

java.util 패키지

1. Arrays 클래스

❖ 배열을 조작하기 위한 메소드를 제공하는 클래스

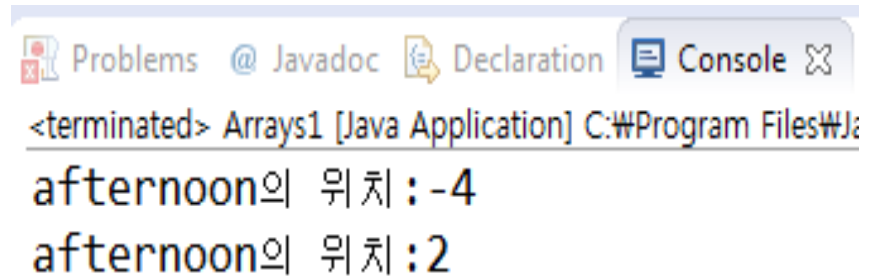
❖ static 메소드만 소유하고 있는 클래스

- ✓ static int binarySearch (Object [] a, Object key): 이분 검색을 이용해서 key를 찾아주는 메소드로 데이터가 정렬이 되어 있어야만 올바른 결과를 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 채워줍니다.
- ✓ static void fill (Object [] a, Object val) : 배열의 모든 요소를 val로 채워줍니다.
- ✓ void sort (Object [] a): 정렬을 수행해 주는 메소드
 - 기본 자료형 배열인 경우는 바로 해주지만 기본 자료형이 아닌 참조형인 경우는 클래스가 Comparable 인터페이스를 implements해서 int compare(T o1) 함수가 재정의 되어 있어야 가능합니다.
 - compare 메소드는 호출하는 객체와 매개변수로 대입된 객체 사이의 요소를 비교해서 1,0,-1 셋 중의 하나의 값을 반환해야 합니다.
 - 만일 Comparable 인터페이스를 implements 하지 않은 경우에는 Comparator 인터페이스를 Implements 한 객체를 대입해도 됩니다.
- ✓ List asList(Object [] o): 배열을 List 인터페이스를 구현한 객체로 변환해서 리턴

실습(Arrays-Arrays1.java)

```
package Arrays;  
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 메소드를 구현해야 합니다.  
        Arrays.sort(ar);  
        System.out.println("morning의 위치:" + Arrays.binarySearch(ar, "morning"));  
    }  
}
```



```
<terminated> Arrays1 [Java Application] C:\Program Files\J...  
afternoon의 위치 : -4  
afternoon의 위치 : 2
```

실습(Arrays-Data.java)

```
package Arrays;
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;
    }
}
```

```
<terminated> Arrays2 [Java Application] C:\Program Files\Java\
Data [name=park, jumsu=2]
Data [name=noname, jumsu=0]
Data [name=kim, jumsu=3]
정렬 후
Data [name=noname, jumsu=0]
Data [name=park, jumsu=2]
Data [name=kim, jumsu=3]
```

실습(Arrays-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;  
}
```



실습(Arrays-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 + "];"  
}  
}
```

실습(Arrays-Arrays2.java)

```
package Arrays;
import java.util.*;
public class Arrays2 {
    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);
    }
}
```

실습(Arrays-Arrays3.java)

```
package Arrays;
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);
        //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());
                }
            };
    };
}
```


실습(Arrays-Arrays3.java)

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



2.Collection

- ❖여러 개의 요소를 묶어서 하나의 객체로 만든 것
- ❖java.util 패키지에 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
 - ✓ 자료구조나 알고리즘을 개발자가 구현할 필요가 없어 편리
 - ✓ 일반 배열을 사용하지 않고 프레임워크가 제공하는 컬렉션을 적절히 선택해서 사용하면 빠른 속도로 작업 처리 가능
 - ✓ 통합된 API 구조로 효율성 증대

2.Collection

❖메소드

- ✓ 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 객체를 생성할 때 사용한 Generics 타입을 리턴

3. Enumeration & Iterator

❖ 컬렉션에 저장되어 있는 객체들을 저장방법에 상관없이 접근 할 수 있도록 한 인터페이스로 컬렉션 객체들의 `enumerable()`이나 `iterator()` 메소드에 의해 구현됩니다.

❖ Enumeration 인터페이스

- ✓ `boolean hasMoreElements()`: 다음 요소가 있는지 없는지 여부를 판단해주는 메소드
- ✓ `E nextElement ()`: 열거에 1개 이상의 요소가 남아 있는 경우는 다음의 요소를 리턴

❖ Iterator 인터페이스

- ✓ `boolean hasNext()`: 다음 요소가 있는지 여부를 리턴
- ✓ `E next()`: 반복 처리로 다음의 요소를 리턴
- ✓ `void remove()`: 마지막으로 요소를 삭제

4. List 인터페이스

- ❖ 데이터를 순서대로 저장하는 컬렉션을 구현하는데 사용되는 인터페이스
- ❖ null을 허용하며 제너릭 사용
- ❖ 구현된 인터페이스: Collection<E>, Iterable<E>
- ❖ 메소드
 - ✓ boolean add(E e)
 - ✓ void add(int index, Object element)
 - ✓ boolean addAll(int index, Collection c)
 - ✓ Object get(index)
 - ✓ int indexOf(Object o)
 - ✓ int lastIndexOf(Object o)
 - ✓ Object remove(int index)
 - ✓ void sort(Comparator c): Comparator의 메소드를 이용해서 정렬
 - ✓ List subList(int fromIndex, int toIndex)
- ❖ Object 타입을 리턴하는 메소드는 List를 생성할 때 제너릭을 사용하면 리턴을 해주는 메소드의 리턴 타입도 제너릭을 적용한 타입으로 리턴됩니다.
- ❖ Vector, ArrayList, LinkedList, Stack 등의 클래스가 List 인터페이스를 implements한 클래스

4-1. java.util.Vector

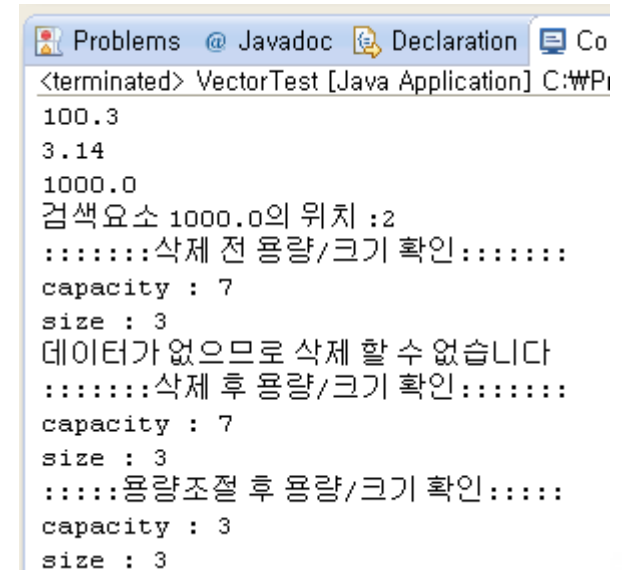
```
public class Vector<E>  
    extends AbstractList<E>  
    implements List<E>, RandomAccess, Cloneable, Serializable
```

- ❖ 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 있습니다. – 동시에 사용하려고 하면 순서대로 접근을 하도록 합니다.

- ❖ 벡터의 생성자
 - ✓ Vector()
 - ✓ Vector(Collection<? extends E> c)
 - ✓ Vector(int initialCapacity)
 - ✓ Vector(int initialCapacity, int capacityIncrement)

실습(ListCollection-VectorTest.java)

```
package ListCollection;
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.");
    }
}
```



```
Problems @ Javadoc Declaration Co
<terminated> VectorTest [Java Application] C:\WP1
100.3
3.14
1000.0
검색요소 1000.0의 위치 :2
:::삭제 전 용량/크기 확인:::
capacity : 7
size : 3
데이터가 없으므로 삭제 할 수 없습니다
:::삭제 후 용량/크기 확인:::
capacity : 7
size : 3
:::용량조절 후 용량/크기 확인:::
capacity : 3
size : 3
```

실습(ListCollection-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 ());
```

```
}
```

```
}
```


4-2. ArrayList

❖ 구현된 인터페이스: Serializable, Cloneable, Iterable <E>, Collection <E>, List <E>, RandomAccess

❖ 생성자

- ✓ ArrayList()
- ✓ ArrayList(Collection c)
- ✓ ArrayList(int initialCapacity)

❖ 메소드는 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 느리다.

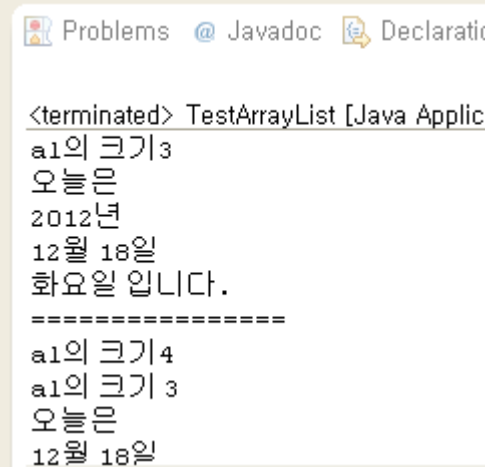
❖ 벡터와 다른 점은 동기화를 지원하지 않으므로 속도는 빠르지만 멀티 스레드 환경에서 사용할 때는 직접 동기화를 구현하거나 동기화된 List 클래스를 사용해야 합니다..

❖ 동기화가 필요하다면 Collections 클래스의 동기화 메소드를 이용해서 생성하면 됩니다.

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

실습(TestArrayList .java)

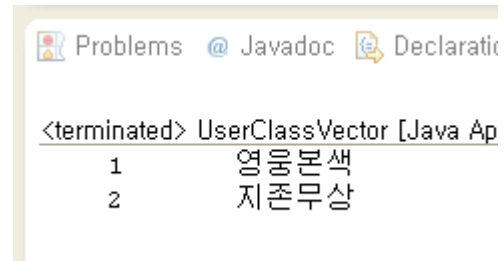
```
package ListCollection;
import java.util.ArrayList;
public class TestArrayList {
    public static void main(String args[]) {
        ArrayList<String> al = new ArrayList<String>();
        al.add("오늘은");
        al.add("2012년");
        al.add("12월 18일");
        System.out.println("al의 크기" + al.size()); // 결과 : al의 크기3
        al.add("화요일 입니다.");
        for (int a = 0; a < al.size(); a++) {
            System.out.println(al.get(a));
        }
        System.out.println("=====");
        System.out.println("al의 크기" + al.size()); // 결과 : al의 크기4
        al.remove(1);
        System.out.println("al의 크기 " + al.size()); // 결과 : al의 크기 3
        for (int a = 0; a < al.size(); a++) {
            System.out.println(al.get(a));
        }
    }
}
```



```
<terminated> TestArrayList [Java Applic
al의 크기3
오늘은
2012년
12월 18일
화요일 입니다.
=====
al의 크기4
al의 크기 3
오늘은
12월 18일
```

실습(UserClassArrayList.java)

```
import java.util.Vector;
class Tape
{
    static int number;
    static { number = 0;}
    public int tapenumber;
    public String title;
    public Tape(String str)
    {
        tapenumber = ++number;
        title = str;
    }
    public String toString(){
        return String.format("%5d %10s",tapenumber,title);
    }
}
```



실습(UserClassArrayList.java)

```
public class UserClassArrayList {  
    public static void main(String[] args) {  
        ArrayList<Tape> v = new ArrayList<Tape>();  
        Tape Obj = new Tape("영웅본색");  
        v.add(Obj);  
        Obj = new Tape("지존무상");  
        v.add(Obj);  
        System.out.println(v.get(0));  
        System.out.println(v.get(1));  
    }  
}
```



4-3. LinkedList

- ❖ 구현된 인터페이스: Serializable, Cloneable, Iterable <E>, Collection <E>, List <E>, Queue <E>
- ❖ 연결된 노드들을 기반으로 구현된 리스트
- ❖ 링크가 반드시 다른 노드에 연결되어 있어야 합니다.
- ❖ LinkedList는 시작과 끝에 있는 요소를 가져올 수 있고 중간에 삽입이나 삭제를 할 수 있습니다.
- ❖ 중간에 삽입과 삭제하는 속도는 빠르지만 검색에는 취약한 자료구조이며 순차적으로 삽입과 삭제를 하면 ArrayList가 빠릅니다.
- ❖ 동기화를 지원하지 않습니다.
- ❖ 메소드
 - ✓ 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

실습(ListCollection-LinkedListTest .java)

```
package ListCollection;
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());
    }
}
```

Problems @ Javadoc Declaration

<terminated> LinkedListTest [Java Applicati

q의 크기 : 3

강남삭제 !

이대삭제 !

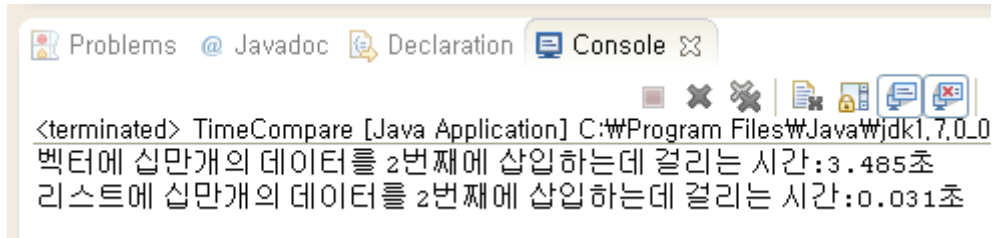
종로삭제 !

q의 크기 : 0

실습(ListCollection-TimeCompare.java)

```
package ListCollection;

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();
    }
}
```



실습(ListCollection-TimeCompare.java)

```
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");  
}  
}
```


4-4. java.util.Stack

java.lang.Object

java.util.AbstractCollection <E>

java.util.AbstractList <E>

java.util.Vector <E>

java.util.Stack <E>

❖인터페이스: Serializable, Cloneable, Iterable <E>, Collection <E>, List <E>, RandomAccess

❖LIFO구조로 삽입과 삭제가 한 쪽 끝에서 발생하는 구조

❖생성자

✓ Stack<(): 비어있는 스택 생성

❖메소드

✓ 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

실습(ListCollection-TestStack.java)

```
package ListCollection;
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"));
    }
}
```

Problems @ Javadoc Declaration Cor

```
<terminated> TestStack [Java Application] C:\WPro
Top이 가리키는 데이터 : 2
Top이 가리키는 데이터 : 1
Top이 가리키는 데이터 : 2
Top이 가리키는 데이터 : 2
검색하고자 하는 데이터가 있는 위치 : 2
검색하고자 하는 데이터가 있는 위치 : -1
```

4-5. java.util.Queue

public interface Queue<E>
extends Collection<E>

- ❖ FIFO 구조로 만들어진 자료구조로 자바에서는 인터페이스 형태로 제공됩니다.
- ❖ Queue 객체는 직접 생성할 수 없고 이 인터페이스를 implements 한 클래스의 객체로 생성합니다.
- ❖ 메소드
 - ✓ boolean add(E e)
 - ✓ E element()
 - ✓ boolean offer(E e)
 - ✓ E peek()
 - ✓ E poll()
 - ✓ E remove()

4-5. java.util.Queue

- ❖ Queue를 implements 한 PriorityQueue는 데이터를 오름차순으로 정렬하면서 삽입합니다.
- ❖ 이 클래스에 삽입하는 객체는 반드시 Comparable 인터페이스를 implements 해야 합니다.
- ❖ 상속 계층
 - java.lang.Object
 - java.util.AbstractCollection<E>
 - java.util.AbstractQueue<E>
 - java.util.PriorityQueue<E>
- ❖ 구현된 인터페이스
 - Serializable, Iterable<E>, Collection<E>, Queue<E>

실습(ListCollection-TestQueue.java)

```
package ListCollection;

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());
    }
}
```

첫번째 요소:Morning

큐의 크기:2

첫번째 요소:Morning

큐의 크기:1

실습(ListCollection-PriorityQueueTest.java)

```
package ListCollection;                                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();
    }
}
```

4-5. java.util.Queue

- ❖ Deque는 양쪽에서 삽입과 삭제가 가능한 자료구조를 표현한 인터페이스
- ❖ ArrayDeque는 Deque 인터페이스를 implements 한 대표적인 클래스
 - ✓ 상속 계층
 - java.lang.Object
 - java.util.AbstractCollection<E>
 - java.util.ArrayDeque<E>
 - ✓ 구현된 인터페이스
 - Serializable, Cloneable, Iterable<E>, Collection<E>, Deque<E>, Queue<E>

4-6. List의 정렬

- ❖ `void sort(Comparator <?> comp)`: 정렬할 수 있는 인스턴스 메소드 제공
 - ✓ `Comparator`를 구현해서 대입하면 리스트를 정렬해 줍니다.
- ❖ `Collections.sort(List list)`: `Comparable` 인터페이스를 implements 한 객체의 List
- ❖ `Collections.sort(List list, Comparator<?> comp)`: `Comparable` 인터페이스를 implements 하지 않은 객체의 List



4-6. List의 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);
        //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());
                }
            };
    }
}
```

4-6. List의 정렬

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



5. Set 인터페이스

- ❖ 중복을 허용하지 않고 저장 순서가 없는 컬렉션 인터페이스
- ❖ 단방향으로 순서대로 접근할 수 있는 SortedSet과 양방향으로 접근할 수 있는 NavigableSet의 상위 인터페이스로 HashSet, LinkedHashSet, TreeSet 클래스가 Set 인터페이스를 implements 한 클래스
- ❖ 메소드
 - ✓ 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)

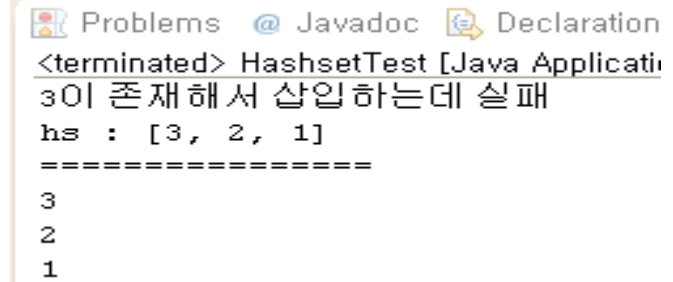
5.1 HashSet

- ❖ 구현된 인터페이스: Serializable, Cloneable, Iterable <E>, Collection <E>, Set <E>
- ❖ 해싱을 이용해서 데이터를 저장
- ❖ 중복 데이터는 한번만 삽입
- ❖ 널을 허용합니다.
- ❖ 동기화는 지원하지 않습니다.
- ❖ 데이터를 빠르게 검색할 수 있습니다.

- ❖ 생성자
 - ✓ HashSet(): 비어있는 해시 셋 생성
 - ✓ HashSet (int initialCapacity): 초기용량을 지정해서 해시 셋 생성
- ❖ 메소드
 - ✓ boolean add (E e): 데이터가 존재하지 않는 경우에 데이터를 추가
 - ✓ void clear (): 모든 데이터를 삭제
 - ✓ boolean remove (Object o): 데이터가 존재하는 경우에 데이터를 삭제
 - ✓ int size (): 데이터 개수를 리턴
 - ✓ Iterator<E> iterator(): iterator 리턴
 - ✓ boolean removeAll()
 - ✓ Object[] toArray()

실습(Set-HashSetTest.java)

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



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

5.2 LinkedHashSet

- ❖ 구현된 인터페이스: Serializable, Cloneable, Iterable <E>, Collection <E>, Set <E>
- ❖ HashSet을 상속받아서 HashSet과 동일한 방식으로 동작하지만 저장 순서를 기억하는 HashSet



실습(Set-LinkedHashSetTest.java)

```
package SetCollection;
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);
    }
}
```

```
Problems @ Javadoi
<terminated> LinkedHash
hs : [3, 2, 1]
=====
lhs : [2, 3, 1]
```

5.3 TreeSet

- ❖ 데이터를 순서대로 정렬하면서 저장하는 Set으로 트리 자료 구조를 이용해서 저장하기 때문에 검색속도는 느리다.
- ❖ 구현된 인터페이스: Serializable, Cloneable, Iterable<E>, Collection<E>, NavigableSet<E>, Set<E>, SortedSet<E>

❖ 생성자

- ✓ TreeSet (): 기본 생성자
- ✓ TreeSet (Collection <? extends E > c): 컬렉션을 가지고 생성

❖ 메소드

- ✓ boolean add(E o): o를 추가
- ✓ void clear(): 모두 삭제
- ✓ boolean contains(Object o): o를 포함하고 있는지 여부를 리턴
- ✓ Iterator <E > iterator (): 반복자 리턴
- ✓ E last(): 마지막 요소 리턴
- ✓ boolean remove(Object o): 지정된 요소가 있으면 세트로부터 삭제
- ✓ int size(): 데이터 개수 리턴

실습(Set-TreeSetTest.java)

```
package SetCollection;
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

실습(LottoTest.java)

```
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 + " ");  
    }  
}
```

6. Map 인터페이스

- ❖ Key와 Value를 하나의 쌍으로 묶어서 저장하는 컬렉션 인터페이스
- ❖ Key는 중복될 수 없지만 Value는 중복될 수 있습니다.
- ❖ 동일한 Key에 새로운 Value를 삽입하면 기존 Value는 제거되고 새로운 Value로 갱신됩니다.
- ❖ HashMap, Hashtable, LinkedHashMap, TreeMap 클래스가 Map 인터페이스를 implements 한 클래스
- ❖ 메소드
 - ✓ 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()

6.1 HashMap

java.lang.Object

java.util.AbstractMap <K, V>

java.util.HashMap<K, V>

- ❖ 인터페이스: Serializable, Cloneable, Map <K, V>
- ❖ Key와 Value를 매핑하는 클래스
- ❖ Key는 절대 중복될 수 없으며 각 Key는 1개의 Value만 매칭 – key는 equals 메소드를 이용해서 비교
- ❖ 모든 검색은 Key로 하므로 Key를 모르면 검색할 수 없습니다.
- ❖ Key와 Value에 널을 사용할 수 있습니다.
- ❖ Hashtable은 Key와 Value에 널이 들어갈 수 없습니다.
- ❖ Key의 비교를 == 이용해서 저장하는 IdentityHashMap 과 Key를 저장할 때 강한 참조가 아니라 약한 참조를 이용하는 WeakHashMap 클래스도 제공

6.1 HashMap

❖생성자

- ✓ HashMap(): 초기 용량을 16으로 하고 적재율은 0.75로 하여 HashMap 객체 생성
- ✓ HashMap(용량): 초기 용량을 용량으로 하고 기본 적재율 0.75로 적재
- ✓ HashMap(용량, 적재율): 초기 용량을 용량으로 하고 기본 적재율로 적재
- ✓ HashMap<자료형, 자료형> 객체명 = new HashMap< 자료형, 자료형 >();

❖메소드

- | | | |
|-----------|-----------------|---------------|
| ✓ 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 |

실습(Map-HashMapTest.java)

```
package Map;
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));
        }
    }
}
```

<terminator>

서울

부산

null

목포

제주

6.1 HashMap

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

실습(Map-Data.java)

```
package Map;
public class Data {
    private String num;
    private String name;
    private String address;
    public Data() {
        super();
    }
    public Data(String num, String name, String address) {
        super();
        this.num = num;
        this.name = name;
        this.address = address;
    }
    public String getNum() {
        return num;
    }
    public void setNum(String num) {
        this.num = num;
    }
}
```


실습(Map-Data.java)

```
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;  
}  
}
```

실습(Map-Main.java)

```
package Map;
import java.util.*;
public class Main {
    public static void main(String[] args) {
        Data obj1 = new Data("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));
        }
    }
}
```

6.1 HashMap

- ❖ HashMap은 2차원 이상의 배열을 표현할 때도 유용합니다.
- ❖ 2차원 배열은 각각의 배열의 의미를 표현하지 못하고 각각의 배열의 인덱스만을 가지고 있습니다.
- ❖ 이에 반해 1차원 배열을 키를 설정한 후 저장하게 되면 출력시에 키를 보고 의미를 판단할 수 있습니다.



실습(Map-MenuHash.java)

```
package Map;
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

실습(Map-MenuHash.java)

```
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();
}
}
```

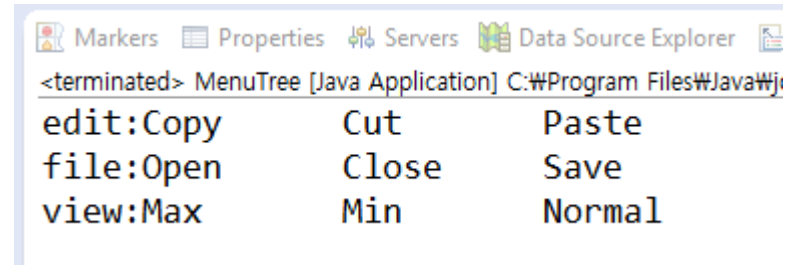
6.2 TreeMap

- ❖ 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



실습(Map-MenuTree.java)

```
package Map;
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);
    }
}
```



실습(Map-MenuTree.java)

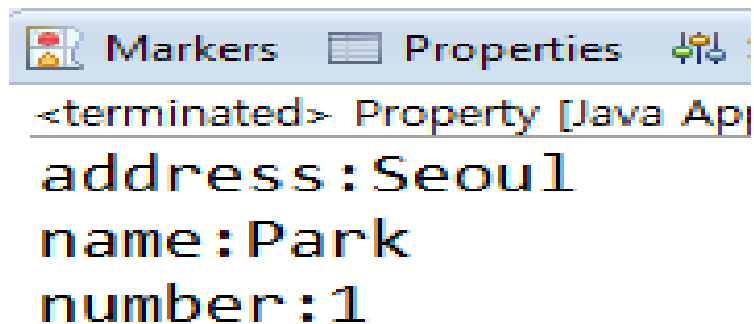
```
Set<String>set = menuMap.keySet();
Iterator<String>it = set.iterator();
String key = "";
String menuTemp[];
while(it.hasNext()){
    key = it.next();
    System.out.print(key + ":");
    menuTemp = menuMap.get(key);
    for(String imsi:menuTemp){
        System.out.printf("%-10s", imsi);
    }
 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


7. Properties

- ❖ 데이터를 쌍으로 저장하지만 Map과 달리 데이터 타입이 `<String, String>`으로 고정되어 있는 형태의 자료구조
- ❖ 애플리케이션의 환경설정을 파일에 저장한 후 읽기 위한 목적으로 많이 사용되고 있습니다.
- ❖ 메소드
 - ✓ `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()`
- ❖ Legacy Collection: Vector, Enumeration, Stack, Dictionary, Hashtable, Properties, BitSet 타입으로 대부분 동기화를 지원하는 구조로 만들어져 있는데 이러한 클래스들은 동기화를 지원하지 않는 클래스들보다 접근 속도가 느린 경우가 대부분이며 최근에는 동기화를 위한 별도의 컬렉션이 제공됩니다.
- ❖ JDK1.7에서는 Generic을 적용할 때 생성자에서 `<>`안에 자료형을 기재하지 않으면 변수의 타입을 보고 결정하는 기능이 추가되었습니다.

실습(Map-Property.java)

```
package Map;
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("c:\\ww\\pro.txt"), "텍스트로 내보내기");
            pro.storeToXML(new FileOutputStream("c:\\ww\\pro.xml"), "xml로 내보내기");
        } catch (IOException e) {
            System.out.println("저장 실패");
        }
    }
}
```



```
<terminated> Property [Java Ap
address:Seoul
name:Park
number:1
```

8. Collections 클래스

- ❖ Collection 과 관련된 메소드를 제공하는 클래스
- ❖ Arrays 클래스처럼 fill(), copy(), sort(), binarySearch() 메소드를 제공
- ❖ 동기화된 컬렉션을 만들어주는 메소드 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)
- ❖ 변경이 불가능한 컬렉션을 만들어주는 메소드 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)

8. Collections 클래스

- ❖ 컬렉션에 하나의 데이터 타입만 저장하는 컬렉션을 만들어주는 메소드 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)

9. Random 클래스

- ❖ 랜덤 한 값을 리턴시켜 주는 클래스
- ❖ 정수형 난수 발생은 특정 범위가 없습니다.
- ❖ 부동소수점을 가지는 난수는 0.0에서 1.0사이의 값을 받도록 되어 있습니다.
- ❖ 생성자
 - ✓ Random()
 - ✓ Random(long seed): seed를 설정합니다.
- ❖ 메소드
 - ✓ float nextFloat(): float 형 난수 생성
 - ✓ boolean nextBoolean();
 - ✓ int nextInt();
 - ✓ long nextLong()
 - ✓ double nextDouble()
 - ✓ void setSeed()

실습(etc-Lotto.java)

```
package etc;
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;
                    }
                }
            }
        }
    }
}
```

Problems @ Javac

```
<terminated> Lotto [Java]
Lotto[0] = 12
Lotto[1] = 42
Lotto[2] = 26
Lotto[3] = 24
Lotto[4] = 19
Lotto[5] = 40
```

실습(Lotto.java)

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

10. StringTokenizer 클래스

❖ 문자열을 분할해주는 클래스

❖ 생성자

- ✓ StringTokenizer(String str): 문자열만 매개변수로 입력해서 생성하는 경우로 공백으로 구분
- ✓ StringTokenizer(String str, String delim): str를 delim으로 구분해서 생성

❖ 메소드

- ✓ 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

실습(TokenTest.java)

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

11. Date 클래스

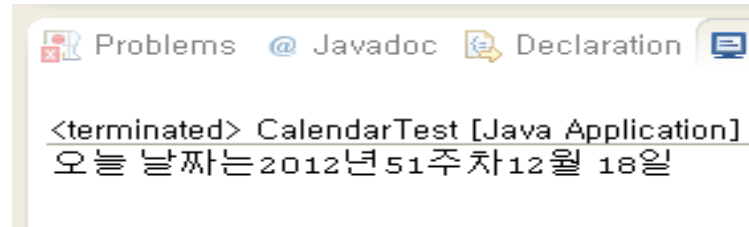
- ❖ 날짜와 시간에 관련된 정보를 저장하고 추출해 주는 클래스입니다.
- ❖ 생성자
 - ✓ Date()
 - ✓ Date(long timeInMillis)
 - ✓ Date(int year, int month, int date)
 - ✓ Date(int year, int month, int date, int hrs, int min, int sec)
- ❖ 정수를 대입할 때는 1970년 1월 1일 자정부부터 지나온 시간을 1/1000 초 단위로 설정
- ❖ 년도는 1900년 이후 지나간 년도를 설정해야 하고 월은 1 적게 설정해야 한다는 것에 주의 해야 합니다.
- ❖ 대부분의 메소드가 deprecated가 되서 Calendar 클래스를 사용하는 것을 권장
- ❖ toString 메소드가 재정의 되어있어 객체를 직접 출력하면 날짜 및 시간이 출력됩니다.
- ❖ 데이터베이스의 Date 자료형과 직접 연동이 가능

12. Calendar 클래스

- ❖ 객체 생성 – 직접 객체 생성은 안됩니다.
 - ✓ Calendar.getInstance()를 이용해서 객체를 생성하거나 Calendar 클래스를 상속받아서 만들어진 GregorianCalendar, BuddhistCalendar 클래스의 객체를 생성해서 사용
- ❖ Calendar 객체의 데이터를 가져오기: Calendar객체.get(상수);
 - ✓ Calendar.YEAR 년
 - ✓ (Calendar.MONTH)+1 월은 1달 적게 나온다
 - ✓ Calendar.DAY_OF_MONTH 일
 - ✓ Calendar.DAY_OF_WEEK 주의 몇 번째 날인가를 추출합니다.
 - ✓ 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)

실습(etc-CalendarTest.java)

```
package etc;
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");
    }
}
```



실습(etc-DateGap.java)

```
package etc;
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");
        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)] + "요일입니
다.");
        long gap = (date2.getTimeInMillis() - date1.getTimeInMillis()) / 1000;
        System.out.println("우리가 만난지 오늘은 " + gap / (24 * 60 * 60) + "일째 입니다.");
    }
}
```

<terminated> DateGap [Java Application] C:\Program Files\Java\jdk1.7.0_75\bin\java.exe

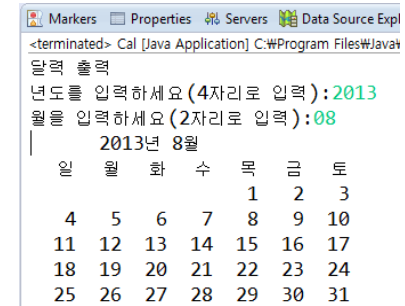
제시카를 만난 날은 1986년 5월 5일 월요일이고

오늘은 2013년 8월 8일 목요일입니다.

우리가 만난지 오늘은 9957일째 입니다.

실습(etc-Cal.java)

```
package etc;
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");
        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 입력하세요!!!");
            }
            year = Integer.parseInt(imsi);
```



달력 출력

년도를 입력하세요 (4자리로 입력): 2013

월을 입력하세요 (2자리로 입력): 08

2013년 8월

일	월	화	수	목	금	토
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

실습(etc-Cal.java)

```
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 입력하세요!!!");
}
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);
```

실습(etc-Cal.java)

```
// 1일의 요일을 구합니다.
START_DAY_OF_WEEK = startDate.get(Calendar.DAY_OF_WEEK);
// 월의 마지막 날을 찾아옵니다.
END_DAY = endDate.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();
}
}
}
```


13. SimpleDateFormat 클래스

❖ java.text패키지에 있는 클래스로 날짜 데이터를 원하는 형태의 문자열로 만들어주는 클래스

❖ 사용자가 입력하는 pattern대로 날짜정보를 추출

문자 의미

G 기원

M 월

W 달에 있어서의 주

d 달에 있어서의 날

E 요일

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

y

w

D

F

a

년

해에 있어서의 주

해에 있어서의 날

달에 있어서의 요일

오전/오후

❖ 사용법

Date 날짜변수 = new Date();


SimpleDateFormat 서식변수 = new SimpleDateFormat("날짜 서식");

String 문자열변수 = 서식변수.format(날짜변수);

❖ parse(문자열) 메소드를 이용하면 포맷의 문자열을 원하는 포맷의 Date 타입으로 변환이 가능

실습(format-DateFormat1.java)

```
package format;
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 EEE요일 a hh시
mm분 ss초");
        System.out.println(sdf.format(day));
    }
}
```



The screenshot shows a console window with the following text: Problems, Javadoc, Declaration, Console. Below the tabs, it says: <terminated> TestDate [Java Application] C:\Program Fil. At the bottom, it shows the date and time: 2012년 12월 18일 화요일 오후 01시 37분 39초.

실습(format-DateFormat2.java)

```
package format;
```

```
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.

실습(format-DateFormat2.java)

```
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.(입력
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 있습니다.");
}
}
```

14. DecimalFormat 클래스

❖ java.text 패키지에 있는 클래스로 숫자 데이터를 원하는 형태의 문자열로 만들어 주는 클래스

❖ 사용자가 입력하는 pattern대로 날짜정보를 추출

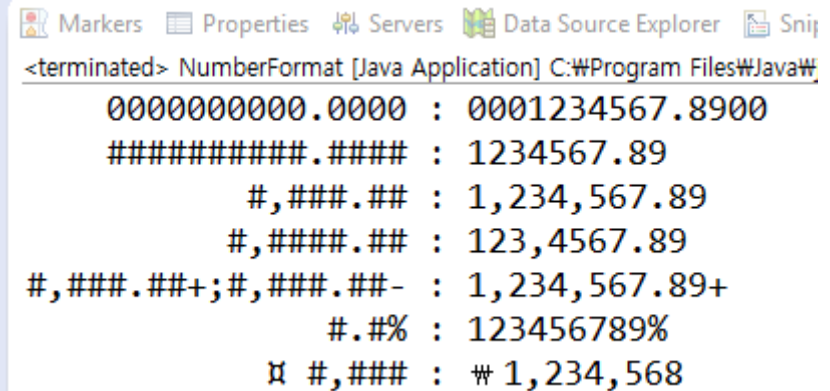
문자	의미
0	10진수(값이 없을 때는 0)
#	10진수
.	소수
-	음수 기호
,	천 구분 기호
;	패턴 구분자
%	퍼센트
₩u2030	퍼밀
₩u00A4	통화 기호

❖ 사용법

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

실습(format-DecimalFormat.java)

```
package format;
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));
        }
    }
}
```



```
<terminated> NumberFormat [Java Application] C:\Program Files\Java\
0000000000.0000 : 0001234567.8900
#####.### : 1234567.89
#,###.## : 1,234,567.89
#,####.## : 123,4567.89
#,###.##+;#,###.##- : 1,234,567.89+
#.##% : 123456789%
₩ #,### : ₩ 1,234,568
```

15. ChoiceFormat 클래스

- ❖ java.text패키지에 있습니다.
- ❖ 2개의 배열을 이용하여 범위에 속하는 숫자를 문자열로 치환해주는 클래스
- ❖ 숫자 배열은 반드시 오름차순으로 정렬되어 있어야 합니다.
- ❖ 사용법

```
ChoiceFormat 포맷변수 = new ChoiceFormat(숫자 배열, 문자열 배열);
String 문자열변수 = 포맷변수.format(값);
```



실습(format-SelectFormat.java)

```
package format;

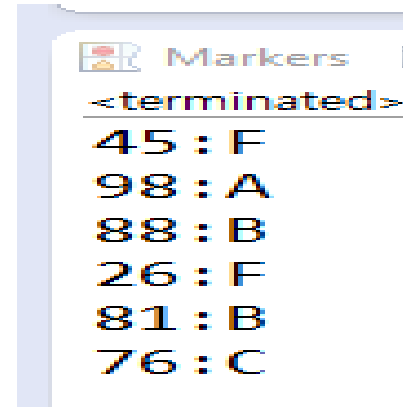
import java.text.*;

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]));
        }
    }
}
```



```
Markers [
<terminated>
45 : F
98 : A
88 : B
26 : F
81 : B
76 : C
```


16. MessageFormat 클래스

- ❖ java.text패키지에 있습니다.
- ❖ 문자열이 삽입될 양식을 미리 만들어 두고 양식에 맞는 문자열로 리턴해주는 클래스
- ❖ 여러 개의 데이터를 같은 서식으로 출력할 때 사용하는 클래스로 일반적으로 반복문을 이용하거나 toString 메소드를 이용해서 작성하는 경우가 많습니다.
- ❖ 양식을 작성할 때 {숫자}를 이용해서 작성하고 Message.format(양식,문자열)을 이용해서 생성합니다.

실습(format-FormFormat.java)

```
package format;
```

```
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 크라이스트처치"
```

```
        };
```

```
        String result = MessageFormat.format(msg, person1);
```

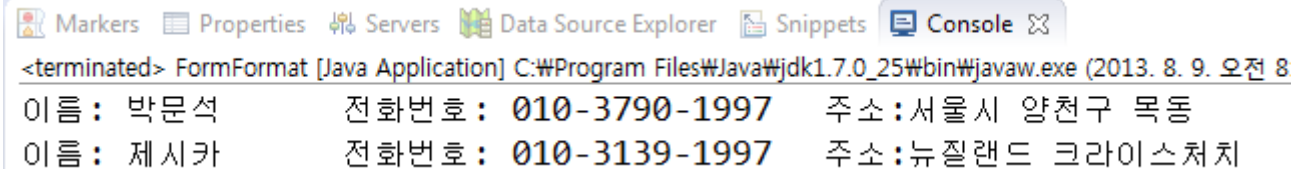
```
        System.out.println(result);
```

```
        result = MessageFormat.format(msg, person2);
```

```
        System.out.println(result);
```

```
    }
```

```
}
```

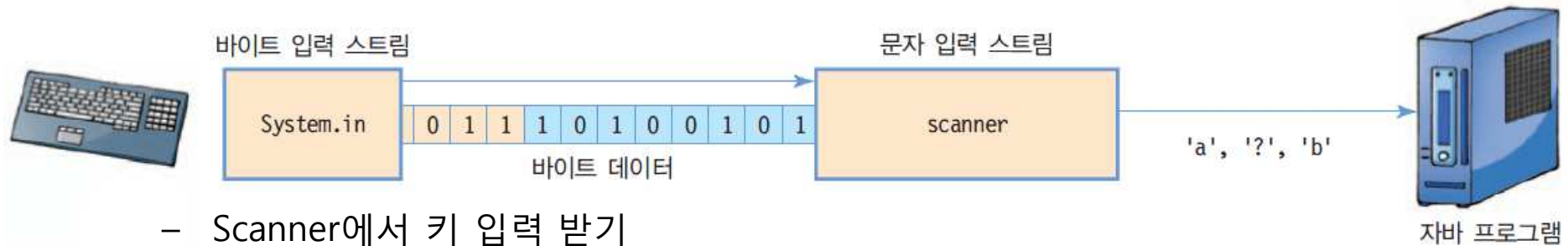


```
<terminated> FormFormat [Java Application] C:\Program Files\Java\jdk1.7.0_25\bin\javaw.exe (2013. 8. 9. 오전 8  
이름 : 박문석      전화번호 : 010-3790-1997      주소 : 서울시 양천구 목동  
이름 : 제시카     전화번호 : 010-3139-1997      주소 : 뉴질랜드 크라이스트처치
```

16. Scanner 클래스

- Scanner 클래스
 - java.util.Scanner 클래스
 - Scanner 객체 생성

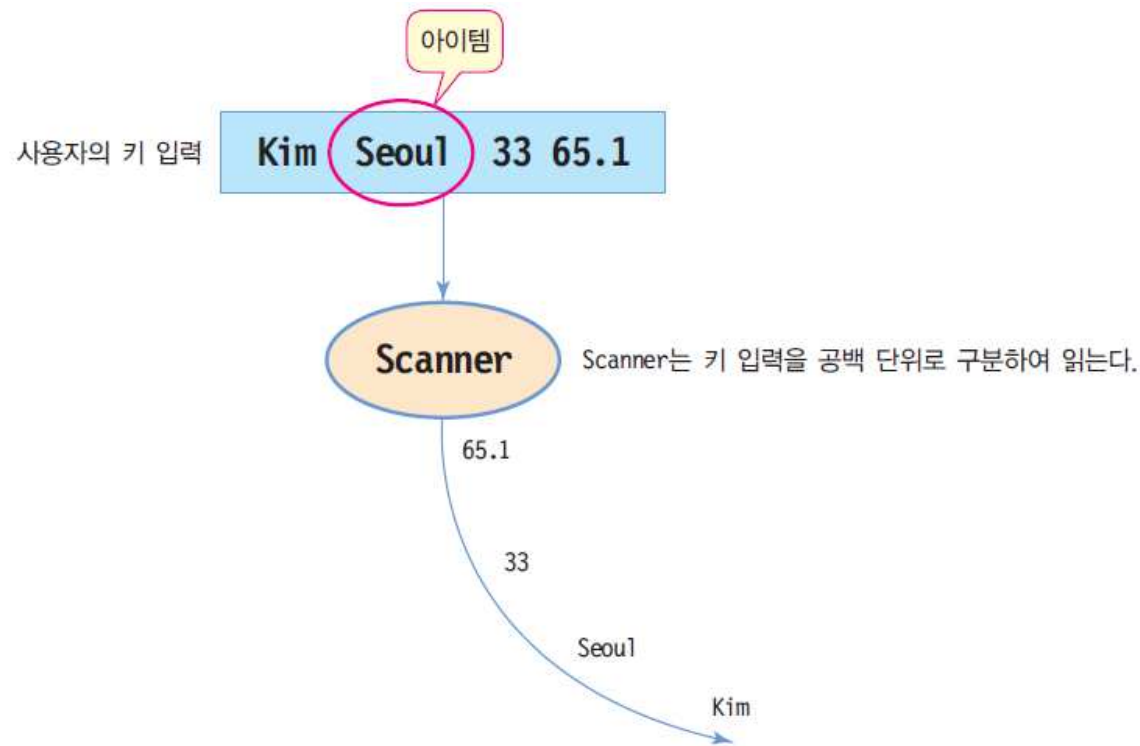
```
Scanner a = new Scanner(System.in);
```



- Scanner에서 키 입력 받기
 - Scanner는 입력되는 키 값을 공백으로 구분되는 아이템 단위로 읽음
 - 공백 문자 : 'wt', 'wf', 'wr', ' ', 'wn'

16. Scanner 클래스

```
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
```



16. Scanner 클래스

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

메소드	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

etc-ConsoleInput.java

```
//Scanner 클래스를 사용하기 위해서 import
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();
    }
}
```

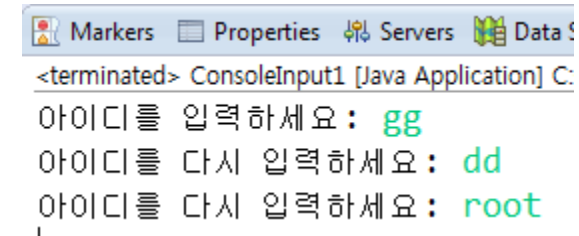
Problems @ Javadoc Declarati
〈terminated〉 ConsoleInput [Java Applic
임의의 숫자를 입력하세요: 35
n:35

etc-ConsoleInput1.java

```
package etc;

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();
    }
}
```



<terminated> ConsoleInput1 [Java Application] C:
아이디를 입력하세요: gg
아이디를 다시 입력하세요: dd
아이디를 다시 입력하세요: root

16. Scanner 클래스

Scanner를 이용하여 나이, 체중, 신장 데이터를 키보드에서 입력 받아 다시 출력하는 프로그램을 작성

```
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.");
    }
}
```

나이, 체중, 신장을 빈칸으로 분리하여 순서대로 입력하세요

35 75 175

당신의 나이는 35살입니다.

당신의 체중은 75.0kg입니다.

당신의 신장은 175.0cm입니다.

17.1 java.time

- ❖ 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.LocalDateTime: 시간과 관련된 클래스
- ❖ 위 2개 클래스의 객체는 now()라는 static 메소드와 of(매개변수)라는 static 메소드를 이용해서 생성합니다.
- ❖ 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 메소드도 있음
- ❖ 객체의 전후 관계를 비교하는 isAfter, isBefore, isEqual 메소드가 있음

Local.java

```
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
```

17.2 java.time.Instant

- ❖ Epoch Time으로 부터 경과된 시간을 저장하는 클래스
- ❖ now()라는 static 메소드를 이용해서 객체 생성 가능하고 Date 객체의 toInstant()를 이용해서 생성가능
- ❖ static 메소드인 from(Instant instan) 메소드를 이용해서 Date 객체로 변환 가능
- ❖ UTC(세계 표준 협정시)를 기준으로 하기 때문에 LocalTime과는 차이가 납니다.
- ❖ 이전에는 GMT를 사용했는데 GMT보다는 UTC가 조금 더 정확합니다.



17.3 LocalDateTime

- ❖ `LocalDateTime`: `LocalDate` 와 `LocalTime`을 합쳐 놓은 클래스
- ❖ `ZonedDateTime`: `LocalDateTime`에 시간대(time zone)을 추가한 클래스
- ❖ `LocalDateTime`은 `LocalDate` 와 `LocalTime`을 합쳐서 만들 수 있는 다양한 형태의 of 메소드를 제공
- ❖ `toLocalDate()` 와 `toLocalTime()`을 이용해서 `LocalDate` 와 `LocalTime`으로 변환 가능한 of 메소드를 제공하고 다양한 형태로 만들 수 있는 메소드도 제공
- ❖ `ZoneId`의 of 메소드를 이용해서 `ZoneId` 객체를 만든 후 `ZonedDateTime` 객체의 `atZone` 메소드의 매개변수로 넘겨주면 Zone을 설정한 `DateTime` 객체를 만들 수 있습니다.
- ❖ `ZoneId`의 목록은 `ZoneId.getAvailZoneIds()`를 이용해서 얻어낼 수 있습니다.

DateTime.java

```
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);
```

```
    }
```

```
}
```

```
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]
```

17.4 Period & Duration

❖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



Different.java

```
import java.io.FileInputStream;
import java.time.Duration;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.time.Period;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.temporal.ChronoUnit;

import javazoom.jl.player.Player;
```

```
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
```

Different.java

```
public class Main {  
  
    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
```


Different.java

```
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());
```

```
}
```

```
}
```

```
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
```

정규 표현식

1. 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 편집기와 프로그래밍 언어에서 문자열의 검색과 치환을 위해 지원하고 있으며, 특히 펄과 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 펄, 자바 스크립트, 루비, Tcl처럼 기본 내장되어 있는 반면 닷넷 언어, 자바, 파이썬, POSIX C, C++ (C++11 이후)에서는 표준 라이브러리를 이용하여 구현
- ❖ 그 밖의 대부분의 언어들은 라이브러리를 통해 정규식을 제공
- ❖ Java에서는 java.util.regex 패키지에 있는 Match 클래스와 Pattern 클래스를 이용하여 문자열을 정규 표현식으로 나타낼 수 있습니다.

2. 문법

❖ '.' 특수문자

- ✓ 임의의 한 문자를 의미
- ✓ '.' 특수문자가 위치한 곳에는 반드시 한 글자가 위치
- ✓ 패턴 : $a.b \rightarrow$ 일치하는 문자열 : acb, adb, azb 등
- ✓ 패턴 : $ab. \rightarrow$ 일치하지 않는 경우 : $ab, abcd$ 등

❖ '*' 특수문자

- ✓ 바로 앞의 문자를 0번 이상 반복
- ✓ * 앞에는 한 글자 이상의 단어가 반드시 위치
- ✓ 패턴 : $ab^*c \rightarrow abc, abbc, abbbc$ 등
- ✓ 패턴 : $^*ab \rightarrow$ 불가능

❖ '+' 특수문자

- ✓ 바로 앞의 문자를 1번 이상 반복
- ✓ + 앞에는 한 글자 이상의 단어가 반드시 위치
- ✓ 패턴 : $a+b \rightarrow aab, aaab, aaaab$ 등
- ✓ 패턴 : $+ab \rightarrow$ 불가능

2. 문법

❖ '?' 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

2. 문법

❖ '[']' 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] → 숫자가 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)

2. 문법

❖ '{ }' 특수문자

- ✓ 특수문자 앞의 문자나 문자열의 반복되는 개수
- ✓ `go{5}ggle` → `goooooogle`
- ✓ `go{3,}ggle` → `google`, `gooogle`, `goooooogle` 등 (o가 3개 이상)
- ✓ `go{2,4}ggle` → `google`, `gooogle`, `goooooogle` (o가 2개 이상 4개 이하)

❖ '(' ')' 특수문자

- ✓ ()안의 글자들을 하나의 문자로 본다.
- ✓ `(hero_){3}` → `hero_hero_hero_`
- ✓ `(hero_)*` → `null`, `hero_`, `hero_hero_` 등

❖ '|' 특수문자

- ✓ 패턴 안에서 OR연산 사용시 사용
- ✓ `man|woman` → `man`, `woman`, `manwoman`, `superman` 등

2. 문법

- ❖ `\w` : 알파벳이나 숫자
- ❖ `\W` : `\w`의 not. 즉 알파벳이나 숫자를 제외한 문자
- ❖ `\d` : [0-9]와 동일
- ❖ `\D` : 숫자를 제외한 모든 문자



3. Pattern 클래스

❖ 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")
```



4. Matcher 클래스

- ❖ `find()` : 패턴이 일치하는 경우 `true`를 반환하고, 그 위치로 이동(여러 개가 매칭되는 경우 반복 실행가능)
- ❖ `find(int start)` : `start`위치 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)` : 지정된 그룹이 매칭되는 끝 다음 문자위치 반환
- ❖ `group()` : 매칭된 부분을 반환
- ❖ `group(int group)` : 매칭된 부분중 `group`번 그룹핑 매칭부분 반환
- ❖ `groupCount()` : 패턴내 그룹핑한(괄호지정) 전체 갯수 반환
- ❖ `matches()` : 패턴이 전체 문자열과 일치할 경우 `true` 반환

5. 샘플

❖String 클래스의 메소드를 이용한 정규식 패턴 검사

```
public class Main {  
    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

5. 샘플

❖ Pattern 클래스와 Matcher 클래스의 메소드를 이용한 정규식 패턴 검사

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

public class Main {
    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

6. 자주 사용하는 Pattern

- ❖ E-Mail : `"^[a-zA-Z0-9]+[@][a-zA-Z0-9]+[.][a-zA-Z0-9]+$`
- ❖ 휴대폰 : `^01(0|1|[6-9])-(\w{3}|\w{4})-\w{4}$`
- ❖ 일반전화 : `^\w{2,3} - \w{3,4} - \w{4}$`
- ❖ 주민등록번호 : `\w{6} \w- [1-4]\w{6}`
- ❖ IP 주소 : `([0-9]{1,3}) \. ([0-9]{1,3}) \. ([0-9]{1,3}) \. ([0-9]{1,3})`

6. 자주 사용하는 Pattern

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

public class Main {
    public static void main(String[] args) {
        Pattern p = Pattern.compile("^01(0|1|[6-9])-(\\Wd{3}|\\Wd{4})-\\Wd{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 전화번호입
니다.");
        }
    }
}
```

6. 자주 사용하는 Pattern

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

6. 자주 사용하는 Pattern

```
9]+$");  
  
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 주소입  
니다.");  
}
```


6. 자주 사용하는 Pattern

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

6. 자주 사용하는 Pattern

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