객체지향 프로그래밍

- 제네릭 클래스
- 제네릭 메소드

제네릭 프로그래밍

- 제네릭 프로그래밍
  - 알고리즘의 추상적인 형태를 표현하기 위한 프로그래밍 방법
  - 데이터 타입을 매개 변수로 사용
  - C++의 템플릿과 유사
  - 자바의 제네릭 프로그래밍
    - 제네릭 클래스, 인터페이스
    - 제네릭 메소드

제네릭 클래스, 인터페이스

- 제네릭 클래스
  - 선언 형태

E는 타입 표현

Formal parameter type이라고 함

```
class GList<E> {
    void add(E  x) { ... }
     ...
}
```

  - 사용 형태

```
GList<Integer> myList = new GList<Integer>();
```

Integer : actual type argument

제네릭 클래스, 인터페이스

- 형태 : 파라미터화된 타입(parameterized type)
  - classOrInterface <referenceType [,ReferenceType]>
- 예
  - Vector<String>
  - Seq<Seq<A>>
  - Seq<String>.Zipper<Integer>
  - Collection<Integer>
  - Paint<String, String>
- 타입제거(type erasure) : 제네릭 프로그램은 제네릭을 사용하지 않은 형태로 변환
  - J2SE5.0 작성, 컴파일 ->J2SE 1.4에서 실행 가능

제네릭 클래스, 인터페이스

- 예제: StringVector.java

```
import java.util.*;
public class StringVector {
      public static void main(String[] args) {
          Vector<String> v = new Vector<String>();
          v.addElement("Hello");
          v.addElement("World");
          //v.add(5);    // 컴파일시에 발견할 수 있는 에러

          for (String s : v) {
              System.out.println(s);
          }
       }
}
```

문자열 원소를 갖는 벡터객체 생성

제네릭 클래스, 인터페이스

- 예제: NormalVector.java (앞예제와의 차이점)

```
import java.util.*;
public class NormalVector {
      public static void main(String[] args) {
          Vector v = new Vector();
          v.add("Hello");
          v.add("World");
          //v.add(5);    // 컴파일시에 발견할 수 없는 에러

          int n = v.size();
          for (int i = 0 ; i < n ; i++) {
              String s = (String) v.elementAt(i);
```


제네릭 클래스, 인터페이스

- 제네릭 생성 예제: ValueWrapper.java

```
public class ValueWrapper<T> {
        private T  value;

        public ValueWrapper(T value) {
                this.value = value;
        }

        public T value() {
                return value;
        }

        public static void main(String args[]) {
                ValueWrapper<String> sf = new ValueWrapper<String>("Hello");
                System.out.println(sf.value());

                ValueWrapper<Integer> si =
                        new ValueWrapper<Integer>(new Integer(10));
                System.out.println(si.value());
        }
}
```

T:타입파라메터

T타입의 멤버-value와 T타입의 메소드 value()가짐

스트링타입선언

정수 타입 선언

제네릭 클래스, 인터페이스

- 로타입(raw type) :제네릭 클래스에서 타입파라미터를 사용하지 않는 것. (예 Cell.java)

```
public class Cell<E> {
        private E value;

        public Cell(E v) {
                value = v;
        }

        public E get() {
                return value;
        }

        public void set(E v) {
                value = v;
        }

        public static void main(String[] args) {
                Cell<String> x = new Cell<String>("abc");
                String value = x.get();
                System.out.println(value);
                x.set("def");

                Cell y = x;
                value = (String) y.get();
                System.out.println(value);
                y.set("hello");
        }
}
```

String type을 갖는 Cell객체 x는 로타입 형태인 y에 값을 할당

y는 로타입이기 때문에 타입파라미터로 object가 사용. 따라서 형변환을 해야 value에 값을 할당 가능

제네릭 클래스, 인터페이스

- 타입 파라메터 범위 제한
  - Number의 자식 클래스인 타입 파라메터

```
public class C1<T  extends  Number> {
     ...
}
```

  - Person의 자식 클래스이고, Comparable 인터페이스를 구현한 타입 파라메터

```
public class C2<T extends  Person & Comparable>
{
    ...
}
```


제네릭 클래스, 인터페이스

- 예제: Pair.java

```
public class Pair<T extends Number & Comparable> {
        private T v1, v2;

        public Pair(T v1, T v2) {
                this.v1 = v1;
                this.v2 = v2;
        }

        public T first() {
                return v1;
        }

        public T second() {
                return v2;
        }

        public static void main(String args[]) {
                Pair<Integer> si = new Pair<Integer>(3, 4);
                System.out.println(si.first());

                Pair<Double> sd = new Pair<Double>(3.0, 4.0);
                System.out.println(sd.second());
        }
}
```

Pair 클래스의 타입파라미터 T는 Number 클래스로부터 상속받은 서브타입

제네릭 클래스, 인터페이스

- 제네릭 클래스의 슈퍼타입 관계


```
Vector<Object> vobj;
Vector<String> vstr = new Vector<String>();
vobj = vstr;    // 컴파일 에러
vobj.add(new Object());
String s = vstr.get(0);
```

vobj는 Vector<Object>타입이기 때문에 Object를 추가할 수 있다.

vstr에 Object타입이 할당된 것이 아니므로 문제 발생

제네릭 클래스, 인터페이스

- 와일드 카드
  - Vector<Object>, Vector<String>처럼 super/sub type의 문제점 발생 : 프로그램 개발할 때 불편함 발생
  - Unknown type을 지칭하기 위해서 타입 파라메터 자리에 기술
  - Vector<?>는 Vector<String>의 슈퍼타입

```
Vector<?> unknown;
Vector<String> vstr = new Vector<String>();
unknown = vstr;
```

와일드 카드를 사용하는 Vector<String>은 Vector<?>의 서브타입

제네릭 클래스, 인터페이스

- 예제: GenericSuperTest.java

```
import java.util.*;

public class GenericSuperTest {
        public static void main(String args[]) {
                Vector<String>  names;
                Vector<Integer> scores;
                Vector<Object>  obj;
                Vector<?>       wild;
                Vector          raw;

                names = new Vector<String>();
                scores = new Vector<Integer>();

                //scores = names;                 // 컴파일 에러
                //obj = names;                    // 컴파일 에러
                //obj = (Vector<Object>)names;    // 컴파일 에러
                wild = names;

                //names = wild;                   // 컴파일 에러
                names = (Vector<String>)wild;     // 경고 메시지
                //scores = (Vector<String>)wild; // 컴파일 에러

                raw = names;
                System.out.println("done.");
        }
}
```


제네릭 클래스, 인터페이스

- 와일드 카드를 사용하는 경우
  - Vector<Shape>의 문제점

```
public void calc(Vector<Shape> shapes) {
    for(Shape s : shapes)
        System.out.println(s.area());
}
```

Vector<Shape>과 Vector<Cicle>이 super/sub관계가 아니라서 Vector<Circle>타입은 매개변수로 받을 수 없다

  - 문제점 해결

```
public void calc2(Vector<? extends Shape> shapes) {
        for(Shape s : shapes)
            System.out.println(s.area());
        }
}
```

Vector<Shape>, Vector<Rect>, Vector<Cicle>등의 타입을 매개변수로 사용가능

제네릭 클래스, 인터페이스

- 예제: ShapeUser2.java

```
import java.util.*;

public class ShapeUser2 {
        public ShapeUser2() { }

        public void calc(Vector<Shape> shapes) {
                for(Shape s : shapes) {
                        System.out.println(s.area());
                }
                System.out.println();
        }

        public void calc2(Vector<? extends Shape> shapes) {
                for(Shape s : shapes) {
                        System.out.println(s.area());
                }
                System.out.println();
        }

        public static void main(String arg[]) {
                ShapeUser2 su = new ShapeUser2();

                Vector<Shape> sh = new Vector<Shape>();
                sh.add(new Circle(7));
                sh.add(new Rect(7, 5));
                su.calc(sh);
                su.calc2(sh);

                Vector<Circle> cs = new Vector<Circle>();
                cs.add(new Circle(7));
                //cs.add(new Rect(7, 5));  // 컴파일 에러
                //su.calc(cs);                // 컴파일 에러
                su.calc2(cs);
        }
}
```

calc()메소드는 Vector<Shape>타입을 매개변수, Vector<Circle>타입인 cs는 매개변수로 사용 못함.

calc2()는 Vector<Circle>도 매개변수로 받을 수 있음

제네릭 메소드

- 제네릭 메소드
  - 메소드에 타입 파라메터를 기입
  - 형태

[*methodModifier*] [<*typeParameter*>] *returnType name* (*arglist*)

[**throws** *exception*] { }

  - 예

```
static <E> void swap(E[] a, int i, int j) {
    E t = a[i];
    a[i] = a[j];
    a[j] = t;
}
```

제네릭 메소드

❑ 예제: Sorter.java

```
public class Sorter {
        public static <E> void swap(E[] a, int i, int j) {
                E t = a[i];
                a[i] = a[j];
                a[j] = t;
        }

        public static <E extends Comparable<E>> void sort(E[] a) {
                for(int i=0; i < a.length; i++) {
                        for(int j=0; j < i; j++) {
                                if(a[j].compareTo(a[i]) > 0) {
                                        swap(a, i, j);
                                }
                        }
                }
        }

        public static void main(String[] args) {
                String data[] = {"zxy", "abc", "def", "ello"};
                Sorter.sort(data);
                for(int i=0; i < data.length; i++) {
                        System.out.println(data[i]);
                }
        }
}
```

E타입의 배열을 사용하는 제네릭메소드. E는타입파라미터

Comparable 인터페이스를 구현하는 E를 타입파라미터로 갖는 제네릭 메소드

제네릭 메소드

❑ 예제: Maxer.java

```
public class Maxer {
        public static <T extends Comparable> T max(T t1, T t2){
                if(t1.compareTo(t2) > 0) {
                        return t1;
                } else {
                        return t2;
                }
        }

        public static void main(String args[]) {
                Integer max = Maxer.max(3, 4);
                System.out.println(max);

                String s = Maxer.max("AAA", "BB");
                System.out.println(s);
        }
}
```

Comparable로부터 상속받는 T타입을 사용하는 제네릭 메소드

제네릭 메소드

❑ 타입 파라메터에서 super
  - 타입 파라메터의 lower bound를 기술

❑ 예제: Sink.java

```
public interface Sink<T> {
    public void flush(T t);
}
```

❑ 예제: ConsoleSink.java

```
public class ConsoleSink<T> implements Sink<T>{
    public void flush(T t) {
        System.out.println(t);
    }
}
```


제네릭 메소드

❑ 예제: Flusher.java

```
import java.util.*;

public class Flusher {
        public static <T> T writeAll(Vector<T> data, Sink<T> s
                T last = null;
                for(T t: data) {
                        last = t;
                        sink.flush(t);
                }
                return last;
        }

        public static <T> T writeAll2(Vector<? extends T> data, Sink<T> sink) {
                T last = null;
                for(T t: data) {
                        last = t;
                        sink.flush(t);
                }
                return last;
        }

        public static <T> T writeAll3(Vector<T> data, Sink<? super T> sink) {
                T last = null;
                for(T t: data) {
                        last = t;
                        sink.flush(t);
                }
                return last;
        }

        public static void main(String args[]) {
                Sink<Object> s = new ConsoleSink<Object>();
                Vector<String> vs = new Vector<String>();
                vs.add("Hello");

                //Flusher.writeAll(vs, s); // 컴파일 에러

                Object o = Flusher.writeAll2(vs, s);
                //String str = Flusher.writeAll2(vs, s); // 컴파일 에러

                String str2 = Flusher.writeAll3(vs, s);
        }
}
```

동일한 T타입 파라미터의 Vector와 Sink를 매개변수로 받는다

T타입 파라미터의 Sink와 T로부터 상속받은 타입을 파라미터로 갖는 Vector를 매개변수로 받는다

T타입파라미터의 Vector와 T의 슈퍼타입을 파라미터로 갖는 Sink를 매개변수로 받는다.

String과 Object는 타입이 다르기 때문에 writeAll() 매개변수 사용할 수 없음.

String은 Object로 부터 상속받기 때문에 호출 가능, 리턴타입은Object

String이 T로 인식하여 리턴타입 String

리턴타입이 Object