7장 오류 처리

* 깨끗한 코드와 오류 처리는 확실히 연관성이 있따. 상당수 코드 기반은 전적으로 오류 처리 코드에 좌우된다. 여기서 좌우 된다는 표현은 코드 기반이 오류만 처리한다는 의미가 아니다. 여기저기 흩어진 오류 처리코드 떄문에 실제 코드가 하는 일을 파악하기가 거의 불가능하다는 의미이다.
* 오류 코드보다 예외를 사용하라
  + 오류를 발생하면 예외를 던지는 편이 더 낫다.
* Try-Catch-Finally문부터 작성하라
  + 에외에서 프로그램 안에다 범위를 정의한다는 사실은 매우 흥미롭다. Try-catch-finally 문에서 try 블록에 들어가는 코드를 실행하면 어느 시점에서든 실행이 중단된 후 catch 블록으로 넘어갈 수 있따.
  + 어떤 면에서는 try블록은 트랜잭션과 비슷. try문에서 어떤 일이 발생하든 catch 블록은 프로그램 상태를 일관성 있게 유지해야 한다. 그러므로 예외가 발생할 코드를 짤 때 try-catch-finally 문으로 시작하는 편이 낫다.
* 미확인 예외를 사용하라
* 예외의 의미를 제공하라
  + 예외를 던질 때는 전호 상황을 충분히 덧붙인다. 오류가 발생한 원인과 위치를 찾기가 쉬워짐
  + 자바는 모든 예외의 호출 스택을 제공한다. 하지만 실패한 코드의 의도를 파악하려면 호출스택만으로는 부족하다
* 호출자를 고려해 예외 클래스를 정의하라
  + 오류를 분류하는 방법은 수없이 많다. 오류가 발생한 위치로 분류가 가능하다.
  + 오류가 발생한 컴포넌트로 분류한다. 예를 들어 디바이스 실패, 네트워크 실패 프로그래밍 오류 등으로 분류한다.
  + 애플리케이션에서 오류를 정의할 떄 프로그래머에게 가장 중요한 관심사는 오류를 잡아내는 방법이 되어야 한다.
* 정상 흐름을 정의하라.
* Null을 반환하지 마라
  + 흔히 저지르는 바람에 오류를 유발하는 행위도 언급해야 한다. 그중 첫째가 null을 반환하는 습관이다.
* Null을 전달하지 마라
  + 메서드에서 null을 반환하는 방식도 나쁘지만 메서드로 null을 전달하는 방식은 더 나쁘다
  + 정상적인 인수로 null을 기대하는 api가 아니라면 메서드를 null을 전달하는 코드는 최대한 피한다.
* 결론
  + 깨끗한 코드는 읽기도 좋아야 하지만 안정성도 높아야 한다. 이 둘을 상충하는 목표가 아니다.
  + 오류처리를 프로그램 논리와 분리해 독자적인 사안으로 고려하면 튼튼하고 깨끗한 코드를 작성할 수 있다.
  + 오류처리를 프로그램 논리와 분리하면 독립적인 추론이 가능해지며 코드 유지보수성도 크게 높아진다.