Generics

일반화된 객체의 타입을 표현하기위해 사용된다. 입력으로 정해진 객체 혹은, 그의 부모, 자식등으로 다양한 객체 타입을 하나의 코드로 사용할수 있는 편리한 기능이다.

Generic 타입의 매개변수

- E: Element, 컬렉션의 요소를 표현할때 사용
- T: Class Type
- V. Value
- K: Key

Why Use?

Generics는 타입(Class, Interface)등을 정의하며 타입을 파라미터로 쓸수 있게 하는것.

타입파라미터는 코드 재사용성 ↑

메소드의 파라미터 = Value / 타입파라미터 = Type.

WildCard

Generic으로 구현된 메소드의 경우에는 선언된 타입으로만 매개변수는 입력해야 한다.

이를 상속받은 클래스, 혹은 부모 클래스를 매개변수로 사용하고 싶어도 불가능하며, 혹은 그 어떤 타입이다로 상관 없는 경우에 대응하기 좋지 않다. 이를 위해 Wild Card 사용.

① Unbound WildCard

List<?> 와 같은 형태로 물음표만 가지고 정의 되어진다. 내부적으로는 Object로 정의 되어서 사용되고 모든 타입을 인자로 받을 수 있다. 타입 파라미터에 의존하지 않는 메소드만을 사용하거나 Object 메소드에서 제공하는 기능으로 충분한 경우에 사용

② Upper Bounded WildCard (Get은 OK / Set은 No)

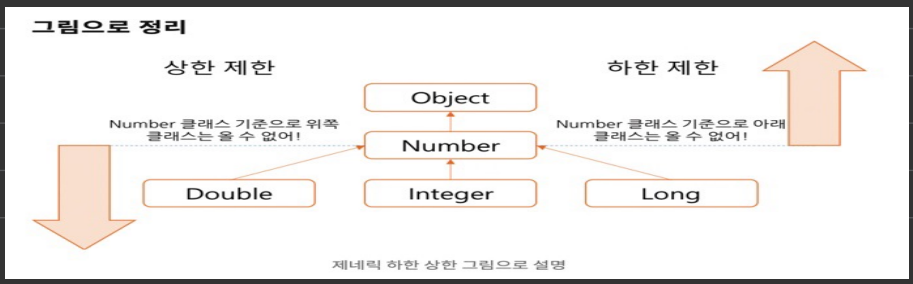
List<? extends Foo> 형태로 사용. 특정 클래스의 자식 클래스만을 인자로 받겠다는 선언.

주로 변수의 제한을 완화 하기 위해서 사용

③ Lower Bounded WildCard (Set은 OK / Get은 No!)

List<? super Foo> 형태로 사용. Upper와 반대로 특정 클래스의 부모클래스만 인자로 받겠다는 선언.

사용 할때는 Object로 취급된다.



Generics Vs WildCard

<T> : 1. 원소를 꺼내보면 Object에 정의되어 있는 기능만 사용하겠다 . equals() . toString() . hashCode() ..

2 List 타입에 뭐가 오든 상관 없다. 인터페이스에 정의되어 있는 기능과 타입파라미터와 관련된 기능을 사용하겠다.
add(), addAll()

<?> : 1. 위와 동일
size(), clear()

2. List 타입이 뭐가 오든 상관 없다. List 인터페이스에 정의 되어있는 기능만 사용하겠다.

Upper-bounded Vs lower-bounded

1). Upper-bounded

Get OK / Set NO

- 컴파일러는 상속구조를 전혀 기억 X.
- 부모클래스 호출시 자식클래스 메소드 호출 X.

2.) lower-bounded

Set OK / Get NO.