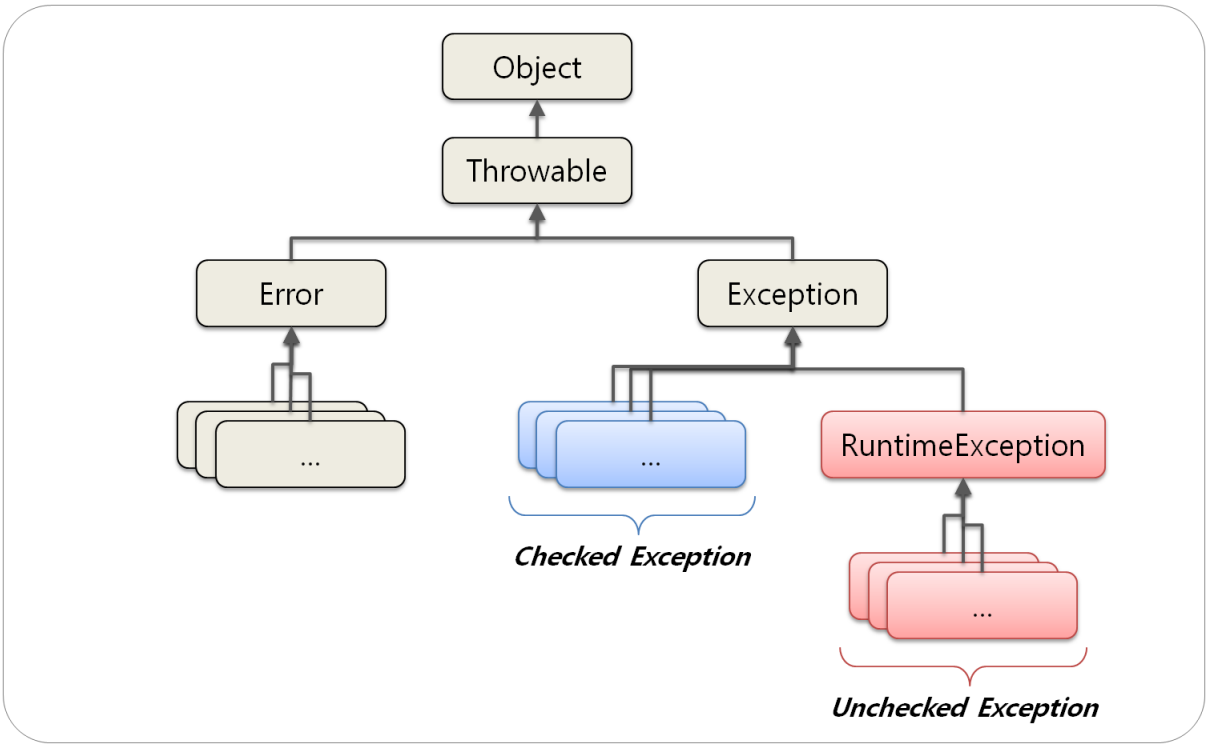
4️⃣ 복구할 수 있는 상황에는 검사 예외를, 프로그래밍 오류에는 런타임 예외를 사용하라

참고문헌 : https://www.nextree.co.kr/p3239/



error : 시스템 레벨 오류(심각) 어차피 못고침 throw로 던지지도 말 것!

exception : 개발자가 구현한 로직에서 발생 → 예외처리 가능

	Checked Exception	Unchecked Exception
처리여부	반드시 예외를 처리해야 함	명시적인 처리를 강제하지 않음
확인시점	컴파일 단계	실행단계
예외발생시 트랜잭션 처리	roll-back 하지 않음	roll-back 함
대표 예외	Exeption의 상속받는 하위 클래스 중 Runtime Exception을 제외한 모든 예외 • IOException • SQLException	RuntimeException 하위 예외 • NullPointerException • IllegalArgumentException • IndexOutOfBoundException • SystemException

- 이 둘을 나누는 기준은 '꼭 처리를 해야 하느냐'이다.
- 검사 예외를 던지는 경우는 예외처리를 강제하는 것, 상황 회복의 여지가 있는 상황이다.
- 이걸 못나누겠으면 unchecked exception으로 처리하는 것이 낫다.
- throwable은 혼란만 야기하니 사용하지 말 것
- 검사 예외라면 복구에 필요한 정보를 알려주는 메서드도 제공하자.

  ex) 쇼핑몰에서 물건 구매하는데 카드 잔고 부족이란 검사 예외가 발생했다면,

  이 예외는 잔고가 얼마나 부족한지 등의 접근자 메서드를 제공해야 한다. → 아이템 75

예외 처리 방법

예외 복구

예외가 발생해도 정상적인 흐름으로 돌려놓음.

ex) 서버가 접속이 안되는 상황

접속x → 예외 발생 → 예외 catch → 일정시간 대기 → 재시도

미리 예측 가능한 예외를 원하는 흐름, 정상적인 흐름으로 돌려놓기 가능

예외 처리 회피

throws 를 이용

호출한 쪽으로 예외를 던지고 회피

던지는 것이 최선의 방법이란 확인이 있을때만 쓰는 것이 좋음

예외 전환

catch 후에 throw 하는 방법

어떤 예외인지 더 구체화, 명확화 하고 싶을 때 쓰인다.

혹은 checked exception을 catch 했는데, 이를 unchecked exception으로 전환하여 해결할 수 있도록 하는 것도 방법이다.

catch를 비우는 짓은 절대 하면 안됨(오류 원인 파악 불가)

https://jaehun2841.github.io/2019/02/24/effective-java-item55/#서론