**Xử lý bất đồng bộ và Logging**

1. **Xử lý bất đồng bộ trong Spring Boot**
2. **Đồng bộ/Bất đồng bộ**
3. **Đồng bộ (Synchronous)**

**-** Các tác vụ được thực thi tuần tự, giữ cho các tài nguyên không bị sai lệch trong chương trình đa luồng.

- Quy tắc của đồng bộ: một bộ phận tài nguyên bị khóa và chỉ cho phép một luồng sử dụng trong một thời điểm.

- Ưu điểm:

* Đơn giản, dễ triển khai vã gỡ lỗi.
* Tính dự đoán: dễ dự đoán thời gian thực thi các yêu cầu.

- Nhược điểm:

* Hiệu suất thấp nếu nhiều tác vụ cần nhiều thời gian.
* Không phù hợp với ứng dụng yêu cầu hiệu suất cao .

1. **Bất đồng bộ (Asynchronous)**

- Trái ngược với đồng bộ. Các tác vụ được thực thi song song trong các luồng riêng biệt.

- Ưu điểm:

* Nâng cao hiệu suất
* Tối ưu sử dụng tài nguyên

- Nhược điểm:

* Phức tạp
* Khó khăn trong quản lý trạng thái và gỡ lỗi
* Không phù hợp với các hệ thống yêu cầu sự chính xác và tuần tự cao

1. **Áp dụng bất đồng bộ**

- Annotation **@Async** có thể áp dụng trên lớp hoặc phương thức.

- Khi **@Async** được đánh dấu trên lớp, tất cả các phương thức bên trong lớp đều được thực thi bất đồng bộ.

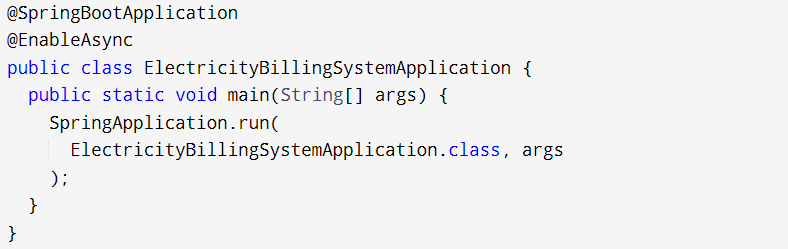
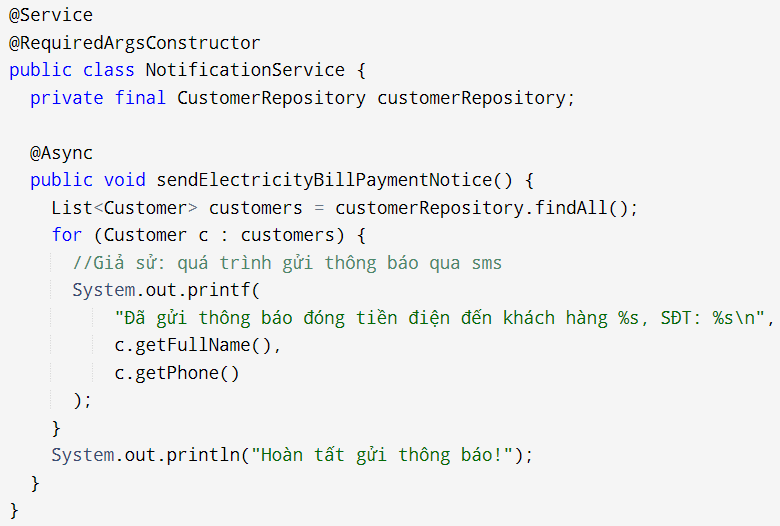
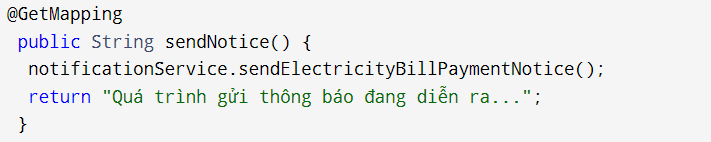
- Khi **@Async** được đánh dấu trên phương thức cụ thể, giúp kiểm soát tốt hơn phương thức nào nên được thực thi bất đồng bộ.

- Không thể gọi phương thức bất đồng bộ từ cùng một lớp: khi gọi từ cùng một lớp, nó sẽ không thực thi bất động bộ (proxy chỉ được kích hoạt khi gọi từ bên ngoài lớp).

1. **Async, Scheduling trong xử lý bất đồng bộ**
2. **Async**

- Annotation **@Async** cho phép chạy các tác vụ bất đồng bộ trong ứng dụng.

- Để sử dụng, cần:

* Kích hoạt async với **@EnableAsync** trong lớp chính hoặc lớp cấu hình: 
* Sử dụng **@Async** với phương thức xử lý bất đồng bộ: 
* Gọi phương thức được đánh dấu bằng **@Async** để chạy các tác vụ bất đồng bộ: 

1. **Scheduling**

- Annotation **@Scheduled** cho phép định nghĩa các phương thức thực hiện các tác vụ theo lịch trình.

- Các tác vụ có thể được lên lịch với các tham số khác nhau của **@Scheduled** như:

* **fixedRate**: khoảng thời gian giữa các lần chạy, tính từ khi tác vụ bắt đầu 
* **fixedDelay**: khoảng thời gian giữa các lần chạy, tính từ khi tác vụ trước đó kết thúc 
* **cron**: kiểm soát chi tiết thời gian thực hiện tác vụ (ví dụ: mỗi ngày, mỗi tuần,...) 

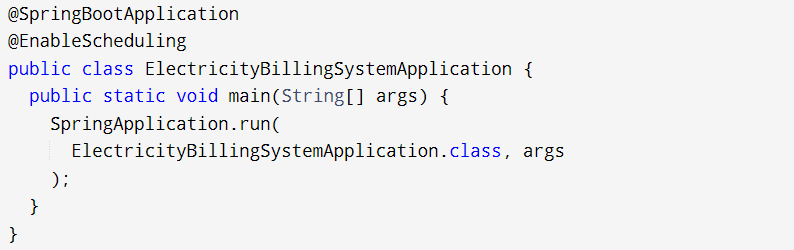
- Cấu trúc biểu thức **cron** trong lập lịch:

***second minute hour day\_of\_month month day\_of\_week***

trong đó:

* second: giây, giá trị từ 0 đến 59 (thứ tự giây trong một phút)
* minute: phút, giá trị từ 0 đến 59 (thứ tự phút trong một giờ)
* hour: giờ, giá trị từ 0 đến 23 (thứ tự giờ trong một ngày)
* day\_of\_month: ngày của tháng, giá trị từ 1 đến 31
* month: tháng của năm, giá trị từ 1 đến 12 hoặc JAN - DEC
* day\_of\_week: ngày của tuần, giá trị từ 0 đến 6 hoặc SUN - SAT

- Để sử dụng, cần:

* Kích hoạt scheduling với annotation **@EnableScheduling** trong lớp chính hoặc lớp cấu hình: 
* Sử dụng **@Scheduled** để lập lịch các tác vụ: 

1. **Kết hợp cả @Async và @Scheduled**

- Việc kết hợp cả hai annotation này giúp tối ưu hóa và quản lý tốt hơn các tác vụ trong ứng dụng.

- Phù hợp với các tác vụ bất đồng bộ và cần được lên lịch để chạy.

- Đánh dấu cả annotation **@Async** và **@Scheduled** với các phương thức để lập lịch các tác vụ chạy bất đồng bộ.

- Ví dụ: 

1. **Tối ưu và quản lý logging trong Spring Boot (Logback, SLF4J)**

- Tối ưu và quản lý logging giúp đảm bảo hiệu suất ứng dụng, dễ dàng theo dõi và phát hiện lỗi.

1. **Logback**

- Thư viện mặc định trong Spring Boot, có tính linh hoạt và hiệu suất cao.

**\* Phân cấp mức độ log**

- Phân cấp log giúp:

* Kiểm soát mức độ chi tiết
* Giảm tải hệ thống
* Dễ dàng theo dõi và phát hiện lỗi

- Các mức độ log chính:

* **TRACE**: chi tiết nhất, ghi lại toàn bộ hành động của hệ thống
* **DEBUG**: dùng để ghi lại các thông tin hữu ích khi phát triển và gỡ lỗi
* **INFO**: ghi lại các sự kiện hoặc hành động bình thường của hệ thống, được sử dụng phổ biến (đặc biệt: production)
* **WARN**: ghi lại những sự cố có thể xảy ra nhưng không ngăn ứng dụng tiếp tục chạy (lỗi nhẹ, chưa ảnh hưởng lớn đến ứng dụng).
* **ERROR**: ghi lại các lỗi nghiêm trọng làm cho ứng dụng không thể hoạt động bình thường.
* **OFF**: Vô hiệu hóa hoàn toàn việc ghi log cho một logger.

**\* Quản lý logging theo môi trường**

- Cấu hình logging cho từng môi trường bằng cách sử dụng các profile của Spring Boot.

- Ví dụ:

* Môi trường phát triển (application-dev.properties): 
* Môi trường sản xuất (application-prod.properties): 

1. **SLF4J - Simple Logging Facade for Java**

- Cung cấp giao diện thống nhất cho các framework logging, cho phép thay đổi giữa các hệ thống logging mà không cần thay đổi mã nguồn.

- SLF4J được dùng như một lớp trung gian giữa mã nguồn và hệ thống logging (Logback).

- **MDC - Mapped Diagnostic Context -** là một tính năng trong SLF4J cho phép thêm thông tin ngữ cảnh vào log, giúp theo dõi theo từng yêu cầu cụ thể.

1. **Tối ưu và quản lý logging**

- Sử dụng SLF4J với Logback, dùng annotation **@Slf4j** của Lombok giúp giảm thiểu mã nguồn.

- Ghi log ở dạng JSON để dễ dàng trong theo dõi hệ thống.

- Sử dụng log đúng mức độ.

- Cấu hình log bên ngoài, tách biệt với mã nguồn: cấu hình thông qua application.properties hoặc logback-spring.xml với các cấu hình phức tạp hơn.

- Ghi log thông tin có ý nghĩa.