Aspect Oriented Programming

What the heck is AOP?

Aspect oriented programming is a programming technique just like object oriented programming where you group codes and software designs into objects rather than keeping it as separate functions and logic.

AOP aim to help mitigate and solve the problem of cross-cutting concerns. Cross-cutting concerns are any logic that are not part of your main business logic that are repeated thus you can extract them out into it's own module.

For example: Say you have a class, User that suppose to facilitate the CRUD operation between the databases for a user. You have methods that does the creation, update, read, and delete. Now your manager wants you to add some logging functionalities and/or authentication capability before the operation is carried out. You could hard code them in method by method if it isn't many. But imagine you have hundreds of these classes, hard coding them in those logics are just not going to cut it.

Besides, those logging functionality and authentication are NOT part of the main business logic. Your main business logic is just CRUD operations in this case. So these are cross-cutting concerns they are extra functionality that you are adding to mingle with the main business logic.

AOP aims to solve that problem by extracting those concerns (**either logging or authentication, really any extra logic or functionality**) out from your main business logic into something called an Aspect.

Then you can just use Aspect whenever you need them.

Terminology

I'm going to now define some of the jargon terminology that's used in AOP :)

JoinPoint

A joinpoint is just anywhere you can intercept / insert an Aspect (functionality such as logging, authentication) in. Methods being called, an exception beign thrown, or even a field being modified. Basically where you can insert the extracted concerns into in your application for new behaviors.

Concrete examples would be:

```
class Employee{
    public String getName(int id){....}
    private int getID(String name){...}
}
```

Here getName and getID would be considereed JoinPoint because they are candidate methods for where you can insert aspects, either before execution, after execution, after return of the method, or after an exception is thrown from the method.

Pointcut

So joinpoint is basically where you can insert the aspects, pointcut defines at which particular joinpoint you would like to invoke the aspect with the associated advice.

Say if you define something like below:

```
@Before("execution(* *.*.get*())")
public void beforeGets() {
  System.out.println("I'm getting");
}
```

First line is a pointcut along with the advice on when the logic of the aspect should be applied. The pointcut being on all get* method invocations, advice being that the logic should be carried out before the execution of the pointcuts.

So whenever you call say `getId`, it will print "I'm getting" before the actual `getId` method is called.

> You can think of pointcuts being a regex expression to match what you would like the aspect to be applied on.
> Advice being when you would like to take the action (logic of the aspect) when you have a pointcut.

Advice

Used together with pointcuts to define WHAT logic you would like to execute and WHEN you would like to execute on the pointcut (matched places).

There are different advice types:

1. @Before: Runs before the execution of the joinpoint method
2. @After: Runs after the joinpoint method finishes executing, normally or when throwing an exception
3. @AfterReturning: Runs after the method returned
4. @AfterThrowing: Runs after the method throws an exception

5. @Around: Most powerful advice. You can choose whether or not to execute the jointpoint method or not. Write advice code that executes both before and after the execution.

Big caviar

If your method that's being intercepted by Spring Boot's Aspect, and you call another method within it and expect it to be also be intercepted, it will NOT.

Aspect's method interception only works on method that's invoked ACROSS class. If you invoke method within the same class, it will not trigger the interception. Meaning those are not valid Pointcut!

https://stackoverflow.com/questions/27635310/spring-aspect-not-working-when-called-by-method-of-the-same-class

---

Revision #5
Created 2023-06-05 20:01:38 UTC by Tamarine
Updated 2023-10-09 00:09:13 UTC by Tamarine