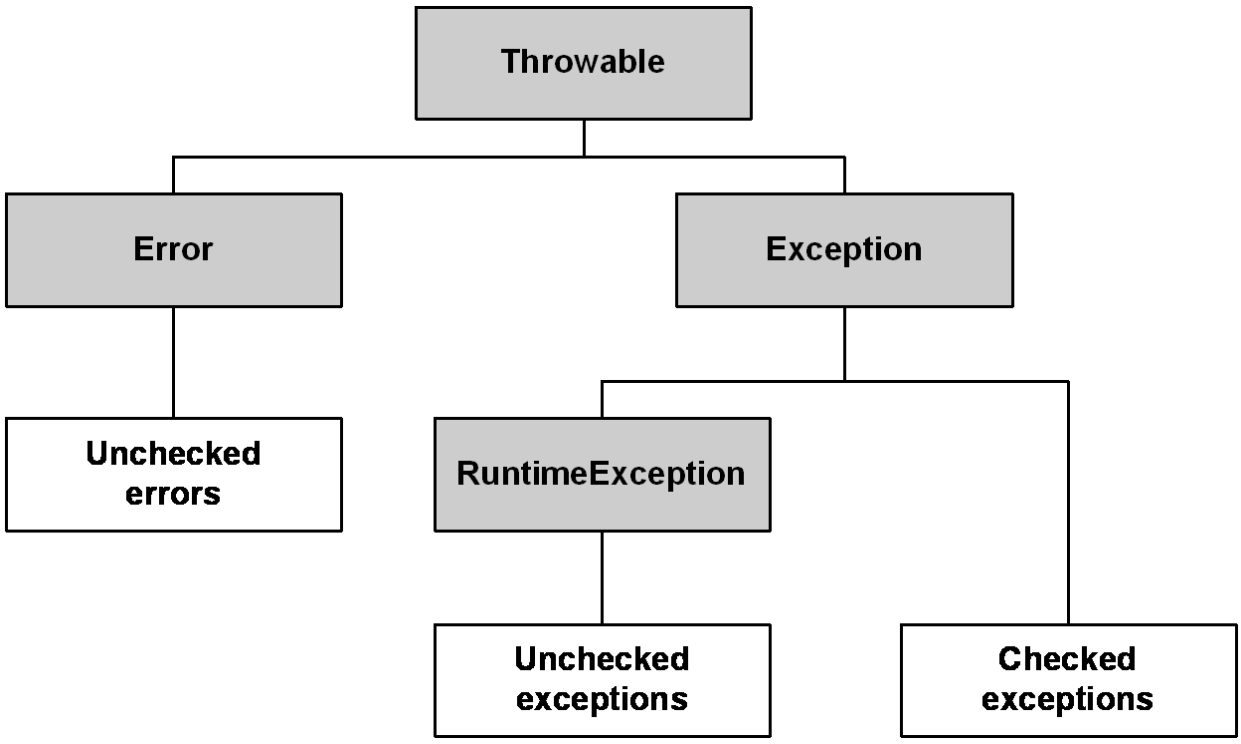
Chapter 02.코틀린 기초 - 06.예외처리

예외처리

- 코틀린의 모든 예외 클래스는 최상위 예외 클래스인 `Throwable`을 상속한다



- Error : 시스템에 비정상적인 상황이 발생. 예측이 어렵고 기본적으로 복구가 불가능 함
  - e.g. OutOfMemoryError, StackOverflowError, etc
- Exception : 시스템에서 포착 가능하여(try-catch) 복구 가능. 예외 처리 강제

    - IOException, FileNotFoundException, etc
    - `@Transactional` 에서 해당 예외가 발생하면 기본적으론 롤백이 동작하지 않음
        - rollbackFor를 사용해야함
- RuntimeException : 런타임시에 발생하는 예외. 예외 처리를 강제하지 않음
    - e.g. NullPointerException, ArrayIndexOutOfBoundsException, etc
- 코틀린에서는 자바의 Exception 계층을 코틀린 패키지로 래핑한다

- 자바에서 체크드 익셉션은 컴파일 에러가 발생하기 때문에 무조건 try-catch로 감싸거나 throws로 예외를 전파해야한다

```
try {
  Thread.sleep(1);
} catch (InterruptedException e) {
    // 예외 처리
}
```

- 코틀린은 `체크드 익셉션` 을 강제하지 않는다
- 아래의 코드는 코틀린에서 컴파일 오류가 발생하지 않는다

```
Thread.sleep(1);
```

- 원하는 경우에는 try-catch를 쓸 수 있다

```
try {
  Thread.sleep(1)
} catch (e: Exception) {
```

```
  // 예외 처리
}
```

- 코틀린의 try-catch는 `표현식`이다

```
val a = try { "1234".toInt() } catch (e: NumberFormatException) { println("catch 동작") }
```

- `finally` 를 사용하면 try-catch의 마지막에 수행될 코드를 작성할 수 있다

```
try {
  throw Exception()
} catch (e: Exception) {
  println("catch 수행!")
} finally {
  println("finally 수행!")
}
```

- 코틀린에서 Exception을 발생시키려면 `throw` 를 사용한다

```
throw Exception("예외 발생!")


Exception in thread "main" java.lang.Exception: 예외 발생!
 at FileKt.main (File.kt:2)
 at FileKt.main (File.kt:-1)
 at jdk.internal.reflect.NativeMethodAccessorImpl.invoke0 (:-2)
```

- throw역시 표현식이기 때문에 throw를 리턴할 수 있다

```
fun failFast(message: String): Nothing {
    throw IllegalArgumentException(message)
}
```

- `Nothing` 타입을 사용하면 컴파일러는 해당 코드 이후는 실행되지 않는 다는 경고를 보여준다

```
failFast("에러발생!")

// Unreachable code
println("이 메시지는 출력될까?")
```

- 엘비스 연산자와 사용하면 null 안전 코드를 작성하지 않아도 된다

```
fun main() {
    val a: String? = null
    val b: String = a ?: failFast("a is null ")

    println(b.length)
}
```