불변 객체(Immutable Object) 및 final을 사용해야 하는 이유

- 조민규 -

불변 객체(Immutable Object)란?

- 객체 생성 이후 내부의 상태가 변하지 않는 객체
- 일반적으로 read-only 메소드만을 제공함
- 내부 상태를 제공하면 방어적 복사(Defensive Copy)를 통해 제공

```
@Test
void temp() {
    String name = "MyName";
    name.toCharArray()[0] = 'm';

    System.out.println(name);
}
```

```
MyName
```

방어적 복사(Defensive Copy)

- 배열이나 객체 등의 참조(Reference)를 전달
- 참조(Reference)를 통해 값을 수정하면 내부의 상태가 변경가능
- 이럴 경우 내부를 복사하여 전달하는 것

```
public char[] toCharArray() {
    // Cannot use Arrays.copyOf because of class initialization order issues
    char result[] = new char[value.length];
    System.arraycopy(value, srcPos: 0, result, destPos: 0, value.length);
    return result;
}
```

Why 불변객체?

1. Thread-Safe하여 병렬 프로그래밍에 유용하며, 동기화를 고려하지 않아도 된다.
2. 실패 원자적인(Failure Atomic) 메소드를 만들 수 있다.
3. Cache나 Map의 또는 Set 등의 요소로 활용하기에 더욱 적합하다.
4. 부수 효과(Side Effect)를 피해 오류가능성을 최소화할 수 있다.
5. 다른 사람이 작성한 함수를 예측가능하며 안전하게 사용할 수 있다.
6. 가비지 컬렉션의 성능을 높일 수 있다.

Thread-Safe하여 병렬 프로그래밍에 유용하며, 동기화를 고려하지 않아도 된다.

- 멀티쓰레드 환경에서 동기화 문제가 발생하는 이유는 “동시 쓰기” 때문
- 불변 객체는 항상 동일한 값을 반환하며, 방어적 복사를 함
- 각각의 쓰레드가 복사된 값을 받으므로 동기화를 고려하지 않아도 됨

실패 원자적인(Failure Atomic) 메소드를 만들 수 있다.

- 가변 객체를 통해 어떠한 작업을 하는 도중 예외가 발생하면 해당 객체가 불안정한 상태에 빠질 수 있음
- 불안정한 객체에 의해 또 다른 에러가 발생할 수 있음
- 불변 객체는 예외가 발생해도 예외 전의 상태를 갖고 있으므로 에러가 발생해도 다음 로직 처리 가능

Cache나 Map의 또는 Set 등의 요소로 활용하기에 더욱 적합하다.

- 값이 변하지 않으므로 캐시나 다른 자료구조에 업데이트 로직이 필요 없음
- 캐시나 다른 자료구조에 적합함

부수 효과(Side Effect)를 피해 오류가능성을 최소화할 수 있다

- 부수 효과란 변수의 값이 변경되거나, 필드 값이 설정되는 등의 변화가 발생하는 효과
- 객체의 내부를 변경하는 메소드가 있다면 객체를 예측하기 어려워짐
- 객체의 내부 변경을 막고, 생성과 사용을 제한함으로써 에러 가능성을 낮추고 유지보수성을 높일 수 있음

다른 사람이 작성한 함수를 예측가능하며 안전하게 사용할 수 있다.

- 불변성이 보장된 함수라면 다른 사람이 개발한 함수를 위험없이 이용할 수 있음
- 다른 사람의 코드를 변경에 대한 불안없이 이용할 수 있음

가비지 컬렉션의 성능을 높일 수 있다

- 불변 객체를 갖는 컨테이너 객체(ImmutableHolder)도 존재함
- 당연히 불변의 객체(Object value)가 먼저 생성되어야 컨테이너 객체가 이를 참조 가능
- 즉, 컨테이너는 컨테이너가 참조하는 가장 젊은 객체들보다 더 젊다는 것(늦게 생성되었다는 것)
  1. Object 타입의 value 객체 생성
  2. ImmutableHolder 타입의 컨테이너 객체 생성
  3. ImmutableHolder가 value 객체를 참조

가비지 컬렉션의 성능을 높일 수 있다

1. Object 타입의 value 객체 생성
2. ImmutableHolder 타입의 컨테이너 객체 생성
3. ImmutableHolder가 value 객체를 참조

```
class MutableHolder {
    private Object value;
    public Object getValue() { return value; }
    public void setValue(Object o) { value = o; }
}

class ImmutableHolder {
    private final Object value;
    public ImmutableHolder(Object o) { value = o; }
    public Object getValue() { return value; }
}
```

```
@Test
public void createHolder() {
    // 1. Object 타입의 value 객체 생성
    final String value = "MyName";

    // 2. Immutable 생성 및 값 참조
    final ImmutableHolder holder = new ImmutableHolder(value);

}
```