Clean Code

3. 함수

작게 만들어라

챕터 요약

- **함수를 만들 때 가장 우선되어야 할 규칙은 "작게 만들기" 이다.**
- **분기문, 반복문 등에 들어가는 블록은 한 줄이어야 한다.**
  - 블록 안에서 함수를 호출 하도록 하고, 그 함수의 이름을 적절히 짓는다면, 코드 가독성이 높아진다.
- **함수 내부에서 들여쓰기의 수준은 2단 이내로 제한해야 가독성이 좋다.**

예시 코드

```java
/** 일반적으로 한 함수의 크기는
 *  이 함수정도의 크기로 만드는 것을 권장한다.
 */
public static String renderPageWithSetupsAndTeardowns(
    PageData pageData, boolean isSuite) throws Exception {
    if (isTestPage(pageData)){
        includeSetupAndTeardownPages(pageData, isSuite);
    return pageData.getHtml();
}
```

의견

- **FP에서는 closure 를 사용하는 경우가 많아서 이 규칙을 완벽하게 지키기 쉽지 않을 것 같지만 내부에 있는 closure 들을 작게 만들도록 노력할 필요는 있다.**
- **비동기 프로그래밍을 할 때 callback 구조로 만드는 경우가 많은데 callback 이 중첩되는 경우 callback hell 문제를 일으켜서 당연히 중첩블록이 많아진다. js에서는 async await 등의 문법을 사용해 이 문제를 해결 할 수 있고, java에서는 람다식의 Method reference 나 java.util.function 패키지 등을 사용해 몇가지 문제를 해결할 수 있지..만 그냥 kotlin 을 쓰는게 훨씬 좋다고 생각한다.**

한가지만 해라

챕터 요약

- **함수는 한가지를 해야 한다. 그 한가지를 잘 해야 한다. 그 한가지만을 해야 한다.**
- **함수가 “한가지만 한다” 를 판단하는 기준**
  - 지정된 함수 이름 아래 추상화 수준이 하나인 단계만 수행한다.
  - 단순히 다른 표현이 아니라 의미 있는 이름으로 다른 함수를 추출할 수 있다면 그 함수는 여러 작업을 하는 함수이다.

의견

- **“한가지” 라는 것을 정의하는 것도 중요하지만, 그 한가지 작업을 나타내는 함수의 네이밍을 잘 정해야 “한가지”를 잘 정의하고, 추상화 수준을 잘 나눌 수 있다고 생각한다.**

함수 당 추상화 수준은 하나로 (1/2)

챕터 요약

- **함수가 '한 가지 작업' 을 한다는 뜻은 함수 내 모든 문장의 추상화 수준이 동일하다 라는 뜻이다.**
  - 추상화 수준이란 해당 코드을 읽으면서 파악할 수 있는 정보의 수준을 의미한다.
- **한 함수 다음에는 추상화 수준이 한 단계 낮은 함수가 오도록 코드를 구성해야한다.**
  - 책을 위에서 아래로 내려가며 읽는 것처럼 코드를 구성해야 한다.

추상화 수준? (1/2)

- **다음 내용을 함수로 만든다고 생각해보자**
  1. App Cache Storage로 부터 Access Token 을 가져온다.
  2. Access Token이 유효한지 검증한다.
  3. 만약 유효하다면 API Server에서 Category 에 대한 private 데이터를 가져온다.
  4. Category private 데이터를 받아왔으면 UI에 private 데이터를 포함한 정보를 UI에 표시한다.
  5. 만약 Access Token이 유효하지 않다면 Category 에 대한 public 데이터를 가져온다.
  6. Category public 데이터를 받아왔으면 UI에 public 데이터를 UI에 표시한다.

- **말로 표현하면 간단해 보이지만 실제 code 로 표현하면 몇십 라인의 코드를 하나의 함수에 작성 해야한다.**

함수 당 추상화 수준은 하나로 (2/2)

추상화 수준? (2/2)

의견

- 함수 2-1 과 2-3을 추상화 단계 2 (책의 표현에 따르면 추상화 중간 단계) 로 놓아야 할까 단계 3에 놓아야 할까 애매한 부분이 있다고 생각한다.
- 함수의 개수가 다소 많아지게 되더라도 추상화 수준은 높음/낮음의 2단계로만 나눠서 개발하는 것이 좋다고 생각한다.
- 위에서 아래로 코드를 작성하도록 하는 규칙은 동의하지만, 한 클래스 내에 추상화 수준이 동일한 함수가 여러 개 있는 경우에는 함수의 배치에 대해 좀 더 고민을 해야 할 필요가 있다고 생각한다.

Switch 문

챕터 요약

- **Switch 문은 최대한 지양한다.**
- **Switch 문이 필요한 경우, abstract factory에 해당 구문을 숨기고 관련 코드에는 단 한번의 switch 만 사용하라.**
  - abstract factory 패턴에 대한 설명 : https://flower0.tistory.com/416

의견

- **실제로 avpt에서 RTSP 와 기존 local play를 구분하는 로직이 app단에서 필요해서, alphaVR의 외부 로직에서 switch로 분기했더니 내부 로직이 줄줄이 switch 구문이 필요한 경우가 있었다. Switch는 최대한 지양해야 함수가 변경되었을 때 줄줄이 변경해야 하는 문제가 발생하지 않을 것 같다.**
  - switch 문제 발생 코드 예시 : http://gitlab.alphacircle.co.kr/alphacircle/avpt-exoplayer-2.17.1/-/blob/master/alphaVR/src/main/java/kr/co/alphacircle/alphavr/AcVrPlayerActivity.java

함수 인수

챕터 요약

- **인수는 적을 수록 좋다.**
  - 2개 이하의 인수를 사용하라.
- **함수를 만들 때 인수가 많아진다면, 함수를 단항 함수/ 인수가 없는 함수로 나눠라**
- **단항 함수는 함수와 인수가 동사/명사 쌍을 이루도록 명명하라**
  - ex) void write(File filename)
- **플래그 인수는 지양하라**
- **인수가 꼭 2~3개 필요하다면 인수를 묶어 클래스로 표현하라**
  - ex) class Point(int x, int y)

의견

- **적은 수의 인수가 있는 함수가 더 이해하기 쉽다는 부분에는 동의하나, 몇가지 딴지를 걸만한 여지가 있다.**
  - 리액트의 경우 컴포넌트를 만들 때 어쩔 수 없이 여러 Props을 받아야하는 상황이 많고 interface 로 props를 묶더라도 내부 컴포넌트들에 따로 props가 필요한 경우가 발생한다.
    - 그리고 실제로 사용할 때는 따로 props의 값을 설정하는 것이 일반적이다.
  - 안드로이드의 exoplayer는 엄청나게 많은 인수를 가진 함수들이 다수 존재한다.
    - 물론 팩토리와 체이닝 메소드를 사용해서 인수사용을 줄이는 코드도 같이 존재한다.

부수 효과를 일으키지 마라

챕터 요약

- **함수가 한가지 일만 담당하도록 설계하여 부수 효과를 일으키지 말고, 시간적 결합이 필요하다면 함수 이름에 명확하게 명시하라**
  - 시간적 결합 : A 함수는 반드시 B 함수보다 먼저 실행되어야 한다.
  - 부수 효과 : 함수 내의 실행으로 인해 **함수 외부가 영향**을 받는 것을 의미한다.
    - https://jinwooe.wordpress.com/2017/12/21/%EB%B6%80%EC%88%98-%ED%9A%A8%EA%B3%BC-side-effect-%EC%B0%B8%EC%A1%B0-%ED%88%AC%EB%AA%85%EC%84%B1-referential-transparency/
- **출력 인수는 되도록이면 피해라 함수에서 상태를 변경해야 한다면 함수가 속한 객체의 상태를 변경하라.**

예시 코드

```
/* 이 함수를 refactoring 한다면, checkPasswordAndInitializeSession
 * 이라는 이름으로 변경하는 것이 좋다.
 */
public boolean checkPassword(String userName, String password){
  User user = UserGateway.find(userName);
  if (user != User.NULL){
    if (isValidUser(user)){
      /* Side Effect 발생!!
       * 함수 이름에 세션을 초기화한다는 정보가 없다.
       * checkPassword 함수는 세션을 초기화해도 괜찮은 경우에만
       * 호출할 수 있는 함수가 된다.
       */
      Session.initialize();
      return true;
    }
  }
  return false;
}
```

의견

- **React.useEffect 는 함수 이름 부터 side effect를 일으키겠다 라고 명시하는 함수이다. deps에 설정한 값에 side effect가 발생하게 된다. React 에서는 side effect 가 거의 필수적으로 필요하긴 하지만 useEffect 내부 로직을 좋은 이름의 함수를 실행하도록 하면 조금 더 좋은 코드가 되지 않을까 라고 생각된다.**
- **ExoPlayer 에서 source 를 읽어 들이는 코드를 보면 read 함수의 인자로 넘긴 출력인수를 변경하는 코드들이 즐비해 있는데 이런 코드들로 인해 더 이해하기 어려운 코드가 생긴다.**

명령과 조회를 분리하라.

챕터 요약

- **명령을 하는 하는 함수는 명령만, 조회를 하는 함수는 조회만 하도록 함수를 설계하라.**
  - ex) set 함수에서 결과값을 반환하는 등의 처리는 혼란을 부추길 수 있다.

오류 코드보다 예외를 사용하라

챕터 요약

- **예외 구문 (try catch) 을 사용하고, 오류 처리도 하나의 함수로 분리하라**
  - 명령을 담당하는 함수에서 오류코드를 반환한다면 명령/조회 분리 규칙을 위반한다.
- **오류 코드를 열거형으로 정의하지 말고 Exception 클래스에서 파생된 예외를 사용하라.**

의견

- **예외 처리는 비용이 비싸다 exception을 주의해서 다루지 않는다면 좋은 코드를 만든다는 명목으로 더 중요한 성능 지표가 떨어지는 결과를 초래할 수 있다. 예외 구문을 사용하되, 주의해서 사용하자**
  - 예외 처리 비용에 관한 포스트 : https://meetup.toast.com/posts/47
- **alphaVR에도 상황에 따른 오류 코드를 activity result로 반환한다. 해당 오류코드 값이 무엇을 의미하는지 정의 파일이나 문서를 찾아본 적이 몇 번이나 있었다.**

반복하지마라

챕터 요약

- DRY (Don’t Repeat Yourself)

의견

- DRY 원칙은 어떤 소프트웨어의 형태에서도 지키도록 노력해야하는 원칙이다. 다만 함수나 클래스를 반복을 피하기 위해 분리할 때는 나중에 관련 함수/클래스의 변경이 일어났을 때, 재귀적으로 변경이 일어나는 문제가 발생하지 않도록 잘 설계해서 분리하자.

결론

- **함수를 만들 때 가장 우선적으로 지켜야 할 두가지 규칙**
  - 하나의 함수가 한가지 역할만 담당하라
  - 부수효과를 일으키지 마라
- **함수를 처음 만들 때는 다소 길어도 좋다. 다만 본인이 생각하기에 함수가 길어진다면 그 때는 함수를 쪼개야 할 때 임을 상기하도록 하자.**
- **저자가 이야기하는 모든 원칙을 지킬 필요까지는 없다고 생각한다. 다만 가장 중요한 2가지 규칙을 지켜서 개발하려고 노력하면 더 좋은 코드를 작성할 수 있을 것이라고 생각한다.**