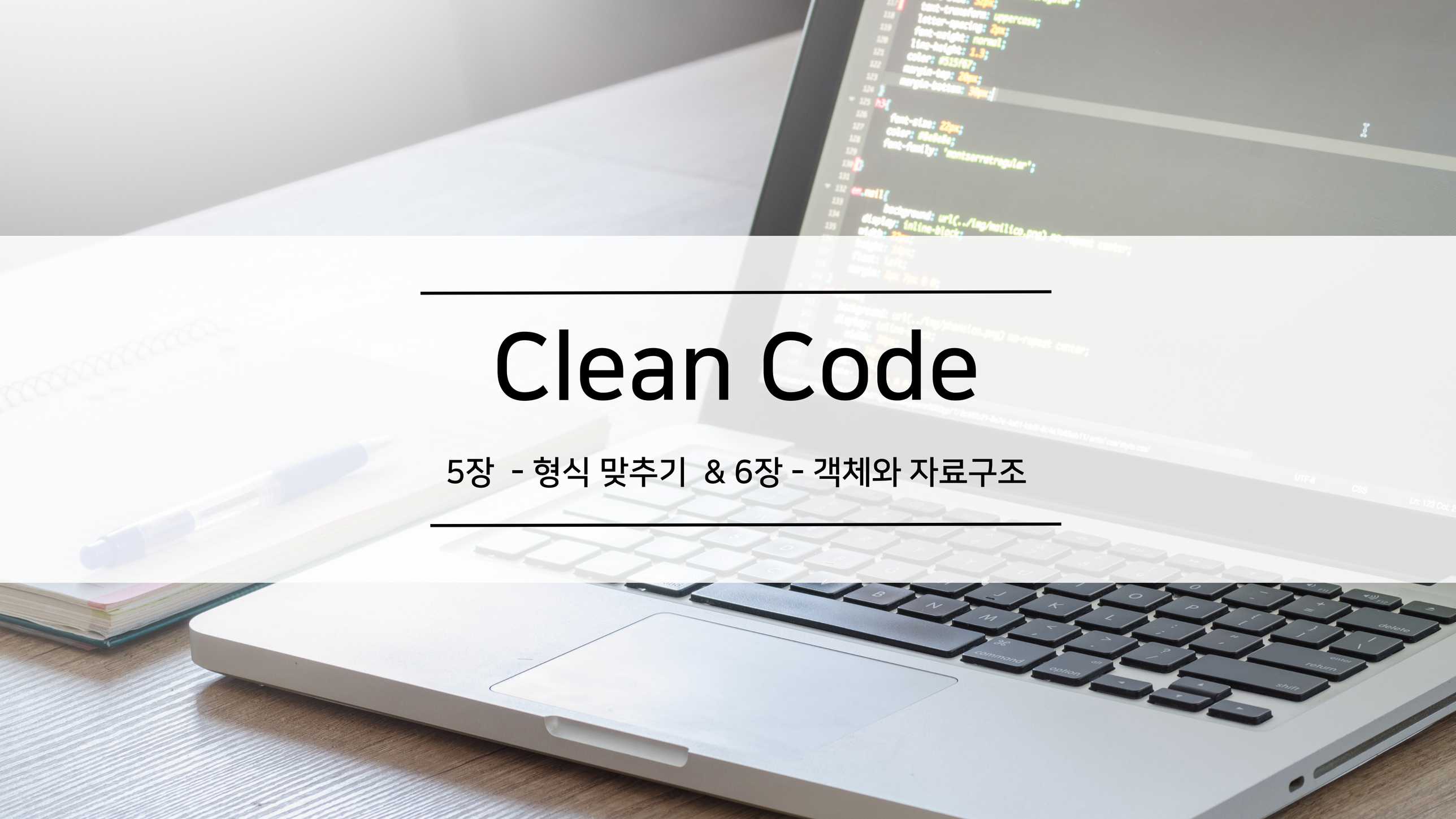

Clean Code

5장 - 형식 맞추기 & 6장 - 객체와 자료구조

왜 형식을 맞춰야 하는가?

전문적인 인상

코드를 봤을 때 **깔끔하고 일관적이면** 전문가가 짰다는 인상

그렇지 않다면 무성의하게 짰을 거라고 생각

의사소통

코드 형식은 의사소통의 일환.

계속해서 변경되는 코드는 **가독성이 유지보수 용이성과 확장성에 영향**을 미친다.

적당한 행 길이를 유지하라

소스 코드의 길이

클래스의 크기는

500줄을 넘지 않고 200줄 정도

신문 기사처럼 작성하라

소스 파일 이름은 간단하면서도 설명이 가능하게

고차원 개념과 알고리즘으로 시작해

아래로 갈수록 저차원 함수와 세부 내역

개념은 빈 행으로 분리

일련의 행 묶음은 완결된 생각 하나

패키지 선언, import 문, 각 함수 사이에 빈 행을 넣어서 분리

적당한 행 길이를 유지하라

세로 밀집도

변수끼리, 메서드끼리 모아놓아라

수직 거리

서로 밀접한 개념은 한 파일에 속해야

protected 변수를 피해라

변수 선언
지역,인스턴스 변수는 함수의 맨 처음
루프 제어 변수는 루프 문 내부

종속 함수
한 함수가 다른 함수를 호출한다면 가까이 배치하고 호출하는 함수가 호출되는 함수보다 먼저 배치

개념적 유사성
친화도가 높은 코드일수록 코드를 가까이 배치
ex.명명법이 똑같고 기본 기능이 유사한 함수들 (assertXXX, setXXX, getXXX)

가로 형식 맞추기

가로 길이

짧은 행이 바람직. 120자 정도

가로 공백과 밀집도

할당 연산자 앞뒤에 공백

함수 이름과 괄호는 공백 없이

함수 인자는 쉼표 다음에 공백

연산자 우선순위를 표현하기 위함

들여쓰기 & 가짜범위

짧은 if, while, 함수 같은 경우

- 한 줄에 쓰기 말고 들여쓰기를 해서 범위를 제대로 표현

비어있는 while문이나 for문의 경우

- 다음 행에 세미콜론을 넣어서 표현

팀 규칙

- 팀은 한 가지 규칙에 합의해야 한다.
  - 괄호는 어디에 쓸건지, 클래스와 변수 메서드 이름은 어떻게 지을지 등등..

함수 이름

- ViewModel을 observe()할때 모아놓는 함수 이름

```
setupXXX()
```

- 서버에서 데이터를 불러올때 함수 이름

```
fetchXXX()
```

- 서버에 저장할때 함수 이름

```
saveXXX()
```

- Return이 있는 데이터를 불러올때 함수 이름

```
getXXX()
```

- 특정 객체를 찾는 함수이름

```
findXXX()
```

- 복수형을 가져올때는 뒤에 s를 붙인다

```
getBrands()      // O
getBrandList()   // X
```

개행

- 생성자, 함수에서 Parameter를 정의할때 한줄로 정의 가능하면 한줄, 그렇지 않으면 각 parameter별로 개행한다.

When Statement

- 한 줄에 들어가는 when 분기는 중괄호를 사용하지 않는다.

```
when (value) {
    0 -> return
    // ...
}
```

- 여러개의 조건을 동시에 사용하는 경우 `->` 를 포함한 블록은 내려서 작성한다.

```
when (value) {
    foo -> // ...
    bar,
    baz
    -> return
}
```

출처: https://github.com/PRNDcompany/android-style-guide/blob/main/Kotlin.md

자료 추상화

- 사용자가 구현을 모른 채 자료의 핵심을 조작할 수 있어야 진정한 의미의 클래스
- 자료를 세세하게 공개하기보다는 추상적인 개념으로 표현

```
interface Vehicle {
    fun getFuelTankCapacityInGallons(): Double
    fun getGallonsOfGasoline(): Double
}
```

```
interface Vehicle {
    fun getPercentFuelRemaining(): Double
}
```

자료/객체 비대칭

- **객체**는 추상화 뒤로 자료를 숨긴채 **자료를 다루는 함수만** 제공
- **자료구조**는 자**료를 그대로 공개**하며 별다른 함수는 제공하지 않음
- **자료구조**를 사용하는 절차적인 코드
    - 기존 자료구조를 변경하지 않으면서 **새 함수를 추가하기 쉬움**
- **객체 지향** 코드는
    - 기존 함수를 변경하지 않으면서 **새 클래스를 추가하기 쉽다.**

디미터 법칙

- 휴리스틱 (경험적인, 직관적 판단에 의하여 문제를 해결하는 방식)
- 모듈은 자신이 조**사하는 객체의 속사정을 몰라야** 한다.
- 클래스 C의 메서드 f는 다음과 같은 객체의 메서드만 호출해야 한다.
  - 클래스 C
  - f가 생성한 객체
  - f 인수로 넘어온 객체
  - C 인스턴스 변수에 저장된 객체

```
val outputDir:String = ctxt.getOptions().getScratchDir().getAbsolutePate()
```

구조체 감추기

- 잡종 구조
  - 자료구조와 객체가 섞여있는 형태
  - 중요한 기능을 하는 함수와 (객체지향) 공개 조회/설정 함수로 비공개 변수를 노출하는 함수 (자료구조)가 섞여있어 새로운 함수는 물론 새로운 자료 구조도 추가하기 어려움

```
val outputDir:String = ctxt.getOptions().getScratchDir().getAbsolutePate()
```

```
val bos: BufferedOutputStream = ctxt.createScratchFileStream(classFileName)
```

자료 전달 객체

- 공개 변수만 있고 함수가 없는 클래스 (자료구조체)
- Data Transfer Object (DTO)라 부른다.
- 데이터베이스와 통신하거나 가공되지 않은 데이터베이스 정보를 애플리케이션코드에서 사용할 객체로 변환하는 단계에서 사용

활성 레코드

- DTO의 특수한 형태
- 자료구조에서 조회/설정 뿐만 아니라 save, find같은 탐색 함수도 제공
- 데이터베이스 테이블이나 다른 소스에서 자료를 직접 변환한 결과.
- 비즈니스 규칙을 담으면서 내부 자료를 숨기는 객체는 따로 생성

결론

객체

동작을 공개하고 자료를 숨김

새 객체 타입 쉬움

새로운 동작 추가 어려움

자료구조

별다른 동작없이 자료를 노출

새 동작 추가 쉬움

새 자료구조 추가 어려움

시스템 구현 시, 새로운 자료 타입 추가 or 새로운 동작 추가 유연성의 필요에 따라 객체 or 자료구조 선택