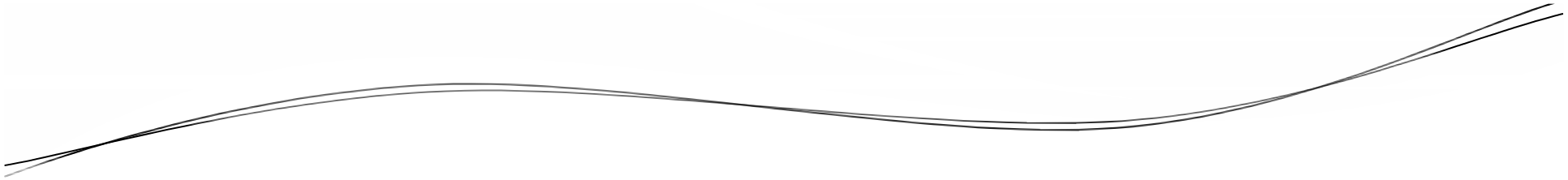


AOP

Objectives

- Understand Aspect Oriented Programming.
- Why AOP
- AOP Terminologies
- Weaving
- Implementing AOP

What is AOP

What is AOP

- AOP is a programming model that promotes separation of business logic from cross-cutting concerns.

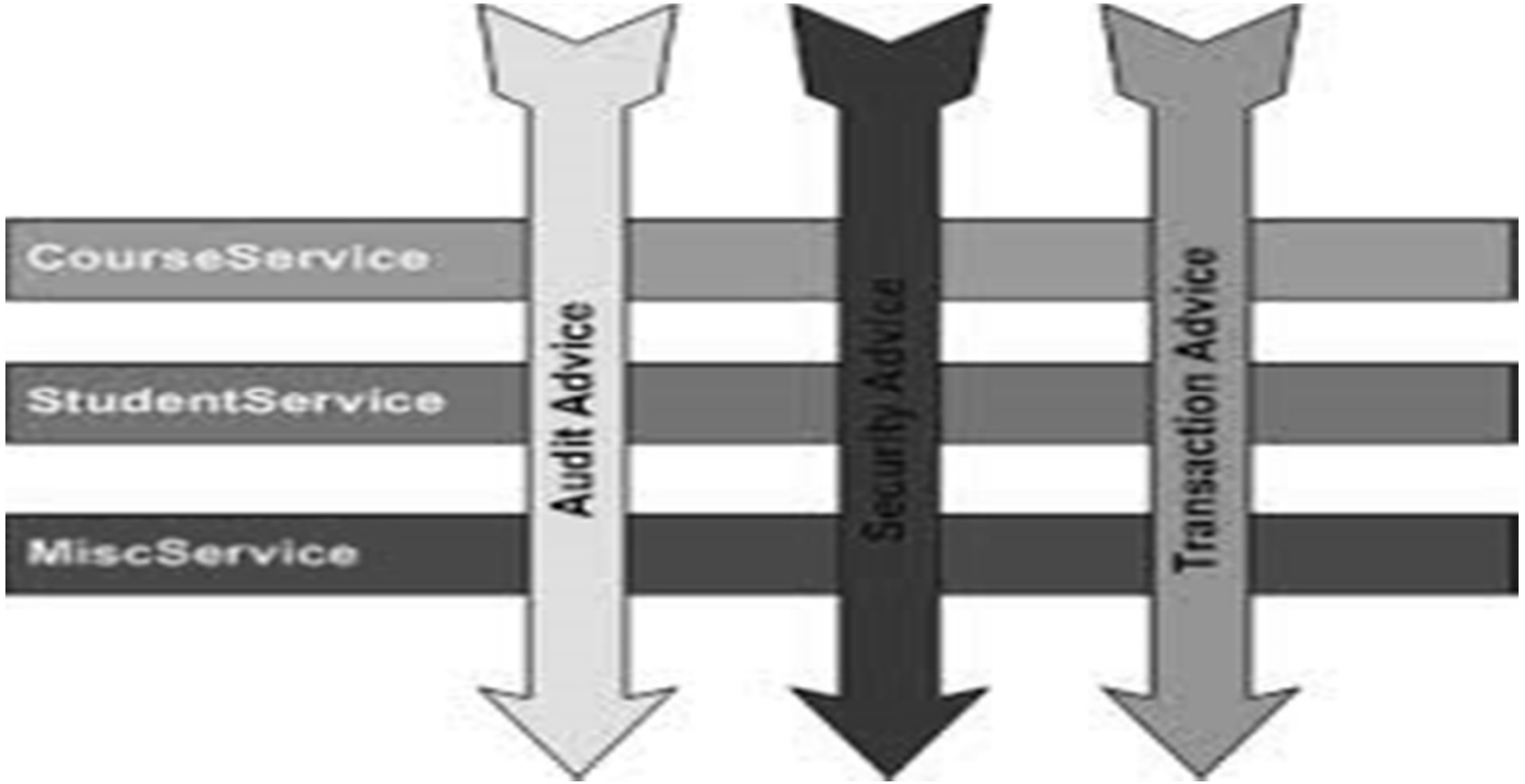
What is AOP

- A cross-cutting concern can be described as any functionality that affects multiple points of an application.

What is AOP

- These cross-cutting concerns can be modularized into special objects called Aspects.

AOP – Big Picture



Benifits

- Aspects offer an alternative to inheritance and delegation that can be cleaner in many circumstances.
- Secondary concerns are decoupled from primary concerns.
- Allows developers to focus upon business logic.

AOP Terminology

AOP Terminology

- There are 5 terms involved in Spring AOP:
  - Advice
  - Joinpoint
  - Pointcut
  - Aspect
  - Weaving

AOP Terminology

- Advice
  - Aspects have a purpose, a job they are meant to do.
  - The job of an aspect is called advice.
  - It defines both: WHAT and WHEN of an aspect.

AOP Terminology

- JoinPoint
  - A joinpoint is a point in the execution of the application where an aspect can be plugged in.

AOP Terminology

- Pointcut
  - If advice defines WHAT and WHEN of an aspect, then pointcut defines WHERE.
  - A pointcut definition matches one or more joinpoints at which advice should be woven.

AOP Terminology

- Aspect
  - An aspect is the merger of advice and pointcut.
  - It specifies what to do, when to do and where to do.

Weaving

- Weaving is the process of applying aspects to a target object to create a new, proxied object.
- The aspects are woven into the target object at the specified join points.

Weaving

- Weaving can take place at three different points:
  - Compile time
  - Classload time
  - Runtime

Weaving

- Compile time
  - Aspects are woven in when the target class is compiled.

Weaving

- Classload Time
  - Aspects are woven in when the target class is loaded into the JVM.
  - This requires a special class loader that enhances the byte code before the class is introduced to the application.

Weaving

- Runtime
  - Aspects are woven in sometime during the execution of the application.
  - Typically, an AOP container will dynamically generate a proxy object that will delegate to the target object while weaving in the aspects.
  - This is how Spring AOP aspects are woven.

Spring AOP support

Spring AOP support

- Spring Advices are written in Java.
- Spring applies an Advice to an Object at runtime, by wrapping them with a proxy class.

Spring AOP support

- Between the time that the proxy intercepts the method call and the time it invokes the target bean's method, the proxy performs the aspect logic.
- Spring only supports method joinpoints.

Types of Spring AOP Advices

Types of Spring AOP Advices

- Before Advice
- After returning Advice
- After throwing Advice
- After Advice
- Around Advice

Before Advice

- Causes a method to be invoked before the invocation of a target method.

After Returning Advice

- Causes a method to be invoked after successful invocation of a target method.

After Throwing Advice

- Causes a method to be invoked if an exception is occurred during an execution of the target method.

After Advice

- Causes a method to be invoked irrespective of whether a target method completes successfully or not.

Around Advice

- A single advice that wraps up all types of advices.
- Provides a single method through which other advice methods are invoked.

Enabling Proxying

Enabling Proxying

- Since in AOP, spring performs weaving with the help of proxies, it's necessary to enable Proxy in the application.

Enabling Proxying

- To enable proxy, the configuration specific class must be annotated with `@EnableAspectJAutoProxy`.

Spring's AOP Annotations

Spring's AOP Annotations

- Spring comes with Annotation Based Support, through AspectJ annotations for AOP.

Spring's AOP Annotations

- AOP Annotations:
  - `@Aspect`
  - `@Before`
  - `@AfterReturning`
  - `@AfterThrowing`
  - `@After`
  - `@Pointcut`
  - `@Around`

@Aspect

- Applied at the class level.
- Tells Spring that the class is not just a POJO, rather it's an aspect.
- Classes annotated with `@Aspect` will only be considered as *Aspects*.

@Before

- Applied at the method level.
- Annotated methods will be called before the execution of advised method.

@AfterReturning

- Applied at the method level.
- Annotated methods will be called after the successful return of the advised method.

@AfterThrowing

- Applied at the method level.
- Annotated methods will be called if some exception is raised during the execution of the advised method.

@After

- Applied at the method level.
- Annotated methods will be called after the execution of the advised method, irrespective of whether the advised method returns successfully or not.

@Pointcut

- Used to define a pointcut.
- Applied at the method level to mark the method as a pointcut.
- The marked method can be used for further referencing.

@Around

- Applied at the method level to mark the method as a an Around Advice.
- Uses a Pointcut expression to apply the aspect.

Let's Summarize

- What is AOP
- Why AOP
- AOP Terminologies
- Types of Advices
- Weaving
- Implementing AOP