Clean Code

-error handling-

AlphaCircle. XIO. 220826.

1. Exception을 사용하라

오류 코드보다 예외를 사용하라

- 실행 알고리즘과 오류 처리 알고리즘을 분리한다.
- try, catch 블록에 들어갈 코드를 따로 메소드로 구분하여 처리한다

```
public void sendShutDown() {
    try {
        tryToShutDown();
    } catch (DeviceShutDownError e) {
        logger.log(e);
    }
}

private void tryToShutDown() throws DeviceShutDownError {
  DeviceHandle handle = getHandle(DEV1);
  ...
}

private DeviceHandle getHandle(DeviceID id) {
  ...
  throw new DeviceShutDownError("Invalid handle for: " + id);
  ...
}
```

➡️ 오류가 발생하는 로직의 코드가 깔끔해 보일 수 있도록(개념이 뒤섞이지 않도록)
위와 같이 메소드를 분리하자

2. try-catch-finally 순서로 작성하라

try-catch-finally문 순서로 작성하라

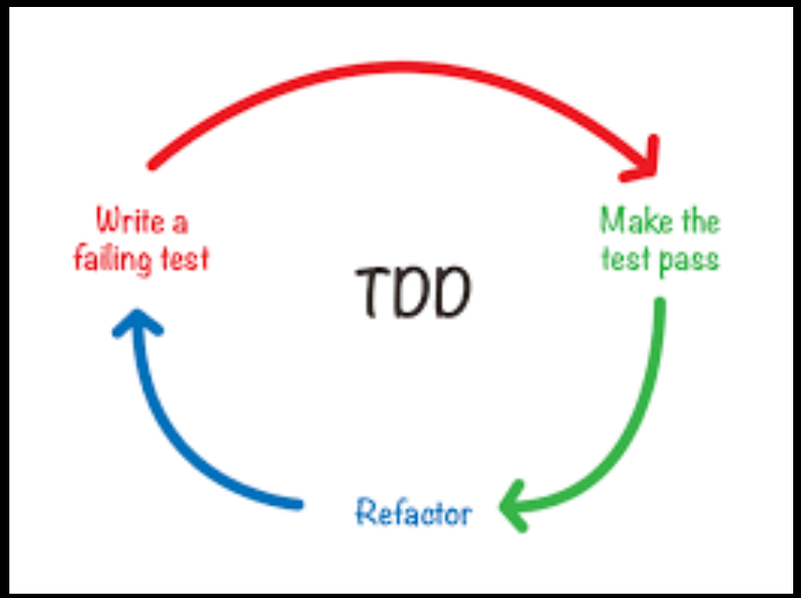
- 먼저 try-catch 구조로 범위를 정의한다.
- 다음으로 TDD를 사용해 필요한 나머지 논리를 추가한다.
  (강제로 예외를 일으키는 테스트 케이스 작성 후
  테스트를 통과하게 코드를 작성하는 방법을 권장한다.
  자연스럽게 try 블록의 트랜잭션 범위부터 구현하게 되므로
  트랜잭션 본질을 유지할 수 있다.)

```java
public List<RecordedGrip> retrieveSection(String sectionName) {
    try {
        FileInputStream stream = new FileInputStream(sectionName);
        stream.close();
    } catch (FileNotFoundException e) {
        throw new StorageException("retrieval error", e);
    }
    return new ArrayList<RecordedGrip>();
}
```

➡️ 실패 테스트 코드부터 작성하여 테스트 하도록 한다.

try-catch-finally문 순서로 작성하라

✴️ Q. TDD란?



✅ Test-Driven Development(테스트 주도 개발)

✅ 프로덕션 코드보다 단위 테스트 코드를 먼저 작성하여
코드를 빠르게 검증하고 리팩토링 시 안정성을 확보한다.

✅ 개발 및 테스트 시간 비용을 절감할 수 있다.

[TDD(Test-Driven Development, 테스트 주도 개발) 방법 및 순서]

1️⃣ 실패하는 작은 단위 테스트를 작성한다. 처음에는 컴파일조차 되지 않을 수 있다.
2️⃣ 테스트를 통과하기 위해 프로덕션 코드를 빨리 작성한다. (가짜 구현 등)
3️⃣ 그 다음의 테스트 코드를 작성한다. 실패 테스트가 없을 경우에만 성공 테스트를 작성한다.
4️⃣ 새로운 테스트를 통과하기 위해 프로덕션 코드를 추가 또는 수정한다.
5️⃣ 1~4단계를 반복하여 실패/성공의 모든 테스트 케이스를 작성한다.
6️⃣ 개발된 코드들에 대해 모든 중복을 제거하며 리팩토링한다.

3. 예외 클래스를 정의하라

예외 클래스를 정의하라

```
LocalPort port = new LocalPort(12);
try {
    port.open();
} catch (PortDeviceFailure e) {
    reportError(e);
    logger.log(e.getMessage(), e);
} finally {
    ...
}
```

```
public class LocalPort {
    private ACMEPort innerPort;
    public LocalPort(int portNumber) {
        innerPort = new ACMEPort(portNumber);
    }
    public void open() {
        try {
            innerPort.open();
        } catch (DeviceResponseException e) {
            throw new PortDeviceFailure(e);
        } catch (ATM1212UnlockedException e) {
            throw new PortDeviceFailure(e);
        } ...
}
```

✴️ 감싸기 기법

- 여기서 LocalPort는 단순히 예외를 잡아 변환하는 wrapper 클래스
- 외부 API를 사용할 때 wrapper클래스를 만들어
  호출하는 라이브러리 API를 감싸서 예외를 처리하도록 한다.
- 외부 라이브러리와 프로그램 사이 의존성이 크게 줄어들며
  외부 API 설계 방식에 발목 잡히지 않는다.
- 예외 클래스에 포함된 정보로 오류를 구분하는 경우 예외 클래스 하나로 처리할 수 있다.

4. 정상 흐름을 정의하라

정상 흐름을 정의하라

- 특수 상황을 처리할 필요가 없도록 하라.

✴️ SPECIAL CASE PATTERN

- 클래스를 만들거나 조작해 특수 사례를 처리하는 방식

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

➡️ 예외가 논리를 따라가기 어렵게 만든다.

VS

```
MealExpenses expenses = expenseReport.ge
m_total += expenses.getTotal();
```

클라이언트 코드가
예외 상황을 처리할 필요 없도록
객체가 예외 상황을 캡슐화하여 처리

```
public class PerDiemMealExpenses implements MealExpenses {
    public int getTotal() {
        ...
        // default setting
        return MealExpenses(DEFAULT_EXPENSES);
    }
}
```

➡️ 특수 사례 패턴을 적용하면 예외없이 처리 로직에 집중하며 코드를 따라갈 수 있다.

5. null 반환/전달하지 마라

null을 반환/전달하지 마라

- 한줄 마다 null을 확인하고 반환하는 코드를 쓰지마라.
- null을 반환하는 코드는 일거리를 늘리고 호출자에게 문제를 떠넘긴다.
- 대신 예외를 던지거나 특수 사례 객체를 반환하도록 한다.
- 특히 null을 전달하는 메소드 작성은 최대한 피한다.

➡️ Java: @NotNull등의 어노테이션 사용으로 null이 인수로 전달될 경우에 발생할 예외를 최대한 줄인다.

➡️ JS: 특수 사례 패턴을 적용하거나 예외 처리 한다.
ex) 비동기에서 Promise객체 적용하는 패턴을 활용

➡️ 예외를 던질 때 오류 발생 원인, 위치, 전후 상황,
실패한 연산 이름, 실패 유형등 충분한 정보를 로그로 남겨 준다.

6. Refactoring Code

Refactoring Code

✴️ 결제 정보를 받아와 저장하는 코드

```
const storeIAPReceipt = () => {
  IAP.getAvailablePurchases()
  ...
  try {
    ...
    /* purchase[itemId]가 없거나 purchaseToken이 없을 수 있음 */
    AsyncStorage.setItem('receipt', purchase[itemId].purchaseToken);
  } catch (e) {
     /* show error toast by message-handling util */
     return await clientMessenger(e.name, e.message)
  }
};
```

⬇️

```
const storeIAPReceipt = () => {
  IAP.getAvailablePurchases()
  ...
  /* INVALID_IAP_TOKEN 이라는 기본 값을 설정하여 불필요한 try-catch 로직 제거 */
  AsyncStorage.setItem('receipt', purchase[itemId].purchaseToken ?? INVALID_IAP_TOKEN);
};
```

✅ bullish coalescing (??) 연산자를 사용하여 예외가 발생하지 않도록 기본 값을 지정해 준다.