Effective Java

Item 69 : 예외는 진짜 예외 상황에만 사용하라

Item 70 : checked exception, unchecked exception

Item 71 : 필요 없는 검사 예외 사용은 피해라

Item 72 : 표준 예외를 사용하라

Item 69

예외는 진짜 예외 상황에만 사용하라

Stack Trace

```java
public class Main {
    public static void main(String[] args) {
        Class1.method1();
    }

    static class Class1 {
        public static void method1() {
            Class2.method2();
        }
    }

    static class Class2 {
        public static void method2() {
            Class3.method3();
        }
    }

    static class Class3 {
        public static void method3() {
            throw new NullPointerException();
        }
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException
    at Main$Class3.method3(Main.java:20)
    at Main$Class2.method2(Main.java:14)
    at Main$Class1.method1(Main.java:8)
    at Main.main(Main.java:3)
```

예외를 다루는 클라이언트

```
try {
    obj.action(args);
} catch (TheCheckedException e) {
    // 예외 상황에 대처
}
```

예외에 정보를 담을 수 있음
try 블록 필요, 스트림에서 사용 불가

```
if (obj.actionPermitted(args)) {
    obj.action(args);
} else {
    // 예외 상황에 대처
}
```

멀티 쓰레드에서 문제 발생 가능성 존재
작업 일부를 중복 수행한다면 성능 하락

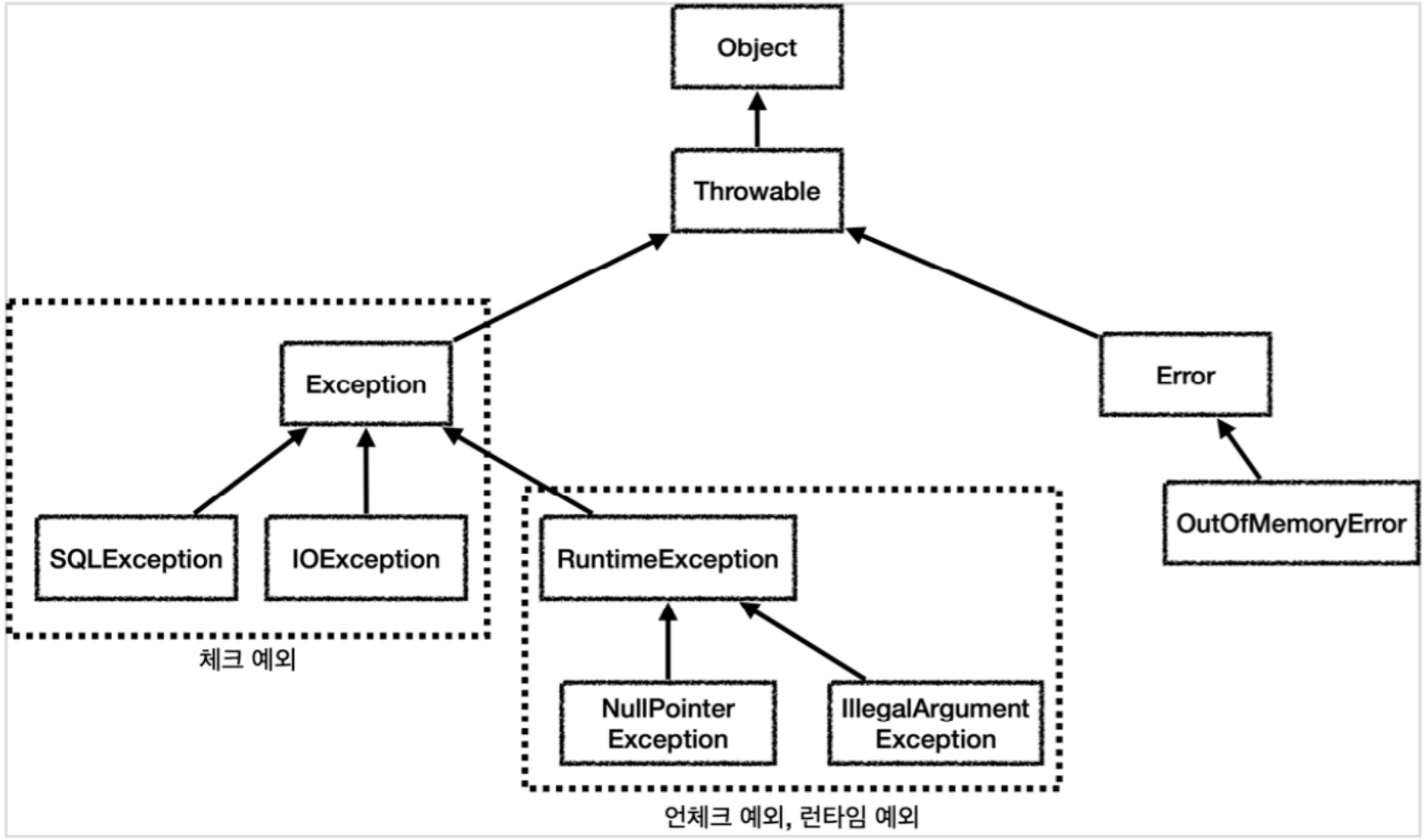
```
Optional<Object> ob = obj.action(args);
if (ob.isEmpty()) {
    // 예외 상황에 대처
}
```

예외가 발생한 부가 정보를 담을 수 없음

Item 70-71

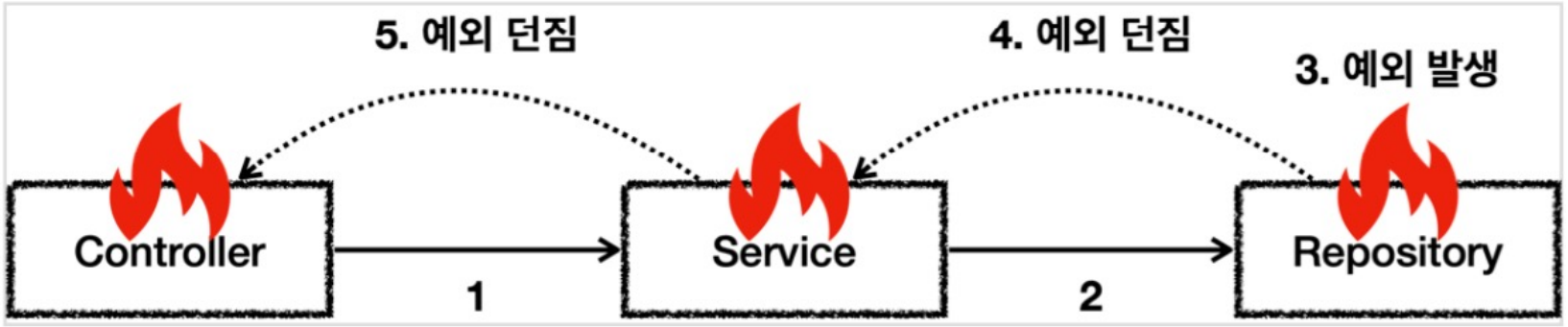
checked exception, unchecked exception

예외 계층


Object
Throwable
Exception
Error
SQLException
IOException
RuntimeException
OutOfMemoryError
NullPointer Exception
IllegalArgument Exception
체크 예외
언체크 예외, 런타임 예외

예외는 폭탄 돌리기다


5. 예외 처리
4. 예외 던짐
3. 예외 발생
6. 정상 흐름 반환
Controller
Service
Repository
1
2


5. 예외 던짐
4. 예외 던짐
3. 예외 발생
Controller
Service
Repository
1
2

checked exception vs. unchecked exception

컴파일러를 통한 안전 장치 vs. 번거로움 해결

기본 원칙

1. 기본적으로 **unchecked exception**을 사용하자
2. 비즈니스 로직상 의도적으로 던지는 예외는 checked exception을 사용하자
   - 해당 예외를 잡아서 반드시 처리해야 하는 문제일 때만 사용
   - 또는 일반적으로 클라이언트가 복구할 수 있는 문제일 때

unchecked exception

문서화를 잘하자

```
/**
 * Issue a single SQL execute, typically a DDL statement.
 * @param sql static SQL to execute
 * @throws DataAccessException if there is any problem
 */
void execute(String sql) throws DataAccessException;
```

예외가 명확하고 중요하다면,
throws에 명시해줘도 괜찮다.

예외 전환 시에는 기존 예외를 포함하자

디버깅, 로그 스택 트레이스를 위해

```
void doSomething() {
    try {
        repository.query();
    } catch (SQLException e) {
        throw new RuntimeSQLException(e);
    }
}
```

checked exception은 롤백되지 않는다.

Service Layer

```
@Transactional
public void saveShopWithUncheckedException() {
    Shop shop = new Shop();
    shopRepository.save(shop);
    throw new RuntimeException();
}

@Transactional
public void saveShopWithCheckedException() throws Exception {
    Shop shop = new Shop();
    shopRepository.save(shop);
    throw new Exception();
}
```

Test

```
@Test
@DisplayName("언체크 예외는 롤백된다.")
void saveShopWithUncheckedException() {
    assertThrows(RuntimeException.class,
            () -> shopService.saveShopWithUncheckedException());

    assertEquals(0, shopRepository.count());
}

@Test
@DisplayName("체크 예외는 롤백되지 않는다.")
void saveShopWithCheckedException() {
    assertThrows(Exception.class,
            () -> shopService.saveShopWithCheckedException());

    assertEquals(1, shopRepository.count());
}
```

Item 72

표준 예외를 사용하라

널리 재사용되는 예외

예외	주요 쓰임
IllegalArgumentException	허용하지 않는 값이 인수로 건네졌을 때(null 제외)
IllegalStateException	객체가 메서드를 수행하기에 적절하지 않은 상태일 때
NullPointerException	null을 허용하지 않는 메서드에 null을 건넸을 때
IndexOutOfBoundsException	인덱스가 범위를 넘어섰을 때
ConcurrentModificationException	허용하지 않는 동시 수정이 발견됐을 때
UnsupportedOperationException	호출한 메서드를 지원하지 않을 때

그 외

Exception, RuntimeException, Throwable, Error은 직접 재사용하지 말자

너무 상위 클래스이므로(여러 예외를 포괄하므로) 안정적으로 테스트할 수 없다.

IllegalStateException과 IllegalArgumentException

인수 값이 무엇이었든 어차피 실패했을 거라면 IllegalStateException

그렇지 않다면 IllegalArgumentException