13장. 제네릭

이것이 자바다(http://cafe.naver.com/thisjava)

한빛미디어 Hanbit Media, Inc.

Contents

1절. 왜 제네릭을 사용해야 하는가?

❖ 제네릭(Generic) 타입이란?

- ‘컴파일 단계’ 에서 ‘잘못된 타입 사용될 수 있는 문제’ 제거 가능

- 자바5부터 새로 추가 !

- 컬렉션, 람다식(함수적 인터페이스), 스트림, NIO에서 널리 사용

Class ArrayList<E>

```
default BiConsumer<T,U> andThen(BiConsumer<? super T,? super U> after)
```

1절. 왜 제네릭을 사용해야 하는가?

❖ 제네릭을 사용하는 코드의 이점

- 컴파일 시 강한 타입 체크 가능
  - 실행 시 타입 에러가 나는 것 방지
  - 컴파일 시에 미리 타입을 강하게 체크해서 에러 사전 방지

- 타입변환 제거 가능

```
List list = new ArrayList();
list.add("hello");
String str = (String) list.get(0);
```

```
List<String> list = new ArrayList<String>();
list.add("hello");
String str = list.get(0);
```

2절. 제네릭 타입

❖ 제네릭

- 특정 타입만 다루지 않고, 여러 종류의 타입으로 변신할 수 있도록 클래스나 메소드를 일반화시키는 기법
- 컬렉션과 같은 컨테이너 클래스에 유연성을 해치지 않으며 다수 유형 가능
- 코드절약 및 코드 재사용성을 증진시켜 유지보수 용이
- 컴파일시 타입오류를 체크하여, 사전에 엄격한 데이터타입 체크를 가능케한다.
  - Object를 사용한 구현의 편리함에는 자료의 안정성의 관점에서는 문제가 있음
- 프로그램 성능저하를 유발하는 캐스팅(강제 데이터타입 변환)을 제거

```
List list = new ArrayList();
list.add("hello");
String str = (String) list.get(0);
```

```
List<String> list = new ArrayList<String>();
list.add("hello");
String str = list.get(0);
```

예제

```
public class Apple {

}
public class Box {
    private Object object;
    public void set(Object object) {
        this.object = object;
    }
    public Object get() {
        return object;
    }
}
public class BoxExample {

    public static void main(String[] args) {
        Box box = new Box();
        box.set("홍길동");
        String name = (String) box.get();

        box.set(new Apple());
        Apple apple = (Apple) box.get();
    }

}
```

2절. 제네릭 타입

❖ 제네릭 타입이란?

- 타입을 파라미터로 가지는 클래스와 인터페이스
- 선언 시 클래스 또는 인터페이스 이름 뒤에 “<>” 부호 붙임
- “<>” 사이에는 타입 파라미터 위치
- 타입 매개변수가 나타내는 타입의 객체 생성 불가
  - ex) ~~T a = new T();~~
- 타입 매개 변수는 나중에 실제 타입으로 구체화
- 일반적으로 대문자 알파벳 한 문자로 표현

```
public class 클래스명<T>{…}
public interface 인터페이스명<T>{…}
```

```
public class MyClass<T> {..}
```

2절. 제네릭 타입

❖ 제네릭 타입 사용 여부에 따른 비교

- 제네릭 타입을 사용하지 않은 경우
  - Object 타입 사용 → 빈번한 타입 변환 발생 → 프로그램 성능 저하

```
public class Box {
   private Object object;
   public void set(Object object) { this.object = object; }
   public Object get() { return object; }
}
```

Object
자바의 최상위 슈퍼클래스
모든 자바의 타입 대입 가능

```
Box box = new Box();
box.set("hello");                 //String 타입을 Object 타입으로 자동 타입 변환해서 저장
String str = (String) box.get();  //Object 타입을 String 타입으로 강제 타입 변환해서 얻음
```

제네릭 만들기

❖ 제네릭 클래스와 인터페이스

- 클래스나 인터페이스 선언부에 일반화된 타입 추가

```
public class MyClass<T> {
   T val;
   void set(T a) {
      val = a;
   }
   T get() {
      return val;
   }
}
```

제네릭 클래스 MyClass 선언, 타입 매개 변수 T

val의 타입은 T

T 타입의 값 a를 val에 지정

T 타입의 값 val 리턴

```
public class MyClass {
   String val;
   void set(String a) {
      val = a;
   }
   String get() {
      return val;
   }
}
```

- 제네릭 클래스 레퍼런스 변수 선언

```
MyClass<String> s;
List<Integer> li;
Vector<String> vs;
```

2절. 제네릭 타입

❖ 제네릭 타입 사용 여부에 따른 비교

▪ 제네릭 타입 사용한 경우

- 클래스 선언할 때 타입 파라미터 사용
- 컴파일 시 타입 파라미터가 구체적인 클래스로 변경

```
public class Box<T> {
    private T t;
    public T get() { return t; }
    public void set(T t) { this.t = t; }
}
```

```
Box<String> box = new Box<String>();
```

```
public class Box<String> {
    private String t;
    public void set(String t) { this.t = t; }
    public String get() { return t; }
}
```

```
Box<String> box = new Box<String>();
box.set("hello");
String str = box.get();
```

```
Box<Integer> box = new Box<Integer>();
```

```
public class Box<Integer> {
    private Integer t;
    public void set(Integer t) { this.t = t; }
    public Integer get() { return t; }
}
```

```
Box<Integer> box = new Box<Integer>();
box.set(6);
int value = box.get();
```

예제

```
public class Box<T> {
    private T t;
    public T get() { return t; }
    public void set(T t) { this.t = t; }
}
public class BoxExample {
    public static void main(String[] args) {
        Box<String> box1 = new Box<String>();
        box1.set("hello");
        String str = box1.get();

        Box<Integer> box2 = new Box<Integer>();
        box2.set(6);
        int value = box2.get();
    }
}
```

실습

- **public class Car를 제네릭 이용하여 만드시오**
  - **모든 메소드나 필드는 같은 타입을 쓴다고 가정**

  - **필드명**
    - 제네릭타입 model
  - **메소드**
    - 메소드명 : getModel
      - 리턴 : 제네릭타입 model
      - 매개변수 : 없음
    - 메소드명 : setMode
      - 리턴 : 없음
      - 매개변수 :
        - » 타입 : 제네릭타입
        - » 매개변수명 : m

제너릭 멀티타입 파라미터

❖ 제네릭 타입은 두 개 이상의 타입 파라미터 사용 가능

- 각 타입 파라미터는 콤마로 구분
  - Ex) class<K, V, ...> { ... }
  - interface<K, V, ...> { ... }

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

```
public class Product<T, M> {
    private T kind;
    private M model;

    public T getKind() { return this.kind; }
    public M getModel() { return this.model; }

    public void setKind(T kind) { this.kind = kind; }
    public void setModel(M model) { this.model = model; }
}
```

```
public class ProductNoGeneric {

    private Tv kind;
    private String model;

    public Tv getKind() { return this.kind; }
    public String getModel() { return this.model; }

    public void setKind(Tv kind) { this.kind = kind; }
    public void setModel(String model) { this.model = model; }
}
```

- 자바 7부터는 다이아몬드 연산자 사용해 간단히 작성과 사용 가능

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

구체화 오류

- **타입 매개 변수에 기본 타입은 사용할 수 없음**
  - **Wrapper 클래스로 사용해야 함**

```
Vector<int> vi = new Vector<int>(); // 컴파일 오류. int 사용 불가
```


```
Vector<Integer> vi = new Vector<Integer>(); // 정상 코드
```

제네릭과 배열

❖ 제네릭에서 배열의 제한

- 제네릭 클래스 또는 인터페이스의 배열을 허용하지 않음

```
GStack<Integer>[] gs = new GStack<Integer>[10];
```

- 제네릭 타입의 배열도 허용되지 않음

```
T[] a = new T[10];
```

- 타입 매개변수의 배열에 레퍼런스는 허용

```
return (T)stck[tos]; // 타입 매개 변수 T타입으로 캐스팅
```

```
public void myArray(T[] a) {....}
```

Wrapper class

❖ 자바에서 데이터 타입

- 기본형(primitive type)
- 참조형(reference type)

wrapper class

기본형	래퍼클래스	생성자
boolean	Boolean	Boolean(boolean value) Boolean(String s)
char	Character	Character(char value)
byte	Byte	Byte(byte value) Byte(String s)
short	Short	Short(short value) Short(String s)
int	Integer	Integer(int value) Integer(String s)
long	Long	Long(long value) Long(String s)
float	Float	Float(double value) Float(float value) Float(String s)
double	Double	Double(double value) Double(String s)

Wrapper class

❖ 사용이유

- 자바의 기본데이터 타입을 객체처럼 다루기 위해
- 기본데이터 타입을 조작하기 위한 메소드 제공

❖ Wrapper 클래스 객체 -> 기본 데이터 타입 변환방법

```
Integer i = new Integer(10);
int i1 = i.intValue();
```

예제

```
public class Tv {

}
public class Product<T, M> {
    private T kind;
    private M model;

    public T getKind() { return this.kind; }
    public M getModel() { return this.model; }

    public void setKind(T kind) { this.kind = kind; }
    public void setModel(M model) { this.model = model; }
}
public class ProductExample {
    public static void main(String[] args) {
        Product<Tv, String> product1 = new Product<Tv, String>();
        product1.setKind(new Tv());
        product1.setModel("스마트Tv");
        Tv tv = product1.getKind();
        String tvModel = product1.getModel();

        Product<Car, String> product2 = new Product<Car, String>();
        product2.setKind(new Car());
        product2.setModel("□□");
        Car car = product2.getKind();
        String carModel = product2.getModel();
    }
}
```

제네릭 메소드

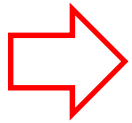
❖ 제네릭 메소드

- **매개변수 타입과 리턴 타입으로 타입 파라미터를 갖는 메소드**
- **제네릭 메소드 선언 방법**
  - 리턴 타입 앞에 "<>" 기호를 추가하고 타입 파라미터 기술
  - 타입 파라미터를 리턴 타입과 매개변수에 사용

```
public <타입파라미터,...> 리턴타입 메소드명(매개변수,...) { ... }
public <T> Box<T> boxing(T t) { ... }
```

```
Box<String> box = new Box<String>();
```

```
public class Box<T> {
  private T t;
  public T get() { return t; }
  public void set(T t) { this.t = t; }
}
```



```
public class Box<String> {
  private String t;
  public void set(String t) { this.t = t; }
  public String get() { return t; }
}
```

```
Box<String> box = new Box<String>();
box.set("hello");
String str = box.get();
```

4절. 제네릭 메소드

- **제네릭 메소드 호출하는 두 가지 방법 (p.661~664)**

```
리턴타입 변수 = <구체적타입> 메소드명(매개값);      //명시적으로 구체적 타입 지정
리턴타입 변수 = 메소드명(매개값);                  //매개값을 보고 구체적 타입을 추정
```

```
Box<Integer> box = <Integer>boxing(100);      //타입 파라미터를 명시적으로 Integer 로 지정
Box<Integer> box = boxing(100);               //타입 파라미터를 Integer 으로 추정
```

예제

```
public class Box<T> {
    private T t;
    public T get() { return t; }
    public void set(T t) { this.t = t; }
}
public class Util {
    public static <T> Box<T> boxing(T t) {
        Box<T> box = new Box<T>();
        box.set(t);
        return box;
    }
}
public class BoxingMethodExample {
    public static void main(String[] args) {
        Box<Integer> box1 = Util.<Integer>boxing(100);
        int intValue = box1.get();

        Box<String> box2 = Util.boxing("홍길동");
        String strValue = box2.get();
    }
}
```

5절. 제한된 타입 파라미터

❖ 타입 파라미터에 지정되는 구체적인 타입 제한할 필요

- 상속 및 구현 관계 이용해 타입 제한

```
public <T extends 상위타입> 리턴타입 메소드(매개변수, ...) { ... }
```

  - 상위 타입은 클래스 뿐만 아니라 인터페이스도 가능
    - 인터페이스라고 implements 를 사용하지 않는다

- 타입 파라미터를 대체할 구체적인 타입
  - 상위타입이거나 하위 또는 구현 클래스만 지정 가능

예제

```
public class Util {
    public static <T extends Number> int compare(T t1, T t2) {
        double v1 = t1.doubleValue();
        //System.out.println(t1.getClass().getName());
        double v2 = t2.doubleValue();
        //System.out.println(t2.getClass().getName());
        return Double.compare(v1, v2);
    }
}
public class BoundedTypeParameterExample {
    public static void main(String[] args) {
        //String str = Util.compare("a", "b"); (x)

        int result1 = Util.compare(10, 20);
        System.out.println(result1);

        int result2 = Util.compare(4.5, 3);
        System.out.println(result2);
    }
}
```

6절. 와일드카드 타입

❖ 와일드카드 타입의 세가지 형태

- 제네릭타입<?> : Unbounded Wildcards (제한없음)
  타입 파라미터를 대치하는 구체적인 타입으로 모든 클래스나 인터페이스 타입이 올 수 있다.

- 제네릭타입<? extends 상위타입> : Upper Bounded Wildcards (상위 클래스 제한)
  타입 파라미터를 대치하는 구체적인 타입으로 상위 타입이나 하위 타입만 올 수 있다.

- 제네릭타입<? super 하위타입> : Lower Bounded Wildcards (하위 클래스 제한)
  타입 파라미터를 대치하는 구체적인 타입으로 하위 타입이나 상위 타입이 올 수 있다.

7절. 제네릭 타입의 상속과 구현

❖ 제네릭 타입을 부모 클래스로 사용할 경우

- 타입 파라미터는 자식 클래스에도 기술해야 !!!

```
public class ChildProduct<T, M> extends Product<T, M> { ... }
```

- 추가적인 타입 파라미터 가질 수 있음

```
public class ChildProduct<T, M, C> extends Product<T, M> { ... }
```

❖ 제네릭 인터페이스를 구현할 경우

- 제네릭 인터페이스를 구현한 클래스도 제네릭 타입

Thank You !

이것이 자바다(http://cafe.naver.com/thisjava)