아이템28: 배열보다는 리스트를 사용하라

목차

배열과 제네릭 타입의 차이

1. 배열은 공변(함께 변한다)이지만 제네릭은 불공변이다.
    - Sub가 Super의 하위 타입이라면 배열 Sub[]는 배열 Super[]의 하위 타입이다.
    - TypeA와 TypeB가 있을 때, List<TypeA>와 List<TypeB>는 서로 상위 타입도 하위타입도 아니다.

```
//컴파일 됨
Object[] objects = new Long[1];
objects[0] = "문자열"; // throw ArrayStoreException

//컴파일 안됨
List<Object> objectList = new ArrayList<Long>();
objectList.add("문자열")
```

    - 두 경우 모두 문제가 있지만 배열의 경우 문법상 문제가 없어 오류를 컴파일 시점이 아닌 런타임 시점에야 알 수 있다.

2. 배열은 실체화(reify)된다.
    - 배열은 런타임 시점에도 자신이 담기로 한 원소의 타입을 인지하고 확인한다.
    - 제네릭은 원소 타입을 컴파일 시점에만 검사하고 런타임 시점에는 소거하여 알 수가 없다. 이러한 소거는 제네릭이 지원되기 전 레거시 코드와 제네릭 타입을 함께 사용할 수 있게 해주는 메커니즘이다.

제네릭 배열을 만들지 못하게 막은 이유

- 배열과 제네릭 타입의 차이 때문에 배열과 제네릭은 잘 어우러지지 못한다.
- 즉, 배열은 제네릭 타입, 매개변수화 타입, 타입 매개변수로 사용할 수 없다.
    - `new List<E>[]`, `new List<String>[]`, `new E[]` → 컴파일 에러
- 제네릭 배열을 만들지 못하게 막는 이유는 타입 안전하기 않기 때문이다. 만약 이를 허용한다면 컴파일러가 자동 생성하는 형변환 코드에서 런타임에 ClassCastException이 발생할 수 있다. 이는 제네릭 타입 시스템의 취지에 어긋난다.

```
List<String>[] stringLists = new List<String>[1]; //(1)
List<Integer> intList = List.of(42);              //(2)
Object[] objects = stringLists;                   //(3)
objects[0] = intLists;                            //(4)
 String s = stringLists[0].get(0);                //(5)
```

- **(4)**: (2)에서 생성한 `List<Integer>`의 인스턴스를 `Object` 배열의 원소로 저장한다. 제네릭이 소거 방식으로 구현되어 문제가 없다. 즉, 런타임 시에 `List<Integer>` 인스턴스의 타입은 List가 되고 `List<Integer>[]` 인스턴스 타입은 `List[]`가 된다. 따라서 ArrayStoreException이 발생하지 않는다.
- **(5)**: `List<String>` 만 담겠다고 선언한 stringLists에 `List<Intger>`가 담겨있다. (5)에서 원소를 꺼낼 때, 컴파일러는 꺼낸 원소를 자동으로 `String` 형변환 하는데 이 원소는 이므로 런타임에 ClassCastException이 발생한다. 이를 막기 위해 (1)에서 컴파일 오류를 내야 한다

배열을 제네릭으로 만들 수 없어 생기는 문제

- 제네릭 컬렉션에서는 자신의 원소 타입을 담은 배열을 반환하는 게 보통은 불가능하다.
- 제네릭 타입과 가변인수 메서드를 함께 쓰면 해석하기 어려운 경고 메시지를 받게된다. 가변인수 메서드를 호출할 때마다 가변인수 매개변수를 담을 배열이 생성되는데, 이때 그 배열의 원소가 실체화 불가 타입이라면 경고가 발생한다. → `@SafeVarargs` 어노테이션으로 경고 제거 가능(주의 필요)

💡 가변인수(Variable Argument) 매개변수로 들어오는 값의 개수와 상관 없이 동적으로 인수 받도록 해주는 문법 - 대표적인 가변인수 메서드 `System.out.printf()`

- 배열로 형변환할 때 제네릭 배열 생성 오류나 비검사 형변환 경고가 뜨는 경우 대부분은 컬렉션인 `List<E>` 를 사용하면 해결된다. 이를 통해 타입 안정성을 확보할 수 있다.

정리

- 배열은 공변이고 실체화 되는 반면, 제네릭은 불공변이고 타입 정보가 소거된다.
- 배열은 런타임에는 타입 안전하지만 컴파일타임에는 그렇지 않다. 제네릭은 반대다.
- 둘을 섞어 쓰다가 컴파일 오류 혹은 경고를 만난다면 배열을 리스트로 바꾸는 것부터 시도하자.

결국 배열보다 리스트를 사용해야 하는 이유는 컴파일 시점에 미리 오류를 알 수 있다는 점인것 같다.