

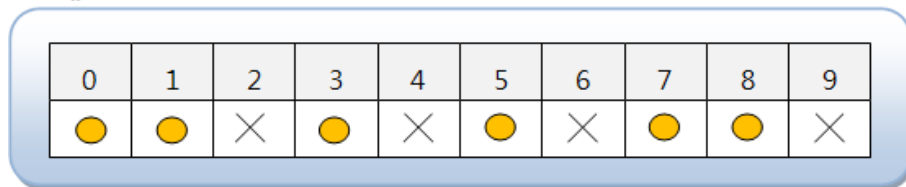
15장 컬렉션 프레임워크

15.1 컬렉션 프레임워크 소개

■ 배열의 문제점

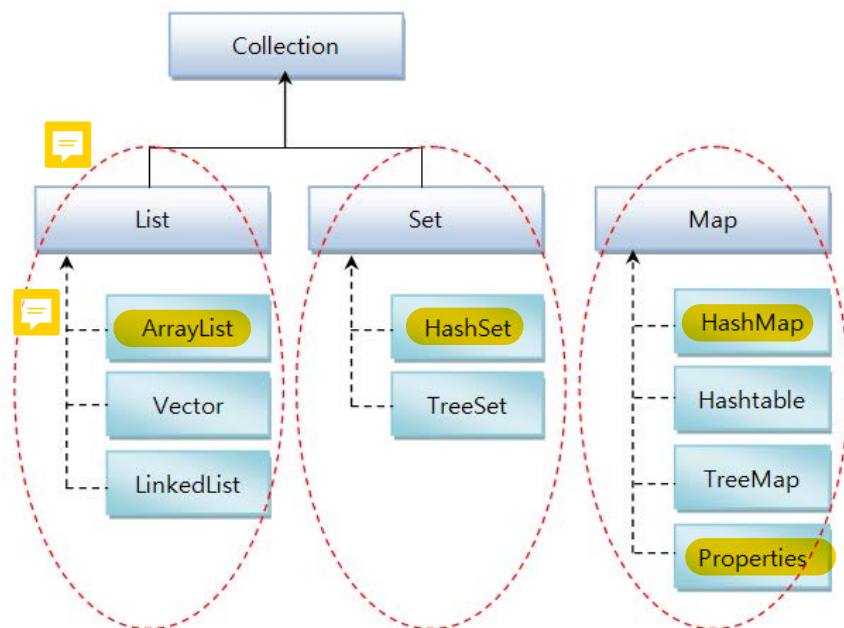
- 저장할 수 있는 객체 수가 배열을 생성할 때 결정 → 불특정 다수의 객체를 저장하기에는 문제
- 객체 삭제했을 때 해당 인덱스가 비게 됨 → 낱알 빠진 옥수수 같은 배열, 객체를 저장하려면 어디가 비어있는지 확인하는 코드도 필요하다.

배열



- 컬렉션이란 사전적 의미로 요소(객체)를 수집해 저장하는 것을 말한다.

- 컬렉션 프레임워크는 몇가지 인터페이스를 통해서 다양한 컬렉션 클래스를 이용할 수 있도록 하고 있다.



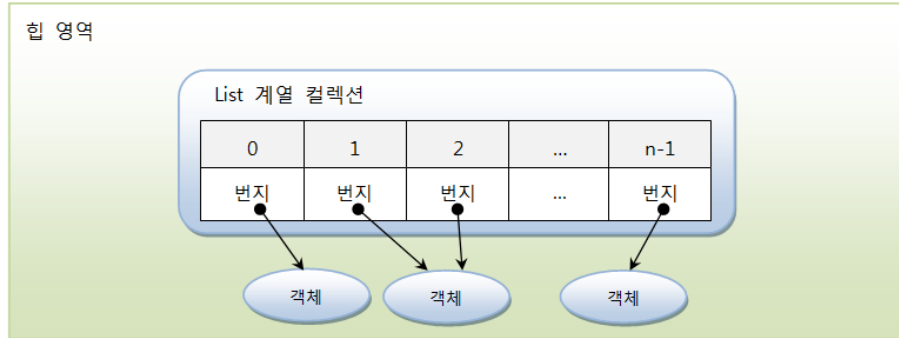
인터페이스 분류		특징	구현 클래스
Collection	List	- 순서를 유지하고 저장 - 중복 저장 가능	ArrayList, Vector, LinkedList
	Set	- 순서를 유지하지 않고 저장 - 중복 저장 안 됨	HashSet, TreeSet
Map		- 키와 값의 쌍으로 저장 - 키는 중복 저장 안 됨	HashMap, Hashtable, TreeMap, Properties



15.2 List 컬렉션

■ 특징

- 인덱스로 관리
- 중복해서 객체 저장 가능 (동일한 번지를 참조)



■ 구현 클래스: ArrayList, Vector, LinkedList

■ 주요 메소드

기능	메소드	설명
객체 추가	<code>boolean add(E e)</code>	주어진 객체를 맨끝에 추가
	<code>void add(int index, E element)</code>	주어진 인덱스에 객체를 추가
	<code>set(int index, E element)</code>	주어진 인덱스에 저장된 객체를 주어진 객체로 바꿈
객체 검색	<code>boolean contains(Object o)</code>	주어진 객체가 저장되어 있는지 여부
	<code>E get(int index)</code>	주어진 인덱스에 저장된 객체를 리턴
	<code>isEmpty()</code>	컬렉션이 비어 있는지 조사
	<code>int size()</code>	저장되어있는 전체 객체수를 리턴
객체 삭제	<code>void clear()</code>	저장된 모든 객체를 삭제
	<code>E remove(int index)</code>	주어진 인덱스에 저장된 객체를 삭제
	<code>boolean remove(Object o)</code>	주어진 객체를 삭제

```

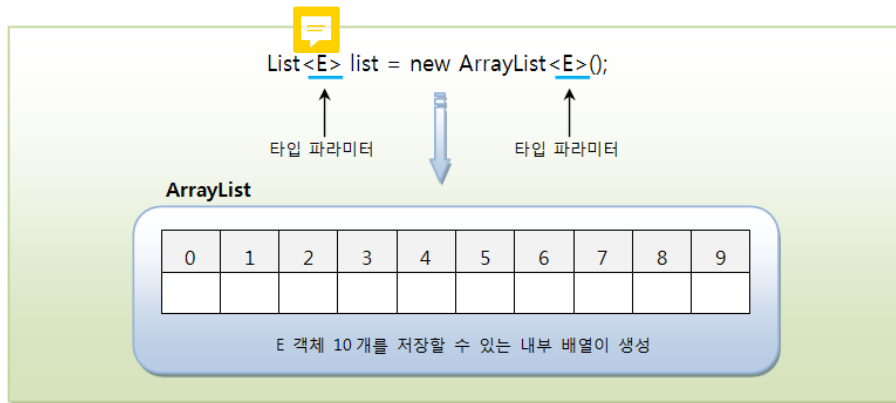
List<String> list = ...;
list.add("홍길동"); // 맨끝에 객체 추가
list.add(1,"신용권"); //지정된 인덱스에 객체 삽입
String str = list.get(1); //인덱스로 객체 찾기
list.remove(0); // 인덱스로 객체 삭제
list.remove("신용권"); // 객체 삭제

for(int i=0; i<list.size(); i++) {
    String str = list.get(i);
}
for(String str: list) { // 저장된 총 객체 수만큼 루핑
}
    
```

15.2.1. ArrayList

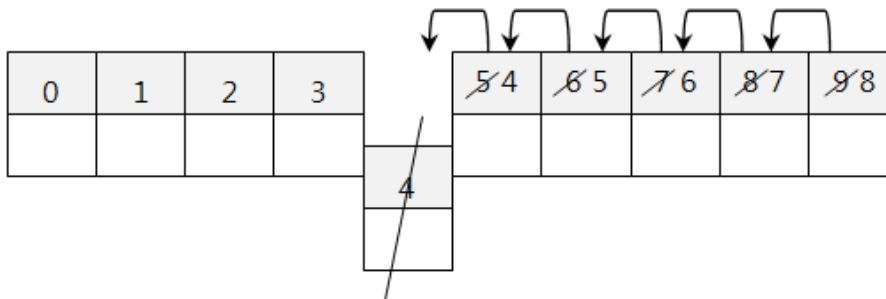
■ 저장 용량(capacity)

- 초기 용량 : 10 (따로 지정 가능)
- 저장 용량을 초과한 객체들이 들어오면 자동적으로 늘어남. 고정도 가능



■ 객체 제거

- 바로 뒤 인덱스부터 마지막 인덱스까지 모두 앞으로 1씩 당겨짐



- 다음 예제는 ArrayList에 String 객체를 추가, 검색, 삭제하는 방법을 보여준다.

[ArrayListExample.java] String 객체를 저장하는 ArrayList

```
package sec02.exam01_arraylist;

import java.util.*;

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

        list.add("Java");
        list.add("JDBC");
        list.add("Servlet/JSP");
        list.add(2, "Database");
        list.add("iBATIS");

        int size = list.size();
        System.out.println("총 객체수: " + size);
        System.out.println();

        String skill = list.get(2);
        System.out.println("2: " + skill);
        System.out.println();

        for (int i = 0; i < list.size(); i++) {
            String str = list.get(i);
            System.out.println(i + ":" + str);
        }
        System.out.println();
    }
}
```

```
list.remove(2);
list.remove(2);
list.remove("iBAtIS");

for (int i = 0; i < list.size(); i++) {
    String str = list.get(i);
    System.out.println(i + ":" + str);
}
}
```

- 다음은 고정된 String 객체를 요소로 갖는 ArrayList 객체를 생성한다. 이런 경우에는 Arrays.asList(T... a) 메소드를 사용하는 것이 간편하다.

[ArraysAsListExample.java] Arrays.asList() 메소드

```
package sec02.exam01_arraylist;

import java.util.Arrays;
import java.util.List;

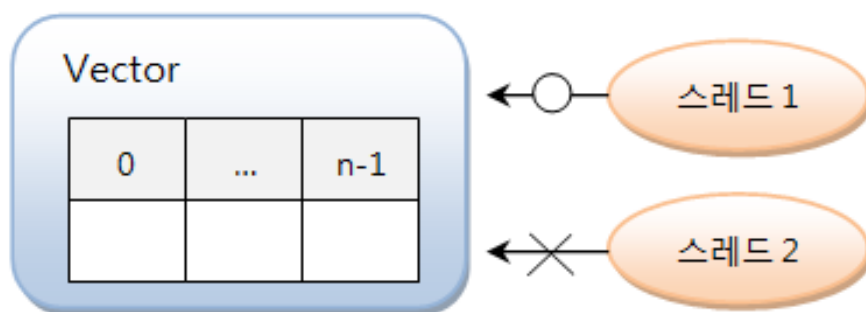
public class ArraysAsListExample {
    public static void main(String[] args) {
        List<String> list1 = Arrays.asList("홍길동", "신용권", "감자바");
        for (String name : list1) {
            System.out.println(name);
        }

        List<Integer> list2 = Arrays.asList(1, 2, 3);
        for (int value : list2) {
            System.out.println(value);
        }
    }
}
```



15.2.2. Vector

- 기본형: List<E> list = new Vector<E>();
- ArrayList와 다르게 Vector는 동기화된(synchronized) 메소드로 구성되어 있기 때문에 멀티 스레드가 동시에 이 메소드들을 실행할 수 없고, 하나의 스레드가 실행을 완료해야만 다른 스레드를 실행할 수 있다. 그래서 멀티 스레드 환경에서 안전하게 객체를 추가, 삭제할 수 있다(thread safe).



- 다음은 Vector를 이용해서 Board 객체를 추가, 삭제, 검색하는 예제이다.

[VectorExample.java] Board 객체를 저장하는 Vector

```

01 package sec02.exam02_vector;
02
03 import java.util.List;
04 import java.util.Vector;
05
06 public class VectorExample {
07     public static void main(String[] args) {
08         List<Board> list = new Vector<Board>();
09
10         list.add(new Board("제목1", "내용1", "글쓴이1"));
11         list.add(new Board("제목2", "내용2", "글쓴이2"));
12         list.add(new Board("제목3", "내용3", "글쓴이3"));
13         list.add(new Board("제목4", "내용4", "글쓴이4"));
14         list.add(new Board("제목5", "내용5", "글쓴이5"));
15
16         list.remove(2);
17         list.remove(3);
18
19         for (int i = 0; i < list.size(); i++) {
20             Board board = list.get(i);
21             System.out.println(board.subject + "\t" + board.content + "\t" +
22 board.writer);
23         }
24     }
25 }

```

[Board.java] 게시물 정보 객체

```

01 package sec02.exam02_vector;
02
03 public class Board {
04     String subject;
05     String content;
06     String writer;
07
08     public Board(String subject, String content, String writer) {
09         this.subject = subject;
10         this.content = content;
11         this.writer = writer;
12     }
13 }

```

[꿀팁] 동기화

- 한 스레드가 공유자원에 작업을 마치기 전까지 다른 스레드의 접근을 제한함 -> 자료의 신뢰성을 보장받을 수 있으나 속도저하가 발생할 수 있다.
- cf) ArrayList 동기화 지원 방법
List list = Collections.synchronizedList(new ArrayList()); // Vector와 동일한 클래스

[EnumIterVector.java]

```

import java.util.*;

```

```
public class EnumIterVector {

    public static void main( String[] args ) {

        // Vector 객체 생성
        Vector v = new Vector( 1, 1 );

        // Vector에 Object 저장
        v.add( 30 );
        v.addElement( new Integer( 10 ));
        v.addElement( "johnharu" );
        v.addElement( "gracedew" );

        // Vector의 Object의 나열형을 리턴
        Enumeration e = v.elements();

        // Enumeration을 이용해 Vector의 Object를 출력
        while( e.hasMoreElements() ) {
            System.out.println( e.nextElement() );
        }

        // Vector의 Object의 나열형을 리턴
        Iterator ie = v.iterator();

        // Iterator을 이용해 Vector의 Object를 출력
        while( ie.hasNext() ) {
            System.out.println( ie.next() );
        }

    } //main end
}
```

[실습] Vector를 이용한 카드놀이 만들기

- Vector를 이용하여 서로 다른 카드 40장을 만들어 보자. (힌트: equals와 hashCode를 오버라이딩한다. Vector는 캐스팅이 필요하다.)

[CardVectorBoxMain.java]

```
01 package verify.exam00;
02
03 import java.util.Vector;
04
05 public class CardVectorBoxMain {
06     public static void main(String[] args) {
07         CardVectorBox box = new CardVectorBox();
08         Vector cards = box.getAllCards();
09         println(cards);
10         System.out.println("\n-----");
11         box.shuffles();// 카드 섞기
12         cards = box.getAllCards();
13         println(cards);
14     }
15
16     public static void println(Vector cards) {
17         for (int i = 0; i < cards.size(); i++) {
18             System.out.print((Card) cards.get(i));
19             if ((i + 1) % 10 == 0) {
20                 System.out.println();
21             }
22         }
23     }
24 }
```

```

22         }
23     }
24 }

```

[CardVectorBox.java]

```

01 package verify.exam00;
02
03 import java.util.Collections;
04 import java.util.Vector;
05
06 public class CardVectorBox {
07     private final int numOfCards = Card.deck.length * Card.stic.length; // 40
08     private Vector vCard;
09
10     public CardVectorBox() {
11         make();
12     }
13
14     private void make() {
15         init();
16         fulling();
17     }
18
19     private void init() {
20         vCard = new Vector(5, 5);
21         vCard.clear();
22     }
23
24     public void fulling() {
25         int count = 0;
26         while (true) {
27             Card cd = new Card(); // 임의의 카드를 만든다.
28             if (!vCard.contains(cd)) { // 같은것이 없다면
29                 vCard.add(cd); // add
30                 count++;
31             }
32             if (count == numOfCards) { // 40
33                 break;
34             }
35         }
36     }
37
38     public Vector getAllCards() {
39         return vCard;
40     }
41
42     public void shuffles() {
43         Collections.shuffle(vCard);
44     }
45 }

```

[Card.java]

```

01 package verify.exam00;
02
03 public class Card {
04     public static final String[] deck = { "H", "D", "S", "C" };
05     public static final String[] stic = { "1", "2", "3", "4", "5", "6", "7", "8", "9", "T" };
06     private String card;
07
08     public Card() {

```

```

09         init();
10     }
11
12     public Card(Card cd) {
13         card = cd.getCard().trim();// 공백제거
14     }
15
16     public void init() {
17         int a = (int) (Math.random() * deck.length); // 0~3
18         int b = (int) (Math.random() * stic.length); // 0~9
19         card = deck[a] + stic[b]; // H 8
20     } // 카드에 값을 할당
21
22     public String getCard() { // 카드값
23         return card;
24     }
25
26     public String toString() { // 출력보기
27         return "[" + card + " "]; // [H8]
28     }
29
30     public boolean equals(Object obj) {
31         boolean isS = false;
32         Card cd = (Card) obj;
33         if (card.equals(cd.getCard())) {
34             isS = true;
35         }
36         return isS;
37     }
38
39     public int hashCode() {
40         return card.hashCode() + 137;
41     }
42 }

```

[실행결과]

```

[H1] [C1] [ST] [C5] [D2] [D8] [DT] [C4] [S7] [D4]
[H9] [S3] [HT] [H8] [H3] [C3] [D7] [C9] [D3] [CT]
[D9] [S4] [C6] [H6] [H4] [S8] [D1] [C8] [D6] [S2]
[H2] [S6] [S9] [H5] [S1] [H7] [C2] [D5] [C7] [S5]

```

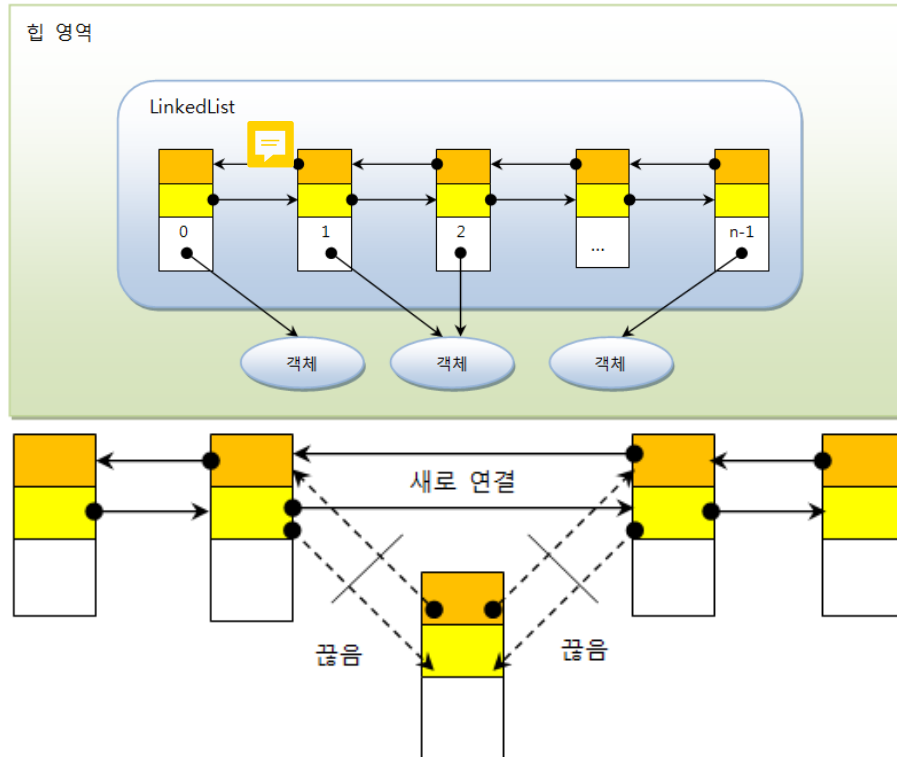
```

-----
[S7] [C6] [H6] [S8] [C7] [H9] [S1] [H2] [S2] [CT]
[S6] [C2] [C4] [D6] [D7] [D3] [D5] [H8] [C5] [S4]
[H3] [D9] [C3] [C9] [S9] [S5] [H1] [ST] [D8] [D2]
[HT] [S3] [C8] [C1] [DT] [H5] [H4] [H7] [D4] [D1]

```

15.2.3. LinkedList

- 기본형: `List<E> list = new LinkedList<E>();`
- 인접 참조를 링크해서 체인처럼 관리
- 특정 인덱스에서 객체를 제거하거나 추가하게 되면 바로 앞뒤 링크만 변경



- 빈번한 객체 삭제와 삽입이 일어나는 곳에서는 ArrayList보다 좋은 성능

[LinkedListExample.java]

```
package sec02.exam03_linkedlist;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class LinkedListExample {
    public static void main(String[] args) {
        List<String> list1 = new ArrayList<String>();
        List<String> list2 = new LinkedList<String>();

        long startTime;
        long endTime;

        startTime = System.nanoTime();
        for (int i = 0; i < 10000; i++) {
            list1.add(0, String.valueOf(i));
        }
        endTime = System.nanoTime();
        System.out.println("ArrayList 걸린시간: " + (endTime - startTime) + " ns");

        startTime = System.nanoTime();
        for (int i = 0; i < 10000; i++) {
            list2.add(0, String.valueOf(i));
        }
        endTime = System.nanoTime();
        System.out.println("LinkedList 걸린시간: " + (endTime - startTime) + " ns");
    }
}
```

[꿀잼] 카드 게임

[GameMain.java] 카드 게임

```
package verify;

public class GameMain {

    public static void main(String[] args) {
        // 1. player 만들고
        Player[] players = new Player[3];

        for (int i = 0; i < players.length; i++) {
            players[i] = new Player();
            players[i].cards = new Card[5];
        }

        players[0].name = "luffy";
        players[1].name = "zoro";
        players[2].name = "choint";

        // 2. 카드덱을 만들고
        Deck deck = new Deck();

        // 3. 카드를 섞어서 나누어 주고
        deck.deal(players);

        // 4. 결과물 보기
        for (int i = 0; i < players.length; i++) {
            System.out.println(players[i].name + ":");
            for (int j = 0; j < 5; j++) {
                System.out.print(players[i].cards[j].kind);
                System.out.print(players[i].cards[j].number);
                System.out.print(" ");
            }
            System.out.println();
        }
    }
}

class Deck {
    Card[] cards;
    String[] kinds = { "spade", "heart", "diamond", "clover" };
    Deck() {
        this.init();
    }

    void deal(Player[] players) {
        this.shuffle();
        for (int i = 0; i < players.length; i++) {
            // 카드의 수는 5장씩...
            for (int j = 0; j < 5; j++) {
                players[i].cards[j] = cards[i * 5 + j];
            }
        }
    }

    void init() {
        this.cards = new Card[52];
        for (int i = 0; i < cards.length; i++) {
            cards[i] = new Card();
        }
    }
}
```

```

        cards[i].number = (i % 13) + 1;
        cards[i].kind = kinds[i / 13];
    }
}

void shuffle() {
    for (int i = 0; i < 10000; i++) {
        int r = (int) (Math.random() * 52);
        Card temp = cards[0];
        cards[0] = cards[r];
        cards[r] = temp;
    }
}

class Player {
    String name;
    Card[] cards;
}

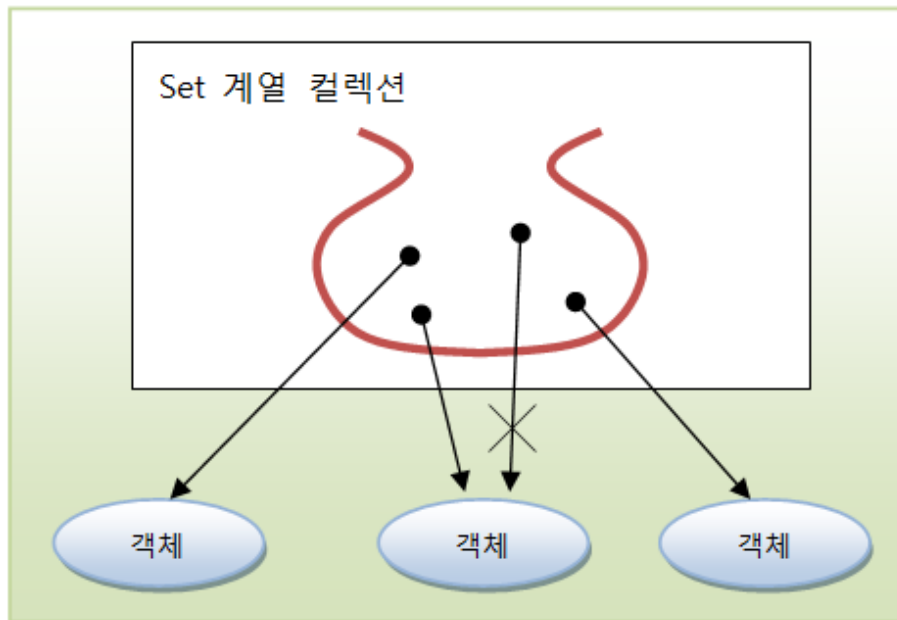
class Card {
    int number;
    String kind;
}

```

15.3 Set 컬렉션

■ 특징

- 수학의 집합에 비유
- 저장 순서가 유지되지 않음
- 객체를 중복 저장 불가
- 하나의 null만 저장 가능



■ 주요 메소드

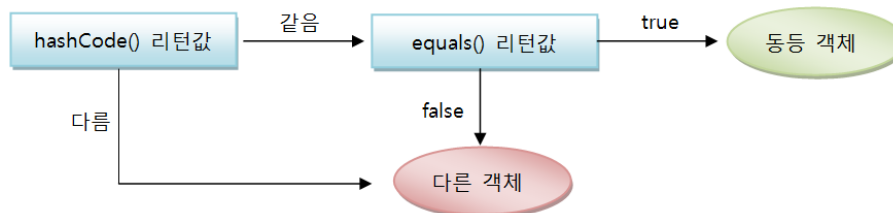
기능	메소드	설명
객체 추가	<code>boolean add(E e)</code>	주어진 객체를 저장, 객체가 성공적으로 저장되면 <code>true</code> 를 리턴하고 중복 객체면 <code>false</code> 를 리턴
객체 검색	<code>boolean contains(Object o)</code>	주어진 객체가 저장되어 있는지 여부
	<code>isEmpty()</code>	컬렉션이 비어 있는지 조사
	<code>Iterator<E> iterator()</code>	저장된 객체를 한번씩 가져오는 반복자 리턴
	<code>int size()</code>	저장되어있는 전체 객체수 리턴
객체 삭제	<code>void clear()</code>	저장된 모든 객체를 삭제
	<code>boolean remove(Object o)</code>	주어진 객체를 삭제

- 인덱스로 객체를 검색해서 가져오는 메소드가 없다. 대신, 전체 객체를 대상으로 한 번씩 반복해 가져오는 반복자(Iterator)를 제공한다.

```
Set<String> set = ...;
Iterator<String> iterator = set.iterator();
while(iterator.hasNext()) { // 저장된 객체 수만큼 루핑한다.
    String str = iterator.next();
    if(str.equals("홍길동")) {
        iterator.remove();
    }
}
```

15.3.1 HashSet

- 기본형: `Set<E> set = new HashSet<E>();`
- 동일 객체 및 동등 객체는 중복 저장하지 않음
- 동등 객체 판단 방법



[HashSetExample1.java] String 객체를 중복 없이 저장하는 HashSet

```
package sec03.exam01_hashset;

import java.util.*;

public class HashSetExample1 {
    public static void main(String[] args) {
        Set<String> set = new HashSet<String>();

        set.add("Java");
        set.add("JDBC");
        set.add("Servlet/JSP");
        set.add("Java");
        set.add("iBatis");

        int size = set.size();
        System.out.println("총 객체수: " + size);
    }
}
```

```

        Iterator<String> iterator = set.iterator();
        while (iterator.hasNext()) {
            String element = iterator.next();
            System.out.println("\t" + element);
        }

        set.remove("JDBC");
        set.remove("iBatis");

        System.out.println("총 객체수: " + set.size());

        for (String element : set) {
            System.out.println("\t" + element);
        }

        set.clear();
        if (set.isEmpty()) {
            System.out.println("비어 있음");
        }
    }
}

```

[과제] Set을 이용한 로또

- 1~45 사이의 정수중에서 6개의 숫자를 추출([7, 41, 25, 27, 29, 15])하는 로또 프로그램을 작성하세요. (단, Set 자료구조를 사용해서 중복 숫자가 나오지 않도록 작성하세요)

[SetTest01.java]

```

package verify;
import java.util.HashSet;
import java.util.Random;
import java.util.Set;

public class SetTest01 {

    public static void main(String[] args) {
        int n;
        Set s = new HashSet();
        // TreeSet s = new TreeSet();

        Random r = new Random();

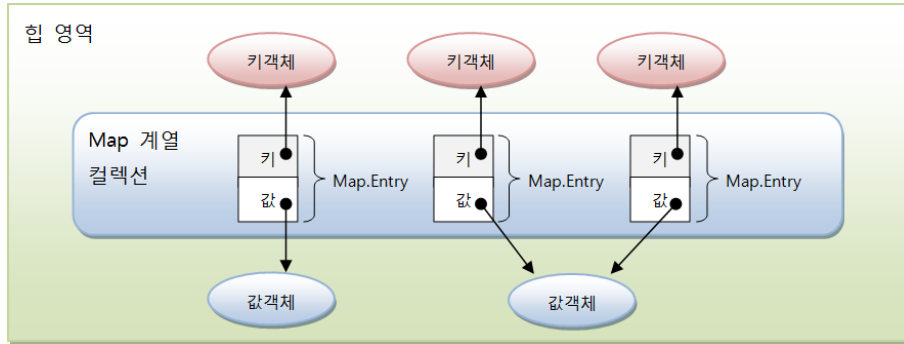
        //...

    }
}

```

15.4 Map 컬렉션

- 특징
 - 키(key)와 값(value)으로 구성된 Map.Entry 객체를 저장하는 구조
 - 키와 값은 모두 객체
 - 키는 중복될 수 없지만 값은 중복 저장 가능



■ 구현 클래스

- **HashMap**, Hashtable, LinkedHashMap, **Properties**, TreeMap

■ 주요 메소드

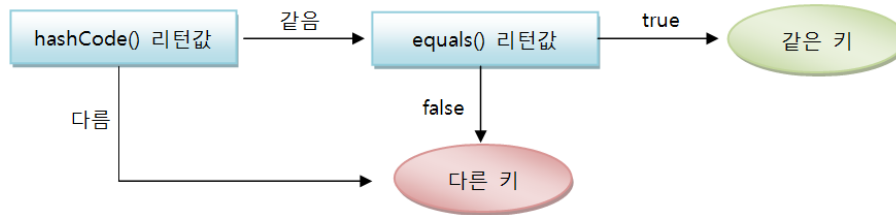
기능	메소드	설명
객체 추가	V put(K key, V value)	주어진 키와 값을 추가, 저장되면 값을 리턴
	boolean containsKey(Object key)	주어진 키가 있는지 여부
객체 검색	boolean containsValue(Object value)	주어진 값이 있는지 여부
	Set<Map.Entry<K,V>> entrySet()	키와 값의 쌍으로 구성된 모든 Map.Entry 객체를 Set에 담아서 리턴
	V get(Object key)	주어진 키의 값을 리턴
	boolean isEmpty()	컬렉션이 비어있는지 여부
	Set<K> keySet()	모든 키를 Set 객체에 담아서 리턴
	int size()	저장된 키의 총 수를 리턴
	Collection<V> values()	저장된 모든 값 Collection에 담아서 리턴
객체 삭제	void clear()	모든 Map.Entry(키와 값)를 삭제
	V remove(Object key)	주어진 키와 일치하는 Map.Entry 삭제, 삭제가 되면 값을 리턴

```
Map<String, Integer> map = ~;
map.put("홍길동", 30);
int score = map.get("홍길동");
map.remove("홍길동");
```

15.4.1 HashMap

- 기본형: Map<K,V> map = new HashMap<K,V>(); // **K=키타입**, **V=값타입**

- HashMap의 키로 사용할 객체는 hashCode()와 equals() 메소드를 재정의해서 동등 객체가 될 조건을 정해야 한다. 주로 키 타입은 String을 많이 사용하는데, String은 문자열이 같을 경우 동등 객체가 될 수 있도록 hashCode()와 equals() 메소드가 재정의되어 있다.



- 키와 값의 타입은 객체이므로 기본 타입을 사용할 수 없고 클래스 및 인터페이스 타입만 가능하다.



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

- 다음 예제는 이름을 키로, 점수를 값으로 저장하는 HashMap 사용 방법을 보여준다.

[HashMapExample1.java] 이름을 키로 점수를 값으로 저장하기

```
package sec04.exam01_hashmap;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class HashMapExample1 {
    public static void main(String[] args) {
        // Map 컬렉션 생성
        Map<String, Integer> map = new HashMap<String, Integer>();

        // 객체 저장
        map.put("신용권", 85);
        map.put("홍길동", 90);
        map.put("동장군", 80);
        map.put("홍길동", 95); // '홍길동' 키가 같기 때문에 제일 마지막에 저장한 값으로 대체
        System.out.println("총 Entry 수: " + map.size());

        // 객체 찾기
        System.out.println("\t홍길동 : " + map.get("홍길동"));
        System.out.println();

        // 객체를 하나씩 처리
        Set<String> keySet = map.keySet();
        Iterator<String> keyIterator = keySet.iterator();
        while (keyIterator.hasNext()) {
            String key = keyIterator.next();
            Integer value = map.get(key);
            System.out.println("\t" + key + " : " + value);
        }
        System.out.println();

        // 객체 삭제
        map.remove("홍길동");
        System.out.println("총 Entry 수: " + map.size());

        // 객체를 하나씩 처리
        Set<Map.Entry<String, Integer>> entrySet = map.entrySet();
        Iterator<Map.Entry<String, Integer>> entryIterator = entrySet.iterator();
        while (entryIterator.hasNext()) {
            Map.Entry<String, Integer> entry = entryIterator.next();
            String key = entry.getKey();
            Integer value = entry.getValue();
            System.out.println("\t" + key + " : " + value);
        }
    }
}
```

```

    }
    System.out.println();

    // 객체 전체 삭제
    map.clear();
    System.out.println("총 Entry 수: " + map.size());
}
}

```

- 다음 예제는 사용자 정의 객체인 Student를 키로하고 점수를 저장하는 HashMap 사용 방법을 보여준다.

[Student.java] 키로 사용할 객체 - hashCode()와 equals() 재정의

```

package sec04.exam01_hashmap;

public class Student {
    public int sno;
    public String name;

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

    public boolean equals(Object obj) {
        if (obj instanceof Student) {
            Student student = (Student) obj;
            return (sno == student.sno) && (name.equals(student.name));
        } else {
            return false;
        }
    }

    public int hashCode() {
        return sno + name.hashCode();
    }
}

```

[HashMapExample2.java] 학번과 이름이 동일한 경우 같은 키로 인식

```

package sec04.exam01_hashmap;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

public class HashMapExample2 {
    public static void main(String[] args) {
        Map<Student, Integer> map = new HashMap<Student, Integer>();

        map.put(new Student(1, "홍길동"), 95);
        map.put(new Student(1, "홍길동"), 95); // 학번과 이름이 동일한 Student를 키로 저장

        System.out.println("총 Entry 수: " + map.size());
    }
}

```


15.4.2 Hashtable



- 키 객체 만드는 법은 HashMap과 동일
- Hashtable은 스레드 동기화(synchronization)가 된 상태
 - 복수의 스레드가 동시에 Hashtable에 접근해서 객체를 추가, 삭제하더라도 스레드에 안전(thread safe)

```
// 기본형
Map<K,V> map = new Hashtable<K,V>();

Map<String, Integer> map = new Hashtable<String, Integer>();
```

[HashtableExample.java]

```
package sec04.exam02_hashtable;

import java.util.*;

public class HashtableExample {
    public static void main(String[] args) {
        Map<String, String> map = new Hashtable<String, String>();

        map.put("spring", "12");
        map.put("summer", "123");
        map.put("fall", "1234");
        map.put("winter", "12345");

        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.println("아이디와 비밀번호를 입력해주세요");
            System.out.print("아이디: ");
            String id = scanner.nextLine();

            System.out.print("비밀번호: ");
            String password = scanner.nextLine();
            System.out.println();

            if (map.containsKey(id)) {
                if (map.get(id).equals(password)) {
                    System.out.println("로그인 되었습니다");
                    break;
                } else {
                    System.out.println("비밀번호가 일치하지 않습니다.");
                }
            } else {
                System.out.println("입력하신 아이디가 존재하지 않습니다");
            }
        }
    }
}
```



15.4.3 Properties

- 특징

- 키와 값을 String 타입으로 제한한 Map 컬렉션
- Properties는 프로퍼티(~.properties) 파일을 읽어 들일 때 주로 사용

■ 프로퍼티(~.properties) 파일

- 옵션 정보, 데이터베이스 연결 정보, 국제화(다국어) 정보를 기록
- 텍스트 파일로 활용
- 애플리케이션에서 주로 변경이 잦은 문자열을 저장
- 유지 보수를 편리하게 만들어 줌
- 키와 값이 = 기호로 연결되어 있는 텍스트 파일
- ISO 8859-1 문자셋으로 저장
- 한글은 유니코드(Unicode)로 변환되어 저장

■ 다음은 database.properties 파일로부터 값을 읽어 출력하는 예제이다.

[database.properties] 키=값으로 구성된 프로퍼티

```
driver=oracle.jdbc.OracleDirver
url=jdbc:oracle:thin:@localhost:1521:orcl
username=scott
password=tiger
```

[PropertiesExample.java] 프로퍼티 파일로부터 읽기

```
package sec04.exam03_properties;

import java.io.FileReader;
import java.net.URLDecoder;
import java.util.Properties;

public class PropertiesExample {
    public static void main(String[] args) throws Exception {
        Properties properties = new Properties();
        String path = PropertiesExample.class.getResource("database.properties").getPath();
        path = URLDecoder.decode(path, "utf-8");
        properties.load(new FileReader(path));

        String driver = properties.getProperty("driver");
        String url = properties.getProperty("url");
        String username = properties.getProperty("username");
        String password = properties.getProperty("password");

        System.out.println("driver : " + driver);
        System.out.println("url : " + url);
        System.out.println("username : " + username);
        System.out.println("password : " + password);
    }
}
```

[과제] 게시판 만들기

- 다음은 게시물을 관리하는 모듈이다. 특정 사용자가 등록한 게시물중에 비밀번호가 같으면 삭제할 수 있도록 코드를 작성하라.

// 실행 결과

```

메뉴를 입력 하세요
1.게시판 글쓰기
2.글 목록 보기
3.글 삭제
4.종료
3
삭제할 글의 작성자와 비밀번호를 입력하세요.
작성자: 2
비밀번호: 2
//if 작성자 혹은 비밀번호가 틀리면
// "해당 작성자가 없거나 비밀번호가 일치하지 않습니다."
//else
// "성공적으로 글이 삭제되었습니다."

```

```

메뉴를 입력 하세요
1.게시판 글쓰기
2.글 목록 보기
3.글 삭제
4.종료

```

[BoardMain.java]

```

package verify;

import java.util.Scanner;

public class BoardMain {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        boolean isStop = false;
        Scanner sc = new Scanner(System.in);
        BoardSVC boardSVC = new BoardSVC();

        do {
            System.out.println();
            System.out.println("메뉴를 입력 하세요");
            System.out.println("1.게시판 글쓰기");
            System.out.println("2.글 목록 보기");
            System.out.println("3.글 삭제");
            System.out.println("4.종료");

            String menu = sc.next();

            switch (menu) {
                case "1":
                    boardSVC.writeArticle(sc);
                    break;
                case "2":
                    boardSVC.listArticles(sc);
                    break;
                case "3":
                    boardSVC.removeArticle(sc);
                    break;
                case "4":
                    isStop = true;
            }
        } while (!isStop);
    }
}

```

[BoardVO.java]

```
package verify;
// VO(Value Object) 클래스

// DTO(Data Transfer Object) 클래스

public class BoardVO extends Object {

    // 멤버변수
    private String register;
    private String subject;
    private String email;
    private String content;
    private String passwd;

    // 생성자
    public BoardVO(String register, String subject, String email, String content, String passwd) {
        super();
        this.register = register;
        this.subject = subject;
        this.email = email;
        this.content = content;
        this.passwd = passwd;
    }

    public String getRegister() {
        return register;
    }

    public void setRegister(String register) {
        this.register = register;
    }

    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getContent() {
        return content;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public String getPasswd() {
        return passwd;
    }
}
```

```

        public void setPasswd(String passwd) {
            this.passwd = passwd;
        }

        @Override
        public String toString() {
            // TODO Auto-generated method stub
            return "작성자:" + register + ",이메일:" + email + ",제목:" + subject + ",글내용:" +
content;
        }
    }
}

```

[BoardSVC.java]

```

package verify;
// DAO(Data Access Object) 클래스

import java.util.ArrayList;
import java.util.Scanner;

public class BoardSVC {

    ArrayList<BoardVO> boardList;

    public BoardSVC() {
        boardList = new ArrayList();
    }

    // 글 입력 처리 메소드
    public void writeArticle(Scanner sc) {
        System.out.println("게시판에 글을 작성 하세요.");
        System.out.print("작성자:");
        String register = sc.next();

        System.out.print("이메일:");
        String email = sc.next();

        System.out.print("비밀번호:");
        String passwd = sc.next();

        System.out.print("제목:");
        String subject = sc.next();

        System.out.print("글내용:");
        String content = sc.next();

        BoardVO boardVO = new BoardVO(register, subject, email, content, passwd);
        addArticle(boardVO);
    }

    // 글 작성
    private void addArticle(BoardVO boardVO) {
        boardList.add(boardVO);
    }

    // 글목록 출력
    public void listArticles(Scanner sc) {
        if (boardList.size() > 0) {
            for (int i = 0; i < boardList.size(); i++) {
                System.out.println(boardList.get(i).toString());
            }
        } else {

```

```

        System.out.println("등록된 글이 없습니다.");
    }

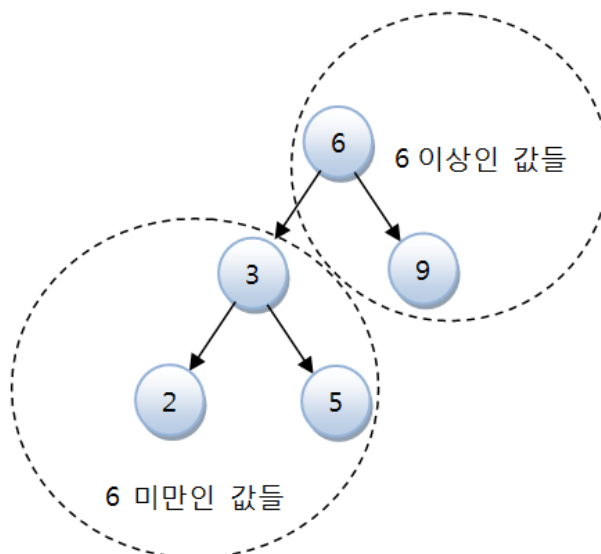
    // 삭제할 글의 작성자 및 비밀번호 입력하는 메소드
    public void removeArticle(Scanner sc) {
        // 코드 작성
    }
}

```

15.5 검색 기능을 강화시킨 컬렉션

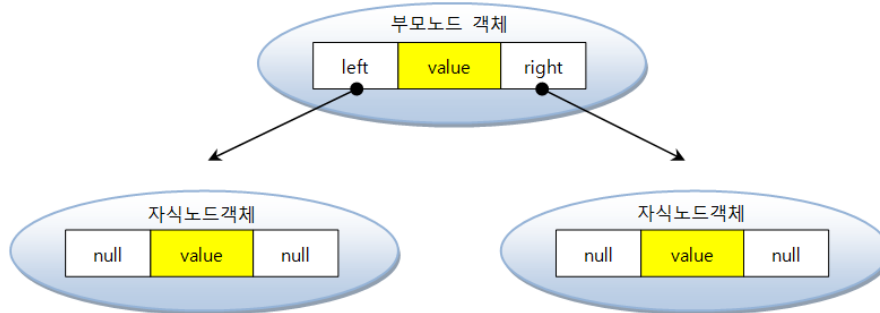
15.5.1 이진 트리 구조

- 부모 노드와 자식 노드로 구성
 - 왼쪽 자식 노드: 부모 보다 적은 값
 - 오른쪽 자식 노드: 부모 보다 큰 값
- 정렬 쉬움
 - 올림 차순: [왼쪽노드→부모노드→오른쪽노드]
 - 내림 차순: [오른쪽노드→부모노드→왼쪽노드]



15.5.2 TreeSet

- 특징
 - 이진 트리(binary tree)를 기반으로 한 Set 컬렉션
 - 왼쪽과 오른쪽 자식 노드를 참조하기 위한 두 개의 변수로 구성



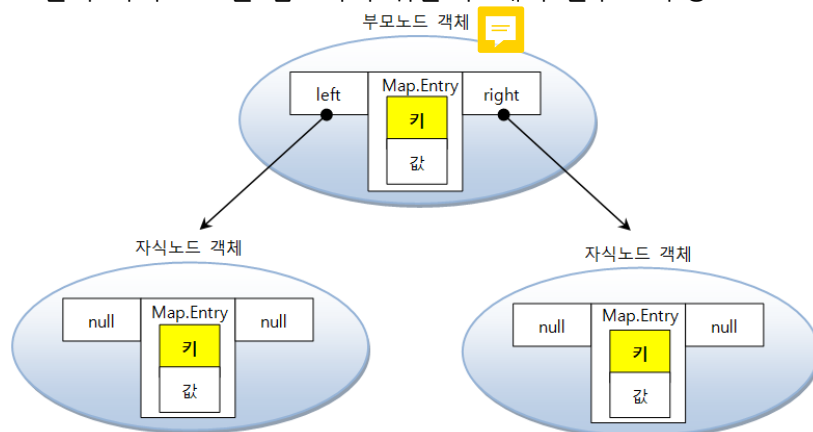
■ 주요 메소드

- 특정 객체를 찾는 메소드: `first()`, `last()`, `lower()`, `higher()`, ...
- 정렬 메소드: `descendingIterator()`, `descendingSet()`
- 범위 검색 메소드: `headSet()`, `tailSet`, `subSet()`

15.5.3 TreeMap

■ 특징

- 이진 트리(binary tree) 를 기반으로 한 Map 컬렉션
- 키와 값이 저장된 `Map.Entry`를 저장
- 왼쪽과 오른쪽 자식 노드를 참조하기 위한 두 개의 변수로 구성



■ 주요 메소드

- 단일 노드 객체를 찾는 메소드: `firstEntry()`, `lastEntry()`, `lowerEntry()`, `higherEntry()`, ...
- 정렬 메소드: `descendingKeySet()`, `descendingMap()`
- 범위 검색 메소드: `headMap()`, `tailMap`, `subMap()`

15.5.4 Comparable과 Comparator

■ TreeSet의 객체와 TreeMap의 키는 저장과 동시에 자동 오름차순 정렬

- 숫자(Integer, Double)타입일 경우에는 값으로 정렬
- 문자열(String) 타입일 경우에는 유니코드로 정렬

- TreeSet과 TreeMap은 정렬 위해 java.lang.Comparable을 구현 객체를 요구
 - Integer, Double, String은 모두 Comparable 인터페이스를 구현하고 있다.
 - 사용자 정의 클래스도 Comparable을 구현한다면 자동 정렬이 가능하다. -> compareTo() 메소드가 재정의(오버라이딩) 되어야 한다.
 - TreeSet의 객체와 TreeMap의 키가 Comparable을 구현하고 있지 않을 경우에는 저장하는 순간 ClassCastException이 발생한다.
- 매개값으로 정렬자(Comparator)를 제공하면 Comparable 비구현 객체도 정렬시킬 수 있다. 정렬자는 Comparator 인터페이스를 구현한 객체이다.

```
// 정렬자
import java.util.Comparator;

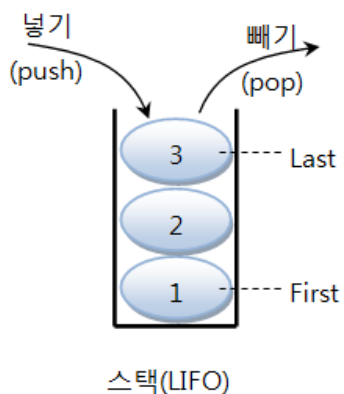
public class DescendingComparator implements Comparator<Fruit> {
    @Override
    public int compare(Fruit o1, Fruit o2) {
        if(o1.price < o2.price) return 1;
        else if(o1.price == o2.price) return 0;
        else return -1;
    }
}
```

```
// 오름차순 또는 내림차순 정렬자
TreeSet<E> treeSet = new TreeSet<E>( new AscendingComparator() );
TreeMap<K,V> treeMap = new TreeMap<K,V>( new DescendingComparator() );
```

15.6 LIFO와 FIFO 컬렉션

15.6.1. Stack

- 특징
 - 후입선출(LIFO: Last In First Out) 구조
 - 응용 예: JVM 스택 메모리



- 주요 메소드

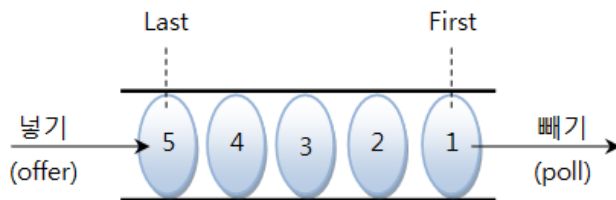
리턴타입	메소드	설명
E	push(E item)	주어진 객체를 스택에 넣는다.
E	peek()	스택의 맨위 객체를 가져온다. 객체를 스택에서 제거하지는 않는다.
E	pop()	스택의 맨위 객체를 가져온다. 객체를 스택에서 제거한다.

```
// 기본형
Stack<E> stack = new Stack<E>();
```

15.6.2. Queue

■ 특징

- 선입선출(FIFO: First In First Out). 예) 작업 큐, 메시지 큐, ...
- Queue 인터페이스를 구현한 대표적인 클래스는 **LinkedList**이다. **LinkedList는 List 인터페이스를 구현했기 때문에 List 컬렉션이기도 하다.** 따라서 LinkedList 객체를 Queue 인터페이스 타입으로 변환하여 사용한다.



큐(FIFO)

■ 주요 메소드

리턴타입	메소드	설명
boolean	offer(E e)	주어진 객체를 넣는다.
E	peek()	객체 하나를 가져온다. 객체를 큐에서 제거하지 않는다.
E	poll()	객체 하나를 가져온다. 객체를 큐에서 제거한다.

```
// 기본형
Queue<E> queue = new LinkedList<E>();
```

- 다음은 Queue를 이용해서 간단한 메시지 큐를 구현한 예제이다.

[Message.java] Message 클래스

```
package sec06.exam02_queue;

public class Message {
    public String command;
    public String to;

    public Message(String command, String to) {
        this.command = command;
        this.to = to;
    }
}
```

[QueueExample.java] Queue를 이용한 메시지 큐

```
package sec06.exam02_queue;

import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {
    public static void main(String[] args) {
        Queue<Message> messageQueue = new LinkedList<Message>();

        messageQueue.offer(new Message("sendMail", "홍길동")); // 메시지 저장
        messageQueue.offer(new Message("sendSMS", "신용권"));
        messageQueue.offer(new Message("sendKakaotalk", "홍두깨"));

        // while (!messageQueue.isEmpty()) {
        while (messageQueue.peek() != null) { // 메시지 큐가 비었는지 확인
            Message message = messageQueue.poll(); // 메시지 큐에서 한 개의 메시지 개념
            switch (message.command) {
                case "sendMail":
                    System.out.println(message.to + "님에게 메일을 보냅니다.");
                    break;
                case "sendSMS":
                    System.out.println(message.to + "님에게 SMS를 보냅니다.");
                    break;
                case "sendKakaotalk":
                    System.out.println(message.to + "님에게 카카오톡을 보냅니다.");
                    break;
            }
        }
    }
}
```

15.7 동기화된 컬렉션

- 비 동기화된 컬렉션을 동기화된 컬렉션으로 래핑
 - Collections의 synchronizedXXX() 메소드 제공

리턴 타입	메소드(매개 변수)	설명
List<T>	synchronizedList(List<T> list)	List를 동기화된 List로 리턴
Map<K,V>	synchronizedMap(Map<K,V> m)	Map을 동기화된 Map으로 리턴
Set<T>	synchronizedSet(Set<T> s)	Set을 동기화된 Set으로 리턴

```
List<T> list = Collections.synchronizedList(new ArrayList<T>());
Set<E> set = Collections.synchronizedSet(new HashSet<E>());
Map<K,V> map = Collections.synchronizedMap(new HashMap<K,V>());
```

15.8 병렬 처리를 위한 컬렉션

- 동기화(Synchronized) 컬렉션의 단점
 - 하나의 스레드가 요소 처리할 때 전체 잠금 발생

- ConcurrentHashMap
 - ConcurrentLinkedQueue
 - 컬렉션 요소를 병렬처리하기 위해 제공되는 컬렉션
- ConcurrentHashMap
- 부분(segment) 잠금 사용
 - 처리하는 요소가 포함된 부분만 잠금
 - 나머지 부분은 다른 스레드가 변경 가능하게 → 부분 잠금
- ConcurrentLinkedQueue
- 락-프리(lock-free) 알고리즘을 구현한 컬렉션
 - 잠금 사용하지 않음
 - 여러 개의 스레드가 동시에 접근하더라도 최소한 하나의 스레드가 성공하도록(안전하게 요소를 저장하거나 얻도록) 처리

[과제] 확인문제

1. 자바의 컬렉션 프레임워크에 대한 설명으로 틀린 것은 무엇입니까?

- (1) List 컬렉션은 인덱스로 객체를 관리하며 중복 저장을 허용한다.
- (2) Set 컬렉션은 순서를 유지하지 않으며 중복 저장을 허용하지 않는다.
- (3) Map 컬렉션은 키와 값으로 구성된 Map Entry를 저장한다.
- (4) Stack은 FIFO(선입선출) 자료구조를 구현한 클래스이다.

2. List 컬렉션에 대한 설명 중 틀린 것은 무엇입니까?

- (1) 대표적인 구현 클래스로는 ArrayList, Vector, LinkedList가 있다.
- (2) 멀티 스레드 환경에서는 ArrayList보다는 Vector가 스레드에 안전하다.
- (3) ArrayList에서 객체를 삭제하면 삭제된 위치는 비어 있게 된다.
- (4) 중간 위치에 객체를 빈번히 삽입하거나 제거할 경우 LinkedList를 사용하는 것이 좋다.

3. Set 컬렉션에 대한 설명 중 틀린 것은 무엇입니까?


- (1) 대표적인 구현 클래스는 HashSet, LinkedHashSet, TreeSet이 있다.
- (2) Set 컬렉션에서 객체를 하나씩 꺼내고 싶다면 Iterator를 이용한다.
- (3) HashSet은 hashCode()와 equals()를 이용해서 중복된 객체를 판별한다.
- (4) Set 컬렉션에 null을 저장할 수 없다.

4. Map 컬렉션에 대한 설명 중 틀린 것은 무엇입니까?


- (1) 대표적인 구현 클래스로는 HashMap, Hashtable, TreeMap, Properties가 있다.
- (2) HashMap과 Hashtable은 hashCode()와 equals()를 이용해서 중복 키를 판별한다.
- (3) 멀티 스레드 환경에서는 Hashtable보다는 HashMap이 스레드에 안전하다.
- (4) Properties는 키와 값이 모두 String 타입이다.

5. 단일(싱글) 스레드 환경에서 Board 객체를 저장 순서에 맞게 읽고 싶습니다. 가장 적합한 컬

렉션을 생성하도록 밑줄 친 부분에 코드를 작성해 보세요.

_____ #1 _____ 변수 = new _____ #2 _____ ;
(타입) (생성자 호출) 

6. 단일(싱글) 스레드 환경에서 학번(String)을 키로, 점수(Integer)를 값으로 저장하는 가장 적합한 컬렉션을 생성하도록 밑줄 친 부분에 코드를 작성해 보세요.

_____ #1 _____ 변수 = new _____ #2 _____ ;
(타입) (생성자 호출) 

7. BoardDao 객체의 getBoardList() 메소드를 호출하면 List<Board> 타입의 컬렉션을 리턴합니다. ListExample 클래스를 실행시켰을 때 다음과 같이 출력될 수 있도록 BoardDao의 getBoardList() 메소드를 작성해 보세요.

[ListExample.java] BoardDao 사용 클래스

```
01 package verify.exam07;
02
03 import java.util.List;
04
05 public class ListExample {
06     public static void main(String[] args) {
07         BoardDao dao = new BoardDao();
08         List<Board> list = dao.getBoardList();
09         for(Board board : list) {
10             System.out.println(board.getTitle() + "-" + board.getContent());
11         }
12     }
13 }
14
15 // 실행 결과
16 // 제목1-내용1
17 // 제목2-내용2
18 // 제목3-내용3
```

[Board.java] 게시물 클래스

```
01 package verify.exam07;
02
03 public class Board {
04     private String title;
05     private String content;
06
07     public Board(String title, String content) {
08         this.title = title;
09         this.content = content;
10     }
11
12     public String getTitle() { return title; }
13     public String getContent() { return content; }
14 }
```

[BoardDao.java] 게시물을 가져오는 클래스

```
01 package verify.exam07;
```

```

02
03 import java.util.ArrayList;
04 import java.util.List;
05
06 public class BoardDao {
07     // 작성 위치
08
09
10
11
12
13
14 }

```

8. HashSet에 Student 객체를 저장하려고 합니다. 학번이 같으면 동일한 Student라고 가정하고 중복 저장이 되지 않도록 하고 싶습니다. Student 클래스에서 재정의해야 하는 hashCode()와 equals() 메소드의 내용을 채워보세요. Student의 해시코드는 학번이라고 가정합니다.

[HashSetExample.java] Student 중복 저장 방지

```

01 package verify.exam08;
02
03 import java.util.HashSet;
04 import java.util.Iterator;
05 import java.util.Set;
06
07 public class HashSetExample {
08     public static void main(String[] args) {
09         Set<Student> set = new HashSet<Student>();
10
11         set.add(new Student(1, "홍길동"));
12         set.add(new Student(2, "신용권"));
13         set.add(new Student(1, "조민우"));
14
15         Iterator<Student> iterator = set.iterator();
16         while(iterator.hasNext()) {
17             Student student = iterator.next();
18             System.out.println(student.studentNum + ":" + student.name);
19         }
20     }
21 }
22
23 // 실행 결과
24 // 1:홍길동
25 // 2:신용권

```

[Student.java] hashCode()와 equals() 재정의

```

01 package verify.exam08;
02
03 public class Student {
04     public int studentNum;
05     public String name;
06
07     public Student (int studentNum, String name) {
08         this.studentNum = studentNum;
09         this.name = name;
10     }
11
12     @Override

```

```

13         public int hashCode() {
14             // #1 작성 위치
15         }
16     }
17
18     @Override
19     public boolean equals(Object obj) {
20         // #2 작성 위치
21     }
22 }
23

```

9. HashMap에 아이디(String)와 점수(Integer)가 저장되어 있습니다. 실행 결과와 같이 평균 점수를 출력하고, 최고 점수와 최고 점수를 받은 아이디를 출력해보세요.

[Student.java] hashCode()와 equals() 재정의

```

01 package verify.exam09;
02
03 import java.util.HashMap;
04 import java.util.Map;
05 import java.util.Set;
06
07 public class MapExample {
08     public static void main(String[] args) {
09         Map<String,Integer> map = new HashMap<String,Integer>();
10         map.put("blue", 96);
11         map.put("hong", 86);
12         map.put("white", 92);
13
14         String name = null;
15         int maxScore = 0;
16         int totalScore = 0;
17
18         // #1 작성 위치
19
20
21
22
23
24
25
26     }
27 }
28
29 // 실행 결과
30 // 평균점수: 91
31 // 최고점수: 96
32 // 최고점수를 받은 아이디: blue

```

10. TreeSet에 Student 객체를 저장하려고 합니다. Student의 score 필드값으로 자동 정렬하도록 구현하고 싶습니다. TreeSet의 last() 메소드를 호출했을 때 가장 높은 score의 Student 객체가 리턴되도록 Student 클래스를 완성해보세요.

[TreeSetExample.java] 가장 높은 점수 출력

```

01 package verify.exam10;
02

```

```

03 import java.util.TreeSet;
04
05 public class TreeSetExample {
06     public static void main(String[] args) {
07         TreeSet<Student> treeSet = new TreeSet<Student>();
08         treeSet.add(new Student("blue", 96));
09         treeSet.add(new Student("hong", 86));
10         treeSet.add(new Student("white", 92));
11
12         Student student = treeSet.last();
13         System.out.println("최고점수: " + student.score);
14         System.out.println("최고점수를 받은 아이디: " + student.id);
15     }
16 }
17
18 // 실행 결과
19 // 최고점수: 96
20 // 최고점수를 받은 아이디: blue

```

[Student.java]

```

01 package verify.exam10;
02
03 public class Student implements _____ #1 {
04     public String id;
05     public int score;
06
07     public Student (String id, int score) {
08         this.id = id;
09         this.score = score;
10     }
11
12     @Override
13     public int compareTo(Student o) {
14         // #2 작성 위치
15
16
17     }
18 }

```