chapter 07.

제네릭과 컬렉션

❍ 제네릭

- Generics는 컬렉션(자료구조) 즉, 쉽게 말해서 객체들을 저장(수집)하는 구조적인 성격을 보강하기 위해 제공되는 것이다.

  간단히 예를 들자면 컵이라는 특정 객체가 있다. 이 컵은 물만 담을 수 있는 물컵, 또는 이 컵은 주스만 담을 수 있는 주스 컵!

  이렇게 상징적인 것이 바로 Generics다!

❍ 제네릭의 필요성

- JDK5.0에 와서 Generics가 포함되면서 이제 프로그래머가 특정 컬렉션(자료구조)에 원하는 객체 타입을 명시하여 실행하기 전에 컴파일단계에서 지정된 객체가 아니면 절대 저장이 불가능하게 할 수 있다.

- 이전 버전까지는 반드시 실행하여 컬렉션(자료구조)에 있는 자원들을 하나씩 검출하여 확인할 수 밖에 없었다. Object로부터 상속받은 객체는 모두 저장이 가능했던 이전의 버전들과는 달리 보다 체계(體系)적이라 할 수 있다.

- 실행 시 자원 검출을 하게 되면 별도의 형 변환(Casting)이 필요 없이 <>사이에 선언하였던 객체자료 형으로 검출되어 편리하다.

❍ <1글자로 된 영문대문자>

- API에서는 전달되는 객체가 현 객체 내에서 자료형(Type)으로 쓰일 때 <T>로 유도를 하고 있으며 만약 전달되는 객체가 현 객체 내에서 하나의 요소(Element)로 자리를 잡을 때는 <E>로 그리고 전달되는 객체가 현 객체 내에서 Key값으로 사용될 때는 <K>로, 만약 전달되는 객체가 현 객체 내에서 Value값으로 사용될 때 <V>로 표현하고 있다.

[접근제한] class 클래스명<유형1,유형2…유형n>

T s; 또는 T[] arr;	객체가 생성시 전달된 상징적 자료형(Generic Type)이 String형이었다면 왼쪽의 코드는 다음과 같이 대체(代替)된다. String s; String[] arr;

- 그럼 앞에서 공부한 Generic Type의 유형을 이용하여 우리가 직접 클래스를 정의해 보도록 하자!

[예제7-1] GenericEx1.java

```
01  import static java.lang.System.out;
02  class GenericEx1<T> {
03
04      T[] v;
05
06      public void set(T[] n){
07              v = n;
08      }
09      public void print(){
10              for(T s : v)
11                  out.println(s);
12      }
13  }
```

❍ 제네릭 타입 사용하기

- 앞서 간단한 Generic class를 우리가 직접 정의해 보았다. 이제 이것을 사용하려면 Generic class의 변수 선언과 생성 법을 익히면 된다. 다음의 구성을 보자!

```
Generic_class명<적용할_Generic_Type> 변수명; // 선언
변수명 = new Generic_class생성자명<적용할_Generic_Type>(); //생성
```

[예제7-2] GenericEx1Main.java

```
01  public class GenericEx1Main{
02          public static void main(String[] args){
03                  GenericEx1<String> t = new GenericEx1<String>();
04
05                  String[] ss = {"애","아","서"};
06                  t.set(ss);
07                  t.print();
08  /* 좋은 방법이 아님
09                  GenericEx1 t1 = new GenericEx1();
10                  Integer[] s = {1,2,3};
11                  t1.set(s);
12                  t1.print();  */
13          }
14  }
```

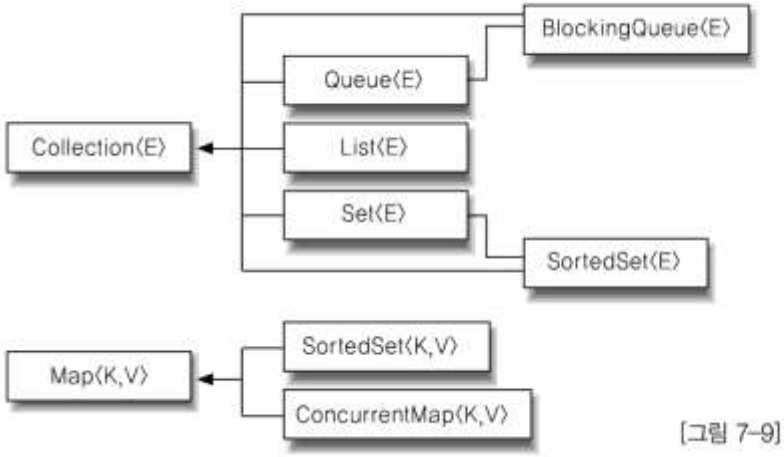
실행결과

```
---------- Java Run ----------
애
아
서
Normal Termination
출력 완료 (0초 경과).
```

- 자바에서 얘기하는 Java Collections Framework는 객체들을 한 곳에 모아 관리하고 또 그것을 편하게 사용하기 위해 제공되는 환경이다. 여기에는 다음과 같이 구조를 이루고 있다.

[표 7-2] 자바 켈렉션 프레임워크 구조

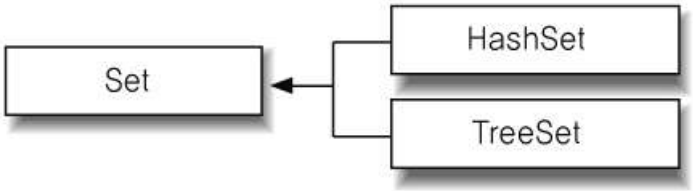
구조	설명
Interfaces(인터페이스)	컬렉션들이 가져야 하는 조작에 대한 설명과 함께 기능들을 추상적으로 표현한 것들이다. 예를 들자면 객체에 대한 검색과 삭제에 관련된 기능들의 목록이다. 그리고 이것은 계층적인 구조를 이루게 한다.
Implementations(구현 객체)	위의 Interface들을 구체적으로 구현한 클래스들을 의미한다. 그러므로 재사용을 할 수 있도록 하는 자료의 구조인 것이다.
Algorithms(메서드)	Interface를 구현한 객체들의 검색 그리고 정렬과 같은 유용한 동작들, 즉 메서드들을 의미한다.



[그림 7-9] 자바 컬렉션 인터페이스의 구조

❍Set인터페이스

- Set내에 저장되는 객체들은 특별한 기준에 맞춰서 정렬되지 않는다. 그리고 저장되는 객체들간의 중복된 요소가 발생하지 못하도록 내부적으로 관리되고 있다. 다음은 Set인터페이스를 구현하고 있는 클래스들이다. 더 자세한 것은 API문서를 참조하길 바라며 다음 그림은 기본적으로 알려진 구현 클래스들이므로 참조하기 바란다.



[그림 7-10] Set의 구조

[표 7-4] Set의 구현 클래스

구현 클래스	설명
HashSet	Set 인터페이스를 구현하고 있으며 내부적으로 HashMap을 사용하고 있다. 얻어지는 Iterator의 정렬 상태를 보장하지 못하므로 특정한 기준으로 정렬을 이루고 있지 않으며 저장 및 검출과 같은 동작에는 일정한 시간을 필요로 한다.
TreeSet	내부적으로 Set 인터페이스를 구현하고 있으며 TreeMap에 의해 후원을 받는다. 그리고 기본적으로 얻어지는 Iterator의 요소들은 오름차순 정렬 상태를 유지하고 있다.

HashSet

- 기본적인 Set인터페이스를 구현하고 있으며 정렬순서나 반복처리 시 처리순서에 대한 기준은 없다. 그리고 반복 처리에 대해서는 저장된 요소(Element)의 수와는 별도로 용량에 비례하는 시간이 필요하므로 반복 처리하는 성능이 중요한 응용프로그램에서는 초기용량을 너무 높게 설정하지 않는 것이 중요하다.

[표 7-5] HashSet 생성자 요약

생성자명	설명
HashSet( )	새로운 HashSet 객체를 생성하고 초기화한다.

[표 7-6] HashSet 메서드 요약

반환형	메서드명	설명
boolean	add(E o)	제네릭 타입으로 넘어온 객체가 Set 구조에 없다면 추가하고 true를 반환한다.
void	clear( )	Set 구조에 있는 모든 요소들을 삭제한다.
boolean	contains(Object o)	인자로 전달된 객체를 현 Collection에서 요소로 가지고 있으면 true를 반환한다.
	isEmpty( )	현 Collection에 저장된 요소가 없다면 true를 반환한다.
Iterator<E>	iterator( )	현 Set 구조의 요소들을 순서대로 처리하기 위해 Iterator 객체로 반환한다.
boolean	remove(Object o)	현 Set 구조에서 인자로 전달된 객체를 삭제한다. 이때 삭제에 성공하면 true를 반환한다.
int	size( )	현 Set 구조에 저장된 요소의 수를 반환한다.

[예제7-10] HashSetEx1.java

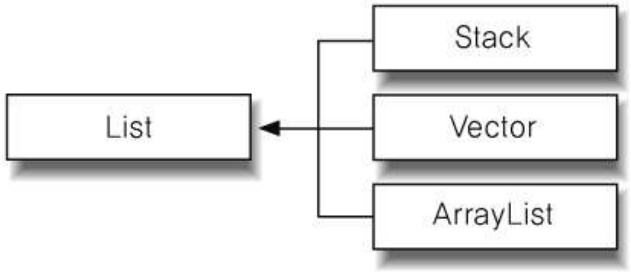
```
01  import java.util.*;
02  import static java.lang.System.out;
03  public class HashSetEx1 {
04  public static void main(String args[]) {
05          String[] str = {"Java","Beans","Java","XML"};
06
07          HashSet<String> hs1 = new HashSet<String>();
08          HashSet<String> hs2 = new HashSet<String>();
09          for (String n : str){
10          if (!hs1.add(n))
11              hs2.add(n);
12          }
13          out.println("hs1 : " + hs1);
14      hs1.removeAll(hs2);
15      out.println("hs1 : " + hs1);
16      out.println("hs2 : " + hs2);
17    }
18  }
```

실행결과

```
---------- Java Run ----------
hs1 : [Java, Beans, XML]
hs1 : [Beans, XML]
hs2 : [Java]
Normal Termination
출력 완료 (0초 경과).
```

List인터페이스

- List구조는 Sequence라고도 하며 시작과 끝이 선정되어 저장되는 요소들을 일괄적인 정렬상태를 유지하면서 요소들의 저장이 이루어진다. 이런 점 때문에 List구조하면 배열을 영상 하게 되는데 무리는 아니다. 어떻게 보면 배열과 컬렉션의 List구조는 같다고 볼 수 있으며 다르다면 배열은 크기가 고정되어 있는 것이고 컬렉션의 List구조는 가변적 길이를 가진다는 것이다. 다음은 List구조에서 알려진 구현 클래스들이다.



[그림 7-13] List의 구조

[그림 7-14] List의 도식화

[표 7-7] List의 구현 클래스

구현 클래스	설명
Stack	Stack 클래스는 객체들의 last-in-first-out(LIFO) 스택을 표현한다. 그리고 Vector 클래스로부터 파생된 클래스다. 요소를 저장할 때의 push( ) 메서드와 요소를 빼낼 때의 pop( ) 메서드 등 총 5개의 메서드를 제공한다.
Vector	배열과 같이 정수 인덱스로 접근할 수 있는 접근 소자를 가지고 있다. 하지만 배열과는 달리 Vector의 크기는 Vector가 생성된 후에 요소를 추가하는 경우에 따라 증대되고 또는 제거할 때에 따라 감소할 수 있다. 그리고 요소들의 작업을 위해 Iterator로 작업할 수 있으며 나중에 배우는 스레드 동기화가 지원되는 List 구조다.
ArrayList	List 인터페이스를 구현하고 있는 것뿐 아니라 ArrayList는 배열의 크기를 조작하기 위하여 메서드들이 제공된다. 공백을 포함한 모든 요소들을 저장할 수 있으며 Vector와 유사하지만 ArrayList는 스레드의 동기화는 지원하지 않는다.

- 이렇게 몇 개의 구현 클래스들을 확인해 보았다. 아무래도 List구조인 객체들은 Set과는 다르게 정렬상태를 유지하면서 각 요소들의 접근력을 Set보다는 쉽게 가지는 구조라 할 수 있겠다.

여기서 Stack구조만 알아보고 나머지는 교재의 361p부터 참조하기 바란다.

- Stack은 객체를 후입선출(後入先出), last-in-first-out(LIFO)이며 객체의 저 정시의 push()메서드와 검출 시 사용하는 pop()과 Stack의 가장 위쪽 객체를 의미하는 peek()메서드 그리고 Stack이 비어있는지 판별해주는 empty()와 객체를 검색해주는 search()메서드들로 다음에 나오는 Vector라는 클래스를 확장하였다.

[표 7-8] Stack 생성자에 대한 요약

생성자명	설명
Stack( )	새로운 Stack 객체를 생성하고 초기화한다.

[표 7-9] Stack의 주요 메서드

반환형	메서드명	설명
boolean	empty( )	Stack이 비었는지 비교하여 비어 있으면 true를 반환한다.
E	peek( )	Stack의 가장 위쪽에 있는 객체를 반환한다.
	pop( )	Stack의 가장 위쪽에 있는 객체를 삭제하고 그 객체를 반환한다.
	push(E item)	Stack의 가장 위쪽에서 객체를 추가한다.
int	search(Object o)	현재 Stack의 구조에서 인자로 전달받은 객체의 인덱스값을 반환한다(참고로 인덱스값은 1부터 시작한다).

:: Stack은 List구조이지만
가방과 같은 구조라고
생각하면 된다.
입구가 하나라서 제일 먼저
넣은 물건(객체)이 가장 아래에
위치하므로 꺼낼 때는
가장 나중에 나오게 된다.

[예제7-11] StackEx1.java

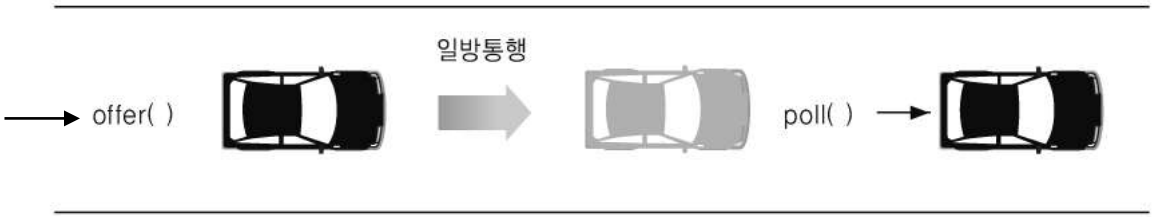
```
01  import java.util.Stack;
02  import static java.lang.System.out;
03  public class StackEx1{
04
05      public static void main(String[] args) {
06          String[] groupA = {"우즈베키스탄","쿠웨이트","사우디","대한민국"};
07
08          Stack<String> stack = new Stack<String>();
09          for(String n : groupA)
10                  stack.push(n);
11
12          while(!stack.isEmpty())
13                  out.println(stack.pop());
14      }
15  }
```

실행결과

```
---------- Java Run ----------
대한민국
사우디
쿠웨이트
우즈베키스탄
Normal Termination
출력 완료 (0초 경과).
```

Queue인터페이스

- JDK5.0에 오면서 새롭게 추가된 인터페이스이며 Queue의 구조는 도로의 일정구간인 일방통행(一方通行)과 같다. 요소(Element)가 들어가는 입구와 요소(Element)가 나오는 출구가 따로 준비 되어 있어 가장 먼저 들어간 요소(Element)가 가장 먼저 나오는 선입선출(先入先出), first-in-first-out(FIFO)방식이다.



[그림 7-23] Queue의 도식화

- API문서를 참조해 보면 Queue를 구현하는 클래스는 여러 가지가 있는 것을 알 수 있으며 여기서는 그 중에서 LinkedList에 대해서 알아보도록 하겠다.

LinkedList

- LinkedList는 add()메서드와 poll()메서드 등에 의해 선입선출(先入先出)법을 제공하는 Queue 인터페이스를 구현하며 Thread동기화는 제공되지 않는다.

[표 7-13] LinkedList 생성자에 대한 요약

생성자명	설명
LinkedList( )	새로운 LinkedList 객체가 생성된다.
LinkedList(Collection 〈? extends E 〉 c)	전달된 컬렉션을 포함하는 LinkedList 객체가 생성된다.

[표 7-14] LinkedList의 주요 메서드

반환형	메서드명	설명
void	add(E o)	마지막으로 전달된 요소를 추가한다.
	addFirst(E o)	첫 번째로 전달된 요소를 추가한다.
E	element( )	가장 첫 번째 요소를 반환한다. 단 삭제하지는 않는다.
boolean	offer(E o)	전달된 요소를 마지막 요소로 추가한다.
E	peek( )	가장 첫 번째 요소를 반환한다. 삭제는 하지 않는다.
	poll( )	가장 첫 번째 요소를 반환한 후 삭제한다.
boolean	remove(Object o)	인자로 전달된 객체를 현재 Queue에서 최초로 검출된 요소를 삭제한다.
E	removeFirst( )	첫 번째 요소를 반환한 후 삭제한다.

[예제7-16] QueueEx1.java

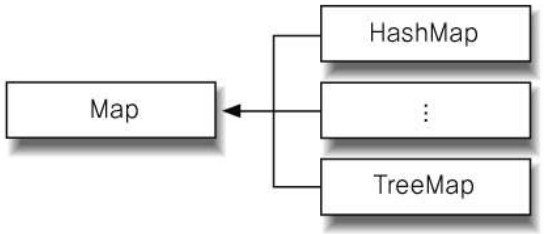
```
01  import java.util.Vector;
02  import static java.lang.System.out;
03  public class QueueEx1{
04
05          public static void main(String[] args) {
06                  String[] item = {"소나타","렉스톤","제규어"};
07                  LinkedList<String> q = new LinkedList<String>();
08
09                  for(String n : item)
10                          q.offer(n); //요소 추가
11                  out.println("q의 크기:"+q.size());
12
13                  String data="";
14                  while((data = q.poll()) != null)
15                          out.println(data+"삭제!");
16
17                  out.println("q의 크기:"+q.size());
18          }
19  }
```

실행결과

```
---------- Java Run ----------
q의 크기:3
소나타삭제!
렉스톤삭제!
제규어삭제!
q의 크기:0
Normal Termination
출력 완료 (0초 경과).
```

Map인터페이스

- Key와 Value를 매핑하는 객체이다. 여기에 사용되는 Key는 절대 중복될 수 없으며 각 Key는 1개의 Value만 매핑할 수 있다. 정렬의 기준이 없으며 이는 마치 각 Value에 열쇠 고리를 달아서 큰 주머니에 넣어두고 오로지 Key로 각 Value를 참조 할 수 있도록 해둔 구조라 할 수 있다.



[그림 7-25] Map의 구조

- 사용자가 원하는 Value의 Key를 알고 있다면 Key를 당겨(get) 해당 Key와 매핑되어 있는 Value를 얻을 수 있는 구조이다. 즉, 검색을 Key로 해야 하므로 Key를 모르면 원하는 Value를 얻어내는 못하게 된다.

[표 7-15] Map의 구현 클래스

구현 클래스	설명
Hashtable	정렬의 기능을 가지지 않는 Map 인터페이스를 구현하고 있다. 이는 Key가 null을 가질 수 없으며, Value 또한 null을 허용하지 않는다. 중복 또한 불가능하며 스레드 동기화를 지원하는 특징을 가지고 있다.
HashMap	앞의 Hashtable과 거의 동일한 객체로 다른 점이 있다면 Key와 Value에 있어 null을 허용한다는 점과 스레드 동기화를 지원하지 않는다는 것이다.

HashMap

- Key와 Value를 하나의 쌍으로 저장되는 구조이며 저장되는 Value와 Key가 null을 허용한다. 하지만 중복은 허용하지 않으므로 null을 가지는 Key가 2개일 수는 없다. 그리고 동기화가 포함되지 않았으므로 나중에 배우는 Multi-Thread환경에서의 구현이 아니라면 Hashtable에 비해서 처리 속도가 빠른 장점을 가지고 있다.

[표 7-16] HashMap의 주요 생성자

생성자명	설명
HashMap( )	초기용량을 16으로 하고 적재율은 0.75로 하여 새로운 HashMap 객체가 생성된다.
HashMap (int initialCapacity)	전달된 인자를 통해 객체를 저장할 수 있는 초기용량으로 설정되고 기본 적재율인 0.75로 HashMap 객체가 생성된다.
HashMap(int initialCapacity, float loadFactor)	전달된 인자인 용량과 적재율로 새로운 HashMap 객체가 생성된다.

[표 7-17] HashMap의 주요 메서드

반환형	메서드명	설명
void	clear( )	모든 매핑을 맵으로부터 삭제한다.
V	get(Object key)	인자로 전달된 key 객체와 매핑되고 있는 Value를 반환한다.
boolean	isEmpty( )	현재 맵이 비어있다면 true를 반환한다.
Set <K>	keySet( )	맵에 저장되고 있는 Key들을 Set 인터페이스로 반환한다.
V	put(K key, V value)	인자로 전달된 Key와 Value를 현재 맵에 저장한다.
	remove(Object key)	인자로 전달된 Key에 대한 것이 있다면 현재 맵에서 삭제하고 매핑된 Value를 반환한다. 전달된 Key에 대한 정보가 없다면 null을 반환한다.
int	size( )	맵에 저장된 Key와 Value로 매핑된 수를 반환한다.
Collection <V>	values( )	현재 맵에 저장된 Value들만 Collection 인터페이스로 반환한다.

[예제7-17] MapEx1.java

```
01  import java.util.*;
02  import static java.lang.System.out;
03  public class MapEx1{
04
05      public static void main(String[] args) {
06          String[] msg = {"Berlin","Dortmund","Frankfurt",
07              "Gelsenkirchen","Hamburg"};
08
09          HashMap<Integer,String> map =
10              new HashMap<Integer,String>(); // HashMap생성
11
12          for(int i=0 ; i<msg.length ; i++)
13              map.put(i,msg[i]); //맵에 저장
14
15          Set<Integer> keys = map.keySet();
16              for(Integer n : keys)
17                  out.println(map.get(n)); //맵에서 읽어오기
18          }
19  }
```

실행결과

```
---------- Java Run ----------
Frankfurt
Hamburg
Dortmund
Gelsenkirchen
Berlin
Normal Termination
출력 완료 (0초 경과).
```