Spring Aspect Oriented Programming with Proxy and Decorator pattern

CHAPTER 4

PRESENTATION BY:
RMN

spring

Chapter Four

We will cover these skills

- Proxy pattern in Spring
- Decorator design pattern
- Aspect-oriented programming
- Problems resolved by AOP
- Core AOP concepts
- Advices Type

Proxy
Pattern in Spring

Proxy design pattern provides an object of class that has the functionality of another class.

Provide a surrogate or placeholder for another object to control access to it.

The intent of this design pattern is to provide a different class for another class with its functionality to the outer world.

Decorator
Pattern and CGLIB

Decorator Pattern:

- Attach additional responsibilities to an object dynamically.
- Decorators provide a flexible alternative to subclassing for extending functionality.
- This pattern allows you to add and remove behaviors to an individual object at the runtime dynamically or statically without changing the existing behavior of other associated objects from the same class.

CGLIB:

- In Spring AOP, CGLIB is used to create the proxy in the application.
- CGLIB proxying works by generating a subclass of the target class at runtime.

Cross-Cutting Concerns

In any application, there is some generic functionality that is needed in many places.

But this functionality is not related to the application's business logic.

The following are examples of the cross-cutting concerns for the enterprise application:

- Logging and tracing
- Transaction management
- Security
- Caching
- Error handling
- Performance monitoring
- Custom business rules

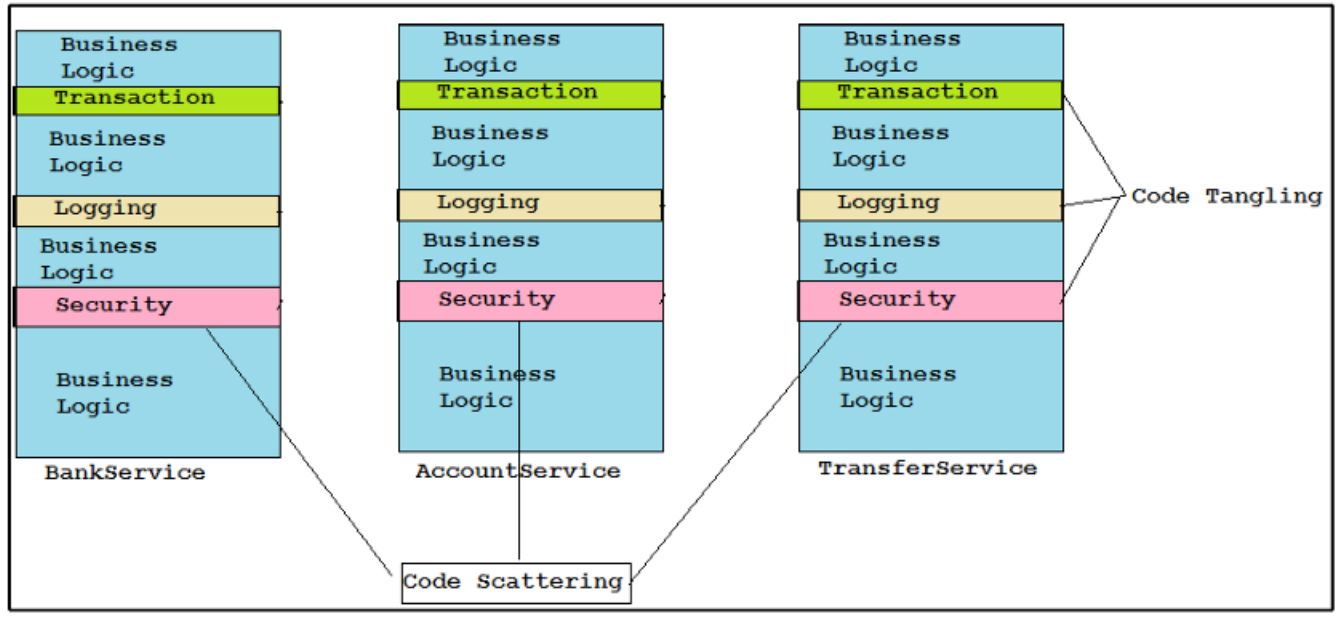
Problems
Resolved by AOP

Code tangling:

- It is a coupling of concerns in the application.
- Code tangling occurs when there is a mixing of cross-cutting concerns with the application's business logic.

Code scattering:

- This means that the same concern is spread across modules in the application.
- Code scattering promotes the duplicity of the concern's code across the application modules.

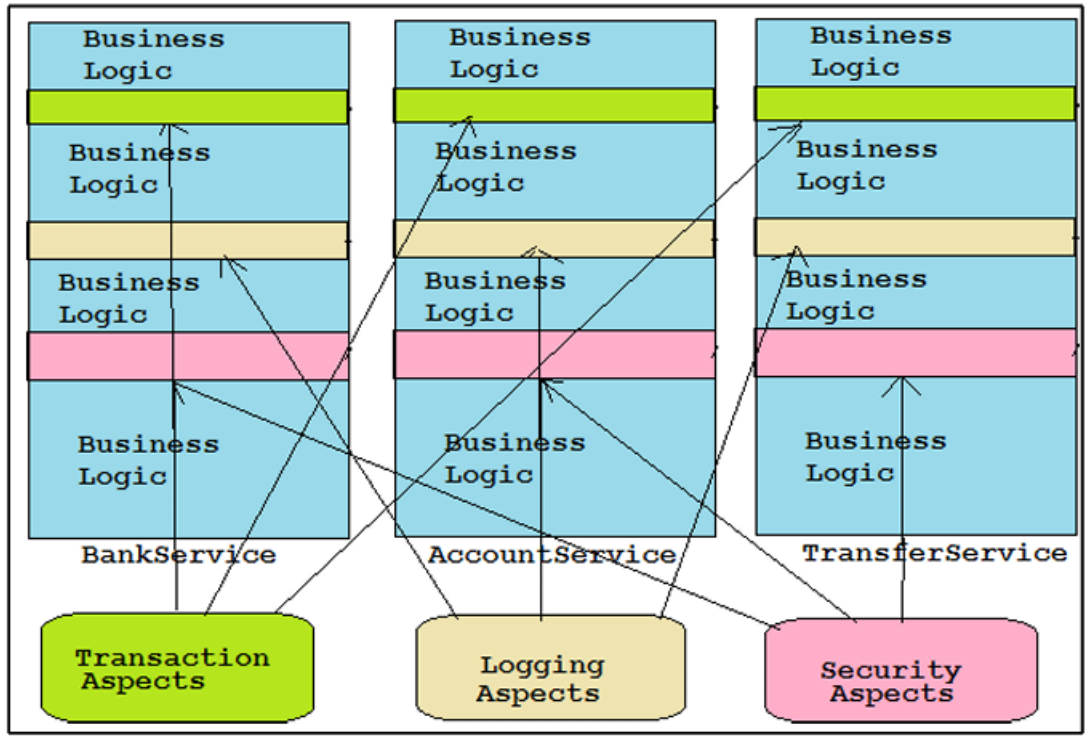


How
AOP Works

Spring AOP allows you to keep cross-cutting concern logic separate from the mainline application logic.

That means, you can implement your mainline application logic and only focus on the core problem of the application.

And you can write aspects to implement your cross-cutting concerns



Core
AOP Terminology

Advice: The job of an aspect is known as advice in the AOP.

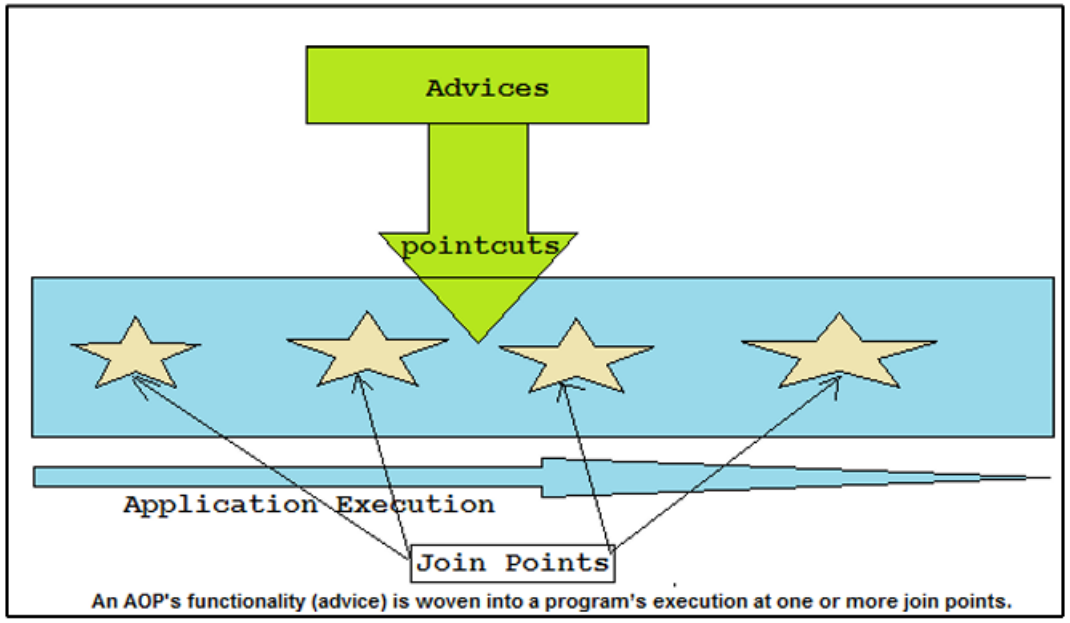
Join Point: A join point is a point in the execution of a program such as a method call or exception thrown.

Pointcut: You can define an expression that selects one or more Join Points in the application. This expression is known as pointcut.

Aspect: In your application, an aspect is a module that encapsulates pointcuts and advice.

Weaving:

- Weaving is a technique by which aspects are combined with the business code.
- This is a process of applying aspects to a target object by creating a new proxy object



Advices Type

Before: Advice's job executes before the advised method is invoked.

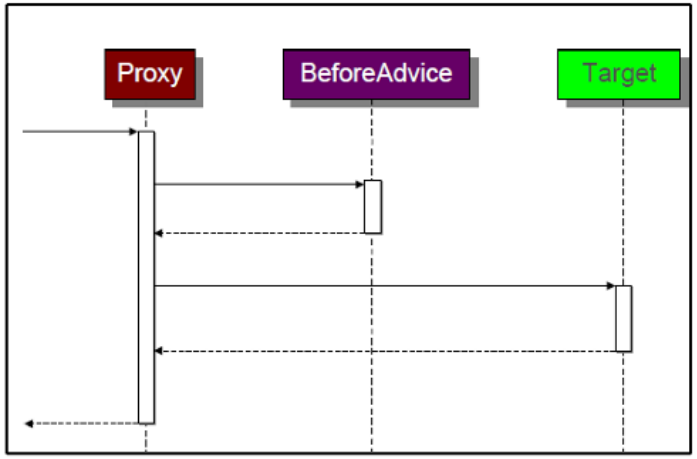
After: Advice's job executes after the advised method completes regardless of whether an exception has been thrown by the target or not.

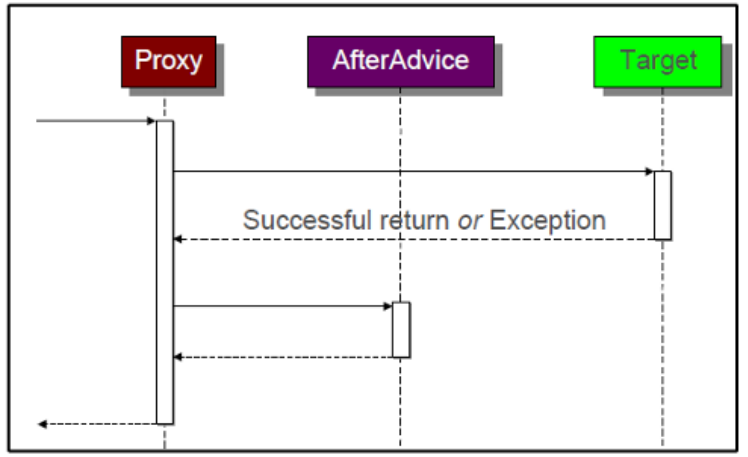
After-returning: Advice's job executes after the advised method successfully completes.

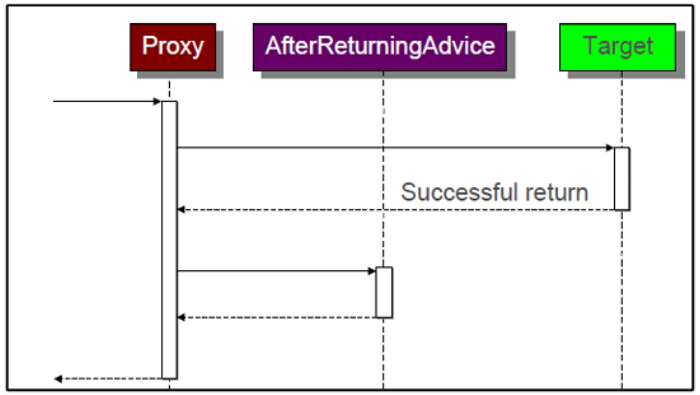
After-throwing: Advice's job executes if the advised method exits by throwing an exception.

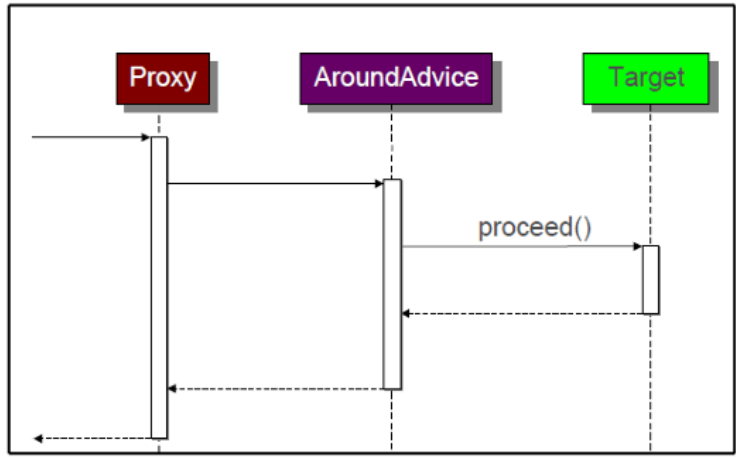
Around:

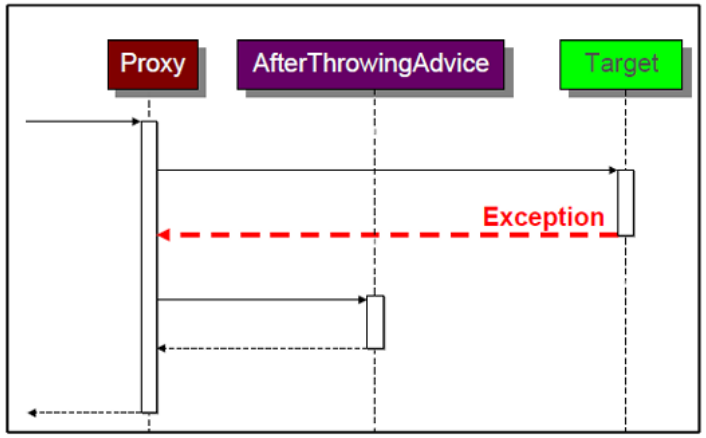
- This is one of the most powerful advice of Spring AOP, this advice surrounds the advised method, providing some advice's job before and after the advised method is invoked.











Thank you

RMN