chapter 7: Error Handling

Use Exceptions

- return code 사용하면 (i.e. `if(getHandle()!=Handle.INVALID)`)
  호출 이후 바로 error 체크가 돌아간다는 문제가 있다.
- 따라서 exception으로 뺴고, try catch를 사용하자.
- Java의 경우 checked exceptions가 지원되긴 하지만 사용하지 말자.
- 예외처리할때 메시지 같이 쓰는 것을 습관화 하자 (provide context)

Wrapping External API

외부 라이브러리를 사용하면 종종 동작과 예외처리가 길어질 수 있다.

이런 경우 "감싸는" 클래스를 하나 정의해서 클래스 메소드 안에 구체적인 예외처리를 구현하고, 호출할 때에는 함수 하나만 호출하도록 하자.

Define Normal Flow

```
try {
  MealExpenses expenses = expenseReportDAO.getMeals(employee.getID());
  m_total += expenses.getTotal();
} catch(MealExpensesNotFound e) {
  m_total += getMealPerDiem();
}
```

- 사실 위 exception 자체를 발생시키지 않게 할 수 있다.

  getMeals 함수 안에서 알아서 not found 상황에 per diem으로 계산하도록 하면 아래와 같이 코드를 예쁘게 쓸 수 있다.

```
MealExpenses expenses = expenseReportDAO.getMeals(employee.getID());
m_total += expenses.getTotal();
```

Null

- `if (item!=null)` 과 같은 코드를 쓰면 치명적인 문제가 생긴다.
  - 바로 모든 곳에서 null check를 해야 한다는 것이다. 그리고 하나라도 놓치면 nullexception을 찾아서 헤매게 될 것이다...

- 따라서 null을 반환하게 하지 말고, empty list와 같은 식으로 반환해서 코드에서 체크를 할 필요가 없도록 함수를 짜자
- 또한 함수의 인자로 null을 넘겨주는 것도 좋지 않다.
    - 유저의 문제로 null이 입력될 수 있다면 `assert`를 사용하자