JDK API

Collection

- Arrays
  - ✓ 배열을 조작하기 위한 메서드를 제공하는 클래스
  - ✓ static 메서드만 소유하고 있는 클래스
  - ✓ 메서드
    - copyOf(복사할 원본 배열, 복사할 개수)
      - 원본 배열에서 복사할 개수 만큼을 복사해서 생성
      - 복사할 개수가 많으면 나머지 부분은 기본값으로 채워서 생성
      - 내부적으로는 System 클래스의 arraycopy 메서드를 호출
    - toString(배열): 배열 각 요소의 toString을 호출해서 [로 묶고 ,로 구분해서 하나의 문자열로 리턴

Collection

- Arrays
  - ✓ 메서드
    - static int binarySearch(Object [] a, Object key): 이분 검색을 이용해서 key를 찾아주는 메서드로 데이터가 정렬이 되어 있어야만 올바른 결과를 리턴하며 검색하고자 하는 데이터가 없는 경우 음수를 리턴
    - static boolean equals(Object [] a, Object [] a2): 지정된 2 개 Object 배열의 내용이 서로 동일한 경우에 true를 리턴
    - static void fill(Object [] a, int fromIndex, int toIndex, Object val): 배열의 fromIndex로부터 toIndex까지 val로 채워줌
    - static void fill(Object [] a, Object val): 배열의 모든 요소를 val로 채워줌
    - void sort(Object [] a): 정렬을 수행해 주는 메서드
      - ◆ 기본 자료형 배열인 경우는 바로 해주지만 기본 자료형이 아닌 참조형인 경우는 클래스가 Comparable 인터페이스를 implements해서 int compare(T o1) 함수가 재정의 되어 있어야 가능
      - ◆ compare 메서드는 호출하는 객체와 매개변수로 대입된 객체 사이의 요소를 비교해서 양수, 0, 음수 셋 중의 하나의 값을 반환하는데 양수가 리턴되면 앞의 데이터가 크다고 생각해서 두 개의 위치를 변경
      - ◆ Comparable 인터페이스를 implements 하지 않은 경우에는 Comparator 인터페이스를 Implements 한 객체를 두번째 매개변수로 대입
    - List asList(Object [] o): 배열을 List 인터페이스를 구현한 객체로 변환해서 리턴

Collection

- java.lang.Comparable<T>
  - ✓ int compareTo(T o) 메서드를 소유한 인터페이스로 Collection 이 데이터를 정렬하기 위해서는 2개의 인스턴스의 크기를 비교해야 하는데 이 때 사용하는 메서드를 소유한 인터페이스
  - ✓ 메서드 내에서 양수가 리턴되면 호출하는 인스턴스가 더 크다고 생각하고 0이 리턴되면 같다고 판단하고 음수가 리턴되면 비교되는 데이터가 더 크다고 생각함
  - ✓ 정렬을 할 때 양수 값이 리턴되면 앞의 데이터가 크다고 생각해서 뒤의 데이터와 자리를 변경하고 그 이외의 경우는 데이터를 그대로 둠
  - ✓ 정수 나 실수 데이터의 경우는 빨셈한 결과나 부등호 연산자를 이용해서 리턴값을 만들어 주어야 함
  - ✓ 숫자가 아닌 경우는 직접 메서드의 내용을 만들어야 하는데 String 이나 Date 클래스에는 Comparable 인터페이스가 구현되어 있어서 크기 비교가 가능한 compareTo 메서드가 존재

Collection

- java.util.Comparator <T>
  - ✓ int compare(T o1, T o2) 메서드를 소유한 인터페이스
  - ✓ 정렬을 위한 sort 메서드의 매개변수로 이 인터페이스를 구현한 클래스의 인스턴스를 대입하면 이 인터페이스의 메서드를 이용해서 크기 비교를 한 후 정렬

Collection

- Arrays
  - ✓ 배열에서의 이분 검색 – Arrays1.java

```
import java.util.*;
public class Arrays1 {

    public static void main(String[] args) {
            String [] ar = {"morning", "afternoon", "evening", "night"};
            //정렬하지 않은 상태에서는 binary search를 제대로 수행하지 못합니다.
            System.out.println("morning의 위치:" + Arrays.binarySearch(ar, "morning"));
            //데이터 정렬 : 배열의 각 멤버가 Comparable 인터페이스의 메서드를 구현해
    야 합니다.
            Arrays.sort(ar);
            System.out.println("morning의 위치:" + Arrays.binarySearch(ar, "morning"));
    }
}
```

Collection

- Arrays
  - ✓ 배열에서의정렬- Arrays2.java

```
[10, 20, 30, 40, 50]
[50, 40, 30, 20, 10]
[50.2, 40.3, 30.0, 20.8, 10.7]
[R, Python, Java, C++, C#]
```

```
import java.util.Arrays;
import java.util.Comparator;

public class Arrays2 {

    public static void main(String[] args) {
        Integer [] ar = {20, 40, 30, 50, 10};
        //정수의 오름차순 정렬
        Arrays.sort(ar);
        System.out.println(Arrays.toString(ar));

        //정수의 내림차순 정렬
        Arrays.sort(ar, new Comparator<Integer>() {

            @Override
            public int compare(Integer o1, Integer o2) {
                return o2 - o1;
            }
        });
        System.out.println(Arrays.toString(ar));
```

Collection

- Arrays
  - ✓ 배열에서의정렬- Arrays2.java

```
Double [] br = {20.8, 40.3, 30.0, 50.2, 10.7};
//실수의 내림차순 정렬
Arrays.sort(br, new Comparator<Double>() {

        @Override
        public int compare(Double o1, Double o2) {
                if(o1 > o2) {
                        return -1;
                }else if(o1 == o2) {
                        return 0;
                }else {
                        return 1;
                }
        }
});
System.out.println(Arrays.toString(br));
```

Collection

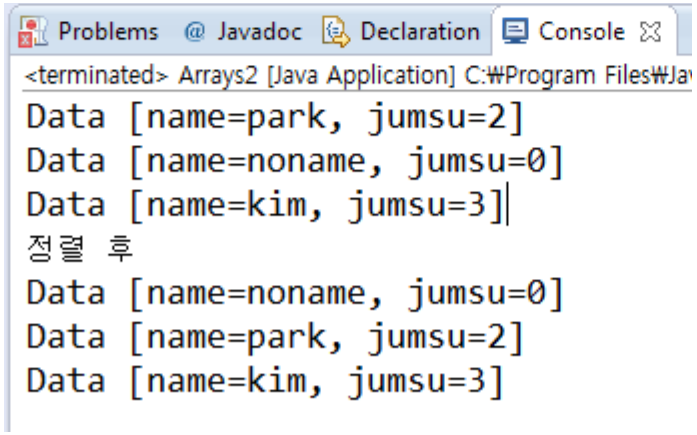
- Arrays
  - ✓ 배열에서의정렬- Arrays2.java

```
                String [] cr = {"Java", "Python", "C++", "C#", "R"};
                //문자열의 내림차순 정렬
                Arrays.sort(cr, new Comparator<String>() {

                        @Override
                        public int compare(String o1, String o2) {
                                return o2.compareTo(o1);
                        }
                });
                System.out.println(Arrays.toString(cr));

        }
}
```

Collection

- Arrays
  - ✓ 사용자 정의 클래스의 정적 정렬– Data.java

```
class Data implements Comparable<Data> {
    private String name;
    private int jumsu;
    public Data() {
            super();
            name = "noname";
    }
    public Data(String name, int jumsu) {
            super();
            this.name = name;
            this.jumsu = jumsu;
    }
```



Collection

- Arrays
  - ✓ 사용자 정의 클래스의 정적 정렬– Data.java

```
public String getName() {
        return name;
}
public void setName(String name) {
        this.name = name;
}
public int getJumsu() {
        return jumsu;
}
public void setJumsu(int jumsu) {
        this.jumsu = jumsu;
}
```

Collection

- Arrays
  - ✓ 사용자 정의 클래스의 정적 정렬- Data.java

```
        public int compareTo(Data obj) {
                if (this.jumsu > obj.jumsu)
                        return 1;
                else
                        return -1;
        }
        @Override
        public String toString() {
                return "Data [name=" + name + ", jumsu=" + jumsu + "]";
        }
}
```

Collection

- Arrays
  - ✓ 사용자 정의 클래스의 정적 정렬– Arrays3.java

```
import java.util.*;
public class Arrays3 {
    public static void main(String args[]) {
            Data data[] = {
                                new Data("lee", 1),
                                new Data("park", 2),
                                new Data("kim", 3),
            };

            for (Data temp : data)
                        System.out.println(temp);
            Arrays.sort(data);
            System.out.println("정렬 후");
            for (Data temp : data)
                        System.out.println(temp);
    }
}
```

Collection

- Arrays
  - ✓ 사용자 정의 클래스의 동적 정렬– Arrays4.java

```
import java.util.*;
public class Arrays4 {
    public static void main(String[] args) {
            Data data[] = {
                                new Data("lee", 1),
                                new Data("park", 2),
                                new Data("kim", 3),
            };
            for(Data temp : data)
                        System.out.println(temp);
            //Comparable 인터페이스를 implements 하지 않은 경우
            Comparator<Data>comp =
                                new Comparator<Data>(){
                                        @Override
                                        public int compare(Data o1, Data o2) {
                                                return
o1.getName().compareTo(o2.getName());
                                        }
            };
```

Collection

- Arrays
  - ✓ 사용자 정의 클래스의 동적 정렬– Arrays4.java

```
                Arrays.sort(data, comp);
                System.out.println("정렬 후");
                for(Data temp : data)
                        System.out.println(temp);
        }
}
```

Collection

- Collection 인터페이스
  - ✓ 여러 개의 요소를 묶어서 하나의 인스턴스로 만든 것을 Collection 이라고 함
  - ✓ 자바에서는 java.util 패키지에 인터페이스 형태로 존재
  - ✓ Set(데이터의 순서를 유지하지 않는 데이터의 집합으로 중복을 허용하지 않음), List(순서가 있는 데이터의 집합으로 중복을 허용)의 상위 인터페이스
  - ✓ 컬렉션 인터페이스를 사용했을 때의 장점
    - 통합된 API 구조로 효율성 증대

Collection

- Collection 인터페이스
  - ✓ 메서드
    - add(Object o), addAll(Collection c)
    - clear()
    - contains(Object o), containsAll(Collection c)
    - equals(Object o)
    - isEmpty()
    - iterator() : 이터레이터 반환
    - remove(Object o), removeAll(Collection c)
    - retainAll(Collection c): Collection에 포함된 데이터를 제외하고 삭제
    - size()
    - toArray(): Object 배열로 반환
    - 데이터의 타입이 Object가 아니고 E, T, K, V로 되어 있으면 Generics가 적용된 것으로 인스턴스를 생성할 때 사용한 Generics 타입을 설정해야 하는데 설정하지 않으면 Object 타입이 되고 설정을 하면 설정한 데이터 타입으로 모두 변경됨

Collection

- Enumeration 과 Iterator
  - ✓ 컬렉션에 저장되어 있는 객체들을 저장방법에 상관없이 접근 할 수 있도록 만든 인터페이스
  - ✓ 반복자라는 단어로 번역하며 데이터베이스에서는 Cursor 라는 표현을 사용하기도 함
  - ✓ Enumeration 인터페이스
    - boolean hasMoreElements(): 다음 요소가 있는지 없는지 여부를 판단해주는 메서드
    - E nextElement (): 열거에 1개 이상의 요소가 남아 있는 경우는 다음의 요소를 리턴
  - ✓ Iterator 인터페이스
    - boolean hasNext(): 다음 요소가 있는 지 여부를 리턴
    - E next(): 반복 처리로 다음의 요소를 리턴
    - void remove(): 마지막으로 요소를 삭제

Collection

- java.util.List
  - ✓ 데이터를 순서대로 저장하는 컬렉션을 구현하는데 사용되는 인터페이스
  - ✓ null을 허용
  - ✓ Generics 적용 – 인스턴스를 만들려고 할 때 요소의 자료형을 결정
  - ✓ 인스턴스를 만들 때 자료형을 결정하지 않으면 경고 발생
  - ✓ 구현된 인터페이스: Collection, Iterable
  - ✓ Vector, ArrayList, LinkedList, Stack 등의 클래스가 List 인터페이스를 implements 한 클래스
  - ✓ toString 메서드를 재정의해서 각 요소의 toString 메서드의 결과를 하나의 문자열로 리턴함

Collection

- java.util.List
  - ✓ 메서드
    - boolean add(E e)
    - void add(int index, Object element): 추가
    - boolean addAll(int index, Collection c)
    - Object set(int index, E element): 수정
    - Object get(index)
    - int indexOf(Object o)
    - int lastIndexOf(Object o)
    - Object remove(int index 또는 Object o)
    - void sort(Comparator c): Comparator의 메서드를 이용해서 정렬
    - List subList(int fromIndex, int toIndex)

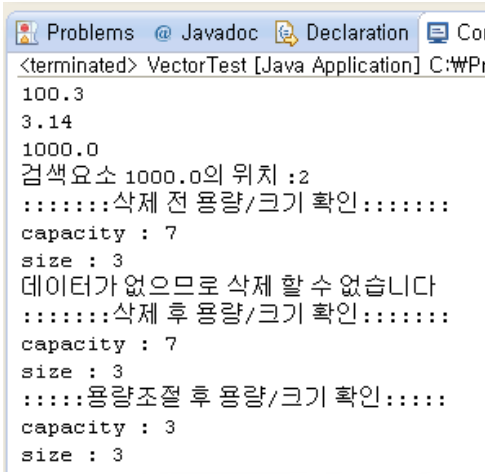
Collection

- java.util.Vector
  - ✓ 참조형 데이터만 저장 가능한 가변 배열
  - ✓ 데이터를 삽입할 때 공간이 부족하면 자동으로 사이즈가 변경되는 List
  - ✓ 벡터는 제너릭 기능을 사용
  - ✓ 생성시 데이터 타입을 결정지어서 사용하는 것이 바람직하며 그렇지 않으면 벡터에서 데이터를 가져올 때 Object 타입으로 리턴하기 때문에 원래의 타입으로 강제 형 변환해서 사용
  - ✓ 멀티 스레드에 대한 동기화가 구현되어 있음 - 동시에 사용하려고 하면 순서대로 접근
  - ✓ 벡터의 생성자
    - Vector()
    - Vector(Collection<? extends E> c)
    - Vector(int initialCapacity)
    - Vector(int initialCapacity, int capacityIncrement)

Collection

- java.util.Vector
  - ✓ VectorTest.java

```
import java.util.Vector;
class VectorTest
{
    public static void main(String[] args)
    {
        Vector<Double> v = new Vector<Double>(2,5);
        v.add(100.3);
        v.add(3.14);
        v.add(1000.0);
        for(Double n : v)
        System.out.println(n);//추가된 요소들 출력
        double search = 1000.0;//검색할 요소
        int index = v.indexOf(search);//검색
        if(index != -1)
            System.out.println("검색요소 "+search+"의 위치 :"+index);
        else
            System.out.println("검색요소 "+search+"가 없습니다.");
```



Collection

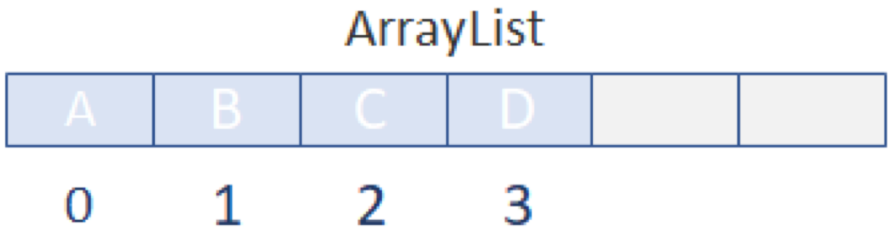
- java.util.Vector
  - ✓ VectorTest.java

```
                System.out.println("::::::삭제 전 용량/크기 확인::::::");
                System.out.println("capacity : "+v.capacity());
                System.out.println("size : "+v.size());
                double del = 3.17;//삭제할 요소
                if(v.contains(del)){
                        v.remove(del);//삭제
                        System.out.println(del+"삭제 완료!");
                }
                else{
                        System.out.println("데이터가 없으므로 삭제 할 수 없습니다");
                }
                System.out.println("::::::삭제 후 용량/크기 확인::::::");
                System.out.println("capacity : "+v.capacity ());
                System.out.println("size : "+v.size ());
                v.trimToSize ();
                System.out.println(":::::용량조절 후 용량/크기 확인:::::");
                System.out.println("capacity : "+v.capacity ());
                System.out.println("size : "+v.size ());
        }
}
```

Collection

- java.util.ArrayList
  - ✓ 구현된 인터페이스: Serializable, Cloneable, Iterable, Collection, List, RandomAccess
  - ✓ 생성자
    - ArrayList()
    - ArrayList(Collection c)
    - ArrayList(int initialCapacity)
  - ✓ 메서드는 Vector와 거의 유사
  - ✓ 요소에 대한 접근은 다른 리스트 기반 클래스보다 빠르지만 요소가 삽입될 때 추가될 공간을 만들기 위해 객체를 이동시켜야 하고 삭제를 할 때는 삭제된 공간을 없애기 위해서 요소들을 이동시켜야 하므로 요소의 삽입과 삭제는 느림
  - ✓ 벡터와 다른 점은 동기화를 지원하지 않으므로 속도는 빠르지만 멀티 스레드 환경에서 사용할 때는 직접 동기화를 구현하거나 동기화된 List 클래스를 사용
  - ✓ 동기화가 필요하다면 Collections 클래스의 동기화 메서드를 이용해서 생성

```
List list = Collections.synchronizedList(new ArrayList());
```



Collection

- java.util.ArrayList
  - ✓ ArrayList 사용

```
import java.util.ArrayList;
public class TestArrayList {
    public static void main(String args[]) {
            ArrayList<String> sm = new ArrayList<String>();
            sm.add("SES");
            sm.add("소녀시대");
            sm.add("레드벨벳");
            System.out.println("sm의 크기" + sm.size()); // 결과 : al의 크기3
            sm.add("에스파");
            for (int a = 0; a < sm.size(); a++)
                    System.out.println(sm.get(a));
            System.out.println("================");
            System.out.println("sm의 크기" + sm.size()); // 결과 : al의 크기4
            sm.remove(1);
            System.out.println("sm의 크기 " + sm.size()); // 결과 : al의 크기 3
            for (int a = 0; a < sm.size(); a++)
                    System.out.println(sm.get(a));
    }
}
```

Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

```
class Team
{
    //팀의 번호를 자동으로 1증가시키도록 하기 위한 static 변수
    static int number;
    //static 초기화
    static { number = 0;}

    public int teamnumber;
    public String teamname;
    public int win;
    public int tie;
    public int defeat;
    //win * 3 + tie * 1 로 계산되는 파생 속성
    public int points;
    //선수 명단
    public String [] players;
```

Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

```
//생성자 – 일련번호 생성
public Team() {
        number = number + 1;
        teamnumber = number;
}
```

Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

```
public static int getNumber() {
        return number;
}

public static void setNumber(int number) {
        Team.number = number;
}

public int getTeamnumber() {
        return teamnumber;
}

public void setTeamnumber(int teamnumber) {
        this.teamnumber = teamnumber;
}
```

Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

```
public String getTeamname() {
        return teamname;
}

public void setTeamname(String teamname) {
        this.teamname = teamname;
}

public int getWin() {
        return win;
}

public void setWin(int win) {
        this.win = win;
}
```

Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

```
public int getTie() {
        return tie;
}

public void setTie(int tie) {
        this.tie = tie;
}

public int getDefeat() {
        return defeat;
}

public void setDefeat(int defeat) {
        this.defeat = defeat;
}
```

Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

```
//points는 파생 속성이므로 값을 가져갈 때 계산
public int getPoints() {
        points = 3 * win + 1 * tie;
        return points;
}

//이 메서드는 필요 없음 직접 설정할 필요가 없기 때문
public void setPoints(int points) {
        this.points = points;
}

public String[] getPlayers() {
        return players;
}

public void setPlayers(String[] players) {
        this.players = players;
}
```

Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

```
        //디버깅을 위한 메서드
        @Override
        public String toString() {
                return "Team [teamnumber=" + teamnumber + ", teamname=" +
        teamname + ", win=" + win + ", tie=" + tie
                                        + ", defeat=" + defeat + ", points=" + points + ",
        players=" + Arrays.toString(players) + "]";
        }
    }
```

Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

```
import java.util.ArrayList;

public class TeamList {
    public static void main(String[] args) {
            ArrayList<Team> epl = new ArrayList<Team>();

            //데이터 추가하기
            Team team = new Team();
            team.setTeamname("토트넘");
            team.setWin(5);
            team.setTie(1);
            team.setDefeat(5);
            team.setPlayers(new String[] {"손흥민", "케인"});

            epl.add(team);
```

Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

```
//새로운 데이터 생성해서 추가
team = new Team();
team.setTeamname("첼시");
team.setWin(8);
team.setTie(2);
team.setDefeat(1);
team.setPlayers(new String[] {"루카쿠", "햐베르츠"});

epl.add(team);


team = new Team();
team.setTeamname("맨시티");
team.setWin(7);
team.setTie(2);
team.setDefeat(2);
team.setPlayers(new String[] {"페르난지뉴", "권도안"});

epl.add(team);
```

Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

```
team = new Team();
team.setTeamname("맨유");
team.setWin(5);
team.setTie(2);
team.setDefeat(4);
team.setPlayers(new String[] {"호날두", "포그바"});

epl.add(team);
```

Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

```
//전체 데이터 개수
System.out.println("팀 수:" + epl.size());

System.out.println("==========================================
======");

//데이터 전체 읽어오기
//빠른 열거 사용
for(Team t : epl) {
        System.out.print(t.getTeamname() + "₩t");
        System.out.print(t.getPoints() + "₩t");
        //내부에 배열이나 List가 있으면 다시 순회
        for(String player : t.getPlayers()) {
                System.out.print(player + " ");
        }
        System.out.println();
}

System.out.println("==========================================
======");
```

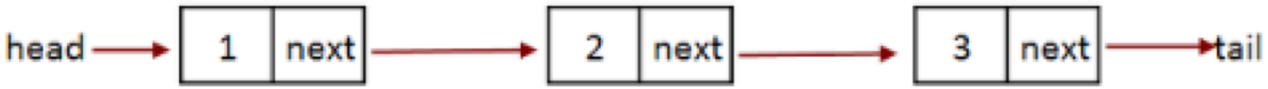
Collection

- java.util.ArrayList
  - ✓ ArrayList 사용 – 사용자 정의 데이터(Team) 사용

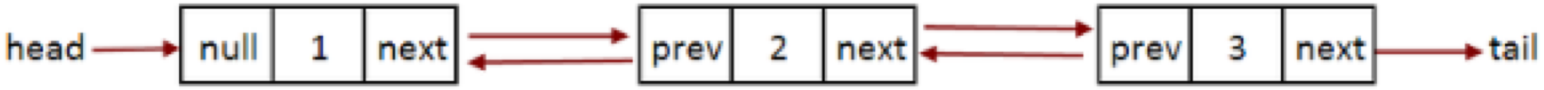
```
                //데이터 1개 가져오기
                //데이터 1개를 가져올 때는 인덱스 나 구분에 사용하는 키를
                //알아야 합니다.
                System.out.println(epl.get(0));


                System.out.println("==========================================
            ======");

            }
        }
```

Collection

- LinkedList



Singly Linked List



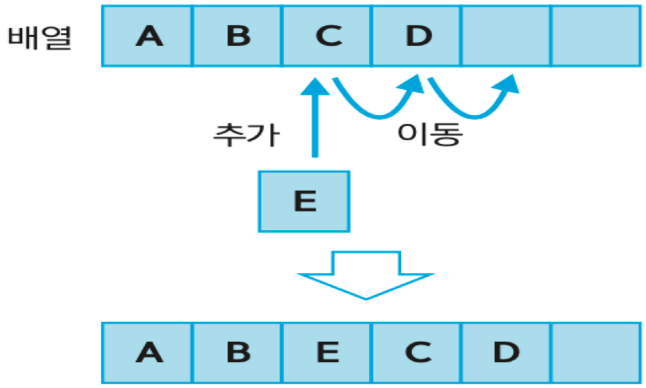
Doubly Linked List

Collection

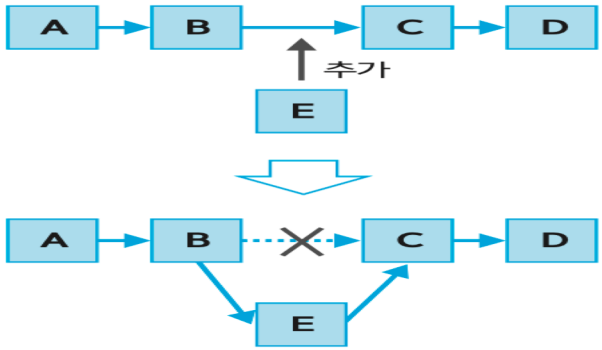
- LinkedList
  - ✓ 구현된 인터페이스: Serializable, Cloneable, Iterable, Collection, List, Queue
  - ✓ 연결된 노드들을 기반으로 구현된 리스트
  - ✓ 링크가 반드시 다른 노드에 연결되어 함
  - ✓ LinkedList는 시작과 끝에 있는 요소를 가져올 수 있고 중간 위치의 데이터도 삽입이나 삭제를 할 수 있음
  - ✓ 중간에 위치한 데이터의 삽입과 삭제하는 속도는 빠르지만 검색에는 취약한 자료구조이며 순차적으로 삽입과 삭제를 하면 ArrayList가 빠름
  - ✓ 동기화를 지원하지 않음
  - ✓ 메서드
    - boolean add (E e): 리스트의 마지막에 데이터를 추가
    - void addFirst (E e): 리스트의 시작에 데이터를 삽입
    - void addLast (E e): 리스트의 마지막에 데이터를 추가
    - E get (int index): 리스트내의 지정된 위치에 있는 데이터를 리턴
    - E getFirst (): 시작 데이터를 리턴
    - E getLast (): 마지막 데이터를 리턴
    - E poll (): 리스트의 시작 위치에 있는 데이터를 삭제

Collection

- LinkedList
  - ✓ ArrayList



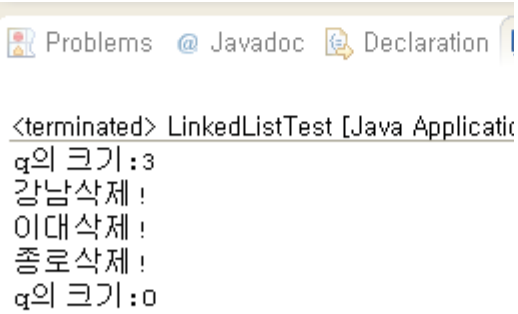
  - ✓ LinkedList



Collection

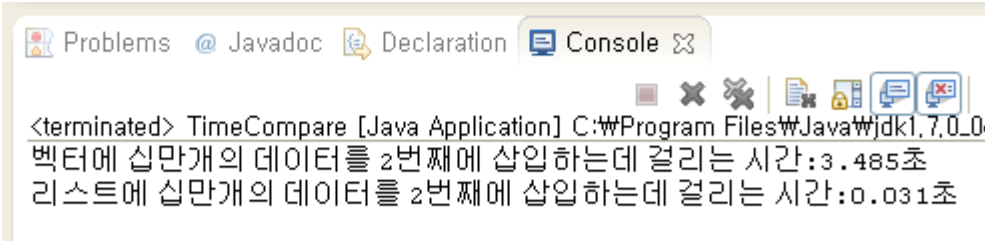
- LinkedList
  - ✓ LinkedList 생성 및 활용

```
import java.util.*;

class LinkedListTest
{
    public static void main(String[] args)
    {
            String[] item = {"강남","이대","종로"};
            LinkedList<String> q = new LinkedList<String>();
            for(String n : item)
                    q.add(n);
            System.out.println("q의 크기:"+q.size());
            String data="";
            while((data = q.poll()) != null)
                    System.out.println(data+"삭제!");
            System.out.println("q의 크기:"+q.size());
    }
}
```



Collection

- LinkedList
  - ✓ ArrayList 와 LinkedList




```
import java.util.*;

class TimeCompare
{
    public static void main(String[] args)
    {
            int i;
            long start, end;
            Vector <String> vec = new Vector<String>();
            vec.add("1");
            vec.add("3");
            vec.add("4");
            start = System.nanoTime();
            for(i=0; i<100000; i++)
            {
                    vec.add(2, "2");
            }
            end = System.nanoTime();
```

Collection

- LinkedList
  - ✓ ArrayList 와 LinkedList

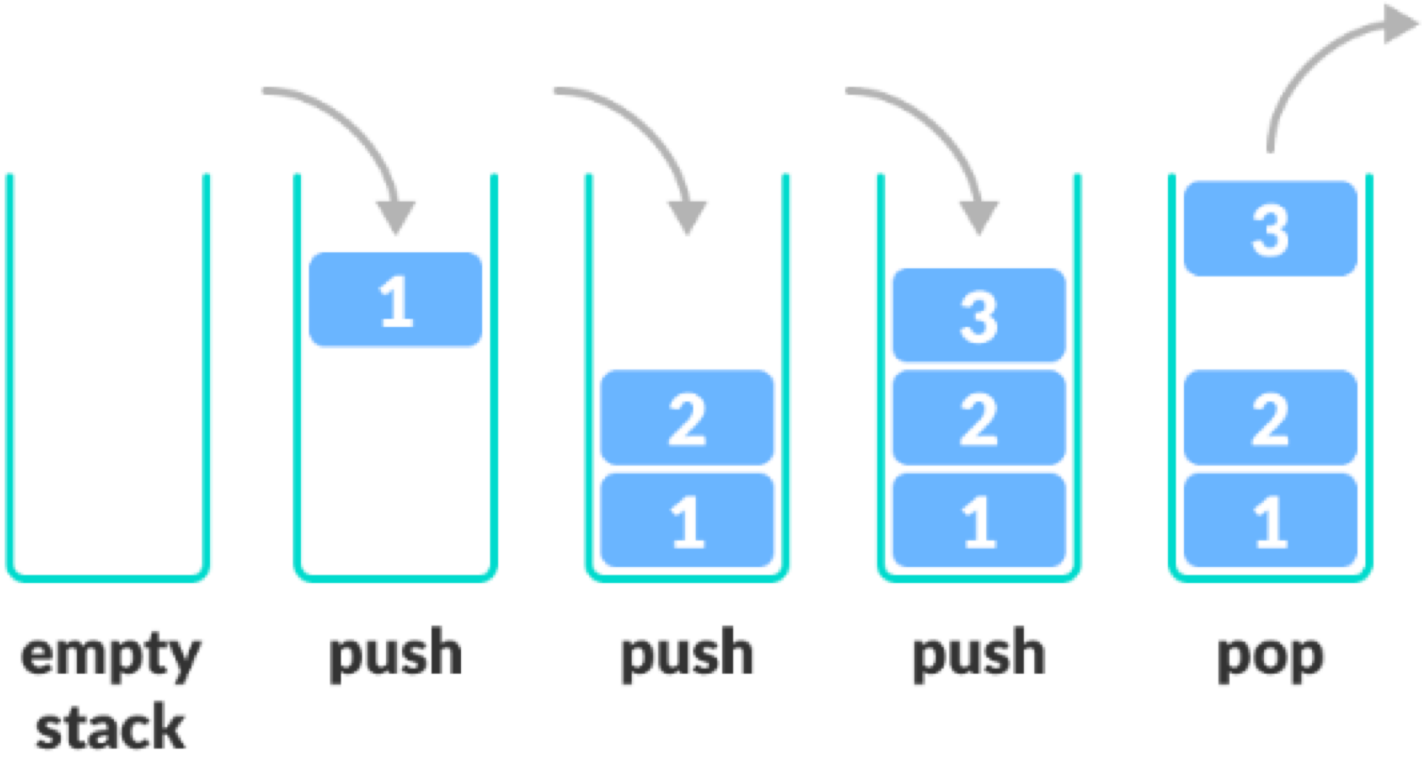
```
System.out.println("벡터에 십만개의 데이터를 2번째에 삽입하는데 걸리는 시간:" +
    (end-start)/1000000000.0 + "초");
            LinkedList <String> li = new LinkedList<String>();
            li.add("1");
            li.add("3");
            li.add("4");
            start = System.nanoTime();
            for(i=0; i<100000; i++)
            {
                        li.add(2, "2");
            }
            end = System.nanoTime();
            System.out.println("리스트에 십만개의 데이터를 2번째에 삽입하는데 걸리는
    시간:" + (end-start)/1000000000.0 + "초");
    }
}
```

Collection

- CopyOnWriteArrayList(java.util.concurrent.CopyOnWriteArrayList)
  - ✓ 여러 개의 스레드로부터 동시에 액세스해도 올바로 처리되는 ArrayList 클래스의 확장
  - ✓ ArrayList 클래스에서는 특정 스레드에서 iteration이나 for-each문을 사용한 루프를 실행하고 있을 때에 별개의 스레드가 그 ArrayList 객체의 요소를 변경하면 ConcurrentModificationException이 발생하기 때문에 ArrayList는 여러 스레드로부터 변경 조작을 포함하는 액세스를 할 수 없음
  - ✓ CopyOnWriteArrayList 클래스는 iterator나 for-each문을 사용한 루프를 실시할 때에 원래의 리스트의 복사본을 작성하여 그 복사본에 대해서 루프를 실시하기 때문에 다른 스레드가 리스트의 조작을 실시해도 루프 쪽의 리스트에 영향을 미치지 않아 예외가 발생하지 않음
  - ✓ 하나의 리스트 객체에 대해 여러 스레드가 동시에 액세스할 가능성이 있는 경우는 CopyOnWriteArrayList 클래스를 사용
  - ✓ CopyOnWriteArrayList 클래스는 성능면에서 ArrayList와 거의 비슷한데 요소를 추 가/변경/삭제할 경우 CopyOnWriteArrayList에서는 내부적으로 갖고 있는 배열을 복사하기 때문에 ArrayList에 비해 성능이 나빠짐
  - ✓ 그 외의 List 구현 클래스로는 리스트 객체를 읽기 전용으로 하는 UnmodifiableList 클래스(Collections 클래스의 unmodifiableList 메서드로 생성) 등이 있음

Collection

- Stack



Collection

- Stack
  - ✓ 구현된 인터페이스: Serializable, Cloneable, Iterable <E>, Collection <E>, List <E>, RandomAccess
  - ✓ LIFO(Last In First Out – 후입선출) 구조로 삽입과 삭제가 한 쪽 끝에서 발생하는 구조
  - ✓ 생성자
    - Stack<(): 비어있는 스택 생성
  - ✓ 메서드
    - boolean empty(): 스택이 비어 있으면 true 리턴
    - E peek(): 삭제 없이 마지막 데이터 리턴
    - E pop(): 마지막 데이터를 삭제하고 마지막 데이터 리턴
    - E push(E item): 데이터 삽입
    - int search(Object o): o의 위치를 리턴하고 없으면 -1을 리턴

Collection

- Stack
  - ✓ Stack 사용

```
import java.util.*;
class TestStack {
    public static void main(String args[]){
            Stack <String> stack=new Stack<String>();
            stack.push("1");
            stack.push("2");
            System.out.println("Top이 가리키는 데이터:" + stack.pop());
            System.out.println("Top이 가리키는 데이터:" + stack.pop());

            stack.push("1");
            stack.push("2");
            System.out.println("Top이 가리키는 데이터:" + stack.peek());
            System.out.println("Top이 가리키는 데이터:" + stack.peek());

            System.out.println("검색하고자 하는 데이터가 있는 위치:" +
    stack.search("1"));
            System.out.println("검색하고자 하는 데이터가 있는 위치:" +
    stack.search("3"));
    }
}
```

Collection

- Stack
  - ✓ 괄호 체크

```
import java.util.Stack;

public class BracketCheck {

    public static void main(String[] args) {
        Stack <Character>st = new Stack<>();
        //String expression = "(((())))";
        //String expression = "((()))";
        String expression = "((()))))";
```

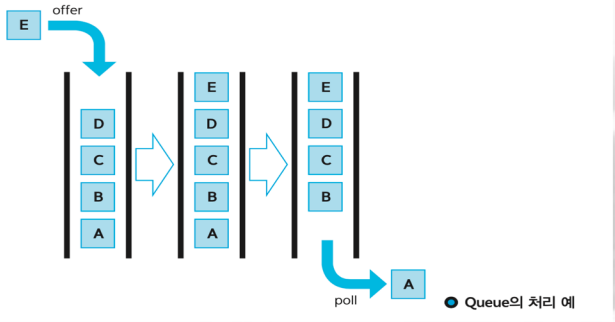
Collection

- Stack
  - ✓ 괄호 체크

```
String expression = "((()))";
try {
        for (int i = 0; i < expression.length(); i++) {
                char ch = expression.charAt(i);

                if (ch == '(') {
                        st.push(ch);
                } else if (ch == ')') {
                        st.pop();
                }
        }
        if (st.isEmpty()) {
                System.out.println("괄호가 일치합니다.");
        } else {
                System.out.println("괄호가 일치하지 않습니다.");
        }
} catch (Exception e) {
        System.out.println("괄호가 일치하지 않습니다.");
}
    }
}
```

Collection

- Queue
  - ✓ FIFO(First In First Out – 선입 선출) 구조로 만들어진 자료구조로 자바에서는 인터페이스 형태로 제공
  - ✓ Queue 객체는 직접 생성할 수 없고 이 인터페이스를 implements 한 클래스의 객체로 생성
  - ✓ 메서드
    - boolean add(E e)
    - E element()
    - boolean offer(E e): 데이터 추가
    - E peek(): 값을 삭제하지 않고 첫번째 데이터 리턴
    - E poll(): 값을 삭제하고 첫번째 데이터 리턴
    - E remove()



Collection

- Queue
  - ✓ PriorityQueue
    - Queue를 implements 한 PriorityQueue는 데이터를 오름차순으로 정렬하는 것처럼 삽입
    - 삽입하는 객체는 반드시 Comparable 인터페이스를 implements 해야 함

Collection

- Queue
  - ✓ LinkedList를 이용한 Queue

```
첫번째 요소:Morning
큐의 크기:2
첫번째 요소:Morning
큐의 크기:1
```

```
import java.util.*;

public class TestQueue {

    public static void main(String[] args) {
            Queue <String> queue=new LinkedList<String>();
            queue.add("Morning");
            queue.add("Afternoon");
            System.out.println("첫번째 요소:" + queue.peek());
            System.out.println("큐의 크기:" + queue.size());
            System.out.println("첫번째 요소:" + queue.poll());
            System.out.println("큐의 크기:" + queue.size());
    }
}
```

Collection

- Queue
  - ✓ 우선 순위 큐

10 11 13 26 31 46 55 57 62 68 70 77 78 80 91

```
import java.util.PriorityQueue;
import java.util.Queue;

class PriorityQueueTest
{
   public static void main(String[] args)
   {
      Queue<Integer> qi = new PriorityQueue<>();
      for (int i = 0; i < 15; i++)
         qi.add((int) (Math.random()*100));
      while (!qi.isEmpty())
         System.out.print(qi.poll()+" ");
      System.out.println();
   }
}
```

Collection

- Deque
  - ✓ Deque는 양쪽에서 삽입과 삭제가 가능한 자료구조를 표현한 인터페이스
  - ✓ ArrayDeque는 Deque 인터페이스를 implements 한 대표적인 클래스

Collection

- List 정렬
  - ✓ void sort(Comparator <?> comp): 정렬할 수 있는 인스턴스 메서드 제공
    - Comparator를 구현해서 대입하면 리스트를 정렬
  - ✓ Collections.sort(List list): Comparable 인터페이스를 implements 한 객체의 List
  - ✓ Collections.sort(List list, Comparator<?> comp): Comparable 인터페이스를 implements 하지 않은 객체의 List

Collection

- List 정렬
  - ✓ 사용자 정의 데이터 정렬

```
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Data [] data = {
                new Data("lee", 1),
                new Data("park", 2),
                new Data("kim", 3),
        };
        List <Data> list = Arrays.asList(data);
```

Collection

- List 정렬
  - ✓ 사용자 정의 데이터 정렬

```
        //Comparable 인터페이스를 implements 하지 않은 경우
        Comparator<Data>comp =
                new Comparator<Data>(){
                        @Override
                        public int compare(Data o1, Data o2) {
                                return
    o1.getName().compareTo(o2.getName());
                        }
                };
        list.sort(comp);
        System.out.println("정렬 후");
        for(Data temp : data)
                System.out.println(temp);
    }
}
```

Collection

- Set
  - ✓ 중복을 허용하지 않고 저장 순서가 없는 컬렉션 인터페이스
  - ✓ 단방향으로 순서대로 접근할 수 있는 SortedSet 과 양방향으로 접근할 수 있는 NavigableSet의 상위 인터페이스
  - ✓ 구현된 클래스
    - HashSet: 해싱한 순서대로 리턴
    - LinkedHashSet: 저장한 순서대로 리턴
    - TreeSet: 오름차순 정렬를 수행한 순서대로 리턴

Collection

- Set
  - ✓ 메서드
    - boolean add(E e)
    - boolean addAll(Collection<? extends E> c)
    - void clear()
    - boolean contains(Object o)
    - boolean containsAll(Collection<?> c)
    - boolean equals(Object o)
    - int hashCode()
    - boolean isEmpty()
    - Iterator<E> iterator()
    - boolean remove(Object o)
    - boolean removeAll(Collection<?> c)
    - boolean retainAll(Collection<?> c)
    - int size()
    - Object[] toArray()
    - <T> T[] toArray(T[] a)

Collection

- Set
  - ✓ HashSet

```
import java.util.*;
public class HashSetTest {
    public static void main(String args[]) {
            String[] str = {"2","3","3","1"};
            HashSet<String> hs = new HashSet<String>();

            for (String n : str){
                    if (!hs.add(n))
                    System.out.println(n + "이 존재해서 삽입하는데 실패");
            }

            //전체 데이터 출력
            System.out.println("hs : " + hs);
            System.out.println("================");
            //각각 데이터 접근
            for (String n : hs){
                    System.out.println(n);
            }
```

```
Problems  @ Javadoc  Declaration
<terminated> HashsetTest [Java Applicati
3이 존재해서 삽입하는데 실패
hs : [3, 2, 1]
================
3
2
1
```

Collection

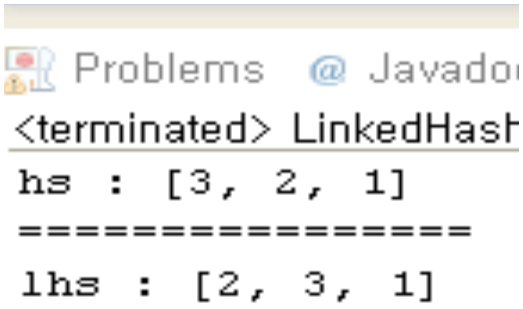
- Set
  - ✓ 메서드

```
                System.out.println("================");
                Iterator<String>it = hs.iterator();
                while(it.hasNext()){
                        System.out.println(it.next());
                }
            }
        }
```

Collection

- Set
  - ✓ LinkedHashSet

```
import java.util.*;
public class LinkedHashSetTest {
    public static void main(String args[]) {
            String[] str = {"2","3","1"};
            HashSet<String> hs = new HashSet<String>();
            for (String n : str){
                    if (!hs.add(n))
                    System.out.println(n + "이 존재해서 삽입하는데 실패");
            }
            //전체 데이터 출력
            System.out.println("hs : " + hs);
            System.out.println("================");
            LinkedHashSet<String> lhs = new LinkedHashSet<String>();
            for (String n : str){
                    if (!lhs.add(n))
                    System.out.println(n + "이 존재해서 삽입하는데 실패");
            }
            //전체 데이터 출력
            System.out.println("lhs : " + lhs);
    }
}
```



Collection

- Set
  - ✓ TreeSet

```
import java.util.*;

public class TreeSetTest {
  public static void main(String[] args) {
    TreeSet<Integer> tSet = new TreeSet<Integer>();

    tSet.add(3);
    tSet.add(1);
    tSet.add(2);

    System.out.println(tSet);

    System.out.println("모든 데이터 순서대로 각각 출력하기");

    Iterator <Integer> it = tSet.iterator();
    while(it.hasNext()){
     System.out.println(it.next());
    }
  }
}
```

```
<terminated> TreeSetTest [Java Application]
[1, 2, 3]
모든 데이터 순서대로 각각 출력하기
1
2
3
```

Collection

- Set
  - ✓ Lotto

```
import java.util.TreeSet;
public class LottoTest {
    public static void main(String[] args) {
            //1-45 사이의 랜덤한 숫자 6개를 저장해서
            //데이터를 정렬한 후 출력
            //중복된 데이터 없이(Set)
            //숫자를 크기 순서대로 저장(Tree)
            //선택해야 하는 자료구조는 TreeSet
            //TreeSet의 데이터가 6개가 될 때 까지 1-45 사이의
            //숫자를 입력받으면 됩니다.
            TreeSet<Integer>lotto =
                                    new TreeSet<Integer>();
            while(lotto.size() != 6){
                    lotto.add((int)(Math.random()*100)%45 + 1);
            }
            for(int imsi : lotto)
                    System.out.print(imsi + "₩t");
    }
}
```

Map

- Map
  - ✓ Key와 Value를 하나의 쌍으로 묶어서 저장하는 컬렉션 인터페이스
  - ✓ Key는 중복될 수 없지만 Value는 중복될 수 있음
  - ✓ 동일한 Key에 새로운 Value를 삽입하면 기존 Value는 제거되고 새로운 Value로 갱신
  - ✓ HashMap, Hashtable, LinkedHashMap, TreeMap 클래스가 Map 인터페이스를 implements 한 클래스

Map

- Map
  - 메서드
    - void clear()
    - boolean containsKey(Object key)
    - boolean containsValue(Object value)
    - Set<Map.Entry<K,V>> entrySet()
    - boolean equals(Object o)
    - V get(Object key)
    - int hashCode()
    - boolean isEmpty()
    - Set<K> keySet()
    - V put(K key, V value)
    - void putAll(Map<? extends K,? extends V> m)
    - V remove(Object key)
    - int size()
    - Collection<V> values()

Map

- HashMap
  - ✓ Key와 Value를 매핑하는 클래스
  - ✓ Key는 절대 중복될 수 없으며 각 Key는 1개의 Value만 매칭 – key는 equals 메서드를 이용해서 비교
  - ✓ 검색은 Key로 하므로 Key를 모르면 검색할 수 없음
  - ✓ 데이터를 가져올 때 Key에 해당하는 데이터가 없으면 null
  - ✓ Key와 Value에 null을 사용할 수 있음
  - ✓ HashTable은 Key와 Value에 NULL이 들어갈 수 없음
  - ✓ Key의 비교를 == 이용해서 저장하는 IdentityHashMap 과 Key를 저장할 때 강한 참조가 아니라 약한 참조를 이용하는 WeakHashMap 클래스도 제공

Map

- HashMap
  - ✓ 생성자
    - HashMap(): 초기 용량을 16으로 하고 적재율은 0.75로 하여 HashMap 객체 생성
    - HashMap(용량): 초기 용량을 용량으로 하고 기본 적재율 0.75.로 적재
    - HashMap(용량, 적재율): 초기 용량을 용량으로 하고 기본 적재율로 적재

      HashMap<자료형, 자료형> 객체명 = new HashMap< 자료형, 자료형 >();
  - ✓ 메서드

● void	clear()	모든 매핑 삭제
● Value	get(Object Key)	Value 반환
● boolean	isEmpty()	비어있으면 true 반환
● Set<K>	keySet()	Key를 설정
● Value	put(key, value)	데이터 삽입
● Value	remove(key)	데이터 삭제
● int	size()	매핑 수 반환

Map

- HashMap
  - ✓ HashMap 사용

```
import java.util.*;
class HashMapTest
{
    public static void main(String[] args)
    {
        String[] msg = {"서울","부산",null, "목포","제주"};
        HashMap<Integer, String> map =  new HashMap<Integer,String>();
        for(int i=0 ; i<msg.length ; i++)
        {
            map.put(i,msg[i]);//맵에 저장
        }
        for(int i=0; i<map.size(); i++)
        {
            Integer n = i;
            System.out.println(map.get(n));
        }
    }
}
```

```
<terminatec
서 울
부 산
null
목 포
제 주
```

Map

- HashMap
  - ✓ HashMap은 데이터를 저장하는 클래스의 대용으로 사용할 수 있음
  - ✓ 클래스는 멤버를 사용하기 위해서 멤버를 직접 호출해야 사용할 수 있음
  - ✓ HashMap은 Iterator를 이용하면 멤버를 직접 호출하지 않고도 저장된 모든 데이터에 접근이 가능
  - ✓ 클래스를 사용하면 인스턴스 변수의 이름이 변경된 경우 클래스는 인스턴스를 사용하는 모든 부분을 전부 수정해야 하지만 HashMap은 저장하는 부분만 수정이 필요하고 출력하는 부분에서는 수정 작업이 발생하지 않도록 할 수 있음
  - ✓ 현재 HashMap은 iBatis(MyBatis), Spring 등에서 데이터베이스를 사용할 때 객체 단위의 입력이나 출력 등에도 사용되고 있으며 Hadoop 이나 MongoDB에서도 데이터를 Map 단위로 다루고 있음

Map

- HashMap
  - ✓ HashMap은 데이터를 저장하는 클래스의 대용으로 사용할 수 있음

```
public class DTO {
    private String num;
    private String name;
    private String address;

    public DTO() {
            super();
    }
    public DTO(String num, String name, String address) {
            super();
            this.num = num;
            this.name = name;
            this.address = address;
    }
```

Map

- HashMap
  - ✓ HashMap은 데이터를 저장하는 클래스의 대용으로 사용할 수 있음

```
public String getNum() {
        return num;
    }
    public void setNum(String num) {
        this.num = num;
    }

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }

}
```

Map

- HashMap
  - ✓ HashMap은 데이터를 저장하는 클래스의 대용으로 사용할 수 있음

```
import java.util.*;
public class MapMain {
    public static void main(String[] args) {
            DTO obj1 = new DTO("1", "jesica", "ChristChurch");
            System.out.println("번호:"+obj1.getNum());
            System.out.println("이름:"+obj1.getName());
            System.out.println("주소:"+obj1.getAddress());
            System.out.println("============================");

            HashMap<String, String>obj2 = new HashMap<String, String>();
            obj2.put("번호", "2");
            obj2.put("이름", "제시카");
            obj2.put("주소","크라이스처치");
            Set<String>set = obj2.keySet();
            Iterator<String>it = set.iterator();
            String key = "";
            while(it.hasNext()){
                    key = it.next();
                    System.out.println(key + ":" + obj2.get(key));
            }
    }
}
```

Map

- HashMap
  - ✓ HashMap은 2차원 이상의 배열을 표현할 때도 유용
  - ✓ 2차원 배열은 각각의 배열의 의미를 표현하지 못하고 각각의 배열의 인덱스만을 가지고 있음
  - ✓ 이에 반해 1차원 배열을 키를 설정해서 저장하게 되면 출력 시에 키를 보고 의미를 판단할 수 있음

Map

- HashMap
  - ✓ HashMap을 이용한 2차원 배열

```
import java.util.*;
public class MenuHash {
    public static void main(String[] args) {
            String [] file = {"Open", "Close", "Save"};
            String [] edit = {"Copy", "Cut", "Paste"};
            String [] view = {"Max", "Min", "Normal"};
            String [][]menu;
            menu = new String[3][];
            menu[0] = file;
            menu[1] = edit;
            menu[2] = view;
            String temp[];
            for(int i=0; i<menu.length; i++){
                    if(i==0)     System.out.print("파일:");
                    else if(i==1) System.out.print("편집:");
                    else System.out.print("보기:");
                    temp = menu[i];
                    for(String imsi : temp)
                            System.out.printf("%-10s", imsi);
                    System.out.println();
            }
```

```
파일:Open      Close     Save
편집:Copy      Cut       Paste
보기:Max       Min       Normal
================================
================================
보기:Max       Min       Normal
파일:Open      Close     Save
편집:Copy      Cut       Paste
```

Map

- HashMap
  - ✓ HashMap을 이용한 2차원 배열

```
                System.out.println("================================");
                System.out.println("================================");
                HashMap <String, String[]>menuMap = new HashMap<String, String[]>();
                menuMap.put("파일", file);
                menuMap.put("편집", edit);
                menuMap.put("보기", view);
                Set<String>set = menuMap.keySet();
                Iterator<String>it = set.iterator();
                String key = "";
                String menuTemp[];
                while(it.hasNext()){
                        key = it.next();
                        System.out.print(key + ":");
                        menuTemp = menuMap.get(key);
                        for(String imsi:menuTemp){
                                System.out.printf("%-10s", imsi);
                        }
                        System.out.println();
                }
        }
}
```

Map

- TreeMap
  - ✓ TreeMap은 HashMap과 달리 Key가 오름차순 정렬이 되어 있는 형태의 Map으로 SortedMap 인터페이스를 implements 한 클래스
  - ✓ TreeMap은 key가 이진 트리로 구성
  - ✓ NavigableMap은 SortedMap의 하위 인터페이스로 접근 방법을 오름차순과 내림차순 모두 제공

Map

- LinkedHashMap
  - ✓ Key의 순서를 기억하는 Map

Map

- LinkedHashMap
  - ✓ 이차원 배열

```
import java.util.*;

public class MenuTree {
    public static void main(String[] args) {
            String [] file = {"Open", "Close", "Save"};
            String [] edit = {"Copy", "Cut", "Paste"};
            String [] view = {"Max", "Min", "Normal"};
            TreeMap <String, String[]>menuMap = new TreeMap<String, String[]>();
            menuMap.put("file", file);
            menuMap.put("edit", edit);
            menuMap.put("view", view);
```

Map

- LinkedHashMap
  - ✓ 이차원 배열

```
Set<String>set = menuMap.keySet();
        Iterator<String>it = set.iterator();
        String key = "";
        String menuTemp[];
        while(it.hasNext()){
                key = it.next();
                System.out.print(key + ":");
                menuTemp = menuMap.get(key);
                for(String imsi:menuTemp){
                        System.out.printf("%-10s", imsi);
                }
                System.out.println();
        }
    }
}
```

Map

- HashMap
  - ✓ DataFrame

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;
public class MapDataFrame {

    public static void main(String[] args) {
        List<Map<String, Object>> list = new ArrayList<>();

        //1개의 데이터를 생성해서 추가하기
        Map<String, Object> map = new HashMap<>();
        map.put("name", "Java");
        map.put("description", "프로그램을 만들기 위한 언어");
        map.put("days", 15);
        list.add(map);
```

Map

- HashMap
  - ✓ DataFrame

```
map = new HashMap<>();
map.put("name", "Database");
map.put("description", "데이터를 효율적으로 관리하기 위한 SW");
map.put("days", 5);
list.add(map);
map = new HashMap<>();
map.put("name", "WebFrontEnd");
map.put("description", "웹 브라우저에 출력하기 위한 기술");
map.put("days", 7);
list.add(map);
map = new HashMap<>();
map.put("name", "JavaWeb");
map.put("description", "Java를 이용해서 Web Server를 만들기 위한 기술");
map.put("days", 8);
list.add(map);
map = new HashMap<>();
map.put("name", "Spring");
map.put("description", "Java Application을 빠르게 구현할 수 있도록 해주는
프레임워크");
map.put("days", 15);
list.add(map);
```

Map

- HashMap
  - ✓ DataFrame

```
map = new HashMap<>();
map.put("name", "Java Web Application Project");
map.put("description", "그 동안 배운 것들을 이용해서 프로젝트를 진행");
map.put("days", 10);
list.add(map);

map = new HashMap<>();
map.put("name", "Python");
map.put("description", "애플리케이션을 제작하기 위한 프로그래밍 언어");
map.put("days", 10);
list.add(map);

map = new HashMap<>();
map.put("name", "데이터 수집 및 탐색");
map.put("description", "데이터를 수집하고 확인하는 작업");
map.put("days", 5);
list.add(map);
```

Map

- HashMap
  - ✓ DataFrame

```
map = new HashMap<>();
map.put("name", "전처리 및 마이닝");
map.put("description", "분석하기 좋게 데이터를 가공하고 마이닝");
map.put("days", 5);
list.add(map);
map = new HashMap<>();
map.put("name", "AI");
map.put("description", "머신러닝, 딥러닝");
map.put("days", 20);
list.add(map);
map = new HashMap<>();
map.put("name", "프로젝트");
map.put("description", "AI 관련된 서비스 구축");
map.put("days", 20);
list.add(map);
map = new HashMap<>();
map.put("name", "Linux");
map.put("description", "서버 및 클라이언트 용 운영체제");
map.put("days", 0);
list.add(map);
```

Map

- HashMap
  - ✓ DataFrame

```
//전체 데이터 출력하기
for(Map<String, Object> temp : list) {
        //Map에 저장된 데이터를 출력할 때는 get으로 꺼내 그대로 사용
하면 됩니다.
        System.out.println(temp.get("name") + ":" +
                temp.get("description"));
}

System.out.println("=======================");
```

Map

- HashMap
  - ✓ DataFrame

```
//일부분의 데이터 출력하기
//현재 출력 중인 데이터 번호를 저장할 인덱스 변수 생성
int i=0;
//스캐너 생성
Scanner sc = new Scanner(System.in);
//무한 반복문 생성
while(true) {
        //페이지당 2개씩 출력할 것이라서 +2
        int size = i + 2;

        for(; i<size; i=i+1) {
                //데이터의 인덱스가 list의 데이터 개수 와 같아지면 더 이상 읽을 필요 없음
                if(i == list.size()) {
                        break;
                }
                System.out.println(list.get(i));
        }
```

Map

- HashMap
  - ✓ DataFrame

```
                        //무한 반복문 종료
                        if(i == list.size()) {
                                break;
                        }
                        //다음 데이터 입력받기
                        System.out.println("다음 페이지 데이터는 ENTER를 누르면 보여
집니다.");
                        sc.nextLine();
                }
                //무한 반복문 종료
                sc.close();
```

Map

- HashMap
  - ✓ DataFrame

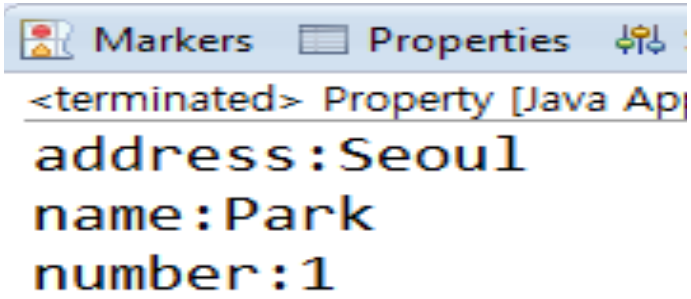
```
                System.out.println("=======================");
                System.out.println("데이터 1개 가져오기");
                System.out.println(list.get(1));
                System.out.println("=======================");
                System.out.println("데이터 정렬하기");
                //map 안에 있는 name의 오름차순 정렬
                list.sort(new Comparator<Map<String, Object>>(){
                        @Override
                        public int compare(
                                Map<String, Object> o1, Map<String, Object> o2) {
                                //이렇게 내부 데이터를 다른 용도로 사용할 때는 형 변
        환해서 사용
                                String front = (String)o1.get("name");
                                String back = (String)o2.get("name");
                                return front.compareTo(back);
                        }
                });
                for(Map<String, Object> temp : list) {
                        System.out.println(temp.get("name"));
                }
        }
}
```

Map

- Properties
  - ✓ 데이터를 쌍으로 저장하지만 Map과 달리 데이터 타입이 <String, String>으로 고정되어 있는 형태의 자료구조
  - ✓ 애플리케이션의 환경설정을 파일에 저장한 후 읽기 위한 목적으로 많이 사용
  - ✓ 메서드
    - String getProperty(String key)
    - String setProperty(String key, String value)
    - void store(OutputStream out, String comment)
    - void storeToXML(OutputStream out, String comment)
    - Enumeration<K> keys()
    - Enumeration<V> elements()

Map

- Properties
  - ✓ 프로퍼티 사용




```
import java.io.*;
import java.util.*;
public class Property {
    public static void main(String[] args) {
            Properties pro = new Properties();
            pro.setProperty("name", "Park");
            pro.setProperty("address", "Seoul");
            pro.setProperty("number", "1");
            Enumeration<Object> keys = pro.keys();
            Enumeration<Object> values = pro.elements();
            while (keys.hasMoreElements())
                    System.out.println(keys.nextElement() + ":" +
    values.nextElement());
            try {
                    pro.store(new FileOutputStream("pro.txt"), "텍스트로 내보내기");
                    pro.storeToXML(new FileOutputStream("pro.xml"), "xml로 내보
    내기");
            } catch (IOException e) {
                    System.out.println("저장 실패");
            }
    }
}
```

Legacy Collection

- Legacy Collection
  - ✓ Vector, Enumeration, Stack, Dictionary, Hashtable, Properties, BitSet 타입은 동기화를 지원하는 구조로 만들어져 있는데 이러한 클래스들은 동기화를 지원하지 않는 클래스들보다 접근 속도가 느린 경우가 대부분이며 최근에는 동기화를 위한 별도의 컬렉션이 제공

Collections

- Collections
  - ✓ Collection 과 관련된 메서드를 제공하는 클래스
  - ✓ java.util.Arrays처럼 fill(), copy(), sort(), binarySearch() 메서드를 제공
  - ✓ 동기화된 컬렉션을 만들어주는 메서드 제공
    - static Collection synchronizedCollection(Collection c)
    - static List synchronizedCollection(List list)
    - static Set synchronizedCollection(Set set)
    - static Map synchronizedCollection(Map map)
    - static SortedSet synchronizedCollection(SortedSet set)
    - static SortedMap synchronizedCollection(SortedMap map)

Collections

- Collections
  - ✓ 변경이 불가능한 컬렉션을 만들어주는 메서드 제공
    - static Collection unmodifiableCollection(Collection c)
    - static List unmodifiableCollection(List c)
    - static Set unmodifiableCollection(Set c)
    - static Map unmodifiableCollection(Map c)
    - static NavigableSet unmodifiableCollection(NavigableSet c)
    - static SortedSet unmodifiableCollection(SortedSet c)
    - static NavigableMap unmodifiableCollection(NavigableMap c)
    - static SortedMap unmodifiableCollection(SortedMap c)

Collections

- Collections
  - ✓ 컬렉션에 하나의 데이터 타입만 저장하는 컬렉션을 만들어주는 메서드 제공
    - static Collection checkedCollection(Collection c, Class type)
    - static List checkedCollection(List c , Class type)
    - static Set checkedCollection(Set c , Class type)
    - static Map checkedCollection(Map c , Class keytype , Class valuetype)
    - static NavigableSet checkedCollection(NavigableSet c , Class type)
    - static SortedSet checkedCollection(SortedSet c , Class type)
    - static NavigableMap checkedCollection(NavigableMap c, Class keytype , Class valuetype)
    - static SortedMap checkedCollection(SortedMap c, Class keytype , Class valuetype)

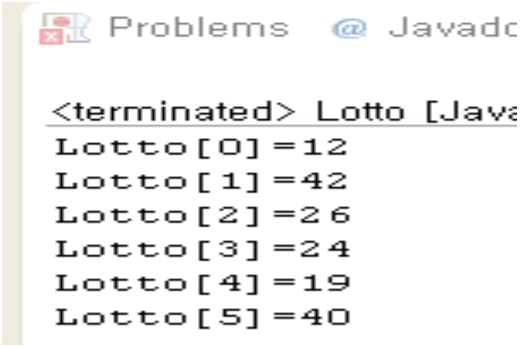
java.util 기타

- Random
  - ✓ 랜덤 한 값을 리턴시켜 주는 클래스
  - ✓ 정수형 난수 발생은 정수 범위 내에서 생성
  - ✓ 부동소수점을 가지는 난수는 0.0에서 1.0사이의 값을 리턴
  - ✓ 생성자
    - Random()
    - Random(long seed): seed를 설정
  - ✓ 메서드
    - float nextFloat(): float 형 난수 생성
    - boolean nextBoolean();
    - int nextInt();
    - long nextLong()
    - double nextDouble()
    - void setSeed()
  - ✓ seed를 고정시키면 동일한 숫자 패턴으로 리턴되고 seed를 고정시키지 않으면 어떤 숫자가 리턴될 지 알기 어려움

java.util 기타

- Random
  - ✓ 로또

```
import java.util.*;
class Lotto {
    public static void main(String[] args) {
        Random rn = new Random();
        int Lotto[] = new int[6];
        boolean flag = false;
        for(int i=0; i<6; i++){
            Lotto[i] = Math.abs(rn.nextInt() % 45) + 1;
            if(i>=1)
            {
                for(int j=0; j<i; j++)
                {
                    if(Lotto[i] == Lotto[j])
                    {
                        flag = true;
                        break;
                    }
                }
            }
```



java.util 기타

- Random
  - ✓ 로또

```
                        if (flag == true)
                        {
                                i--;
                        }
                }
                for(int i=0; i<6; i++)
                {
                        System.out.println("Lotto[" + i + "]=" + Lotto[i]);
                }
        }
}
```

java.util 기타

- StringTokenizer
  - ✓ 문자열을 분할해주는 클래스
  - ✓ 생성자
    - StringTokenizer(String str): 문자열만 매개변수로 입력해서 생성하는 경우로 공백으로 구분
    - StringTokenizer(String str, String delim): str를 delim으로 구분해서 생성
  - ✓ 메서드
    - int countTokens(): 토큰의 개수 리턴
    - boolean hasMoreElements(): 다음 요소의 존재 유무
    - boolean hasMoreTokens(): 다음 토큰의 존재유무
    - Object nextElement(): 다음 요소
    - String nextToken(): 다음 요소
    - String nextToken(String delim): delim에 의해서 구분된 다음 요소

java.util 기타

- StringTokenizer
  - ✓ 문자열을 분할해주는 클래스

```
import java.util.*;
public class TokenTest {
    public static void main(String args[]){
            StringTokenizer st = new StringTokenizer("C++        Java        C#
            JavaScript", "₩t");
            while (st.hasMoreTokens()) {
                    System.out.println(st.nextToken());
            }
    }
}
```

날짜 및 시간

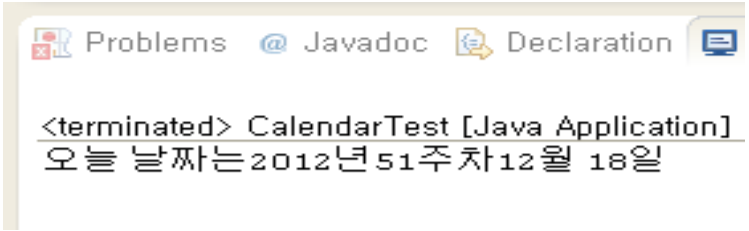
- java.util.Date
  - ✓ 날짜와 시간에 관련된 정보를 저장하고 추출해 주는 클래스
  - ✓ 생성자
    - Date()
    - Date(long timeInMillis)
    - Date(int year, int month, int date)
    - Date(int year, int month, int date, int hrs, int min, int sec)
  - ✓ 정수 1개를 이용해서 생성할 때는 1970년 1월 1일 자정(epoch time)부터 지나온 시간을 1/1000 초 단위로 설정
  - ✓ 년도는 1900년 이후 지나간 년도를 설정해야 하고 월은 1 적게 설정
  - ✓ 대부분의 메서드가 deprecated가 되서 Calendar 클래스를 사용하는 것을 권장
  - ✓ toString 메서드가 재정의 되어있어 객체를 직접 출력하면 날짜 및 시간이 출력
  - ✓ 데이터베이스의 Date 자료형과 직접 연동이 가능

날짜 및 시간

- Calendar
  - ✓ 인스턴스 생성 – 추상 클래스라서 생성자를 이용해서 직접 인스턴스 생성을 못함
    - Calendar.getInstance()를 이용해서 인스턴스를 생성
    - Calendar 클래스를 상속받아서 만들어진 GregorianCalendar 클래스의 인스턴스를 생성해서 사용
  - ✓ Calendar 객체의 데이터를 가져오기: Calendar객체.get(상수);
    - Calendar.YEAR 년
    - (Calendar.MONTH)+1 월은 0부터 시작
    - Calendar.DAY_OF_MONTH 일
    - Calendar.DAY_OF_WEEK 주의 몇 번째 날인가를 추출
    - Calendar.HOUR 시
  - ✓ Calendar 객체에 데이터 설정: Calendar객체.set(상수, 값);
    - set(Calendar.YEAR, 2012); ==> 2012년으로 설정
  - ✓ Calendar 클래스의 객체와 Date 클래스의 객체 사이의 변환
    - Date 변수 = new Date(Calendar객체.getTimeInMillis());
    - Calendar객체. setTime(Date 객체)
  - ✓ Calendar 객체에 데이터 추가: Calendar객체.add(상수, 값);
    - add(Calendar.DAY_OF_MONTH, 3); ==> 3일 후

날짜 및 시간

- Calendar
  - ✓ 인스턴스 생성




```
import java.util.Calendar;

class CalendarTest
{
    public static void main(String[] args)
    {
            Calendar date = Calendar.getInstance();
            int y = date.get(Calendar.YEAR);
            int w = date.get(Calendar.WEEK_OF_YEAR);
            int m = date.get(Calendar.MONTH)+1;
            int d = date.get(Calendar.DATE);
            System.out.println("오늘 날짜는 " + y +"년 " + w + "주차 " + m + "월 "+ d +"
    일");
    }
}
```

날짜 및 시간

```
<terminated> DateGap [Java Application] C:₩Program Files₩Java₩jdk1.
제시카를 만난 날은 1986년 5월 5일 월요일이고
오늘은 2013년 8월 8일 목요일입니다.
우리가 만난지 오늘은 9957일째 입니다.
```

- Calendar
  - ✓ 날짜 와 시간 계산

```
import java.util.*;
//1986년 5월 5일 부터 지나간 날짜와 시간을 구하기
public class DateGap {
    public static void main(String[] args) {
            final String[] DAYOFWEEK = { "", "일", "월", "화", "수", "목", "금", "토" };
            Calendar date1 = new GregorianCalendar();
            Calendar date2 = new GregorianCalendar();
            date1.set(1986, 4, 5);
            System.out.println("제시카를 만난 날은 " + date1.get(Calendar.YEAR) + "년
"
                                + (date1.get(Calendar.MONTH) + 1) + "월 "
                                + date1.get(Calendar.DATE) + "일 "
                               + DAYOFWEEK[date1.get(Calendar.DAY_OF_WEEK)]
    + "요일이고 ");
```

날짜 및 시간

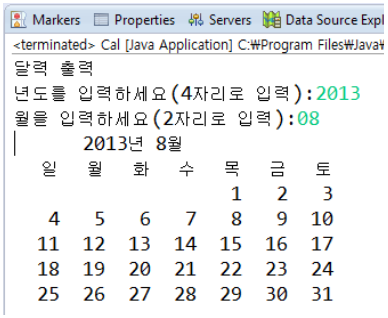
- Calendar
  - ✓ 날짜 와 시간 계산

```
                System.out.println("오늘은 " + date2.get(Calendar.YEAR) + "년 "
                                    + (date2.get(Calendar.MONTH) + 1) + "월 "
                                    + date2.get(Calendar.DATE) + "일 "
                                   + DAYOFWEEK[date2.get(Calendar.DAY_OF_WEEK)]
        + "요일입니다.");
                long gap = (date2.getTimeInMillis() - date1.getTimeInMillis()) / 1000;
                System.out.println("우리가 만난지 오늘은 " + gap / (24 * 60 * 60) + "일째
        입니다.");
        }
    }
```

날짜 및 시간

- Calendar
  - ✓ 날짜 와 시간 계산

```
import java.io.*;
import java.util.*;
public class Cal {
    public static void main(String[] args) {
            final String[] DAYOFWEEK = {"일", "월", "화", "수", "목", "금", "토" };
            String imsi = "";
            int year = 0;
            int month = 0;
            BufferedReader in = new BufferedReader(new
    InputStreamReader(System.in));
            System.out.println("달력 출력");
```



날짜 및 시간

- Calendar
  - ✓ 날짜 와 시간 계산

```
try {
    while (true) {
        System.out.print("년도를 입력하세요(4자리로 입력):");
        imsi = in.readLine();
        if (imsi.compareTo("0001") >= 0 &&
imsi.compareTo("9999") <= 0)
            break;
        else
            System.out.println("올바르게 입력하세
요!!!");
    }
    year = Integer.parseInt(imsi);
```

날짜 및 시간

- Calendar
  - ✓ 날짜 와 시간 계산

```
                                        while (true) {
                                    System.out.print("월을 입력하세요(2자리로 입력):");
                                    imsi = in.readLine();
                                    if (imsi.compareTo("01") >= 0 &&
imsi.compareTo("12") <= 0)
                                        break;
                                    else
                                        System.out.println("올바르게 입력하세
요!!!");
                        }
                        month = Integer.parseInt(imsi);
                } catch (Exception e) {
                        System.out.println("입력 예외 발생");
                }
                int START_DAY_OF_WEEK = 0;
                int END_DAY = 0;
                Calendar startDay = Calendar.getInstance();
                Calendar endDay = Calendar.getInstance();

                startDay.set(year, month - 1, 1);
                endDay.set(year, month, 1);
                endDay.add(Calendar.DATE, -1);
```

날짜 및 시간

- Calendar
  - ✓ 날짜 와 시간 계산

```
                    // 1일의 요일을 구합니다.
                    START_DAY_OF_WEEK = startDay.get(Calendar.DAY_OF_WEEK);
                    // 월의 마지막 날을 찾아옵니다.
                    END_DAY = endDay.get(Calendar.DATE);
                    System.out.println("      " + year + "년 " + month + "월");
                    for(int i=0; i<7; i++)
                              System.out.printf("%4s",DAYOFWEEK[i]);
                    System.out.println();
                    // 해당 월의 1일이 어느 요일인지에 따라서 공백을 출력합니다.
                    for (int i = 1; i < START_DAY_OF_WEEK; i++) {
                              System.out.printf("%4s", " ");
                    }
                    for (int i = 1, n = START_DAY_OF_WEEK; i <= END_DAY; i++, n++) {
                              System.out.printf("%4d", i);
                              if (n % 7 == 0)
                                        System.out.println();
                    }
          }
     }
```

날짜 및 시간

- java.time package
  - ✓ JDK 1.8에서 Date 와 Calendar 클래스의 단점을 보완하기 위해서 추가한 패키지
    - java.time: 날짜와 시간을 다루는 클래스
    - java.time.chrono: 달력 시스템을 위한 클래스
    - java.time.format: 날짜와 시간을 파싱하고 형식화하기 위한 클래스
    - java.time.temporal: 날짜와 시간을 다루기 위한 필드를 제공
    - java.time.zone: 시간대와 관련된 클래스
  - ✓ java.time.LocalDate: 날짜와 관련된 클래스
  - ✓ java.time.LocalTime: 시간과 관련된 클래스
  - ✓ 객체는 now()라는 static 메서드와 of(매개변수)라는 static 메서드를 이용해서 생성
  - ✓ 특정 필드의 값 가져오기
    - get(TemporalField field): field에 해당하는 값 리턴
    - getXXX(): XXX에 해당하는 값 리턴
  - ✓ 데이터 설정하기
    - withXXX(int value): XXX의 값을 values로 설정
    - with(TemporalField field, int value): field의 값을 values로 설정
    - value의 값을 더하거나 빼주는 plus 와 minus 메서드도 있음
  - ✓ 객체의 전후 관계를 비교하는 isAtfer, isBefore, isEqual 메서드가 있음

날짜 및 시간

- java.time package
  - ✓ LocalDate 와 LocalTime

```
import java.time.LocalDate;
import java.time.LocalTime;

public class Local {
    public static void main(String[] args) {
        LocalDate today = LocalDate.now(); // 오늘의 날짜
        LocalTime now = LocalTime.now(); // 현재 시간

        LocalDate meetDay = LocalDate.of(1986, 5, 5); // 1986년 5월 5일
        LocalTime meetTime = LocalTime.of(13, 30, 0); // 13시 30분 0초

        System.out.println("today=" + today);
        System.out.println("now=" + now);
        System.out.println("meetDay=" + meetDay);
        System.out.println("meetTime=" + meetTime);

        System.out.println(meetDay.withYear(2000)); // 2000-5-5
        System.out.println(meetDay.plusDays(1)); // 1986-5-6
    }
}
```

```
today=2016-08-10
now=13:55:59.429
meetDay=1986-05-05
meetTime=13:30
2000-05-05
1986-05-06
```

날짜 및 시간

- java.time package
  - ✓ Instant
    - Epoch Time으로 부터 경과된 시간을 저장하는 클래스
    - now()라는 static 메서드를 이용해서 인스턴스를 생성할 수 있고 Date 객체의 toInstant()를 이용해서 생성 할 수 있음
    - static 메서드인 from(Instant instant) 메서드를 이용해서 Date 객체로 변환 가능
    - UTC(세계 표준 협정시)를 기준으로 하기 때문에 LocalTime과는 차이가 남
    - 이전에는 GMT를 사용했는데 GMT보다는 UTC가 조금 더 정확

날짜 및 시간

- java.time package
  - ✓ LocalDateTime
    - LocalDateTime: LocalDate 와 LocalTime을 합쳐 놓은 클래스
    - ZonedDateTime: LocalDateTime에 시간대(time zone)을 추가한 클래스
    - LocalDateTime은 LocalDate 와 LocalTime을 합쳐서 만들 수 있는 다양한 형태의 of 메서드를 제공
    - toLocalDate() 와 toLocalTime()을 이용해서 LocalDate 와 LocalTime으로 변환 가능한 of 메서드를 제공하고 다양한 형태로 만들 수 있는 메서드도 제공
    - ZoneId의 of 메서드를 이용해서 ZoneId 객체를 만든 후 ZoneDateTime 객체의 atZone 메서드의 매개변수로 넘겨주면 Zone을 설정한 DateTime 객체를 생성할 수 있음
    - ZoneId의 목록은 ZoneId.getAvailZoneIds()를 이용해서 리턴

날짜 및 시간

- java.time package
  - ✓ LocalDateTime

```
2015-12-31T12:34:56+09:00[Asia/Seoul]
2016-08-11T13:52:25.314+09:00[Asia/Seoul]
2016-08-11T00:52:25.357-04:00[America/New_York]
```

```
import java.time.*;
public class DateTime {
    public static void main(String[] args) {
            LocalDate date = LocalDate.of(2015, 12, 31); // 2015년 12월 31일
            LocalTime time = LocalTime.of(12, 34, 56); // 12시 23분 56초
            // 2015년 12월 31일 12시 23분 56초
            LocalDateTime dt = LocalDateTime.of(date, time);
            ZoneId zid = ZoneId.of("Asia/Seoul");
            ZonedDateTime zdt = dt.atZone(zid);
            System.out.println(zdt);
            // String strZid = zdt.getZone().getId();

            ZonedDateTime seoulTime = ZonedDateTime.now();
            ZoneId nyId = ZoneId.of("America/New_York");
            ZonedDateTime nyTime =
    ZonedDateTime.now().withZoneSameInstant(nyId);
            System.out.println(seoulTime);
            System.out.println(nyTime);
    }
}
```

날짜 및 시간

- java.time package
  - ✓ Period
    - 날짜의 차이를 저장하는 클래스
    - Period.between(LocalDate date1, LocalDate date2)의 형식으로 2개의 날짜 차이를 계산해서 저장하는 클래스
  - ✓ Duration
    - 시간의 차이를 저장하는 클래스
    - Duration.between(LocalTime time1, LocalTime time2)의 형식으로 2개의 시간 차이를 계산해서 저장하는 클래스

날짜 및 시간

- java.time package
  - ✓ Period

```
import java.time.*;
import java.time.temporal.ChronoUnit;

public class Different {

    public static void main(String[] args) {
        LocalDate date1 = LocalDate.of(2014, 1, 1);
        LocalDate date2 = LocalDate.of(2015, 12, 31);

        Period pe = Period.between(date1, date2);

        System.out.println("date1=" + date1);
        System.out.println("date2=" + date2);
        System.out.println("pe=" + pe);

        System.out.println("YEAR=" + pe.get(ChronoUnit.YEARS));
        System.out.println("MONTH=" + pe.get(ChronoUnit.MONTHS));
        System.out.println("DAY=" + pe.get(ChronoUnit.DAYS));
```

```
date1=2014-01-01
date2=2015-12-31
pe=P1Y11M30D
YEAR=1
MONTH=11
DAY=30
time1=00:00
time2=12:34:56
du=PT12H34M56S
HOUR=12
MINUTE=34
SECOND=56
NANO=0
```

날짜 및 시간

- java.time package
  - ✓ Period

```
        LocalTime time1 = LocalTime.of(0, 0, 0);
        LocalTime time2 = LocalTime.of(12, 34, 56); // 12시간 23분 56초

        Duration du = Duration.between(time1, time2);

        System.out.println("time1=" + time1);
        System.out.println("time2=" + time2);
        System.out.println("du=" + du);

        LocalTime tmpTime = LocalTime.of(0, 0).plusSeconds(du.getSeconds());

        System.out.println("HOUR=" + tmpTime.getHour());
        System.out.println("MINUTE=" + tmpTime.getMinute());
        System.out.println("SECOND=" + tmpTime.getSecond());
        System.out.println("NANO=" + tmpTime.getNano());
    }
}
```

서식 설정

- SimpleDateFormat
  - ✓ java.text 패키지에 있는 클래스로 날짜 데이터를 원하는 형태의 문자열로 만들어주는 클래스
  - ✓ 사용자가 입력하는 pattern대로 날짜정보를 추출

문자	의미	문자	의미
G	기원	y	년
M	월	w	해에 있어서의 주
W	달에 있어서의 주	D	해에 있어서의 날
d	달에 있어서의 날	F	달에 있어서의 요일
E	요일	a	오전/오후
H	하루에 있어서의 시간 (0 ~ 23)		
k	하루에 있어서의 시간 (1 ~ 24)		
K	오전/오후 (0 ~ 11)		
h	오전/오후 (1 ~ 12)		
m	분		
s	초		
S	밀리 세컨드		

  - ✓ 사용법

```
Date 날짜변수 = new Date();
SimpleDateFormat 서식변수 = new SimpleDateFormat("날짜 서식");
String 문자열변수 = 서식변수.format(날짜변수);
```

  - ✓ parse(문자열) 메서드를 이용하면 포맷의 문자열을 원하는 포맷의 Date 타입으로 변환이 가능

서식 설정

- SimpleDateFormat
  - ✓ 날짜 서식


```
import java.text.*;
import java.util.*;  //Date

public class DateFormat1
{
    public static void main(String args[])
    {
            Date day=new Date();
            SimpleDateFormat sdf=new SimpleDateFormat("yyyy년 MM월 dd일 EEE요
    일 a hh시 mm분 ss초");
            System.out.println(sdf.format(day));
    }
}
```

서식 설정

- SimpleDateFormat
  - ✓ 날짜 서식

```
import java.io.*;
import java.text.*;
import java.util.*;

public class DateFormat2 {
    public static void main(String args[]) {
            String pattern = "yyyy/MM/dd";
            SimpleDateFormat df = new SimpleDateFormat(pattern);
            Date inDate = null;
            BufferedReader in = new BufferedReader(new
    InputStreamReader(System.in));
```

```
날짜를 yyyy/MM/dd의 형태로 입력해주세요.(입력 예:2010/05/11)2011/22222
날짜를 yyyy/MM/dd의 형태로 입력해주세요.(입력 예:2010/05/11)
2013/08/01
입력하신 날짜는 현재와 -199시간 차이가 있습니다.
```

서식 설정

- SimpleDateFormat
  - ✓ 날짜 서식

```
System.out.print("날짜를 " + pattern + "의 형태로 입력해주세요.(입력
    예:1986/05/05):");
            while (true) {
                    try {
                            inDate = df.parse(in.readLine());
                            break;
                    } catch (Exception e) {
                            System.out.print("날짜를 " + pattern
                                    + "의 형태로 입력해주세요.(입
    력 예:1986/05/05):");
                    }
            }
            Calendar cal = Calendar.getInstance();
            cal.setTime(inDate);
            Calendar today = Calendar.getInstance();
            long day = (cal.getTimeInMillis() - today.getTimeInMillis())
                            / (60 * 60 * 1000);
            System.out.println("입력하신 날짜는 현재와 " + day + "시간 차이가 있습니
    다.");
    }
}
```

서식 설정

- DecimalFormat
  - ✓ java.text 패키지에 있는 클래스로 숫자 데이터를 원하는 형태의 문자열로 만들어주는 클래스
  - ✓ 사용자가 입력하는 pattern대로 날짜정보를 추출

문자	의미
0	10진수(값이 없을 때는 0)
#	10진수
.	소수
-	음수 기호
,	천 구분 기호
;	패턴 구분자
%	퍼센트
₩u2030	퍼밀
₩u00A4	통화 기호

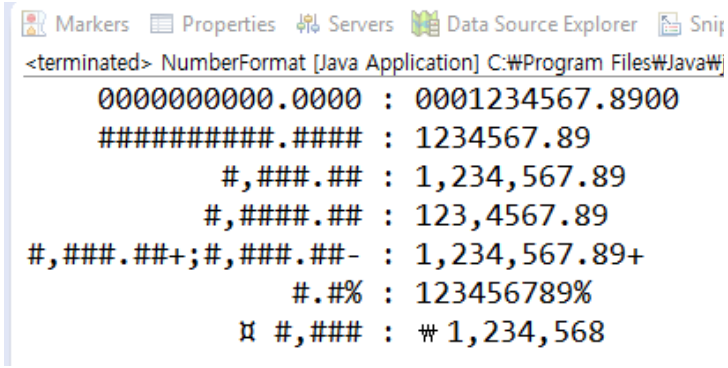
  - ✓ 사용법

```
DecimalFormat 서식변수 = new DecimalFormat (“숫자 서식”);
String 문자열변수 = 서식변수.format(숫자);
```

서식 설정

- DecimalFormat
  - ✓ 숫자 서식

```
import java.text.*;
public class NumberFormat {
    public static void main(String[] args) {
        double number  = 1234567.89;
        String[] pattern = {
                "0000000000.0000",
                "##########.####",
                "#,###.##",
                "#,####.##",
                "#,###.##+;#,###.##-",
                "#.#%",
                "₩u00A4 #,###",
        };
        for(int i=0; i < pattern.length; i++) {
            DecimalFormat df = new DecimalFormat(pattern[i]);
            System.out.printf("%19s : %s₩n",pattern[i], df.format(number));
        }
    }
}
```



서식 설정

- ChoiceFormat
  - ✓ 2개의 배열을 이용하여 범위에 속하는 숫자를 문자열로 치환해주는 클래스
  - ✓ 숫자 배열은 반드시 오름차순으로 정렬되어 있어야 하며 double 배열이어야 함
  - ✓ 사용법

    ChoiceFormat 포맷변수 = new ChoiceFormat(숫자 배열, 문자열 배열);

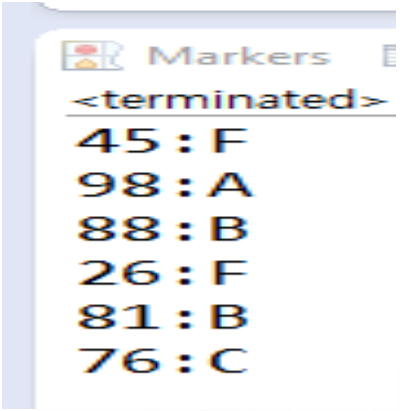
    String 문자열변수 = 포맷변수.format(값);

서식 설정

❖ ChoiceFormat

✓ 2개의 배열을 이용하여 범위에 속하는 숫자를 문자열로 치환해주는 클래스

```
import java.text.*;

public class SelectFormat {
    public static void main(String[] args)
    {
            double[] limits = {0, 60, 70, 80, 90};
            String[] grades = {"F", "D", "C", "B", "A"};

            int[] scores = { 45, 98, 88, 26, 81, 76};

            ChoiceFormat form = new ChoiceFormat(limits, grades);

            for(int i=0;i<scores.length;i++) {
                    System.out.println(scores[i]+":"+form.format(scores[i]));

            }
    }
}
```



서식 설정

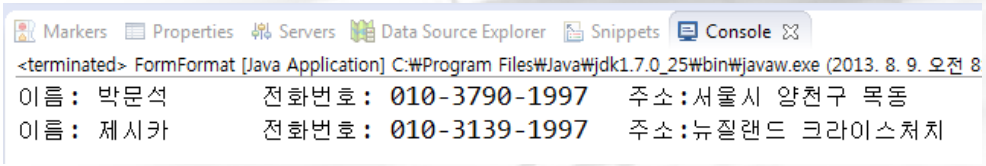
- MessageFormat
  - ✓ 문자열이 삽입될 양식을 미리 만들어 두고 양식에 맞는 문자열로 리턴해주는 클래스
  - ✓ 여러 개의 데이터를 같은 서식으로 출력할 때 사용하는 클래스
  - ✓ 양식을 작성할 때 {숫자}룰 이용해서 작성하고 Message.format(양식,문자열)을 이용해서 생성

서식 설정

❖ MessageFormat

✓ 문자열이 삽입될 양식을 미리 만들어 두고 양식에 맞는 문자열로 리턴해주는 클래스

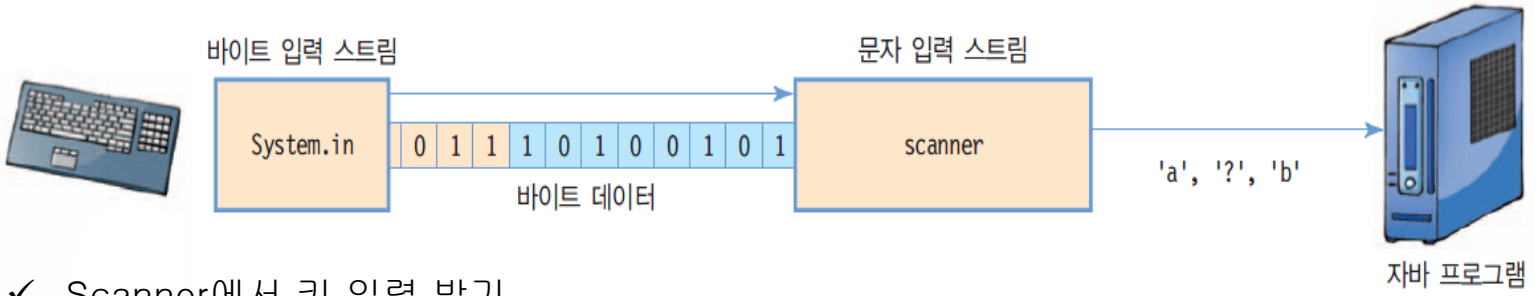
```
import java.text.*;

public class FormFormat {
    public static void main(String[] args)
    {
            String msg = "이름: {0} ₩t전화번호: {1} ₩t주소:{2}";
            Object[] person1 = {
                    "박문석","010-3790-1997", "서울시 양천구 목동"
            };
            Object[] person2 = {
                            "제시카","010-3139-1997", "뉴질랜드 크라이스처치"
                    };

            String result = MessageFormat.format(msg, person1);
            System.out.println(result);
            result = MessageFormat.format(msg, person2);
            System.out.println(result);
    }
}
```



Scanner

- Scanner
  - ✓ 입력을 받기 위한 스트림 – java.util.Scanner
  - ✓ Scanner 객체 생성

```
Scanner a = new Scanner(System.in);
```

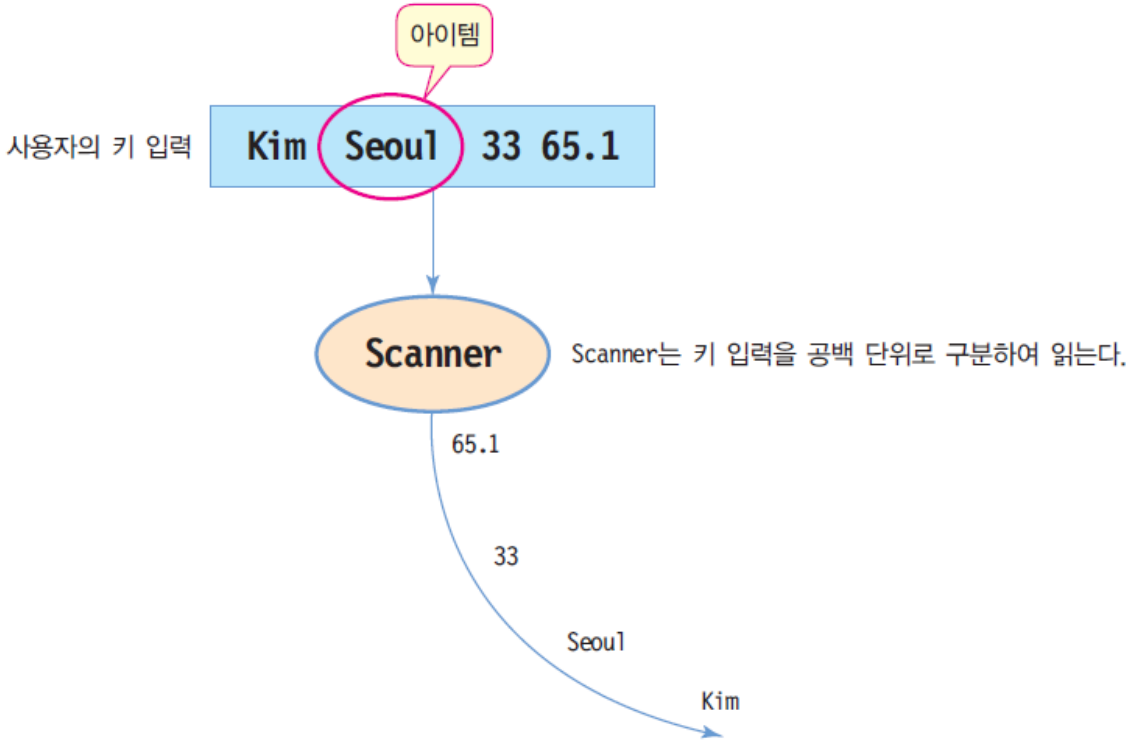


  - ✓ Scanner에서 키 입력 받기
    - Scanner는 입력되는 키 값을 공백으로 구분되는 아이템 단위로 읽음
    - 공백 문자 :‘₩t’,‘ ₩f’,‘ ₩r’,‘ ’,‘ ₩n’

Scanner

❖ Scanner

```
Scanner scanner = new Scanner(System.in);
String name = scanner.next();                    //문자열
String addr = scanner.next();                    // 문자열
int age = scanner.nextInt();                     // 정수
double weight = scanner.nextDouble();            // 실수
```



Scanner

- Scanner
  - ✓ 생성자

Scanner(File Source)	파일로부터 입력
Scanner(InputStream Source)	네트워크나 키보드(System.in)로부터 입력
Scanner(Readable Source)	다음 아이템을 찾아 byte로 변환하여 반환
Scanner(String Source)	문자열로부터 입력

Scanner

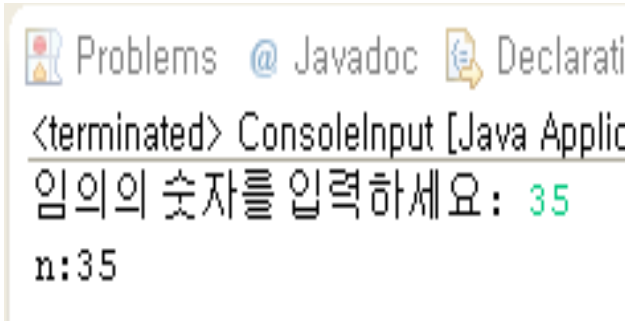
❖ Scanner

메서드	설명
String next()	다음 아이템을 찾아 문자열로 반환
boolean nextBoolean()	다음 아이템을 찾아 boolean으로 변환하여 반환
byte nextByte()	다음 아이템을 찾아 byte로 변환하여 반환
double nextDouble()	다음 아이템을 찾아 double로 변환하여 반환
float nextFloat()	다음 아이템을 찾아 float로 변환하여 반환
int nextInt()	다음 아이템을 찾아 int로 변환하여 반환
long nextLong()	다음 아이템을 찾아 long으로 변환하여 반환
short nextShort()	다음 아이템을 찾아 short로 변환하여 반환
String nextLine()	한 라인 전체('₩n' 포함)를 문자열 타입으로 반환
boolean hasNextLine()	입력된 라인이 있는지 여부를 리턴

Scanner

- Scanner
  - ✓ ConsoleInput

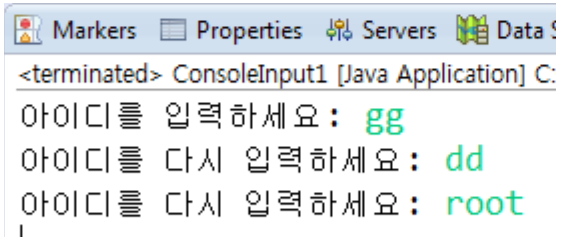
```
import java.util.*;
public class ConsoleInput
{
    public static void main(String[] args) {
            System.out.print("임의의 숫자를 입력하세요: ");
            //키보드로부터 입력받는 Scanner 객체를 생성
            Scanner scanner = new Scanner(System.in);
            //Enter를 누를 때까지의 내용을 정수로 변환해서 n에 대입
            //실수면 nextFloat 이나 nextDouble 문자열이면 next 로 변환
            int n = scanner.nextInt();
            System.out.println("n:" + n);
            scanner.close();
    }
}
```



Scanner

- Scanner
  - ✓ ConsoleInput

```
import java.util.*;

public class ConsoleInput1 {
    public static void main(String[] args) {
            // 키보드로부터 입력받는 Scanner 객체를 생성
            Scanner scanner = new Scanner(System.in);
            String id = "";
            System.out.print("아이디를 입력하세요: ");
            while (scanner.hasNextLine()) {
                    id = scanner.nextLine();
                    if (id.equals("root"))
                            break;
                    System.out.print("아이디를 다시 입력하세요: ");
            }
            System.out.print("로그인에 성공하셨습니다. ");
            scanner.close();

    }
}
```



Scanner

❖ Scanner

✓ Scanner를 이용하여 나이, 체중, 신장 데이터를 키보드에서 입력 받아 출력하는 프로그램

```
import java.util.Scanner;

public class ScannerExam {
   public static void main (String args[]) {
      Scanner a = new Scanner(System.in);
      System.out.println("나이, 체중, 신장을 빈칸으로 분리하여 순서대로 입력하세요");
      System.out.println("당신의 나이는 " + a.nextInt() + "세 입니다.");
      System.out.println("당신의 체중은 " + a.nextDouble() +"kg 입니다.");
      System.out.println("당신의 신장은 " + a.nextDouble()+ "cm 입니다.");
      a.close();
   }
}
```

```
나이, 체중, 신장을 빈칸으로 분리하여 순서대로 입력하세요
35 75 175
당신의 나이는 35살입니다.
당신의 체중은 75.0kg입니다.
당신의 신장은 175.0cm입니다.
```

정규 표현식

- 정규 표현식
  - ✓ 정규 표현식(regular expression, 간단히 regexp 또는 regex) 또는 정규식은 특정한 규칙을 가진 문자열의 집합을 표현하는 데 사용하는 형식 언어
  - ✓ 정규 표현식은 많은 텍스트 편집기 와 프로그래밍 언어에서 문자열의 검색과 치환을 위해 지원하고 있으며 Perl과 Tcl은 언어 자체에 강력한 정규 표현식을 구현
  - ✓ 컴퓨터 과학의 정규 언어로부터 유래하였으나 구현체에 따라서 정규 언어보다 더 넓은 언어를 표현할 수 있는 경우도 있으며 심지어 정규 표현식 자체의 문법도 여러 가지 존재
  - ✓ 수많은 프로그래밍 언어가 정규 표현식 기능을 제공하고 있으며 일부는 Perl, JavaScript, Ruby, Tcl처럼 기본 내장되어 있는 반면 .Net 언어, Java, Python, POSIX C, C++ (C++11 이후)에서는 표준 라이브러리를 이용하여 구현
  - ✓ 그 밖의 언어들은 외부 라이브러리를 통해 정규식을 제공
  - ✓ Java에서는 java.util.regex 패키지에 있는Match 클래스와 Pattern 클래스를 이용하여 문자열을 정규 표현식으로 나타낼 수 있음

정규 표현식

❖ 정규 표현식

✓ 사용 방식

```
// (1)정규 표현의 패턴을 생성
Pattern pattern = Pattern.compile("This is a .*₩₩.");
// (2)정규 표현의 패턴에 적합한지를 체크하는 문자열 String sentence = "This is a pen.";
// (3)정규 표현 처리를 실시하기 위한 클래스를 취득 Matcher matcher =
pattern.matcher(sentence);
// (4)정규 표현의 패턴에 적합한지를 체크 if (matcher.matches()) {
System.out.println("적합하다"); } else {
System.out.println("적합하지 않다"); }
```

1) java.util.regex.Pattern 클래스의 compile 메서드를 사용하여 인수에 문자열로 지정한 정규 표현 으로부터 패턴 객체를 생성

2) 정규 표현의 패턴에 적합한지를 체크하는 문자열 이번에 체크 대상이 되는 문자열

3) 1)에서 생성한 패턴 객체의 matcher 메서드를 정규 표현 처리의 대상 문자열을 인수로 하여 실행하면 정규 표현의 처리를 위한 클래스인 java.util.regex.Matcher 클래스의 객체를 리턴

4) 3)에서 리턴받은 Matcher 클래스의 객체를 사용하여 정규 표현의 처리를 실시하는데 정규 표현의 패턴에 적합한지를 체크하기 위해 Matcher 클래스의 matches 메서드를 사용했으며 matches 메서드는 문자열이 정규 표현에 일치하면 true를 일치하지 않으면 false를 리턴함

정규 표현식

- 와일드 카드 문자
  - ✓ '.' 특수문자
    - ❑ 임의의 한 문자를 의미
    - ❑ '.' 특수문자가 위치한 곳에는 반드시 한 글자가 위치
    - ❑ 패턴 : a.b → 일치하는 문자열 : acb, adb, azb 등
    - ❑ 패턴 : ab. → 일치하지 않는 경우 : ab, abcd 등
  - ✓ '*' 특수문자
    - ❑ 바로 앞의 문자를 0번 이상 반복
    - ❑ * 앞에는 한 글자 이상의 단어가 반드시 위치
    - ❑ 패턴 : ab*c → abc, abbc, abbbc 등
    - ❑ 패턴 : *ab → 불가능
  - ✓ '+' 특수문자
    - ❑ 바로 앞의 문자를 1번 이상 반복
    - ❑ + 앞에는 한 글자 이상의 단어가 반드시 위치
    - ❑ 패턴 : a+b → aab, aaab, aaaab 등
    - ❑ 패턴 : +ab → 불가능

정규 표현식

- 와일드 카드 문자
  - ✓ '?' 특수문자
    - ❑ 특수문자 바로 앞의 문자가 하나 있거나 없는 것
    - ❑ 패턴 : try? → 일치하는 문자열 : tr, try
    - ❑ 패턴 : a?c → 일치하는 문자열 : c, ac
  - ✓ '^' 특수문자
    - ❑ 문장의 처음
    - ❑ 패턴 : ^Hello → 일치하는 문자열 : Hello abc, Hello world 등의 Hello로 시작하는 문자열
  - ✓ '$' 특수문자
    - ❑ 문장의 끝
    - ❑ 패턴 : world$ → 일치하는 문자열 : hello world, welcome my world 등의 world로 끝나는 문자열

정규 표현식

- 와일드 카드 문자
  - ✓ '[ ]' 특수문자
    - 괄호 안의 문자 중 일치하는 것을 찾고자 할 경우 사용
    - 패턴 : [abc] → 일치하는 문자열 : a, b, c, ab, abc 등
    - [a-z] → 소문자가 포함된 모든 문자열
    - [A-Z] → 대문자가 포함된 모든 문자열
    - [0-9] → 숫자가 포함된 모든 문자열
    - [가-힣] → 한글
    - ^[a-zA-Z0-9] → 숫자나 영문자로 시작되는 모든 문자열
  - ✓ '[ ]' 특수문자 안에서의 '^' 특수문자
    - []안의 문자와 일치하지 않는 문자열을 포함하고 있는 패턴을 의미
    - ex) 패턴 : [^abc]de → .de패턴과 동일 (ade, bde, cde 제외)

정규 표현식

- 와일드 카드 문자
  - ✓ '{ }' 특수문자
    - ❑ 특수문자 앞의 문자나 문자열의 반복되는 개수
    - ❑ go{5}ggle → goooooogle
    - ❑ go{3,}ggle → gooogle, goooogle, gooooogle 등 (o가 3개 이상)
    - ❑ go{2,4}ggle → google, gooogle, goooogle (o가 2개 이상 4개 이하)
  - ✓ '( )' 특수문자
    - ❑ ( )안의 글자들을 하나의 문자로 간주
    - ❑ (hero_){3} → hero_hero_hero_
    - ❑ (hero_)* → null, hero_, hero_hero_ 등
  - ✓ '|' 특수문자
    - ❑ 패턴 안에서 OR연산 사용시 사용
    - ❑ man|woman → man, woman, manwoman, superman 등

정규 표현식

- 와일드 카드 문자
  - ✓ ₩w : 알파벳이나 숫자
  - ✓ ₩W : ₩w의 not. 즉 알파벳이나 숫자를 제외한 문자
  - ✓ ₩d : [0-9]와 동일
  - ✓ ₩D : 숫자를 제외한 모든 문자

정규 표현식

- Pattern
  - ✓ 정규식 객체 생성

    Pattern pattern = Pattern.compile(“정규 표현식”);
  - ✓ 문자열에 정규식이 포함되어 있는지 확인

    Matcher matcher = pattern.matcher(조회할 문자열);
  - ✓ String 클래스에서의 정규식 비교

    boolean matches(“정규 표현식”)

정규 표현식

- Matcher
  - ✓ find() : 패턴이 일치하는 경우 true를 반환하고, 그 위치로 이동(여러 개가 매칭되는 경우 반복 실행가능)
  - ✓ find(int start) : start위치 이후부터 매칭 검색을 수행
  - ✓ start() : 매칭되는 문자열 시작위치 반환
  - ✓ start(int group) : 지정된 그룹이 매칭되는 시작위치 반환
  - ✓ end() : 매칭되는 문자열 끝 다음 문자위치 반환
  - ✓ end(int group) : 지정되 그룹이 매칭되는 끝 다음 문자위치 반환
  - ✓ group() : 매칭된 부분을 반환
  - ✓ group(int group) : 매칭된 부분중 group번 그룹핑 매칭부분 반환
  - ✓ groupCount() : 패턴내 그룹핑 한(괄호 지정) 전체 갯수 반환
  - ✓ matches() : 패턴이 전체 문자열과 일치할 경우 true 반환

정규 표현식

❖ String 클래스의 메서드를 이용한 정규식 패턴 검사

```
public class RegCheck {
        public static void main(String[] args) {
                String regExpStr = "^a.c$";
                String[] strArr = { "abc", "acc", "adc", "aec", "afc", "agc", "ahc", "aiic" };
                for (String str : strArr) {
                        System.out.println("[" + str + "] : " + str.matches(regExpStr));
                }
        }
}
```

```
[aec] : true
[afc] : true
[agc] : true
[ahc] : true
[aiic] : false
```

정규 표현식

❖ Pattern 클래스와 Matcher 클래스의 메서드를 이용한 정규식 패턴 검사

```
import java.util.regex.Pattern;

public class PatternCheck {
        public static void main(String[] args) {
                Pattern p = Pattern.compile("^a.c$");
                String[] strArr = { "abc", "acc", "adc", "aec", "afc", "agc", "ahc", "aiic" };
                for (String str : strArr) {
                        System.out.println("[" + str + "] : " + p.matcher(str).find());
                }
        }
}
```

```
[aec] : true
[afc] : true
[agc] : true
[ahc] : true
[aiic] : false
```

정규 표현식

❖ 정규식을 이용한 문자열 사용

```
This
is
a
pen.
This is a pen.
```

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class RegString {

	public static void main(String[] args) {
		// (1)정규 표현의 패턴을 생성('₩₩s'는 공백을 나타내는 정규 표현)
		Pattern pattern = Pattern.compile("₩₩s+");
		// (2)정규 표현의 패턴에 적합한지를 체크하는 문자열
		String sentence = "This is a pen.";
		// (3)정규 표현을 사용하여 문자열을 분할
		String[] words = pattern.split(sentence);

		for (String word : words) {
			System.out.println(word);
		}

		// (4)정규 표현 처리를 실시하기 위한 클래스를 취득
		Matcher matcher = pattern.matcher(sentence);
		// (5)정규 표현과 일치한 문자열을 치환
		System.out.println(matcher.replaceAll(" "));
	}
}
```

정규 표현식

- 자주 사용하는 패턴
  - ✓ E-Mail : "^[a-zA-Z0-9]+[@][a-zA-Z0-9]+[₩₩.][a-zA-Z0-9]+$
  - ✓ 휴대폰 : ^01(0|1|[6-9])-(₩₩d{3}|₩₩d{4})-₩₩d{4}$
  - ✓ 일반전화 : ^₩d{2.3} - ₩d{3,4} - ₩d{4}$
  - ✓ 주민등록번호 : ₩d{6} ₩- [1-4]₩d{6}
  - ✓ IP 주소 : ([0-9]{1,3}) ₩. ([0-9]{1,3}) ₩. ([0-9]{1,3}) ₩. ([0-9]{1,3})
  - ✓ 비밀번호 : ^(?=.*₩₩d)(?=.*[~`!@#$%₩₩^&*()-])(?=.*[a-z])(?=.*[A-Z]).{9,12}$

정규 표현식

❖ 자주 사용하는 패턴

```
import java.util.regex.Pattern;

public class FrequentlyPattern {
        public static void main(String[] args) {
                Pattern p = Pattern.compile("^01(0|1|[6-9])-(₩₩d{3}|₩₩d{4})-
₩₩d{4}$");
                String mobile1 = "010-3790-1997";
                String mobile2 = "012-3790-1997";
                boolean r = p.matcher(mobile1).find();
                if(r){
                        System.out.println(mobile1 + "은 올바른 전화번호입니다.");
                }
                else{
                        System.out.println(mobile1 + "은 올바르지 않은 전화번호입니
다.");
                }
```

```
010-3790-1997은 올바른 전화번호입니다.
012-3790-1997은 올바르지 않은 전화번호입니다.
ggangpae1@gmail.com은 올바른 이메일 주소입니다.
ggangpae1gmail.com은 올바르지 않은 이메일 주소입니다.
ggangpae1@은 올바르지 않은 이메일 주소입니다.
```

정규 표현식

❖ 자주 사용하는 패턴

```
r =p.matcher(mobile2).find();
if(r){
        System.out.println(mobile2 + "은 올바른 전화번호입니다.");
}
else{
        System.out.println(mobile2 + "은 올바르지 않은 전화번호입니
다.");
}

p = Pattern.compile("^[a-zA-Z0-9]+[@][a-zA-Z0-9]+[₩₩.][a-zA-Z0-
9]+$");
String email1 = "ggangpae1@gmail.com";
String email2 = "ggangpae1gmail.com";
String email3 = "ggangpae1@";
r =p.matcher(email1).find();
if(r){
        System.out.println(email1 + "은 올바른 이메일 주소입니다.");
}
else{
        System.out.println(email1 + "은 올바르지 않은 이메일 주소입니
다.");
}
```

정규 표현식

❖ 자주 사용하는 패턴

```
                        r =p.matcher(email2).find();
                        if(r){
                                System.out.println(email2 + "은 올바른 이메일 주소입니다.");
                        }
                        else{
                                System.out.println(email2 + "은 올바르지 않은 이메일 주소입니
다.");
                        }

                        r =p.matcher(email3).find();
                        if(r){
                                System.out.println(email3 + "은 올바른 이메일 주소입니다.");
                        }
                        else{
                                System.out.println(email3 + "은 올바르지 않은 이메일 주소입니
다.");
                        }
                }
}
```

정규 표현식

❖ 문자열 객체에 정규식 사용

```
public class StringReg {

        public static void main(String[] args) {
                String sentence = "This is a pen.";
                // (1)정규 표현과 일치하는지 체크
                System.out.println("(1)");
                System.out.println(sentence.matches("Th.* is a .*₩₩."));
                // (2)정규 표현을 사용한 분할
                System.out.println("(2)");
                String[] words = sentence.split("₩₩s+");
                for (String word : words) {
                        System.out.println(word); }
                // (3)정규 표현을 사용한 치환
                System.out.println("(3)");
                System.out.println(sentence.replaceAll("₩₩s+", " "));

        }

}
```

```
(1)
true
(2)
This
is
a
pen.
(3)
This is a pen.
```

연습문제

- Array List 와 Linked List 의 서로 간의 장단점 확인
- Stack 과 Queue의 용도를 확인
- 번호(정수), 이름(문자열), 전화번호(문자열), 주소(문자열)을 저장할 수 있는 클래스를 만들고 이 클래스의 인스턴스를 3개 만들어서 ArrayList에 저장하고 테이블 형태로 출력
- 위의 내용을 클래스 대신에 Map을 이용해서 데이터를 저장하고 Map의 LinkedList를 만들고 테이블 형태로 출력
- 랜덤한 1-45 사이의 서로 다른 정수를 6개를 생성해서 ArrayList에 대입하고 Comparable 인터페이스를 이용해서 내림차순 정렬해서 출력