intrest Amount:"+proxy.calcSimpleIntrestAmount(100000, 12,2 ));  (w)
            System.out.println("-------------------------");
            System.out.println("compound intrest amount:"+proxy.calcCompoundIntrestAmount(100000, 12,2));
          }
          catch(Exception e) {
                e.printStackTrace();
          }
          //close container
          ((AbstractApplicationContext) ctx).close();
    }//main
}//class
```

note: When no pointcuts are configure .. All advices will be applied on all the target methods (joinpoints) of target class..

=> While Generating Proxy classes using CGLIB Libraries some limitations while designing Target class becoz here Proxy class comes as the sub class of the target class.. are there

(a) Target class can not be final (becoz final classes can not sub classed)

(b) Target class target method can not be final or static becoz final and static method can not overriden..

solution :: Work with JDK Libraries to generate the proxy class as impl class of proxy interface not the sub class of target class..

=>To guide spring AOP/ProxyFactoryBean to generate the proxy class as the implementation class of proxy interface we need to

(a) Make target class implementing Proxy Interface |—>interface having the decl of b.methods/target methods

(b) specify Proxy Interface name as input value/injected value for "ProxyFactoryBean" (optional) |—proxyInterfaceNames []

//Sample Proxy class code when it is generated through Jdk Libraries (spring AOP Decl around advice)

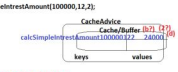
```
public class $Proxy5 extends  java.lang.reflect.Proxy  implements IBankService,ApplicationContextAware{
   private ApplicationContext ctx;

   public void setApplicationContext(ApplicationContext ctx){
      this.ctx=ctx;
   }

   public  float  calcSimpleIntrestAmount(float pAmt,float rate,float time){
      //get Advice class object
      PerformenceMonitoringAdvice advice=ctx.getBean("pmAdvice",PerformenceMonitoringAdvice.class);
      //get Target class object
      IBankService target=ctx.getBean("bankService",IBankService.class);
      //get Method object pointing to Target method
      Method method=target.getDeclaredMethod("calcSimpleIntrestAmount");
      // prepare Object[] having target method args
       Object args[]=new Object[]{pAmt,rate,time};
       //create MethodInvocation object
       MethodInvocation invocation=new ReflectiveMethodInvocation();
        invocation.setTarget(target);
        invocation.setMethod(method);
        invocation.setArgs(args);
        //call advice class invoke method
          Object retVal=advice.invoke(invocation);
        return retVal;
     }

     public  float  calcCompoundIntrestAmount(float pAmt,float rate,float time){
        ....
        ....
        .....
     }
}
```

F3 or ctrl + click on code :: Takes u to source code
ctrl +shift + T = To get source code any class/interace/... by typing the class name or interface name

If we apply multiple around advices on the target method , then the order of their execution will be in the order they are cfg in applicationContext.xml as the array element values of "interceptorNames" property..

Caching /buffering usecase

```
proxy.calcSimpleIntrestAmount(100000,12,2);
                (1)
proxy.calcSimpleIntrestAmount(100000,12,2);
    (3)
```

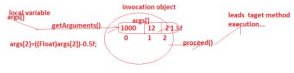


(a to e) :: method call with inputs
(1 to 3) :: method call with same inputs

(Cahing improves the performence..)

=>Generally we develop caching logics in singleton java class to make all requests and comps using single copy of cache across the multiple requests or invocations.. When we develop same logic in singleton scope spring bean there is no need of making it singleton java class.. becoz the singleton scope itself given singleton behaviour..

While developing around advice in spring aop
--> we can access and modify the target method arg value
--> we can access and modify taget method return value
--> we can control target method execution.



Pointcut

==========

Pointcut is collection of join points on whom aspects are adviced.. out of multiple joinpoints (target class methods) we can apply aspects only specific methods that refered by pointcuts..

(refered at Pointcut() (filtering methods)

Q) Can we bring pointcut effect with out using the support of pointcut ?

Ans) Yes ... but not recomanded... becoz it does not generate optimized code in proxy class..

```
public class PerformenceMonitoringAdvice implements MethodInterceptor {

    public Object invoke(MethodInvocation invocation){
      if(invocation.getMethod().getName().equals("calcCompoundIntrestAmount")){
          ....
          .... // execute Performence Monitoring logic
          ....
      } retrun retVal;
  }
```

The PRoblem here is ... Proxy class override all the methods taget class and those call invoke(..) advice class internally having clummy but only certain method "calcCompoundIntrest Amount" the advice logic will be applied with last minute verification.. This says our proxy class code is not optimized code...

Pointcut

==========

=> Every Pointcut is a class that implements Pointcut(I) having set of method names/join points on whom we want to advice the aspects..

Two types of pointcuts

==================

(a) static pointcuts

--> applies advices on the methods based on the given method names and target class name

eg:: apply Advice only when the method name is "add" in ArithmeticService class

(b) dynamic pointcuts

--> applies advices on the methods based on the given method names , target class name and method argument values..

eg:: apply Advice only when the method name is "add" in ArithmeticService class and If the arg values are >=100

Pre-defined static pointcut classes

==========

=>StaticMethodMatcherPointcut (AC)

=>NameMatchMethodPointcut (CC)

=>JdkRegexpMethodPointctu (CC)

=> here we get matches(-,-) having two args like target classname and method name.

pre-defined dynamic Pointcut classes

==========

=>DynamicMethodMatcherPointcut (AC)

=>Here we get matches(-,-,-) having 3 args like target class names, method name and arg [] values.

note: In both cases if matches() method returns true for certain target method then that participates in proxy class generation to apply advices.. otherwise not..

Advisor

=======

=> It is the combination of advice +pointcut

=> we give Advisor to ProxyFactoryBean along wth target class object, So the advice of Advisor will be applied on certain taget class taget methods that are specified in pointcut of advisor.

Ready Made Advisor classes

==========

=>DefaultPointcutAdvisor (with no implicit pointcuts)

=>NameMatchMethodPointcutAdvisor (with implicit pointcut)

=>JdkRegexpMethodPointcutAdvisor (with implicit pointcut)

All these advisor classes are implementation classes of Advisor(I).

Procedure to work with Pointcuts

==========

(a) Develope target class

(b) Develop advice class

(c) Develop pointcut class extending abstract pointcut class or choose concrete Pointcut class

(d) develope spring bean cfg file

i) cfg target class as spring Bean

ii) cfg Advice class as spring bean

iii) cfg user-defined/pre-defined Pointcut class specifiying /having method names

iv) Cfg Advisor (generally Pre-defined) injecting advice,pointcut

v) Cfg ProxyFactoryBean injecting

->taget class

->advisor

Generates the optimized code in Proxy class..

(e) Develop the Client App

same as any old app..

note: we do not concret classes representing Dynamic Poincut and do not pre-defined Advisor class having dynamic Pointcut as implicit class..So dynamic put should always be developed as user-defined pointcut class...