다중정의는 신중히 사용하라

최 혁

다중정의의 문제점

```
public class CollectionClassifier {
    public static String classify(Set<?> s) { return "집합"; }
    public static String classify(List<?> lst) { return "리스트"; }
    public static String classify(Collection<?> c) { return "그 외"; }
    public static void main(String[] args) {
        Collections<?>[] collections = {
            new HashSet<String>,
            new ArrayList<BigInteger>(),
            new HashMap<String, String>().values()
        };
        for (Collection<?> c : collections)
            System.out.println(classify(c));
    }
}
```

오버로딩된 메서드들 중 어느 메서드가 호출될지는 컴파일타임에 정해진다!

이처럼 직관과 어긋나는 이유는 재정의한 메서드는 동적으로 선택되고, 다중정의한 메서드는 정적으로 선택되기 때문이다.

(위 문제를 해결하려면 모든 classify 메서드를 합친 후 instanceof로 명시적으로 검사하면 해결된다.)

```
public static String classify(Collection<?> c) {
    return c instanceof Set ? "집합" :
           c instanceof List ? "리스트" : "그 외";
}
```

매개변수 수가 같은 다중정의는 만들지 말자!

API 사용자가 매개변수를 넘기면 어떤 다중정의 메서드가 호출될지를 모른다면 프로그램이 오동작하기 쉽다.

그러므로 안전하고 보수적으로 가려면 매개변수 수가 같은 다중정의는 만들지 말자!

다중정의하는 대신 메서드 이름을 다르게 지워주는 길도 있다!
(ObjectOutputStream 클래스의 write 메서드들: writeInt(int), writeLong(long) ...)

생성자 다중정의

생성자의 경우 이름을 다르게 지을 수 없으니 두 번째 생성자부터는 무조건 다중정의가 된다.

안전 대책

- 매개변수 수가 같은 다중정의 메서드가 많더라도, 매개변수들이 명확히 구분된다면 햇갈릴 일이 없을 것이다.
  (예를 들어 ArrayList의 int, Collection 생성자)

만약 다중정의한다면.. 1

```
public interface List<E> extends Collection<E> {
    boolean remove(Object o);
    E remove(int index);
}

int a = 1;
list.remove((Integer) a); //list 안에 값이 1인 원소 삭제
list.remove(a); //index가 1인 원소 삭제
```

만약 다중정의한다면.. 2

```
//Thread(Runable runable)
new Thread(System.out::println).start();

//<T> Future<T> submit(Callable<T> task);
//Future<?> submit(Runnable task);
ExecuterService exec = Executers.newCachedThreadPool();
exec.submit(System.out::println);
```

Thread는 컴파일 성공하지만, ExecuterService는 실패한다.

그 이유에 대해 이해하기 어려우니 그냥 메서드를 다중정의할 때, 서로 다른 함수형 인터페이스라도 같은 위치의 인수를 받아서는 안 된다고 외우자.

결론

- 일반적으로 매개변수 수가 같을 때는 다중정의를 피하는 게 좋다.
- 만약 불가능하다면, 형변환하여 정확한 다중정의 메서드가 선택되도록 하자.
- 형변환도 불가능하다면, 다중정의 메서드들이 모두 동일하게 동작하도록 만들자. (예를 들어 System.out.println)