Spring AOP

When you are developing any Enterprise Application, You need the following services commonly.

1. Low Level services
2. Middle Level Services
3. High Level Services

Low level Services:

Some of the Low Level Services are IO, Threading, Networking etc which will be supplied by servers freely.

Middle Level Services:

Some of the Middle Level Services are Transactions, Security, Logging, Messaging etc which has to be implemented by you.

Normally, when you implement Business Operation You need to write the code for Business Logic and middle Level services.

Consider the following requirement:

```
            Class Account Service {
                        Public void deposit (…)d{
                        If(x.isCallerInRole("Telller")){
                        Try{
                        Log.info(….);
                        Tx.begin();
                        OP1;
                        OP2;
                        Tx.commit();
                        Log.info(…);
                        }catch (Exception e) {
                        Tx.rollback();
                        Log.info();
                        }
                        }else{
                        Throw some security Exception;
            }
    }
}
```

In the above code,

- Core Business logic is mixed with middle level services like Transaction, security and Logging.
- When you want to change the existing Transactions API or Security API or Logging API With new one then you need to re-modify the entire application.
- This may give maintenance problem.

Spring AOP

- AOP stands for Aspect oriented programming.
- AOP is new Kind of Programming technique which is mainly used to separate the commonly required middle level services logic from core business logic of the application.
- Transactions, security, Logging, Messaging etc are the middle level services which are also called as cross cutting concerns.

With AOP

BusnessServices	Middle Level Services

Class Account Service {
Public void dep0osit(….) {
OP1;
OP2;
OP3;
}
….
}
Class CustomerService {
Public void and Customer (….){
OP1;
OP2;
}
….
}
	Class TxService {
	Void begin() {….}
	Void commit() {….}
	Void commit() {….}
	}
	Class Security Service {
	Public void verify User () {
	…..
	}
	}
	Class LogService {
	Void Log(…){
	….
	}
	}

In the above code,

1. Core Business logic is completely separated form middle level services.
2. Now when you want to change the existing Transactions, Security or Logging implementations with new one then that will not impact the business service

Following are various AOP Frameworks available

1. Spring AOP
2. Aspectj
3. JBoss Aop

Spring 2.0 AOP is integrated with Aspectj.

Spring AOP Terminology

1. Aspect
2. Advice
3. joinPoint
4. PointCut

5. Advisor
6. Target
7. Proxy
8. Weaving

Aspect

- Commonly Required Middle Level services which you are implementing for your Enterprise application are called as Aspects.
- Security, Transactions, Logging etc are aspects.

Advice

- Implementation of a Middle Level service is called as advice.
- Implementation of an Aspect is called as advice.
- i.e. Advice is a class which contains Code for Aspects like security, Txs, logging etc.
  Ex:
  Class TxService{….}
  Class security Service {….}
  Class LogService {….}

Join Point

- Join Point is a point in the Program execution where you want to apply advices.
- Join Point is point in the program execution where you want to run middle level services code.

```
Try{
        Txs.begin();      Before Business Operation
        As.deposity();    Business Operation
        Txs.commit();     After Business Operation returns the control Successfully
}catch(Exception e){
        Txs.rollback();   After Business Operation throws Exception
}
```

Spring AOP supports the following joinPoints

1.	MethodBefore	Before invoking method
2.	MethodReturing	when method Returns the control successfully
3.	MethodThrowing	When method throws an Exception
4.	MethodAfter	When mothod returns control any way.

PointCut

- Collection of joinPoints is called as PointCut.
- By default, advices will be applied for all the business operations of all the business services.
- When you want apply the advices for some specified business operations of specified Business Services then you must define point-cut with the required AspectJ Expression.

  Execution(“com.jtcindia.*Service.*(…))
  Advice will be applied for

getBal() of AccountService
mydeposit() of AccountService
mywithdraw() of AccountService
add Customer () of AccountService
updateCustomer()of CustomerService

execution(* com.jtcindia.AccountService.my*(…))
advice will be applied for
mydeposit() of AccountService
mywithdraw() of AccountService

execution(* com.jtcindia.CustomerService.update*(…))
advice will be applied for
updateCustomer () of CustomerService

Define the pointCut
To define the pointCuts, you can use the following;
JDK Regular Expression
Aspectj Expression.
Using aspect Expression
Syntax

Execution (modifier Pattern return Type Pattern?
Business Service Pattern. Method Pattern (Params Pattern) throws Exception Pattern?)

Ex:
1) Any public method form any business service
Execution (public ** (..))
2) Any setter method form any business service
Execution(***.set*(…)throws*)
Execution(**.set*(..))
3) Any setter method form HelloService.
Execution(*com.jtc.HelloService.set*(..))
4) any static methods starting with add form HelloService with return type int.
Execution (static int com.jtc.Helloservice.add*.(..))
5) any methods starting with add from services ending service.
Execution (**service.add*(..)

To specify the required Aspectj expression, you need to configure the bean in the spring configuration Document with AspectjExpressionPointcut class as follows.

```
<bean id=”MyPC1” class=”org.springframework.aop.aspectj.AspectjExpressionPointcut”>
<property name=”expression” value=”executionhn(*com.jtc.accountService.My*(..))”/>
</bean>
```

Advisor

Advisor is combination of Advice and point Cut.

Defining the Advisors

To specify the Required Advisors, you need to configure the bean in the spring configuration Document with default Pointcut Advisor class as follows.

```
<bean name=”MBA dvisor” class=”org.springfrom ework.aop.support.DefaultPointcutAdvisor”>
        <property name=”advice” ref=”mba”/>
        < property name=”pointcut” ref=”MyPC1”/>
</bean>
```

Now you have to refer advisor in the proxy Factory Bean instead of Advice.

```
<bean id=”baseProxy” class=”org.springframework.aop.framework.proxyFactoryBean” abstract=”true”>
        <property name=”interceptor Name”>
        <list>
                <value>MBAdvisor</value>
        </list>
        </property>
</bean>
```

Note:

- interceptorNames are list of advices or advisors.
- When you use list of advices then advices will be applied to all the methods of all the business services.
- When you use list of advisors then advices will be applied to only for the methods which are matching with the given pointcut expression.

Target

Target is an object of your business service before applying the Advices or advisors.

Proxy

Proxy is an object of you business service after applying the advices or advisors.

Weaving

Weaving is the process of applying the advices or advisors to the Target objects at given pointcuts to get the proxy objects.

Steps with MyEclipse:

1. Create the Java Project.
2. Add the spring Capabilities as follows.
    Select project, Right click and then select My Eclipse > Add spring capabilities
    Do the following
        Select spring 3.1 version
        Select spring 3.1 core Libraries and click on Next Button.
        Change file name to jtcindia.xml and click on finish button.

3. Open jtcindia.xml and enable the context namespace.

Spring provides 3 ways to implement AOP
1. using spring API based AOP (Available from spring 1.0)
2. using Annotation based AOP (Available from spring 2.O)
3. Using Schema based AOP (Available form spring 2.0)

Using spring API based AOP
Steps:

Consider some Business services with some business operations.

With class model
```
        Class Account Service {
        Public void getBal(){
        ....
        }
        Public void mydeposit() {
        .....
        }
        Public void mywithdraw(){
        ....
        Throw new Insufficient Funds Fecep0tion();

        }
        }
```
With interface Model

```
 Class  Account Service{
Public void getBal(){
....
}
Public void my deposid(){
....
}
Public void mywithdraw(){
....
Throw new Insufficient Funds Exception();
}
}
```

With interface model

```
Interface customer Service{
Public void addCustomer();
```

Public void updateCustomer();
}
Class Customer serviceImpl implements customerService{
Public void addCustomer() {
….
}
Public void updateCustomer(){
…
}
}

Configure these two business service beans in spring configuration Document.
<bean id=" asTarget" class="com.jtcindia.spring.Account Service"/>
<bean id="csTarget" class="come.jtcindia.spring.CustomerServiceImpl"/>

Identify the Aspects or middle Level services required for your Application.
I need security, Transaction and Log Aspects.
Write Service classess for your Aspects or middle level services.
TxService.java
securityService.java
secuirtyService.java
logService.java
configure these three service beans in spring configuration Document.
<bean id="ss" class=" com.jtcindia.spring.securityService"/>
<bean id="ts" class=" com.jtcindia.spring.TxService"/>
<bean id="ls" class=" com.jtcindia.spring.LogService"/>

Identify the joinPonits required for you business operation.
I need 3 join Points.
Method before
Method returning
Method throwing

Write the Required advice classes as per the joinpoints required.
Class jtcmbadvice implements MethodBeforeAdvice{
Public void before(…){
//verify use
//log the info
//begin transaction
}
}
Class jtcmradvice implements after Returning Advice{
Public void after Returing(…){
//commit transaction.
//log the info

```
}
}
Class jtcmtadvice implemtnts Throws Advice{
Public void after Throwing (…){
//rollback transaction.
//log the info
}
}
```

Configure these three advice beans in spring configuration Document.

```
        <bean id=”mba” class=”com.jtcindia.spring.JTCMBAvice”/>
        <bean id=”mra” class=”com.jtcindia.spring.JTCMRAvice”/>
        <bean id=”mta” class=”com.jtcindia.spring.JTCMTAvice”/>
```

Configure proxy objects for your business services using proxy Factory Bean

```
<bean id=”asProxy” class=”org.springframework.aop.framework.proxyFactoryBean”>
        <property name=”targetClass” value=”com.jtcindia.spring.aop.AccountService”/>
        <property name=”target” ref=”asTarget”/>
        <property name=”interceptor Names”>
                <list>
                <value>mba</value>
                <value>mra</value>
                <value>mta</value>
        </list>
        </property>
</bean>

<bean id=”csProxy” class=”org.springframework.aop.framework.proxyFactoryBean”>
        <property name=”targetClass” value=”com.jtcindia.spring.aop.CustomerService”/>
        <property name=”target” ref=”asTarget”/>
        <property name=”interceptor Names”>
                <list>
                <value>mba</value>
                <value>mra</value>
                <value>mta</value>
        </list>
        </property>
</bean>
```

Get the target object and invoke the business operations.

- When you invoke the business operation on target object then business operation will be called directly without applying advices.

```
customerService cst=(Customerservice) ctx.getBean(“csTarget”);
cst.add Customer ();

AccountService ast=(AccountService) ctx.getBean(“asTarget”);
Ast.mydeposit();
```

Get the proxy object and invoke the business operations.

- When you invoke the business operation on proxy object then business operation will be called by applying advices at specified joinPoints.

```
customerService cst=(Customerservice) ctx.getBean(“cs proxy”);
cst.add Customer ();

AccountService ast=(AccountService) ctx.getBean(“as proxy”);
Ast.mydeposit();
```

Object Representation inside the spring container

Jtc34: Files required

Jtc34.java	AccountService.java
customerService.java	customerServiceImpl.java
LogService.java	SecurityService.java
TxService.java	JTCMBAdvice.jave
JTCMRAdvice.java	JTCMTAdvice.java
insufficientFundsException.java	Jtcindia.xml

Jtc 34.java

```
Package com.jtcindia.spring;
Import org.springframework.context.ApplicationContext;
Import org.springfromework.Context.Support.ClassPathXml ApplicationContext;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit         : www.jtcindia.org
**/
Public class Jtc34{
```

```
Public static void main(String[] args) {
ApplicationContext ctx=new ClassPathXmlApplicationContext(“jtcindia.xml”);
system.out.println(“—Using Target object—“);
customerService cst=(CustomerService)ctx.getbean(“csTarget”);
cst.addCustomer();
cst.updateCustomer();
AccountService ast=(AccountService)ctx.getBean(“asTarget”);
Ast.mydeposit();
Ast.getBal();
Try{
Ast.mywithdraw();
}catch(Exception e){
System.out.println(“sorry---“);
}
System.out.println(“-------------“)
System.out.println(“—Using proxy object—“);
Customerservice csp=(AccountService)ctx.getBean(“asProxy”);
Asp.getBal():
Try{
Asp.mywithdraw();
}catch(Exception e) {
System.out.println(“Sorry---“);
}
} }
```

AccountService.java	CustomerServiceImple.java

Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit : www.jtcindia.org
**/
Public class Account Source {
Public void getBal(){
System.out.println(“getBal-begin”);
System.out.println(“getBal-done”);
System.out.println(“getBal-end”);
}
Public void mydeposit(){
System.out.println(“deposit-begin”);
System.out.println(“deposit -done”);
	Package com.jtcindia.spring;
	/*
	*@Author: Som Prakash Rai
	*@Company : java Training Center
	*@ visit : www.jtcindia.org
	**/
	public class customerServiceImpl implements CustomerService{
	Public void addCustoemr(){
	System.out.println(“addCustomer-begin”);
	System.out.println(“addCustomer- done”);
	System.out.println(“addCustomer- end”);
	}
	Public void updateCustomer(){
	System.out.println(“updateCustomer -begin”);

```
System.out.println("deposit -end");
}
Public void mywithdraw() throws Exception{
System.out.println("withdraw-begin");
If(1==1){
Throw new insufficient Funds Exception();
}
System.out.println("withdraw-end");
}
}
```

customerService.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit         : www.jtcindia.org
**/
Public interface CustomerService {
Public void addCustomer();
Public void update Customer();
}
```

SecurityService.java

```
Package com.jtcindia.spring;
Public class securityService {
Public void verify User(){
System.out.println("**SS – VerifyUser");
}
}
```

```
System.out.println("updateCustomer - done");
System.out.println("updateCustomer - end");
}
}
```

LogService.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit         : www.jtcindia.org
**/
Public class logService {
Public void logBefore(){
System.out.println("**LS-logBefore");
}
Public void logReturning(){
System.out.println("**LS-LogReturning");
}
Public void logThrowing (Exception e){
System.out.println("** LS-logThrowing");
System.out.println(e);
}
}
```

TxService.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit         : www.jtcindia.org
**/
Public class TxService {
Public void begin(){
System.out.println("**TS- begin");
}
Public void commit(){
System.out.println("**TS- commit");
```

JTCMRAadvice.java

```
Package com.jtcindia.spring;
Import java.lang.reflect.Method;
Import
org.springframework.aop.AfterReturningAdvice;
Import
org.springframework.beans.factory.annotation.Au
towired;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit         : www.jtcindia.org
**/
```

```
}
Public void rollback(){
System.out.println("** TS- rollback");
}
}
```

JTCMRAadvice.java

```
Package com.jtcindia.spring;
Import java.lang.reflect.Method;
Import
org.springframework.aop.MethodBeforeAdvice;
Import
org.springframework.beans.factory.annotation.Au
towired;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit        : www.jtcindia.org
**/
Public class JTCMRAadvice implements
MethodBeforAdvice {
@Autowired
LogService is=null;
@Autowired
logService ss=null;
@Autowired
logService ts=null;
@override
Public void before(method method, object[]
args,object target)
throws Throwable {
}
```

InsufficientFundsException.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit        : www.jtcindia.org
**/
Public class insufficientFundsException extends
Exception {}
```

```
Public class JTCMRAadvice implements
AfterReturningAdvice {
@Autowired
LogService is=null;
@Autowired
TxService ts=null;
@override
Public void afterReturning(Object rv, Method
method,
Object[] args, object target) throws Throwable {
Ts.commit();
Is.logReturning();
}
}
```

JTCMRAadvice.java

```
Package com.jtcindia.spring;
Import org.springframework.aop.ThrowsAdvice;
Import
org.springframework.beans.factory.annotation.Au
towired;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit        : www.jtcindia.org
**/
Public class JTCMRAadvice implements
ThrowsAdvice {
@Autowired
LogService is=null;
@Autowired
TXService ts=null;
Public void after Throwing (Exception e) {
Ts.rollback();
Is.logThrowing(e);
}
}
```


Jtcindia.xml

```
<?xml version ="1.0" encoding="UTF-8"?>
<beans xmlns=http://www.springgramework.org/schema/beans
        Xmlns:p=http://www.springframework.org/schema/p
        Xmlns:context=http://www.springframewok.org/schema/context
        Xsi:schemaLocation="http://www.springframework.org/schema/beans
        Xmlns:xsi="http://www.w3.org/2001/xmlschema-instance:
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org./schema/context http://www.springframework.org/schema/context/
spring context-3.0.xsd">

<ontext:annotation-config/>
<bean id="ss" class="com.jtcindia.spring.securityService"/>
<bean id="ls" class="com.jtcindia.spring.LogService"/>
<bean id="ts" class="com.jtcindia.spring.TxService"/>
<bean id="mba" class="com.jtcindia.spring.JTCMBAdvice"/>
<bean id="mra" class="com.jtcindia.spring. JTCMBAdvice"/>
<bean id="mta" class="com.jtcindia.spring. JTCMBAdvice"/>
<bean id="asTarget" class="com.jtcindia.spring.AccountService"/>
<bean id="csTarget" class="com.jtcindia.spring.CustomerServiceImpl"/>
<bean id="baseProxy" class="com.springframework.aop.framework.proxyFactoryBean"
abstract="true">
<property name="interceptorName">
<list>
<value>mba</value>
<value>mra</value>
<value>mta</value>
</list>
</property>
</bean>
<bean id=" asProxy" parent="baseProxy">
<property name="targetClass" value="com.jtcindia.spring.AccountService"/>
Property name="target"ref="as Target"/>
<bean>
<bean id="csProxy" parent="baseProxy">
<property name="proxylenterfaces" value="com.jtcindia.spring.customerService"/>
<property name="target" ref="csTarget"/>
<bean> </beans>
```

Jtc34 Flow:
What you invoke the method on target object then directly that method will be called and only business logic will run.

When you invoke the method on proxy object then following tasks will be done by proxy object.

1. Proxy checks whether method invoked is available in targetClass or proxyInterface specified by you.
2. If method is found in targetClass or proxy Interface specified by you then collect the list of interceptor Names opecified by you.
3. Proxy checks whether any interceptor (Advice)is there is list of interceptor Names which has be applied before invoking method.
4. If MethodBeforeAdvice is found then before () method will be called.
5. Invokes the method on Target object.
6. If method called on Target object is retruning control successfully then proxy checks whether any interceptoer (Advice) is there is list of interceptor Names which has be applied after retuning control successfully.
7. If After Returning Aduvice is found then after Returning() method will be called.
8. If method called on target Object is throwing some exception then proxy checks whether any interceptor (Advice) is there is list of interceptor Names which has be applied after throwing exception.
9. If ThrowsAdcice is found then after Throwing() method will be called.

Autoproxying

- Autoproxying is process of creating proxy object for your business services automatically without configuring the proxy objects explicitly using proxyFactoryBean.

- To enable Autoproxying, you need do the following:
  Enable the aop namespace.(it is like context namespace)
  You need to write the following tag in spring configuration Document.
  <aop:aspect-autoproxy/>

Explicit Proxying:

- When you configure Proxy object with the type proxyFactoryBean with 3 properties called interceptorNames, target, targetClass/proxyInterfaces then It is called as Explicit proxying.

Autoproxying:

- Proxy object will be created by the container for every business service object (target object) by detecting the list of Advisors configured by you in spring configuration Document.
- At container start-up, spring container will scan all the advisors and prepares the List of Advisors.

When client invokes the method container will do the following:

1. Intercept the method call and checks whether that method is matching with any point cut expression specified with any Advisor.
2. When method is not matching with any point cut expression specified with any Advisor then that method will be called directly without applying any advice.
3. If method is matching with point cut expression specified with Advisors then container takes the list of advices associated with that matching point-cut Expression.
4. Apply Those Identified Advices on the method at appropriate Join points.

Jtc35: Files required

Jtc35.java	AccountService.java
customerService.java	customerServiceImpl.java
LogService.java	SecurityService.java
TxService.java	JTCMBAdvice.jave
JTCMRAdvice.java	JTCMTAdvice.java
insufficientFundsException.java	Jtcindia.xml

Jtc 35.java

```
Package com.jtc india.spring;
Import org.springframework.context.ApplicationContext;
Import org.springfromework.Context.Support.ClassPathXml ApplicationContext;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit         : www.jtcindia.org
**/
Public class lab35{
Public static void main(String[] args) {
ApplicationContext ctx=new ClassPathXmlApplicationContext("jtcindia.xml");
customerService cst=(CustomerService)ctx.getbean("cs");
cst.addCustomer();
cst.updateCustomer();
AccountService ast=(AccountService)ctx.getBean("as");
Ast.mydeposit();
Ast.getBal();
Try{
Ast.mywithdraw();
}catch(Exception e){
System.out.println("sorry---");
} } }
```

Jtcindia.xml

```
<?xml version ="1.0" encoding="UTF-8"?>
<beans xmlns=http://www.springgramework.org/schema/beans
        Xmlns:p=http://www.springframework.org/schema/p
        Xmlns:context=http://www.springframewok.org/schema/context
        Xsi:schemaLocation="http://www.springframework.org/schema/aop
        Xmlns:xsi="http://www.w3.org/2001/xmlschema-instance:
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org./schema/context http://www.springframework.org/schema/context/
spring context-3.0.xsd">
context-3.0.xsd http:// www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

<context:annotation-config/>
<aop:aspect-autoproxy/>
<bean id="ss" class="com.jtcindia.spring.securityService"/>
<bean id="ls" class="com.jtcindia.spring.LogService"/>
<bean id="ts" class="com.jtcindia.spring.TxService"/>
<bean id="mba" class="com.jtcindia.spring.JTCMBAdvice"/>
<bean id="mra" class="com.jtcindia.spring. JTCMBAdvice"/>
<bean id="mta" class="com.jtcindia.spring. JTCMBAdvice"/>
```

```
<bean id=”as” class=”com.jtcindia.spring.AccountService”/>
<bean id=”cs” class=”com.jtcindia.spring.coustomerServiceImpl”/>
<bean id=”PC1” class=”org.springframework.aop.aspectj.aspectJExpressionPointcut”>
<property name=”expression” value=”execution(*com.jtcindia.spring.AccountService.My*(..))”/>
</bean>
<bean name=”MBAdvisor1” class=”org.springframework.aop.support.DefaultPointcutAdvisor”>
<property name=”advice”ref=”mba”/>
<property name=”pointcut” ref=”pc1”/>
</bean>
<bean name=”MRAdvisor1” class=”org.springframewok.aop.support.DefaultPointcutAdvisor”>
<property name=”advice” ref=”mra”/>
<property name=”pointcut”ref=”pc1”>
</bean>
<bean name=”MRAdvisor1” class=”org.springframewok.aop.support.DefaultPointcutAdvisor”>
<property name=”advice” ref=”mta”/>
<property name=”pointcut”ref=”pc1”>
</bean>
<bean id=”PC2 class=”org.springframework.aop.aspectj.AspectJExpressionpointcut”>
<property name=”expression” value=”execution(* com.jtcindia.spring.customerService.update*(..))”/>
</bean>
<bean name=”MBAdvisor2” class=”org.springframework.aop.support.DefaultPointcutAdvisor”>
<property name=”advice”ref=”mba”/>
<property name=”pointcut” ref=”pc2”/>
</bean>
<bean name=”MRAdvisor2” class=”org.springframewok.aop.support.DefaultPointcutAdvisor”>
<property name=”advice” ref=”mra”/>
<property name=”pointcut”ref=”pc2”>
</bean>
<bean name=”MRAdvisor2” class=”org.springframewok.aop.support.DefaultPointcutAdvisor”>
<property name=”advice” ref=”mta”/>
<property name=”pointcut”ref=”pc2”>
</bean>
</beans>
```

Jtc 34:

API based AOP
Method before, method Returning, Method Throws
Explicit Proxying

Jtc 34:

API based AOP
Method before, method Returning, Method Throws
Pointcuts, Advisors
Autoproxying

Using Annotation based AOP (2.0)

1. @Aspect
2. @PointCut
3. @Before
4. @AfterReturning
5. @AfterThrowing
6. @After
7. @Around

Steps:

1. Consider some Business Services with some Business operations.
   Account service. Java
   CustomerService.java
   Customer ServiceImpl.java
2. Define these two business service beans in spring Configuration Document.
3. Identify the Aspects required for your Application.
   I need security, transaction and Log aspects.
4. Write advice classes for your aspects or middle level services.
   TxService.java
   secuirtyService.java
   logService.java
5. Define these three advices in spring configuration Document.
6. Identify the join points required for your business operations.
   I need 4 join Points.
   method before
   method returning
   method throwing.
   after method exection.
7. Use the required annotations to specify the join points for the operations of middle level services

When you are writing Annotation based advices, you need to check the following:

- You advice class has to marked with @ Aspect annotation.
- You must have one or mote special methods with @ pointcut annotation with AspectJ Expression.
- These special methods should not have any parameters and should have void as return type.
- Refer that special method name with @Before, @After, @After Returning and @AfterThrowing
  Ex:

```
@Aspect
Public class logService{
@Pointcut(value="execution(***service.m*(..))")
        Public void jtc(){}
```

```
@pointcut(value=”execution(***Service.am*(..))”)
        Public void jtc2() {}
@Before(“jtc1()”)
        Public void logBegin(){
        ….
        }
@AfterReturning(“jtc2()”)
        Public void logEnd(){
        …
}
```

Get the business service object (It is the proxy object by default with autoproxying concept)

Jtc36:

- Annotation based AOP
- @Before, @After Returning, @After Throwing, @After
- Pointcuts based on annotations with @PointCot
- Autoproxying

Jtc36: Files required

Jtc36.java	AccountService.java
customerService.java	customerServiceImpl.java
LogService.java	SecurityService.java
TxService.java	insufficientFundsException.java
Jtcindia.xml	

Jtc 36.java

```
Package com.jtcindia.spring;
Import org.springframework.context.ApplicationContext;
Import org.springfromework.Context.Support.ClassPathXml ApplicationContext;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit         : www.jtcindia.org
**/
Public class Jtc36{
Public static void main(String[] args) {
ApplicationContext ctx=new ClassPathXmlApplicationContext(“jtcindia.xml”);
customerService cst=(CustomerService)ctx.getbean(“cs”);
cs.addCustomer();
system.out.println(“……….”);
cs.updateCustomer();
system.out.println(“………….”);
AccountService as=(AccountService)ctx.getBean("as");
As.mydeposit();
System.out.println("………………);
As.getBal();
System.out.println("……….");
Try{
As.mywithdraw();
}catch(Exception e){
System.out.println("sorry…."); 
} } }
```

AccountService.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit       : www.jtcindia.org
**/
Public class Account Source {
Public void getBal(){
System.out.println("getBal-begin");
System.out.println("getBal-done");
System.out.println("getBal-end");
}
Public void mydeposit(){
System.out.println("deposit-begin");
System.out.println("deposit -done");
System.out.println("deposit -end");
}
Public void mywithdraw() throws Exception{
System.out.println("withdraw-begin");
If(1==1){
Throw new insufficient Funds Exception();
}
System.out.println("withdraw-end");
}
}
```

customerService.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
```

CustomerServiceImple.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit       : www.jtcindia.org
**/
public class customerServiceImpl implements
CustomerService{
Public void addCustoemr(){
System.out.println("addCustomer-begin");
System.out.println("addCustomer- done");
System.out.println("addCustomer- end");
}
Public void updateCustomer(){
System.out.println("updateCustomer -begin");
System.out.println("updateCustomer - done");
System.out.println("updateCustomer - end");
}
}
```

Security Service.java

```
Package com.jtcindia.spring;
Import org.aspectj.lang.annotation.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit       : www.jtcindia.org
**/
Public class Security Service{
@Pointcut(value="execution(*
Com.jtcindia.spring.AccountService.my*(..))")
```

```
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public interface CustomerService {
Public void addCustomer();
Public void update Customer();
}
```

```
Public void jtc1(){
}
@Pointcut(value="execution(*
Com.jtcindia.spring.CustomerService.my*(..))")
Public void jtc2(){
}
@Before("jtc1()")
Public void verifyUser(){
System.out.println("**SS-verifyUser");
}}
```

TxService.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
@Aspect
Public class Tx Service{
@Pointcut(value="execution(*
Com.jtcindia.spring.AccountService.my*(..))")
Public void jtc1(){
}
@Pointcut(value="execution(*
Com.jtcindia.spring.CustomerService.up*(..))")
Public void jtc2(){
}
@Before("jtc1()")
Public void bigin(){
System.out.println("**ts- bigin");
}
@Before("jtc1()")
Public void commit(){
System.out.println("**ts- commit");
}
@Before("jtc1()")
Public void rollback(){
System.out.println("**ts- rollback");
```

LogService.java

```
Package com.jtcindia.spring;
Import org.aspectj.lang.annotation.*;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
@Aspect
Public class Log Service{
@Pointcut(value="execution(*
Com.jtcindia.spring.AccountService.my*(..))")
Public void jtc1(){
}
@Pointcut(value="execution(*
Com.jtcindia.spring.CustomerService.up*(..))")
Public void jtc2(){
}
@Before("jtc1()") or ("jtc2()")
Public void logbigin(){
System.out.println("**ls- logbigin");
}
@AfterReturning("jtc1()")
Public void logReturning(){
System.out.println("**ls- logReturning");
}
@ AfterReturning ("jtc1()")
Public void logThrowing (){
```

}
 	
 InsufficientFundsException.java
 Package com.jtcindia.spring;
 /*
 *@Author: Som Prakash Rai
 *@Company : java Training Center
 *@ visit : www.jtcindia.org
 **/
 Public class insufficientFundsException extends Exception {}
	System.out.println("**Is- logThrowing");
	 }
	 @After("jtc1()")
	 Public void logOK(){
	 System.out.println("**Is- logOK");
	 }

Jtcindia.xml

```
<?xml version ="1.0" encoding="UTF-8"?>
<beans xmlns=http://www.springgramework.org/schema/beans
       Xmlns:p=http://www.springframework.org/schema/p
       Xmlns:context=http://www.springframewok.org/schema/context
       Xsi:schemaLocation="http://www.springframework.org/schema/aop
       Xmlns:xsi="http://www.w3.org/2001/xmlschema-instance:
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org./schema/context http://www.springframework.org/schema/context/
spring context-3.0.xsd">
context-3.0.xsd http:// www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

<ontext:annotation-config/>
<aop:aspect-autoproxy/>
<bean id="log" class="com.jtcindia.spring.LogService"/>
<bean id="security" class="com.jtcindia.spring.SecurityService"/>
<bean id="ts" class="com.jtcindia.spring.TxService"/>
<bean id="as" class="com.jtcindia.spring.AccountService"/>
<bean id="cs" class="com.jtcindia.spring. CustomerServiceImpl"/>
```

What is happening with Jtc36?

At container start up

1. Spring Container scans and prepares the advice classes which are marked with @Aspect annotation.

2. Spring Container scans and prepares the point cut Expressions specified inside the advice classes.
3. Speing container scans and prepares the methods with are referring point-cut specific special method.
4. Speing container scans and prepares the methods which are marked with join point related annotations (@Before, @AfterReturning….)

When you invoke any business operation

1. Takes the method invoked by you and check whether that method is matching with any of point cut Expressions specified.
2. When method is not matching with any of point cut Expressions given then that method will be called directly without applying any advices.
3. What method is matching with any of point cut Expressions given then specified advices will be applied for that method.

Using @ Around
When you specifying @Around for any method then

- That method has to take proceedingJoinPoint as parameter.
- With proceedingJoinPoint, you have to call proceed () method to invoke the business operation.

Jtc 37:

- Annotation based Aop
- @Around, @After Throwing, @After
- Pointcuts based on annotations with @pointCut
- Autoproxying

Jtc37: Files required

Jtc37.java	AccountService.java
customerService.java	customerServiceImpl.java
LogService.java	SecurityService.java
TxService.java	insufficientFundsException.java
Jtcindia.xml	

Security Service.java

```
Package com.jtcindia.spring;
Import org.aspectj.lang.annotation.*;
Import org.aspectj.lang.proceedingJoinPoint;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
```

```
*@ visit          : www.jtcindia.org
**/
@Aspect
Public class SecurityService {
@Pointcut (value=”execution(*
Com.jtcindia.spring.AccountService.my*(..))”)
Public void jtc1() {
}
@pointcut(value=”execution(*
Com.jtcindia.spring.coustomerService.up*(..))”)
Public void jtc 2() {
}
@Around(“jtc1() or jtc2()”)
Public void verifyUser(proceedingJoinPoint pjp) throws
Throwable {
System.out.println(“**verifyUser begin..”);
Pjp.proceed();
System.out.println(“**verifyUser End..”);
}
}
```

LogService.java	TxService.jave

Package com.jtcindia.spring;
Import org.aspectj.lang.annotation.*;
Import org.aspectj.lang.proceedingJoinPoint;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit : www.jtcindia.org
**/
@Aspect
Public class LogService {
@Pointcut (value=”execution(*
Com.jtcindia.spring.AccountService.my*(..))”)
Public void jtc1() {
}
@pointcut(value=”execution(*
Com.jtcindia.spring.coustomerService.up*(..))”)
Public void jtc 2() {
	Package com.jtcindia.spring;
	Import org.aspectj.lang.annotation.*;
	Import org.aspectj.lang.proceedingJoinPoint;
	/*
	*@Author: Som Prakash Rai
	*@Company : java Training Center
	*@ visit : www.jtcindia.org
	**/
	@Aspect
	Public class txService {
	@Pointcut (value=”execution(*
	Com.jtcindia.spring.AccountService.my*(..))”)
	Public void jtc1() {
	}
	@pointcut(value=”execution(*
	Com.jtcindia.spring.coustomerService.up*(..))”)
	Public void jtc 2() {

```
}
@Around("jtc1() or jtc2()")
Public void verifyUser(proceedingJoinPoint pjp)
throws
Throwable {
System.out.println("**LogService...log()...bigin");
}

Public void logBefore(){
System.out.println("** LS-logBefore");
}
Public void log Returning(){
System.out.println("**LS-logReturning");
}
@AfterThrowing("jtc1() or jtc2()")
Public void logThrowing(){
System.out.println("** LS-logThrowing");
}
@After("jtc1() or jtc()")
Public void logOK() {
System.out.println("**LS...logOk ()...");
}
}
```

```
}
@Around("jtc1() or jtc2()")
Public void runTx(proceedingJoinPoint pjp) throws
Throwable {
System.out.println("**TxService...runTx()...bigin");
Begin();
Pjp.proceed();
Commit();
System.out.println("**TxService....runTx()...end");
}
Public void begin (){
System.out.println("** TS-begin");
}
Public void log commit (){
System.out.println("**TS-commit");
}
@After Throwing("jtc1() or jtc2()")
Public void rollback() {
System.out.println("**TS...rollback ()...");
}
}
```

XML Schema based AOP (2.0)
You must enable AOP namespace to use the following tags.

1. <aop:config>
2. <aop:aspect>
3. <aop:pointcut>
4. <aop:befor>
5. <aop:after-returning>
6. <aop:after-throwing>
7. <aop:after>
8. <aop:around>

Steps
Consider come Business services with some business operations.

- AccountService.java
- CustomerService.java
- CustomerServiceImpl.java.

Define these two business service beans in spring configuration Document.
Identify the Aspects required for your Application.
Ineed security, transaction and Log Aspects.
Write Advice classes for your aspects or middle level services.

- TxService.java
- SecuirtyService.java
- LogService.java

Define these three advices in spring configuration Document.
Indentify the join points required for your business operation.

- Method before
- Method returning
- Method throwing.
- After method exection.

Specify the following in the spring configuration Document.

```
<bean id=”log” class=”com.jtcindia.spring.aop.logService”/>
<aop:config>
<aop:pointcut id=”PC1” expression=”execution(*
Com.jtcindia.spring.aop.AccountService.my*(…))”/>
<aop:aspect ref=”log”>
<aop:before method=”logBefore” pointcut-ref=”PC1”/>
<aop:after-returning method=”logAfterReturing” pointcut-ref=”PC1”/>
<aop:after-throwing method=”logAfterThrowing”pointcut-ref=”PC1”/>
<aop:after method=”logOk” pointcut-ref=”PC1”/>
</aop:aspect>
</aop:config>
```

LogService.java

```
Public class logService{
Public void logBefore(){
        System.out.println(“logBefore-LS”);
}
Public void logAfterReturing(){
        System.out.println(“logAfterReturing-LS”);
}
Public void logAfterThrowing(){
        System.out.println(“logAfterThrowing-LS”);
}
```

```
        Public void logOK(){
                System.out.println("logOK  LS");
        }
        }
```

Get the business service object (It is proxy object by default withoutoproxying concept).
Jtc 38:

- Schema based Aop
- <aop:before>,<aop:after returning>, <aop:after-throwing>,<aop:after>
- Pointcuts based on <aop:point-cut>
- Autoproxying

Lob38: Files required

Jtc38.java	AccountService.java
customerService.java	customerServiceImpl.java
LogService.java	SecurityService.java
TxService.java	insufficientFundsException.java
Jtcindia.xml	

SecurityService.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit         : www.jtcindia.org
**/
Public class security Service {
Public void verifyUser(){
System.out.println("**SS-verifyUser");
}
}
```

TxService.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit         : www.jtcindia.org
**/
Public class TxService {
Public void begin(){
            System.out.println("**TS – begin");
}
Public void commit(){
            System.out.println("**TS-commit"):
}
Public void rollback(){
            System.out.println("**TS-rollback");
}
```

LogService.java

```
Package com.jtcindia.spring;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit         : www.jtcindia.org
**/
```

```
Public class LogServic {
Public void LogBefore(){
            System.out.println("**TS – LogBefore");
}
Public void LogReturning(){
            System.out.println("**TS – LogReturning");
}
Public void LogThrowing (){
            System.out.println("**TS – LogThrowing");
}
Public void logOK(){
            System.out.println("**TS – logOK");
}
}
```

Jtcindia.xml

```
<?xml version ="1.0" encoding="UTF-8"?>
<beans xmlns=http://www.springgramework.org/schema/beans
       Xmlns:p=http://www.springframework.org/schema/p
       Xmlns:context=http://www.springframewok.org/schema/context
       Xsi:schemaLocation="http://www.springframework.org/schema/aop
       Xmlns:xsi="http://www.w3.org/2001/xmlschema-instance:
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
http://www.springframework.org./schema/context http://www.springframework.org/schema/context/
spring context-3.0.xsd">
context-3.0.xsd http:// www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">

<context:annotation-config/>
<aop:aspect-autoproxy/>
<bean id="log" class="com.jtcindia.spring.LogService"/>
<bean id="security" class="com.jtcindia.spring.SecurityService"/>
<bean id="ts" class="com.jtcindia.spring.TxService"/>
<bean id="as" class="com.jtcindia.spring.AccountService"/>
<bean id="cs" class="com.jtcindia.spring. CustomerServiceImpl"/>
```

```
<aop:config>
<aop:pointcut id="myPC1" expression="execution(*com.jtcindia.spring.AccountService.my*(..))"/>
<aop:pointcut id="myPC2" expression="execution(*com.jtcindia.spring.AccountService.my*(..))"/>

<aop:aspect ref="log">
```

```
<aop:before method="logBefore" Pointcut-ref=myPC1"/>
<aop:after-returning method="logReturing" pointcut-ref="myPC1"
</aop:aspect>

<aop:aspect ref="security">
<aop:before method="verifyUser" pointcut="execution(* com.jtcindia.spring.AccountService.my*(..))"/>
</aop:aspect>

<aop:aspect ref="tx">
<aop:before method="begin" pointcut-ref="myPC1"/>
<aop:after-returning method="commit" pointcut-ref="myPC1"/>
<aop:after-throwing method="rollback" pointcut-ref="myPC1"/>
</aop:aspect>

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

What is happening in Jtc38?
At container start-up

1. Spring container scans and prepares the advice classes which are specified/refered with <aop:aspect>
2. Spring container scans and prepares the point-cut Expressions specified with <aop:point-cut>
3. Spring container scans and prepares the methods which are referring point cuts specified with <aop:point cut>
4. Spring container scans and prepares the methods which are specified with join point related tags(<aop:before>,<aop:after>,<aop:after-returnting> etc)

When you invoke any business operation

1. Takes the method invoked by you and check whether that method is matching with any of point cut Expressions specified.
2. When method is not matching with any of point cut Expressions given then that method will be called directly without applying any advices.
3. When method is matching with any of point cut Expressions give then specified advices will be applied for that method.

Using <aop:around>:

When you specifying any method with <aop:around> then

- That method has to take proceedingJoinPoint as parameter.
- With proceedingJoinPoint, you have to call proceed () method to invoke the business operation.

Jtc39:

- Schema based Aop
- <aop:around>,<aop:after throwing> <aop:after>
- Pointcuts based on <aop:point-cut>

- Autoproxying

Jtc37: Files required

Jtc37.java	AccountService.java
customerService.java	customerServiceImpl.java
LogService.java	SecurityService.java
TxService.java	insufficientFundsException.java
Jtcindia.xml	

SecurityService.java	TxService.java

Package com.jtcindia.spring;
Import org.aspectj.lang.proceedingJoinPoint;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit : www.jtcindia.org
**/
Public class security Service {
Public void verifyUser(proceedingJoinPoint pjp) throws
Throwable{
System.out.println("**verifyUser begin...");+
Pjp.proceed();
System.out.println("**verifyUser End..");
}
}
	Package com.jtcindia.spring;
	/*
	*@Author: Som Prakash Rai
	*@Company : java Training Center
	*@ visit : www.jtcindia.org
	**/
	Public class TxService {
	Public void run Tx(proceedingJoinPoint Pjp)throws
	Throwable{
	System.out.println("**TxService..runTx()..begin");
	Begin();
	Pjp.proceed
	Commit();
	System.out.println("**TxService...runTx()...end");
	}
	Public void begin() {
	System.out.println("**TS – begin");
	}
	Public void commit(){
	System.out.println("**TS-commit"):
	}
	Public void rollback(){
	System.out.println("**TS-rollback");
	} }

LogService.java

```
Package com.jtcindia.spring;
Import org.aspectj.lang.proceedingJoinPoint;
/*
*@Author: Som Prakash Rai
*@Company : java Training Center
*@ visit          : www.jtcindia.org
**/
Public class LogService{
Public void log(proceedingjoinPoint pjp)throws
Throwable{
System.out.println("**LogService…log()…bigin");
logBefore();
pjp.proceed();
logReturning();
system.out.println("**LogService…Log()…end");
}

Public void logBefore(){
System.out.println("** LS-logBefore");
}
Public void log Returning(){
System.out.println("**LS-logReturning");
}
@AfterThrowing("jtc1() or jtc2()")
Public void logThrowing(){
System.out.println("** LS-logThrowing");
}
@After("jtc1() or jtc()")
Public void logOK() {
System.out.println("**LS…logOk ()…");
}
}
```

Jtcindia.xml

```
<?xml version ="1.0" encoding="UTF-8"?>
<beans xmlns=http://www.springgramework.org/schema/beans
      Xmlns:p=http://www.springframework.org/schema/p
      Xmlns:context=http://www.springframewok.org/schema/context
      Xsi:schemaLocation="http://www.springframework.org/schema/aop
      Xmlns:xsi="http://www.w3.org/2001/xmlschema-instance:
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
```

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

```
<context:annotation-config/>
<aop:aspect-autoproxy/>

<bean id="log" class="com.jtcindia.spring.LogService"/>
<bean id="security" class="com.jtcindia.spring.SecurityService"/>
<bean id="ts" class="com.jtcindia.spring.TxService"/>
<bean id="as" class="com.jtcindia.spring.AccountService"/>
<bean id="cs" class="com.jtcindia.spring. CustomerServiceImpl"/>

<aop:config>
<aop:pointcut id="myPC1" expression="execution(*com.jtcindia.spring.AccountService.my*(..))"/>
<aop:pointcut id="myPC2" expression="execution(*com.jtcindia.spring.AccountService.my*(..))"/>

<aop:aspect ref="log">
<aop:before method="logBefore" Pointcut-ref=myPC1"/>
<aop:after-returning method="logReturing" pointcut-ref="myPC1"
<aop:before method="logOK" Pointcut-ref=myPC1"/>

</aop:aspect>

<aop:aspect ref="security">
<aop:before method="verifyUser" pointcut="execution(* com.jtcindia.spring.AccountService.my*(..))"/>
</aop:aspect>

<aop:aspect ref="tx">
<aop:before method="begin" pointcut-ref="myPC1"/>
<aop:after-returning method="commit" pointcut-ref="myPC1"/>
</aop:aspect>

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

Q126) What is main goal of AOP?
Ans: AOP is new kind of Programming technique which is mainly used to make the clean separation between commonly required middle level services and business logic of the application.

Q127) what are ways available to implement AOP?
Ans: Refer notes

Q128) what is Aspect?
Ans: Refer notes

Q129) what is Advice?
Ans: Refer notes

Q130) what is joinPoint? What are the join points supported spring AOP?
Ans: Refer notes
Q131) what is pointCut? How to define PointCuts in spring API Based AOP?
Ans: Refer notes

Q132) what is pointCut? How to define pointCuts in spring API Based AOP?
Ans: Refer notes

Q133) write the AspectJ Pointcut Expression for invoking any method from any business service?
Ans: execution (****(..))

Q134) Write the Aspectj Pointcut Expression for invoking any method starting with add from any business service?
Ans: execution(**Service.add*(..))

Q135) what is an Advisor? How to define Advisor in spring API Based AOP?
Ans: Refer Notes.

Q136) what is the difference between target object and Proxy object?
Ans: Refer notes

Q139) what is Explicit Proxying?
Ans: Refer notes

Q140) What properties has to be configured with proxyFactoryBean when you are using class model?
Ans: you need to configure the following three properties.

1. targetClass
2. target
3. interceptorNames

Q141) what properties has to be configured with proxyFactoryBean when you are using interface model?
Ans: You need to configure the following three properties:

1. proxyInterfaces
2. target
3. interceptorNames

Q142) what is Autoproxying?

Ans: Refer Notes

Q143) How proxy object works?

Ans: class AccountServiceImpl extends AccountService{

```
        Void deposit(){
        Try{
                Mba.before();
                As.deposit();
                Mra.afterReturning();
        }catch(Exception e){
                Mta.afterThrowing();
        }}}
        Asproxy.deposit();
```

Here asproxy is an object of AccountServiceImpl.
When invoke the deposit() method on asproxy then overridden deposit() of AccountServiceImpl will be invoked.

Q144) How can I implement AOP using AOP API?

Ans: Using API

MethodBeforeAdvice	before()
AfterReturingAdvice	afterReturing()
ThrowsAdvice	afterThrowing()

Q145) How can I implement AOP using Annotation support?

Ans: using Annotation based AOP(2.0)

@Aspect	@PointCut	@Before	@AfterReturning
@AfterThrowing	@After	@Around	

Q146) How can I implement AOP using schema Support?

Ans: you must enable aop namespace to use the following tags.

Q147) what is the use of ProceedingJoinPoint?

Ans: Refer notes

Q148) what is the difference between @AfterRetutning and @After?

Ans: Refer notes

Q149) what is the use of@Aspect annotation?

Ans: Refer notes

Q150) what are GOF design Patterns discussed in AOP?

Ans: proxy used heavily in AOP