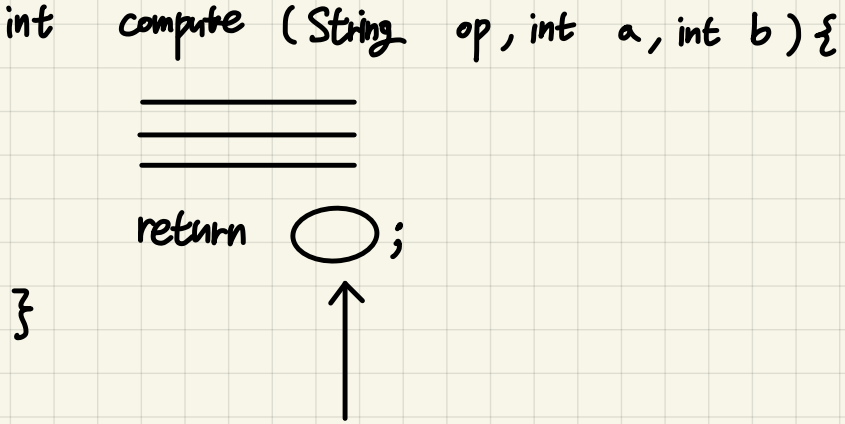
2021.08.30 (월 45일차) 09:40

*호출자에게 예외상황을 알리는 방법 - 리턴값

com.eomcs.exception.ex1
.Exam0130 - 0131



그러나 그 리턴값이
정상적인 계산결과일 수도 있다. ←
리턴값을 이용하여 예외상황을
알리는 방법의 한계다.

2021.08.30 (월 45일차)

* 예외처리 문법의 등장이유.

왜 사용? 예외상황을 상대방에게 "정확하게" 알려주기 위해서

↳ return은 정확하지 X

cx2

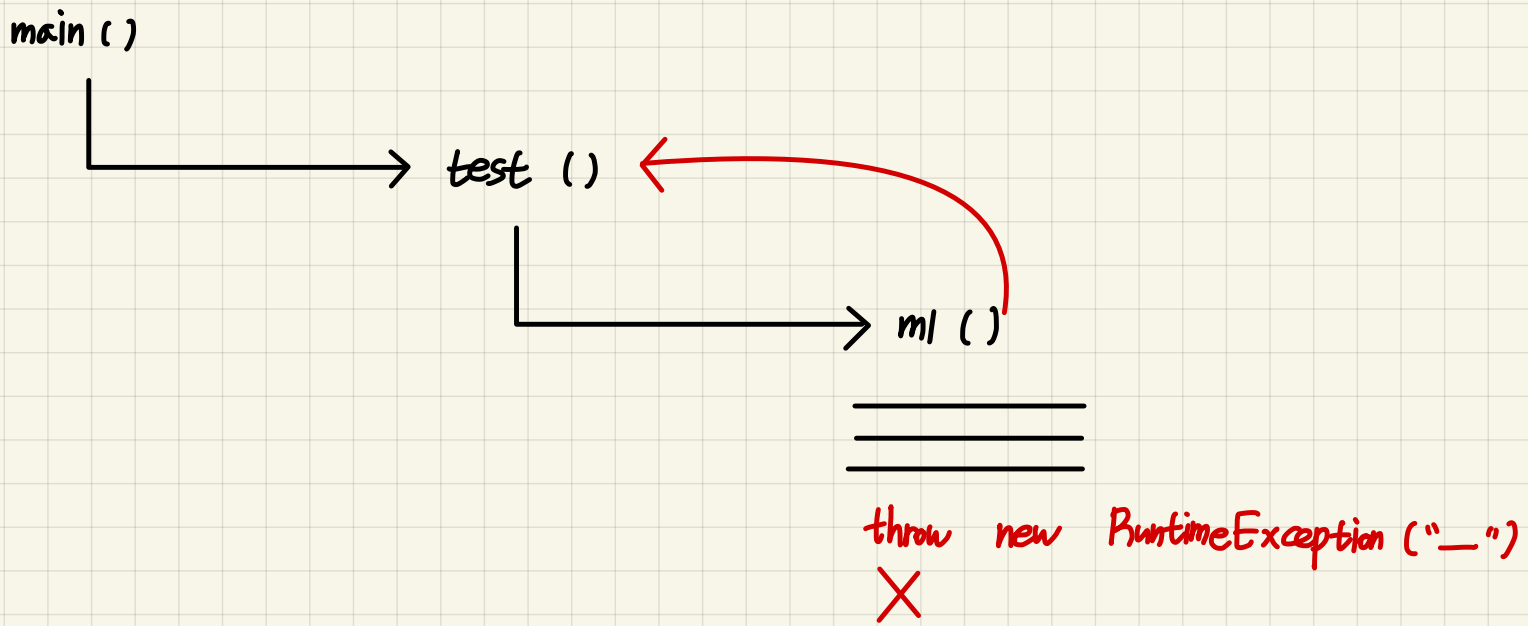
① 리턴값으로 예외상황을 알리는 방식의 한계 극복!

② 예외가 발생하더라도 JVM을 멈추지 않게 만드는 것!

2021.08.30 (월 45일차)

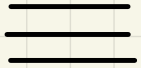
* 예외 던지고 받기

. ex3 . Exam 0111 10:30



2021.08.30 (월 45일차)

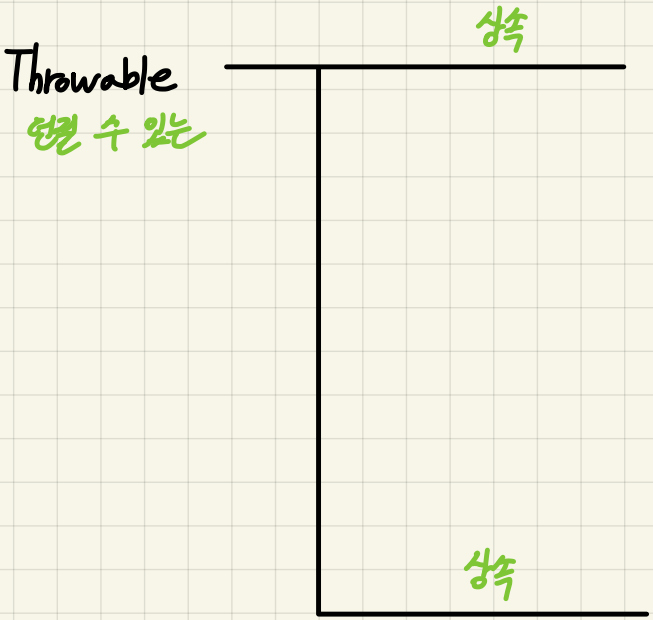
try {



} catch

throw

2021.08.30 (월 45일차) * 예외의 종류



Error (언첵 수 없음)
- JVM 에서 발생
- 개발자가 처리해서도 안됨
- 간단한 조치 후 즉시 종료해야 한다
(오류 내용 출력)

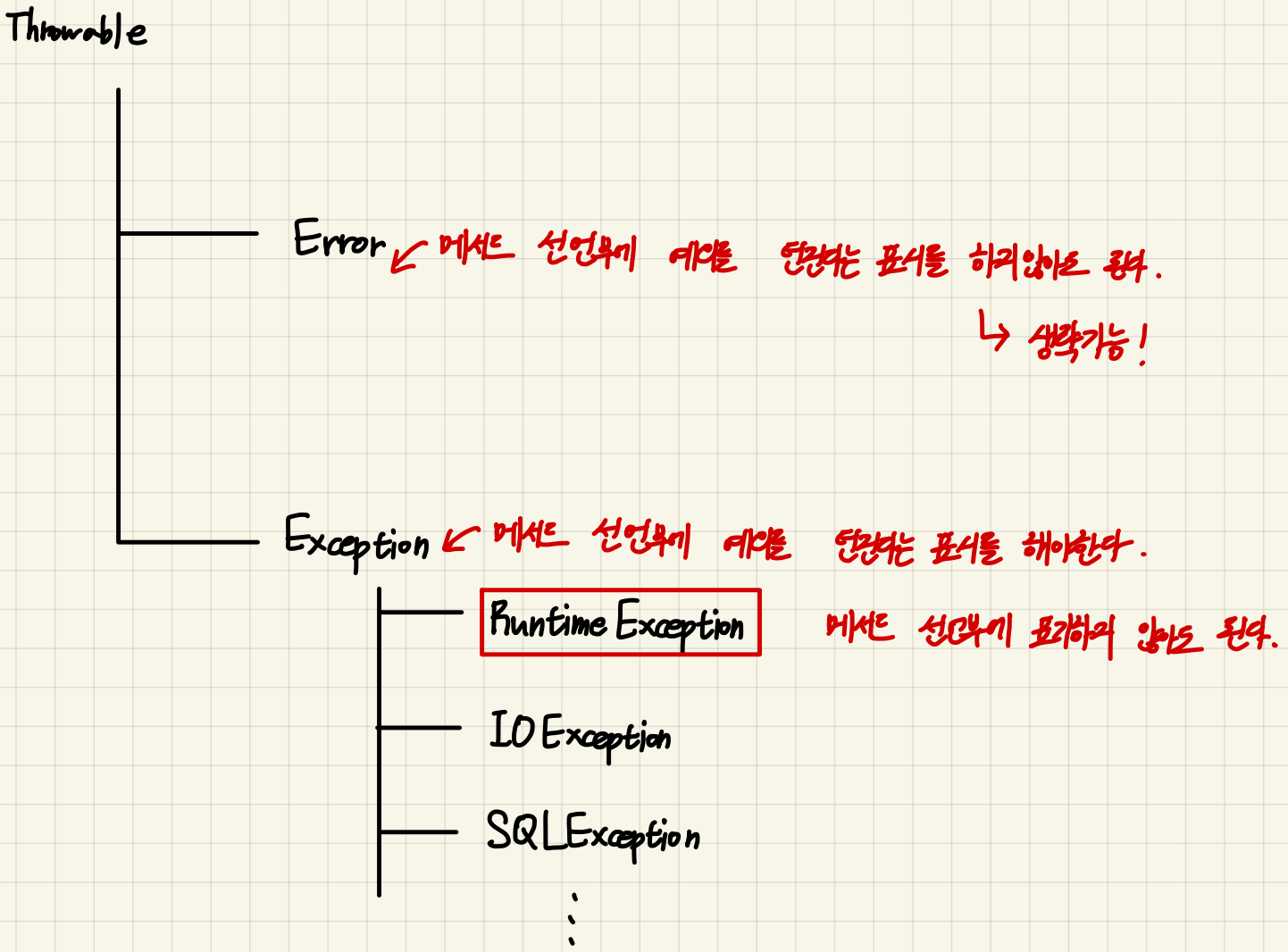
예) 메모리 부족

Exception (언첵 수 있음)
- 애플리케이션에서 발생
- 개발자가 처리해야 한다.
- 적절한 조치 후 계속 실행시켜야 한다.

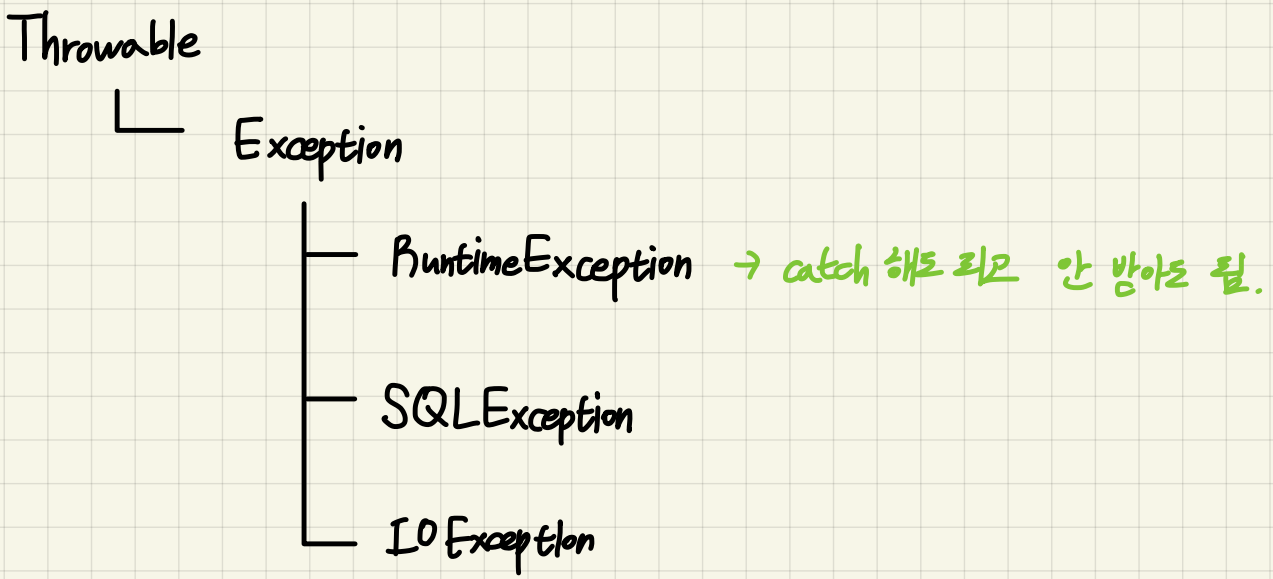
예) 입력 형식 오류
인덱스 범위 초과 오류

// throws String (X)
↑ 언첵수 모음

2021.08.30 (월 45일차) 11:10



2021.08.30 (월 45일차) 11:30



2021.08.30 (월 45일차) 11:50

exam0460

* catch 문 parameter 에서 (RuntimeException | SQLException | IOException) 가능

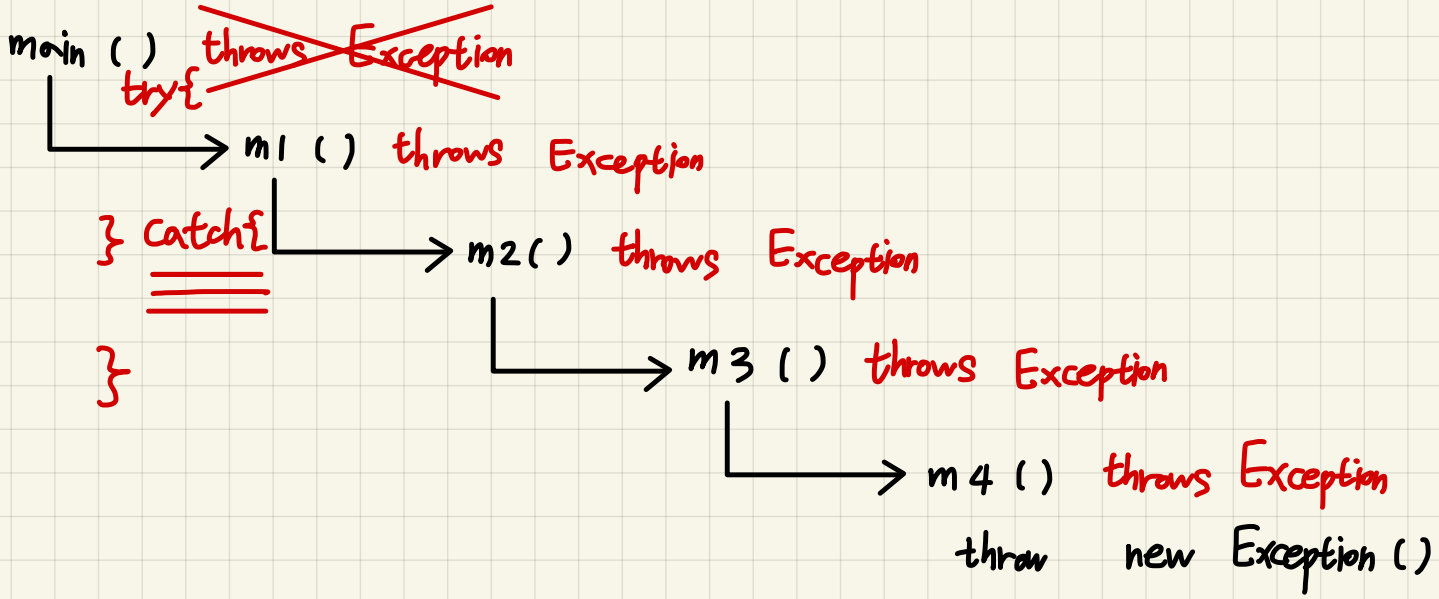
13:20

exam0610

2021.08.30 (월 45일차) 13:55

ex4 . Exam0130

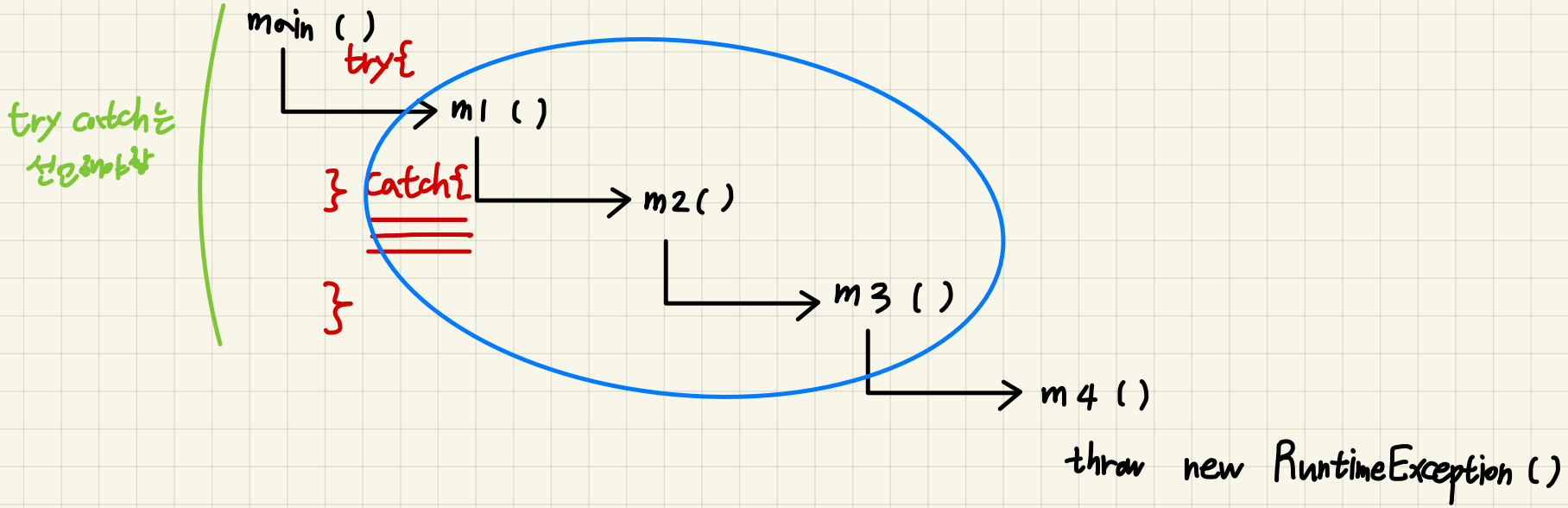
* Exception 객체 던지기



2021.08.30 (월 45일차)

ex4 . Exam0130

* Exception 예외 던지기

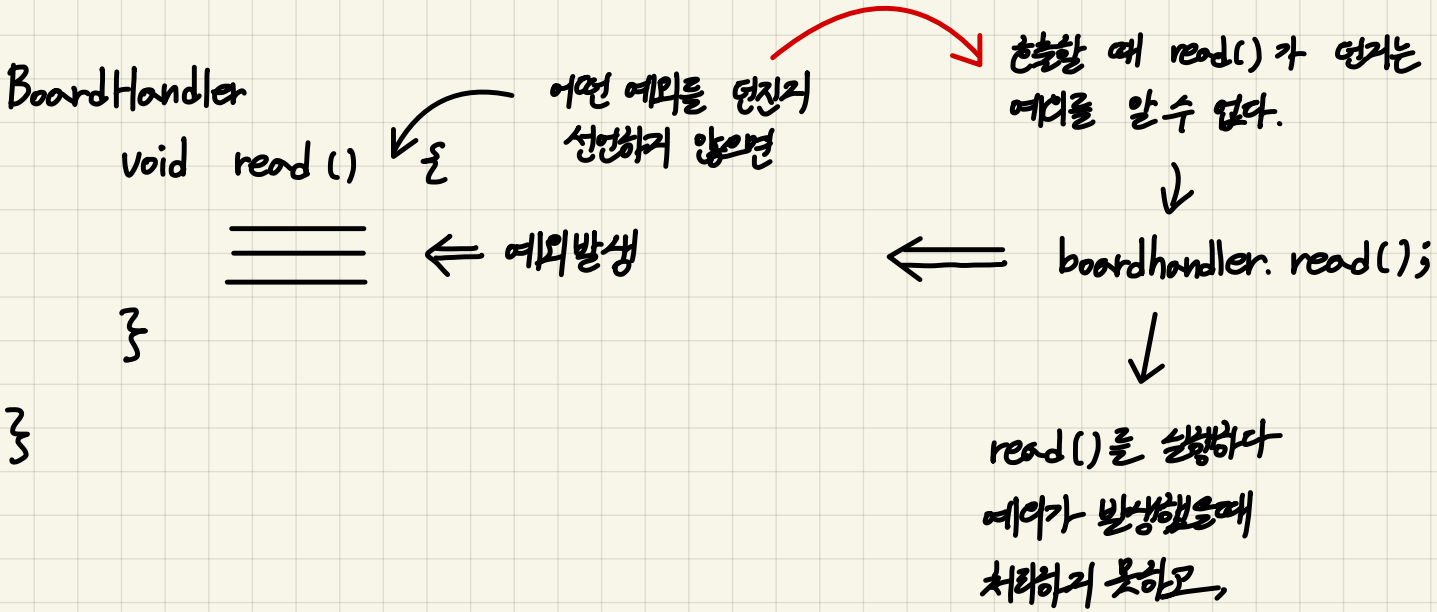


throw new RuntimeException ( )

2021.08.30 (월 45일차)

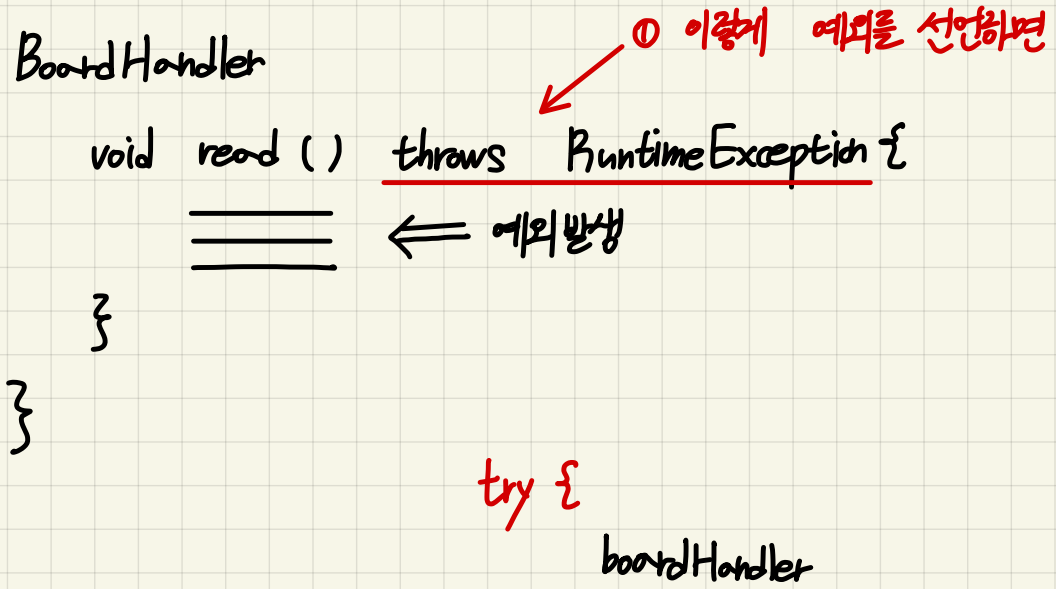
* 예외 클래스 상속받기

14:30



2021.08.30 (월 45일차)

* 예외 클래스 상속받기



2021.08.30 (월 45일차)

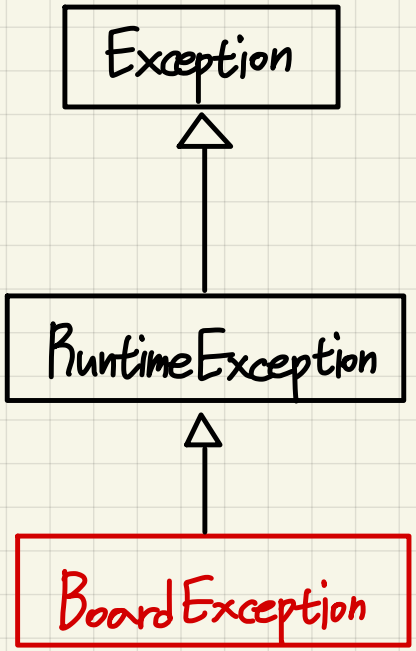
* 예외 클래스 상속받기

① 이왕 예외를 던지는 김에 의미있는 이름의 예외를 던진다면

```
BoardHandler
    void read() throws BoardException {
        ========  ⇐ 예외발생
    }
}
```

```
try {
    boardHandler.read();
} catch (BoardException e) {
    ========  ②
}
```

* 예외 클래스 상속 받기



서브클래스는 이름을 통해
예외에 대한 의미를 전달하는 것이 목적이다.

⇓

그래서

ex03 ex05
0640 -

com.comcs.exception.ex91 ~ ex92 자동