

# 2주 차 공통 피드백

## README.md를 상세히 작성한다

미션 저장소의 README.md는 소스코드에 앞서 해당 프로젝트가 어떠한 프로젝트인지 마크다운으로 작성하여 소개하는 문서이다. 해당 프로젝트가 어떠한 프로젝트이며, 어떤 기능을 담고 있는지 기술하기 위해서 마크다운 문법을 검색해서 학습해 보고 적용해 본다.

## 기능 목록을 재검토한다

기능 목록을 클래스 설계와 구현, 함수(메서드) 설계와 구현과 같이 너무 상세하게 작성하지 않는다. 클래스 이름, 함수(메서드) 시그니처와 반환값은 언제든지 변경될 수 있기 때문이다. 너무 세세한 부분까지 정리하기보다 구현해야 할 기능 목록을 정리하는 데 집중한다. **정상적인 경우도 중요하지만, 예외적인 상황도 기능 목록에 정리한다.** 특히 예외 상황은 시작 단계에서 모두 찾기 힘들기 때문에 기능을 구현하면서 계속해서 추가해 나간다.

## 기능 목록을 업데이트한다

README.md 파일에 작성하는 기능 목록은 기능 구현을 하면서 변경될 수 있다. 시작할 때 모든 기능 목록을 완벽하게 정리해야 한다는 부담을 가지기보다 기능을 구현하면서 문서를 계속 업데이트한다. 죽은 문서가 아니라 살아있는 문서를 만들기 위해 노력한다.

## 값을 하드 코딩하지 않는다

문자열, 숫자 등의 값을 하드 코딩하지 마라. 상수(static final)를 만들고 이름을 부여해 이 변수의 역할이 무엇인지 의도를 드러내라. 구글에서 "java 상수"와 같은 키워드로 검색해 상수 구현 방법을 학습하고 적용해 본다.

## 구현 순서도 코딩 컨벤션이다

클래스는 상수, 멤버 변수, 생성자, 메서드 순으로 작성한다.

```
class A {  
    상수(static final) 또는 클래스 변수  
  
    인스턴스 변수  
  
    생성자
```

```
    메서드  
}
```

## 변수 이름에 자료형은 사용하지 않는다

변수 이름에 자료형, 자료 구조 등을 사용하지 마라.

```
String carNameList = Console.readLine();  
String[] arrayString = carNameList.split(",");
```

## 한 함수가 한 가지 기능만 담당하게 한다

함수 길이가 길어진다면 한 함수에서 여러 일을 하려고 하는 경우일 가능성이 높다. 아래와 같이 한 함수에서 안내 문구 출력, 사용자 입력, 유효값 검증 등 여러 일을 하고 있다면 이를 적절하게 분리한다.

```
public List<String> userInput() {  
    System.out.println("경주할 자동차 이름을 입력하세요(이름은 쉼표(,)를 기준으로  
구분).");  
    String userInput = Console.readLine().trim();  
    String[] splittedName = userInput.split(",");  
    for (int index = 0; index < splittedName.length; index++) {  
        if (splittedName.length < 1 || splittedName.length > 5) {  
            throw new IllegalArgumentException("[ERROR] 자동차 이름은 1자 이상  
5자 이하만 가능합니다.");  
        }  
    }  
    return Arrays.asList(splittedName);  
}
```

## 함수가 한 가지 기능을 하는지 확인하는 기준을 세운다

만약 여러 함수에서 중복되어 사용되는 코드가 있다면 함수 분리를 고민해 본다. 또한, 함수의 길이를 15라인을 넘어가지 않도록 구현하며 함수를 분리하는 의식적인 연습을 할 수 있다.

## 테스트를 작성하는 이유에 대해 본인의 경험을 토대로 정리해본다

단지 기능을 점검하기 위한 목적으로 테스트를 작성하는 것은 아니다. 테스트를 작성하는 과정을 통해서 나의 코드에 대해 빠르게 피드백을 받을 수 있을 뿐만 아니라 학습 도구([학습테스트를 통해 JUnit 학습하기.pdf](#))로도 활용할 수 있다. 이런 경험을 통해 테스트에 대해 어떤 유용함을 느꼈는지 알아본다.

## 처음부터 큰 단위의 테스트를 만들지 않는다

테스트의 중요한 목적 중 하나는 내가 작성하는 코드에 대해 빠르게 피드백을 받는 것이다. 시작부터 큰 단위의 테스트를 만들게 된다면 작성한 코드에 대한 피드백을 받기까지 많은 시간이 걸린다. 그래서 문제를 작게 나누고, 그 중 핵심 기능에 가까운 부분부터 작게 테스트를 만들어 나간다.

### 큰 단위의 테스트

- 자동차경주를 시작해서 사용자가 이름, 진행 횟수를 입력하면, 게임을 진행한 후 그 결과를 알려준다.

### 작은 단위의 테스트

- 무작위 값이 4 이상이면 자동차가 전진한다.
- 무작위 값이 3 이하이면 자동차가 전진하지 않는다.