Consistently use the Override annotation

java library는 유용한 어노테이션을 제공합니다. `@Override`가 중요한 애너테이션 중 하나이지요. 이 애너테이션은 method에서만 선언할 수 있으며, superType의 method를 오버라이드했다고 나타내주는데 사용할 수 있습니다.

이 애너테이션을 일관성있게 사용한다면 여러 악명 높은 버그들을 예방해줍니다.

다음 예시를 통해서 알아보겠습니다.

Bigram

Bigram이라고 영어 알파벳 2개로 구성된 문자열을 표현해주는 클래스를 예로 들어보겠습니다.

```
// Can you spot the bug?
public class Bigram {
  private final char first;
  private final char second;
  public Bigram(char first, char second) {
    this.first  = first;
    this.second = second;
  }
  public boolean equals(Bigram b) {
    return b.first == first && b.second == second;
  }
  public int hashCode() {
    return 31 * first + second;
  }
  public static void main(String[] args) {
    Set<Bigram> s = new HashSet<>();
    for (int i = 0; i < 10; i++)
      for (char ch = 'a'; ch <= 'z'; ch++)
        s.add(new Bigram(ch, ch));
    System.out.println(s.size());
  }
}
```

main method에서 똑같은 소문자 2개로 구성된 바이그램 2개를 10번 반복해서 Set에 집어넣습니다. 그리고 그 집합의 크기를 출력하고 있습니다.

문제

무엇이 출력될까?

딱 봤을 때 문제가 보이시나요? 예상대로라면 Set은 중복된 값을 허용하지 않기 때문에 26이 출력되어야 하나, 위 코드는 260을 출력됩니다.

바로 오버라이딩 문제

바로 equals에서 오버라이딩을 제대로 하지 못했기 때문입니다. Object class의 method의 parameter는 Object 타입입니다. 따라서 equals 2개를 선언할 꼴이 되지요.

개선

`@Override`를 equals method에 달면 문제가 해결됩니다!

컴파일 타임에 실제로 오버라이딩했는지 체크해줍니다. 만약에 오버라이딩하지 않았다면 에러를 던지죠.

추가로,,

대부분 IDE에선 이를 자동으로 지원해줍니다. 추가적으로, abstraction class에서 abstract method는 컴파일러가 자동으로 알려줘서 달아줄 필요는 없지만, 일관성을 위해서라면 달아줘도 됩니다.

인터페이스 메서드를 재정의할 때도 쓰시면 좋습니다. 디폴드 메서드 때문에 실제로 구현했는지 확인하는데 좋습니다.

물론, 디폴트 메서드가 없다면 가독성을 위해 애너테이션을 달아주지 않아도 됩니다.