

Effective Java

Item 28. 배열보다는 리스트를 사용하라

개요

1. 배열과 제네릭 타입의 차이
2. 배열 대신 리스트 사용하기
3. Summary

개요

1. 배열과 제네릭 타입의 차이
2. 배열 대신 리스트 사용하기
3. Summary

배열과 제네릭 타입의 차이

배열은 **공변**이지만, 제네릭은 **불공변**이다.

- 공변 : Sub과 Super의 하위 타입이라면, Sub[]는 Super[]의 하위 타입이다.
- 불공변 : 서로 다른 타입 Type1, Type2가 있을 때, List<Type1>은 List<Type2>의

하위 타입도 아니고 상위 타입도 아니다.

제네릭은 타입 매개변수가 다르면 완전히 다른 타입으로 취급된다.

배열과 제네릭 타입의 차이

■ 공변 : Sub과 Super의 하위 타입이라면, Sub[]는 Super[]의 하위 타입이다.

■ 불공변 : 서로 다른 타입 Type1, Type2가 있을 때, List<Type1>은 List<Type2>의 하위 타입도 아니고 상위 타입도 아니다.

```
public static void main(String[] args){
    String[] stringArray = new String[3];
    Object[] objectArray = stringArray;

    objectArray[0] = "문자열";
    objectArray[1] = Integer.valueOf(42); // ArrayStoreException
}
```

✓ 공변성으로 인해 런타임에서야 타입 오류가 발생한다.

```
public static void main(String[] args) {
    List<String> stringList = new ArrayList<>();
    List<Object> objectList = stringList; // 컴파일 오류

    stringList.add("밍트");
    stringList.add("자바");
    // stringList.add(Integer.valueOf(11)); // 컴파일 오류
}
```

✓ 불공변성 덕분에 타입 안전성이 보장된다.

배열과 제네릭 타입의 차이

배열은 **실체화**되지만, 제네릭은 실체화되지 않는다.

자신이 어떤 타입의 객체를 담는지 알고 있다

■ 배열: 런타임에도 자신의 타입 정보를 유지한다.

■ 제네릭 : 런타임에 타입 정보가 소거된다.

```
public static void main(String[] args){
    Object[] objectArray = new String[2];

    System.out.println(objectArray.getClass().getComponentType());
    // class java.lang.String
}
```

```
public static void main(String[] args){
    List<String> stringList = new ArrayList<>();

    System.out.println(stringList.getClass().getComponentType());
    // class java.util.ArrayList
}
```

참고) 실체화 가능 타입

비한정적 와일드카드 타입은 실체화 가능하다.

모든 타입을 나타내는 ?가 실제로 Object 타입으로 대체된다.

```
public static void main(String[] args) {  
    // List<String>[] stringListArray = new List<String>[10]; // 컴파일 에러  
    List<?>[] wildcardListArray = new List<?>[10]; // 가능  
}
```

참고) 런타임에 타입 정보가 소거되는 이유

기존 코드와 새로운 제네릭 코드가 함께 동작하기 위함이다.

- 기존 코드 : 타입 지정 없이 캐스팅하여 사용 (raw type)
- 기존 코드(raw type)과 제네릭 혼용

```
public static void main(String[] args) {  
    List myList = new ArrayList();  
    myList.add("밍트");  
    myList.add(Integer.valueOf(3));  
    myList.add(new LocalDateTime());  
  
    String s = (String) myList.get(0);  
}
```

```
public static void main(String[] args) {  
    List<String> stringList = new ArrayList<>();  
    stringList.add("밍트");  
  
    // 레거시 메서드 (List 타입 파라미터에 대입 가능)  
    legacyMethod(stringList);  
}  
  
// 레거시 코드  
void legacyMethod(List list) {  
    for (Object obj : list) {  
        String str = (String) obj;  
        System.out.println(str);  
    }  
}
```

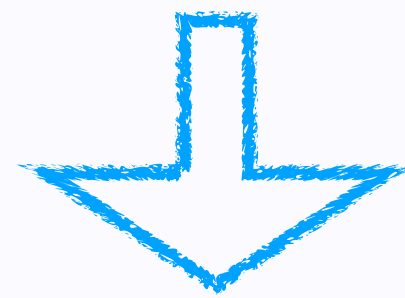

개요

1. 배열과 제네릭 타입
- 2. 배열 대신 리스트 사용하기**
3. Summary

제네릭 배열 생성을 막는 이유

런타임에 ClassCastException이 발생하여 타입 안전하지 않다.

```
public static void main(String[] args){  
    List<String>[] stringLists = new List<String>[1]; // 실제로는 컴파일 에러  
    Object[] objects = stringLists; // 배열은 공변이므로 가능  
  
    List<Integer> intList = List.of(42);  
    objects[0] = intList; // List[] 배열에 List 객체 추가  
    String s = stringLists[0].get(0); // ClassCastException  
}
```



처음부터 제네릭 배열 생성을 컴파일 에러로 차단

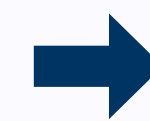
배열 대신 리스트를 사용하라


비검사 형변환 경고가 뜰 수 있다.

컴파일러가 런타임에 타입 안전성 체크 불가

```
public class Stack<E> { new *  
  
    private E[] elements;  
  
    // @SuppressWarnings("unchecked")  
    public Stack(int capacity) { new *  
        | elements = (E[]) new Object[capacity];  
    }  
}
```

You, Moments ago • Uncommitted changes



```
public class Stack<E> { new *  
  
    private List<E> elements;  
  
    public Stack(int capacity) { new *  
        |  elements = new ArrayList<>(capacity);  
    }  
  
    public void push(E item) { new *  
        | elements.add(item);  
    }  
}
```

✅ 컴파일 시간에 타입 검사를 수행하여 타입 안전성이 보장된다.

개요

1. 배열과 제네릭 타입의 차이
2. 배열 대신 리스트 사용하기
3. Summary

Summary

1. 배열과 제네릭 타입의 차이

- 배열은 공변이지만, 제네릭은 불공변이다.
- 배열은 실체화되지만, 제네릭은 실체화되지 않는다.

2. 배열 대신 리스트 사용하기

- 제네릭 배열을 생성하는 것을 막아 타입 안전성을 보장한다.

꺾