예외와 예외 처리

in Java

박연종

2025.11.19.

System Software Lab.

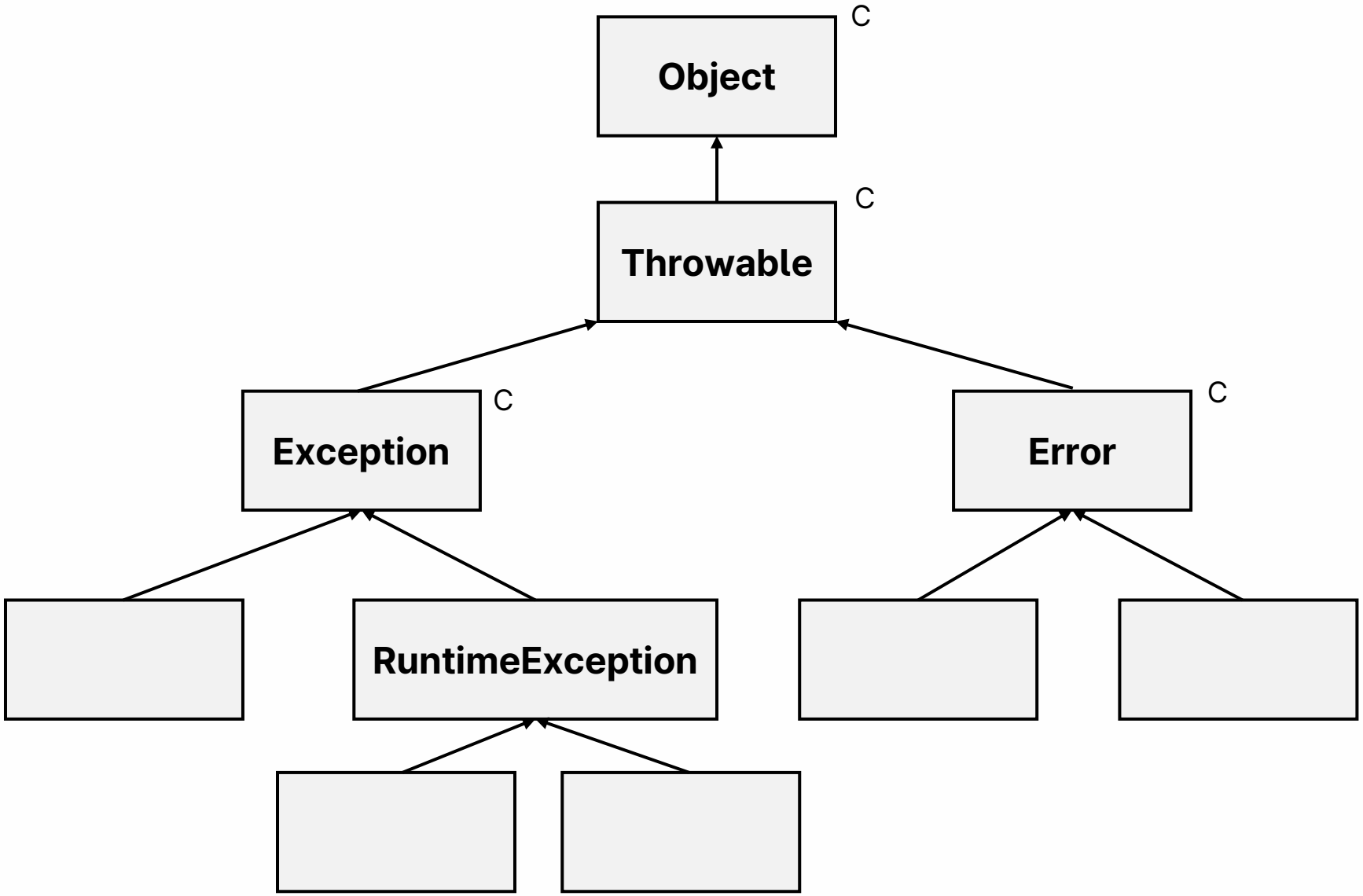
오늘 할 이야기

01 Error와 Exception

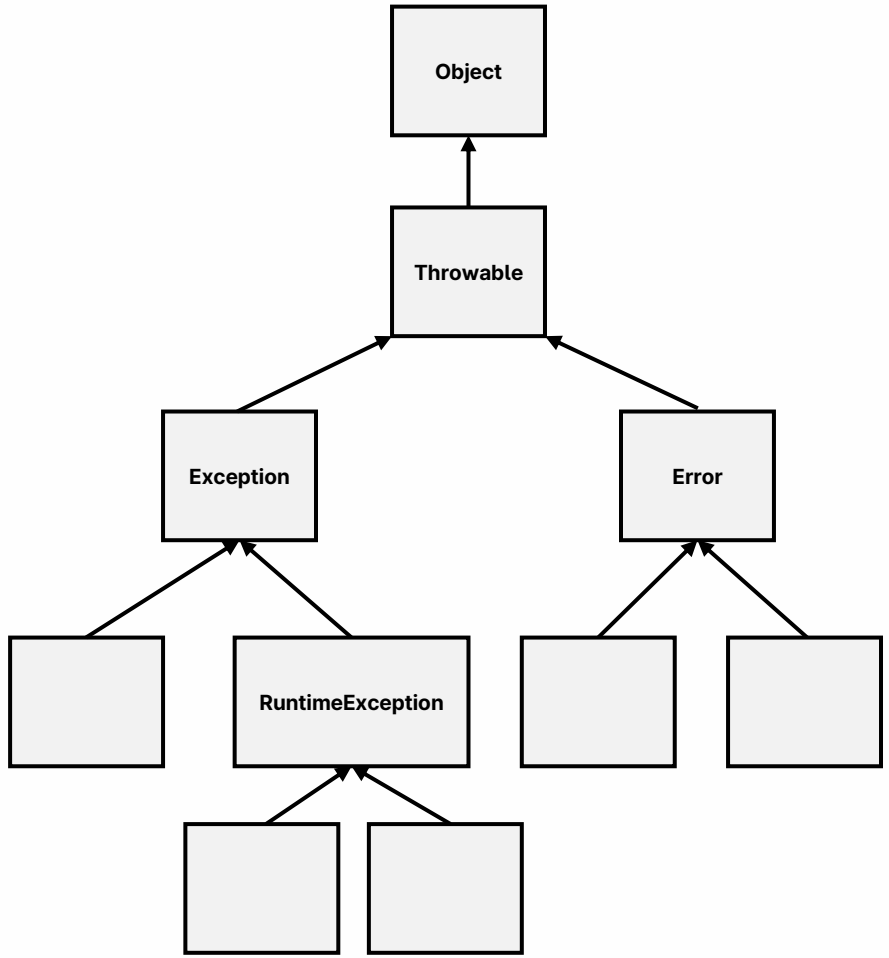
02 Checked Exception과 Uncheck Exception

Error와 Exception

Error와 Exception



Error와 Exception

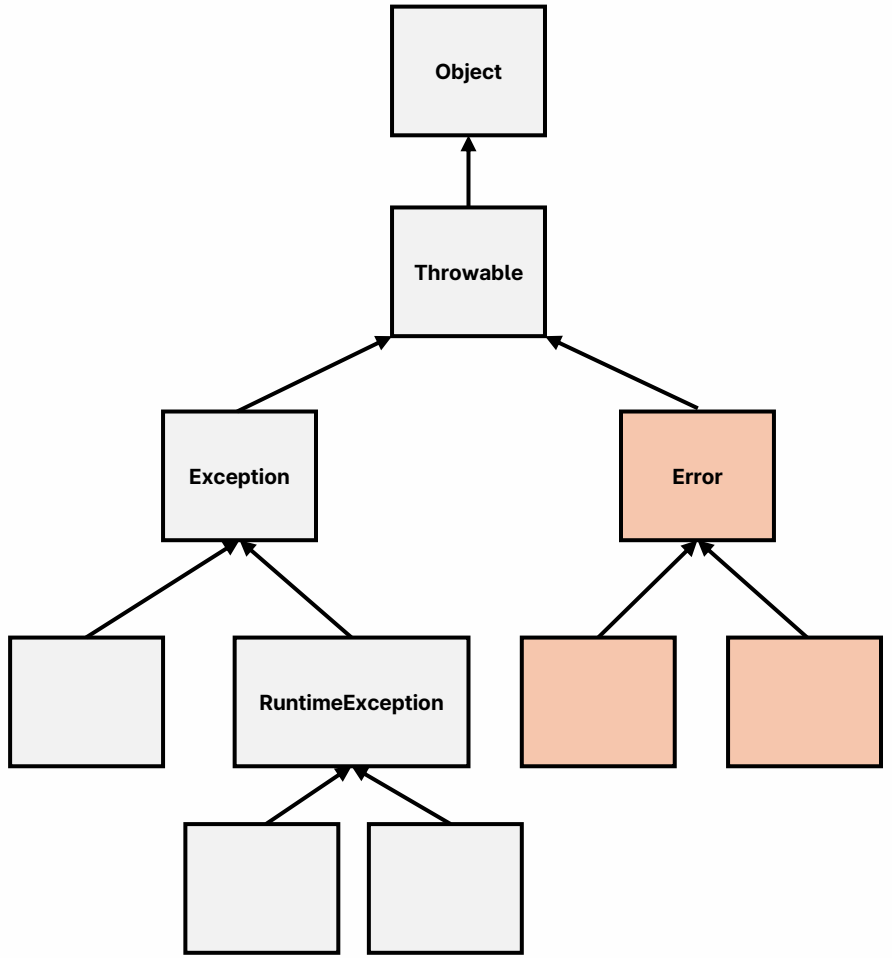


Error와 Exception

Error 개발자가 처리할 수 없는 오류

메모리 부족, 스택 오버 플로우처럼 발생하면 복구할 수 없는 심각한 오류

자바 가상 머신 오류이므로 개발자가 처리할 수 없음 *(* 난리난 상황)*



Error와 Exception

Error 개발자가 처리할 수 없는 오류

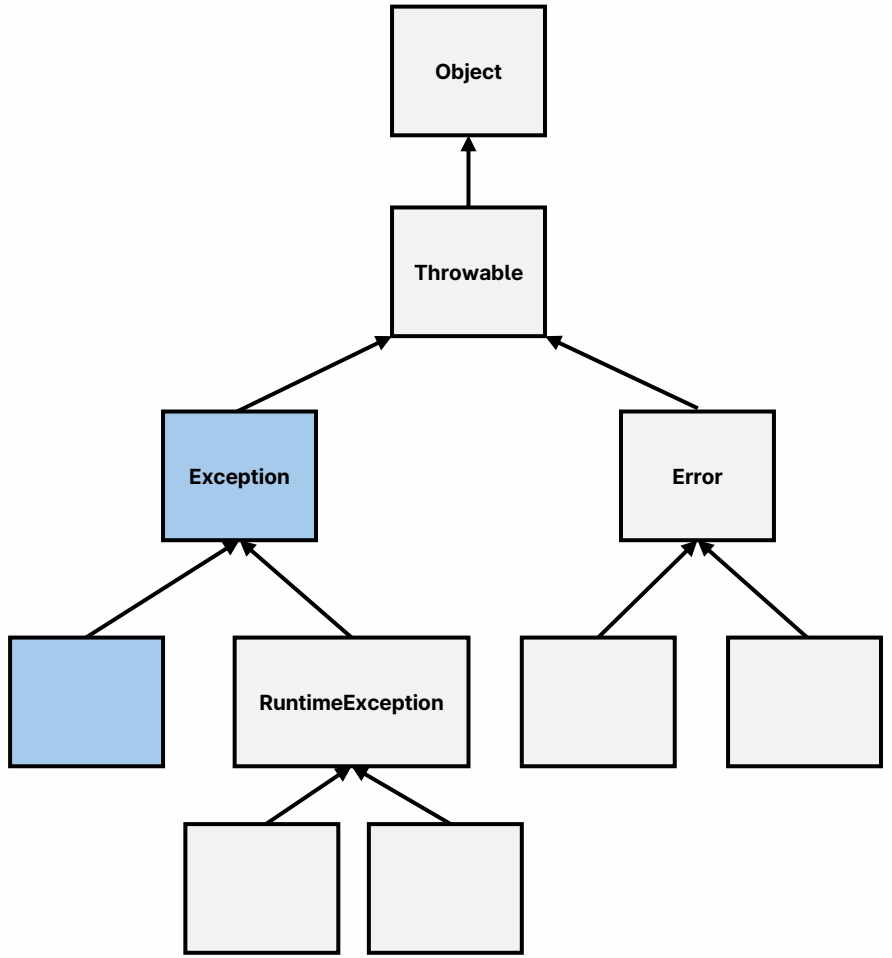
메모리 부족, 스택 오버 플로우처럼 발생하면 복구할 수 없는 심각한 오류

자바 가상 머신 오류이므로 개발자가 처리할 수 없음 *(* 난리난 상황)*

Exception 개발자가 처리할 수 있는 컴파일 오류

존재하지 않는 파일 입력, 네트워크 오류 등 주로 외부 환경에 의해 발생하는 오류

컴파일러가 컴파일 시점에 try-catch로 예외 처리했는지 확인



Error와 Exception

Error 개발자가 처리할 수 없는 오류

메모리 부족, 스택 오버 플로우처럼 발생하면 복구할 수 없는 심각한 오류

자바 가상 머신 오류이므로 개발자가 처리할 수 없음 *(* 난리난 상황)*

Exception 개발자가 처리할 수 있는 컴파일 오류

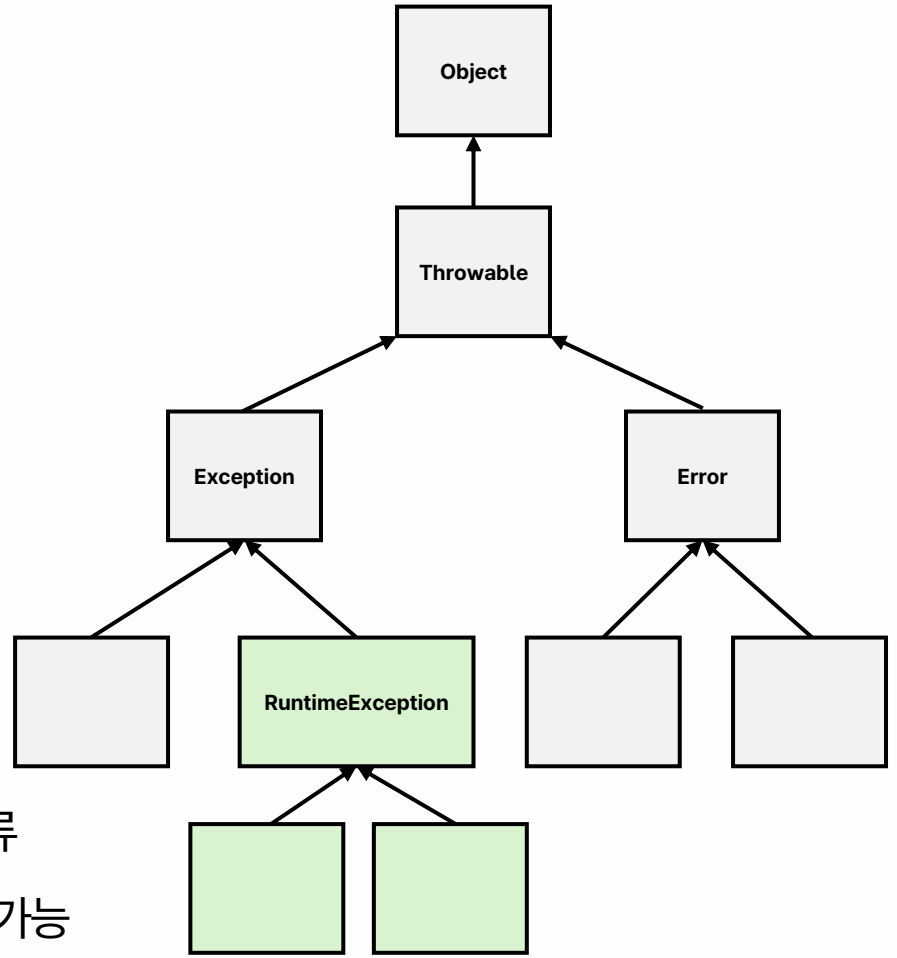
존재하지 않는 파일 입력, 네트워크 오류 등 주로 외부 환경에 의해 발생하는 오류

컴파일러가 컴파일 시점에 try-catch로 예외 처리했는지 확인

RuntimeException 개발자가 처리할 수 있는 런타임 오류

배열 범위를 벗어남, null인 필드를 참조 등 개발자 실수나 잘못된 로직으로 발생하는 오류

컴파일러가 컴파일 시점에 try-catch로 예외 처리했는지 확인하지 않아 런타임 시 확인 가능



Error와 Exception

Error 개발자가 처리할 수 없는 오류

메모리 부족, 스택 오버 플로우처럼 발생하면 복구할 수 없는 심각한 오류

자바 가상 머신 오류이므로 개발자가 처리할 수 없음 *(* 난리난 상황)*

Exception 개발자가 처리할 수 있는 컴파일 오류

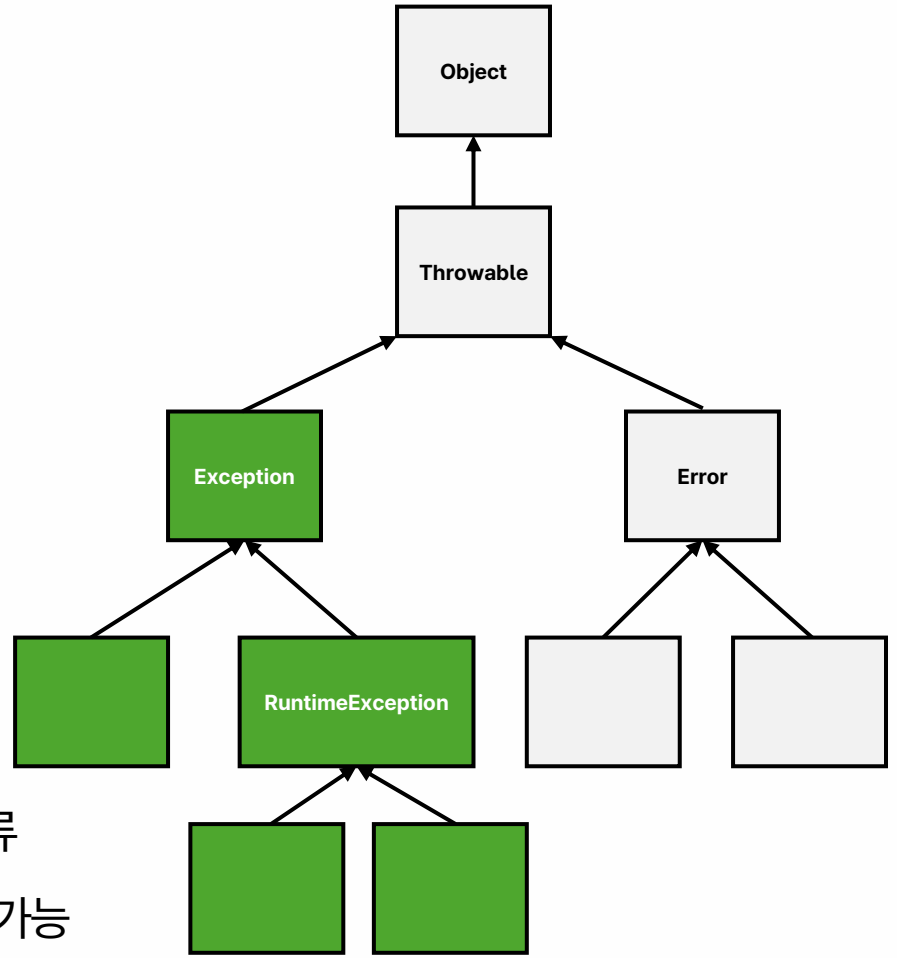
존재하지 않는 파일 입력, 네트워크 오류 등 주로 외부 환경에 의해 발생하는 오류

컴파일러가 컴파일 시점에 try-catch로 예외 처리했는지 확인

RuntimeException 개발자가 처리할 수 있는 런타임 오류

배열 범위를 벗어남, null인 필드를 참조 등 개발자 실수나 잘못된 로직으로 발생하는 오류

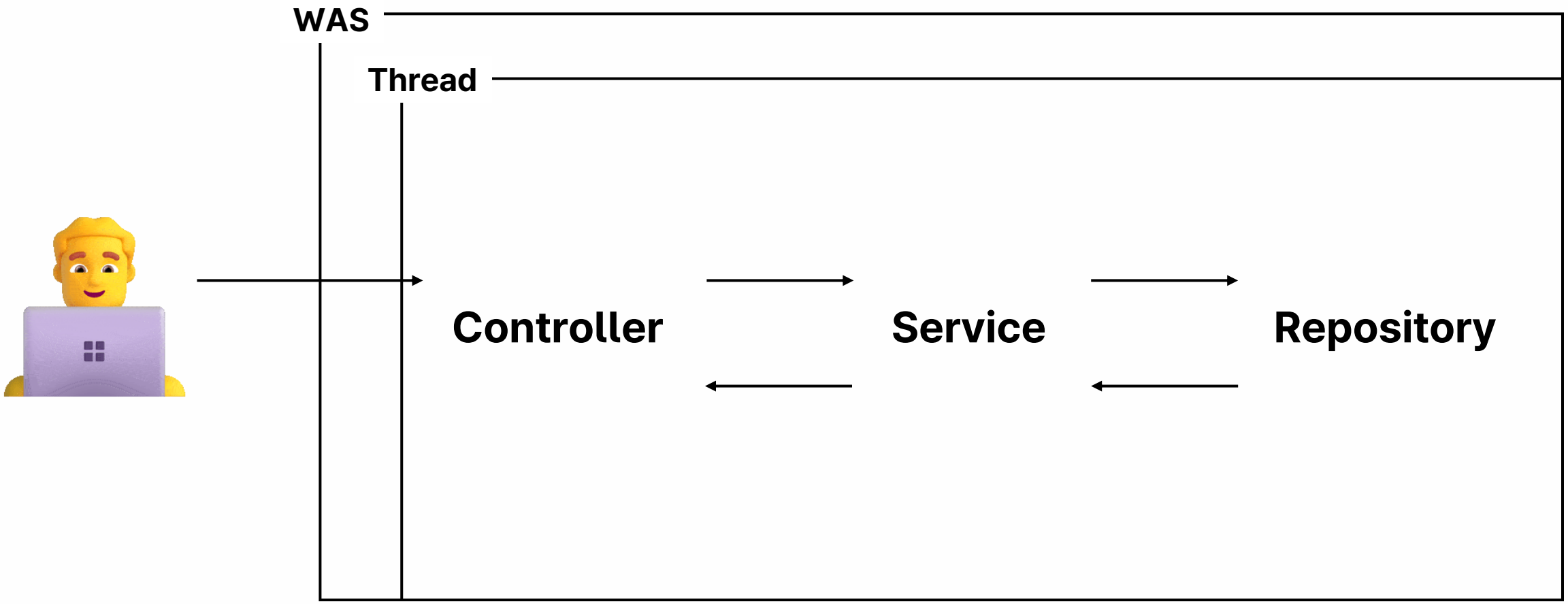
컴파일러가 컴파일 시점에 try-catch로 예외 처리했는지 확인하지 않아 런타임 시 확인 가능



예외 처리 흐름

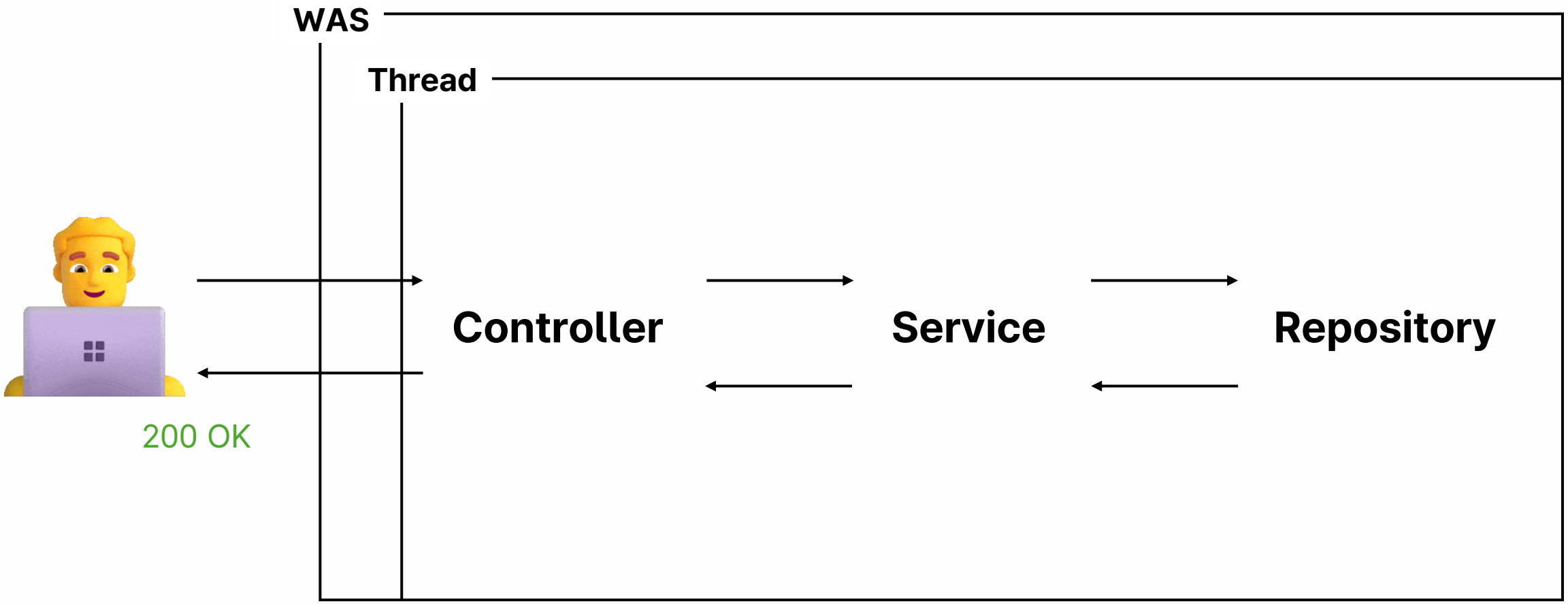
예외 처리 흐름

Spring MVC를 예로 살펴보기 (1) : 예외가 발생하지 않았을 때



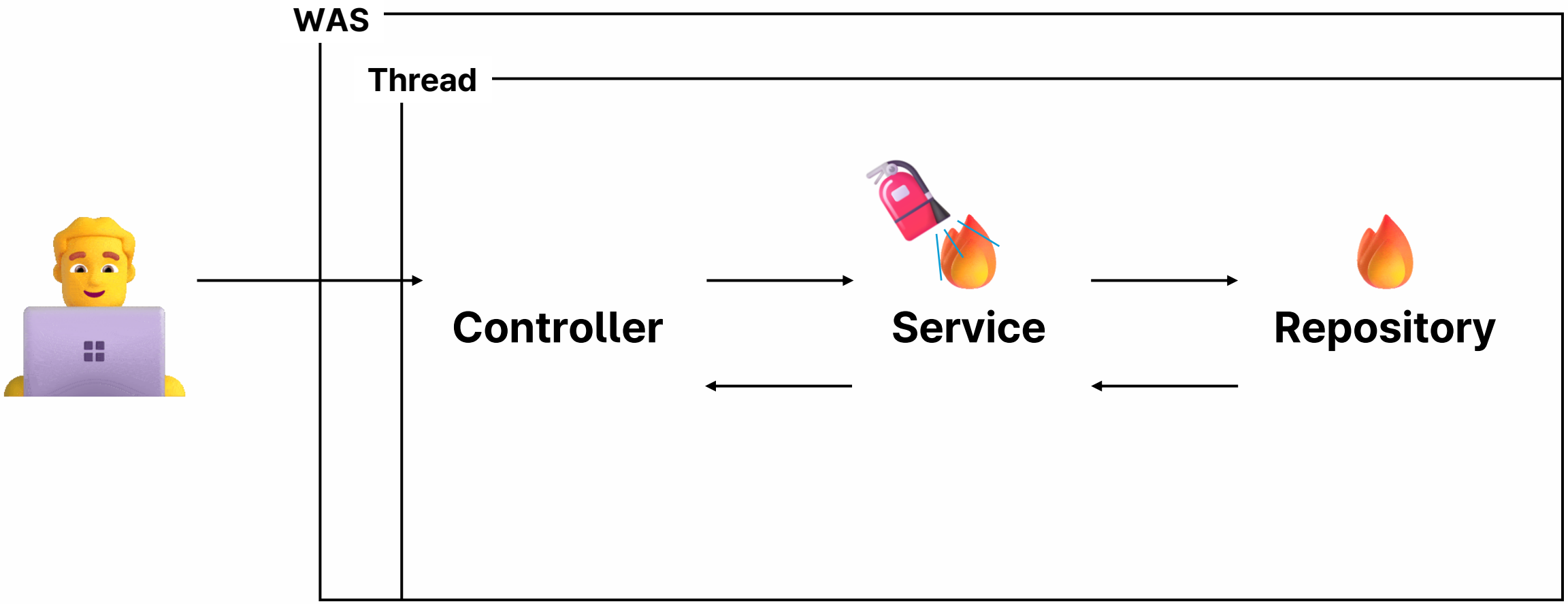
예외 처리 흐름

Spring MVC를 예로 살펴보기 (1) : 예외가 발생하지 않았을 때



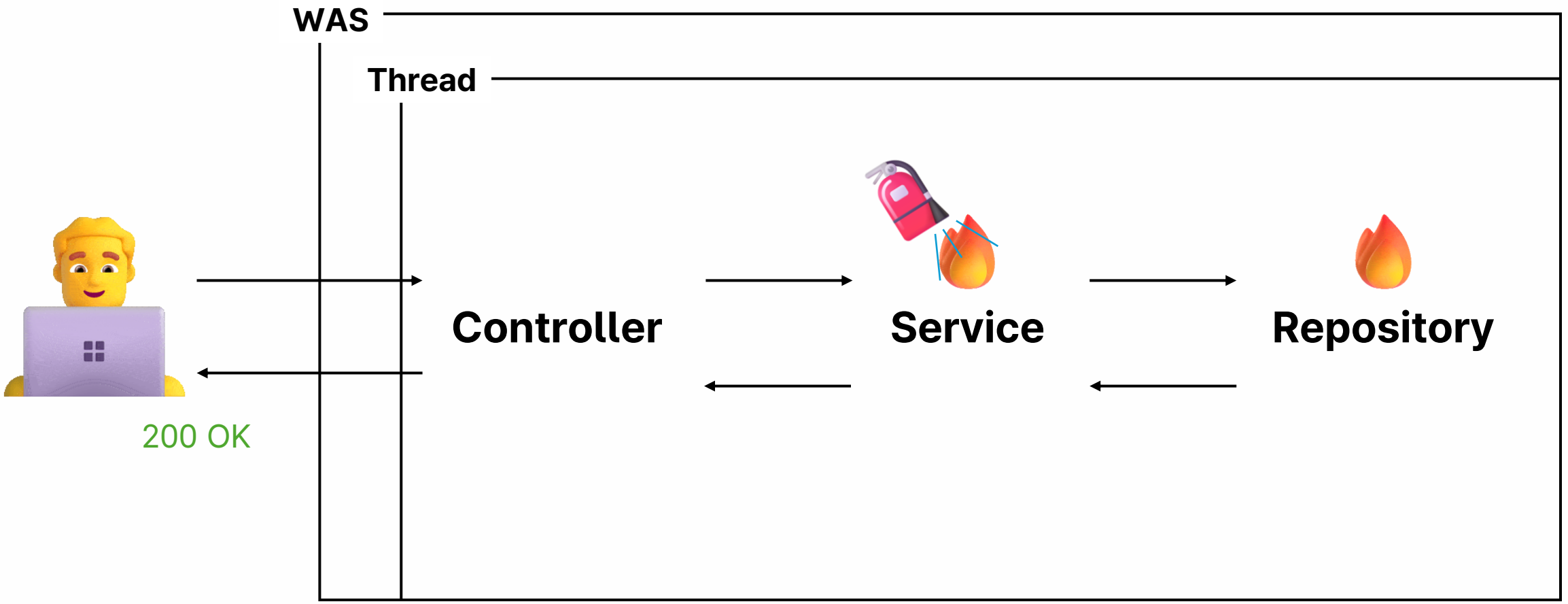
예외 처리 흐름

Spring MVC를 예로 살펴보기 (2) : 예외가 발생했고, 잘 처리했을 때



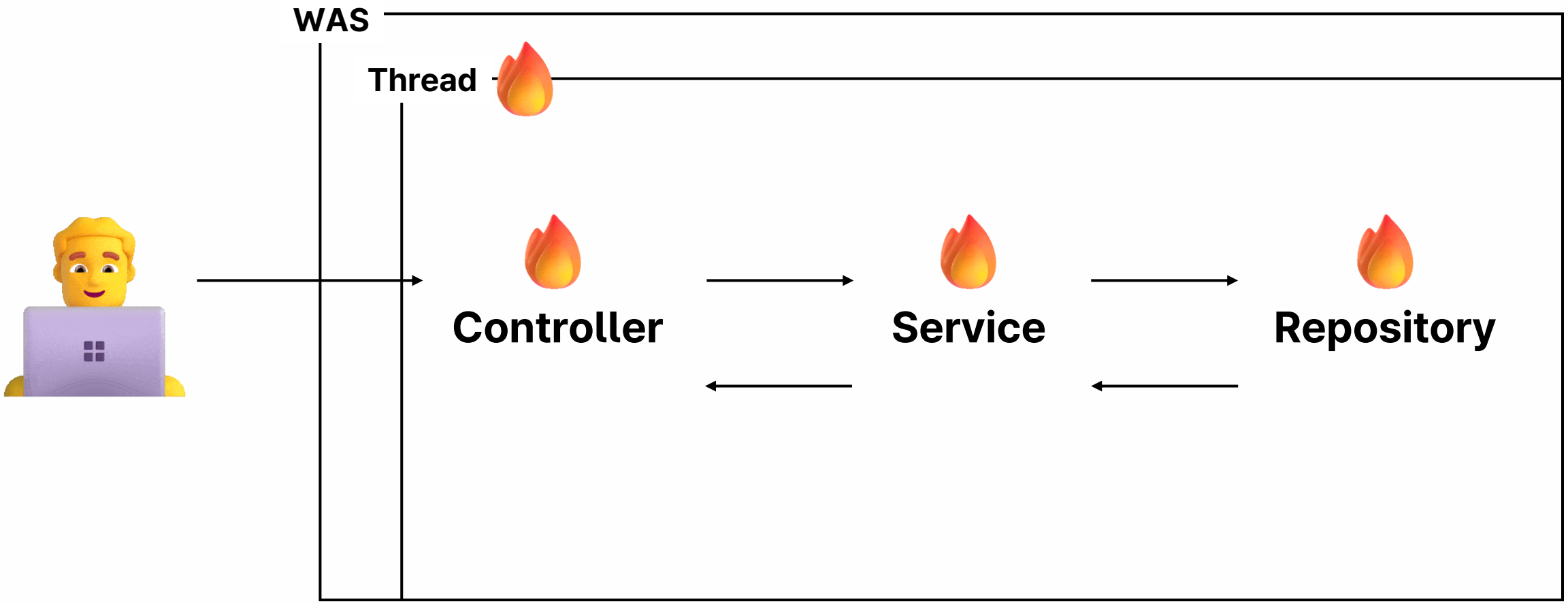
예외 처리 흐름

Spring MVC를 예로 살펴보기 (2) : 예외가 발생했고, 잘 처리했을 때



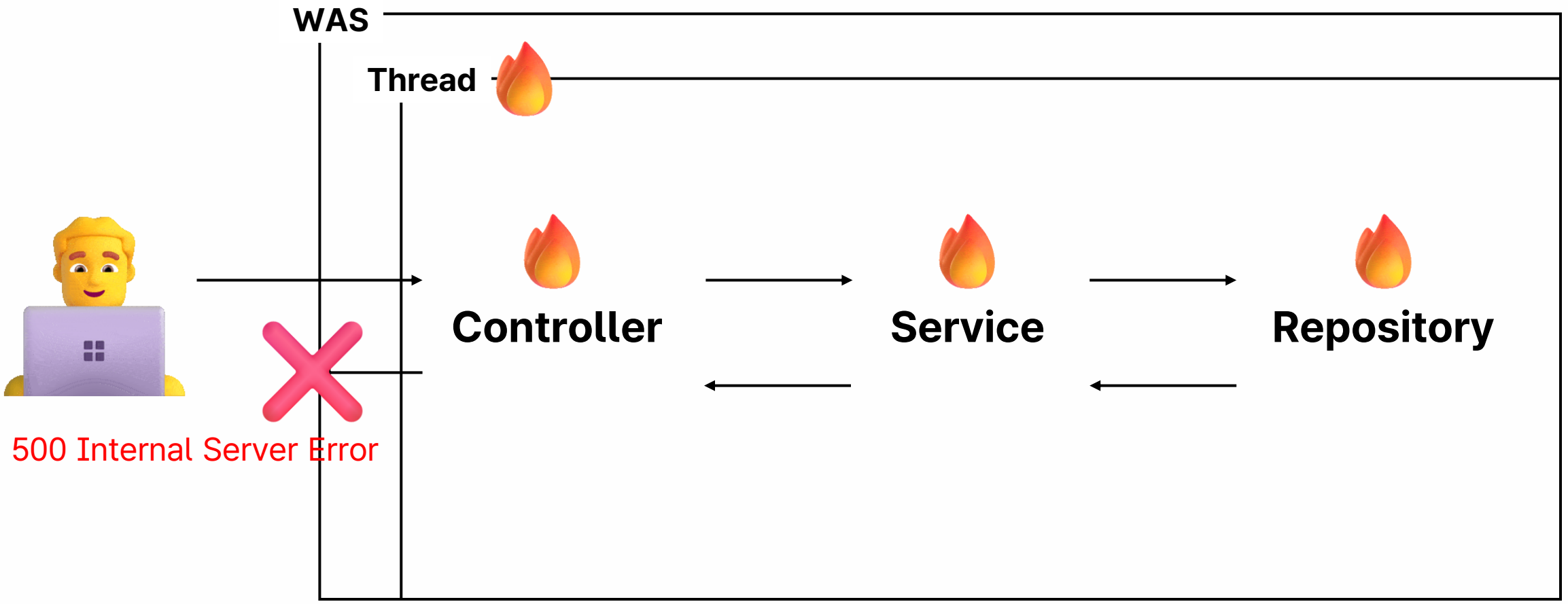
예외 처리 흐름

Spring MVC를 예로 살펴보기 (3) : 예외가 발생했고, 잘 처리하지 못 했을 때



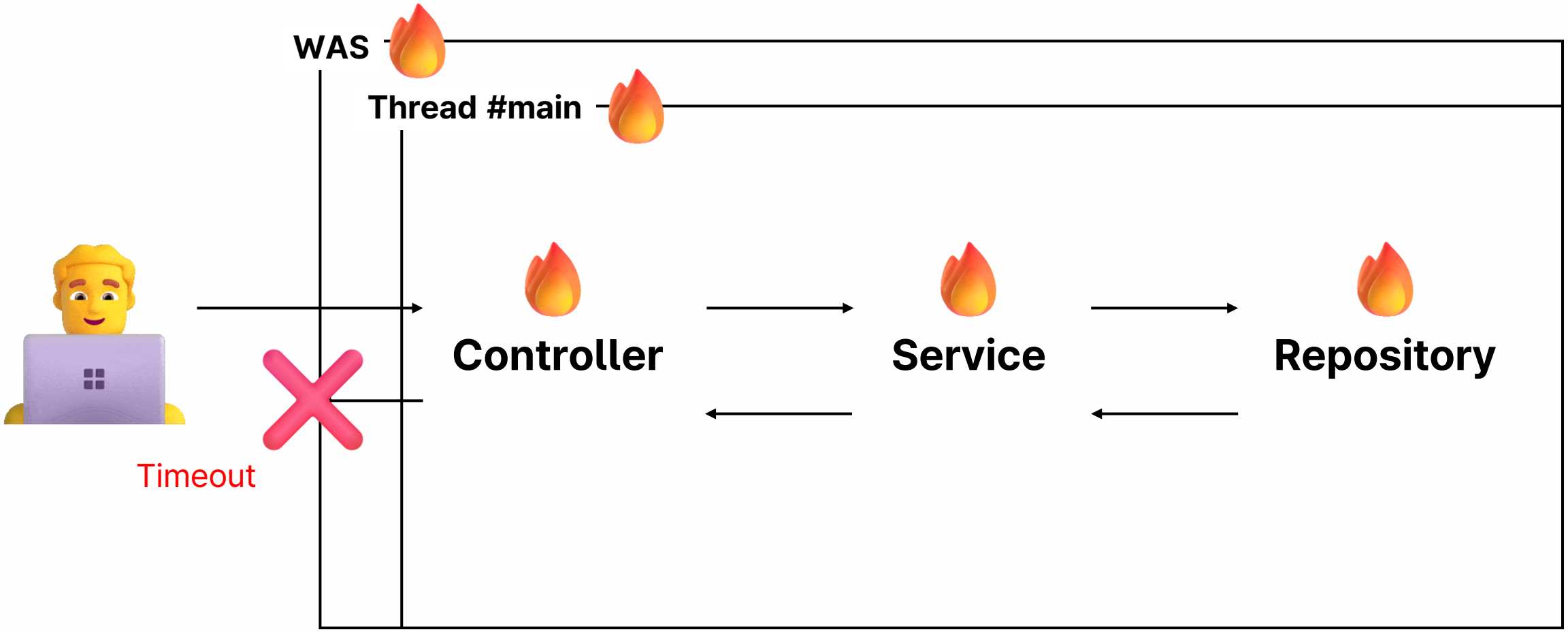
예외 처리 흐름

Spring MVC를 예로 살펴보기 (3) : 예외가 발생했고, 잘 처리하지 못 했을 때



예외 처리 흐름

Spring MVC를 예로 살펴보기 (3) : 예외가 발생했고, 잘 처리하지 못 했을 때



Check Exception과 Unchecked Exception

Checked Exception과 Unchecked Exception

Checked Exception

- 컴파일 시점에 예외를 처리하는지 확인
  - 예외를 처리하는 try-catch 블록이 없으면 컴파일 오류 발생
- 호출부에서 합리적으로 복구할 수 있는 예외
  - 외부 API 통신 일시 실패
    - 재시도 또는 대체 API 사용
  - 파일이 존재하지 않음
    - 경로 재입력, 새 파일 만들기
  - 트래픽 초과
    - 대기 후 재시도
- IOException, SQLException 등이 해당
  - 개발자는 이 예외들에 대해 반드시 처리해 주어야 함

```
public void validateInput(String in)
    throws IOException {
    if (in == null || in.isBlank())
        throw new IOException();
}
```

1

```
public void run(String in) {
    validateInput(in);
}
```

2

```
public void run(String in) {
    try {
        validateInput(in);
    } catch (IOException ex) {
        System.out.println("문자열이 없거나, 빈 문자열입니다.");
    } catch (Exception ex) {
        System.out.println("이 것은 권장하지 않습니다.");
    }
}
```

Checked Exception과 Unchecked Exception

Unchecked Exception

- 컴파일 시점에 예외를 처리하는지 확인하지 않음
  - 예외를 처리하는 try-catch 블록이 없어도 컴파일 오류가 발생하지 않음

- 호출부에서 복구할 수 없거나 조치할 수 없는 예외
  - 프로그래머 실수 (null 참조, 인덱스 범위 초과)
  - 유효성 검증 실패
    - 다른 값으로 대체하면 프로그램은 종료되지 않지만, 엉뚱한 값이 저장되거나 버그를 찾기 힘들 수 있음

```
public void validateInput(String in) {
    if (in == null || in.isBlank())
        throw new RuntimeException();
}
```

1

```
public void run(String in) {
    validateInput(in);
}
```

2

```
public void run(String in) {
    try {
        validateInput(in);
    } catch (RuntimeException ex) {
        System.out.println("문자열이 없거나, 빈 문자열입니다.");
    } catch (Exception ex) {
        System.out.println("이 것은 권장하지 않습니다.");
    }
}
```

Checked Exception과 Unchecked Exception

Unchecked Exception (이어서)

- NullPointerException, ArrayIndexOutOfBoundsException 등이 해당
- 개발자는 이 예외들에 대해 반드시 처리하지 않아도 됨
  - 예외를 처리하지 않으면 예외는 계속 전파
  - 최상위에 도달한 처리되지 않은 예외는 스레드 실행을 중단
  - 메인 스레드라면 프로그램이 종료

```
public void validateInput(String in) {
    if (in == null || in.isBlank())
        throw new RuntimeException();
}
```

1

```
public void run(String in) {
    validateInput(in);
}
```

2

```
public void run(String in) {
    try {
        validateInput(in);
    } catch (RuntimeException ex) {
        System.out.println("문자열이 없거나, 빈 문자열입니다.");
    } catch (Exception ex) {
        System.out.println("이 것은 권장하지 않습니다.");
    }
}
```

Checked Exception과 Unchecked Exception

실무에서 예외 처리 패턴

- 기본적으로 Unchecked Exception을 사용하여 신경 쓰고 싶지 않은 예외의 의존 관계를 참조하지 않도록 함
  - Checked Exception을 사용하면 해당 예외를 처리하지 않는 메서드에 throws 키워드와 예외 import 문을 사용해야 함

- 비즈니스 로직 상 매우 중요하고 의도적으로 던지는 예외에만 Checked Exception을 사용하는 것을 권장
  - 계좌 이체 실패
  - 결제 실패
  - …

  ➡️ Checked Exception을 사용해 개발자는 **해당 예외를 반드시 처리**하게 됨

  이러한 경우 사용자에게는 오류 페이지를 보여주고, 별도의 오류 로그와 알림 등을 통해 개발자가 빠르게 파악하도록 해야 함

Checked Exception과 Unchecked Exception

Checked Exception 필요성에 대한 논의

> *Kotlin은 Exception을 Java에서 가져왔지만 Java와 동작을 달리한다. 다른 최신 언어처럼 Kotlin 개발자도 Checked Exception을 포함하지 않기로 했다.*
>
> — Dmitry Jemerov와 Svetlana Isakova, <Kotlin In Action>, 2017.

> *소규모 프로그램은 예외 처리를 요구하여 개발자 생산성과 코드 품질을 높일 수 있다. 그러나 대규모 소프트웨어 프로젝트에서는 생산성은 떨어지고 코드 품질은 거의 또는 전혀 향상되지 않는다.*
>
> — Kotlin 공식 문서

> *Java의 Checked Exception은 실험이었지만 대체로 실패했다. 애플리케이션 아키텍처의 연결부에서 사실상 재앙에 가깝다. 중간 수준 API는 하위 수준에서 발생할 수 있는 특정 유형의 오류를 알 필요도 없고 일반적으로 알고 싶어하지 않으며 발생할 수 있는 예외 조건에 적절하게 대응하기도 어렵다.*
>
> — Rod Waldhoff, <Java's checked exceptions were a mistake>, 2003.

Checked Exception과 Unchecked Exception

Checked Exception 필요성에 대한 논의

> *Checked Exception은 OCP(Open/Closed Principle, 개방-폐쇄 원칙)를 위반한다.*
>
> *하위 단계 코드에서 새로운 Checked Exception을 던지도록 수정하면,*
>
> *상위 단계 메서드 선언부를 전부 고쳐야 한다. 이는 캡슐화를 깨뜨린다.*
>
> — Robert C. Martin, <Clean Code>, 인사이트, 2013.

> *Checked Exception은 버전 관리와 확장성에 문제를 일으킨다.*
>
> *작은 프로그램에서는 훌륭해 보이지만, 거대한 시스템을 만들 때 Checked Exception은 의존성 덩어리가 되어버린다.*
>
> — Anders Hejlsberg, <The Trouble with Checked Exceptions>, Artima, 2003.

Checked Exception과 Unchecked Exception

Checked Exception 필요성에 대한 논의

> *나는 Checked Exception이 실수였다고 생각하지 않는다.*
> *예외를 무시하는 것은 너무나 쉽다. Checked Exception은 개발자가 오류 상황을 강제로 처리함으로써*
> *신뢰성 있는 시스템을 만들게 한다.*
>
> — James Gosling, <Failure and Exceptions>, Artima, 2003.

> *Checked Exception은 프로그램의 견고함을 보장하기 위한 핵심 기능이다.*
> *컴파일 시간에 예외 처리를 강제해 런타임에 발생 가능한 처리되지 않은 오류를 획기적으로 줄인다.*
>
> — 자바 언어 명세 Java SE 8 에디션, 2015.

> *Checked Exception을 싫어하는 Java 프로그래머가 많지만 제대로 활용하면 API와 프로그램의 질을 높일 수 있다.*
> *결과를 코드로 반환하거나 예외를 던지는 것과 달리, Checked Exception은 발생한 문제를*
> *프로그래머가 처리해 안정성을 높이게끔 해준다.*
>
> — Joshua Bloch, <Effective Java>, 인사이트, 2018.

들어 주셔서 감사합니다

박연종

E-mail	park@duck.com
LinkedIn	yeonjong-park
Instagram	yeonjong.park
GitHub	patulus

2025.11.19.