JAVA Generic Method

프로그램은 사람이 이해하는 코드를 작성.

느려도 꾸준하면 경기에서 이긴다.

작성자 : 홍효상
이메일 : hyomee@naver.com
소스 : https://github.com/hyomee/JAVA_EDU

Content

6. Generic Method

1. Generic 이란
2. Generic Class
3. Generic Method
4. Generic WildCard

1. Generic 이란

" Data type을 특정한 type하나로 정하지 않고 사용할 때마다 바뀔 수 있게 범용적이고 포괄적으로 지정 "

Generic

- 포괄적인, 총칭의, 회사 이름이 붙지 않은, 일반 명칭으로 판매되는 사전적 의미로, 딱 하나를 정하지 않고 범용적이고 포괄적이라는 의미
- Data type을 특정한 type하나로 정하지 않고 사용할 때마다 바뀔 수 있게 범용적이고 포괄적으로 지정한다 라는 의미
- Object Class의 한계를 극복
- **호출되는 시점에 실제 Generic 타입을 지정**

01. 왜 Generic가 만들어 졌을까 ?

```java
public class GenericMethodMain {
    public static void main(String... args) {

        List list = new ArrayList<>();
        list.add("홍길동");
        list.add("홍당무");
        list.add("홍사과");

        for (int i = 0; i < list.size(); i++) {
            String str = list.get(i);
            System.out.println(str);
        }
    }
}
```

java: incompatible types: java.lang.Object
cannot be converted to java.lang.String

```java
public class GenericMethodMain {
    public static void main(String... args) {

        List list = new ArrayList<>();
        list.add("홍길동");
        list.add("홍당무");
        list.add("홍사과");

        for (int i = 0; i < list.size(); i++) {
            String str = (String) list.get(i);
            System.out.println(str);
        }
    }
}
```

Type Casting 이 빈번하게 일어나서 성능 저하 발생
- 수만개의 Data가 있다면

```java
public class GenericMethodMain {
    public static void main(String... args) {

        List<String> list = new ArrayList<>();
        list.add("홍길동");
        list.add("홍당무");
        list.add("홍사과");

        for (int i = 0; i < list.size(); i++) {
            String str = list.get(i);
            System.out.println(str);
        }
    }
}
```

Bast Code

```java
for (String str: list) {
    System.out.println(str);
}
```

1. Generic Class

Generic Class

- Class or Interface로 정의 할 수 있음
- 클래스 내부에서 사용하는 데이터의 타입(Type)을 클래스의 인스턴스를 생성할 때 결정하는 것.
- 기본 데이터 타입(int, long..)에 대해서는 지정이 불가능
- Class 를 정의 하는 시점에 타입을 지정 하는 것이 아니라 객체 생성 시점에 실제 타입을 지정

➢ 문법 구조

class or Interface에서 사용 되는 Generic Type 정의

선언 : 접근지정자 class 클래스명<GenericType [, GenericType ..] >

호출 : 제너릭클래스명 <실제 Type> 변수명 = new 제너릭클래스명<실제 Type>()

입력 매개변수의 실제 타입을 의미함

선언 : 접근지정자 interface 클래스명<GenericType [, GenericType ..] >

➢ 관례적 표기와 의미

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

➢ 주의사항

- 기본 유형으로 제네릭 유형을 인스턴스화 할 수 없음
- 유형 매개 변수의 인스턴스를 생성 할 수 없음
- 유형이 유형 매개 변수 인 정적 필드를 선언 할 수 없음
- 매개 변수가있는 유형에 캐스트 또는 instanceof를 사용할 수 없음
- 매개 변수가있는 유형의 배열을 만들 수 없음
- 매개 변수가있는 유형의 개체를 생성, 캐치 또는 던질 수 없음
- 각 오버로드의 형식 매개 변수 유형이 동일한 원시 유형으로 지워지는 메서드를 오버로드 할 수 없음

1. Generic Class

Class 내부에 사용 하는 Generic Type

```java
public class GenericClass<T, V> {
  private T name;
  private V price;

  GenericClass(T name, V price) {
    this.name = name;
    this.price = price;

  }
  public T getName() {
    return name;
  }

  public void setName(T name) {
    this.name = name;
  }

  public V getPrice() {
    return price;
  }

  public void setPrice(V price) {
    this.price = price;
  }
```

```java
public static void main(String... args) {

  List<GenericClass<String, Integer>> cars = new ArrayList<>();
  cars.add((new GenericClass<>("쏘나타", 10000)));
  cars.add((new GenericClass<>("그랜저", 20000)));

  for(GenericClass<String, Integer> car: cars) {
    System.out.println(car.getName() + "\t가격 : " + String.format("%,d", car.getPrice() )
            + " (부가세 : " + String.format("%,.1f",car.getPrice().floatValue() * 0.1) +")");
  }


  List<GenericClass<String, Float>> fruits = new ArrayList<>();
  fruits.add((new GenericClass<>("사과", 256.5f)));
  fruits.add((new GenericClass<>("복숭아", 345.6f)));

  for(GenericClass<String, Float> fruit: fruits) {
    System.out.println(String.format("%s \t가격 : %,.1f ( 부가세 : %,.1f )"
            , fruit.getName()
            , fruit.getPrice()
            , fruit.getPrice() * 0.1 ));

  }
```

1. Generic Method

Generic Method

- Method 만들 때 파라메터와 반환 값의 자료형을 Generic로 선언 한 Method
- 입력 매개 변수 값으로 Generic Type을 유추 할 수 있을 경우는 생략 가능
- Generic Method 내부에서는 매개변수로 한 참조 변수의 메서드로 Object Class의 Method만 가능

➢ 관례적 표기와 의미

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

➢ 문법 구조

입력 매개변수의 타입의 개수 만큼

선언 : 접근지정자 < T [, ..] > T 메서드명(T t [, V v, ...]);

호출 : 참조객체.<실제 제네릭 타입>메서드명(입력매개변수)

입력 매개변수의 실제 타입을 의미함

```
GenericMethod genericMethod = new GenericMethod();
int num01 =   genericMethod.<Integer> method(10);
System.out.println(num01);

// 입력 매개 변수가 Genetic Type가 유추 할 수 있는 경우 생략 가능
int num02 =   genericMethod.method(10);
System.out.println(num02);

String str02 = genericMethod.methodRnString("안녕" , 10);
System.out.println(str02);

genericMethod.methodPrint("name" , "홍길동");
```

```
10
I0
안녕10
name : 홍길동
```

```
<T> T method(T t) {
  return t;
}

// Generic Method 내부에서 매개변수의 산술 연산 되지 않음 ->  Object
<T> String methodRnString(T t1, T t2){
  // t1 +  t2 : Operator '+' cannot be applied to 'T', 'T'
  return t1 + "" + t2;
}

<K,V> void methodPrint(K t1, V v1){
  System.out.println(t1 + " : " + v1 );
}
```

2. Generic Method

Generic Method -

- extends : 매개 변수에 특정 타입만 받게 제한 할 때 사용
- 여러 개 사용시 임의로 타입 설정

```
String str03 =  genericMethod.methodPrintRnString("name" , "홍길동");
System.out.println(str03);

System.out.println(genericMethod.checkEquals(10 , 2.1));
System.out.println(genericMethod.checkEquals("사과" , "배"));
System.out.println(genericMethod.checkEquals("사과" , "사과"));

System.out.println(genericMethod.<String, Integer, Object>extednsMethod("사과" , 0, "첫문자 : "));
System.out.println(genericMethod.extednsMethod("사과" , 0, "첫문자 : "));
```

첫번째 파라메터 : 문자
두번째 매개변수 : 숫자
세번째 파라메터 : Object

```
<K,V> void methodPrint(K t1, V v1){
  System.out.println(t1 + " : " + v1 );
}

<K,V> V methodPrintRnString(K t1, V v1){
  // t1 +  t2 : Operator '+' cannot be applied to 'T', 'T'
  System.out.println(t1 + " : " + v1 );
  return (V) (t1 + " : " + v1);
}


<T, V> boolean checkEquals(V v1, V v2){
  return v1.equals(v2);
}
// Generic Method 내부에서는 매개변수로 한 참조 변수의 메서드로
// Object Class의 Method만 가능
<T extends String, I extends Integer, V> String extednsMethod(T t, I i, V v){
  char ch = t.charAt(i);
  return  v.toString() + ch ;
}
```

1. Generic WildCard

Generic WildCard

- **와일드카드 타입에는 총 세가지의 형태가 있으며 물음표(?)라는 키워드로 표현**
  : 제네릭타입<?>
  - 타입 파라미터를 대치하는 것으로 모든 클래스나 인터페이스타입이 올 수 있음
  : 제네릭타입<? extends 상위타입>
  - 와일드카드의 범위를 특정 객체의 하위 클래스만 올 수 있음.
  : 제네릭타입<? super 하위타입> :
  - 와일드카드의 범위를 특정 객체의 상위 클래스만 올 수 있음

```java
public class GenericWildCard {
  public void printList(List<?> list) {
    for (Object obj : list) {
      System.out.println(obj + " ");
    }
  }

  public int sum(List<? extends Number> list) {
    int sum = 0;
    for (Number i : list) {
      sum += i.doubleValue();
    }        return sum;
  }

  public List<? super Integer> addList(List<? super Integer> list) {
    for (int i = 1; i < 5; i++) {
      list.add(i);
    }
    return list;
  }
}
```

```java
GenericWildCard genericWildCard = new GenericWildCard();
List<String> stringList = new ArrayList<>();
stringList.add("우");
stringList.add("리");

genericWildCard.printList(stringList);

// incompatible types: java.util.List<java.lang.String> cannot be converted to
// java.util.List<? extends java.lang.Number>
// genericWildCard.sum(stringList);

// incompatible types: java.util.List<java.lang.String> cannot be converted
//to java.util.List<? super java.lang.Integer>
// genericWildCard.addList(stringList);

List<Integer> integerList = new ArrayList<>();
integerList.add(10);
integerList.add(20);
genericWildCard.addList(integerList);
genericWildCard.printList(integerList);
```