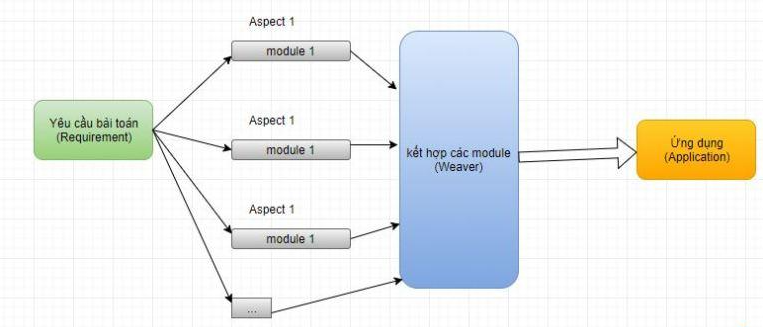
**SPRING AOP**

**1. Lập trình AOP**

**AOP là gì?**

* Aspect Oriented Programming (AOP) là một kỹ thuật lập trình cho phép phân tách chương trình thành cách module riêng rẽ, không phụ thuộc nhau. Khi hoạt động, chương trình sẽ kết hợp các module lại để thực hiện các chức năng nhưng khi sửa đổi chức năng thì chỉ cần sửa đổi trên một module cụ thể
* AOP không phải dùng để thay thế OOP mà để bổ sung cho OOP.



**Ví dụ**

* Ví dụ 1:

Trong code chúng ta thường thực hiện viết log cho các method (method nào chạy, lỗi gì xảy ra…). Rõ ràng việc viết log chẳng liên quan gì đến chức năng của method.

Bây giờ yêu cầu chuyển sang chỉ viết log khi bắt đầu các method chẳng hạn, ta lại phải sửa tất cả các method đó.

Bây giờ áp dụng AOP, ta sẽ phân tách chức năng log ra một module riêng và dùng nó thực hiện các lát cắt với các method cần thực hiện log.

Muốn log ở đầu method, điểm cắt sẽ là ở đầu các method, muốn cắt ở cuối method thì điểm cắt sẽ là cuối method.

* Ví dụ 2:

Trong lập trình web, có 1 khái nhiệm filter, tức là tất cả các request được chọn sẽ đều chạy qua filter đó.

filter ở đây cũng được coi như 1 module độc lập riêng rẽ. Chẳng hạn mỗi request gửi đến cần xác thực (kiểm tra login) ta chỉ cần cho nó đi qua 1 filter thực hiện xác thực là được, muốn sửa đổi cách xác thực thì ta chỉ cần sửa filter là xong.

**Ưu nhược điểm của AOP**

* Ưu điểm:
  + Thiết kế đơn giản: “You aren’t gonna need it (YAGNI)” – chúng ta chỉ cài đặt những thứ chúng ta thực sự cần mà không bao giờ cài đặt trước.
  + Cài đặt chương trình một cách trong sáng: mỗi một module chỉ làm cái mà nó cần phải làm, giải quyết được hai vấn đề code tangling và code scattering.
  + Tái sử dụng dễ dàng.
* Nhược điểm:
  + Khái nhiệm khá trừu tượng, độ trừu tượng của chương trình cao
  + Luồng chương trình phức tạp.

**Các thuật ngữ trong AOP**

* Core concern/ Primary concern: là requirement, logic xử lý chính của chương trình.
* Cross-cutting concern: là những logic xử lý phụ cần được thực hiện của chương trình khi core concern được gọi như security, logging, tracing, monitoring, …
* Joinpoint: là một điểm trong chương trình, là những nơi có thể được chèn những cross-cutting concern. Chẳng hạn chúng ta cần ghi log lại sau khi chạy method nào đó thì điểm ngay sau method đó được thực thi gọi là một Jointpoint. Một Jointpoint có thể là một phương thức được gọi, một ngoại lệ được throw ra, hay một field được thay đổi.
* Pointcut: Là expression language giúp khớp nối với joinpoint.
* Advice: là một hành động thực hiện ở joint point. Advice được chia thành các loại sau:
  + Before: được thực hiện trước join point.
  + After: được thực hiện sau join point.
  + Around: được thực hiện trước và sau join point.
  + After returning : được thực hiện sau join point hoàn thành một cách bình thường.
  + After throwing : được thực hiện sau khi join point được kết thúc bằng một Exception
* Aspect: đóng gói toàn bộ cross-cutting concern và có thể chứa các JointPoint, PointCut, Advice.
* Target Object : là những đối tượng mà advice được áp dụng.

**Một vài cross-cutting concern thường thấy trong ứng dụng**

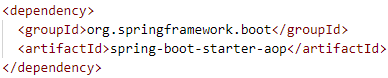
* Logging
* Monitor
* Access control
* Error handling
* Transaction management
* Session management
* Input/output validation

**Thực hiện AOP trong Java**

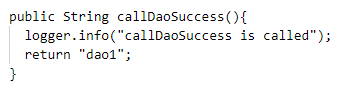
Để thực hiện AOP trong Java ta có thể sử dụng các cài đặt cung cấp bởi: AspectJ, Spring AOP, JBoss AOP

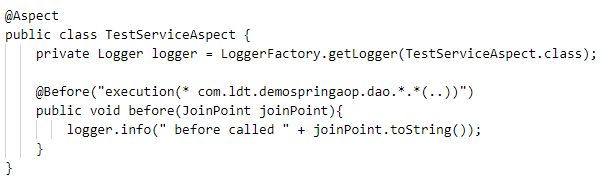
**2. Spring AOP**

* Dependency:



* Demo:





Khi chạy ta sẽ thấy 2 log được sinh ra 1-> Log sinh ra tư Aspect, nó gọi trước khi gọi method 2-> Log sinh ra trong service



* Các annotation:
  + @Before: Thực hiện trước khi phương thức mục tiêu bắt đầu.
  + @After: Thực hiện sau khi phương thức mục tiêu đã hoàn thành (bất kể thành công hay thất bại).
  + @AfterReturning: Thực hiện sau khi phương thức mục tiêu đã thành công.
  + @AfterThrowing: Thực hiện sau khi phương thức mục tiêu ném ra exception.
  + @Around: Thực hiện trước và sau khi phương thức mục tiêu được thực hiện. Nó bao quát cả quá trình thực hiện của phương thức