컬렉션

- 컬렉션(collection)은 자바에서 자료 구조를 구현한 클래스
- 자료 구조로는 리스트(list), 스택(stack), 큐(queue), 집합(set), 해쉬 테이블(hash table) 등이 있다.

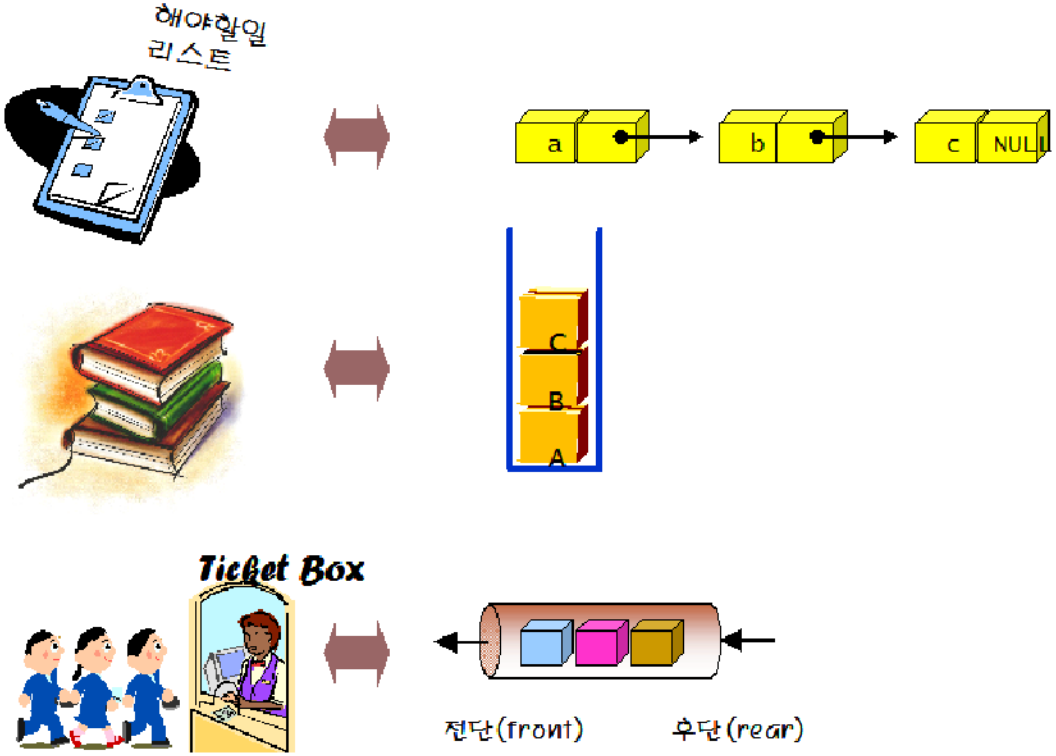
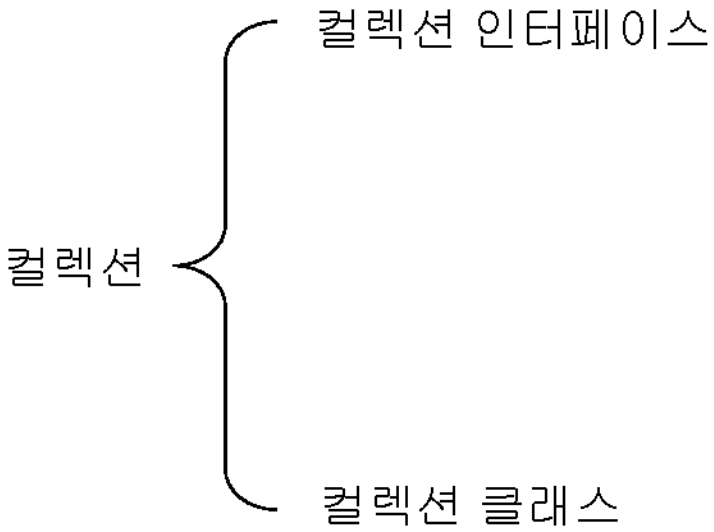


그림 14-4 자료 구조의 예

자바에서의 컬렉션



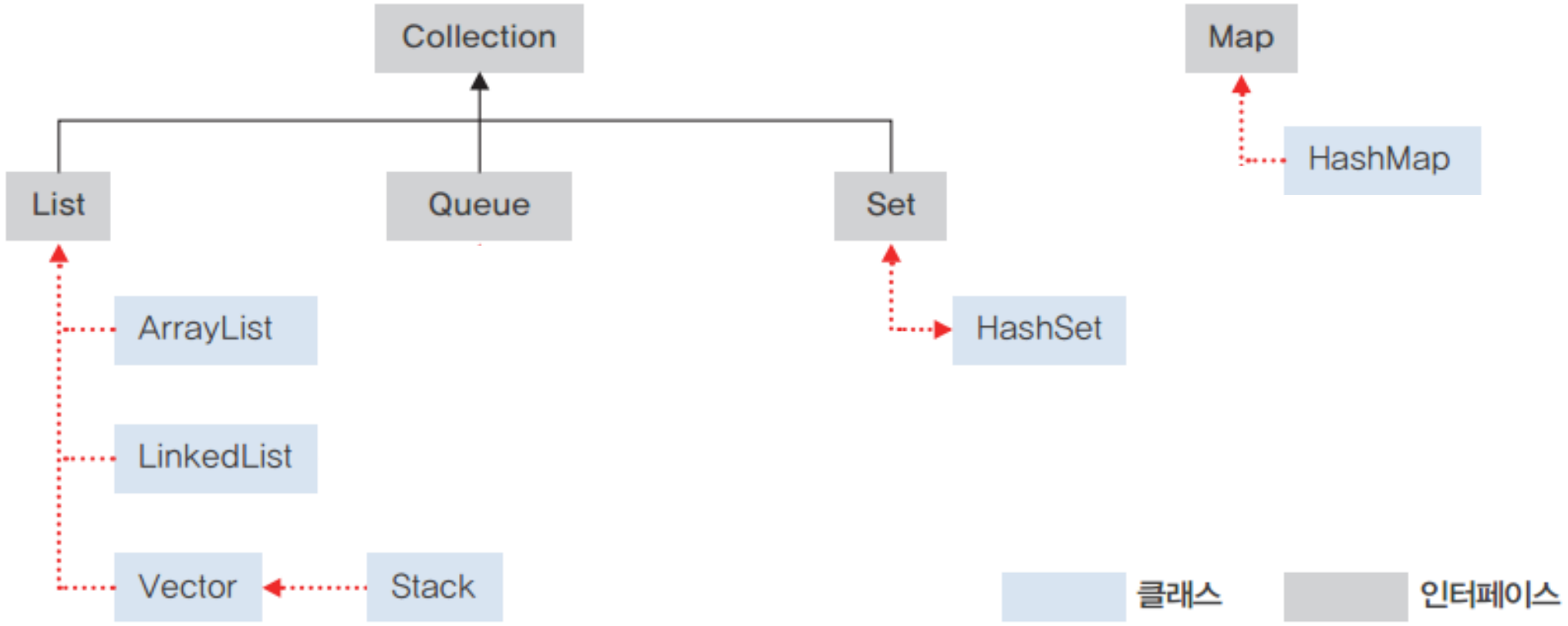
컬렉션 인터페이스

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

컬렉션의 역사

- 초기 버전: Vector, Stack, HashTable, Bitset, Enumeration이 그것이다.
- 버전 1.2부터는 풍부한 컬렉션 라이브러리가 제공
  - 인터페이스와 구현을 분리
  - (예) List 인터페이스를 ArrayList와 LinkedList 클래스가 구현

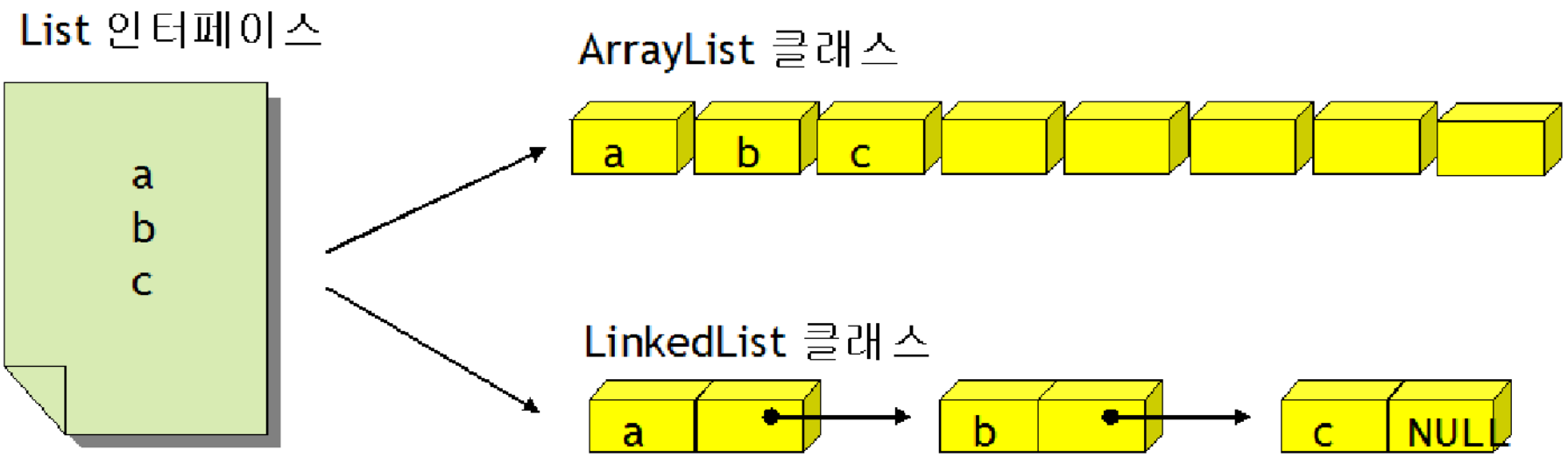
컬렉션 인터페이스


Collection
Map
HashMap
List
Queue
Set
ArrayList
HashSet
LinkedList
Vector
Stack
클래스
인터페이스

인터페이스가 제공하는 메소드

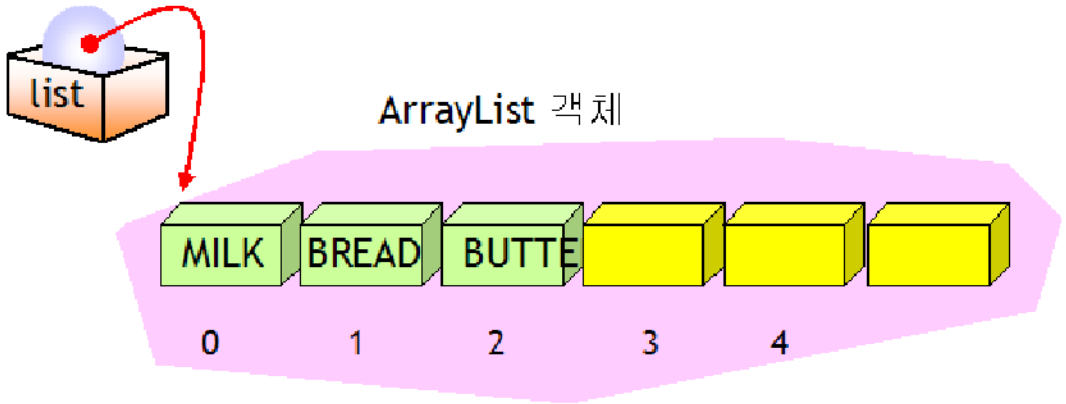
분류	메소드	설명
기본 연산	int size()	원소의 개수 반환
	boolean isEmpty()	공백 상태이면 true 반환
	boolean contains(Object obj)	obj를 포함하고 있으면 true 반환
	boolean add(E element);	원소 추가
	boolean remove(Object obj)	원소 삭제
	Iterator<E> iterator();	원소 방문
벌크 연산	boolean addAll(Collection<? extends E> from)	c에 있는 모든 원소 추가
	boolean containsAll(Collection<?> 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

List 인터페이스

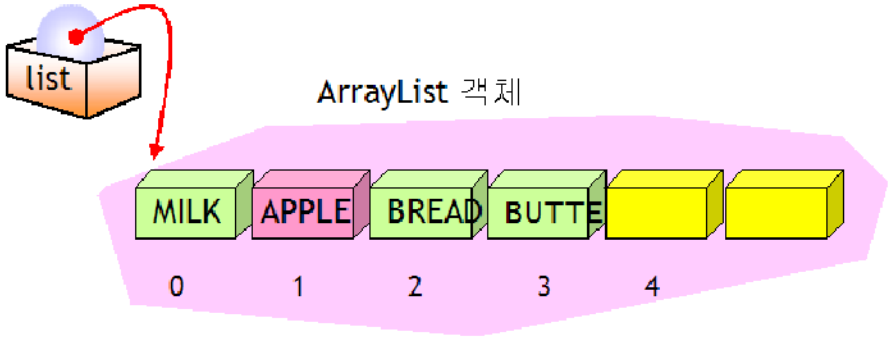


ArrayList

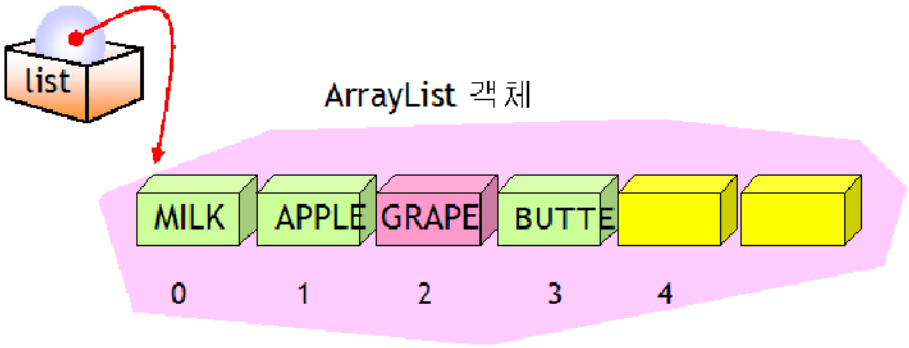
- ArrayList를 배열(Array)의 향상된 버전 또는 가변 크기의 배열이라고 생각하면 된다.
- ArrayList의 생성
  - ArrayList<String> list = **new** ArrayList<String>();
- 원소 추가
  - list.add("MILK");
  - list.add("BREAD");
  - list.add("BUTTER");



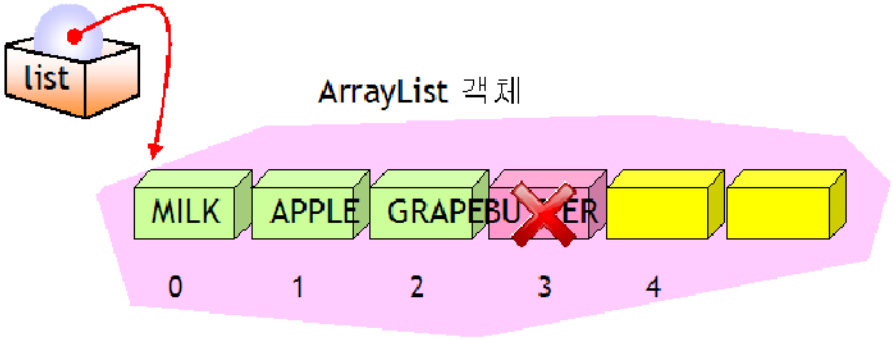
```
list.add( 1, "APPLE" );          // 인덱스 1에 "APPLE"을 삽입
```




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




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



예제

ArrayListTest.java

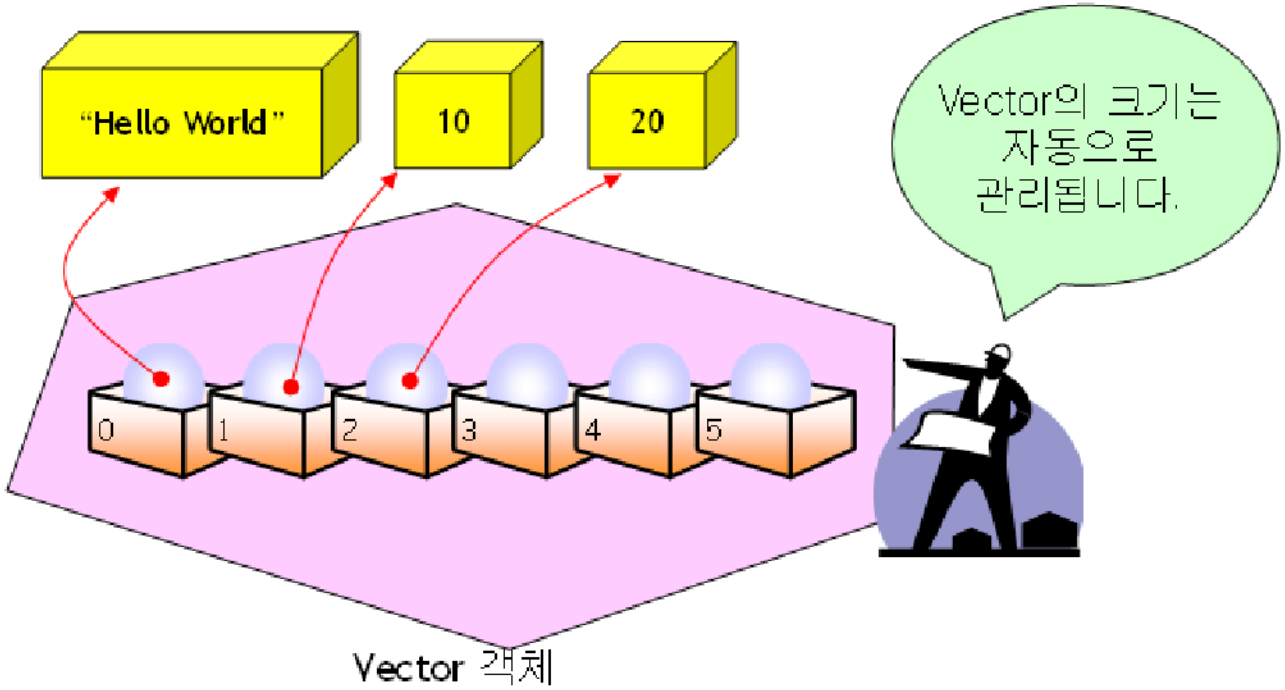
```
import java.util.*;

public class ArrayListTest {
        public static void main(String args[]) {
                ArrayList<String> list = new ArrayList<String>();

                list.add("MILK");
                list.add("BREAD");
                list.add("BUTTER");
                list.add(1, "APPLE"); // 인덱스 1에 "APPLE"을 삽입
                list.set(2, "GRAPE"); // 인덱스 2의 원소를 "GRAPE"로 대체
                list.remove(3); // 인덱스 3의 원소를 삭제한다.

               for (int i = 0; i < list.size(); i++)
                        System.out.println(list.get(i));
        }
}
```

실행결과

```
MILK
APPLE
GRAPE
```

컬렉션의 예: Vector 클래스

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



예제

```
import java.util.Vector;
public class VectorTest {
        public static void main(String[] args) {
                Vector vc = new Vector();
                vc.add("Hello World!");
                vc.add(new Integer(10));
                vc.add(20);
                System.out.println("vector size :" + vc.size());
                for (int i = 0; i < vc.size(); i++) {
                  System.out.println("vector element " + i + " :" + vc.get(i));
                }
                String s = (String)vc.get(0);
        }
}
```

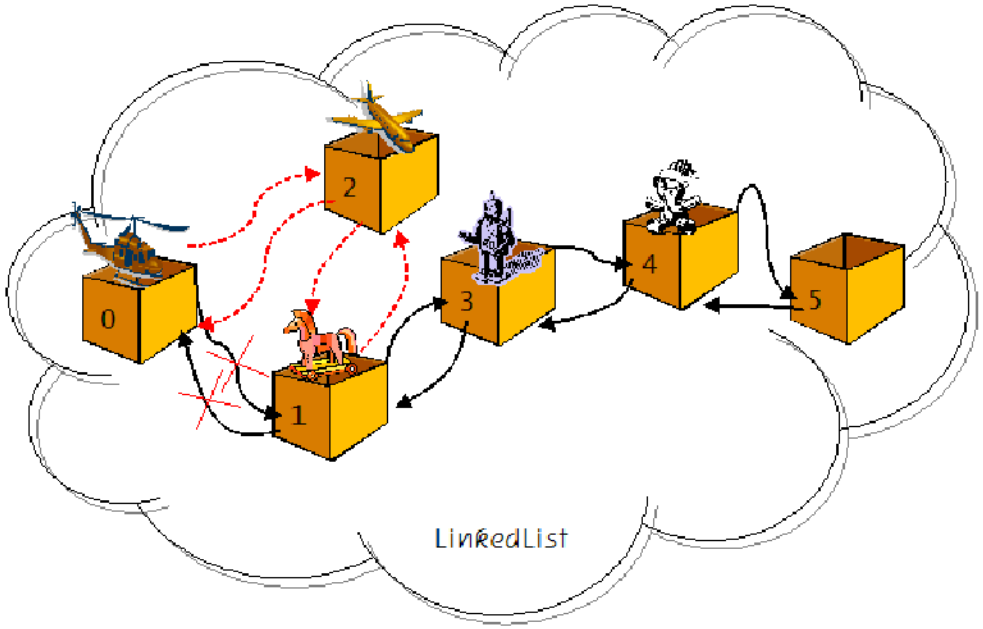
반복자

- 반복자(iterator): 반복자는 컬렉션의 원소들을 하나씩 처리하는데 사용

```
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())
{
      s = (String)e.next();          // 반복자는 Object 타입을 반환!
      System.out.println(s);
}
```

LinkedList

- 빈번하게 삽입과 삭제가 일어나는 경우에 사용



예제

LinkedListTest.java

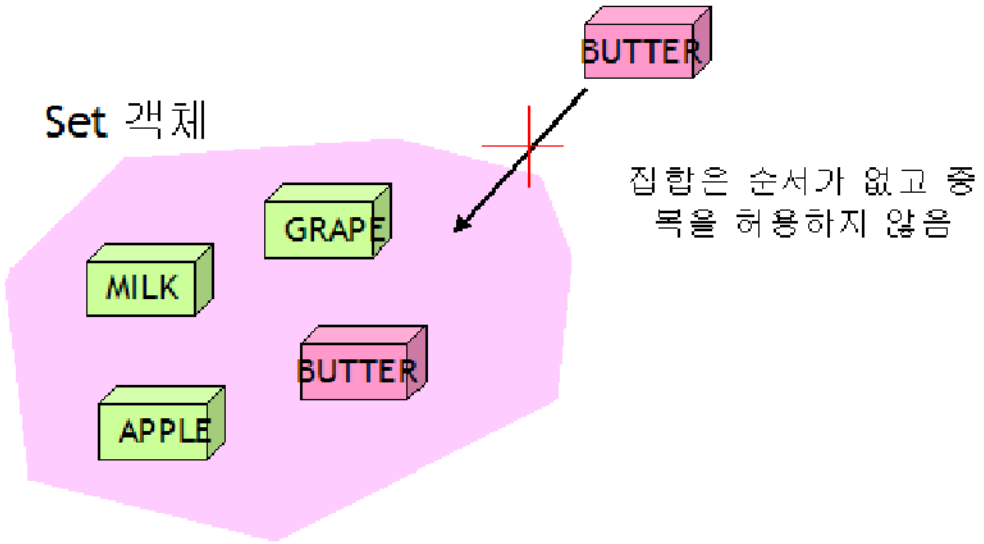
```
import java.util.*;

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

                list.add("MILK");
                list.add("BREAD");
                list.add("BUTTER");
                list.add(1, "APPLE"); // 인덱스 1에 “APPLE"을 삽입
                list.set(2, "GRAPE"); // 인덱스 2의 원소를 “GRAPE"로 대체
                list.remove(3); // 인덱스 3의 원소를 삭제한다.

               for (int i = 0; i < list.size(); i++)
                        System.out.println(list.get(i));
        }
}
```

배열을 리스트로 변환하기

- List<String> list = Arrays.asList(new String[size]);
  - 일반적인 배열을 리스트로 변환한다.

Set

- 집합(Set)은 원소의 중복을 허용하지 않는다.



Set 인터페이스를 구현하는 방법

- 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

```
import java.util.*;

public class SetTest {
  public static void main(String args[]) {
    HashSet<String> set = new HashSet<String>();

    set.add("Milk");
    set.add("Bread");
    set.add("Butter");
    set.add("Cheese");
    set.add("Ham");
    set.add("Ham");

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

[Bread, Milk, Butter, Ham, Cheese]

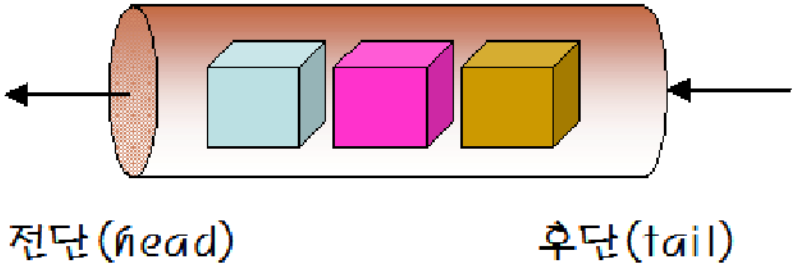
```java
import java.util.*;
public class SetTest1 {
    public static void main(String[] args) {
        Set<String> s1 = new HashSet<String>();
        Set<String> s2 = new HashSet<String>();

        s1.add("A");
        s1.add("B");
        s1.add("C");
        s2.add("A");
        s2.add("D");

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

        System.out.println("합집합 " + union);
        System.out.println("교집합 " + intersection);
    }
}
```

실행결과

```
합집합 [D, A, B, C]
교집합 [A]
```

Queue

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



Queue 인터페이스

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

예제

QueueTest.java

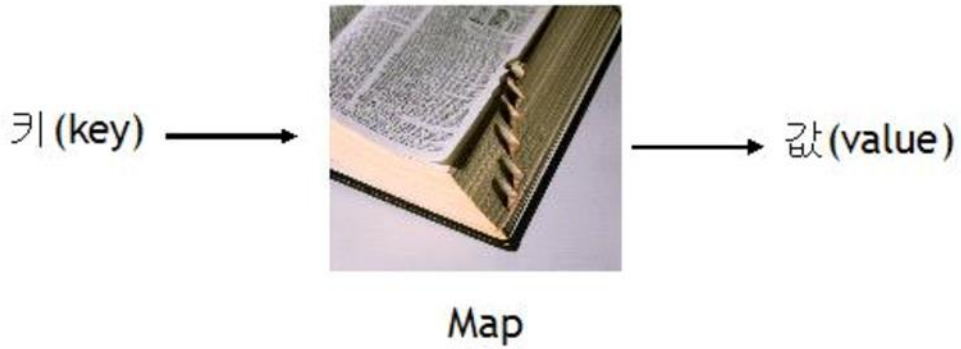
```
import java.util.*;

public class QueueTest {
    public static void main(String[] args) throws InterruptedException {
        int time = 10;
        Queue<Integer> queue = new LinkedList<Integer>();
        for (int i = time; i >= 0; i--)
            queue.add(i);
        while (!queue.isEmpty()) {
            System.out.print(queue.remove()+" ");
            Thread.sleep(1000);        // 현재의 스레드를 1초간 재운다.
        }
    }
}
```

실행결과

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

Map

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



Map

예제

MapTest.java

```
import java.util.*;

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"));

		// 모든 항목을 출력한다.
		System.out.println(st);

		// 하나의 항목을 삭제한다.
		st.remove("20090002");
		// 하나의 항목을 대치한다.
		st.put("20090003", new Student(20090003, "소이정"));
		// 값을 참조한다.
		System.out.println(st.get("20090003"));
		// 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
```