Always override toString

개요

toString은 일반적으로 원하는 결과를 반환하지 않습니다. PhoneNumber 클래스가 있다고 하면 PhoneNumber@hashCode 형태로 반환되기 때문이지요.

toString을 하면 707-867-5309가 훨씬 읽기 좋고 유용합니다. 그래서 toString contract를 보면 항상 override하라고 합니다.

장점

toString을 구현하면 사용하기 즐겁고 디버깅이 쉬워집니다.

- println, printf, +, assert, 디버거가 객체를 출력할 때 toString이 자동으로 불려지기 때문.
- 따라서 다음 코드만으로 문제를 진단하기 쉬워짐

  `System.out.println(phoneNumber + "에 연결할 수 없습니다.")`

유의 사항

1. 핵심 정보 반환

실전에서 사용될 때 to String은 핵심정보를 모두 반환하는게 좋습니다.

안그러면 다음과 같은 상황이 발생할 수 있죠.

```
Assertion failure: expecrted {abc, 123}, but was {abc, 123}.
```

즉, 객체를 asertion 했을 때 핵심 정보가 달라 assertion이 실패한 상황했습니다.
그런데 toString이 핵심정보를 출력하지 못해서 동일해보이는 현상입니다.

2. 포맷 문서화

장점

- value calss라면 추후 csv file로도 변환하기 쉬움

포맷을 정하기로 했다면 포맷에 맞는 문자열을 객체로 변환시켜주는 static factory나 constructor도 제공하면 좋다.
(자바 플랫폼의 value class들이 따르는 방식)

단점

- 포맷에 종속되기 때문에 추후 변경이 어려움

만약에 포맷을 명시하지 않을 경우엔 변경될 수 있다고 작성해줄 것.

의도 명시

그리고 포맷을 작성하든 안하든, 의도를 명시해주면 좋다.

3. toString 값을 반환할 수 있는 API 제공

toString이 반환하는 값에 포함된 정보를 얻어올 수 있는 API를 제공하면 좋다.

만약 안그렇다면 toString에서 얻은 정보를 파싱해야 하는데 이는 성능저하를 일으키고, toString의 포맷 변경시 시스템이 망가질 수 있기 때문.

(원서에는 provide programmatic access to the information이라고 표현)

4. static class의 toString

toString제공할 필요 없는 경우

- static utility class는 toString을 제공할 이유가 없음
- Enum Type의 toString은 JAVA가 완벽히 제공 중

필요한 경우

- sub class들이 공유해야할 문자열 표현이 있는 추상 클래스라면 toString 재정의 필요.

  대다수 collection 구현체들은 추상 컬렉션 클래스들의 toString 메서드를 상속해 사용.

5. 자동생성

구글의 AutoValue 프래임워크는 각 필드를 잘 나타내주나, 클래스의 의미는 파악하긴 어렵습니다.

예를 들어 PhoneNumber class는 전화번호 표준 체계에 대해 명시해줘야 하므로 문제가 될 수 있습니다. (끝번호가 123 일경우엔 0123으로 표기 4자리로 표현해야 하므로.)