1. 제네릭스 기본

◆ 제네릭스

- 매개변수화된 자료형(**parameterized type**)
- 클래스에 적용- 제네릭 클래스, 인터페이스에 적용-제네릭 인터페이스, 메소드에 적용-제네릭 메소드
- 제네릭스를 이용하면 매개변수에 다양한 자료형의 데이터를 넘길 수 있음.
- 제네릭스는 코드 재사용성을 높여 줌.

void set(Integer x)
{
  System.out.println(x);
}
	void set(T x)
	{
	  System.out.println(x);
	}

매개변수 x에는 Integer 객체를 넣을 수 있습니다.	매개변수 T 자리에는 자료형이 와야 하는데, T라고 적었습니다. 이것은 어떤 자료형도 T에 넣을 수 있다는 표현입니다.

1. 제네릭스 기본

◆ 제네릭 클래스

- 클래스의 객체를 생성할 때에는 < > 기호를 사용하여 어떤 타입의 인수를 넣을지를 알려야 함.
- 형태

클래스명<매개변수 타입 리스트> {
...
}
	클래스명<타입> 객체명 = new 클래스명<타입>();

1. 제네릭스 기본

- 예제

코드 229

```
class Data<T> {   // 클래스명 옆에 제네릭 기호 <>를 적고 그 안에 매개변수 기술함.
    T obj;        // 인스턴스 변수 obj의 자료형은 T임.
    Data(T ob) {  // 생성자 Data는 자료형이 T인 인수 한 개를 입력받음.
        obj = ob;
    }
    T getObj() {  // 인스턴스 변수 obj의 자료형은 T임.
        return obj;
    }
    void showType() {
        System.out.println("Type of T : " + obj.getClass().getName());
    }

}

public class Code229 {
    public static void main(String[] args) {
        Data<Integer> d1 = new Data<Integer>(100);  // 정수 100 인수
        System.out.println(d1.getObj());
        d1.showType();

        Data<String> d2 = new Data<String>("JAVA");  // 문자열 "JAVA" 인수
        System.out.println(d2.getObj());
        d2.showType();
    }
}
```

결과

```
100
Type of T : java.lang.Integer
JAVA
Type of T : java.lang.String
```

1. 제네릭스 기본

- 제네릭에 이용할 수 있는 자료형

-레퍼런스 형.

-기본 자료형을 제네릭으로 사용하고자 한다면, **wrapper** 클래스인 **Integer, Double** 등을 이용.

- 제네릭 타입도 엄격하게 문법에 따라야 함

```
Data<Integer> d1 = new Data<Integer>(100);
Data<String> d2 = new Data<String>("100");

if (d1 == d2)  // 에러 발생
    System.out.println("same data");
```

1. 제네릭스 기본

- 예제

코드 230

```
class TwoGenerics<T, V> {
  T data1;
  V data2;
  TwoGenerics(T d1, V d2 ) {
          data1 = d1;
          data2 = d2;
  }
  void showGenericType() {
          System.out.println("Type of T : " + data1.getClass().getName());
          System.out.println("Type of V : " + data2.getClass().getName());
  }
  T getData1() { return data1; }
  V getData2() { return data2; }
}
public class Code230 {
  public static void main(String[] args) {
      TwoGenerics<Integer, String> x =
                          new TwoGenerics<Integer, String>(100, "hello");
          x.showGenericType();
          int y = x.getData1();
          System.out.println("value : " + y);
          String z = x.getData2();
          System.out.println("value : " + z);
  }
}
```

결과

```
Type of T : java.lang.Integer
JAVA of V : java.lang.String
value : 100
value : hello
```

1. 제네릭스 기본

◆ 제한된 제네릭 타입

- 형태 `< T extends V >` -제네릭 T 자리에는 클래스 타입이 V이거나 V 클래스의 하위 클래스 타입만 올 수 있다는 뜻

```
class Parent
{

}

class Child1 extends Parent
{

}

class Child2 extends Parent
{

}
```

```
class Data<T extends Parent>
{
    ......
}
```

이 자리에는 Parent 클래스 또는 Parent 클래스의 하위 클래스만 넣을 수 있습니다.

```
Data<Parent>
Data<Child1>   } OK
Data<Child2>

Data<String>   } error
```

1. 제네릭스 기본

- 예제

코드 231

```
class Data<T extends Number> {  // Number 클래스의 하위 클래스 타입
    T obj;
    Data(T ob) {
        obj = ob;
    }
    int calcMultiple(int n) {
        return obj.intValue() * n;
    }
}

public class Code231 {
    public static void main(String[] args) {
        Data<Integer> d = new Data<Integer>(100);  // Integer는 Number의 하위 클래스임.
        int result = d.calcMultiple(5);
        System.out.println(result);

        Data<Double> e = new Data<Double>(17.5);  // Double 역시 Number의 하위 클래스임.
        int result2 = e.calcMultiple(5);
        System.out.println(result2);
    }
}
```

결과

```
500
85
```

1. 제네릭스 기본

◆ 와일드카드 인수

- '?'로 나타냄-와일드카드 자리에는 어떤 클래스 타입도 올 수 있다는 의미.

코드 232

```
class WithWild<T extends Number> {
        T data;
        WithWild (T d) { data = d; }
        boolean same(WithWild<?> x) {
                if (Math.abs(data.doubleValue()) == Math.abs(x.data.doubleValue()))
                        return true;
                return false;
        }
}
public class Code232 {
        public static void main(String[] args) {
                WithWild<Integer> a= new WithWild<Integer>(6);
                WithWild<Double> b = new WithWild<Double>(-6.0);
                WithWild<Long> c = new WithWild<Long>(5L);
                if (a.same(b)) System.out.println("a and b are equal");
                else System.out.println("a and b are different");
                if (a.same(c)) System.out.println("a and c are equal");
                else System.out.println("a and c are different");
        }
}
```

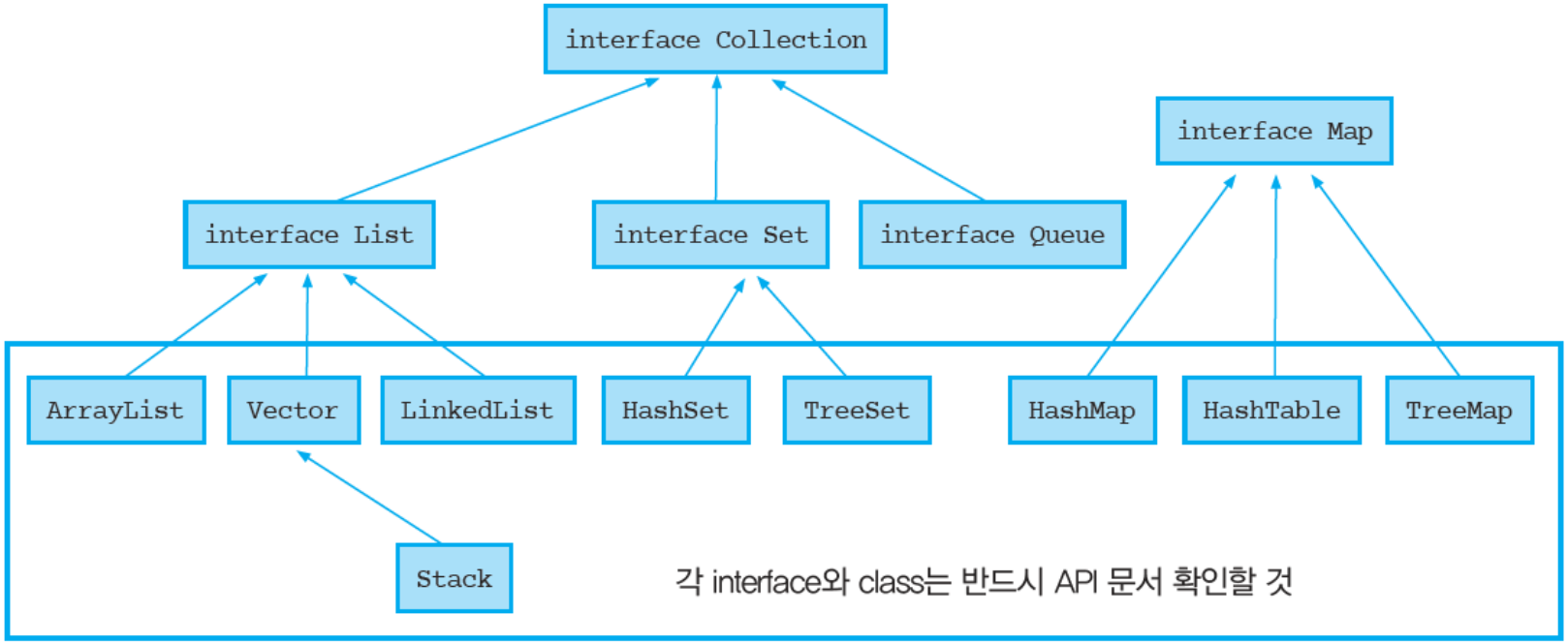
결과

```
a and b are equal
a and c are different
```

2. 컬렉션 프레임워크(Collection Framework)

◆ 컬렉션 프레임워크

- 자료 구조(**data structure**)는 컴퓨터 메모리에 데이터를 저장하는 형태. 대표적인 자료 구조는 '배열'. 배열 외에도 **ArrayList, LinkedList, Stack** 등과 같은 자료 구조가 자바 **API**에서 제공.컬렉션 프레임워크는 이러한 자료 구조 패키지들을 말함.



2. 컬렉션 프레임워크(Collection Framework)

- **List, Set, Map** 특징

인터페이스	특징
List	순서가 있는 데이터 집합으로 데이터 중복을 허용함.
	구현 클래스 : ArrayList, LinkedList, Stack, Vector 등
Set	순서가 없는 데이터 집합으로 데이터 중복을 허용하지 않음.
	구현 클래스 : HashSet, TreeSet 등
Map	〈key, value〉 쌍으로 이루어진 데이터 집합으로 순서가 없음. 키는 중복될 수 없고, 값은 중복 가능함.
	구현 클래스 : HashMap, TreeMap, Hashtable, Properties 등

-위의 클래스에는 레퍼런스 타입의 데이터인 객체만 저장 가능. 기본 자료형을 저장하려면 **wrapper** 클래스를 이용.

-List, Set, Map 인터페이스 모두 **Collection** 인터페이스를 상속받음.