제네릭의 사용이유

: 잘못된 타입이 사용될 수 있는 문제를 컴파일 과정에서 제거할 수 있게 되었다.

* 컴파일 시 강한 타입 체크를 할 수 있다.
* 타입 변환(casting)을 제거한다.
* Object 클래스란?

: 모든 자바 클래스의 최상위 조상 클래스이므로 자식객체는 부모 타입에 대입할 수 있다.라는 성질 때문에 모든 자바 객체는 Object 타입으로 자동 타입 변환 되어 저장된다.

제네릭 타입

: 타입을 파라미터로 가지는 클래스와 인터페이스를 말한다.

제네릭 타입은 클래스 또는 인터페이스 이름 뒤에 “<>” 부호가 붙고, 사이에 타입 파라미터가 위치한다.

<T> 타입으로의 초기화는 객체 생성 당시의 타입을 받아 자동적으로 들어가는 타입으로 <T, V> 등과 같이 여러 개의 타입을 받는 것도 가능하다. 이를 통해 형변환을 최소화 시킬 수 있으며, 그와 동시에 안정성을 확보할 수 있다.

멀티 타입 파라미터( class<K, V, …>, interface<K, V, …>)

: 제네릭 타입은 두 개 이상의 멀티 타입 파라미터를 사용할 수 있다. 이 경우 각 타입 파라미터를 콤마로 구분한다.

\*\* 자바 7부터 제네릭 타입 파라미터의 중복 기술을 줄이기 위해 다이몬드 연산자<>를 제공한다.

Ex) Product<Tv, String> product = new Product<>();

제네릭 메소드(<T, R> R method(T t))

Ex) public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V>p2){}

ㄴ> 첫번째 <K, V> 는 매개변수 및 메소드 내에서의 제네릭이고, boolean은 리턴타입이다.

제한된 타입 파라미터(<T extends 최상위타입>)

Ex) public <T extends Number> int compare(T t1, T t2){

Double v1 = t1.doubleValue();

Double v2 = t2.doubleValue();

Return Double.compare(v1, v2);

\*\* Double.compare() 메소드는 첫번째 매개값이 작으면 -1을, 같으면 0을, 크면 1을 리턴한다.

와일드카드 타입(<?>, <? Extends …>, <? Super …>)

? : 타입 파라미터를 대치하는 구체적인 타입으로 모든 클래스나 인터페이스 타입이 올 수 있다.

? extends 상위타입 : 타입 파라미터를 대치하는 구체적인 타입으로 상위타입이나 하위 타입만 올 수 있다.

? super 하위타입 : 타입 파라미터를 대치하는 구체적인 타입으로 하위 타입이나 상위 타입이 올 수 있다.

// extends는 상위타입을 적고 상위 타입에 해당하는 하위 타입까지 포섭이 가능하다.

// super는 하위타입을 적고 하위 타입에 해당하는 상위 타입까지 포섭이 가능하다.

\*\* 타입 파라미터 배열 \*\*

: 타입 파라미터로 배열을 생성하려면 new T[n] 형태로 생성할 수 없고, (T[])(new Object[capacity]);

제네릭 타입의 상속과 구현

: 제네릭도 다른 타입과 마찬가지로 부모 클래스가 될 수 있다.

Ex) public class ChildProduct<T, M, C> extends Product<T, M>{};