



Generic

혼자 공부하는 자바 (신용권 저)

제네릭 타입

❖ 필요성

- 자바는 다양한 종류의 객체를 관리하는 컬렉션이라는 자료구조를 제공
- 초기에는 Object 타입의 컬렉션을 사용
- Object 타입의 컬렉션은 실행하기 전에는 어떤 객체인지?



❖ 소개

■ 제네릭 타입의 의미

하나의 코드를 다양한 타입의 객체에 재사용하는 객체 지향 기법
클래스, 인터페이스, 메서드를 정의할 때 타입을 변수로 사용



■ 제네릭 타입의 장점

컴파일할 때 타입을 점검하기 때문에 실행 도중 발생할 오류 사전 방지
불필요한 타입 변환이 없어 프로그램 성능 향상

❖ 제네릭 타입 선언

```
class 클래스이름<타입매개변수> {
```

```
    필드;
```

```
    메서드;
```

메서드나 필드에 필요한 타입을 타입 매개변수로 나타낸다.

```
}
```

- 타입 매개변수는 객체를 생성할 때 구체적인 타입으로 대체

- 전형적인 타입 매개변수

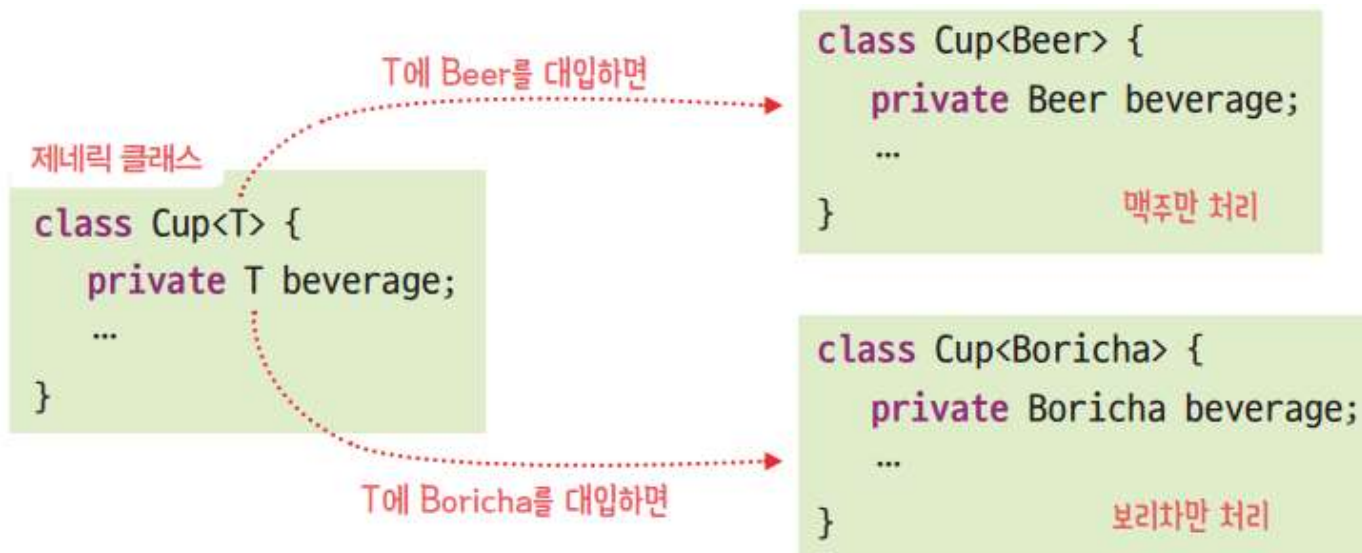
타입 매개변수	설명
E	원소(Element)
K	키(Key)
N	숫자(Number)
T	타입(Type)
V	값(Value)

❖ 제네릭 객체 생성

제네릭클래스 <적용할타입> 변수 = new 제네릭클래스<적용할타입>();

생략할 수 있다.

- <적용할타입>에서 적용할 타입을 생략할 경우 <>를 다이아몬드 연산자라고 함
- 제네릭 클래스의 적용



```

3 public class Beverage {
4 }
5

```

```

3 public class Boricha extends Beverage {
4 }
5

```

```

3 public class Beer extends Beverage {
4 }
5

```

```

3 public class Cup<T> {
4     private T beverage;
5
6     public T getBeverage() {
7         return beverage;
8     }
9
10    public void setBeverage(T beverage) {
11        this.beverage = beverage;
12    }
13 }
14
15 public class GenericClass3Demo {
16     public static void main(String[] args) {
17         Cup c = new Cup();
18
19         c.setBeverage(new Beer());
20
21         // Beer beer = c.getBeverage();
22         Beer beer = (Beer) c.getBeverage();
23     }
24 }

```

❖ 제네릭 타입 응용

❖ Raw 타입의 필요성 및 의미

- 이전 버전과 호환성을 유지하려고 Raw 타입을 지원
- 제네릭 클래스를 Raw 타입으로 사용하면 타입 매개변수를 쓰지 않기 때문에 Object 타입이 적용

❖ 제네릭 타입의 상속 관계

- 예를 들어

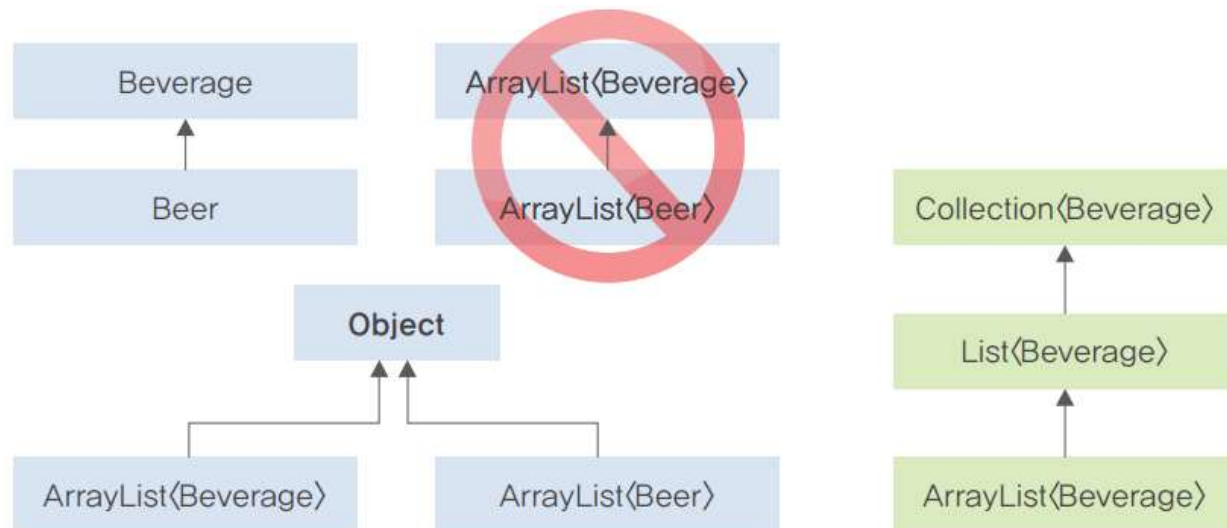
```
ArrayList<Beverage> list = new ArrayList<>();  
list.add(new Beer());           // OK  
list.add(new Boricha());        // OK
```

- 그러나 ArrayList<Beverage> 타입과 ArrayList<Beer>의 경우는 상속 관계가 없다.

제네릭 상속 및 타입 한정

❖ 제네릭의 제약

- 기초 타입을 제네릭 인수로 사용 불가
- 정적 제네릭 타입 금지
- 제네릭 타입의 인스턴스화 금지. 즉, `new T()` 등 금지
- 제네릭 타입의 배열 생성 금지
- 실행 중에 제네릭 타입 점검 금지. 예를 들어, `a instanceof ArrayList<String>`
- 제네릭 클래스의 객체는 예외로 던지거나 잡을 수 없다
- 제네릭의 서브 타입 허용 않음



제네릭 상속 및 타입 한정

❖ 타입 한정

```
<T extends 특정클래스> 반환타입 메서드이름(...) { ... }  
<T extends 인터페이스> 반환타입 메서드이름(...) { ... }
```

부모가 인터페이스라도 extends를 사용한다.

```

1 package generic_package;
2
3 public class CupCup<T extends Beverage> {
4     private T beverage;
5     public T getBeverage() {
6         return beverage;
7     }
8
9     public void setBeverage(T beverage) {
10        this.beverage = beverage;
11    }
12 }
13

```

```

1 package generic_package;
2
3 public class Cup<T> {
4     private T beverage;
5     public T getBeverage() {
6         return beverage;
7     }
8
9     public void setBeverage(T beverage) {
10        this.beverage = beverage;
11    }
12 }
13

```

```

1 package generic_package;
2
3 public class BoundedTypeDemo {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         Cup<String> strCup = new Cup<>();
8         //CupCup<String> strCup1 = new CupCup<>();
9
10        Cup<Beer> beer = new Cup<>();
11        Cup<Boricha> cup2 = new Cup<>();
12        Cup<Beverage> cup3 = new Cup<>();
13
14    }
15
16 }
17

```

제네릭 메서드

❖ 의미와 선언 방법

- 타입 매개변수를 사용하는 메서드
- 제네릭 클래스뿐만 아니라 일반 클래스의 멤버도 될 수 있음
- 제네릭 메서드를 정의할 때는 타입 매개변수를 반환 타입 앞에 위치

```
< 타입매개변수 > 반환타입 메서드이름(...) {  
    ...  
}
```

2개 이상의 타입 매개변수도 가능하다.

- 제네릭 메서드를 호출할 때는 구체적인 타입 생략 가능
- JDK 7과 JDK 8의 경우 익명 내부 클래스에서는 다이아몬드 연산자 사용 불허
- JDK 9부터는 익명 내부 클래스에서도 다이아몬드 연산자 사용 가능

제네릭 메서드

❖ 예제

- 배열의 타입에 상관없이 모든 원소 출력

```
3 public class GenMethod1Demo {  
4     static class Utils {  
5         public static <T> void showArray(T[] a) {  
6             for (T t : a)  
7                 System.out.printf("%s ", t);  
8             System.out.println();  
9         }  
10  
11         public static <T> T getLast(T[] a) {  
12             return a[a.length - 1];  
13         }  
14     }  
15  
16     public static void main(String[] args) {  
17         Integer[] ia = { 1, 2, 3, 4, 5 };  
18         Character[] ca = { 'H', 'E', 'L', 'L', 'O' };  
19  
20         Utils.showArray(ia);  
21         Utils.<Character>showArray(ca);  
22  
23         System.out.println(Utils.getLast(ia));  
24     }  
25 }
```

제네릭 메서드

❖ 제네릭 타입에 대한 범위 제한

■ 사용 방법

```
<T extends 특정클래스> 반환타입 메서드이름(...) { ... }
```

```
<T extends 인터페이스> 반환타입 메서드이름(...) { ... }
```

부모가 인터페이스라도 extends를 사용한다.

■ 예제


```
3 public class GenMethod2Demo {
4     static class Utils {
5         public static <T extends Number> void showArray(T[] a) {
6             for (T t : a)
7                 System.out.printf("%s ", t);
8                 System.out.println();
9         }
10    }
11
12    public static void main(String[] args) {
13        Integer[] ia = { 1, 2, 3, 4, 5 };
14        Double[] da = { 1.0, 2.0, 3.0, 4.0, 5.0 };
15        Character[] ca = { 'H', 'E', 'L', 'L', 'O' };
16
17        Utils.showArray(ia);
18        Utils.showArray(da);
19        // Utils.<Character>showArray(ca);
20    }
21 }
```



```

3  class Ticket implements Comparable {
4      int no;
5
6      public Ticket(int no) {
7          this.no = no;
8      }
9
10     public int compareTo(Object o) {
11         Ticket t = (Ticket) o;
12         return no < t.no ? -1 : (no > t.no ? 1 : 0);
13     }
14 }
15
16 public class GenMethod3Demo {
17     public static <T extends Comparable> int countGT(T[] a, T elem) {
18         int count = 0;
19         for (T e : a)
20             if (e.compareTo(elem) > 0)
21                 ++count;
22         return count;
23     }
24
25     public static void main(String[] args) {
26         Ticket[] a = { new Ticket(5), new Ticket(3), new Ticket(10), new Ticket(7), new Ticket(4) };
27
28         System.out.println(countGT(a, a[4]));
29     }
30 }

```



Thank You!