제18장
제네릭(Generic)-1

1. 제네릭의 개요

▣ **제네릭 타입이란 타입을 파라메터화 하여, 실행 시에 구체적으로 해당하는 타입으로 결정이 되는 것**을 의미한다.

- JDK1.5부터 추가된 기능이며, 스레드, 컬렉션, 람다식, 스트림 등에서 사용된다.
- 아울러, java docs에 보면 제네릭 타입이 매우 많아서, 반드시 제네릭의 개념을 알고 접근해야 한다.

▣ **제네릭을 사용하면 컴파일 시에 강한 타입 체크 뿐만 아니라, 타입변환(Casting)을 사전에 제거**할 수가 있다.

```
List list = new ArrayList();
list.add(100);
int value = (Integer)list.get(0);
```

[비제네릭 코드]

★ 타입변환이 자주 일어나면, Application P/G의 기능이 떨어진다.

```
List<Integer> list = new ArrayList<Integer>();
list.add(100);
int value = list.get(0);
```

[제네릭 코드]

2. 제네릭의 개념 및 선언

- **타입을 파라메터로 갖는 클래스 및 인터페이스를 칭한다.**
  - 선언을 할 때, 클래스 또는 인터페이스명 뒤에 “〈〉”(꺽쇠)가 붙는다.
  - 아울러, 꺽쇠 사이에는 타입 파라메터가 위치하게 된다.
- **타입 파라메터란 것은 제네릭 클래스나 인터페이스를 설계시에 보통 알파벳 한 문자로 표식**을 한다. 그 후, 개발 코드에서는 직접 타입 파라메터에 구체적 클래스를 지정해야 한다.

```
public class Box<T> {

    private T t;

    public T getT() {
        return t;
    }
    public void setT(T t) {
        this.t = t;
    }
}
```

```
public interface A<T> {

    public void method(T t);

}
```

* <T>란 아직 타입이 결정이 안됨을 의미 설계단계에서 <T>를 이용함.(개발 시 구체적 타입 결정함.

3. 제네릭의 사용 예

▣ **제네릭 타입을 사용한 경우**

– 클래스를 선언할 때, 타입 파라메터를 기술하고, 컴파일 시 타입 파라메터가 구체적인 클래스로 변경이 된다.

```
public class Person<T> {
    private T t;

    public T getT() {
        return t;
    }

    public void setT(T t) {
        this.t = t;
    }
}
```

```
Person<String> person = new Person<String>();
person.setT("hello");
String str = person.getT();
```

```
Person<Integer> person2 = new Person<Integer>();
person2.setT(100);
Integer value = person2.getT();
```

```
Person<Double> person3 = new Person<Double>();
person3.setT(100.17);
Double value1 = person3.getT();
```

* 개발 시, 구체적 타입을 기술 하면 설계단계에 있는 제네릭 타입 T가 구체적 클래스 타입인 String, Integer, Double, 사용자정의 클래스 등으로 컴파일러가 대체시킨다.(타입변환 無)

4. 멀티 타입 파라메터

▣ 멀티 타입 파라메터

– 제네릭은 2개 이상의 타입 파라메터를 사용해서 선언할 수가 있으며,
각 타입 파라메터는 콤마(,)로 구분한다.

```
public class Student<T,M> {
    private T t;
    private M m;

    public T getT() { return this.t; }
    public void setT(T t) { this.t = t; }

    public M getM() { return this.m; }
    public void setM(M m){ this.m = m; }
}
```

* T,M처럼 의미 있는 알파벳이 오는것이 권장 사항이지만,
프로그래머 마음대로 결정할 수도 있다.

* 단, static멤버에는 제네릭 타입을 선언할 수가 없다.
그 이유는 제네릭 타입은 인스턴스가 생성될 때, 결정
되어지기 때문에 static멤버에는 쓸 수가 없다.
또한 배열 역시도 instanceof연산자에도 쓸 수가 없다.

```
Student<Integer, String> s = new Student<Integer, String>();
```

감사합니다.