Java

제 18 강

제네릭

1. 지네릭 프로그래밍
    1.1 제네릭 클래스
    1.2 제네릭 메소드

1.1 제네릭 클래스

1.1 제네릭 프로그래밍(generic programming)

- 다양한 타입의 객체를 동일한 코드로 처리하는 기법
- 제네릭은 컬렉션 라이브러리에 많이 사용

제네릭 클래스

```
class Box<T>
{
  ...
}
```

String을 저장하는 Box

```
class Box<String>
{
  ...
}
```

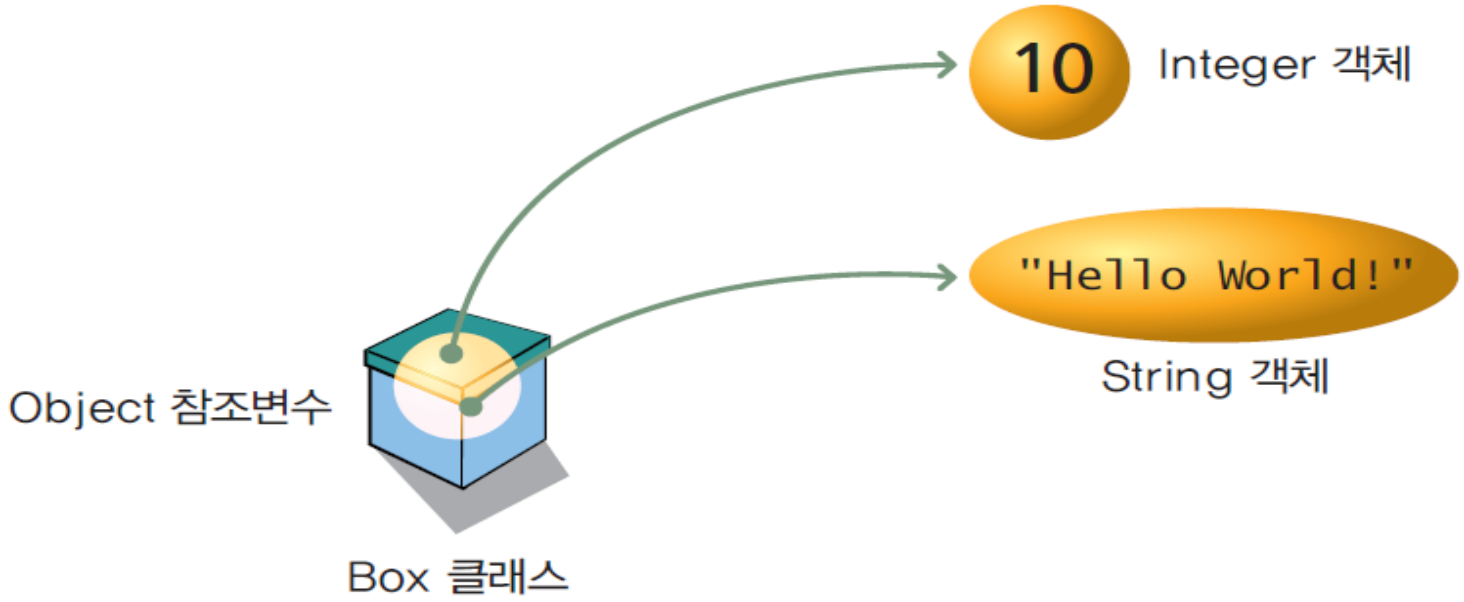
Integer을 저장하는 Box

```
class Box<Integer>
{
  ...
}
```

1.2 기존방법

- 먼저 단 하나의 데이터만을 저장할 수 있는 Box라는 간단한 클래스를 작성해 보자.

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



1.3 예제

- 실제로 Box 클래스는 여러 가지 다양한 타입의 데이터를 저장할 수 있다.

```
Box b = new Box();
b.set(new Integer(10));         // ❶ 정수 객체 저장
b.set("Hello World!");          // 정수 객체가 없어지고 문자열 객체를 저장
String s = (String)b.get();     // ❷ Object 타입을 String 타입으로 형변환
```

- 문자열을 저장하고서도 부주의하게 Integer 객체로 형변환을 할 수도 있으며 이것은 실행 도중에 오류를 발생한다.

```
b.set("Hello World!");
Integer i = (Integer)b.get( );     // 오류! 문자열을 정수 객체로 형변환
```

실행결과

```
Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot
be cast to java.lang.Integer at GenericTest.main(GenericTest.java:10)
```

1.4 제네릭을 이용한 방법

- 제네릭 클래스 (generic class) 에서는 타입을 변수로 표시한다.
- 이것을 타입 매개변수 (type paramter)라고 하는데 타입 매개변수는 객체 생성 시에 프로그래머에 의하여 결정된다.

```
class name<T1, T2, ..., Tn> {    ...   }
```

- Box 클래스를 제네릭으로 다시 작성하여 보면 다음과 같다.
- "public class Box"을 "public class Box<T>"으로 변경하면 된다. 여기서는 T가 타입 매개변수가 된다.

```
public class Box<T> {  <-------------------------------------------- T는 타입을 의미한다.
    private T data;
    public void set(T data)  {   this.data = data;    }
    public T get()           {   return data;         }
}
```

- 타입 매개변수의 값은 객체를 생성할 때 구체적으로 결정된다. 예를 들어서 문자열을 저장하는 Box 클 래스의 객체를 생성하려면 T 대신에 String을 사용하면 된다.

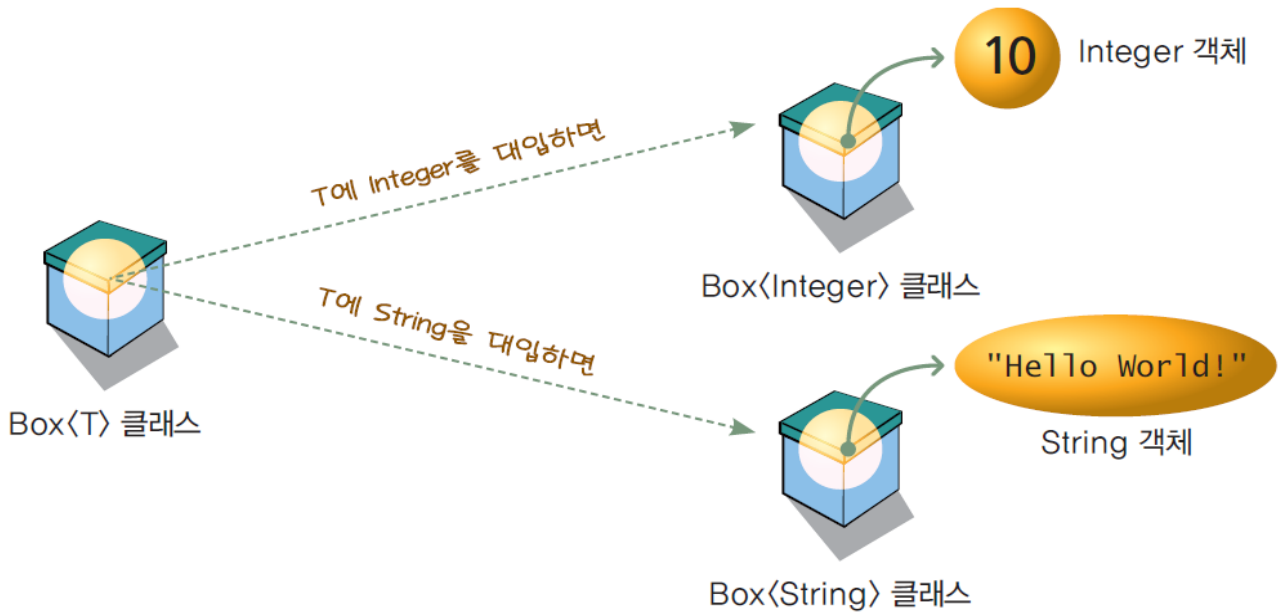
```
Box<String> b = new Box<String>();
```

1.4 제네릭을 이용한 방법(2)

- 만약 정수를 저장하는 Box 클래스의 객체를 생성하려면 다음과 같이 T 대신에 <Integer>를 사용하면 된다.

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

하지만 int는 사용할 수 없는데, int는 기초(기본)자료형이고 클래스가 아니기 때문이다



1.4 제네릭을 이용한 방법(3)

- 문자열을 저장하는 객체를 생성하여 사용하면 다음과 같다

```
Box<String> b = new Box<String>();
b.set("Hello World!");            // 문자열 타입 저장
String s = Box.get();
```

- 만약 Box<String>에 정수 타입을 추가하려고 하면 컴파일러가 컴파일 단계에서 오류를 감지할 수 있 다. 따라서 더 안전하게 프로그래밍할 수 있다.

```
Box<String> stringBox = new Box<String>();
stringBox.set(new Integer(10));          // 정수 타입을 저장하려고 하면 컴파일 오류!
```

실행결과

```
The method set(String) in the type Box<String> is not applicable for the arguments
(Integer) at GenericTest.main(GenericTest.java:27)
```

1.5 타입 매개 변수의 표기

- E–Element(요소: 자바 컬렉션 라이브러리에서 많이 사용된다.)
- K - Key
- N - Number
- T - Type
- V–Value
- S, U, V 등 - 2번째, 3번째, 4번째 타입

1.5 다이아몬드

- 자바 SE 7 버전부터는 제네릭 클래스의 생성자를 호출할 때, 타입 인수를 구체적으로 주지 않아도 된다. 컴파일러는 문맥에서 타입을 추측한다.

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

생성자 호출 시 구체적인 타입을 주지 않아도 된다.

1.6 다중타입매개변수(Multiple Type Parameters)

```
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}

public class OrderedPair<K, V> implements Pair<K, V> {
    private K key;
    private V value;

    public OrderedPair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey(){ return key; }
    public V getValue() { return value; }
}
```

타입 매개변수가 2개인 인터페이스를 정의한다.

K는 key의 타입이고, V는 value의 타입이다.

- 위의 정의를 이용하여서 객체를 생성해보면 다음과 같다

```
Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Even", 8);
Pair<String, String>  p2 = new OrderedPair<String, String>("hello", "world");
```

1.7 Raw 타입

- Raw 타입은 타입 매개 변수가 없는 제네릭 클래스의 이름이다.
- 앞의 box 클래스를 다음과 같이 사용하면 Raw 타입이 된다.

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

```
Box rawBox = new Box();
```

- Raw 타입은 JDK 5.0 이전에는 제네릭이 없었기 때문에 이전 코드와 호환성을 유지하기 위하여 등장
- 타입을 주지 않으면 무조건 Object 타입으로 간주

```
Box<String> stringBox = new Box<>();
Box rawBox = stringBox;                    // OK
```

제네릭스 + ArrayList 예제

```java
// <String> 제네릭스를 사용한 경우
ArrayList<String> mStringList = new ArrayList<String>();
mStringList.add("안녕하세요");
mStringList.add("3");

// <Integer> 제네릭스를 사용한 경우
ArrayList<Integer> mIntegerList = new ArrayList<Integer>();
mIntegerList.add(1);
mIntegerList.add(2);

// <Class명> 제네릭스를 사용한 경우
ArrayList<DtoTv> mTvList = new ArrayList<DtoTv>();
mTvList.add(new DtoTv());
mTvList.add(new DtoTv("New Tv"));

for(DtoTv t : mTvList){
    System.out.println(t);
}

// < 제네릭스를 사용하지 않은 경우 >
// 모든타입을 "add(Object)" Object형으로 List에 Add를
// 시켜 주므로, List에서 자료를 꺼내올때는 각각에 맞는 형병환을
// 꼭 해주어야 하기 때문에 번거 롭다.
ArrayList mOriginalList = new ArrayList();
mOriginalList.add(1);
mOriginalList.add("string");
mOriginalList.add(new DtoTv("original"));

int a = (Integer) mOriginalList.get(0);
String b = (String) mOriginalList.get(1);
DtoTv c = (DtoTv) mOriginalList.get(2);
```

중간점검

중간점검

1. 왜 데이터를 Object 참조형 변수에 저장하는 것이 위험할 수 있는가?
2. Box 객체에 Rectangle 객체를 저장하도록 제네릭을 이용하여 생성하여 보라.
3. 타입 매개변수 T를 가지는 Point 클래스를 정의하여 보라. Point 클래스는 2차원 공간에서 점을 나타낸다.

1.2 제네릭 메소드

1.8 제네릭 메소드

- 메소드에서도 타입 매개 변수를 사용하여서 제네릭 메소드를 정의할 수 있다.
- 타입 매개 변수의 범위가 메소드 내부로 제한된다.

```
public class Array
{

    public static <T> T getLast(T[] a)
    {
        return a[a.length-1];
    }
}
```

제네릭 메소드 정의

- 제네릭 메소드를 호출하기 위해서는 실제 타입을 꺽쇠 안에 적어준다.

```
String[]  language = { "C++", "C#", "JAVA" };
String  last = Array.<String>getLast(language);    // last는 "JAVA"
```

- 여기서 메소드 호출시에는 <String>는 생략하여도 된다. 왜냐하면 컴파일러는 이미 타입 정보를 알고 있기 때문이다. 즉 다음과 같이 호출하여도 된다.

```
String  last = Array.getLast(language);        // last는 "JAVA"
```

1.9 한정된 타입 매개변수

- 배열 원소 중에서 가장 큰 값을 반환하는 제네릭 메소드를 작성하여 보자

```
public class Array
{
   public static <T> T getMax(T[] a)
   {
      if (a == null || a.length == 0)
         return null;
      T largest = a[0];
      for (int i = 1; i < a.length; i++)
         if (largest.compareTo(a[i]) > 0)
            largest = a[i];
      return largest;
   }
}
```

- 타입 매개변수 T가 가리킬 수 있는 클래스의 범위를 Comparable 인터페이스를 구현한 클래스로 제한하는 것이 바람직하다

```
public static <T extends Comparable> T getMax(T[] a)
{
   ...
}
```

중간점검

중간점검

1. 제네릭 메소드 sub()에서 매개변수 d를 타입 매개변수를 이용하여서 정의하여 보라.
2. displayArray()라는 메소드는 배열을 매개변수로 받아서 반복 루프를 사용하여서 배열의 원소를 화면에 출력한다. 어떤 타입의 배열도 처리할 수 있도록 제네릭 메소드로 정의하여 보라.

감사합니다.