Power Java

제22장 제네릭과 컬렉션 (Generic and Collection)

이번 장에서 학습할 내용

- Generic class
- Generic method
- Collection
- ArrayList
- LinkedList
- Set
- Queue
- Map
- Collections class

Generic이란?

- Generic programming
  - 다양한 타입의 객체를 동일한 코드로 처리하는 기법
  - Generic은 Collection 라이브러리에 많이 사용

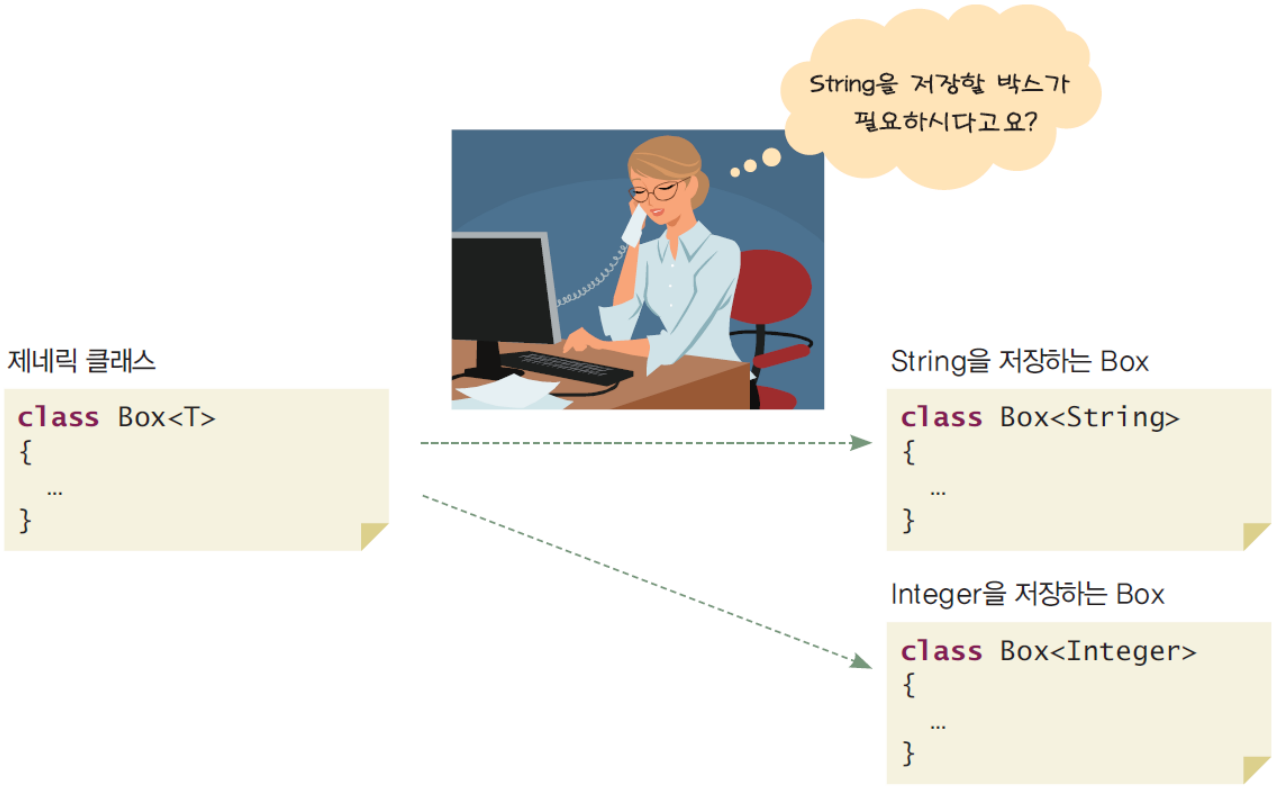


그림22-1. Generic 프로그래밍의 개념

기존의 방법

- 먼저 단 하나의 데이터만을 저장할 수 있는 Box라는 간단한 class를 작성하여 보자.

```
public class  Box {
    private Object data;
    public void set(Object data)    {   this.data = data; }
    public Object get()             {   return data;      }
}
```

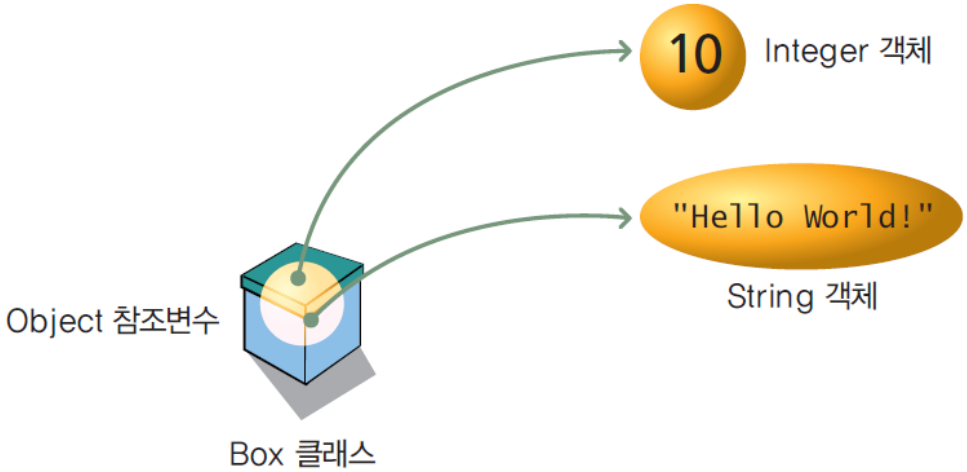


그림22-2. Box class는 다양한 타입의 객체 한 개를 저장할 수 있는 class이다.

예제

- 실제로 Box class는 여러 가지 다양한 타입의 데이터를 저장할 수 있다.

```
Box b = new Box();
b.set(new Integer(10));          // ❶ 정수 객체 저장
b.set("Hello World!");           // 정수 객체가 없어지고 문자열 객체를 저장
String s = (String)b.get();      // ❷ Object 타입을 String 타입으로 형변환
```

- 문자열을 저장하고서도 부주의하게 Integer 객체로 형 변환을 할 수도 있으며 이것은 실행 도중에 오류를 발생한다.

```
b.set("Hello World!");
Integer i = (Integer)b.get( );     // 오류! 문자열을 정수 객체로 형변환
```

실행결과

```
Exception in thread "main" java.lang.ClassCastException: java.lang.String cannot
be cast to java.lang.Integer at GenericTest.main(GenericTest.java:10)
```

Generic 이용 방법

- Generic class에서는 타입을 변수로 표시한다.
- 이것을 타입 매개변수 (type parameter)라고 하는데 타입 매개변수는 객체 생성 시에 프로그래머에 의하여 결정된다.

```
class name<T1, T2, ..., 쑤> {  ...  }
```

- Box class를 generic으로 다시 작성하여 보면 다음과 같다.
- " **public class** Box " 을 " **public class** Box<T> " 로 변경하면 된다. 여기서는 T가 타입 매개변수가 된다.

```
public class Box<T> {
    private T data;
    public void set(T data) { this.data = data; }
    public T get() { return data; }
}
```

- 타입 매개변수의 값은 객체를 생성할 때 구체적으로 결정된다. 예를 들어서 문자열을 저장하는 Box class의 객체를 생성하려면 T 대신에 String을 사용하면 된다.

```
Box<String>  b = new Box<String>();
```

- 타입 매개변수의 값은 객체를 생성할 때 구체적으로 결정된다. 예를 들어서 문자열을 저장하는 Box class의 객체를 생성하려면 T 대신에 String을 사용하면 된다.

Generic 이용 방법

- 만약 정수를 저장하는 Box class의 객체를 생성하려면 다음과 같이 T 대신에 <Integer>를 사용하면 된다.

```
Box<Integer> b = new Box<Integer>();
```

- 하지만 int는 사용할 수 없는데, int는 기초 자료형이고 class가 아니기 때문이다

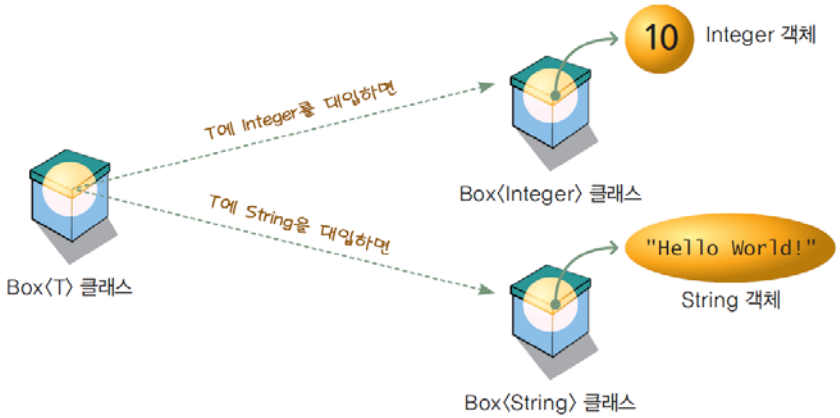


그림22-3. Box class에 저장하는 데이터의 타입은 객체 생성 시에 결정된다.

Generic 이용 방법

- 문자열을 저장하는 객체를 생성하여 사용하면 다음과 같다

```
Box<String> b = new Box<String>();
b.set("Hello World!");              // 문자열 타입 저장
String s = Box.get();
```

- 만약 Box<String>에 정수 타입을 추가하려고 하면 컴파일러가 컴파일 단계에서 오류를 감지할 수 있다. 따라서 더 안전하게 프로그래밍할 수 있다.

```
Box<String> stringBox = new Box<String>();
stringBox.set(new Integer(10));          // 정수 타입을 저장하려고 하면 컴파일 오류!
```

실행결과

```
The method set(String) in the type Box<String> is not applicable for the arguments
(Integer) at GenericTest.main(GenericTest.java:27)
```

타입 매개 변수의 표기

- E - Element(요소: Java Collection 라이브러리에서 많이 사용된다.)
- K - Key
- N - Number
- T - Type
- V - Value
- S, U, V 등 - 2번째, 3번째, 4번째 타입

다이아몬드(<>)

- Java SE 7 버전부터는 generic class의 생성자(constructor)를 호출할 때, 타입 인수를 구체적으로 주지 않아도 된다. 컴파일러는 문맥에서 타입을 추측한다.

```
Box<String> Box = new Box<>();
```

생성자 호출 시 구체적인 타입을 주지 않아도 된다.

다중타입매개변수(Multiple Type Parameters)

```
public interface Pair<K, V> {
    public K getKey();
    public V getValue();
}

public class OrderedPair<K, V> implements Pair<K, V> {
    private K key;
    private V value;

    public OrderedPair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey(){ return key; }
    public V getValue() { return value; }
}
```

타입 매개변수가 2개인 인터페이스를 정의한다.

K는 key의 타입이고, V는 value의 타입이다.

- 위의 정의를 이용하여서 객체를 생성해보면 다음과 같다

```
Pair<String, Integer> p1 = new OrderedPair<String, Integer>("Even", 8);
Pair<String, String>  p2 = new OrderedPair<String, String>("hello", "world");
```

Raw 타입

- Raw 타입은 타입 매개 변수가 없는 generic class의 이름이다.
- 앞의 box class를 다음과 같이 사용하면 Raw 타입이 된다.

```
Box<Integer> intBox = new Box<>();
```

```
Box rawBox = new Box();
```

```
public class Box<T> {
   private T data;
   public void set(T data) { this.data = data; }
   public T get() { return data; }
}
```

- Raw 타입은 JDK 5.0 이전에는 generic이 없었기 때문에 이전 코드와 호환성을 유지하기 위하여 등장
- 타입을 주지 않으면 무조건 Object 타입으로 간주

```
Box<String> stringBox = new Box<>();
Box rawBox = stringBox;                    // OK
```

중간점검

중간점검

1. 왜 데이터를 Object 참조형 변수에 저장하는 것이 위험할 수 있는가?
2. Box 객체에 Rectangle 객체를 저장하도록 제네릭을 이용하여 생성하여 보라.
3. 타입 매개변수 T를 가지는 Point 클래스를 정의하여 보라. Point 클래스는 2차원 공간에서 점을 나타낸다.

Generic Method

- Method에서도 타입 매개 변수를 사용하여서 generic method를 정의할 수 있다.
- 타입 매개 변수의 범위가 method 내부로 제한된다.

```
public class Array
{

    public static <T> T getLast(T[] a)
    {
        return a[a.length-1];
    }
}
```

제네릭 메소드 정의

- Generic method를 호출하기 위해서는 실제 타입을 꺽쇠 안에 적어준다.

```
String[]  language = { "C++", "C#", "JAVA" };
String  last = Array.<String>getLast(language);    // last는 "JAVA"
```

- 여기서 method 호출시에는 <String>는 생략하여도 된다. 왜냐하면 컴파일러는 이미 타입 정보를 알고 있기 때문이다. 즉 다음과 같이 호출하여도 된다.

```
String  last = Array.getLast(language);        // last는 "JAVA"
```

한정된 타입 매개변수

- 배열 원소 중에서 가장 큰 값을 반환하는 generic method를 작성하여 보자

```
public class Array
{
   public static <T> T getMax(T[] a)
   {
      if (a == null || a.length == 0)
         return null;
      T largest = a[0];
      for (int i = 1; i < a.length; i++)
         if (largest.compareTo(a[i]) > 0)
            largest = a[i];
      return largest;
   }
}
```

- 타입 매개변수 T가 가리킬 수 있는 class의 범위를 Comparable interface를 구현한 class로 제한해야 한다. 그렇지 않으면 error.

```
public static <T extends Comparable> T getMax(T[] a)
{
   ...
}
```

중간점검

중간점검

1. 제네릭 메소드 sub()에서 매개변수 d를 타입 매개변수를 이용하여서 정의하여 보라.
2. displayArray()라는 메소드는 배열을 매개변수로 받아서 반복 루프를 사용하여서 배열의 원소를 화면에 출력한다. 어떤 타입의 배열도 처리할 수 있도록 제네릭 메소드로 정의하여 보라.

Generic과 상속

- 우리는 다형성에 의하여 Integer 객체를 Object 객체 변수로 가리키게 할 수 있음을 알고 있다. Integer 가 Object로부터 상속받았기 때문이다.

```
Object obj = new Object();
Integer i = new Integer(10);
obj = i;  // OK
```

- Number를 타입 매개변수로 주어서 객체를 생성하였으면 Number의 자식 class인 Integer, Double의 객체도 처리할 수 있다.

```
Box<Number> box = new Box<Number>();
box.add(new Integer(10));   // OK
box.add(new Double(10.1));  // OK
```

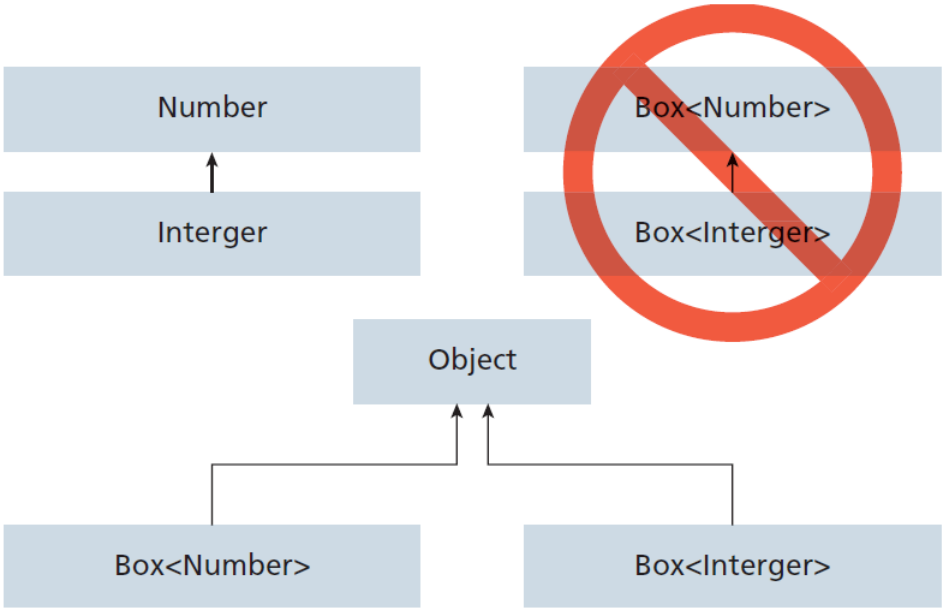
- 하지만 다음과 같은 generic method를 고려해보자. Box<Integer>나 Box<Double>을 받을 수 있을까?

```
public void sub(Box<Number> n) {   ...   }
```

```
public class Box<T> {
   private T data;
   public void set(T data) { this.data = data; }
   public T get() { return data; }
}
```

Generic과 상속

- Java 튜토리얼에 보면 다음과 같은 그림을 사용하여서 설명하고 있다

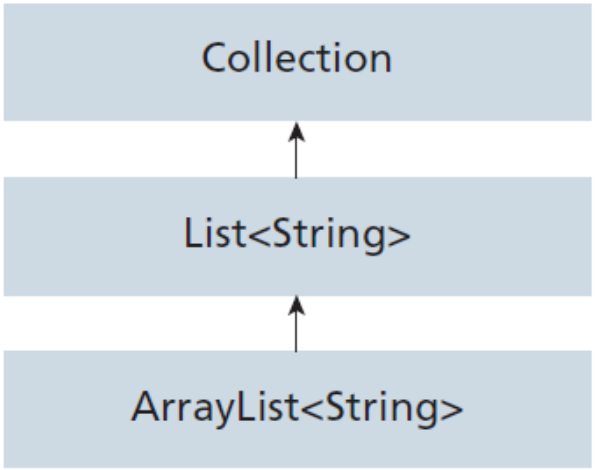


- Integer가 Number의 자식이긴 하지만, Box<Integer>는 Box<Number>의 자식은 아니다.

Generic Class의 상속

```
ArrayList<E> implements List<E> {   ...   }
List<E> extends Collection<E> {...}
```

- ArrayList<String>는 List<String>의 자식 class이다.
- List<String>은 Collection<String>의 자식 class이다.



한도가 있는 와일드 카드

- 물음표(?)는 와일드 카드(wild card)라고 불린다. 와일드 카드는 어떤 타입이든지 나타낼 수 있다.

```
public static double sumOfList(List<? extends Number> list) {
    double s = 0.0;

    for (Number n : list)
        s += n.doubleValue();
    return s;
}
```

Number 클래스의 모든 자식 클래스에 대하여 매치되는 와일드 카드이다.

… sumOfList(List<Number> list) { … }
무엇이 다른가?

- 위의 method는 다음과 같이 호출이 가능하다

```
List<Integer> li = Arrays.asList(1, 2, 3)
System.out.println("sum = " + sumOfList(li))
```

한도가 없는 와일드 카드

- 리스트 안의 모든 요소들을 출력하는 printList() method를 다음과 같이 작성해 보자.

```
public static void printList(List<Object> list) {
    for (Object elem : list)
        System.out.println(elem + " ")
    System.out.println();
}
```

무엇이 다른가?

- 올바르게 작성된 printList()는 다음과 같다.

```
public static void printList(List<?> list) {
    for (Object elem: list)
        System.out.print(elem + " ")
    System.out.println();
}
```

- 어떤 타입 A에 대하여 List<A>는 List<?>의 자손 class가 되므로 다음과 같이 printList()를 이용 하여서 다양한 타입의 리스트들을 출력할 수 있다.

```
List<Integer> li = Arrays.asList(1, 2, 3)
List<String>  ls = Arrays.asList("one", "two", "three")
printList(li);
printList(ls);
```

하한이 있는 와일드 카드

- List<Integer>, List<Number>, List<Object>와 같은 Integer 값을 가지고 있는 모든 객체에 대하여 작동시키려고 한다.

```
public static void addNumbers(List<? super Integer> list) {
    for (int i = 1 i <= 10 i++) {
        list.add(i);
    }
}
```

참고사항

제네릭은 상당히 복잡하다. 와일드 카드에 대하여 완전하게 학습하려면 자바 튜토리얼 사이트(java.sun.com)를 참조하기 바란다.

Collection

- Collection은 Java에서 자료 구조를 구현한 interface
- 자료 구조로는 리스트(list), 스택(stack), 큐(queue), 집합(set), 해쉬 테이블(hash table) 등이 있다.

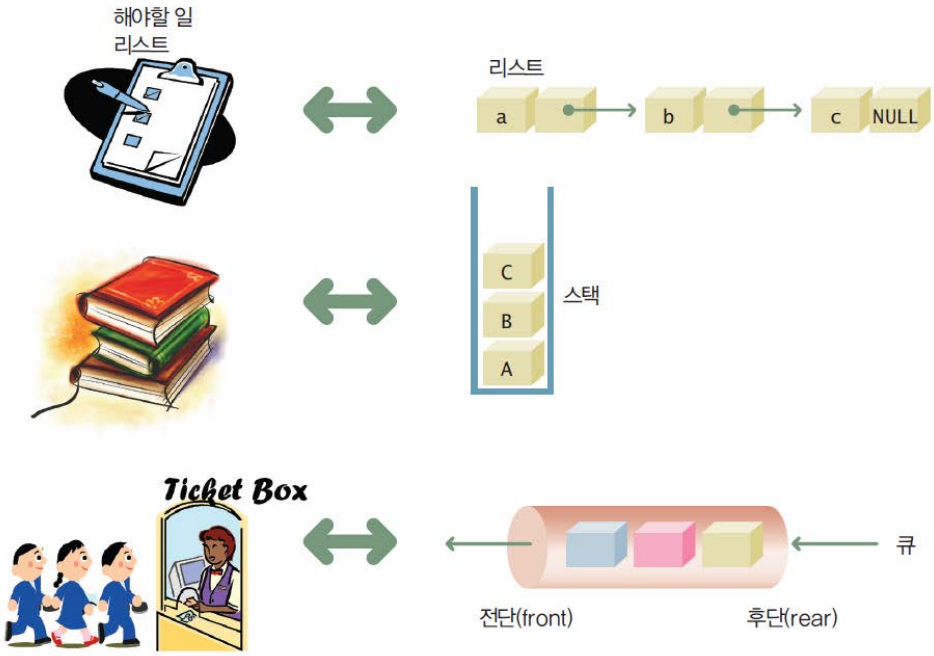
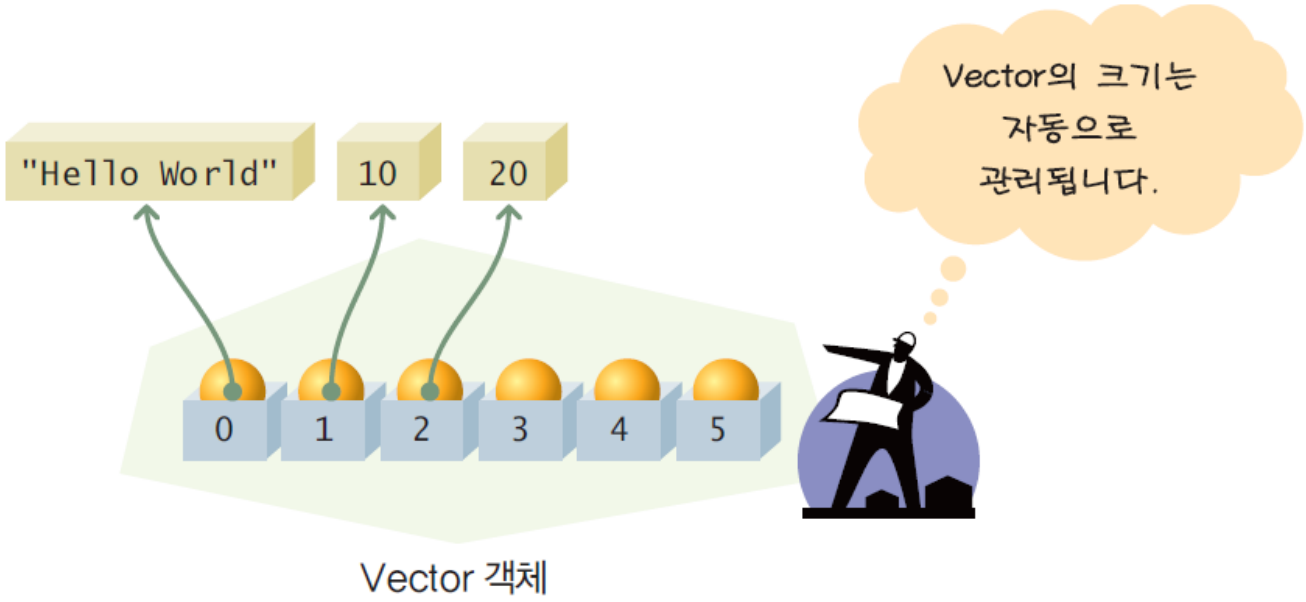


그림22-4. 자료 구조의 예

Collection의 예: Vector Class

- Vector class는 java.util 패키지에 있는 Collection의 일종으로 가변 크기의 배열(dynamic array)을 구현



예제

VectorTest.java

```
01  import java.util.Vector;
02
03  public class VectorTest {
04
05     public static void main(String[] args) {
06
07        Vector vc = new Vector();
08
09        vc.add("Hello World!");
10        vc.add(new Integer(10));
11        vc.add(20);
12
13        System.out.println("vector size :" + vc.size());
14
15        for (int i = 0; i < vc.size(); i++) {
16           System.out.println("vector element " + i + " :" + vc.get(i));
17        }
18        String s = (String)vc.get(0);
19
20     }
21  }
```

벡터 객체를 생성할 때, 크기를 안 주어도 된다. 물론 크기를 줄 수도 있다.

어떤 타입의 객체도 추가가 가능하다.

get()은 Object 타입으로 반환하므로 형변환하여서 사용한다.

실행결과

```
vector size :3
vector element 0 :Hello World!
vector element 1 :10
vector element 2 :20
```

Collection의 종류

인터페이스	설명
Collection	모든 자료 구조의 부모 인터페이스로서 객체의 모임을 나타낸다.
Set	집합(중복된 원소를 가지지 않는)을 나타내는 자료 구조
List	순서가 있는 자료 구조로 중복된 원소를 가질 수 있다.
Map	키와 값들이 연관되어 있는 사전과 같은 자료 구조
Queue	극장에서의 대기줄과 같이 들어온 순서대로 나가는 자료구조

중간점검

1. 컬렉션에는 어떤 것들이 있는가?
2. 컬렉션 클래스들은 어디에 이용하면 좋은가?

Collection Interface

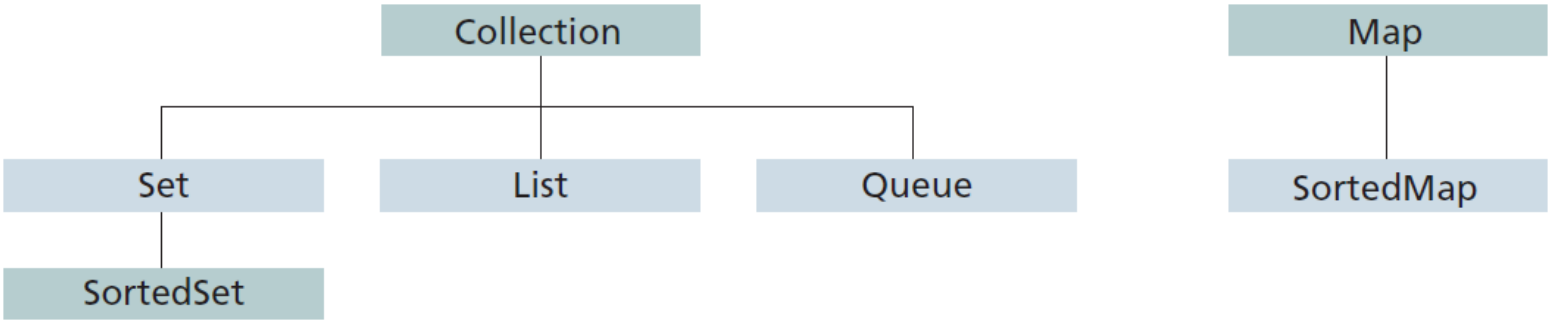


그림22-4. Interface들의 계층구조

Collection Interface

분류	메소드	설명
기본 연산	`int size()`	원소의 개수 반환
	`boolean isEmpty()`	공백 상태이면 true 반환
	`boolean contains(Object obj)`	obj를 포함하고 있으면 true 반환
	`boolean add(E element);`	원소 추가
	`boolean remove(Object obj)`	원소 삭제
	`Iterator<E> iterator();`	원소 방문
벌크 연산	`boolean addAll(Collection<? extends E> from)`	c에 있는 모든 원소 추가
	`boolean containsAll(Collection<?> c)`	c에 있는 모든 원소가 포함되어 있으면 true
	`boolean removeAll(Collection<?> c)`	c에 있는 모든 원소 삭제
	`void clear()`	모든 원소 삭제
배열 연산	`Object[] toArray()`	컬렉션을 배열로 변환
	`<T> T[] toArray(T[] a);`	컬렉션을 배열로 변환

와일드 카드

중간점검

중간점검

1. Collection 인터페이스의 각 메소드들의 기능을 자바 API 웹페이지를 이용하여서 조사하여 보자.

ArrayList

그림22-5. 리스트

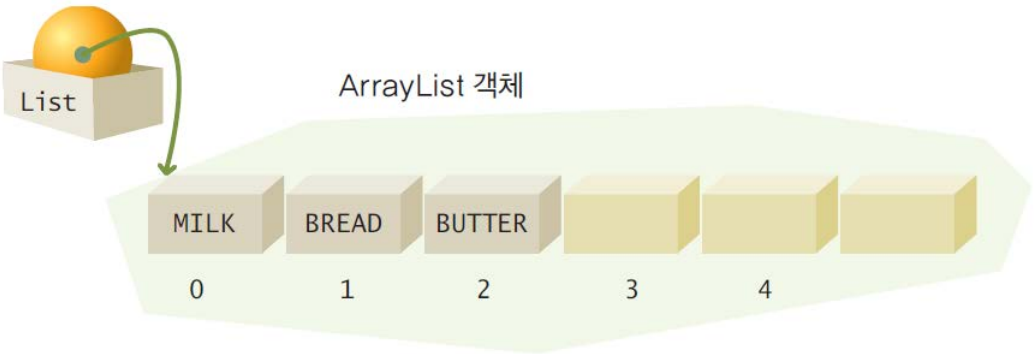
ArrayList의 기본 연산

- ArrayList 는 타입 매개변수를 가지는 generic class로 제공된다.

```
ArrayList<String> list = new ArrayList<String>();
```

- 생성된 ArrayList 객체에 데이터를 저장하려면 add() method를 사용한다. add() method는 Collection interface에 정의된 method로서 ArrayList class가 구현한 method이다.

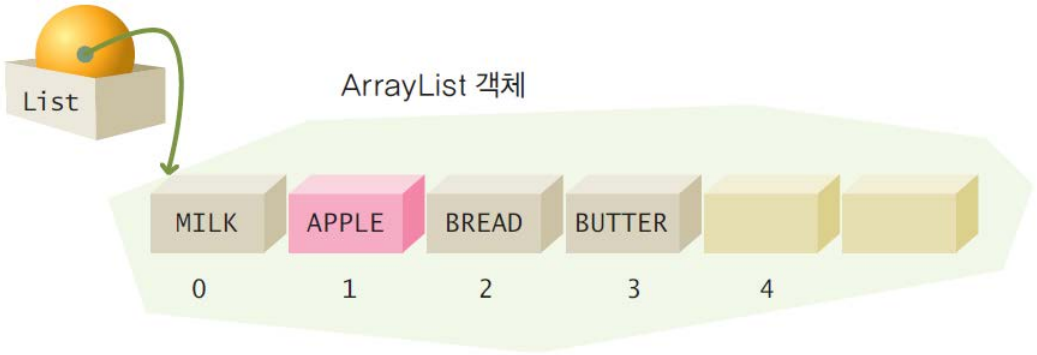
```
list.add( "MILK" );
list.add( "BREAD" );
list.add( "BUTTER" );
```



ArrayList의 기본 연산

- 만약에 기존의 데이터가 들어 있는 위치를 지정하여서 add()를 호출하면 새로운 데이터는 중간에 삽 입된다.

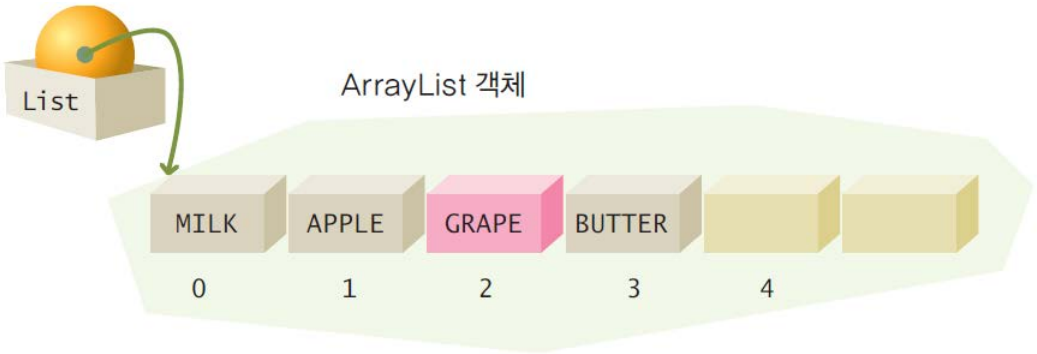
```
list.add( 1, "APPLE" );      // 인덱스 1에 "APPLE"을 삽입
```



ArrayList의 기본 연산

- 만약 특정한 위치에 있는 원소를 바꾸려면 set() method를 사용한다.

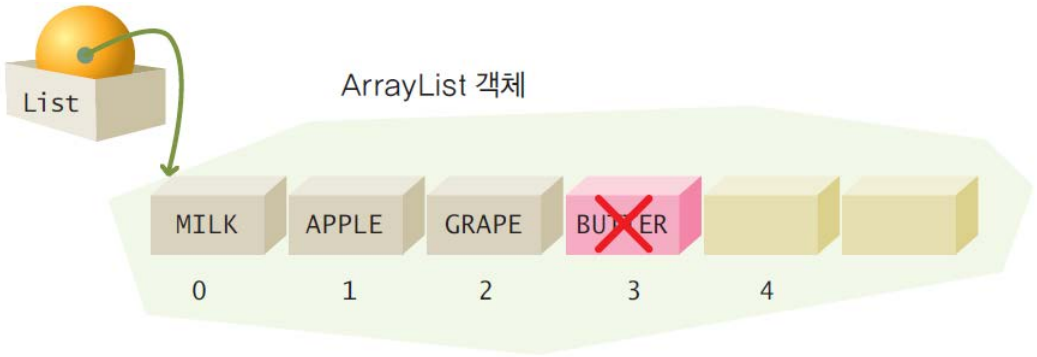
```
list.set( 2, "GRAPE" );      // 인덱스 2의 원소를 "GRAPE"로 대체
```



ArrayList의 기본 연산

- 데이터를 삭제하려면 remove() method를 사용한다.

```
list.remove( 3 );          // 인덱스 3의 원소를 삭제한다.
```



- ArrayList 객체에 저장된 객체를 가져오는 method는 get()이다. get()은 인덱스를 받아서 그 위치에 저장된 원소를 반환한다. 예를 들어서 list.get(1)이라고 하면 인덱스 1에 저장된 데이터가 반환된다.

```
String s = list.get(1);
```

예제

ArrayListTest.java

```
01  import java.util.*;
02
03  public class ArrayListTest {
04     public static void main(String args[]) {
05        ArrayList<String> list = new ArrayList<String>();
06
07        list.add("MILK");
08        list.add("BREAD");
09        list.add("BUTTER");
10        list.add(1, "APPLE");        // 인덱스 1에 "APPLE"을 삽입
11        list.set(2, "GRAPE");        // 인덱스 2의 원소를 "GRAPE"로 대체
12        list.remove(3);              // 인덱스 3의 원소를 삭제한다.
13
14        for (int i = 0; i < list.size(); i++)
15           System.out.println(list.get(i));
16     }
17  }
```

String 타입의 객체를 저장할 수 있는 ArrayList 객체 생성

```
for (String s : list)
   System.out.println(s);
```

실행결과

```
MILK
APPLE
GRAPE
```

ArrayList의 추가 연산

- ArrayList는 동일한 데이터가 여러 번 저장될 수 있으므로, indexOf() method는 맨 처음에 있는 데이터의 위치가 반환된다.

```
int index = list.indexOf("APPLE");    // 1이 반환된다.
```

- 검색을 반대 방향으로 하려면 lastIndexOf()를 사용한다.

```
int index = list.lastIndexOf("MILK");    // 0이 반환된다.
```

참고사항

불행하게도 자바에서는 배열, ArrayList, 문자열 객체의 크기를 알아내는 방법이 약간 다르다.

- 배열: array.length
- ArrayList: arrayList.size()
- 문자열: string.length()

반복자 사용하기

- ArrayList에 있는 원소에 접근하는 또 하나의 방법은 반복자**(iterator)**를 사용하는 것이다.
- 반복자(iterator)
  - iterator() method에 의해서 return 된다.

메소드	설명
hasNext()	아직 방문하지 않은 원소가 있으면 true를 반환
next()	다음 원소를 반환
remove()	최근에 반환된 원소를 삭제한다.

반복자 사용하기

- 반복자 객체의 hasNext()와 next() method를 이용해서 Collection의 각 원소들을 접근할 수 있다.

```
ArrayList<String> list = new ArrayList<String>();
list.add("하나");
list.add("둘");
list.add("셋");
list.add("넷");

String s;
Iterator e = list.iterator();
while(e.hasNext())                                    ← 다음 원소가 있으면
{
    s = (String)e.next();          // 반복자는 Object 타입을 반환!
    System.out.println(s);                            ← 다음 원소를 얻는다.
}
```

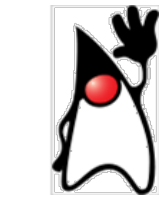
중간점검

참고사항

반복자 사용을 보다 간편하게 한 것이 버전 1.5부터 도입된 for-each 루프이다. 반복자보다는 for-each 루프가 간편하지만 아직도 반복자는 널리 사용되고 있다. 따라서 그 작동 원리를 알아야 한다.

중간점검

1. ArrayList가 기존의 배열보다 좋은 점은 무엇인가?
2. ArrayList의 부모 클래스는 무엇인가?
3. 왜 인터페이스 참조 변수를 이용하여서 컬렉션 객체들을 참조할까?
4. ArrayList 안의 객체들을 반복 처리하는 방법들을 모두 설명하라.



LinkedList

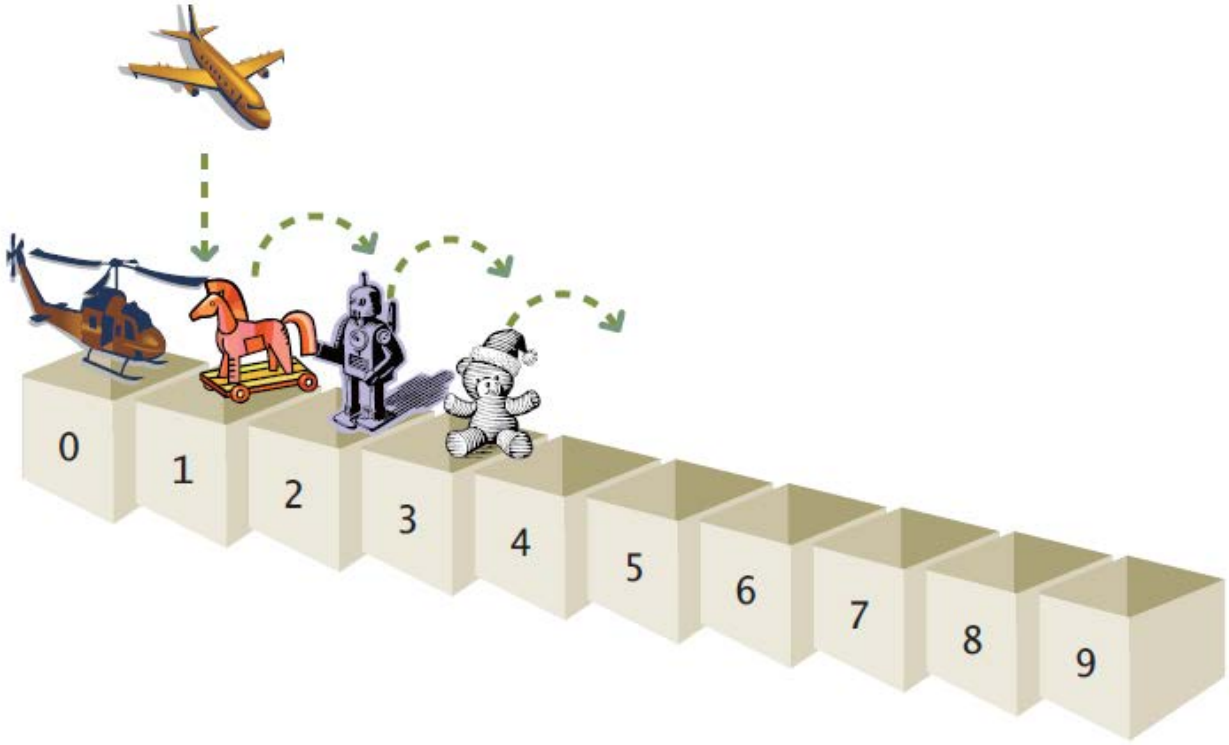
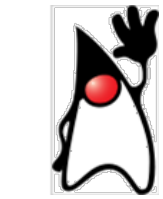


그림22-6. 배열의 중간에 삽입하려면 원소들을 이동하여야 한다.



LinkedList

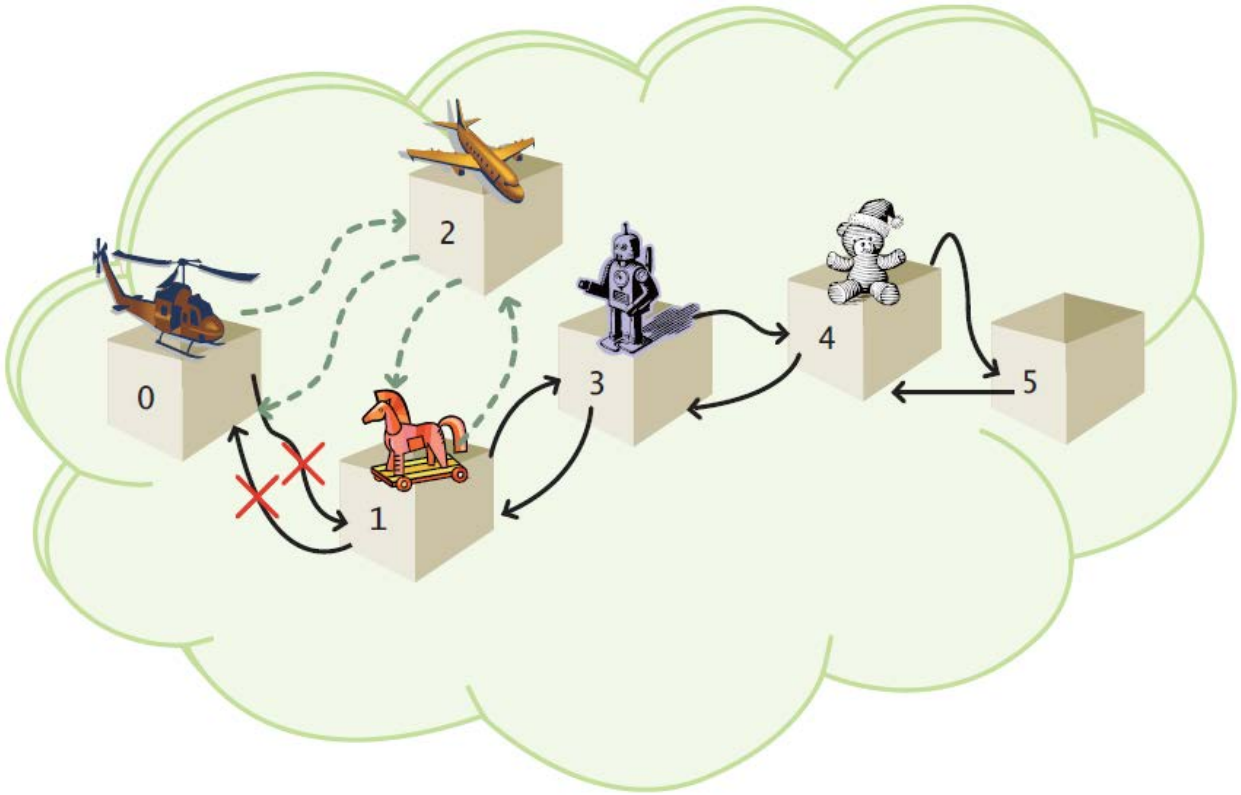


그림22-7. 연결 리스트 중간에 삽입하려면 링크만 수정하면 된다.

예제

LinkedListTest.java

```
01  import java.util.*;
02
03  public class LinkedListTest {
04     public static void main(String args[]) {
05        LinkedList<String> list = new LinkedList<String>();
06
07        list.add("MILK");
08        list.add("BREAD");
09        list.add("BUTTER");
10        list.add(1, "APPLE");        // 인덱스 1에 "APPLE"을 삽입
11        list.set(2, "GRAPE");        // 인덱스 2의 원소를 "GRAPE"로 대체
12        list.remove(3);              // 인덱스 3의 원소를 삭제한다.
13
14        for (int i = 0; i < list.size(); i++)
15           System.out.println(list.get(i));
16     }
17  }
```

실행결과

```
MILK
APPLE
GRAPE
```

반복자 사용하기

- LinkedList도 반복자를 지원한다. 다음과 같은 형식으로 사용한다.

```
Iterator e = list.iterator();
String first = e.next();      // 첫 번째 원소
String second = e.next();     // 두 번째 원소
e.remove();                   // 최근 방문한 원소 삭제
```

- ArrayList나 LinkedList와 같은 리스트에서 사용하기가 편리한 반복자는 다음과 같이 정의되는 ListIterator이다.

```
interface ListIterator<E> extends Iterator<E>
{
   void add(E element);
   E previous();
   boolean hasPrevious();
   ...
}
```

배열을 리스트로 변경하기

- Arrays.asList() method는 배열을 받아서 리스트 형태로 반환한다.

```
List<String> list = Arrays.asList(new String[size]);
```

중간점검

1. ArrayList와 LinkedList의 차이점은 무엇인가?
2. 어떤 경우에 LinkedList를 사용하여야 하는가?

Set

- 집합(Set)은 원소의 중복을 허용하지 않는다.

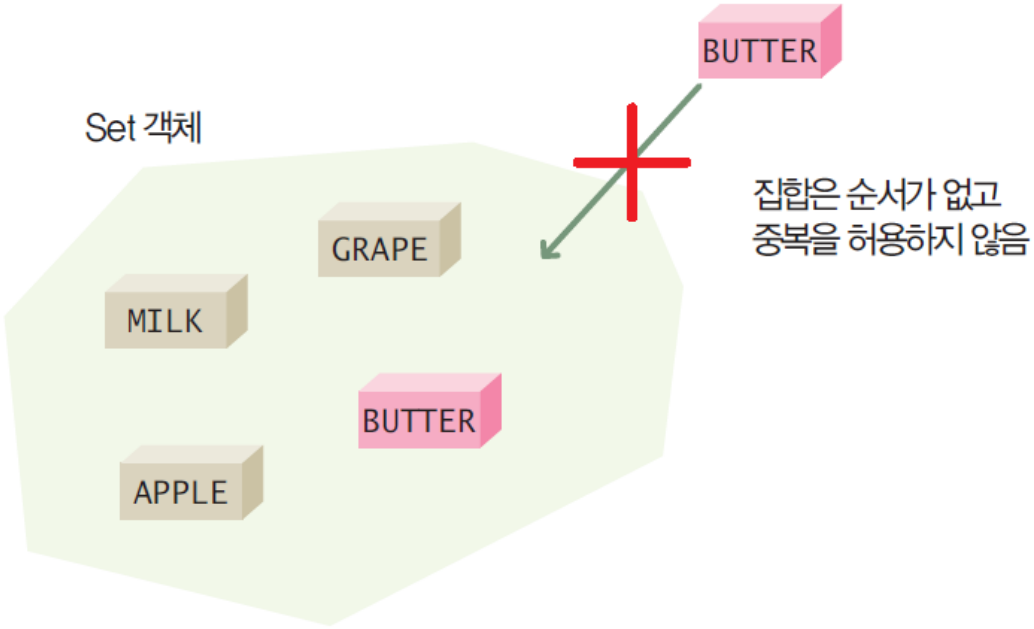


그림22-8. 집합

Set

- 집합을 구현하는 방법으로 가장 잘 알려진 방법이 hash table이다.
- Java에서 hash table은 linked list로 구현된다.

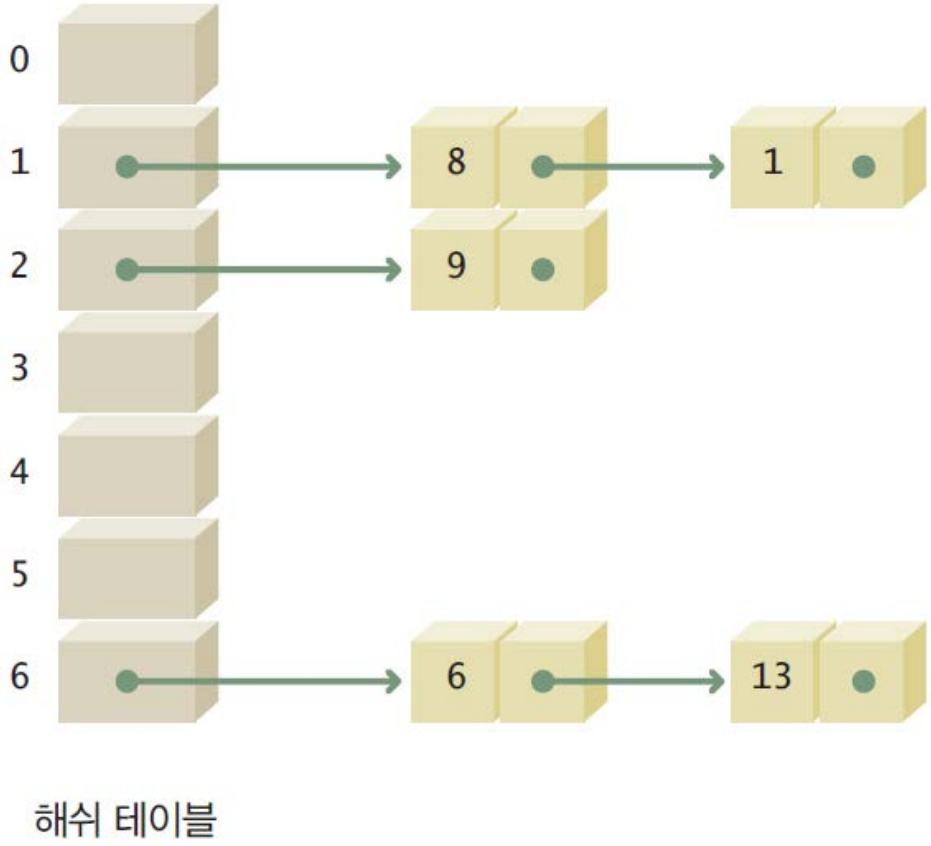


그림22-9. 해쉬 테이블

Set Interface를 구현하는 Classes

- HashSet
  - HashSet은 해쉬 테이블에 원소를 저장하기 때문에 성능면에서 가장 우수하다. 하지만 원소들의 순서가 일정하지 않은 단점이 있다.
- TreeSet
  - 레드-블랙 트리(red-black tree)에 원소를 저장한다. 따라서 값에 따라서 순서가 결정되며 하지만 HashSet보다는 느리다.
- LinkedHashSet
  - 해쉬 테이블과 연결 리스트를 결합한 것으로 원소들의 순서는 삽입되었던 순서와 같다.

예제

SetTest.java

```
01  import java.util.*;
02
03  public class SetTest {
04     public static void main(String args[]) {
05        HashSet<String> set = new HashSet<String>();
06
07        set.add("Milk");
08        set.add("Bread");
09        set.add("Butter");
10        set.add("Cheese");
11        set.add("Ham");
12        set.add("Ham");
13
14        System.out.println(set);
15     }
16  }
```

HashSet인 경우의 출력 결과: [Bread, Milk, Butter, Ham, Cheese]
LinkedHashSet인 경우의 출력 결과: [Milk, Bread, Butter, Cheese, Ham]
TreeSet인 경우의 출력 결과: [Bread, Butter, Cheese, Ham, Milk] ※ sort되어 있음

예제

FindDupplication.java

```
01  import java.util.*;
02
03  public class FindDupplication {
04     public static void main(String[] args) {
05        Set<String> s = new HashSet<String>();
06        String[] sample = { "단어", "중복", "구절", "중복" };
07        for (String a : sample)
08           if (!s.add(a))
09              System.out.println("중복된 단어 " + a);
10
11           System.out.println(s.size() + " 중복되지 않은 단어: " + s);
12     }
13  }
```

실행결과

```
중복된 단어 중복
3 중복되지 않은 단어: [중복, 구절, 단어]
```

대량 연산 Method

- s1.containsAll(s2): 만약 s2가 s1의 부분 집합이면 참이다.

- s1.addAll(s2): s1을 s1과 s2의 합집합으로 만든다.
  s1이 변함
- s1.retainAll(s2): s1을 s1과 s2의 교집합으로 만든다.
  s1이 변함
- s1.removeAll(s2): s1을 s1과 s2의 차집합으로 만든다.
  s1이 변함
- String 타입의 집합 s1과 s2의 합집합을 구하려고 한다. s1과 s2가 변하지 않게 하려면, 먼저 s1을 복사해서 여기에 s2를 더해야 한다.

```
Set<String> union = new HashSet<String>(s1);
union.addAll(s2);
```

예제

SetTest1.java

```
01  import java.util.*;
02
03  public class SetTest1 {
04     public static void main(String[] args) {
05      Set<String> s1 = new HashSet<String>();
06      Set<String> s2 = new HashSet<String>();
07
08      s1.add("A");
09      s1.add("B");
10      s1.add("C");
11
12      s2.add("A");
13      s2.add("D");
14
15      Set<String> union = new HashSet<String>(s1);
16      union.addAll(s2);
17
18      Set<String> intersection = new HashSet<String>(s1);
19      intersection.retainAll(s2);
20
21      System.out.println("합집합 " + union);
22      System.out.println("교집합 " + intersection);
23     }
24  }
```

실행결과

실행결과

```
합집합 [D, A, B, C]
교집합 [A]
```

중간점검

1. Set은 어떤 타입의 애플리케이션에 유용한가?
2. Set과 List의 차이점은 무엇인가?

Queue

- Queue는 먼저 들어온 데이터가 먼저 나가는 자료 구조
- FIFO(First-In First-Out)

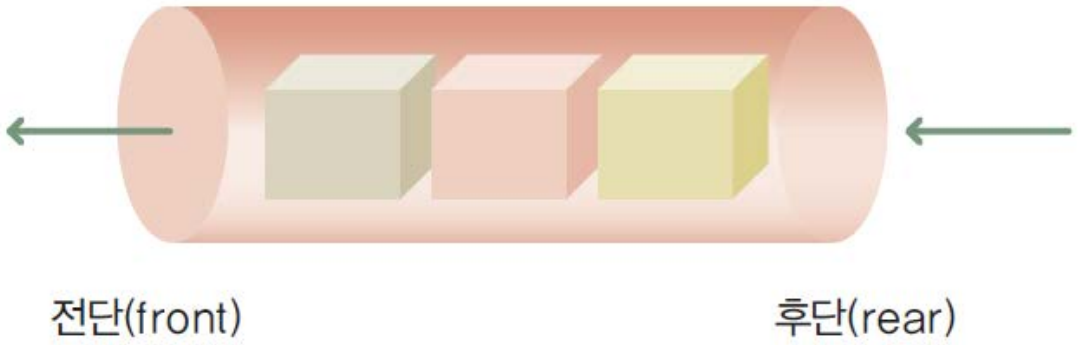


그림22-10. Queue

Queue Interface

```
public interface Queue<E> extends Collection<E> {
    E element();
    boolean offer(E e);
    E peek();
    E poll();
    E remove();
}
```

예제

QueueTest.java

```
01  import java.util.*;
02
03  public class QueueTest {
04     public static void main(String[] args) throws InterruptedException {
05        int time = 10;
06        Queue<Integer> queue = new LinkedList<Integer>();
07        for (int i = time; i >= 0; i--)
08           queue.add(i);
09        while (!queue.isEmpty()) {
10           System.out.print(queue.remove()+" ");
11           Thread.sleep(1000);       // 현재의 스레드를 1초간 재운다.
12        }
13     }
14  }
```

Integer를 저장하는 큐를 생성한다.
실제로는 LinkedList 안에 Queue 인터페이스가 구현되어 있다.

실행결과

```
10 9 8 7 6 5 4 3 2 1 0
```

우선순위 Queue

- 우선순위 Queue는 원소들이 무작위로 삽입되었더라도 정렬된 상태로 원소들을 추출한다.
- remove()를 호출할 때마다 가장 작은 원소가 추출된다.
- 우선순위 queue는 heap이라고 하는 자료 구조를 내부적으로 사용한다.
- Heap은 이진 tree의 일종으로서 add()와 remove()를 호출하면 가장 작은 원소가 효율적으로 tree의 root로 이동하게 된다.
- 우선순위 queue의 가장 대표적인 예는 작업 스케쥴링(job scheduling)이다. 각 작업은 우선순위를 가지고 있고 가장 높은 우선순위의 작업이 queue에서 먼저 추출되어서 시작된다

예제

PriorityQueueTest.java

```
01  import java.util.*;
02
03  public class PriorityQueueTest {
04     public static void main(String[] args) {
05        PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
06        pq.add(30);
07        pq.add(80);
08        pq.add(20);
09
10        for (Integer o : pq)
11           System.out.println(o);
12        System.out.println("원소 삭제");
13        while (!pq.isEmpty())
14           System.out.println(pq.remove());
15     }
16  }
```

우선순위 큐 생성

실행결과

```
20
80
30
원소 삭제
20
30
80
```

Map

- 사전과 같은 자료 구조
- 키(key)에 값(value)이 매핑된다.

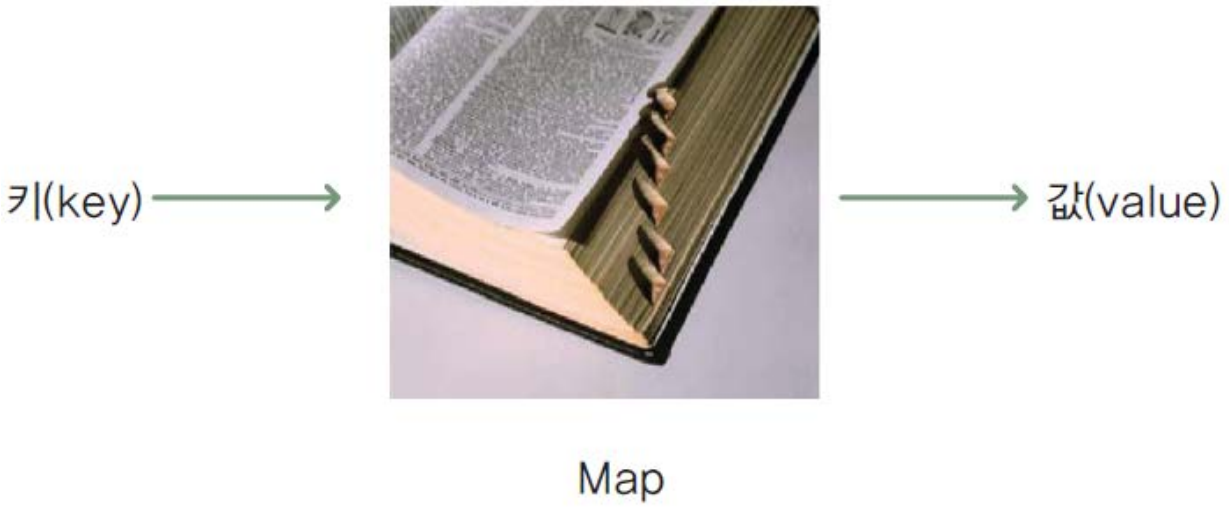


그림22-11. Map의 개념

예제#1

```
class Student {
    int number;
    String name;
    public Student(int number, String name) {
        this.number = number;
        this.name = name;
    }
    public String toString() { return name; }
}

public class MapTest {
    public static void main(String[] args) {
        Map<String, Student> st = new HashMap<String, Student>();
        st.put("20090001", new Student(20090001, "구준표"));
        st.put("20090002", new Student(20090002, "금잔디"));
        st.put("20090003", new Student(20090003, "윤지후"));

        System.out.println(st);   // 모든 항목을 출력한다.
        st.remove("20090002"); // 하나의 항목을 삭제한다.
        st.put("20090003", new Student(20090003, "소이정")); // 하나의 항목을 대치한다.
        System.out.println(st.get("20090003")); // 값을 참조한다.
        // 모든 항목을 방문한다.
        for (Map.Entry<String, Student> s : st.entrySet()) {
            String key = s.getKey();
            Student value = s.getValue();
            System.out.println("key=" + key + ", value=" + value);
        }
    }
}
```

실행결과

```
{20090001=구준표, 20090002=금잔디, 20090003=윤지후}
소이정
key=20090001, value=구준표
key=20090003, value=소이정
```

Map에 저장된 데이터를 방문할 때는 Map.Entry라는 interface를 사용한다.

예제#2

WordFreq.java

```
01  import java.util.*;
02
03  public class WordFreq {
04     public static void main(String[] args) {
05        Map<String, Integer> m = new HashMap<String, Integer>();
06
07        String[] sample = { "to", "be", "or", "not", "to", "be", "is", "a", "problem" };
08        // 문자열에 포함된 단어의 빈도를 계산한다.
09        for (String a : sample) {
10           Integer freq = m.get(a);
11           m.put(a, (freq == null) ? 1 : freq + 1);
12        }
13
14        System.out.println(m.size() + " 단어가 있습니다.");
15        System.out.println(m.containsKey("to"));
16        System.out.println(m.isEmpty());
17        System.out.println(m);
18     }
19  }
```

Map 객체 생성

단어를 꺼내서 빈도를 증가시킨다.

실행결과

```
7 단어가 있습니다.
true
false
{not=1, to=2, is=1, or=1, a=1, problem=1, be=2}
```

먼저 String 배열에서 조건 연산자를 사용하여 만약 단어가 한 번도 등장한 적이 없으면 1로 설정한다. 만약 한 번이라도 등장하였으면 빈도를 나타내는 값을 하나 증가시킨다.

중간점검

중간점검

1. Map의 각 원소들은 _____와 _____의 두 부분으로 구성되어 있다.
2. Map의 두 가지의 기본적인 연산은 무엇인가?

Collections Class

- Collections Class
  - 여러 유용한 algorithm을 구현한 static method을 제공한다.
  - Method들은 generic method이다.
  - Method의 첫 번째 매개변수는 알고리즘이 적용되는 collection이다.
- 중요한 algorithm: 정렬(Sorting), 섞기(Shuffling), 탐색(searching)

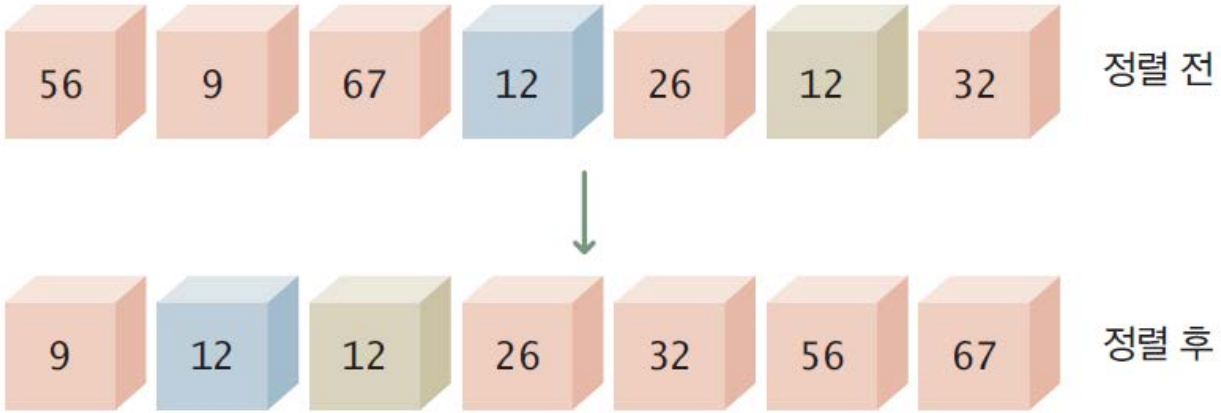


그림22-12. 안정된 정렬

Collections Class

- Collections class의 sort() method는 List interface를 구현하는 Collection에 대하여 정렬을 수행한다.

```
List<String> list = new LinkedList<String>();
list.add("김철수");
list.add("김영희");
Collections.sort(list);      // 리스트 안의 문자열이 정렬된다.
```

정렬의 예제#1

- 문자열을 정렬하는 간단한 예를 살펴보자.

Sort.java

```
01  import java.util.*;
02
03  public class Sort {
04     public static void main(String[] args) {
05        String[] sample = { "i", "walk", "the", "line" };
06        List<String> list = Arrays.asList(sample);          // 배열을 리스트로 변경
07        Collections.sort(list);
08        System.out.println(list);
09     }
10  }
```

리스트를 정렬한다.

실행결과

```
[i, line, the, walk]
```

프로그램설명

정렬 알고리즘을 실행하기 위하여 asList() 메소드를 이용하여 배열을 리스트로 변환한다. Collections 인터페이스가 가지고 있는 정적 메소드인 sort()을 호출하여서 정렬을 수행한다.

정렬의 예제#2

```
class Student implements Comparable<Student> {
    int number;
    String name;

    public Student(int number, String name) {
        this.number = number;
        this.name = name;
    }

    public String toString() {
        return name;
    }

    public int compareTo(Student s) {
        return number - s.number;
    }
}

public class SortTest {
    public static void main(String[] args) {
        Student array[] = { new Student(20090001, "김철수"),
                            new Student(20090002, "이철수"), new Student(20090003, "박철수"), };
        List<Student> list = Arrays.asList(array);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

실행결과

```
[김철수, 이철수, 박철수]
```

만약 역순으로 정렬하기를 원한다면 다음과 같이 하면 된다.
Collections.sort(list, Collections.reverseOrder())

섞기(Shuffling)

Shuffle.java

```
01  import java.util.*;
02
03  public class Shuffle {
04     public static void main(String[] args) {
05        List<Integer> list = new ArrayList<Integer>();
06        for (int i = 1; i <= 10; i++)
07           list.add(i);
08        Collections.shuffle(list);
09        System.out.println(list);
10     }
11  }
```

순서를 무작위로 만든다.

실행결과

```
[5, 9, 7, 3, 6, 4, 8, 2, 1, 10]
```

탐색(Searching)

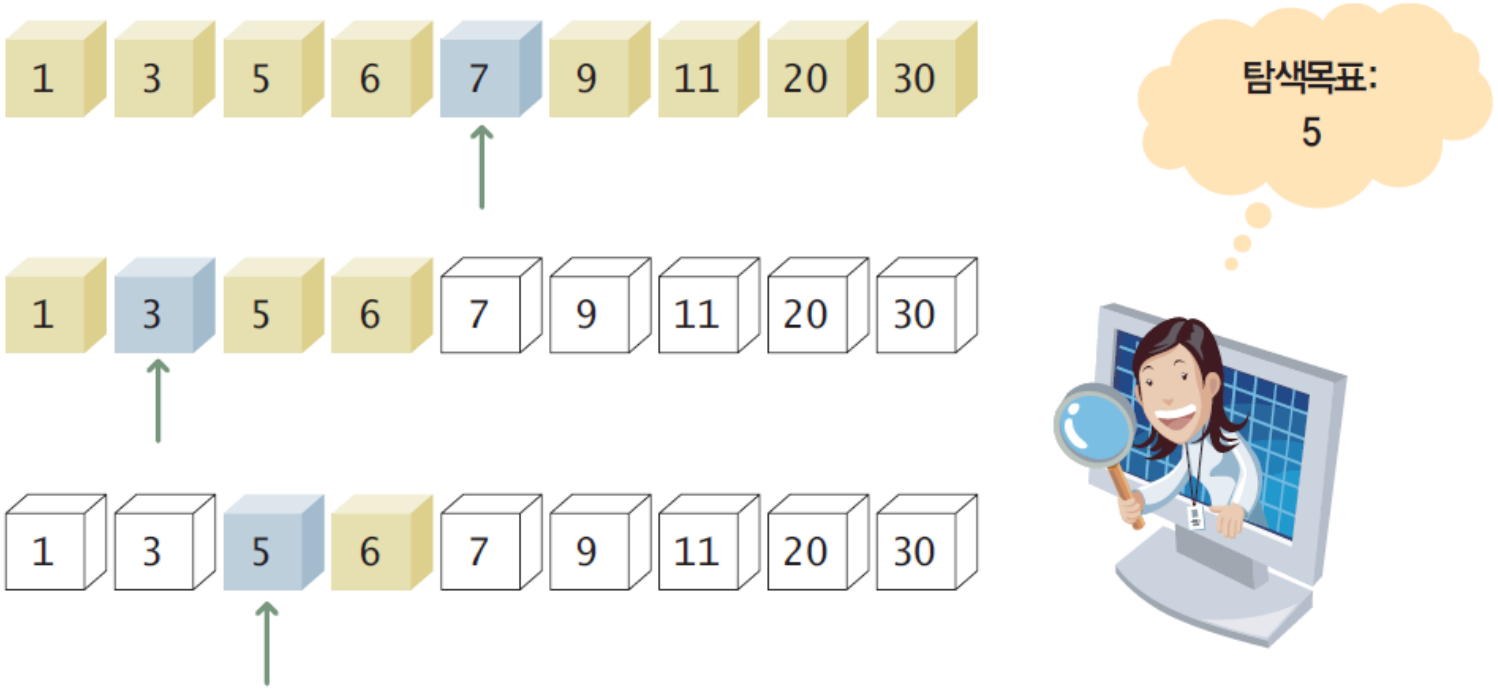


그림22-13. Generic 프로그래밍의 개념

탐색(Searching)

Search.java

```
01  import java.util.*;
02
03  public class Search {
04     public static void main(String[] args) {
05        int key = 50;
06        List<Integer> list = new ArrayList<Integer>();
07        for (int i = 0; i < 100; i++)
08           list.add(i);
09        int index = Collections.binarySearch(list,key);
10        System.out.println("탐색의 반환값  =" + index);
11     }
12  }
```

이진 탐색하여 list에서 key의 위치를 반환한다.

실행결과

```
탐색의 반환값  =50
```

Collections Class의 기타 Method

- min(), max(): 리스트에서 최대값과 최소값을 찾는다.
- reverse(): 리스트의 원소들의 순서를 반대로 한다.
- fill(): 지정된 값으로 리스트를 채운다.
- copy(): 목적 리스트와 소스 리스트를 받아서 소스를 목적지로 복사한다
- swap(): 리스트의 지정된 위치의 원소들을 서로 바꾼다.
- addAll(): Collection 안의 지정된 모든 원소들을 추가한다
- frequency(): 지정된 Collection에서 지정된 원소가 얼마나 많이 등장하는 지를 반환한다
- disjoint(): 두 개의 Collection이 겹치지 않는지를 검사한다

중간점검

1. 배열에 저장된 데이터를 Collections 클래스를 사용하여서 정렬하는 절차를 설명하라.
2. Collections 클래스의 기타 메소드의 매개변수와 반환값들을 조사하여 보라.

Q & A