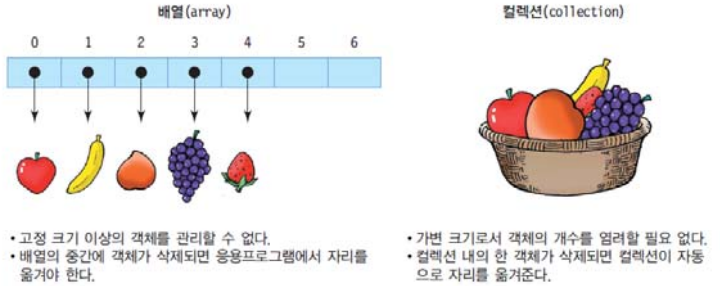
Collections, Generic

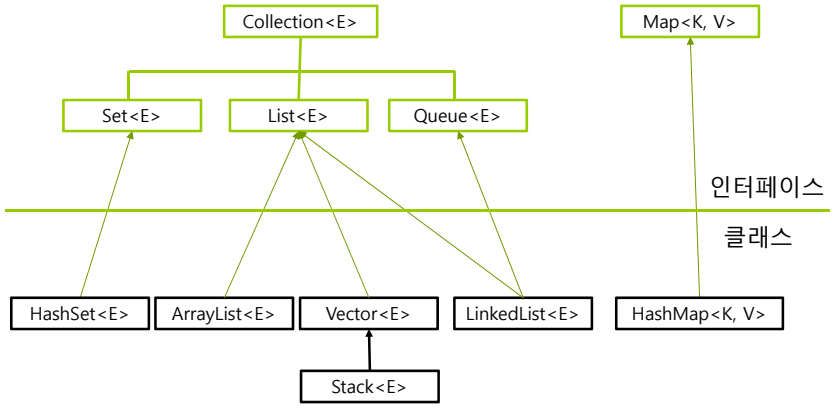
514760-1
2017년 가을학기
11/6/2017
박경신

컬렉션(collection)의 개념

- 컬렉션
  - 요소(element)라고 불리는 가변 개수의 객체들의 저장소
    - 객체들의 컨테이너라고도 불림
    - 요소의 개수에 따라 크기 자동 조절
    - 요소의 삽입, 삭제에 따른 요소의 위치 자동 이동
  - 고정 크기의 배열을 다루는 어려움 해소
  - 다양한 객체들의 삽입, 삭제, 검색 등의 관리 용이



컬렉션을 위한 인터페이스와 클래스

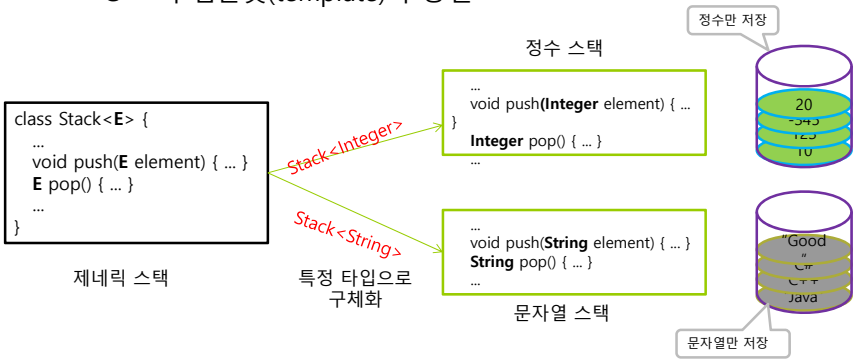


컬렉션과 제네릭

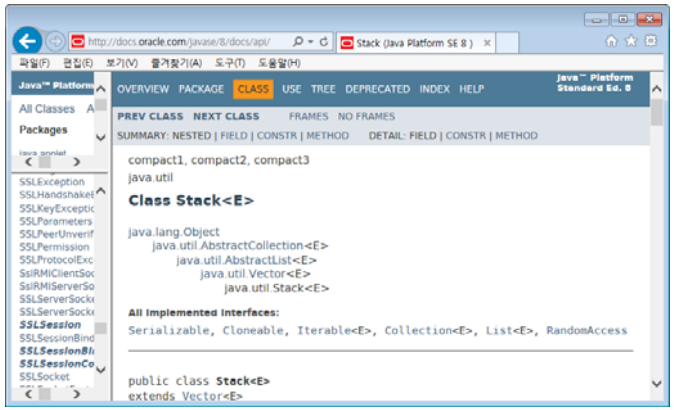
- 컬렉션은 제네릭(generics) 기법으로 구현됨
- 컬렉션의 요소는 객체만 가능
  - 기본적으로 int, char, double 등의 기본 타입 사용 불가
    - JDK 1.5부터 자동 박싱/언박싱 기능으로 기본 타입 사용 가능
- 제네릭
  - 특정 타입만 다루지 않고, 여러 종류의 타입으로 변신할 수 있도록 클래스나 메소드를 일반화시키는 기법
    - <E>, <K>, <V> : 타입 매개 변수
      - 요소 타입을 일반화한 타입
  - 제네릭 클래스 사례
    - 제네릭 벡터 : Vector<E>
    - E에 특정 타입으로 구체화
    - 정수만 다루는 벡터 Vector<Integer>
    - 문자열만 다루는 벡터 Vector<String>

제네릭의 기본 개념

- JDK 1.5에서 도입(2004년 기점)
- 모든 종류의 데이터 타입을 다룰 수 있도록 일반화된 타입 매개 변수로 클래스나 메소드를 작성하는 기법
  - C++의 템플릿(template)과 동일



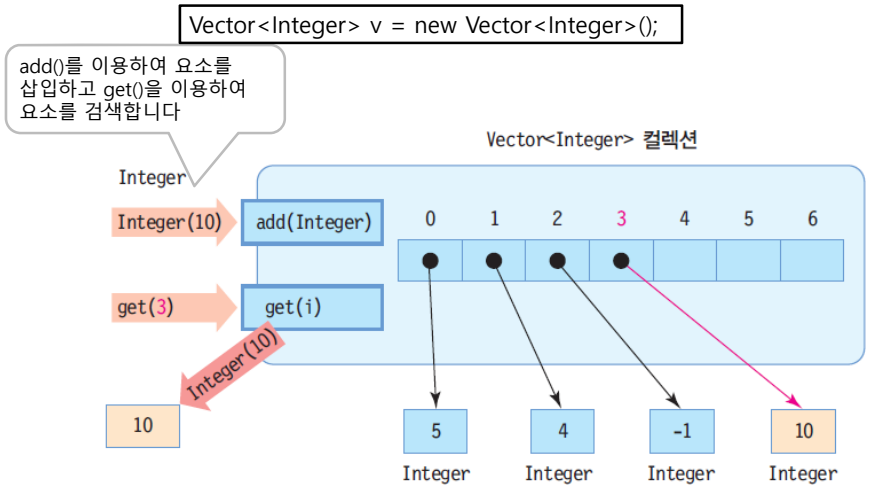
제네릭 Stack<E> 클래스의 JDK 매뉴얼



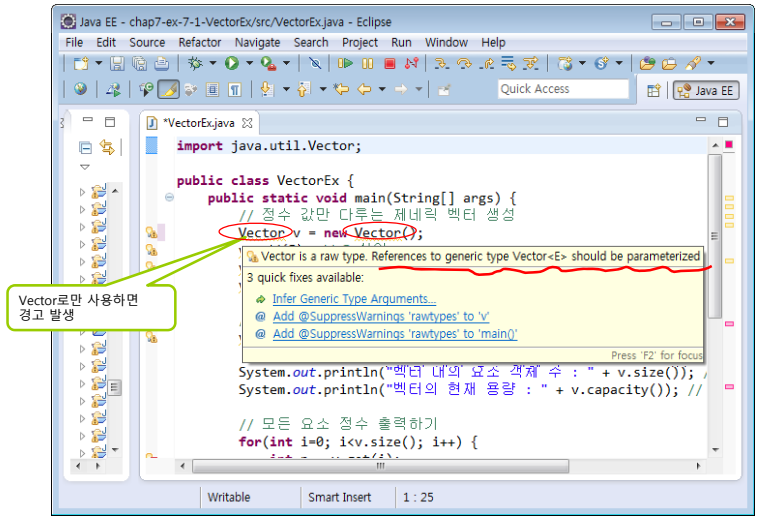
Vector<E>

- Vector<E>의 특성
  - java.util.Vector
    - <E>에서 E 대신 요소로 사용할 특정 타입으로 구체화
  - 여러 객체들을 삽입, 삭제, 검색하는 컨테이너 클래스
    - 배열의 길이 제한 극복
    - 원소의 개수가 넘쳐나면 자동으로 길이 조절
  - Vector에 삽입 가능한 것
    - 객체, null
    - 기본 타입은 박싱/언박싱으로 Wrapper 객체로 만들어 저장
  - Vector에 객체 삽입
    - 벡터의 맨 뒤에 객체 추가
    - 벡터 중간에 객체 삽입
  - Vector에서 객체 삭제
    - 임의의 위치에 있는 객체 삭제 가능 : 객체 삭제 후 자동 자리 이동

Vector<Integer> 컬렉션 내부 구성



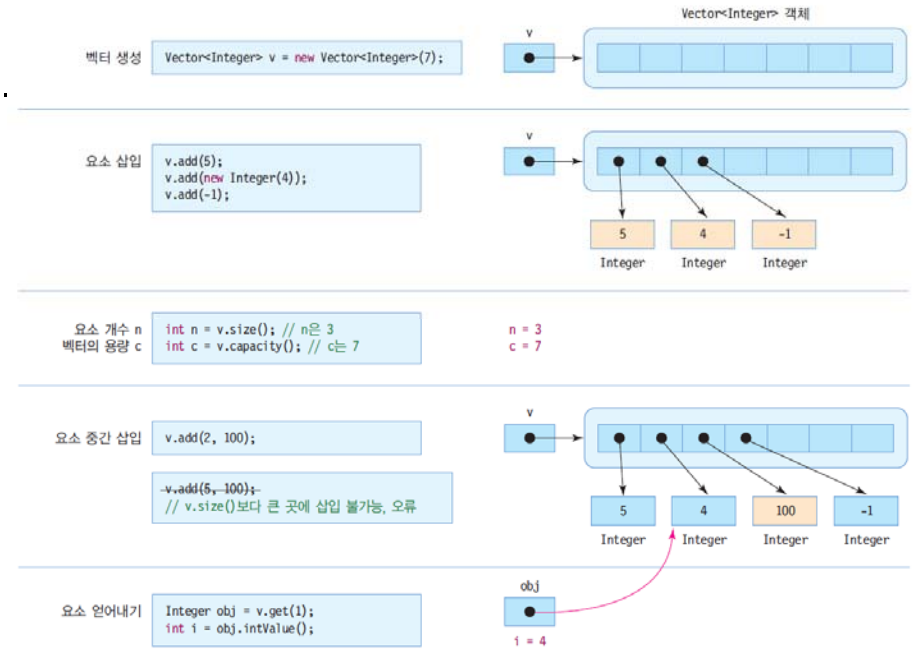
타입 매개 변수 사용하지 않는 경우 경고 발생

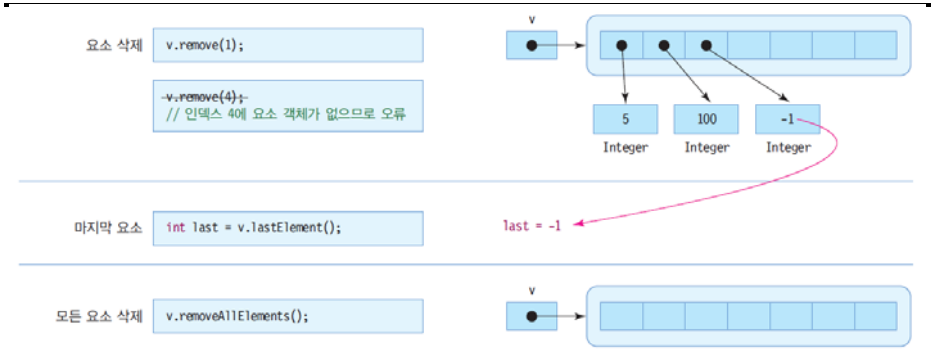


Vector<Integer>나 Vector<String> 등 타입 매개 변수를 사용하여야 함

Vector<E> 클래스의 주요 메소드

메소드	설명
boolean add(E element)	벡터의 맨 뒤에 element 추가
void add(int index, E element)	인덱스 index에 element를 삽입
int capacity()	벡터의 현재 용량 리턴
boolean addAll(Collection<? extends E> c)	컬렉션 c의 모든 요소를 벡터의 맨 뒤에 추가
void clear()	벡터의 모든 요소 삭제
boolean contains(Object o)	벡터가 지정된 객체 o를 포함하고 있으면 true 리턴
E elementAt(int index)	인덱스 index의 요소 리턴
E get(int index)	인덱스 index의 요소 리턴
int indexOf(Object o)	o와 같은 첫 번째 요소의 인덱스 리턴. 없으면 -1 리턴
boolean isEmpty()	벡터가 비어 있으면 true 리턴
E remove(int index)	인덱스 index의 요소 삭제
boolean remove(Object o)	객체 o와 같은 첫 번째 요소를 벡터에서 삭제
void removeAllElements()	벡터의 모든 요소를 삭제하고 크기를 0으로 만듦
int size()	벡터가 포함하는 요소의 개수 리턴
Object[] toArray()	벡터의 모든 요소를 포함하는 배열 리턴





컬렉션과 자동 박싱/언박싱

- JDK 1.5 이전
  - 기본 타입 데이터를 Wrapper 클래스를 이용하여 객체로 만들어 사용

```
Vector<Integer> v = new Vector<Integer>();
v.add(new Integer(4));
```

  - 컬렉션으로부터 요소를 얻어올 때, Wrapper 클래스로 캐스팅 필요

```
Integer n = (Integer)v.get(0);
int k = n.intValue(); // k = 4
```

- JDK 1.5부터
  - 자동 박싱/언박싱이 작동하여 기본 타입 값 사용 가능

```
Vector<Integer> v = new Vector<Integer> ();
v.add(4); // 4 → new Integer(4)로 자동 박싱
int k = v.get(0); // Integer 타입이 int 타입으로 자동 언박싱, k = 4
```

    - 제네릭의 타입 매개 변수를 기본 타입으로 구체화할 수는 없음

```
Vector<int> v = new Vector<int> (); // 오류
```

예제 : 정수 값만 다루는 Vector<Integer>

정수 값만 다루는 제네릭 벡터를 생성하고 활용하는 사례를 보인다.
다음 코드에 대한 결과는 무엇인가?

```
import java.util.Vector;

public class VectorEx {
  public static void main(String[] args) {
    // 정수 값만 다루는 제네릭 벡터 생성
    Vector<Integer> v = new Vector<Integer>();

    v.add(5); // 5 삽입
    v.add(4); // 4 삽입
    v.add(-1); // -1 삽입

    // 벡터 중간에 삽입하기
    v.add(2, 100); // 4와 -1 사이에 정수 100 삽입
    System.out.println("벡터 내의 요소 객체 수 : " + v.size());
    System.out.println("벡터의 현재 용량 : " + v.capacity());
```

```
  // 모든 요소 정수 출력하기
    for(int i=0; i<v.size(); i++) {
      int n = v.get(i);
      System.out.println(n);
    }

    // 벡터 속의 모든 정수 더하기
    int sum = 0;
    for(int i=0; i<v.size(); i++) {
      int n = v.elementAt(i);
      sum += n;
    }
    System.out.println("벡터에 있는 정수 합 : " + sum);
  }
}
```

```
5
4
100
-1
벡터에 있는 정수 합 : 108
```

예제: Point 클래스의 객체들만 저장하는 벡터 만들기

(x, y) 한 점을 추상화한 Point 클래스를 만들고
Point 클래스의 객체만 저장하는 벡터를 작성하라.

```
import java.util.Vector;

class Point {
  private int x, y;
  public Point(int x, int y) {
    this.x = x;
    this.y = y;
  }

  public String toString() {
    return "(" + x + "," + y + ")";
  }
}
```

예제: Point 클래스의 객체들만 저장하는 벡터 만들기

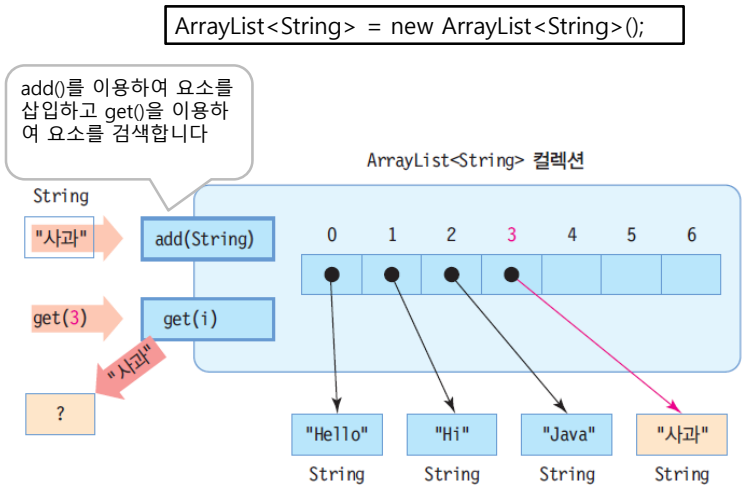
```
public class PointVectorEx {
  public static void main(String[] args) {
    // Point 객체를 요소로만 가지는 벡터 생성
    Vector<Point> v = new Vector<Point>();

    // 3 개의 Point 객체 삽입
    v.add(new Point(2, 3));
    v.add(new Point(-5, 20));
    v.add(new Point(30, -8));

    // 벡터에 있는 Point 객체 모두 검색하여 출력
    for(int i=0; i<v.size(); i++) {
      Point p = v.get(i); // 벡터에서 i 번째 Point 객체 얻어내기
      System.out.println(p); // p.toString()을 이용하여 객체 p 출력
    }
  }
}
```

```
(2,3)
(-5,20)
(30,-8)
```

ArrayList<E>

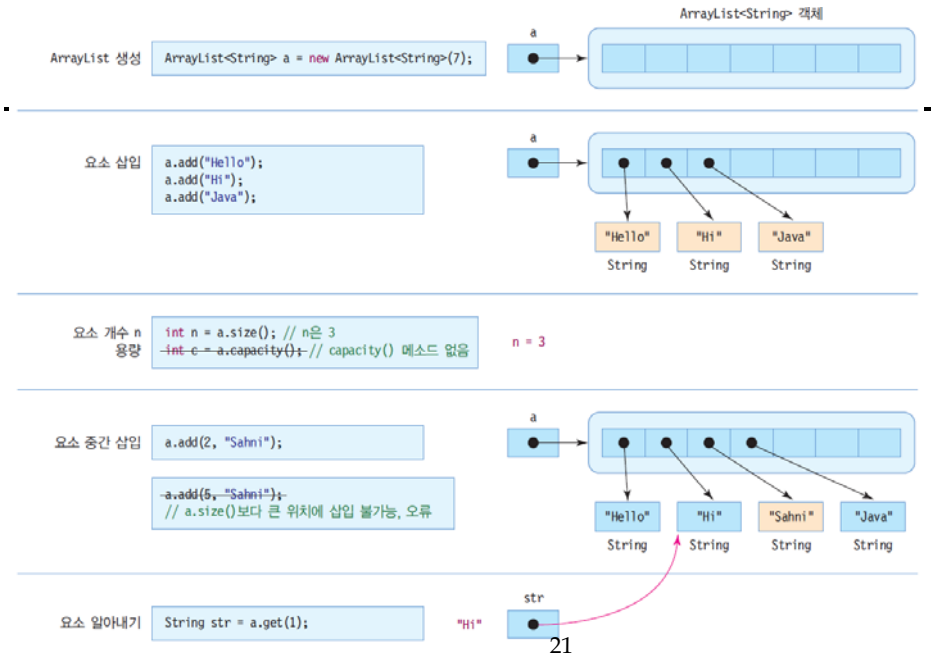
- ArrayList<E>의 특성
  - java.util.ArrayList, 가변 크기 배열을 구현한 클래스
    - <E>에서 E 대신 요소로 사용할 특정 타입으로 구체화
  - ArrayList에 삽입 가능한 것
    - 객체, null
    - **기본 타입은 박싱/언박싱으로 Wrapper 객체로 만들어 저장**
  - ArrayList에 객체 삽입/삭제
    - 리스트의 맨 뒤에 객체 추가
    - 리스트의 중간에 객체 삽입
    - 임의의 위치에 있는 객체 삭제 가능
  - 벡터와 달리 스레드 동기화 기능 없음
    - 다수 스레드가 동시에 ArrayList에 접근할 때 동기화되지 않음
    - 개발자가 스레드 동기화 코드 작성

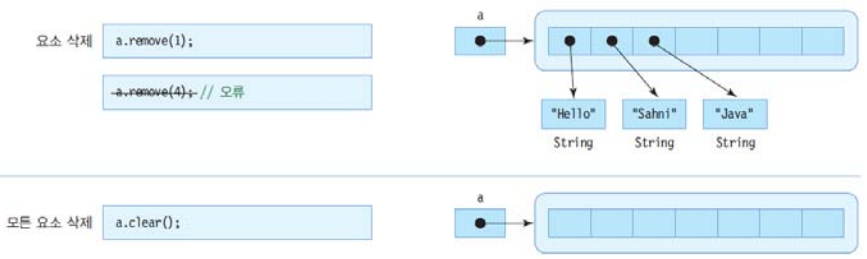
ArrayList<String> 컬렉션의 내부 구성



ArrayList<E> 클래스의 주요 메소드

메소드	설명
boolean add(E element)	ArrayList의 맨 뒤에 element 추가
void add(int index, E element)	인덱스 index에 지정된 element 삽입
boolean addAll(Collection<? extends E> c)	컬렉션 c의 모든 요소를 ArrayList의 맨 뒤에 추가
void clear()	ArrayList의 모든 요소 삭제
boolean contains(Object o)	ArrayList가 지정된 객체를 포함하고 있으면 true 리턴
E elementAt(int index)	index 인덱스의 요소 리턴
E get(int index)	index 인덱스의 요소 리턴
int indexOf(Object o)	o와 같은 첫 번째 요소의 인덱스 리턴. 없으면 -1 리턴
boolean isEmpty()	ArrayList가 비어 있으면 true 리턴
E remove(int index)	index 인덱스의 요소 삭제
boolean remove(Object o)	o와 같은 첫 번째 요소를 ArrayList에서 삭제
int size()	ArrayList가 포함하는 요소의 개수 리턴
Object[] toArray()	ArrayList의 모든 요소를 포함하는 배열 리턴





예제 : ArrayList에 문자열을 달기

키보드로 문자열을 입력 받아 ArrayList에 삽입하고 가장 긴 이름을 출력하라.

```
import java.util.*;

public class ArrayListEx {
  public static void main(String[] args) {
    // 문자열만 삽입가능한 ArrayList 컬렉션 생성
    ArrayList<String> a = new ArrayList<String>();

    // 키보드로부터 4개의 이름 입력받아 ArrayList에 삽입
    Scanner scanner = new Scanner(System.in);
    for(int i=0; i<4; i++) {
      System.out.print("이름을 입력하세요>>");
      String s = scanner.next(); // 키보드로부터 이름 입력
      a.add(s); // ArrayList 컬렉션에 삽입
    }
```

예제: ArrayList에 문자열을 달기

```
    // ArrayList에 들어 있는 모든 이름 출력
    for(int i=0; i<a.size(); i++) {
      // ArrayList의 i 번째 문자열 얻어오기
      String name = a.get(i);
      System.out.print(name + " ");
    }
    // 가장 긴 이름 출력
    int longestIndex = 0;
    for(int i=1; i<a.size(); i++) {
      if(a.get(longestIndex).length() < a.get(i).length())
        longestIndex = i;
    }
    System.out.println("₩n가장 긴 이름은 : " + a.get(longestIndex));

  }
}
```

```
이름을 입력하세요>>Mike
이름을 입력하세요>>Jane
이름을 입력하세요>>Ashley
이름을 입력하세요>>Helen
Mike Jane Ashley Helen
가장 긴 이름은 : Ashley
```

컬렉션의 순차 검색을 위한 Iterator

- Iterator<E> 인터페이스
  - Vector<E>, ArrayList<E>, LinkedList<E>가 상속받는 인터페이스
    - 리스트 구조의 컬렉션에서 요소의 순차 검색을 위한 메소드 포함
  - Iterator<E> 인터페이스 메소드

메소드	설명
boolean hasNext()	다음 반복에서 사용될 요소가 있으면 true 리턴
E next()	다음 요소 리턴
void remove()	마지막으로 리턴된 요소 제거

  - iterator() 메소드
    - iterator()를 호출하면 Iterator 객체 반환
    - Iterator 객체를 이용하여 인덱스 없이 순차적 검색 가능

컬렉션의 순차 검색을 위한 Iterator

```
Vector<Integer> v = new Vector<Integer>();
Iterator<Integer> it = v.iterator();
while(it.hasNext()) { // 모든 요소 방문
        int n = it.next(); // 다음 요소 리턴
        …
}

또는

for (int n : v) {
    …
}
```

예제: Iterator를 이용하여 Vector의 모든 요소 출력하고 합 구하기

Vector<Integer>로부터 Iterator를 얻어내고 벡터의 모든 정수를 출력하고 합을 구하라

```
import java.util.*;

public class IteratorEx {
  public static void main(String[] args) {
    // 정수 값만 다루는 제네릭 벡터 생성
    Vector<Integer> v = new Vector<Integer>();
    v.add(5); // 5 삽입
    v.add(4); // 4 삽입
    v.add(-1); // -1 삽입
    v.add(2, 100); // 4와 -1 사이에 정수 100 삽입
    // Iterator를 이용한 모든 정수 출력하기
    Iterator<Integer> it = v.iterator(); // Iterator 객체 얻기
    while(it.hasNext()) {
      int n = it.next();
      System.out.println(n);
    }
```

```
for (int n : v) {

System.out.println(n);
}
```

예제 : Iterator를 이용하여 Vector의 모든 요소 출력하고 합 구하기

```
    // Iterator를 이용하여 모든 정수 더하기
    int sum = 0;
    it = v.iterator(); // Iterator 객체 얻기
    while(it.hasNext()) {
      int n = it.next();
      sum += n;
    }
    System.out.println("벡터에 있는 정수 합 : " + sum);
  }
}
}
```

```
for (int n : v) {
    sum += n;
}
```

```
5
4
100
-1
벡터에 있는 정수 합 : 108
```

Remove Objects from Collection while Iterating

- ArrayList는 remove(int index) 또는 remove(Object obj) 메소드를 제공함. 단 remove() 메소드는 ArrayList를 iterating하지 않은 경우에만 사용함.
- ArrayList에서 iterating하면서 remove() 해야할 경우, **Iterator**를 사용함.

```
ArrayList<String> list = new ArrayList<String>(Arrays.asList("a","b","c","d"));
for (int I = 0; i < list.size(); i++) {
    list.remove(i); // 원소가 삭제될 때 list 사이즈가 줄면서 다른 원소들의 index도 바뀜
}
for (String s : list) {
    list.remove(s); // ConcurrentModificationException 발생
}
Iterator<String> it = list.iterator();
while (it.hasNext()) {
    String s = it.next(); // Iterator의 next()가 remove()보다 먼저 호출되어야 함
    it.remove();
}
```

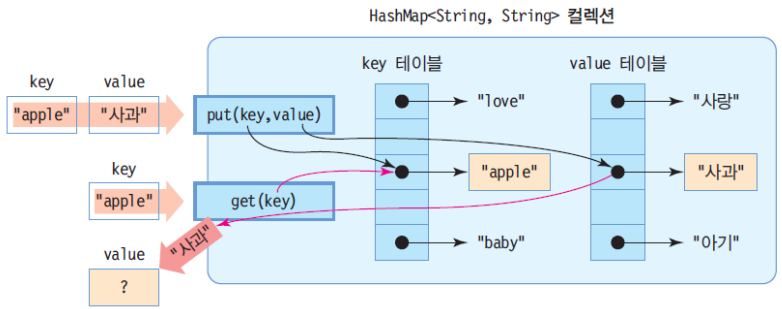
HashMap<K,V>

- HashMap<K,V>
  - 키(key)와 값(value)의 쌍으로 구성되는 요소를 다루는 컬렉션
    - java.util.HashMap
    - K는 키로 사용할 요소의 타입, V는 값으로 사용할 요소의 타입 지정
    - 키와 값이 한 쌍으로 삽입
    - 키는 해시맵에 삽입되는 위치 결정에 사용 **(키는 중복이 허용안됨 – 중복시 마지막 키로 대체됨)**
    - 값을 검색하기 위해서는 반드시 키 이용 **(값은 중복이 허용됨)**
  - 삽입 및 검색이 빠른 특징
    - 요소 삽입 : put() 메소드
    - 요소 검색 : get() 메소드
  - 예) HashMap<String, String> 생성, 요소 삽입, 요소 검색

```
HashMap<String, String> h = new HashMap<String, String>();
h.put("apple", "사과"); // "apple" 키와 "사과" 값의 쌍을 해시맵에 삽입
String kor = h.get("apple"); // "apple" 키로 값 검색. kor는 "사과"
```

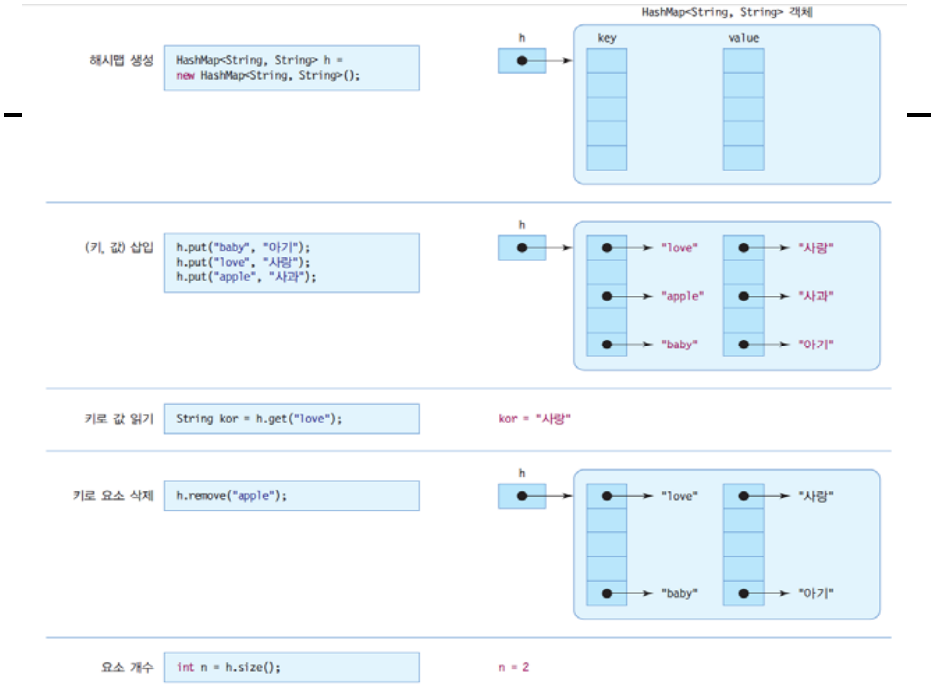
HashMap<String, String>의 내부 구성과 put(), get() 메소드

```
HashMap<String, String> map = new HashMap<String, String>();
```



HashMap<K,V>의 주요 메소드

메소드	설명
void clear()	HashMap의 모든 요소 삭제
boolean containsKey(Object key)	지정된 키(key)를 포함하고 있으면 true 리턴
boolean containsValue(Object value)	하나 이상의 키를 지정된 값(value)에 매핑시킬 수 있으면 true 리턴
V get(Object key)	지정된 키(key)에 매핑되는 값 리턴. 키에 매핑되는 어떤 값도 없으면 null 리턴
boolean isEmpty()	HashMap이 비어 있으면 true 리턴
Set<K> keySet()	HashMap에 있는 모든 키를 담은 Set<K> 컬렉션 리턴
V put(K key, V value)	key와 value를 매핑하여 HashMap에 저장
V remove(Object key)	지정된 키(key)와 이에 매핑된 값을 HashMap에서 삭제
int size()	HashMap에 포함된 요소의 개수 리턴



예제: HashMap을 이용하여 영어 단어와 한글 단어를 쌍으로 저장하고 검색하는 사례

영어 단어와 한글 단어를 쌍으로 HashMap에 저장하고
영어 단어로 한글 단어를 검색하는 프로그램을 작성하라.

```
import java.util.*;

public class HashMapDicEx {
  public static void main(String[] args) {
    // 영어 단어와 한글 단어의 쌍을 저장하는 HashMap 컬렉션 생성
    HashMap<String, String> dic = new HashMap<String, String>();
    // 3 개의 (key, value) 쌍을 dic에 저장
    dic.put("baby", "아기"); // "baby"는 key, "아기"은 value
    dic.put("love", "사랑");
    dic.put("apple", "사과");
```

예제 : HashMap을 이용하여 영어 단어와 한글 단어를 쌍으로 저장하고 검색하는 사례

```
    // dic 컬렉션에 들어 있는 모든 (key, value) 쌍 출력
    Set<String> keys = dic.keySet(); // key 문자열을 가진 Set 리턴
    Iterator<String> it = keys.iterator();
    while(it.hasNext()) {
      String key = it.next();
      String value = dic.get(key);
      System.out.println("(" + key + "," + value + ")");
    }
    // 영어 단어를 입력 받고 한글 단어 검색
    Scanner scanner = new Scanner(System.in);
    for(int i=0; i<3; i++) {
      System.out.print("찾고 싶은 단어는?");
      String eng = scanner.next();
      System.out.println(dic.get(eng));
    }
  }
}
```

예제: HashMap을 이용하여 영어 단어와 한글 단어를 쌍으로 저장하고 검색하는 사례

```
(love,사랑)
(apple,사과)
(baby,아기)
찾고 싶은 단어는?apple
사과
찾고 싶은 단어는?babo
null
찾고 싶은 단어는?love
사랑
```

babo"를 해시맵에서 찾을 수 없기 때문에 null 리턴

예제: HashMap을 이용하여 자바 과목의 점수를 기록 관리하는 코드 작성

HashMap을 이용하여 학생의 이름과 자바 점수를 기록 관리해보자.

```
import java.util.*;

public class HashMapScoreEx {
  public static void main(String[] args) {
    // 사용자 이름과 점수를 기록하는 HashMap 컬렉션 생성
    HashMap<String, Integer> javaScore =
        new HashMap<String, Integer>();

    // 5 개의 점수 저장
    javaScore.put("한홍진", 97);
    javaScore.put("황기태", 34);
    javaScore.put("이영희", 98);
    javaScore.put("정원석", 70);
    javaScore.put("한원선", 99);

    System.out.println("HashMap의 요소 개수 :" + javaScore.size());
```

예제: HashMap을 이용하여 자바 과목의 점수를 기록 관리하는 코드 작성

```
    // 모든 사람의 점수 출력.
    // javaScore에 들어 있는 모든 (key, value) 쌍 출력
     // key 문자열을 가진 집합 Set 컬렉션 리턴
    Set<String> keys = javaScore.keySet();

    // key 문자열을 순서대로 접근할 수 있는 Iterator 리턴
    Iterator<String> it = keys.iterator();

    while(it.hasNext()) {
      String name = it.next();
      int score = javaScore.get(name);
      System.out.println(name + " : " + score);
    }
  }
}
```

```
HashMap의 요소 개수 :5
한원선 : 99
한홍진 : 97
황기태 : 34
이영희 : 98
정원석 : 70
```

예제: HashMap을 이용한 학생 정보 저장

id와 전화번호로 구성되는 Student 클래스를 만들고, 이름을 '키'로 하고 Student 객체를 '값'으로 하는 해시맵을 작성하라.

```
import java.util.*;

class Student { // 학생을 표현하는 클래스
  int id;
  String tel;
  public Student(int id, String tel) {
    this.id = id; this.tel = tel;
  }
}
```

예제: HashMap을 이용한 학생 정보 저장

```
public class HashMapStudentEx {
  public static void main(String[] args) {
    // 학생 이름과 Student 객체를 쌍으로 저장하는 HashMap 컬렉션 생성
    HashMap<String, Student> map = new HashMap<String, Student>();

    // 3 명의 학생 저장
    map.put("황기태", new Student(1, "010-111-1111"));
    map.put("한원선", new Student(2, "010-222-2222"));
    map.put("이영희", new Student(3, "010-333-3333"));

    System.out.println("HashMap의 요소 개수 :" + map.size());
```

예제: HashMap을 이용한 학생 정보 저장

```
    // 모든 학생 출력. map에 들어 있는 모든 (key, value) 쌍 출력
    // key 문자열을 가진 집합 Set 컬렉션 리턴
    Set<String> names = map.keySet();

     // key 문자열을 순서대로 접근할 수 있는 Iterator 리턴
    Iterator<String> it = names.iterator();
    while(it.hasNext()) {
      String name = it.next(); // 다음 키. 학생 이름
      Student student = map.get(name);
      System.out.println(name + " : " + student.id + " " + student.tel);
    }
  }
}
```

```
HashMap의 요소 개수 :3
한원선 : 2 010-222-2222
황기태 : 1 010-111-1111
이영희 : 3 010-333-3333
```

출력된 결과는 삽입된 결과와 다르다는 점을 기억하기 바람

```
    // 모든 학생 출력. map에 들어 있는 모든 (key, value) 쌍 출력
    // key 문자열을 가진 집합 Set 컬렉션 리턴
    Set<String> names = map.keySet();

     // key 문자열을 순서대로 접근할 수 있는 Iterator 리턴
    for (String name : names) {
      Student student = map.get(name);
      System.out.println(name + " : " + student.id + " " + student.tel);
    }
  }
}
```

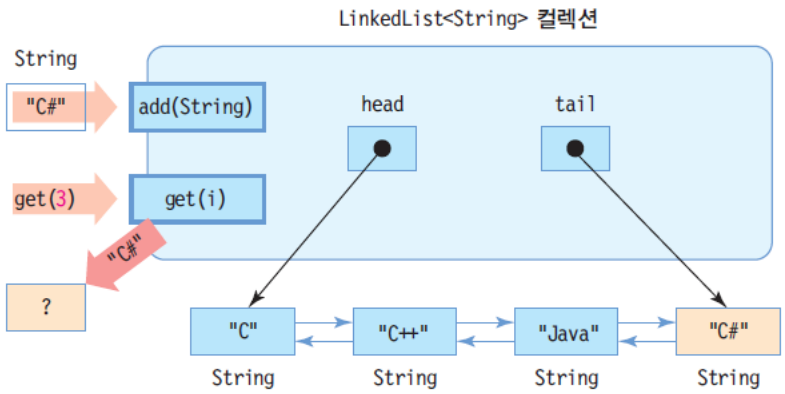
```
HashMap의 요소 개수 :3
한원선 : 2 010-222-2222
황기태 : 1 010-111-1111
이영희 : 3 010-333-3333
```

LinkedList<E>

- LinkedList<E>의 특성
  - java.util.LinkedList
    - E에 요소로 사용할 타입 지정하여 구체와
  - List 인터페이스를 구현한 컬렉션 클래스
  - Vector, ArrayList 클래스와 매우 유사하게 작동
  - 요소 객체들은 양방향으로 연결되어 관리됨
  - 요소 객체는 맨 앞, 맨 뒤에 추가 가능
  - 요소 객체는 인덱스를 이용하여 중간에 삽입 가능
  - 맨 앞이나 맨 뒤에 요소를 추가하거나 삭제할 수 있어 스택이나 큐로 사용 가능

LinkedList<String>의 내부 구성과 put(), get() 메소드

```
LinkedList<String> l = new LinkedList<String>();
```



Collections 클래스 활용

- Collections 클래스
  - java.util 패키지에 포함
  - 컬렉션에 대해 연산을 수행하고 결과로 컬렉션 리턴
  - 모든 메소드는 static 타입
  - 주요 메소드
    - 컬렉션에 포함된 요소들을 소팅하는 sort() 메소드
    - 요소의 순서를 반대로 하는 reverse() 메소드
    - 요소들의 최대, 최소값을 찾아내는 max(), min() 메소드
    - 특정 값을 검색하는 binarySearch() 메소드

예제: Collections 클래스의 활용

Collections 클래스를 활용하여 문자열 정렬, 반대로 정렬, 이진 검색 등을 실행하는 사례를 살펴보자.

```
import java.util.*;

public class CollectionsEx {
    static void printList(LinkedList<String> l) {
      Iterator<String> iterator = l.iterator();
      while (iterator.hasNext()) {
          String e = iterator.next();
          String separator;
          if (iterator.hasNext())
              separator = "->";
          else
              separator = "₩n";
          System.out.print(e+separator);
      }
    }
```

예제: Collections 클래스의 활용

```
  public static void main(String[] args) {
      LinkedList<String> myList = new LinkedList<String>();
      myList.add("트랜스포머");
      myList.add("스타워즈");
      myList.add("매트릭스");
      myList.add(0,"터미네이터");
      myList.add(2,"아바타");

      Collections.sort(myList); // 요소 정렬
      printList(myList); // 정렬된 요소 출력

      Collections.reverse(myList); // 요소의 순서를 반대로
      printList(myList); // 요소 출력

      int index = Collections.binarySearch(myList, "아바타") + 1;
      System.out.println("아바타는 " + index + "번째 요소입니다.");
  }
}
```

static 메소드이므로
클래스 이름으로 바로 호출

예제: Collections 클래스의 활용

소팅된
순서대로 출력

매트릭스->스타워즈->아바타->터미네이터->트랜스포머
트랜스포머->터미네이터->아바타->스타워즈->매트릭스
아바타는 3번째 요소입니다.

거꾸로
출력

Custom 클래스에 대한 sort 함수 사용

- 개인적으로 만든 클래스에 대해서 컬렉션에 추가하고, Collections.sort 기능을 이용해서 정렬하고 싶다면 java.lang.Comparable 인터페이스를 구현해주어야 함

```
public interface Comparable<T> {
   int compareTo(T o);
}
```

- CompareTo(T o) 메소드는 현 객체를 인자로 주어진 o와 비교해서 순서를 정한 후에 정수(int) 값을 반환함
  - 만약 현 객체가 주어진 인자보다 작다면 음수를 반환
  - 만약 현 객체가 주어진 인자와 동일하다면 0을 반환
  - 만약 현 객체가 주어진 인자보다 크다면 양수를 반환

Custom 클래스에 대한 sort 함수 사용

```
import java.util.ArrayList;
import java.util.Collections;

class A implements java.lang.Comparable<A> {
    int num;
    String s;

    public A(String s, int n) {
        this.s = s;
        num = n;
    }
```

Custom 클래스에 대한 sort 함수 사용

```
  public int compareTo(A a) {
        if (s.compareTo(a.s) == 0) {
            if (num > a.num)
                return 1;
            else if (num < a.num)
                return -1;
            else
                return 0;
        }
        else {
            return s.compareTo(a.s);
        }
    }

    public String toString() {
        return "String: " + s + "₩t num = " + num;
    }
}
```

Custom 클래스에 대한 Collections.sort 함수 사용

```
public class Main {
    public static void main(String[] args) {
        ArrayList<A> list = new ArrayList<A>();
        list.add(new A("Kim", 30));
        list.add(new A("Cho", 20));
        list.add(new A("Cho", 30));
        list.add(new A("Lee", 20));
        for (A b : list) {
          System.out.println(b);
        }
        System.out.println("After sorting");
        Collections.sort(list);
        for (A b : list) {
          System.out.println(b);
        }
    }
}
```

== & equals & hashCode

- **equals**는 두 객체의 **내용이 같은지** 동등성(equality)을 비교하는 연산자

```
Person p1 = new Person("Jason", 10);
Person p2 = new Person("Jason", 10);
Person p3 = p1;
// ==
if (p1 == p2) System.out.println("p1 == p2");
else System.out.println("p1 != p2"); //동일한 ref 아니므로 p1 != p2
if (p1 == p3) System.out.println("p1 == p3"); // 동일한 ref므로 p1 == p3
else System.out.println("p1 != p3");
// equals
// equals override 되어있으면 true, 안되어있으면 false
if (p1.equals(p2)) System.out.println("p1 equals p2");
else System.out.println("p1 is not equal to p2");
```

== & equals & hashCode

- **hashCode**는 두 객체가 **같은 객체인지** 동일성(identity)을 비교하는 연산자

```
Map<Person, Integer> pMap = new HashMap<Person, Integer>();
pMap.put(p1, 1); // equals&hashCode override있다면, p1과 p2는 동일
pMap.put(p2, 2); // equals override&no hashCode라면, p1과 p2는 다름
for (Map.Entry<Person, Integer> entry : pMap.entrySet()) {
    System.out.println("P : "+entry.getKey()+" Index : " +entry.getValue());
}
System.out.println("pMap=" + pMap.size()); // pMap=1
pMap.remove(p1); // p1과 p2가 같은 hashCode, p1으로 p2 를 같이 지움
for (Map.Entry<Person, Integer> entry : pMap.entrySet()) {
    System.out.println("P: "+entry.getKey()+" Index : " +entry.getValue());
}
System.out.println("after remove pMap=" + pMap.size()); // pMap=0
```

제네릭 만들기

- 제네릭 클래스와 인터페이스
  - 클래스나 인터페이스 선언부에 일반화된 타입 추가

```
public class MyClass<T> {
  T val;
  void set(T a) {
    val = a;
  }
  T get() {
    return val;
  }
}
```

val의 타입은 T

제네릭 클래스 MyClass 선언, 타입 매개 변수 T

T 타입의 값 a를 val에 지정

T 타입의 값 val 리턴

  - 제네릭 클래스 레퍼런스 변수 선언

```
MyClass<String> s;
List<Integer> li;
Vector<String> vs;
```

제네릭 객체 생성 – 구체화(specialization)

- 구체화
  - 제네릭 타입의 클래스에 구체적인 타입을 대입하여 객체 생성
  - 컴파일러에 의해 이루어짐

```
// 제네릭 타입 T에 String 지정
MyClass<String> s = new MyClass<String>();
s.set("hello");
System.out.println(s.get()); // "hello" 출력

// 제네릭 타입 T에 Integer 지정
MyClass<Integer> n = new MyClass<Integer>();
n.set(5);
System.out.println(n.get()); // 숫자 5 출력
```

  - 구체화된 MyClass<String>의 소스 코드

제네릭 객체 생성 – 구체화(specialization)

```
public class MyClass<T> {
  T val;
  void set(T a) { val = a; }
  T get() { return val; }
}
```

T가 String으로 구체화

```
public class MyClass<String> {
  String val; // 변수 val의 타입은 String
  void set(String a) {
    val = a; // String 타입의 값 a를 val에 지정
  }
  String get() {
    return val; // String 타입의 값 val을 리턴
  }
}
```

구체화 오류

- 타입 매개 변수에 기본 타입은 사용할 수 없음

```
Vector<int> vi = new Vector<int>(); // 컴파일 오류. int 사용 불가
```

수정

```
Vector<Integer> vi = new Vector<Integer>(); // 정상 코드
```

타입 매개 변수

- 타입 매개 변수
  - '<'과 '>'사이에 하나의 대문자를 타입 매개 변수로 사용
  - 많이 사용하는 타입 매개 변수 문자
    - E : Element를 의미하며 컬렉션에서 요소를 표시할 때 많이 사용한다.
    - T : Type을 의미한다.
    - V : Value를 의미한다.
    - K : Key를 의미
  - 타입 매개변수가 나타내는 타입의 객체 생성 불가
    - //T a = new T(); // 오류!!
  - 타입 매개 변수는 나중에 실제 타입으로 구체화
  - 어떤 문자도 매개 변수로 사용 가능

예제: 제네릭 스택 만들기

스택을 제네릭 클래스로 작성하고, String과 Integer형 스택을 사용하는 예를 보여라.

```
class GStack<T> {
  int tos;
  Object [] stck;
  public GStack() {
    tos = 0;
    stck = new Object [10];
  }
  public void push(T item) {
    if(tos == 10) return;
    stck[tos] = item;
    tos++;
  }
  public T pop() {
    if(tos == 0) return null;
    tos--;
    return (T)stck[tos];
  }
}
```

예제: 제네릭 스택 만들기

```
public class MyStack {
  public static void main(String[] args) {
    GStack<String> stringStack = new GStack<String>();
    stringStack.push("seoul");
    stringStack.push("busan");
    stringStack.push("LA");
    for(int n=0; n<3; n++)
      System.out.println(stringStack.pop());

    GStack<Integer> intStack = new GStack<Integer>();
    intStack.push(1);
    intStack.push(3);
    intStack.push(5);

    for(int n=0; n<3; n++)
      System.out.println(intStack.pop());
  }
}
```

```
LA
busan
seoul
5
3
1
```

제네릭과 배열

- 제네릭에서 배열의 제한
  - 제네릭 클래스 또는 인터페이스의 배열을 허용하지 않음

```
//GStack<Integer>[] gs = new GStack<Integer>[10]; // 오류!!
```

  - 제네릭 타입의 배열도 허용되지 않음

```
//T[] a = new T[10]; // 오류!!
```

    - 앞 예제에서는 Object 타입으로 배열 생성 후 실제 사용할 때 타입 캐스팅

```
return (T)stck[tos]; // 타입 매개 변수 T타입으로 캐스팅
```

  - 타입 매개변수의 배열에 레퍼런스는 허용

```
public void myArray(T[] a) { ... }
```

제네릭 메소드

- 제네릭 메소드 선언 가능

```
class GenericMethodEx {
  static <T> void toStack(T[] a, GStack<T> gs) {
    for (int i = 0; i < a.length; i++) {
      gs.push(a[i]);
    }
  }
}
```

  - 제네릭 메소드를 호출할 때는 컴파일러가 메소드의 인자를 통해 이미 타입을 알고 있으므로 타입을 명시하지 않아도 됨

```
String[] sa = new String[100];
GStack<String> gss = new GStack<String>();
GenericMethodEx.toStack(sa, gss);
```

    - sa는 String[], gss는 GStack<String> 타입이므로 T를 String으로 유추

예제 : 스택의 내용을 반대로 만드는 제네릭 메소드 만들기

GStack을 이용하여 주어진 스택의 내용을 반대로 만드는 제네릭 메소드 reverse()를 작성하라.

```
public class GenericMethodExample {
  // T가 타입 매개 변수인 제네릭 메소드
  public static <T> GStack<T> reverse(GStack<T> a){
    GStack<T> s = new GStack<T>();
    while (true) {
      T tmp;
      tmp = a.pop(); // 원래 스택에서 요소 하나를 꺼냄
      if (tmp==null) // 스택이 비었음
        break;
      else
        s.push(tmp); // 새 스택에 요소를 삽입
    }
    return s; // 새 스택을 반환
  }
```

```
public static void main(String[] args) {
    // Double 타입의 GStack 생성
    GStack<Double> gs =
        new GStack<Double>();

    // 5개의 요소를 스택에 push
    for (int i=0; i<5; i++) {
      gs.push(new Double(i));
    }
    gs = reverse(gs);
    for (int i=0; i<5; i++) {
      System.out.println(gs.pop());
    }
  }
}
```

```
0.0
1.0
2.0
3.0
4.0
```

제네릭의 장점

- 컬렉션과 같은 컨테이너 클래스에 유연성을 해치지 않으며 type-awareness를 첨가
- 메소드에 type-awareness 첨가
- 컴파일 시에 타입이 결정되어 보다 안전한 프로그래밍 가능
- 개발 시 다운캐스팅(타입 캐스팅) 절차 불필요
- 런타임 타입 충돌 문제 방지
- ClassCastException 방지