AOP:- aspect oriented programming

That we are having a class with different methods

Say our class name is UserBankAccount

Deposit,

Withdrawal

Checkbalance

Transfer

Minbalancealert

Each when we called one of the method we try to check that user is logged in or it should be authorize to access the account

We can say only cashier, and one person who clear the cheque deposit/withdrawal can the balance of that particular customer.

And other can check the details like name, aadhar no pan card no

But a bank manager can check each and everything of any customer.

objUBA.deposit()

inside our deposit method what I need to do

I need to check, is user logged in

Public boolean deposit()

{

Checkloggedin();

}

When we add new methods in that case i need to always put the method checklogged in inside our methods

And if I need to put more functions before the access of main methods of operations I need to put each and every methods call inside the operation methods like deposit, withdrawal etc.

Deshraj is going to deposit the fund to Sandeep Gupta: so removed the method checkloggedin from deposit.

This is going to be very much fix in nature. It hard coded.

We just creats some methods which are going to be called when we are about to call main logic or business logic in our application.

AOP Provides following concepts and terminology

* **Join Point**
* **Advice**
* **Pointcut**
* **Introduction**
* **Target Object**
* **Aspect**
* **Interceptor**
* **Auto Proxy**
* **Weaving**

**Join point basically Is such a method execution or exception handling or field access.**

**Advice :- we call a method before, after, in between the execution of the actual business logic**

**Pointcut:- it is an expression where we called a aop that matches the join point**

**@Pointcut(execution( classname.nameofmethod or we can provide initials of methods like we can do in this way execution( Test.m\*(..)) I have mention the point cut for all methods of Test which start with m**

**Public void pointcutMethods(){}**

**@After(“pointcutMethods()”)**

**Public returntype methodname(){}**

**JointPoint which helps us in finding which method is called upon**

**We can also pass the object of ProceedingJointPoint**

**Introduction :- is declaring new class, methods, interfaces, fields in our aspects**

**Target Object:- it is going to be monitor, but in aop we say that target is being advised by one or more aspects.**

**Aspect:- it declare a class as aspect and method of class is going to be called upon execution of another class.**

**Interceptor**

**It only contain one advice.**

**Auto Proxy:- it is used to implements aspect contracts,**

**Weaving:- we can link two different application advised object to create new advised object.**

**From Spring 4 we have annotation based AOP**

**Means we need to declare some annotation to declare a pointcut etc.**

**Aspectj**

**Spring AOP**

**JBoss AOP**

**We use here to AspectJ along with Spring AOP**

**Spring boot it use full annotation for all of AOP no xml configuration is going to be used.**

**@Aspect :- it is use to declare a class as Aspect**

**@Pointcut:- it is use to declare point cut on methods of a class.**

**@Before :- It declare a advice before execution of main business logic**

**@After :- to be called when execution of main business logic is over**

**@AfterReturning :- it will called when a method executed and return the value**

**@Around:- it will be called before and after execution of Business logic**

**@AfterThrowing:- use to call when any method throws any exception.**

**How we apply point cut in our application**

**@PointCut(“Execution(public \*\* (..))”) it is going to apply on all public methods of all classes.**

**@PointCut(“Execution(public UserProfile.\* (..))”) it is going to be apply on each methods which are public in class userprofile**

**@PointCut(“Execution(\* UserProfile.\* (..))”) it is going to apply on each methods of userprofile**

**@PointCut(“Execution(public Userprofile.set\* (..))”) it is going to apply on setter method which are public**

**@PointCut(“Execution(returntype classname. \* (..))”)**

It is going to apply on methods which are going to return any value.

@PointCut(“execution(String userProfile.\*(..))”) this will only called when any method is returning string

@Around is use to call on or before of the any method calling

@AfterThrowing()

Com.yash.projectname:- com.yash.ems

Com.yash.ems.bl

Com.yash.ems.exception

@Pointcut(“execution(\* classname)

Aop it is not testing

You can see that we are having so many of transection done at point of time

@Around annotation which will call a method manageTransection()

This will put a mark that payment is going to transfer from one account to another one

After completion we use to mark transection done

If in any case transection is not completed by any mean it means light is gone, network issue, server crash, harddisk crash all crash points are managed