Clean Code

5. 형식 맞추기

개요

- **형식을 맞추는 목적**
  - 개발자의 일차적인 의무는 '돌아가는 코드' 가 아니라 '코드 형식을 통한 의사소통' 이다. 다른 개발자와의 의사소통을 위해 형식을 맞추어야 한다.
- **적절한 행 길이를 유지하라**
  - 일반적으로 큰 파일보다 작은 파일이 이해하기 쉽다.
    - 반드시 지켜야할 규칙은 아니지만 **한 파일 당 500줄 미만**으로 구현하라.
- **신문기사처럼 작성하라**
  - 파일의 상단에는 고차원 개념을 배치하고, 아래로 내려갈수록 저차원 수준의 개념과 세부 내용을 묘사하라.
- **개념은 빈 행으로 분리하라.**
  - 새로운 개념을 시작할 때 빈 행을 삽입해 개념을 분리하여 가독성을 높여라

세로 밀집도 & 수직거리(1/3)

개요

- 서로 밀접한 개념의 코드 행은 세로 가까이에 놓아야 한다.
- protected 변수를 피해라

Protected 변수를 피해야하는 이유?

- Protected 변수는 상속한 클래스에서 해당 변수를 사용 가능
  - 상속한 클래스에서 해당 변수를 사용한다면 밀접한 개념이 세로로 가까워야 한다는 원칙 위배

```
class Parent {
    protected int protectedMember;

    public void setProtectedMember(int protectedMember){
      this.protectedMember = protectedMember
    }
}

class Child extends Parent {
  public addProtectedMember(int operand) {
    this.protectedMember += operand
  }
}
```

상속받은 클래스에서 protected 변수를 사용

세로 밀집도 & 수직거리(2/3)

변수선언

- **변수는 사용하는 위치에 최대한 가까이 선언한다**
- **함수 내 지역 변수 : 각 함수의 처음에 선언**
- **루프를 제어하는 변수는 루프 문 내부에 선언 (ex. for(int i=0; i < length ; ++i)**
  - 루프 내부에서 사용하는 변수는 루프의 처음부분에 선언
- **인스턴스 변수는 클래스의 처음에 선언**

논의

- **여기서 말 그대로 '변수' 는 함수의 상단에 배치하고 아래 구문에서 변경가능 하지만, '상수' (javascript 의 const 나 java 의 final) 는 변경불가능 하기 때문에 정확히 '변수' 에 적용해야한다.**

세로 밀집도 & 수직거리(3/3)

개념적 유사성

- **개념적 친화도가 높을수록 코드를 가까이 배치한다.**
- **개념적 친화도**
  - 호출에 의한 종속성이 있으면 개념적 친화도가 높다고 할 수 있다.
  - 비슷한 동작을 하는 함수들은 개념적 친화도가 높다고 할 수 있다.
- **일반적으로 함수 호출 종속성은 아래 방향으로 유지한다.**

```
  public final void discardTo(long timeUs, boolean toKeyframe, boolean stopAtReadPosition)
{   sampleDataQueue.discardDownstreamTo(
        discardSampleMetadataTo(timeUs, toKeyframe, stopAtReadPosition));
  }

  /** Discards up to but not including the read position. */
  public final void discardToRead() {
    sampleDataQueue.discardDownstreamTo(discardSampleMetadataToRead());
  }

  /** Discards all samples in the queue and advances the read position. */
  public final void discardToEnd() {
    sampleDataQueue.discardDownstreamTo(discardSampleMetadataToEnd());
  }
```

[Exoplayer code 일부]
discard라는 비슷한 동작을 함수들
(개념적 친화도가 높은 함수들)을
세로 가까이 배치했다

들여쓰기

개요

- 블록 내부의 구문이 하나이더라도 가급적 들여쓰기를 사용하라
- 내부 구문이 필요 없는 블록에서는 개행 후 세미콜론을 꼭 추가하라

```
if (bypassEnabled) {
    TraceUtil.beginSection("bypassRender");
    while (bypassRender(positionUs, elapsedRealtimeUs)) {}
    TraceUtil.endSection();
} else if (codec != null) {
    long renderStartTimeMs = SystemClock.elapsedRealtime();
    TraceUtil.beginSection("drainAndFeed");
    while (drainOutputBuffer(positionUs, elapsedRealtimeUs)
        && shouldContinueRendering(renderStartTimeMs)) {}
    while (feedInputBuffer() && shouldContinueRendering(renderStartTimeMs)) {}
    TraceUtil.endSection();
}
```

[Exoplayer code 일부]
내부 구문이 없는 while 문을 한 줄에 모두 작성해서 코드 읽기에 위험성이 생길 가능성이 있다.

결론

- 개요에서 말했던 것처럼 개발자의 일차적인 의무는 ‘돌아가는 코드‘ 가 아니라 ‘코드 형식을 통한 의사소통’ 이다. 다른 개발자와의 의사소통을 위해 형식을 맞추어야 한다.
- 형식을 맞추는 일은 개개인의 취향이 반영되는 일이지만, 개인의 취향보다 우선적으로 팀 규칙을 만들고 팀 규칙을 최우선으로 적용하라.
- 요즘은 형식을 맞추기 위한 다양한 라이브러리나 ide의 도구들이 있으니 해당 tool들을 활용하자.