Spring Material: Part-3

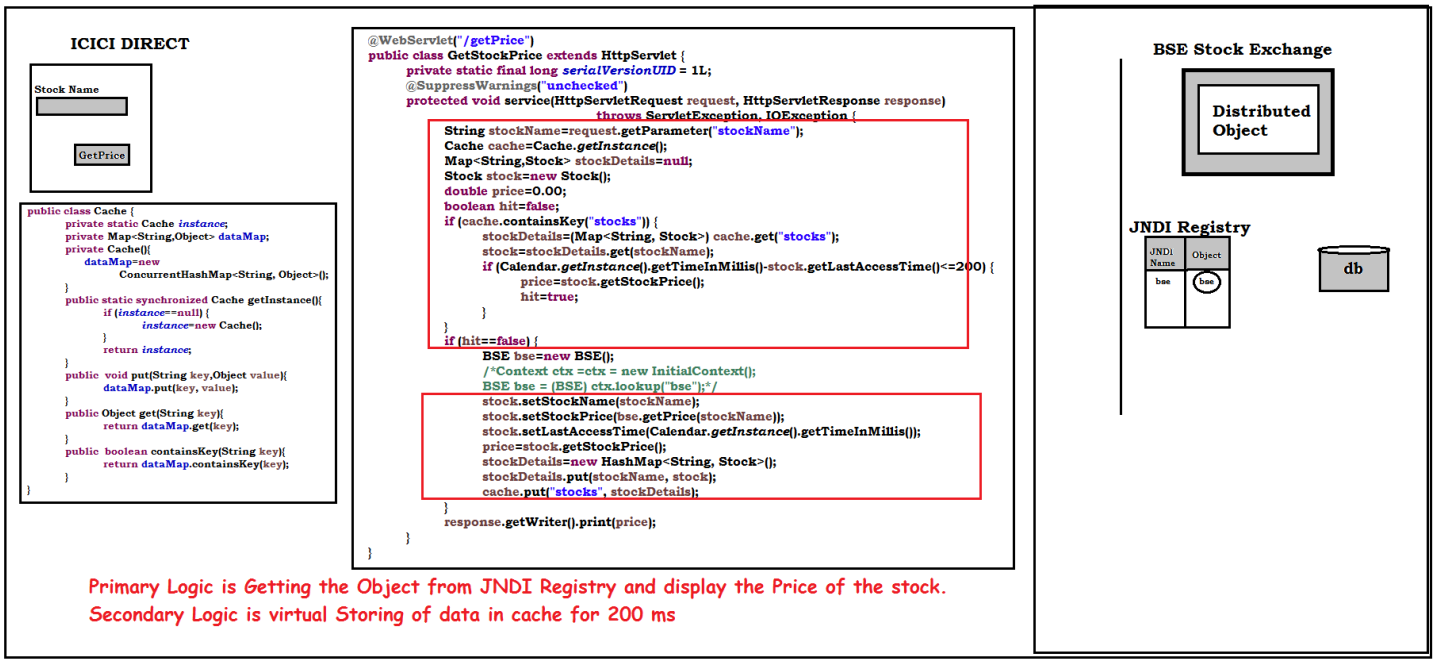
AOP (Aspect Oriented Programming)

What is AOP?

➔AOP is the process of separating the primary logic from the secondary logic.

➔**AOP is made for managing and applying the crosscutting logic that seems to be common across several classes with in our application.**

➔We can say it is the methodology which will separate the crosscutting logic from the primary logic or core logic.



For Example:

➔In an application we are going to write some logic.

➔The logic that we have written will address various different aspects of the application.

➔Some logic related to fetching the data from remote application server.

➔Some logic related to optimize the performance of the application like caching logic.

➔We are writing multiple pieces of logic various different aspect of application.

➔The logic that we are writing can be classified into two types primary logic and secondary logic.

Which logic can be called as primary business logic?

➔The logic that seems to be mandatory without having that logic the existence of my application will not be there such kind of piece of code is being considered as primary business logic .

Which logic is called secondary logic?

➔The logic that is not mandatory but that will exist as part of application even the presence of that logic or absence of that logic the core business functionality of the application will not get impacted such kind of logic called as secondary logic.

If without existence of secondary logic if application will work then why we are writing secondary business logic?

➔Secondary logic acts as a helper logic that helps the primary business logic to work in a better way.

➔But always it is not mandatory because without secondary logic also my primary logic will work in such case my secondary logic is optional.

Why secondary logic called as crosscutting logic?

➔The secondary logic will not appear in one place of my application, most of the time secondary logic is common logic that may be write in multiple places of the application.

➔For example

caching is applied not only for getting the stock price cache may have to applied for various other functionality that are part of our application.

➔That itself tells you the secondary that we are writing is by nature is common logic that can be appear in various different places which can be called as cross cutting logic .

What is the problem when we are writing secondary business logic with primary business logic?

➔We are writing secondary logic and primary logic intermingle with each other and they are written at one place within our application but there is a problem if I want to separate the secondary logic it is not so easy.

➔As it is optional sometimes I wanted and sometimes I don't wanted to have the secondary logic .In this case if I don't wanted to apply the secondary logic then I can't easily separate the secondary business logic from application.

➔Because these are intermingle with each other so it is not so easy to separate secondary business logic from primary business logic,

➔We may have to modify lots of code within our application component. Some time I may have to crash the existing logic that has been written and rewrite the whole functionality of my class.

➔In this we can implement Flag based solution in our application let's see what is it?

Flag Driven Development

➔We can go for flag driven development and set the flag and you can enable certain feature by enabling the flag or you can disable the flag.

➔With out AOP, in core java application we can enable the system property and manage the cross cutting logic.

➔Whenever I get the request I have to add system property and ask for enable cache =false enableCache =system.getproperty (enableCache).

➔Now checking whether the data is in cache or not if enable cache =true then only do this .

➔In our Application we are passing the system property as enableCache=true then cache logic will be executed.

➔If I enableCache=false then Never the pieces of logic corresponding to the cache will not executed.

➔Means my own application will not store into the cache, if I need cache then I enable cache =true .

➔So now I can add my cross cutting logic to my primary business logic or I can remove my cross cutting logic. I can do it such kind of technique or can accomplish such a behaviour by using the flag driven development.

Problems with Flag Based Solution

➔In flag driven development how we enable the cache or how we disable the cache without using AOP.

➔We need to pass a flag to the application using which we can enable the specific aspect of a functionality of your application.

When I send the request and enableCache=true then what will happen?

➔Servlet receive the parameter value then check enable cache is true or false it is true then check data is available in cache or not.

➔If it is there it is use if it is not there it goes to the remote application get the price of stock store in the cache then display the price.

➔Then 2nd request came it check whether cache is enable or not if it is enable go to the cache check whether the cache contains the data or not? Yes data is there so it display.

➔My application is running In production everything is going fine then suddenly BSE stock exchange people is came , What is the BSE stock exchange people say you shouldn't use cache ,don't use cache as part your application then what I need to do?

➔Then we have to change enableCache=false. Now Cache will not be apply and Cache has not removed,

➔If you don't wanted it why to wanted to specify your application and every time you wanted to have an extra check.

➔Suppose BSE stock exchange people say Remove Cache means you have to remove whole life time of the application you should not use cache.

➔The whole life time of the application i don't want to cache, we are running application but without caching functionality .

➔It is Exist in our application even it is not been use and additional condition check will be happen runtime even you don't want caching .

➔These logic never required for the life time of the application .Then using the flag eats the CPU time.

➔To evaluate the cache It is not optimal solution performance issue problem will come.

➔So we need to ignore such type of code in my application.

➔Event it is not required we never use the code we are not remove it the code it is logically there. So unnecessarily it is evaluated for every request of our application is it a problem .

➔If I never remove your code from the application .It is additional maintenance .

➔Still cross cutting logic is in my actual application even if you don't want it. It is a Dade code and consume memory

➔To resolve these types of problems we should go for AOP.

Why we should go for AOP?

➔There are number of pitfall (problem) when we use the secondary logic with primary logic.
➔ Crosscutting logic is the logic which can be used in many places into the application, if we mix both primary and secondary logic we end up with duplicating the code across the application.
➔ If we write crosscutting logic with primary logic and there is change in secondary logic it may affect the primary logic also.
➔ If both written in one single place programmer much worry about secondary rather than primary logic.
➔ If both written in one place, we cannot easily separate them.
➔ Some time if we don't want to execute secondary logic then, we cannot escape, even though if we place condition to evaluating the condition and escape, but the presence of the code will be there into the application. We cannot eliminate permanently.
➔Aop is a programming methodology. It means it is programming principle which helps in running the primary business logic and cross cutting logic (secondary logic) written in two different components without referencing each other.
Note:- AOP is not a programming language.

AOP Principles

There are seven principles in AOP these are:

- ➔Aspect
- ➔Advice
- ➔JoinPoint
- ➔Pointcut
- ➔Weaving
- ➔Target
- ➔Proxy

Aspect:

➔It **is piece of code (**secondary logic or crosscutting logic**) which will be separate from the primary logic that has to be applied across various classes of the application.**

Advice:

➔**This principle talks about how we can apply that aspect.**
➔There are multiple types of advices are there

- ➔Before advice
- ➔After advice
- ➔Around advice
- ➔Throw advice.

JoinPoint:

➔**This principle will tells about how many places we can advice the aspect**. Generally in spring we can apply an aspect at method execution.

PointCut:

➔**set of joinpoint where actually we advice the aspect will describe by pointcut.**

Weaving:

➔The process of advising the aspect of a target class based on a point cut to build proxy is called weaving.

or

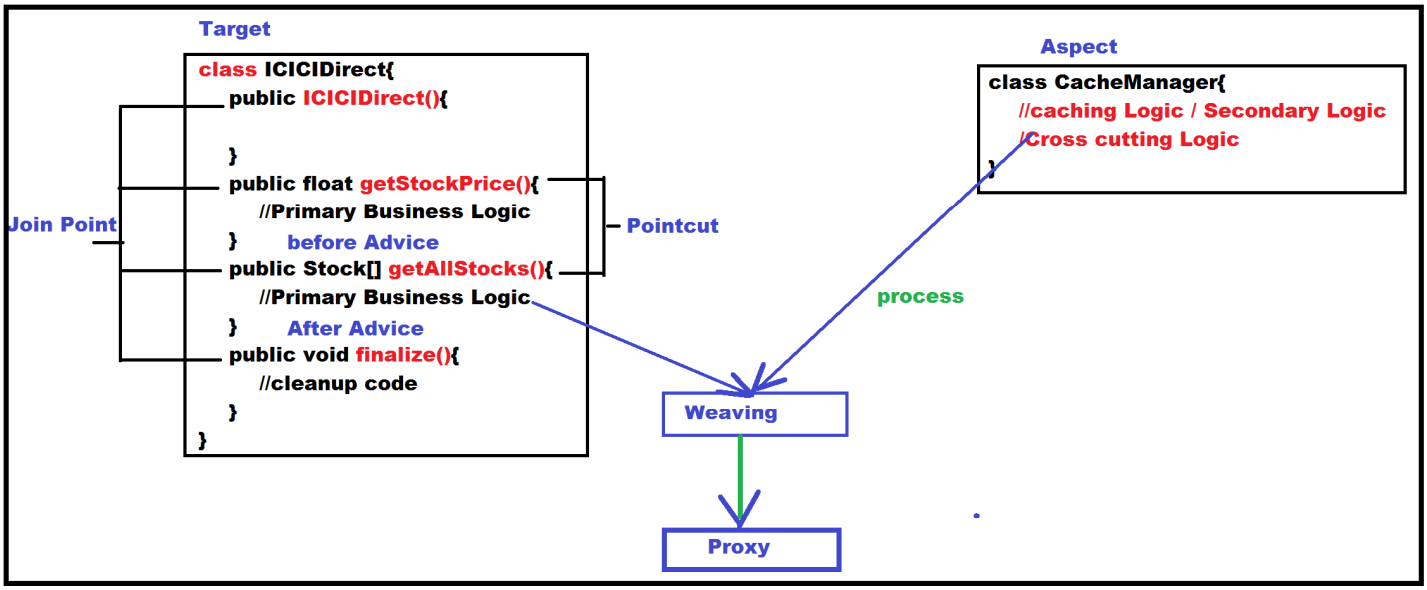
➔ The combination of aspect and PointCut called as weaving.

Target:

➔ On which class we are going to apply the aspect.

Proxy:

➔ After Weaving we will get the proxy as the result



There are three ways of working with spring AOP these are:

➔Spring programmatic approach/Spring AOP API

➔Spring Aspect-J declarative approach

➔Annotation driven Aspect-J approach AOP.

➔AOP is not particular to specific spring rather it is a programming technique similar to OOP, so the above principles are generalized principles which can be applied to any framework, supporting AOP Programming style.

➔There are many frameworks in the market which allows you to work with AOP Programming few of them are…

Open Source Aspect-Oriented Frameworks in Java

➔AspectJ	➔AspectWerkz	➔Nanning
➔JBossAOP	➔Dynaop	➔CAESAR
➔EAOP	➔JAC	➔Colt
➔DynamicAspects	➔PROSE	➔Azuki Framework
➔CALI		

What is Difference Between Spring and AspectJ?

Spring AOP	Aspect J AOP
➔Only Support Joinpoint as methods.	➔It supports various types of joinpoints Constructor Instance Method Static Method Instance Block Static Block finalize() finally
Spring supports static and dynamic pointcuts.	Aspect J supports only static pointcut
➔Spring supports Runtime Weaving ,that means proxy object will be created at runtime in the JVM Memory.	➔Aspect J using compile-time weaving, this indicates your proxy classes will be available whenever you compile the code.

How many types of PointCut are there?

➔PointCut is the set of Join point where advices are applied to execute the Aspect.

➔In principle of Aop itself these point cut are 2-type .

- ⇨ Static PointCut
- ⇨ Dynamic PointCut

➔First let's try understanding what is a static point cut and what is a dynamic Point cut .

```
public class Runner {
        public void timer (int sec) {
                //Business Logic
        }
        public int run(int km){
                //Business Logic
        }
}
```

➔I have some crosscutting logic I wanted to apply this crosscutting logic on my target class.

➔What is JoinPoint? It is the point on the target class where Advice can be applied to execute the Aspect. What are those? timer (…) and run(…).

➔Now I want to apply on Runner what do I need to do? I will create point cut representing this join point.

➔I don't want to apply Runner i go to AOP say what this is my target this is my aspect this is the PointCut you create proxy .

➔I have target class i have crosscutting logic both I have give to Aop but in target class there are multiple join Point ,how do i need to tell these point cut ?

➔Create a PointCut with method name then what the Aop will do? It goes to the target class looks for join point two are there goes to the PointCut .

➔How many i told? One then the aspect will apply on one join point that is specified by PointCut that use by proxy.

➔Now what **i created PointCut with what fixed method name this is static PointCut.**

➔I told this is my target class this is my crosscutting logic and has to apply on PointCut,

➔When you have to apply if someone call the method with fist parameter value is greater than ten then only apply that **means timer() has called with minute has more than ten minute then only apply the aspect on target it means proxy will created or not ?**
➔Proxy will be created but when? At run time when the method has called because then only we know the value of the minute.

➔Proxy will created at run time because we have condition applied .that means aspect has to applied or aspect shouldn't applied decided based on condition means that PointCut is dynamic PointCut .

How many types of weaving are there?

➔The process of combining the aspect in the Target class join point is produces Proxy the whole process is called weaving.
There are two types of weaving

➔Compile Time Weaving
➔Run Time Weaving

Compile Time Weaving

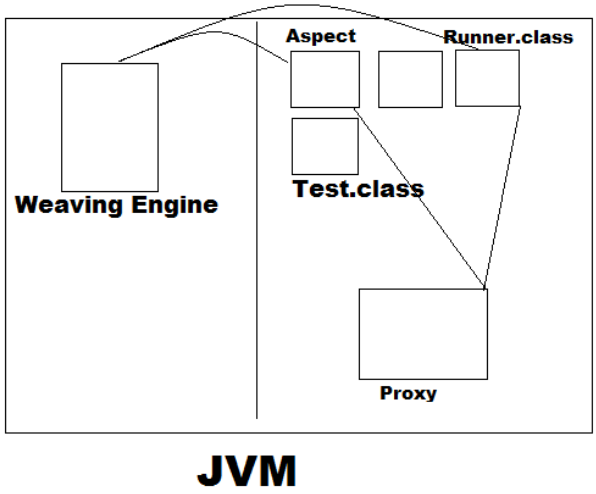
➔We will not write crosscutting logic within the target class
➔We have to write the aspect logic with in a separate class.
➔We have to keep target class separately, and Aspect class separately.
➔Now how can we combine these two how can we build two together that's why fist what weaving compiler say. Compile your classes to java compiler.
➔Then you will get .class file of the target class.
➔Then compile your aspect logic with java compiler now you will get .class file.
➔These two are separately now we have to combine and execute, When two .class file executed together when both are in the same class file.
➔That's why every **Aop framework provides weaving compiler .**
➔Now we have to pass target. Class + aspect. class so that they can compile and create proxy .class file.
➔This is called compile time weaving.
➔As proxy is generating at compile time that's why it is called compile time weaving.

Run time weaving

➔When it comes to runtime weaving all AOP frame work will check some component within your JVM .
➔When you work with Runtime weaving compiler supported Aop framework , those Aop framework vendors will check some component .
➔Component is a class a kind of bunch of libraries or classes that's will loaded and sitting as per the JVM memory
➔Which is one kind of weaving engine or weaving compiler .Which is a class which is present inside the JVM memory now within a JVM memory what else we needed , one is target classes and Aspect classes also .
➔So there is randomly one class within our application let's say some Test class ,The test class want what, target class Runner class

➔What he has to do? Never create a object of target class when we are using Run time weaving support .
➔Every class there is in our application that is part of your JVM should never go and create the object of another class directly.
➔In spring we never create the object of other class .
➔We never create the object of other class otherwise we are not getting the benefit of spring frame work. Unless until the class is in bean that is being manage by IOC container we are not get benefited.
➔So no class in our application directly create the object of other classes.

➔So Every classes who ever want the object has go to Weaving Engine, that is there in the JVm.
➔Asking what can you please give me the runner class. what does the weaving engine or weaving compiler check in runtime ? Is the runner class has to apply with in an aspect or not?
Yes , it has to apply an aspect then it will not create the object of target It will combine Aspect with target and it will create one more class object and return to the Test.



➔Then the proxy is created at Compilation or the proxy has been created at runtime? Proxy is created at runtime that's reason it is called runtime weaving compilation process.
➔There are few frame work that is support runtime weaving,There are few frame work that support compilation weaving depend on they have implemented .

Problem with weaving
➔There are two type of weaving
- ➔Compile time weaving
- ➔Runtime Weaving

➔So what kind of benefit do we have when we go for compile time weaving or Runtime weaving let's comparing the difference between them?

Problems with Compile time weaving
➔We create the logic we compile the class file after creating the .class file we are give the .class file to weaving compiler then proxy will created.
➔That's why it has taken more time because two time compilation required.
➔That is the disadvantage of compile time weaving because after compilation of java compiler we can't run the application rather we can do one more compilation.

➔We wrote Target class we wrote Aspect class, now I need to compile java compiler then with in weaving compilation.
➔Then Proxy will be created now I tested, it is not working now.
➔Then I modified the code in target classes then run it ,Then There could be a chance where we miss again we get weaving compilation process into which the code we have to modified may not be execute and still the old proxy version will get executed due to which there is an chances that they may raises a issue where we couldn't able to find the issues why my code not affected .

➔When we are using Compile time weaving packaging the project is difficult.

Problem with Run time weaving

➔In Runtime weaving whenever we modify the target class type proxy class we no need to re-compile it because never the proxy is created in compile time it is created in run time.
➔Such type of old proxy chance has not in there in runtime weaving .
➔But only one disadvantage in runtime weaving is the proxy is created in run time there will be performance issues there.
➔It take execution time with the application additional with CPU cycle time to creating the proxy. We can also manage only fist time we create the proxy.

Important question:

➔While compairing Spring AOP API approach principle with AspectJ declarative approach there is **no change in Aspect**.

➔In case of **Advice** also there is **no changes** in between Spring AOP approach and AspectJ declarative approach.

How many types of Pointcut are there?

Ans:There are Two types of pointcuts are there:
1.Static pointCut
2.Dynamic pointCut
➔Spring framework support both static pointCut and Dynamic PointCut.
➔AspectJ support only static pointCut not support Dynamic Pointcut.
➔When we comes to Spring AspectJ declarative approach it only support Static pointCut.

What can be act as joinPoint when we are working with AspectJ?

➔In AspectJ approach Instance block,Static block,Instance method,Static method,constuctor,finalize method are act as joinPoint.
➔spring only allow to apply the cross cutting logic inside method only.
Means in the above area we can write our cross cutting logic.
When we are integratting spring with AspectJ then it only support method to apply the Aspect logic.

How many types of Weaving are there?

➔There are two types of weaving are there:
1.Compile time weaving
2.Run time wiving
➔Spring support Runtime weaving.
➔AspectJ support Compile time weaving.
➔When we are integrating spring with AspectJ then it only supports Runtime weaving.

Why spring will not support all the join Point?

➔Spring support only method as join point but AspectJ support all the possible join Point ,
➔When it comes to integrated with spring and AspectJ then it only support method as join Point because spring IOC container manage the join point and Spring integrating AspectJ so if you write any **AspectJ logic in AspectJ it undergoes with spring IOC which support only method as join Point**.

In which circumstances we should go for AOP?

➔Logging (Should not because we can't not capture all the information for logging)
➔Caching
➔Performance Monitoring
➔Security check
➔Auditing
➔Transaction flag based check try {}finally{}

Types of Advices

Around Advice

➔This is the advice which executes the aspect logic (cross-cutting logic) around the target class join point.
➔In this case advice method will be called before the target class method execution and after the target class method execution.
➔This advice can control the target class method execution.

Before Advice

➔In this advice before executing the target class method it will execute the advice method and after finishing the execution of advice method control will not come back to the advice method.

After Returning Advice

➔In this advice method will executes after executing the target class method successfully but before returning the value to the caller.

Throws Advice

➔This advice method will be invoked only when the target class method throws an exception.

Around Advice

➔ We wanted to advice the aspect before the method execution and after the method execution finished then use Around advice.
➔ **Around advice play crucial role in project because of this we can easily track the flow of the application we can easily identify the problems in the application.**
➔ For providing the logging facilities to the application Around advice will plays vital role.
➔ Spring has the support for AOP, to handle every aspect of the AOP, spring has provided different classes and interface.
➔ To work with Around advice spring has provided one of the interface called **MethodInterceptor.**
➔ MethodInterceptor has one method called invoke, which will used for applying the crosscutting logic on target class. And method looks like:

Public Object invoke(MethodInvocation methodInvocation){}

➔ Method Interceptor means Around the method and we can apply the crosscutting logic.
➔ To apply the crosscutting logic we can all the information about the target class without target class information we cannot apply the crosscutting logic.
➔ So invoke method has one parameter called MethodInvocation which able grab all the information of the target class.
➔ Means by help MethodInvocation can we get all information of the target class easily, to get all the information there are several methods are available.

➔ Once we got the all information about the target class now we can perform the before advice and can proceed the execution of the target class by calling methodInvocation.proceed() method.
➔ methodInvocation.proceed() method will call the target class corresponding method, once target method execution has been finished it will return the cursor to the invoke method then again invoke method can apply the crosscutting logic after the method execution.
➔ Invoke method not only apply the crosscutting logic, it can modify the input data, even it can modify the return output also.
➔ Because target class method called by invoke method and target method will return output to the invoke method and before returning the original output we can modify it the return value.

There are three control points available while working with the Around advice.
➔ Once we got all information we can modify the input and proceed.
➔ Method invocation under control invoke method as per requirement we can call or not.
➔ Once we call the target class method, target class method will return output to the invoke method, means again control comes to the invoke method.

➔ Caller will call target class method before execute target method invoke method gets executed this sentence is wrong, **Actually Instead of target class method first Aspect class method gets execute** and the control given to the target class by invoking methodinvovation.proceed() method.

Example:

```
public class Calculator {
        public int add(int a,int b){
                System.out.println("add()");
                return a+b;
        }
}
```

Primary Logic

```
import org.springframework.aop.framework.ProxyFactory;
public class Test {
        public static void main(String[] args) {
                ProxyFactory factory=new ProxyFactory();
                factory.addAdvice(new AspectAdvice());
                factory.setTarget(new Calculator());
                Calculator proxy=(Calculator)factory.getProxy();
                int sum=(Integer)proxy.add(10, 20);
                System.out.println("Sum :"+sum);
        }
}
```

```
import org.aopalliance.intercept.MethodInterceptor;
import org.aopalliance.intercept.MethodInvocation;

public class AspectAdvice implements MethodInterceptor {
        @Override
        public Object invoke(MethodInvocation methodInvocation) throws Throwable {
                String methodName=methodInvocation.getMethod().getName();
                Object[] args=methodInvocation.getArguments();
                Object ret=null;

                //Log Statement before execution

                System.out.print("Enter in to the Method "+methodName+"(");
                for (int i = 0; i < args.length; i++) {
                        if (i==0) {
                                System.out.print(args[i]);
                                continue;
                        }
                        System.out.print(", "+args[i]);
                }
                System.out.println(")");

                //Modify the argument before calling
                System.out.println("Modifing the arguments");
                args[0]=(Integer)args[0]+10;
                args[1]=(Integer)args[1]+10;

                //proceed() calls to tthe target method add()
                ret=methodInvocation.proceed();

                //log statement after execution
                System.out.println("Exit from Method "+methodName+"  and return value is:"+ret);


                System.out.println("Modifing the return value");
                ret=(Integer)ret+10;
                return ret;
        }

}
```

➔Now let's see the control flow of Around Advice When we call proxy.add(10,20)

➔Control does not go to target class add(-) method.
➔Instead of going to target class add(-) method it will goes to Advice class MethodInterceptor invoke(MethodInvocation methodInvocation) method.
➔Internally Aop will pass the details of target class and its method details on which we want to apply crosscutting logic to methodInvocation.
➔Now we can get the details of target class method like following-

String methodName=methodInvocation.getMethod().getName();

➔It will give the name of the method on which we want to apply aspect that is cross-cutting logic.

Object[] args=methodInvocation.getArguments();

➔It will give the arguments value by which target class is called.

Object arg=methodInvocation.getThis();

➔It will gives the target class object. By using target class we can access the attribute which is declared at class level.

Now by getting the details of method we can perform some action on target class method before executing its method.

➔Before calling the proceed () method, we are modifying the parameters, so that with these modified values proceed() will be called ,once the target method execution finished it returns the value as Object in the proceed method call.
➔Here we modified the return value and return.

➔Once the target and advice has been built , we can perform weaving to attach the aspect to the target.
➔In order to perform weaving to build proxy, we need to use ProxyFactory class.
➔And we need to add Advice and Supply the Target class on which you want to apply the advice.
➔The ProxyFactory will apply the advice on that given Target and generates an in-memory proxy class and instantiate and returns that object.

Let's see the Example How to apply cache on Calculator Application

```
public class Calculator {
	public int add(int a,int b){
		System.out.println("add()");
		return a+b;
	}
}
```

```
public class Cache {
	private static Cache instance;
	private Map<String, Object> dataMap;

	private Cache() {
		dataMap = new ConcurrentHashMap<String, Object>();
	}

	public static synchronized Cache getInstance() {
		if (instance == null) {
			instance = new Cache();
		}
		return instance;
	}

	public void put(String key, Object value) {
		dataMap.put(key, value);
	}

	public Object get(String key) {
		return dataMap.get(key);
	}

	public boolean containsKey(String key) {
		return dataMap.containsKey(key);
	}

}
```

```
public class Test {
	public static void main(String[] args) {
		ProxyFactory factory=new ProxyFactory();
		factory.addAdvice(new CacheAdvice());
		factory.setTarget(new Calculator());
		Calculator proxy=(Calculator)factory.getProxy();

		System.out.println("Sum :"+proxy.add(10, 20));
		System.out.println("Sum :"+proxy.add(10, 20));
		System.out.println("Sum :"+proxy.add(30, 50));

	}
}
```

OutPut

```
Method Invoked With Value:add(10 , 20)
Signature of the Method is :add(int , int)
Cache Object Created
Searching the data in cache
add()
Storing the data in cache
Sum :30

Method Invoked With Value:add(10 , 20)
Signature of the Method is :add(int , int)
Searching the data in cache
Getting the data from cache
Sum :30

Method Invoked With Value:add(30 , 50)
Signature of the Method is :add(int , int)
Searching the data in cache
Getting the data from cache
add()
Sum :80
```

```java
public class CacheAdvice implements MethodInterceptor {

	@Override
	public Object invoke(MethodInvocation methodInvocation) throws Throwable {
		String methodName = null;
		String methodSignature=null;
		Object[] args = null;
		Method method=null;
		Map<String,Integer> methodMap=null;
		String key=null;
		Cache cache=null;
		Object ret = null;

		methodName=methodInvocation.getMethod().getName();
		args=methodInvocation.getArguments();
		method=methodInvocation.getMethod();

		key=buildKey(methodName, args);
		methodSignature=getMethodSignature(method);
		System.out.println("Method Invoked With Value:"+key);
		System.out.println("Signature of the Method is :"+methodSignature);
		cache=Cache.getInstance();

		/**
		 * Checking in cache whether the data is there or not?
		 * If it is there return the value
		 */
		if (cache.containsKey(methodSignature)==true) {
			methodMap=(Map<String, Integer>) cache.get(methodSignature);
			if (methodMap.containsKey(key)) {
				return methodMap.get(key);
			}
		}

		/**
		 * If it is not there call the target class method
		 */
		ret = (Integer)methodInvocation.proceed();
```

```
            /**
             *Store the data in cache and return.
             */
            if (methodMap==null) {
                methodMap=new HashMap<String,Integer>();
                methodMap.put(key, (Integer) ret);
                cache.put(methodSignature, methodMap);
            }
            return ret;
        }
        public String getMethodSignature(Method method){
            StringBuilder key=null;
            key=new StringBuilder();
            Class<?>[] clazz=method.getParameterTypes();
            key.append(method.getName()).append("(");
            for (int i = 0; i < clazz.length; i++) {
                if (i == 0) {
                    key.append(clazz[i]);
                    continue;
                }
                key.append(" , " + clazz[i]);
            }
            key.append(")");

            return key.toString();
        }
        public String buildKey(String methodName,Object[]args){
            StringBuilder key=null;
            key=new StringBuilder();
            key.append( methodName).append("(");
            for (int i = 0; i < args.length; i++) {
                if (i == 0) {
                    key.append(args[i]);
                    continue;
                }
                key.append(" , " + args[i]);
            }
            key.append(")");

            return key.toString();
        }
}
```

Let's See the Previous Example of BSE Stock Exchange how to apply AOP

```
public class BSE {
        public int getPrice(String stockName){
                int price=new Random().nextInt(999);
                return price;
        }
}
```

```
public class Stock {
        private  String stockName;
        private int stockPrice;
        private long lastAccessTime;
```

```
public class Cache {
        private static Cache instance;
        private Map<String,Object> dataMap;
        private Cache(){
                dataMap=new ConcurrentHashMap<String, Object>();
        }
        public static synchronized Cache getInstance(){}
        public  void put(String key,Object value){}
        public Object get(String key){}
        public  boolean containsKey(String key){}
}
```

Stock name

GetPrice

```
@WebServlet("/getPrice")
public class GetStockPriceServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;
        protected void service(HttpServletRequest request, HttpServletResponse response)
                                                throws ServletException, IOException {
                String stockName=request.getParameter("stockName");
                        ProxyFactory factory=new ProxyFactory();
                        factory.addAdvice(new CachingAdvice());
                        factory.setTarget(new BSE());
                        BSE proxy=(BSE) factory.getProxy();
                        int price=proxy.getPrice(stockName);
                        response.getWriter().print(price);
        }
}
```

OutPut

```
Checking from cache
Getting from cache
Invoking Target Class Method :getPrice
Storing stock in cache:Stock [stockName=cipla, stockPrice=299, lastAccessTime=1488324573290]
Store in cache
299
--------------------------------------------------------------------------------
Checking from cache
Getting from cache
Checks Time <2000 ms:true
Getting data is:299
--------------------------------------------------------------------------------
Checking from cache
Getting from cache
Checks Time <2000 ms:true
Getting data is:299
--------------------------------------------------------------------------------
Checking from cache
Getting from cache
Invoking Target Class Method :getPrice
Storing stock in cache:Stock [stockName=cipla, stockPrice=91, lastAccessTime=1488324575561]
Store in cache
91
--------------------------------------------------------------------------------
```

```
public class CachingAdvice implements MethodInterceptor {
	@SuppressWarnings("unchecked")
	@Override
	public Object invoke(MethodInvocation methodInvocation) throws Throwable {
		Cache cache = Cache.getInstance();
		Map<String, Object> stockMap = null;
		Stock stock = new Stock();
		Object[] args = methodInvocation.getArguments();
		String stockName = (String) args[0];
		Object ret = null;
		boolean flag = false;

		if (cache.containsKey("stocks")) {
			stockMap = (Map<String, Object>) cache.get("stocks");
			if (stockMap.containsKey(stockName)) {
				stock = (Stock) stockMap.get(stockName);

				if (Calendar.getInstance().getTimeInMillis()
						- stock.getLastAccessTime() <= 2000) {
					System.out.println("Checks Time <2000 ms:"
							+ (Calendar.getInstance().getTimeInMillis()
									- stock.getLastAccessTime() <= 20000));
					System.out.println("Getting data is:"
							+ stock.getStockPrice());
					System.out
							.println("----------------------------------------------------------------------");
					flag = true;
					return stock.getStockPrice();
				}
			}
		}

		System.out.println("Invoking Target Class Method :"
				+ methodInvocation.getMethod().getName());
		ret = methodInvocation.proceed();

		if (flag == false) {
			stockMap = new HashMap<String, Object>();
			stock.setStockName(stockName);
			stock.setStockPrice((int) ret);
			stock.setLastAccessTime(Calendar.getInstance().getTimeInMillis());
			System.out.println("Storing stock in cache:" + stock);
			stockMap.put(stockName, stock);
			cache.put("stocks", stockMap);
		}
		System.out.println(ret);
		System.out
				.println("----------------------------------------------------------------------");
		return ret;
	}
}
```

Internals of AroundAdvice

➔Whenever we create a Object of ProxyFactory pf=new ProxyFactory(); thenwe have to set the target class and add the advices then we can call pf.getProxy();

➔When we call pf.getProxy() then a proxy class Object will be created by extending from Target class at JVM memory in runtime using cglib,ASM libraries.

➔To apply the crosscutting logic we can all the information about the target class without target class information we cannot apply the crosscutting logic.

➔So invoke method has one parameter called MethodInvocation which able grab all the information of the target class.

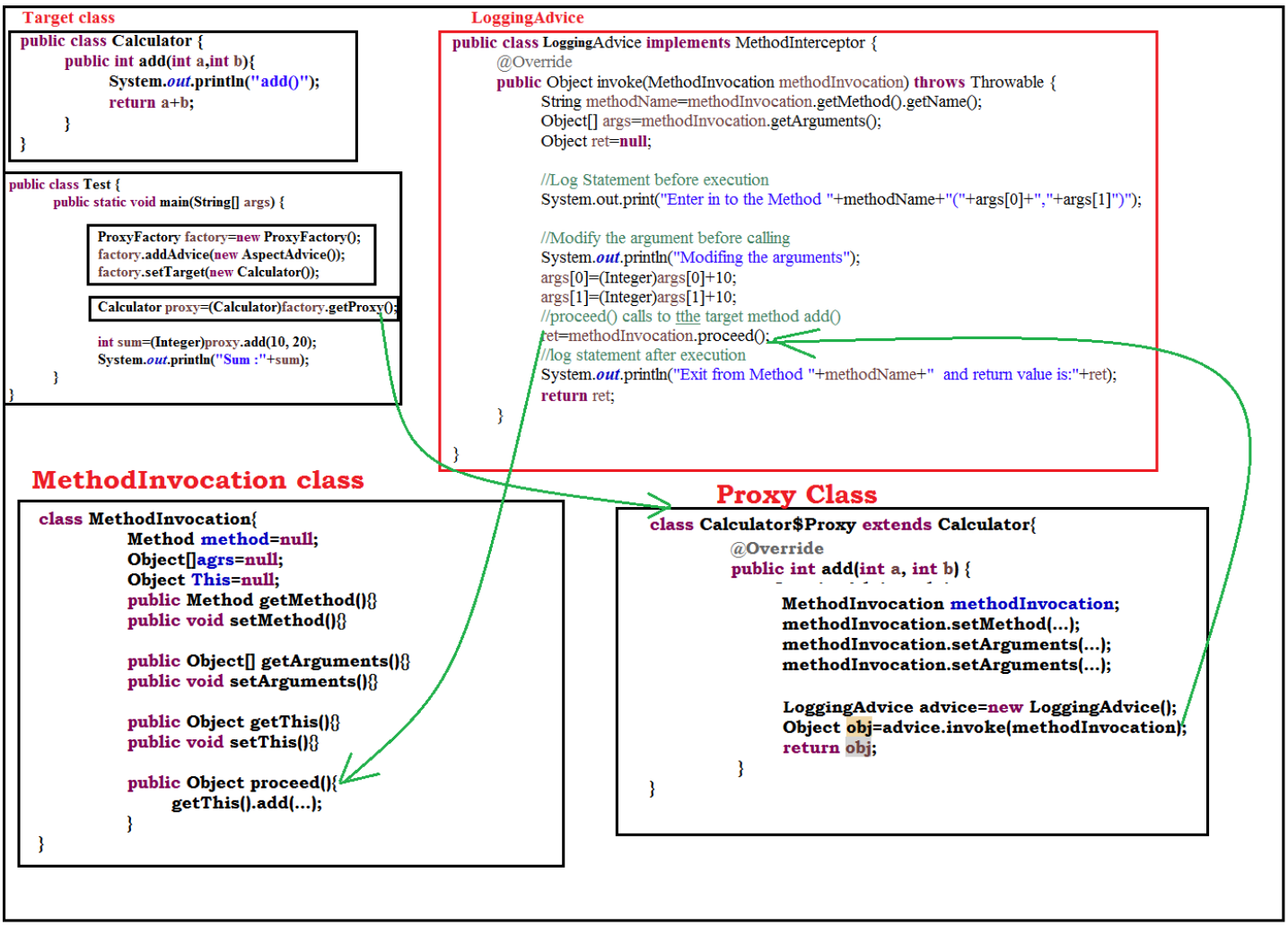
➔Means by help MethodInvocation can we get all information of the target class easily, to get all the information which methods we have to use by seeing the example we can get it.

➔Once we got the all information about the target class now we can perform the before advice and can proceed the execution of the target class by calling methodInvocation.proceed() method.

➔methodInvocation.proceed() method will call the target class corresponding method, once target method execution has been finished it will return the cursor to the invoke method then again invoke method can apply the crosscutting logic after the method execution.

➔Invoke method not only apply the crosscutting logic, it can modify the input data, even it can modify the return output also.

➔Because target class method called by invoke method and target method will return output to the invoke method and before returning the original output to the callee invoke method can modify it.



Let's see the another example of Around Advice related to Performance Monitoring

```
public class Engine {
        public int start(){
                return new Random().nextInt(2);
        }
}
```

```
public class PerformanceCheckAdvice implements MethodInterceptor {
        @Override
        public Object invoke(MethodInvocation methodInvocation) throws Throwable {
                String methodName = methodInvocation.getMethod().getName();
                long startTime = Calendar.getInstance().getTimeInMillis();
                Object ret = methodInvocation.proceed();
                long endTime = Calendar.getInstance().getTimeInMillis();
                System.out.println(methodName + "() takes " + (endTime - startTime)
                                + " milli seconds");
                return ret;
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                ProxyFactory pf=new ProxyFactory();
                pf.setTarget(new Engine());
                pf.addAdvice(new  PerformanceCheckAdvice());

                Engine engine=(Engine)pf.getProxy();
                int status=engine.start();
                if (status==1) {
                        System.out.println("Engine Started");
                }
                else{
                        System.out.println("Engine falis");
                }
        }
}
```

BeforeAdvice

➔Before Advice apply the aspects before the join point. It means before executing the target class method it will apply the aspects on join point.

Usecase-1 -Auditing

Let's see an example:

Primary Logic

```
public class LoanManager {
        public boolean approveLoan(int amount){
                        System.out.println("in approveLoan()");
                        return new Random().nextBoolean();
        }
}
```

SecondaryLogic/Aspect/Crosscutting Logic

```
public class AuditAdvice implements MethodBeforeAdvice{

        @Override
        public void before(Method method, Object[] args, Object target)throws Throwable {
                System.out.println("John called "+method.getName()+"() with
                                                arguments"+method.getName()+"("+args[0]+")");
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                ProxyFactory pf=new ProxyFactory();
                pf.setTarget(new LoanManager());
                pf.addAdvice(new AuditAdvice());
                LoanManager manager=(LoanManager)pf.getProxy();
                System.out.println(manager.approveLoan(100000));
        }
}
```

OutPut

```
John called approveLoan() with argumentsapproveLoan(100000)
in approveLoan()
true
```

➔My target class is LoanManager and My method is Public boolean approveLoan(long loanNumber) ,approveLoan() method will check the eligible criteria and either return true or false .

➔So we have the primary business logic I want to apply audit logic to my approveLoan() because I wanted to keep tracing who is accessing the method and who is approving the loan i want to know the information means i wanted to do adding.

➔So whenever someone call approveLoan() method wheatear they are able to approved or they are not able to approved I don't care but I want entering the auditing information saying that so and so user trying accessing the approve loan from the loan number so and so I wanted to print such adding information .

➔So do I need to write the audit information or do I need to write audit logic as part of my approveLoan() method or i shouldn't write? I shouldn't write.

➔ approveLoan() is your primary business logic Auditing is not your primary business logic it is a crosscutting logic.
➔Because There could be lot of crucial functionality that are there in your application from whom every access information has to be audited that means access information related to such functionality are access level auditing .
➔ We wanted to maintain access level auditing, Access levels means which critical areas are there who ever are trying to access we wanted to record that information in a log file or in a database table.
➔Now we can see that approveLoan() method is the joinpoint in target class which contains the primary business.
➔Now we can understand that approving of loan cannot be done by any one so we need auditing which will keep the records of the user who access the approveLoan() and who try to access it.
➔Now we can see Auditing logic is crosscutting logic which is to be used before the execution of approve() method so we need Before Advice.
➔Now when we call proxy.approve(), before calling the target class approveLoan() method control will goes to MethodBeforeAdvice before(Method method,Object[] args,Object target) method.
➔We can see the details of target class method by using "this" we can perform some auditing operation. Once the before operation finished control automatically goes to target approveLoan() method which will execute and returns the value to caller.
➔It will not return to before(—) because it will not called from there.
➔Target class method will be called automatically by AOP after finishing the before(—) operation.

If I want to apply logging and cashing around the target class method then can I combine both to one single advice or should not?

➔You can add any number of advice but in which order you added the advice the execution order only of advice are applied.
➔So when applying we should carefully apply which one is applied fist and which one added in last.
➔When we add login and cashing Advice at a time .always add cashing first because if we added cashing first target class method is not executed and data is retrieve the data from cache in this case also it is showing logging is happened .so always add cashing fist.
➔ If more number of advice there first evaluate which one will be apply in which order base on we need to advice.

Why MethodInvocation is not there in MethodBeforeAdvice?

➔After auditing has been completed. Once advice has completed execution automatically control goes to the target class method .Why?
➔Because i don't want to eliminate the execution of target class method we want primary business logic to be executed .
➔That means the before() method has completed execution and then before executing the target class method the control automatically comes to the target class method .
➔So we don't need to call methodinvocation.proceed () it automatically comes to execute the method.
➔If you want to apply cross cutting logic before method execution only taking a MethodIntercepter who makes you even writes the logic for handling the things unnecessarily after which is a headache.

➔Means methodInvokaction.procced () getting the return value returning the return value means rather than what you want you are doing additionally.
➔So there is a chance if you forget to write methodInvokaction.procced () target never executed.
➔So instead of doing to a MethodIntercepter as you have only the crosscutting logic apply before target class tell to AOP MethodBeforeAdvice.

Why before () method is void?

➔As we execute only method before execution we will not get a chance to see the return value and modify the return value.
➔Only we can get a chance method before execution we can throw an exception.
➔**So we have one and half control point we cannot stop the method execution we can't see the return value because before method execution we are doing the operation that's why return type to before method is void.**

Internals

➔Whenever I called proxyFactory.grtProxy (), then a runtime proxy will be generated using the Byte code generation libraries and the name of the class is TargetClass$Proxy extends from Target class and overrides the target class method.
➔It is creating a proxy class extending form target class because it has to add some additional functionalities to the target class method without modifying the method this only can possibly in case of overriding.
➔And inside that method AOP will add the information about Target class method, arguments, and target class.
➔And pass those values as parameter to before (--,--,--)
➔In Before Method Advice we have 1 ½ control point.

Let's See the perfect example where we use MethodBeforeAdvice ?

```
public class LoanManager {
        public boolean approveLoan(int amount){
                System.out.println("in approveLoan()");
                return new Random().nextBoolean();
        }
}
```

```
public class Credential {
        private String user;
        private String password;
        public Credential(String user, String password) {
                this.user = user;
                this.password = password;
        }
        //setters & getters
```

```
public class Test {
        public static void main(String[] args) {
                ProxyFactory pf=new ProxyFactory();
                pf.setTarget(new LoanManager());
                pf.addAdvice(new SecurityCheckAdvice());
                pf.addAdvice(new AuditAdvice());

                LoanManager manager=(LoanManager)pf.getProxy();
                SecurityHolder sHolder=SecurityHolder.getInstance();
                sHolder.login("Dhananjaya", "realspeed");


                System.out.println(manager.approveLoan(100000));

                sHolder.logout();
        }
}
```

```
public class SecurityCheckAdvice implements MethodBeforeAdvice {

        @Override
        public void before(Method method, Object[] args, Object target)
                        throws Throwable {
                String methodName=method.getName();
                SecurityHolder sHolder=SecurityHolder.getInstance();
                boolean flag=sHolder.authenticate();

                /**
                 *
                 * if authentication fails then Throws IllegalAccessException("Invalid user/password")
                 */
                if (flag==false) {
                        throw new IllegalAccessException("Invalid user/password");
                }

        }
}
```

```
public class AuditAdvice implements MethodBeforeAdvice {

        @Override
        public void before(Method method, Object[] args, Object target)
                        throws Throwable {
                SecurityHolder sHolder = SecurityHolder.getInstance();

                System.out.println(sHolder.getUserDetails() + " called "
                                + method.getName() + "() with arguments  " +
                                                                method.getName()
                                + "(" + args[0] + ")");
        }
}
```

```
public class SecurityHolder {
	private static SecurityHolder instance;
	private ThreadLocal<Credential> userCred;
	private Credential userDetails;
	private SecurityHolder() {
		userCred=new ThreadLocal<Credential>();
	}

	public static synchronized SecurityHolder getInstance(){
		if (instance==null) {
			instance=new SecurityHolder();
		}
		return instance;
	}

	/**
	 * Storing user & password in localThread
	 * @param user
	 * @param password
	 */
	public void login(String user,String password){
		userDetails=new Credential(user,password);
		userCred.set(userDetails);
	}

	/**
	 * remove values from localThread
	 */
	public void logout(){
		userCred.set(null);
	}

	/**
	 * Checks whether the credentials are valid or not
	 * @return
	 */
	public boolean authenticate(){
		if (userCred!=null) {
			if (userCred.get().getUser().equals("Dhananjaya")
					&& userCred.get().getPassword().equals("realspeed")) {
				return true;
			}
		}
		return false;
	}

	/**
	 * getting the current user details
	 * @return
	 */
	public String getUserDetails(){
		if (userCred!=null) {
			return userCred.get().getUser();
		}
		return null;
	}
}
```

AfterReturnning Advice

➔In this advice method will be executed only after the target class method finishes the execution and before it returns the value to the caller.

➔This Indicates that the target method has almost returned the value but just before returning the value it allows the advice method to see the return value but doesn't allow to modify it.

➔In order to create and after returning advice ,after building the target class , you need to write the aspect class implementing from the AfterReturningAdvice and should override the method afterReturning , the parameters to this method are Object returnValue()(returned by actual method),java.lang.reflect.Method method (Origional method),Object[]args(target method arguments),Object target(with which the target method being called)

➔The Return value of afterReturning method is void , which means you can't modify and return the value.

ControlPoint

➔We can see the parameter of the target method , even we modify there is use, because by the time the advice method is called the target method finished execution , so there is no effect of changing the parameter values.

➔We cannot control the target method execution as the control will enters into advice method only after the target method completes.

➔We can see the return value being returned by the target method, but you can't modify it, as the target method has almost returned the value.

➔But we can stop/abort the return value by throwing exception in the advice method.

Example-1

```
public class KeyGenerator {
        public int generateKey(int key){
                if (key<8) {
                        return 0;
                }
                return 1;
        }
}
```

```
public class CheckKeyAdvice implements AfterReturningAdvice {
        @Override
        public void afterReturning(Object ret, Method method, Object[] args,
                        Object target) throws Throwable {
                if ((Integer)ret<=0) {
                        throw new IllegalArgumentException("Invalid Key/Length not supported");
                }
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                ProxyFactory pf=new ProxyFactory();
                pf.setTarget(new  KeyGenerator());
                pf.addAdvice(new CheckKeyAdvice());

                KeyGenerator proxy=(KeyGenerator) pf.getProxy();
                int key=proxy.generateKey(8);
                System.out.println(" key : "+key);

                key=proxy.generateKey(6);
                System.out.println(" key : "+key);
        }
}
```

➔In this case control comes to the afterReturning () when target class execute normally.
➔If target class throw any exception then control does not comes to afterReturning().
➔After executing afterReturning () method of VerifyAdvice class return value will automatically goes to callee.
➔In this case we can abrupt the execution of target class by throwing an exception in afterReturning () method.
➔In After returning advice we have ½ control.

Example-2

```
public class RewardCard {
        private String rewardCardNo;
        private String memberName;
        private String cardType;

        //Setters & Getters
```

```
public class Billing {
        public double doBilling(List<String>products,RewardCard rewardCard){
                return 5679.00;
        }
}
```

```
public class BillingAdvice implements AfterReturningAdvice{
        @Override
        public void afterReturning(Object ret, Method method, Object[] args,
                        Object target) throws Throwable {
                RewardCard rewardCard=(RewardCard) args[1];
                double billAmount=(double) ret;
                if (rewardCard.getCardType().equals("gold") && billAmount>=3000) {
                        System.out.println("Congratulation!!! you will get 40% on next purchase and your Coupon No:FLAT40");
                }
                if (rewardCard.getCardType().equals("silver") && billAmount>=4000) {
                        System.out.println("Congratulation!!! you will get 30% on next purchase and your Coupon No:FLAT30");
                }
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                ProxyFactory pf=new ProxyFactory();
                pf.setTarget(new Billing());
                pf.addAdvice(new BillingAdvice());

                List products=new ArrayList();
                products.add("SAMSUNG");
                products.add("ONIDA");
                products.add("NOKIA");

                RewardCard rewardCard=new RewardCard();
                rewardCard.setCardType("silver");
                rewardCard.setMemberName("Dhananjaya");
                rewardCard.setRewardCardNo("1223 8756 0096 4461");

                Billing proxy=(Billing) pf.getProxy();
                double billAmount=proxy.doBilling(products, rewardCard);
                System.out.println(billAmount);
        }
}
```

Throws Advice

➔This advice method will be invoked only when the target class method throws an exception.
➔The main purpose of the throws advice is to centralize the error handling mechanism or some kind of central processing whenever a class throws exception.
➔Throws Advice will be called only when the target throws exception rather it has a chance of seeing the exception and do further processing based on the exception.
➔Once the throws advice method finishes execution the control will flow through the normal exception handling hierarchy till a catch handler is available to catch it.

```
public class Thrower {
	public int willThrow(int i){
		if (i<0) {
			throw new IllegalArgumentException("Invalid Parameter");
		}
		return i+new Random().nextInt(10000);
	}
}
```

```
public class ExceptionLoggerAdvice implements ThrowsAdvice{

	public void afterThrowing(Method method,Object[]args,Object target,IllegalArgumentException e){
		e.printStackTrace();
	}
	public void afterThrowing(Exception e){
		System.out.println(e.getMessage());
	}
}
```

```
public class Test {
	public static void main(String[] args) {
		ProxyFactory pf=new ProxyFactory();
		pf.setTarget(new Thrower());
		pf.addAdvice(new ExceptionLoggerAdvice());
		Thrower proxy=(Thrower) pf.getProxy();
		int i=(Integer)proxy.willThrow(10);
		System.out.println(i);
		i=(Integer)proxy.willThrow(-1);
		System.out.println(i);
	}
}
```

➔In order to work with throws advice, after building the target class, we need to write a class implementing from ThrowsAdvice Interface.
➔ThrowsAdvice is a Marker Interface that means it doesn't define any method in it.
➔We need to write method to monitor the exception raised by the target class.
➔ The method signature should public and void with name afterThrowing.
➔It takes the arguments as exception class type indicating which type of exception you are interested in monitoring.
➔The word monitoring is used here because we are not catching the exception, but we are just seeing and propagating it to the top hierarchies.
➔When a target class throws exception spring IOC container will tries to find a method with name afterThrowing in the advice class, having the appropriate argument representing the type of execution class.
➔There are two method signatures of afterThrowing.

```
public void afterThrowing(Method method,Object[]args,Object target,Exception e){
}
```

```
public void afterThrowing(Exception e){
}
```

➔In the above Signature Method,args and target is optional and only mandatory parameter is subclass of throwable.
➔If advice class contain afterThrowing method with both the signatures handling the same subclass of throwable , then we can't say which method will be executed because it changes version to version.

Control Points

➔It can see the parameters that are passed to the original method.
➔There is no point in controlling the target method execution, as it would be invoked when the target method raises exception.
➔When a method throws exception, it cannot return a value, so there is nothing like seeing the return value or modifying the return value.
➔But we can change the Exception from one Exception to another Exception.
➔So in ThrowsAdvice we have $\frac{1}{2}$ control point.

PointCut

➔In all the above Example , we have not specified any pointcut while advising the aspect on a target class.
➔This means the advice will be applied on all the joinpoints of the target class.
➔With this all the methods of the target class will be advised , so if you want to skip the execution of the advice logic on a specific method of a target class, In the advice logic we can check the method name on which the advice is being called and base on it we can execute logic.
➔But the problem is even you don't want to execute advice logic for some methods, still the call to these methods will delegates to advice due to this your application will get a performance impact.

➔In order to overcome this you need to attach a pointcut while performing the weaving , so that the proxy will be generate base on the pointcut specified and do some optimization in generating proxies.
➔With this if you call a method that is not specified in the pointcut ,spring IOC container will makes a direct call to the method rather than calling a advice for it.
➔Spring AOP supports two types of pointcuts.
➔Static Pointcut ➔Dynamic Pointcut
➔All the pointcuts implementing from Pointcut interface .
➔Spring has provided in-built implementation classes from this interface.
➔StaticMethodMatcherPointcut ➔JDKRegexpMethodpointcut
➔In order to use Static MethodMatcherPointcut, we need to write a class extending from StaticMethodMatcherPointcut and need to override the method matches().
➔The matches() takes arguments as Method,and Class as arguments.
➔Spring while performing the weaving process, it will determines whether to attach an advice on a method of a target class by calling matches method on the pointcut class we suppilied.
➔While calling the method it will passes the current class and method it is checking for.
➔If the matches() method returns true it will advise that method , otherwise will skip advicing.

Static Pointcut

➔In order to use Static method matcher pointcut, we need to write a class extending from StaticMethodMatcherPointcut and needs to override the method matches. The matches() method takes arguments as Class classType and Method method as arguments.
➔Spring while performing the weaving process,It will determine whether to attach an advice on a method of target class by calling matches method on the pointcut class we suppilied.
➔While calling the method it will passes the current class and method it is checking for ,If the matches method return true then it will advice that method , otherwise will skip advising

```
public class Test {
    public static void main(String[] args) {          [Weaving]
        ProxyFactory factory = new ProxyFactory();
        factory.setTarget(new Calculator());
        Advisor loggingAdvisor = new DefaultPointcutAdvisor(
                new LoggingPointcut(), new LoggingAdvice());

        factory.addAdvisor(loggingAdvisor);

        Calculator proxy = (Calculator) factory.getProxy();
        int sum = (Integer) proxy.add(10, 20);
        System.out.println("Sum :" + sum);

        int mul = (Integer) proxy.multiply(10, 20);
        System.out.println("mul:" + mul);

        int sub = (Integer) proxy.substract(10, 20);
        System.out.println("sub :" + sub);
    }
}
```

```
public class Calculator {
    public int add(int a,int b){
        System.out.println("add()");
        return a+b;
    }
    public int multiply(int a,int b){
        System.out.println("multiply()");
        return a*b;
    }
    public int substract(int a,int b){
        System.out.println("substract()");
        return b-a;
    }
}                    [Target Class]
```

```
public class LoggingPointcut extends StaticMethodMatcherPointcut {
    @Override
    public boolean matches(Method method, Class<?> classType) {
        if (classType == Calculator.class && method.getName().equals("add")
                || method.getName().equals("multiply")) {
            return true;
        }
        return false;
    }
}                                                    [PointCut]
```

```
public class LoggingAdvice implements MethodInterceptor {
    @Override
    public Object invoke(MethodInvocation methodInvocation) throws Throwable {
        String methodName = methodInvocation.getMethod().getName();
        Object[] args = methodInvocation.getArguments();
        Object ret = null;

        // Log Statement before execution

        System.out.print("Enter in to the Method " + methodName + "(");
        for (int i = 0; i < args.length; i++) {
            if (i == 0) {
                System.out.print(args[i]);
                continue;
            }
            System.out.print(", " + args[i]);
        }
        System.out.println(")");

        // proceed() calls to tthe target method add()
        ret = methodInvocation.proceed();

        // log statement after execution
        System.out.println("Exit from Method " + methodName
                + "  and return value is:" + ret);

        return ret;
    }
}                                                    [Aspect Advice]
```

➔While performing the weaving, we need to supply the pointcut as input to the ProxyFactory

[Target Class]

```
public class Calculator {
        public int add(int a,int b){
                System.out.println("add()");
                return a+b;
        }
        public int multiply(int a,int b){
                System.out.println("multiply()");
                return a*b;
        }
        public int substract(int a,int b){
                System.out.println("substract()");
                return b-a;
        }
}
```

[Weaving]

```
public class Test {
        public static void main(String[] args) {
                ProxyFactory factory = new ProxyFactory();
                factory.setTarget(new Calculator());
                Advisor loggingAdvisor = new DefaultPointcutAdvisor(
                                new LoggingPointcut(), new LoggingAdvice());
                factory.addAdvisor(loggingAdvisor);

                Calculator proxy = (Calculator) factory.getProxy();
                int sum = (Integer) proxy.add(10, 20);
                System.out.println("Sum :" + sum);

                int mul = (Integer) proxy.multiply(10, 20);
                System.out.println("mul:" + mul);

                int sub = (Integer) proxy.substract(10, 20);
                System.out.println("sub :" + sub);
        }
}
```

[Advisor]

[Advice]

```
public class LoggingAdvice implements MethodInterceptor {
        @Override
        public Object invoke(MethodInvocation methodInvocation) throws Throwable {
                String methodName = methodInvocation.getMethod().getName();
                Object[] args = methodInvocation.getArguments();
                Object ret = null;
                // Log Statement before execution
                System.out.print("Enter in to the Method " + methodName + "("+args[0]+","+args[1]+")
                // proceed() calls to tthe target method add()
                ret = methodInvocation.proceed();
                // log statement after execution
                System.out.println("Exit from Method " + methodName+ "  and return value is:" + ret);
                return ret;
        }
}
```

[Pointcut]

```
public class LoggingPointcut extends StaticMethodMatcherPointcut {
        @Override
        public boolean matches(Method method, Class<?> classType) {
                if (classType == Calculator.class && method.getName().equals("add")
                                || method.getName().equals("multiply")) {
                        return true;
                }
                return false;
        }

}
```

[Proxy]

```
public class CalculatorSProxy17b64 {
        public int add(int a,int b){
                aspect=new LoggingAspect()
               ret=aspect.invoke(...)
        }
        public int multiply(int a,int b){
                aspect=new LoggingAspect()
               ret=aspect.invoke(...)
        }
}
```

Example: We are no applying caching logic on every method rather on some specific method.

Note:

➔In case of StaticMethodMatcherPointcut only the specific method will be overridden by proxy which are return true value in conditional check.

➔Pointcut will be called depends on how many methods are there in target class including hashcode and equals method from object class. Means extra 2 time more it will be called.

Dynamic PointCut

➔If you observe the static pointcut, we can have hard coded the class and method names on which we need to advice the aspect, so the decision of advising the aspect on a target would be done at the time of weaving and would not be delayed till the method call. In case of dynamic PointCut, the decision whether to attach an aspect on a target class would be made based on the parameter with which the target class method has been called.

[Target Class]

```
public class Calculator {
        public int add(int a,int b){
                System.out.println("add()");
                return a+b;
        }
        public int multiply(int a,int b){
                System.out.println("multiply()");
                return a*b;
        }
        public int substract(int a,int b){
                System.out.println("substract()");
                return b-a;
        }
}
```

[proxy]

```
public class CalculatorSProxy745b34 {
        public int add(int a,int b){
                LoggingPointcut pointcut=new LoggingPointcut();
                pointcut.matches(....., ....)
                return a+b;
        }
        public int multiply(int a,int b){
                LoggingPointcut pointcut=new LoggingPointcut();
                pointcut.matches(....., ....)
                return a*b;
        }
        public int substract(int a,int b){
                LoggingPointcut pointcut=new LoggingPointcut();
                pointcut.matches(....., ....)
                return b-a;
        }
}
```

Note:

➔In case of DynamicMethodMatcherPointcut all the methods of target class will be overridden by Proxy class , and every time at runtime it will call to pointcut for checks the conditions of point cut.

[Weaving]

```
public class Test {
        public static void main(String[] args) {
                ProxyFactory factory = new ProxyFactory();
                factory.setTarget(new Calculator());
                Advisor loggingAdvisor = new DefaultPointcutAdvisor(
                                new LoggingPointcut(), new LoggingAdvice());

                factory.addAdvisor(loggingAdvisor);

                Calculator proxy = (Calculator) factory.getProxy();
                int sum = (Integer) proxy.add(10, 20);
                System.out.println("Sum :" + sum);

                int mul = (Integer) proxy.multiply(10, 20);
                System.out.println("mul:" + mul);

                int sub = (Integer) proxy.substract(10, 20);
                System.out.println("sub :" + sub);
        }
}
```

[Advisor]

[Advice]

```
public class LoggingAdvice implements MethodInterceptor {
        @Override
        public Object invoke(MethodInvocation methodInvocation) throws Throwable {
                String methodName = methodInvocation.getMethod().getName();
                Object[] args = methodInvocation.getArguments();
                Object ret = null;
                // Log Statement before execution
                System.out.print("Enter in to the Method " + methodName + "("+args[0]+","+args[1]+")
                // proceed() calls to tthe target method add()
                ret = methodInvocation.proceed();
                // log statement after execution
                System.out.println("Exit from Method " + methodName+ "  and return value is:" + ret);
                return ret;
        }
}
```

[Pointcut]

```
public class LoggingPointcut extends DynamicMethodMatcherPointcut{
        @Override
        public boolean matches(Method method, Class<?> classType, Object[] args) {
                if (classType == Calculator.class)

                        if(method.getName().equals("add")&& (Integer)args[0]>50 &&
                                (Integer)args[1]>70 || method.getName().equals("multiply")) {
                        return true;
                }
                return false;
        }
}
```

Summary of all the Advice Type

AdviceType /Control Poing	When Advice Method Executes	Access to Target Method Parameter	Control of method execution	Access to return value
Around Advice	Around the Target class Method	Can See and Modify	Yes	Can see and modify the return value
Method Before Advice	Before Target Method executes	Can See and Modify	Don't have contorl and target method execution, only can abort target method by throwing an Exception	No Applicable (Control will not come back to advice method after finishes the target method execution)
After Returning Advice	After Target Method Finished execution, but before returning value to called	can see the parameters	Not applicable	Can see the return value, but cannot modify them,Can abort returning the control value by throwing an Exception
ThrowsAdvice	Only when the target method throws Exception	Can see parameters	Not applicable	Not applicable

Declarative AOP

➔Spring 2.X has added support to declarative AspectJ AOP.
➔The main problem with programmatic approach is your application code will tightly couple with spring so that there is no non-invasiveness of spring, we can't detach from spring.
➔If we use Declarative Approach our Advice/Aspect classes are still pojo's, we don't need to implement or extend from any spring specific interface or class.
➔Our Advice/Aspect classes use AspectJ AOP API classes.
➔In order to declare the pojo as aspect we need to declare it in the spring bean configuration file rather than weaving it using programmatic approach.
➔The declarative approach supports all the four types of advices and static pointcut.

Q.Why spring integrated with AspectJ?
➔By Using spring AOP,the application is losing Non-invasiveness.
➔So by avoiding the non-invasiveness spring people integrated Aspectj AOP with spring framework.
➔Instated of using spring classes in implementing the AOP people stated using the Aspectj classes as part of spring work with Aop.
➔So that our application will become non-invasive.
When pointcut called:

➔Threre are 2-types of pointcut.
    ➔Static Pointcut
    ➔Dynamic Pointcut

Q.What is the different between static and dynamic pointcut?
➔In case of static pointcut at the time of proxy itself the join point will be advice list of the pointcut.
➔Every method of join point will not be advice as part of the proxy. So proxy creation will be optimizing.
➔But when it comes to the dynamic Pointcut the pointcut expression can be evaluated only at the runtime weather the method has to be advice or not.
➔Advice cannot decide during the creation of proxy only at the run time doing execution of my target class method only I will know whether aspect is advice or not because the pointcut can't be evaluated the proxy generation time is only at the run time so optimize not happen and it has impacted as a performance problem.

Q. How many times the static pointcut matches method will be called?
➔Depends on the number of method are present the target class to create the proxy.
➔Once the proxy has created only the number of method applied that method has been called and also 2 times extra for equals and hashcode.

Around Advice

➔In this approach the aspect class is not required to implements from any spring specific class of interface.
➔Rather it should be declared as aspect in spring bean configuration file.
➔But in Aspect class we need to follow some signature of the method, method name can be any name but return type must be Object and parameter should be ProceedingJoinPoint like as follows.

public Object log(ProceedingJoinPoint pjp)

Rules to Follow

➔This act as an advice method,the Return type of the method should be Object.

➔As the Around advice has control over the return value, The method name could be anything but should take the parameter as ProceedingJoinPoint .

➔We can control the execution of the joinpoint in Around advice , so the parameter for this method should be ProceedingJoinPoint.

➔Using the ProceedingJoinPoint we can access the actual information similar to method interceptor.

```
public class Test {
	public static void main(String[] args) {
		ApplicationContext context=
			new ClassPathXmlApplicationContext("com/aa/common/application-context.xml");
		Calculator proxy=(Calculator)context.getBean("calculator",Calculator.class);
		System.out.println(proxy.getClass().getName());
		int sum=(Integer)proxy.add(10, 20);
		System.out.println("Sum :"+sum);
	}
}
```

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

	<bean id="calculator" class="com.aa.beans.Calculator"/>
	<bean id="loggingAspect" class="com.aa.beans.LoggingAspect"/>
	<aop:config>
		<aop:aspect ref="loggingAspect">
			<aop:around method="log" pointcut="execution(int com.aa.beans.Calculator.add(int,int))"/>
		</aop:aspect>
	</aop:config>
```

```
public class LoggingAspect {
	public Object log(ProceedingJoinPoint pjp) throws Throwable {
		String methodName=pjp.getSignature().getName();
		Object[] args=pjp.getArgs();
		Object ret=null;
		//Log Statement before execution
		System.out.print("Enter in to the Method "+methodName+"(");
		for (int i = 0; i < args.length; i++) {
			if (i==0) {
				System.out.print(args[i]);
				continue;
			}
			System.out.print(", "+args[i]);
		}
		System.out.println(")");
		//proceed() calls to the target method add()
		ret=pjp.proceed();
		//log statement after execution
		System.out.println("Exit from Method "+methodName+"  and return value is:"+ret);
		return ret;
	}
}
```

➔If you see the above class , we have not implemented or extended from any interface or class.
➔So in order to make this as aspect class we need to this in spring bean configuration file.

Steps:

➔First you need to declare our target and aspect classes as bean.
➔While declare a class as a aspect you need to import the 'aop' namespace and need to use the tag **<aop:config>**.
➔Inside the <aop:config> , you need to declare the bean as aspect using <aop:aspect ref="">.
➔Under this you need to declare which method you need to expose as advice method and how do you want to apply this method like aroundAdvice/beforeAdvice/afterReturning etc..
➔Once this is done you need to supply pointcut indicating on which target class method you want to apply the aspect.
➔For this we need to follow pointcut expression rather referring to a class representing as pointcut
➔ This pointcut expression is a static pointcut expression. And it has been written OGNL(Object Graph Navigation Language) expression language.
➔Expression starts with execution() as a word representing apply the advice to all the class execution.

```
<aop:around method="log" pointcut="execution(int com.aa.beans.Calculator.add(int,int))"/>

<aop:around method="log" pointcut="execution(* com.aa.beans.Calculator.add(int,int))"/>

<aop:around method="log" pointcut="execution(* com.aa.beans.Calculator.add(..))"/>

<aop:around method="log" pointcut="execution(* com.aa.beans.Calculator.*(..))"/>

<aop:around method="log" pointcut="execution(* com.aa.beans.*.*(..))"/>

<aop:around method="log" pointcut="execution(* *.*.*.*.*.*(..))"/>
```

Internals

➔When we create ApplicationContext ,while creating Spring IOC container will apply the advice on a target class (based on the pointcut).
➔In the above "calculator" Bean object will be created on proxy after applying the advice,
➔So when we request for **Calculator proxy=context.getBean("calculator",Calculator.class); instead of returning original Calculator class object ,IOC Container instantiate the proxy of Calculator class as the pointcut matching to it and returns that object to us** (com.aa.beans.Calculator$$EnhancerByCGLIB$$dfee1604**).**
➔It creates a proxy classes and instantiate these classes and host in IOC container at runtime using CGLIB runtime byte code generator libraries.
➔So when we call target class it will return Proxy class, it never return target class reference.
➔Inside Aapect class we can see and can modify the but it will not effect unless until we pass it as argument to proceed method.
➔Actually spring maintains a cloned copy and pass it through ProceedingJoinPoint ,If we want apply modification original copy of the target class then we have to pass modified arguments to the proceed method.

```java
public class LoggingAspect {
	public Object log(ProceedingJoinPoint pjp) throws Throwable {
		String methodName=pjp.getSignature().getName();
		Object[] args=pjp.getArgs();
		Object ret=null;

		//Log Statement before execution
		System.out.print("Enter in to the Method "+methodName+"("+args[0]+","+args[1]+")");

		//Modifing arguments
		args[0]=(Integer)args[0]+100;
		args[1]=(Integer)args[1]+100;
		ret=pjp.proceed(args);

		//log statement after execution
		System.out.println("Exit from Method "+methodName+"  and return value is:"+ret);

		ret=(Integer)ret+10000;
		return ret;
	}
}
```

```java
public class Test {
	public static void main(String[] args) {
		ApplicationContext context=new
				ClassPathXmlApplicationContext("com/aa/common/application-context.xml");
		Calculator proxy=context.getBean("calculator",Calculator.class);
		System.out.println(proxy.getClass().getName());
		int sum=(Integer)proxy.add(10, 20);
		System.out.println("Sum :"+sum);
	}
}
```

➔When ever I said context.getBean ("calcularor", Calculator.class) will you get the object of Calculator class or not?

➔No because it provides proxy class object. How it will treat proxy class object lets stared understanding.

➔When we are trying to creating IOC container by using ApplicationContext the classpathXmlApplicationContext directly goto the classpath of the project.

➔And Looks under the classpath within the specific directory location the application-context.xml and read the xml and check for well formness and validity .

➔If the xml is well formness and validity it creates ICO container reads the physical configuration file and store it as a In-memory metadata of IOC container .

➔It is not returning the reference to the ioc container as a context because ApplicationContext is a eagerInitializer at the time of creating the ioc contener Object of bean definition has been instantiated .

➔That means once the IOC container has been created once the physical configuration file has been loaded and has been placed as an in memory metadata then it goes to the spring bean configuration file .then it start reading bean definitions.

➔First it reading Calculator second one is LoggingAspect .

➔It create the object of calculator and create the object of LoggingAspect.

➔Always create the target class object even create the proxy class object also because proxy holds the reference of real object.

➔Both the object has been instantiated then it goes to <aop:config>

➔Then IOC understand the user provided AOP related information to me so he wanted to invoke advice the aspect on the joinpoint and wanted to create the proxy.

So what does the ioc contener do ?

➔It Goes to the aop configuration looks for the aspect i.e loggingAspectand the object already with him and looks the method ,log() method with around advice and evaluate the pointcut expression and find out the class of Calculator class all the method

➔Now it evaluate the pointcut expression to all the method of the Calculator class, so it quickly goes to the calculator bean and creat Claculator$proxy with override all the method of Calculator class.and apply log() method of the advice .

➔Now it creates the object of Claculator$proxy and replace the calculator original object into proxy i.eMeans the Ioc contener contens Claculator$proxy object rather than Calculator object because we want proxy class object.

➔So when we are writing Calculator calculator=context.getBean ("calcularor", Calculator.class) .Ioc container providing proxy class object. So we are calling proxy object with add(10,20) so the control goes to proxy class and perform the operation.

If i don't want to proxy class object and i want to target class object what i need to do?

Go to application-context.xml and comments the <aop:config >tag .proxy will not be created.

Before Advice

➔While working with before advice the entire approach is same like creating the aspect class and should declare a method.

➔Here the method should be public and the return type should be void ,As the before advice cannot control the return value.

➔The parameter to the advice method is JoinPoint rather than ProceedingJoinPoint as we don't have the control on the method execution.

```
public class Test {
	public static void main(String[] args) {
		ApplicationContext context=
				new ClassPathXmlApplicationContext("com/ba/common/application-context.xml");
		LoanManager proxy=context.getBean("loanManager",LoanManager.class);
		System.out.println("Loan Status:"+proxy.approveLoan(100000));
	}
}
```

```
public class LoggingAspect{
	public void log(JoinPoint jp) throws IllegalAccessException{
		String methodName=jp.getSignature().getName();
		Object[]args=jp.getArgs();
		System.out.println("Dhananjaya is  Accessing "+methodName+"("+args[0]+")");
	}
}
```

```
<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/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
		http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
	<bean id="loanManager" class="com.ba.beans.LoanManager"/>
	<bean id="loggingAspect" class="com.ba.beans.LoggingAspect" />

	<aop:config>
		<aop:aspect ref="loggingAspect">
			<aop:before method="log" pointcut="execution(public * com.ba.beans.LoanManager.approveLoan(..))"/>
		</aop:aspect>
	</aop:config>

</beans>
```

Q.How to configure order in spring bean configuration file if we have multiple Aspects ?

```
public class LoanManager {
        public boolean approveLoan(int amount){
                        System.out.println("in approveLoan()");
                        return new Random().nextBoolean();
        }
}
```

```
public class SecurityHolder {
        private static SecurityHolder instance;
        private ThreadLocal<Credential> userCred;
        private Credential userDetails;
        private SecurityHolder() {
                userCred=new ThreadLocal<Credential>();
        }

        public static synchronized SecurityHolder getInstance(){
                if (instance==null) {
                        instance=new SecurityHolder();
                }
                return instance;
        }
        /**
         * Storing user & password in localThread
         * @param user
         * @param password
         */
        public void login(String user,String password){
                userDetails=new Credential(user,password);
                userCred.set(userDetails);
        }
        /**
         * remove values from localThread
         */
        public void logout(){
                userCred.set(null);
        }
        /**
         * Checks whether the credentials are valid or not
         * @return
         */
        public boolean authenticate(){
                if (userCred!=null) {
                        if (userCred.get().getUser().equals("Dhananjaya")
                                && userCred.get().getPassword().equals("realspeed")) {
                                return true;
                        }
                }
                return false;
        }
        /**
         * getting the current user details
         * @return
         */
        public String getUserDetails(){
                if (userCred!=null) {
                        return userCred.get().getUser();
                }
                return null;
        }
}
```

```
import org.aspectj.lang.JoinPoint;
public class AuditAspect{
        private SecurityHolder sHolder;
        public void setsHolder(SecurityHolder sHolder) {
                this.sHolder = sHolder;
        }
        public void log(JoinPoint jp) throws IllegalAccessException{
                String methodName=jp.getSignature().getName();
                Object[]args=jp.getArgs();
                System.out.print(sHolder.getUserDetails()+"
                                is  Accessing "+methodName+"(");
                for (int i = 0; i < args.length; i++) {
                        if (i==0) {
                                System.out.print(args[i]);
                                continue;
                        }
                        System.out.print(","+args[i]);
                }
                System.out.println(")");
        }
}
```

```
import org.aspectj.lang.JoinPoint;
public class SecurityCheckAspect{
        private SecurityHolder sHolder;
        public void setsHolder(SecurityHolder sHolder) {
                this.sHolder = sHolder;
        }
        public void check(JoinPoint jp) throws IllegalAccessException{
                boolean flag=sHolder.authenticate();
                if (flag==false) {
                        System.out.println("Authentication Fails");
                        throw new IllegalAccessException("Invalid User/Password");
                }
                System.out.println("Authentication Success");
        }
}
```

```
public class Credential {
        private String user;
        private String password;
        public Credential(String user, String password) {
                this.user = user;
                this.password = password;
        }
        //setters & getters
        public String getUser() {
                return user;
        }
        public void setUser(String user) {
                this.user = user;
        }
        public String getPassword() {
                return password;
        }
        public void setPassword(String password) {
                this.password = password;
        }
}
```

```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/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
		http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
	<bean id="loanManager" class="com.ba.beans.LoanManager"/>
	<bean id="sHolder"  class="com.ba.beans.SecurityHolder"  factory-method="getInstance" />

	<bean id="auditAspect" class="com.ba.beans.AuditAspect">
		<property name="sHolder"  ref="sHolder"/>
	</bean>

	<bean id="securityCheckAspect" class="com.ba.beans.SecurityCheckAspect" >
		<property name="sHolder"  ref="sHolder"/>
	</bean>

	<aop:config>
		<aop:pointcut expression="execution(public * com.ba.beans.LoanManager.approveLoan(..))" id="pc1"/>

		<aop:aspect ref="securityCheckAspect" order="1">
			<aop:before method="check" pointcut-ref="pc1"/>
		</aop:aspect>
		<aop:aspect ref="auditAspect" order="2">
			<aop:before method="log" pointcut-ref="pc1"/>
		</aop:aspect>

	</aop:config>
</beans>
```

```java
public class Test {
	public static void main(String[] args) {
		ApplicationContext context=
		new ClassPathXmlApplicationContext("com/ba/common/application-context.xml");
		LoanManager proxy=context.getBean("loanManager",LoanManager.class);
		SecurityHolder sHolder=context.getBean("sHolder",SecurityHolder.class);
		sHolder.login("Dhananjaya", "realspeed");
		System.out.println("Loan Status:"+proxy.approveLoan(100000));
		sHolder.logout();
	}
}
```

How to configure Multiple Advices in a single Aspect class?

```
public class MonitorAspect{
        private SecurityHolder sHolder;
        public void setsHolder(SecurityHolder sHolder) {
                this.sHolder = sHolder;
        }
        public void check(JoinPoint jp) throws IllegalAccessException{
                boolean flag=sHolder.authenticate();
                if (flag==false) {

                        System.out.println("Authentication Fails");
                        throw new IllegalAccessException("Invalid User/Password");
                }
                System.out.println("Authentication Success");
        }

        public void log(JoinPoint jp) throws IllegalAccessException{
                String methodName=jp.getSignature().getName();
                Object[]args=jp.getArgs();
                System.out.print(sHolder.getUserDetails()+" is  Accessing "+methodName+"(");
                for (int i = 0; i < args.length; i++) {
                        if (i==0) {
                                System.out.print(args[i]);
                                continue;
                        }
                        System.out.print(","+args[i]);
                }
                System.out.println(")");
        }
}
```

```
<?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/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
                http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
        <bean id="loanManager" class="com.ba.beans.LoanManager"/>
        <bean id="sHolder"  class="com.ba.beans.SecurityHolder"  factory-method="getInstance" />

        <bean id="monitorAspect" class="com.ba.beans.MonitorAspect" >
                <property name="sHolder"  ref="sHolder"/>
        </bean>
        <aop:config>
                <aop:pointcut expression="execution(public * com.ba.beans.LoanManager.approveLoan(..))" id="pc1"/>
                <aop:aspect ref="monitorAspect">
                        <aop:before method="check" pointcut-ref="pc1"/>
                        <aop:before method="log" pointcut-ref="pc1"/>
                </aop:aspect>
        </aop:config>
</beans>
```

➔By using Order we can apply multiple aspect but two method are there in same aspect we can't write order only way to mange is declaring a order format.

Pointcut Designator(PCD)

There is total 5-pointcut Designators are there

- ⇨ within
- ⇨ execution
- ⇨ target
- ⇨ this
- ⇨ bean

➔Out of all Bean is not there in AspectJ this pointcut designator is coming from spring provided PCD.

➔Because bean is not there in aspectJ it is only in IOC container that's why bean came from spring but rest of all are in spring also AspectJ also.

There are 5 PointCut designator are available.

➔@Pointcut("execution(* com.aa.beans.Calculator.*(..))")

➔@Pointcut("target(com.aa.beans.Calculator)")

➔@Pointcut("within(com.aa.beans.*)")

➔@Pointcut("bean(calculator)")

Execution:

➔Before wanted to tell the methods for whom I wanted to apply my aspect,Advice my aspect then pointing to the method that whom I wanted to apply

➔@Pointcut("execution(* *.*.*.*.*(..))")

➔@Pointcut("execution(* com.aa.beans.Calculator.*(..))")

➔When we can write @Pointcut("execution(* *.*.*.*.*(..))") but evaluation of the expression is costly because every time it identify the method it goes to every class and identify the method which is performance impact .

➔Instead of it go to within(),

➔It designated package with in classes,it is not talking with classes with in method that is what a pointcut will be designated for something so what i need to write

@Pointcut("within(com.aa.beans.*)")

Within:

➔I want all the classes, we advice under a package that means every method of the class of the classes of so on so on package.

➔Within this package go and advice all the classes.

@Pointcut("within(com.aa.beans.*)")

➔Not it will evaluate every method or it goes to the bean package and advice the classes itself.

➔If the class is bean package it advice all the method so cost of advising will be less ,proxy generation time will be less

Target

➔I don't want all the classes with in my package,I want a specific class .

➔One class only but all the method within the class so how i need to do it.

@Pointcut("target(com.aa.beans.Calculator)")

This

➔It talks Object reference of a class type means it talks about abject reference ScientificCalculator Object we will created which is extending from Calculator .

➔Now this of the type Calculator or not? yes but not the target of Calculator

This is same as isAssinableform

➔This.classname(this object belongs to the class type)

➔Then advice otherwise don't advice .Which object proxy object of the target object whom we wanted toproxy .

➔@Pointcut("com.aa.beans.ScientificCalculator")

Bean

➔I have <bean id="calcularor1" class="Calculator"> and<bean id="calculator2" class="Calculator">

➔I wrote target(com.aa.beans.Calculator) so how many beans are advice? two why? ➔Because both the target are calculator but I don't want both the beans I want calculator1 only.

@Pointcut("bean(calculator1)")

➔This is spring specific we can specify id of bean whom you wanted to apply using the bean that is the reason spring specification only it is not aspect.

➔We have several classes defines as a bean if you use target, this or execution all the beans of will be advice .

After Returning Advice

➔In after returning advice while writing the advice method in the aspect you need to have declare the method with the return type as void as it cannot control the return value.

➔The method takes two parameters, the first parameter is the Joinpoint and the second would be the variables in which you want to receive the return value of the target method execution.

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

		ApplicationContext context=new
			ClassPathXmlApplicationContext("com/ad/common/application-context.xml");
		KeyGenerator proxy=context.getBean("keyGenerator",KeyGenerator.class);
		int key=proxy.generateKey(8);
		System.out.println(" key : "+key);

		key=proxy.generateKey(6);
		System.out.println(" key : "+key);
	}
}
```

```
public class CheckKeyAspect{
	public void validate(JoinPoint jp,Object ret){
		if ((Integer)ret<=0) {
			throw new IllegalArgumentException("Invalid Key/Length not supported");
		}
	}
}
```

```
<?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/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
		http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
	<bean id="keyGenerator" class="com.ad.beans.KeyGenerator"/>
	<bean id="checkKeyAspect" class="com.ad.beans.CheckKeyAspect"/>

	<aop:config>
		<aop:pointcut expression="execution(public * com.ad.beans.*.*(..))" id="pc1"/>
		<aop:aspect ref="checkKeyAspect">
			<aop:after-returning method="validate" returning="ret" pointcut-ref="pc1"/>
		</aop:aspect>
	</aop:config>
</beans>
```

Throws Advice

➔In this we need to write the aspect class with advice method taking the signature as

```
public void <methodName> (JoinPoint ,Throwable)
```

```
import java.util.Random;
public class Thrower {
        public int willThrow(int i){
                if (i<0) {
                        throw new IllegalArgumentException("Invalid Parameter");
                }
                return i+new Random().nextInt(10000);
        }
}
```

```
public class ExceptionLoggerAspect{
        public void log(JoinPoint jp,IllegalArgumentException iae){
                String methodName=jp.getSignature().getName();
                System.out.println("Exception has raised in "+methodName);
        }
}
```

➔We need to declare a variable in which you are receiving the exception as throwing.

```
<?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-3.0.xsd">
        <bean id="thrower" class="com.ta.beans.Thrower"/>
        <bean id="exceptionLoggerAdvice" class="com.ta.beans.ExceptionLoggerAspect"/>
        <aop:config>
                <aop:pointcut expression="execution(public * com.ta.beans.Thrower.*(..))" id="pc1"/>
                <aop:aspect ref="exceptionLoggerAdvice">
                        <aop:after-throwing method="log" pointcut-ref="pc1" throwing="iae" />
                </aop:aspect>
        </aop:config>
</beans>
```

```
public class Test {
        public static void main(String[] args) {
                ApplicationContext context=
                        new ClassPathXmlApplicationContext("com/ta/common/application-context.xml");
                Thrower proxy=context.getBean("thrower",Thrower.class);
                int i=(Integer)proxy.willThrow(10);
                System.out.println(i);
                i=(Integer)proxy.willThrow(-1);
                System.out.println(i);
        }
}
```

How to use AND Operator(Multiple join points)

```
public class LoggingAspect {
        public Object log(ProceedingJoinPoint pjp,int num1,int num2) throws Throwable {
                String methodName=pjp.getSignature().getName();
                //Object[] args=pjp.getArgs();
                Object ret=null;
                System.out.print("Enter in to the Method "+methodName+"("+num1+","+num2+")");
                ret=pjp.proceed();
                System.out.println("Exit from Method "+methodName+"  and return value is:"+ret);
                ret=(Integer)ret+10000;
                return ret;
        }
}
```

```
<bean id="calculator" class="com.aa.beans.Calculator"/>
<bean id="loggingAspect" class="com.aa.beans.LoggingAspect"/>
<aop:config>
        <aop:aspect ref="loggingAspect">
                <aop:around method="log" pointcut="execution(* *.*.*.*.*(..)) and args(num1,num2)"/>
        </aop:aspect>
</aop:config>
```

```
<bean id="calculator" class="com.aa.beans.Calculator"/>
<bean id="loggingAspect" class="com.aa.beans.LoggingAspect"/>
<aop:config>
        <aop:aspect ref="loggingAspect">
                <aop:around method="log" pointcut="execution(public int com.aa.beans.Calculator.add(..)) and within(com.aa.beans.*)"/>
        </aop:aspect>
</aop:config>
```

AspectJ Annotation Approach

➔In this approach instead of using declaration to expose the classes as aspect and advice,we will annotate our classes with annotations.

➔ In order to make a class as aspect we need to annotate the class with **@Aspect**

➔To expose a method as advice method we need to annotate the method in aspect class with **@Around or @Before or @AfteReturning ,@AfterThrowing** represents the types of the advice.

➔This annotation takes the pointcut expression as value in them

@Around with Spring Bean Configuration & Stereotype approach

```
@Component
public class Calculator {
        public int add(int a,int b){
                System.out.println("add()");
                return a+b;
        }
        public int multiply(int a,int b){
                System.out.println("multiply()");
                return a*b;
        }
        public int sub(int a,int b){
                System.out.println("sub()");
                return a-b;
        }
}
```

```
@Component
@Aspect
public class LoggingAspect {
        @Around(value="execution(* com.aa.beans.*.*(..))")
        public Object log(ProceedingJoinPoint pjp) throws Throwable {
                String methodName=pjp.getSignature().getName();
                Object[] args=pjp.getArgs();
                Object ret=null;
                System.out.print("Enter in to the Method "+methodName+"("+args[0]+","+args[1]+")");
                ret=pjp.proceed();
                System.out.println("Exit from Method "+methodName+"  and return value is:"+ret);
                ret=(Integer)ret+10000;
                return ret;
        }
}
```

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

        <context:component-scan base-package="com.aa.beans"></context:component-scan>
        <aop:aspectj-autoproxy/>
</beans>
```

@Before with Spring Bean configuration approach

```
public class LoanManager {
        public boolean approveLoan(int amount){
                        System.out.println(" inside approveLoan()");
                        return new Random().nextBoolean();
        }
}
```

```
@Aspect
public class LoggingAspect {
        @Before("within(com.ba.beans.*)")
        public void log(JoinPoint jp) throws IllegalAccessException {
                String methodName = jp.getSignature().getName();
                Object[] args = jp.getArgs();
                System.out.println("Dhananjaya is  Accessing " + methodName + "("
                                + args[0] + ")");
        }
}
```

```
<?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/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
                http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
        <bean id="loanManager" class="com.ba.beans.LoanManager"/>
        <bean id="loggingAspect" class="com.ba.beans.LoggingAspect" />
        <aop:aspectj-autoproxy/>
        <!-- <aop:config>
                <aop:aspect ref="loggingAspect">
                        <aop:before method="log" pointcut="execution(public * com.ba.beans.LoanManager.approveLoan(..))"/>
                </aop:aspect>
        </aop:config> -->
</beans>
```

@AfterReturning with Spring Bean configuration approach

```
@Component
public class KeyGenerator {
        public int generateKey(int key){
                if (key<8) {
                        return 0;
                }
                return 1;
        }
}
```

```
@Component
@Aspect
public class CheckKeyAspect{

        @AfterReturning(value="within(com.ad.beans.*)" ,returning="ret")
        public void validate(JoinPoint jp,Object ret){
                if ((Integer)ret<=0) {
                        throw new IllegalArgumentException("Invalid Key/Length not supported");
                }
        }
}
```

```
<context:component-scan base-package="com.ad.beans"></context:component-scan>
<aop:aspectj-autoproxy/>
```

@AfterThrowing with Spring Bean configuration approach

```
public class Test {
	public static void main(String[] args) {
		ApplicationContext context=new
		 ClassPathXmlApplicationContext("com/ta/common/application-context.xml");
		Thrower proxy=context.getBean("thrower",Thrower.class);
		int i=(Integer)proxy.willThrow(10);
		System.out.println(i);
		i=(Integer)proxy.willThrow(-1);
		System.out.println(i);
	}
}
```

```
@Component
public class Thrower {
	public int willThrow(int i){
		if (i<0) {
			throw new IllegalArgumentException("Invalid Parameter");
		}
		return i+new Random().nextInt(10000);
	}
}
```

```
@Component
@Aspect
public class ExceptionLoggerAspect{
	@AfterThrowing(value="target(com.ta.beans.Thrower)",throwing="iae")
	public void log(JoinPoint jp,IllegalArgumentException iae){
		String methodName=jp.getSignature().getName();
		System.out.println("Exception has raised in "+methodName);
	}
}
```

```
<?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/aop http://www.springframework.org/schema/aop/spring-aop-3.0.xsd
		http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd">
		<context:component-scan base-package="com.ta.beans"></context:component-scan>
	<aop:aspectj-autoproxy/>
</beans>
```

@Order

Q.How to order If multiple Aspect classes are there ?

```
@Component
public class LoanManager {
	public boolean approveLoan(int amount){
			System.out.println("in approveLoan()");
			return new Random().nextBoolean();
	}
}
```

```
@Component("sHolder")
public class SecurityHolder {
	private static SecurityHolder instance;
	private ThreadLocal<Credential> userCred;
	private Credential userDetails;
	private SecurityHolder() {
		userCred=new ThreadLocal<Credential>();
	}

	public static synchronized SecurityHolder getInstance(){
		if (instance==null) {
			instance=new SecurityHolder();
		}
		return instance;
	}
	/**
	 * Storing user & password in localThread
	 * @param user
	 * @param password
	 */
	public void login(String user,String password){
		userDetails=new Credential(user,password);
		userCred.set(userDetails);
	}
	/**
	 * remove values from localThread
	 */
	public void logout(){
		userCred.set(null);
	}
	/**
	 * Checks whether the credentials are valid or not
	 * @return
	 */
	public boolean authenticate(){
		if (userCred!=null) {
			if (userCred.get().getUser().equals("Dhananjaya")
	&& userCred.get().getPassword().equals("realspeed")) {
				return true;
			}
		}
		return false;
	}
	/**
	 * getting the current user details
	 * @return
	 */
	public String getUserDetails(){
		if (userCred!=null) {
			return userCred.get().getUser();
		}
		return null;
	}
}
```

```
@Component
@Aspect
@Order(value=2)
public class AuditAspect{
	@Autowired
	private SecurityHolder sHolder;
	@Before("execution(public * com.ba.beans.LoanManager.*(..))")
	public void log(JoinPoint jp) throws IllegalAccessException{
		String methodName=jp.getSignature().getName();
		Object[]args=jp.getArgs();
		System.out.print(sHolder.getUserDetails()+" is  Accessing "+methodName+"(");
		for (int i = 0; i < args.length; i++) {
			if (i==0) {
				System.out.print(args[i]);
				continue;
			}
			System.out.print(","+args[i]);
		}
		System.out.println(")");
	}
}
```

```
@Component
@Aspect
@Order(value=1)
public class SecurityCheckAspect{
	@Autowired
	private SecurityHolder sHolder;

	@Before("execution(public * com.ba.beans.LoanManager.*(..))")
	public void check(JoinPoint jp) throws IllegalAccessException{
		boolean flag=sHolder.authenticate();
		if (flag==false) {
			System.out.println("Authentication Fails");
			throw new IllegalAccessException("Invalid User/Password");
		}
		System.out.println("Authentication Success");
	}
}
```

```
<context:component-scan base-package="com.ba.beans"/>
<aop:aspectj-autoproxy/>
```

```
public class Test {
	public static void main(String[] args) {
		ApplicationContext context=new
		 ClassPathXmlApplicationContext("com/ba/common/application-context.xml");
		LoanManager proxy=context.getBean("loanManager",LoanManager.class);
		SecurityHolder sHolder=context.getBean("sHolder",SecurityHolder.class);
		sHolder.login("Dhananjayas", "realspeed");
		System.out.println("Loan Status:"+proxy.approveLoan(100000));
		sHolder.logout();
	}
}
```

@Pointcut

```
@Component
@Aspect
public class CommonAspect {

    @Pointcut("execution(* com.aa.beans.Calculator.*(..))")
    public void monitorPointcut(){
    }

    @Around("monitorPointcut()")
    public Object log(ProceedingJoinPoint pjp) throws Throwable {
        String methodName=pjp.getSignature().getName();
        Object[] args=pjp.getArgs();
        Object ret=null;
        System.out.print("Enter in to the Method "+methodName+"("+args[0]+","+args[1]+")");
        ret=pjp.proceed();
        System.out.println("Exit from Method "+methodName+"  and return value is:"+ret);
        ret=(Integer)ret+10000;
        return ret;
    }

    @Around("monitorPointcut()")
    public Object doCache(ProceedingJoinPoint pjp) throws Throwable {
        String methodName=pjp.getSignature().getName();
        Object ret=null;
        System.out.println("Checks data in cache");
        ret=pjp.proceed();
        System.out.println("Exit from Method "+methodName+"  and return value is:"+ret);
        return ret;
    }

}
```

```
<context:component-scan base-package="com.aa.beans"></context:component-scan>
    <aop:aspectj-autoproxy/>
```

```
@Component
public class Calculator {
    public int add(int a,int b){
        System.out.println("add()");
        return a+b;
    }
    public int multiply(int a,int b){
        System.out.println("multiply()");
        return a*b;
    }
    public int sub(int a,int b){
        System.out.println("sub()");
        return a-b;
    }
}
```

```
public class Test {
    public static void main(String[] args) {
        ApplicationContext context=new
        ClassPathXmlApplicationContext("com/aa/common/application-context.xml");
        Calculator proxy=context.getBean("calculator",Calculator.class);
        System.out.println(proxy.getClass().getName());
        int sum=(Integer)proxy.add(10, 20);
        System.out.println("Sum :"+sum);
    }
}
```

➔By using Order we can apply multiple aspects but two methods are there in same aspect class we can't write order only way to mange is declaring an order format.

AroundAdvice with Configuration class Approach

➔To enable AspectJ we need to write a annotation @EnableAspectJAutoProxy
➔No need to annotate Aspect class as @Component

```
@Configuration
@ComponentScan(basePackages={"com.aa.beans"})
@EnableAspectJAutoProxy
public class AppConfig {
        @Bean
        public Calculator newCalculator(){
                return new Calculator();
        }
        @Bean
        public CommonAspect newCommonAspect(){
                return new CommonAspect();
        }
}
```

```
@Component
public class Calculator {
        public int add(int a,int b){
                System.out.println("add()");
                return a+b;
        }
        public int multiply(int a,int b){
                System.out.println("multiply()");
                return a*b;
        }
        public int sub(int a,int b){
                System.out.println("sub()");
                return a-b;
        }
}
```

```
@Aspect
public class CommonAspect {
        @Pointcut("execution(* com.aa.beans.Calculator.*(..))")
        public void monitorPointcut(){
        }
        @Around("monitorPointcut()")
        public Object log(ProceedingJoinPoint pjp) throws Throwable {
                String methodName=pjp.getSignature().getName();
                Object[] args=pjp.getArgs();
                Object ret=null;
                System.out.print("Enter in to the Method "+methodName+"("+args[0]+","+args[1]+")");
                ret=pjp.proceed();
                System.out.println("Exit from Method "+methodName+"  and return value is:"+ret);
                ret=(Integer)ret+10000;
                return ret;
        }
}
```

```
public class Test {
        public static void main(String[] args) {
                ApplicationContext context=new AnnotationConfigApplicationContext(AppConfig.class);
                Calculator proxy=context.getBean("calculator",Calculator.class);
                System.out.println(proxy.getClass().getName());
                int sum=(Integer)proxy.add(10, 20);
                System.out.println("Return Sum :"+sum);
        }
}
```