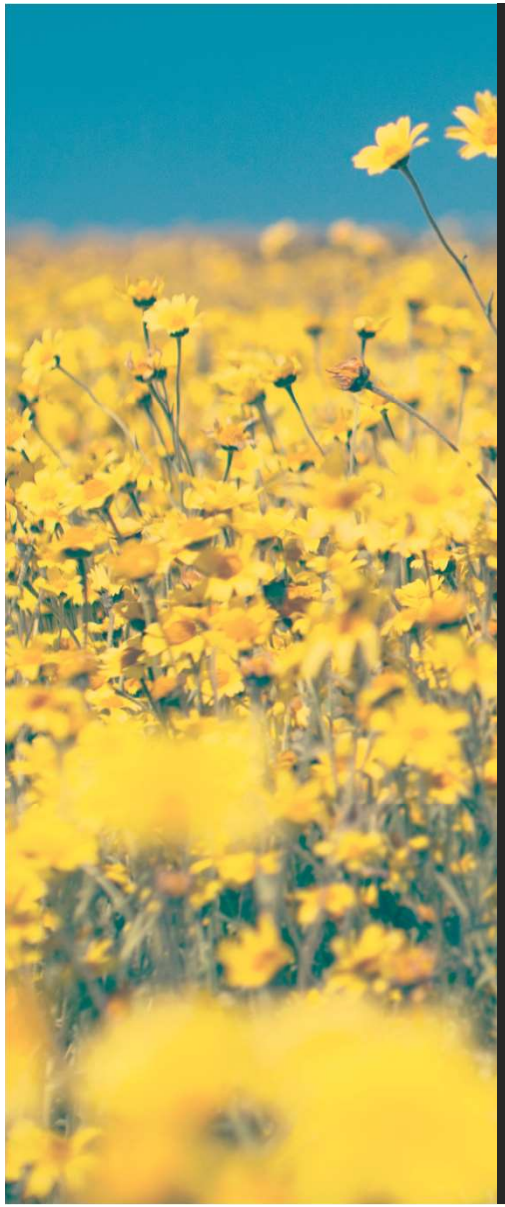


JAVA

GU, DA HAE


제 네 릭
컬 렉 션
JAVA
제 네 릭

JAVA

제네릭

Generics

제네릭은 **자료형을 일반화하는 방법**이다. 클래스의 기능(실행 코드)은 이미 결정되어 있으나 자료형(재료)은 정하지 않았다는 특징을 지니고 있다.

```
class 식별자<식별자>{
    ...
}
```

```
접근지시자 <식별자> 반환형 식별자(){
    ...
}
```

제네릭은 선언이 포함된 클래스는 인스턴스를 생성할 때 사용할 자료형 정보를 선택해야 한다.

제네릭에서 사용할 수 있는 자료형은 클래스 뿐이다. 기본 자료형은 사용할 수 없다.

JAVA

예제

```java
class Test<T>{

// 제네릭, 자료형 일반화(추상화), T : Type의 약자

    private T a;

    private T b;


    public Test(T a, T b) {

        this.a = a; this.b = b;

    }

    public void showTest() {

        System.out.println(a);

        System.out.println(b);

    }

}
```

```java
public class Ex01 {

    public static void main(String[] args) {

        Test<Integer> t1 = new Test(10, 20);

        Test<Float> t2 = new Test(3.1f, 7.8f);

        Test<String> t3 = new Test("Hi", "World");


        t1.showTest();

        t2.showTest();

        t3.showTest();

    }

}
```

클래스를 제네릭으로

JAVA

예제

```
class Book{
    private String title;
    public Book(String title) {this.title = title;}
    public void showTitle() {
        System.out.println("책 제목 : " + title);
    }
}


class Cloth{
    private String color;
    public Cloth(String color) {this.color = color;}
    public void showColor() {
        System.out.println("옷 색 : " + color);
    }
}


class Box<T>{
    private T t;
    public void putIn(T t) {this.t= t;}
    public T pull() {return t;}
}
```

```
public class Ex01 {
    public static void main(String[] args) {
        Box<Book> box1 = new Box<Book>();
        Box<Cloth> box2 = new Box<Cloth>();
        box1.putIn(new Book("검은 집"));
        box2.putIn(new Cloth("red"));

        Book book = box1.pull();
        Cloth cloth = box2.pull();

        book.showTitle();
        cloth.showColor();
    }
}
```

일반 클래스 내의 특정 함수만 제네릭으로

JAVA

예제

```java
class Book{
    private String title;
    public Book(String title) {
        this.title = title;
    }
    public String toString() {
        return "책 제목 : " + title;
    }
}

class Box{
    public <T> void show(T x) {
        System.out.println(x);
    }
}
```

```java
public class Ex01 {
    public static void main(String[] args) {
        Box t = new Box();

        t.<Integer>show(10);
        t.<Book>show(new Book("나무인간"));
    }
}
```

여러가지 타입을 사용하는 제네릭 메소드

JAVA

예제

```
class Book{
    private String title;
    public Book(String title) {this.title = title;}
    public String toString() {return "책 제목 : " + title;}

}


class Cloth{
    private String color;
    public Cloth(String color) {this.color = color;}
    public String toString() {return "옷 색 : " + color;}

}


class Box{
    public <T1, T2> void show(T1 x, T2 y) {
        System.out.println(x);
        System.out.println(y);
    }
}
```

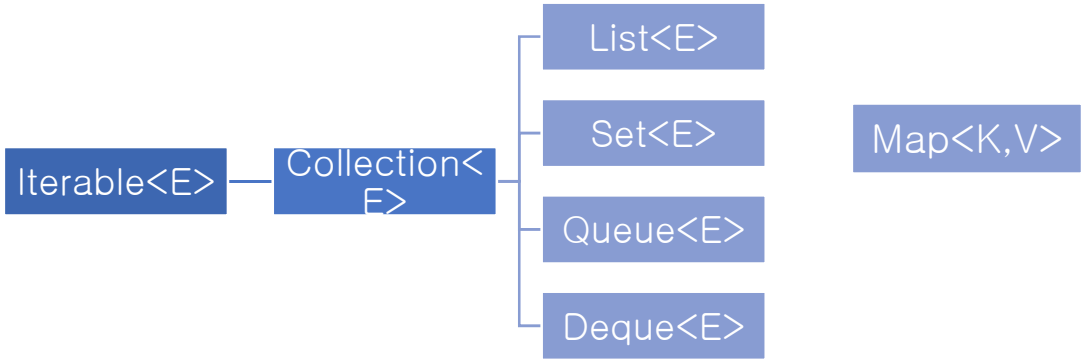
```
public class Ex01 {

    public static void main(String[] args) {

        Box t = new Box();



        t.<Integer, Integer>show(10, 20);

        t.<Book, Cloth>show(new Book("나무인간"), new
Cloth("Green"));

    }

}
```

JAVA

컬렉션

Collection

컬렉션은 데이터 저장, 관리와 관련된 알고리즘을 모아 놓은 프레임워크(Framework)다.



인터페이스	설명	구현 클래스
List<E>	순서 있는 데이터 목록, 데이터 중복 가능	ArrayList, LinkedList, Stack, Vector
Queue<E>	List의 한 종류	Deque, BlockingDeque, BlockingQueue, TransferQueue
Set<E>	순서 없는 데이터 집합, 데이터 중복 불가능	HashSet, TreeSet
Map<K, V>	키·값 쌍으로 이루어지고 순서 없는 데이터 집합, 키는 중복 불가능, 값은 중복 가능	HashMap, Treemap, Hashtable, Properties

JAVA

Collection 인터페이스

Collection 인터페이스는 컬렉션을 다루는데 가장 기본적인 동작들을 정의하고, 메소드로 제공한다.

메소드	설명
boolean add(E e)	해당 컬렉션(collection)에 전달된 요소를 추가함. (선택적 기능)
void clear()	해당 컬렉션의 모든 요소를 제거함. (선택적 기능)
boolean contains(Object o)	해당 컬렉션이 전달된 객체를 포함하고 있는지를 확인함.
boolean equals(Object o)	해당 컬렉션과 전달된 객체가 같은지를 확인함.
boolean isEmpty()	해당 컬렉션이 비어있는지를 확인함.
Iterator<E> iterator()	해당 컬렉션의 반복자(iterator)를 반환함.
boolean remove(Object o)	해당 컬렉션에서 전달된 객체를 제거함. (선택적 기능)
int size()	해당 컬렉션의 요소의 총 개수를 반환함.
Object[] toArray()	해당 컬렉션의 모든 요소를 Object 타입의 배열로 반환함.

JAVA

List 인터페이스

List 인터페이스에서 제공하는 주요 메소드는 다음과 같습니다.

메소드	설명
boolean add(E e)	해당 리스트(list)에 전달된 요소를 추가함. (선택적 기능)
void add(int index, E e)	해당 리스트의 특정 위치에 전달된 요소를 추가함. (선택적 기능)
void clear()	해당 리스트의 모든 요소를 제거함. (선택적 기능)
boolean contains(Object o)	해당 리스트가 전달된 객체를 포함하고 있는지를 확인함.
boolean equals(Object o)	해당 리스트와 전달된 객체가 같은지를 확인함.
E get(int index)	해당 리스트의 특정 위치에 존재하는 요소를 반환함.
boolean isEmpty()	해당 리스트가 비어있는지를 확인함.
Iterator<E> iterator()	해당 리스트의 반복자(iterator)를 반환함.
boolean remove(Object o)	해당 리스트에서 전달된 객체를 제거함. (선택적 기능)
boolean remove(int index)	해당 리스트의 특정 위치에 존재하는 요소를 제거함. (선택적 기능)
E set(int index, E e)	해당 리스트의 특정 위치에 존재하는 요소를 전달받은 객체로 대체함. (선택적 기능)
int size()	해당 리스트의 요소의 총 개수를 반환함.
Object[] toArray()	해당 리스트의 모든 요소를 Object 타입의 배열로 반환함.

JAVA

예제

```java
import java.util.ArrayList;


public class Ex01 {

    public static void main(String[] args) {

        ArrayList<Integer> list = new ArrayList<Integer>();

         // add() 메소드를 이용한 요소의 저장
        list.add(new Integer(1));
        list.add(new Integer(2));
        list.add(new Integer(3));
        list.add(new Integer(4));

         // for 문과 get() 메소드를 이용한 요소의 출력
        for(int i = 0; i < list.size(); i++)
            System.out.println(list.get(i)); // i번째 요소 참조

         // remove() 메소드를 이용한 요소의 제거
        for(int i = 0, size = list.size(); i < size; i++)
            list.remove(0); // 모든 요소 삭제

        System.out.println("요소 개수 : " + list.size());

    }
}
```

JAVA

예제

```java
import java.util.*;

public class Ex01 {
    public static void main(String[] args) {
        LinkedList<String> lnkList = new LinkedList<String>();

        // add() 메소드를 이용한 요소의 저장
        lnkList.add("넷");
        lnkList.add("둘");
        lnkList.add("셋");
        lnkList.add("하나");

        // for 문과 get() 메소드를 이용한 요소의 출력
        for (int i = 0; i < lnkList.size(); i++) {
            System.out.print(lnkList.get(i) + " ");
        }

        // remove() 메소드를 이용한 요소의 제거
        lnkList.remove(1);

        // Enhanced for 문과 get() 메소드를 이용한 요소의 출력
        for (String e : lnkList) {
            System.out.print(e + " ");
        }

        // set() 메소드를 이용한 요소의 변경
        lnkList.set(2, "둘");

        for (String e : lnkList) {
            System.out.print(e + " ");
        }

        // size() 메소드를 이용한 요소의 총 개수
        System.out.println("리스트의 크기 : " + lnkList.size());
    }
}
```

JAVA

Stack 클래스

Stack 클래스는 스택 메모리 구조를 표현하기 위해, Vector 클래스의 메소드를 5개만 상속받아 사용한다.

메소드	설명
boolean empty()	해당 스택이 비어 있으면 true를, 비어 있지 않으면 false를 반환함.
E peek()	해당 스택의 제일 상단에 있는(제일 마지막으로 저장된) 요소를 반환함.
E pop()	해당 스택의 제일 상단에 있는(제일 마지막으로 저장된) 요소를 반환하고, 해당 요소를 스택에서 제거함.
E push(E item)	해당 스택의 제일 상단에 전달된 요소를 삽입함.
int search(Object o)	해당 스택에서 전달된 객체가 존재하는 위치의 인덱스를 반환함. 이때 인덱스는 제일 상단에 있는(제일 마지막으로 저장된) 요소의 위치부터 0이 아닌 1부터 시작함.

JAVA

예제

```java
import java.util.*;


public class Ex01 {

    public static void main(String[] args) {

        Stack<Integer> st = new Stack<Integer>(); // 스택 생성


        //Deque<Integer> st = new ArrayDeque<Integer>();

        // 더 빠른 스택이 필요하다면 Deque를 사용

        // push() 메소드를 이용한 요소의 저장


        st.push(4);

        st.push(2);

        st.push(3);

        st.push(1);


        // peek() 메소드를 이용한 요소의 반환

        System.out.println(st.peek());

        System.out.println(st);


        // pop() 메소드를 이용한 요소의 반환 및 제거

        System.out.println(st.pop());

        System.out.println(st);


        // search() 메소드를 이용한 요소의 위치 검색

        System.out.println(st.search(4));

        System.out.println(st.search(3));

    }

}
```

JAVA

Queue 인터페이스

Queue 인터페이스는 큐 메모리 구조를 표현하기 위해, 다음과 같은 Collection 인터페이스 메소드만을 상속받아 사용한다.

메소드	설명
boolean add(E e)	해당 큐의 맨 뒤에 전달된 요소를 삽입함. 만약 삽입에 성공하면 true를 반환하고, 큐에 여유 공간이 없어 삽입에 실패하면 IllegalStateException을 발생시킴.
E element()	해당 큐의 맨 앞에 있는(제일 먼저 저장된) 요소를 반환함.
boolean offer(E e)	해당 큐의 맨 뒤에 전달된 요소를 삽입함.
E peek()	해당 큐의 맨 앞에 있는(제일 먼저 저장된) 요소를 반환함. 만약 큐가 비어있으면 null을 반환함.
E poll()	해당 큐의 맨 앞에 있는(제일 먼저 저장된) 요소를 반환하고, 해당 요소를 큐에서 제거함. 만약 큐가 비어있으면 null을 반환함.

JAVA

예제

```java
import java.util.*;


public class Ex01 {

    public static void main(String[] args) {

        LinkedList<String> qu = new LinkedList<String>();

        // 큐의 생성


        //Deque<String> qu = new ArrayDeque<String>();

        // 더 빠른 큐가 필요하다면 ArrayDeque 사용


        // add() 메소드를 이용한 요소의 저장

        qu.add("넷");

        qu.add("둘");

        qu.add("셋");

        qu.add("하나");


        // peek() 메소드를 이용한 요소의 반환

        System.out.println(qu.peek());

        System.out.println(qu);


        // poll() 메소드를 이용한 요소의 반환 및 제거

        System.out.println(qu.poll());

        System.out.println(qu);


        // remove() 메소드를 이용한 요소의 제거

        qu.remove("하나");

    }

}
```

★HashSet★
키워드 - - > Hash 함수 - -> 인덱스 번호
smith [2] 010-2222-2222
☆TreeSet☆

JAVA

Set 인터페이스

Set 인터페이스를 구현한 모든 Set 컬렉션 클래스는 다음과 같은 특징을 갖는다.

1. 요소의 저장 순서를 유지하지 않는다.
2. 같은 요소의 중복 저장을 허용하지 않는다.

[HashSet]

HashSet 클래스는 Set 인터페이스를 구현하므로, 요소를 순서에 상관없이 저장하고 중복된 값은 저장하지 않는다.만약 요소의 저장 순서를 유지해야 한다면 JDK 1.4부터 제공하는 LinkedHashSet 클래스를 사용한다.

[TreeSet]

TreeSet 클래스는 데이터가 정렬된 상태로 저장되는 이진 검색 트리(binary search tree)의 형태로 요소를 저장한다. TreeSet 클래스는 Set 인터페이스를 구현하므로, 요소를 순서에 상관없이 저장하고 중복된 값은 저장하지 않는다.

JAVA

Set 인터페이스

Set 인터페이스는 Collection 인터페이스를 상속받으므로, Collection 인터페이스에서 정의한 메소드도 모두 사용할 수 있다.

메소드	설명
boolean add(E e)	해당 집합(set)에 전달된 요소를 추가함. (선택적 기능)
void clear()	해당 집합의 모든 요소를 제거함. (선택적 기능)
boolean contains(Object o)	해당 집합이 전달된 객체를 포함하고 있는지를 확인함.
boolean equals(Object o)	해당 집합과 전달된 객체가 같은지를 확인함.
boolean isEmpty()	해당 집합이 비어있는지를 확인함.

JAVA

예제

```java
import java.util.*;

public class Ex01 {
    public static void main(String[] args) {
        HashSet<String> hs01 = new HashSet<String>();
        HashSet<String> hs02 = new HashSet<String>();

        // add() 메소드를 이용한 요소의 저장
        hs01.add("홍길동");
        hs01.add("이순신");
        System.out.println(hs01.add("임꺽정"));
        System.out.println(hs01.add("임꺽정"));

        // 중복된 요소의 저장


        // Enhanced for 문과 get() 메소드를 이용한 요소의 출력
        for (String e : hs01) {
            System.out.print(e + " ");
        }

        // add() 메소드를 이용한 요소의 저장
        hs02.add("임꺽정");
        hs02.add("홍길동");
        hs02.add("이순신");


        // iterator() 메소드를 이용한 요소의 출력
        Iterator<String> iter02 = hs02.iterator();


        while (iter02.hasNext()) {
            System.out.print(iter02.next() + " ");
        }

        // size() 메소드를 이용한 요소의 총 개수
        System.out.println("집합의 크기 : " + hs02.size());

    }

}
```

JAVA

예제

```java
import java.util.*;

public class Ex01 {
    public static void main(String[] args) {
        TreeSet<Integer> ts = new TreeSet<Integer>();

        // add() 메소드를 이용한 요소의 저장
        ts.add(30);
        ts.add(40);
        ts.add(20);
        ts.add(10);

        // Enhanced for 문과 get() 메소드를 이용한 요소의 출력
        for (int e : ts) {
            System.out.print(e + " ");
        }

        // remove() 메소드를 이용한 요소의 제거
        ts.remove(40);

        // iterator() 메소드를 이용한 요소의 출력
        Iterator<Integer> iter = ts.iterator();
        while (iter.hasNext()) {
            System.out.print(iter.next() + " ");
        }

        // size() 메소드를 이용한 요소의 총 개수
        System.out.println("이진 검색 트리의 크기 : " +
ts.size());

        // subSet() 메소드를 이용한 부분 집합의 출력
        System.out.println(ts.subSet(10, 20));
        System.out.println(ts.subSet(10, true, 20, true));

    }

}
```

JAVA

Map 인터페이스

Map 인터페이스를 구현한 모든 Map 컬렉션 클래스는 다음과 같은 특징을 갖는다.

1. 요소의 저장 순서를 유지하지 않는다.
2. 키는 중복을 허용하지 않지만, 값의 중복은 허용한다.

[HashMap]

HashMap 클래스는 Map 컬렉션 클래스에서 가장 많이 사용되는 클래스 중 하나다. JDK 1.2부터 제공된 HashMap 클래스는 해시 알고리즘(hash algorithm)을 사용하여 검색 속도가 매우 빠르다.

[TreeMap]

TreeMap 클래스는 키와 값을 한 쌍으로 하는 데이터를 이진 검색 트리(binary search tree)의 형태로 저장한다. 이진 검색 트리는 데이터를 추가하거나 제거하는 등의 기본 동작 시간이 빠르다.

JAVA

예제

```java
import java.util.*;

public class Ex01 {
    public static void main(String[] args) {
        HashMap<String, Integer> hm = new HashMap<String, Integer>();

        // put() 메소드를 이용한 요소의 저장
        hm.put("삼십", 30);
        hm.put("십", 10);
        hm.put("사십", 40);
        hm.put("이십", 20);

        // Enhanced for 문과 get() 메소드를 이용한 요소의 출력
        System.out.println("맵에 저장된 키들의 집합 : " + hm.keySet());
        for (String key : hm.keySet()) {
            System.out.println(String.format("키 : %s, 값 : %s", key, hm.get(key)));
        }

        // remove() 메소드를 이용한 요소의 제거
        hm.remove("사십");

        // iterator() 메소드와 get() 메소드를 이용한 요소의 출력
        Iterator<String> keys = hm.keySet().iterator();
        while (keys.hasNext()) {
            String key = keys.next();
            System.out.println(String.format("키 : %s, 값 : %s", key, hm.get(key)));
        }

        // replace() 메소드를 이용한 요소의 수정
        hm.replace("이십", 200);

        for (String key : hm.keySet()) {
            System.out.println(String.format("키 : %s, 값 : %s", key, hm.get(key)));
        }

        // size() 메소드를 이용한 요소의 총 개수
        System.out.println("맵의 크기 : " + hm.size());

    }
}
```

JAVA

예제

```java
import java.util.*;

public class Ex01 {
    public static void main(String[] args) {
        TreeMap<Integer, String> tm = new TreeMap<Integer,
String>();

        // put() 메소드를 이용한 요소의 저장
        tm.put(30, "삼십");
        tm.put(10, "십");
        tm.put(40, "사십");
        tm.put(20, "이십");

        // Enhanced for 문과 get() 메소드를 이용한 요소의 출력
        System.out.println("맵에 저장된 키들의 집합 : " +
tm.keySet());
        for (Integer key : tm.keySet()) {
            System.out.println(String.format("키 : %s,
값 : %s", key, tm.get(key)));
        }

        // remove() 메소드를 이용한 요소의 제거
        tm.remove(40);

        // iterator() 메소드와 get() 메소드를 이용한 요소의 출력
        Iterator<Integer> keys = tm.keySet().iterator();
        while (keys.hasNext()) {
            Integer key = keys.next();
            System.out.println(String.format("키 : %s,
값 : %s", key, tm.get(key)));
        }

        // replace() 메소드를 이용한 요소의 수정
        tm.replace(20, "twenty");


        for (Integer key : tm.keySet()) {
            System.out.println(String.format("키 : %s,
값 : %s", key, tm.get(key)));
        }

        // size() 메소드를 이용한 요소의 총 개수
        System.out.println("맵의 크기 : " + tm.size());

    }

}
```