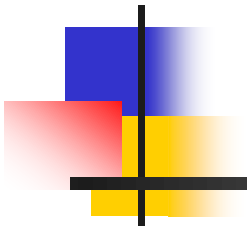
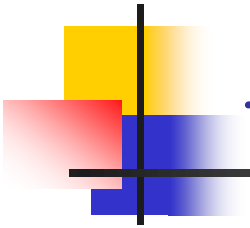


java 13강 컬렉션

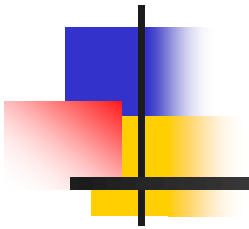
양 명 숙
[now4ever7@gmail.com]



목차

- 컬렉션 프레임워크
  - ArrayList
  - HashSet
  - HashMap
- Generic
- Iterator

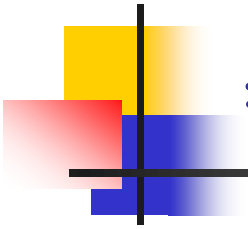
- TreeSet<E> 클래스
- Comparable<T> 인터페이스의 구현



컬렉션

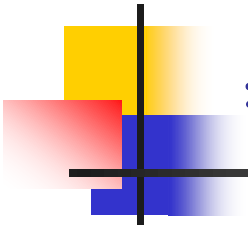
컬렉션 프레임워크(Collection Framework)

- 프레임워크(Framework)
  - 잘 정의된, 약속된 구조나 골격
  - 잘 정의된 클래스들의 모임
  - 컬렉션과 관련된 클래스들의 정의에 적용되는 설계의 원칙, 또는 구조가 존재함
    - 모든 컬렉션 클래스를 표준화된 방식으로 다룰 수 있도록 체계화됨
- 컬렉션(Collection)
  - 데이터의 저장, 그리고 이와 관련 있는 알고리즘을 구조화 해 놓은 프레임워크
  - 컨테이너 클래스
  - 데이터의 저장을 위해 정의된 클래스
  - 변수들의 조직적인 집합
- 컬렉션 프레임워크
  - **데이터 그룹을 저장하는 클래스들을 표준화한 설계**
  - 다수의 데이터를 다루는 데 필요한 다양하고 풍부한 클래스들을 제공함



컬렉션(Collection)

- 컬렉션(Collection)
  - ArrayList, HashSet, HashMap 등
- 컬렉션 클래스들의 특징
  - 컬렉션 클래스들은 데이터를 보관할 수 있으며 삭제, 검색, 삽입 등의 기능을 함
- 배열과 구분 짓는 특징은 메모리의 사이즈를 동적으로 확장할 수 있다는 것
  - 배열은 첨자로 배열을 생성하고, 데이터를 할당할 때 정확하게 첨자의 범위 내에서만 사용할 수 있음
- 컬렉션 클래스는 데이터를 삽입했을 때 동적으로 메모리를 늘림



컬렉션(Collection)

- 배열
  - 같은 타입의 변수 여러 개를 저장하기 위한 메모리 공간
  - 사용하기 간단, 첨자연산이 빠르기 때문에 효율이 좋은 편
  - 단점
    - 최초 생성할 때 크기를 한 번 지정하면 변경할 수 없다는 것(최초 생성할 때 크기가 고정)
    - 정수형의 첨자만 쓸 수 있다는 것
- ArrayList 컬렉션
  - 실행 중에 언제든지 크기를 확장, 축소할 수 있는 동적 배열
    - 가변적인 크기를 다룰 수 있음
  - 실행 중에 신축적으로 크기를 변경할 수 있기 때문에 생성할 때 미리 크기를 결정하지 않아도 상관없음
  - 처음에는 작은 크기로 생성했다가 입력량이 많아지면 그때 늘릴 수 있음
  - 꼭 필요한 만큼만 메모리를 할당할 수 있고, 부족하면 늘릴 수도 있음

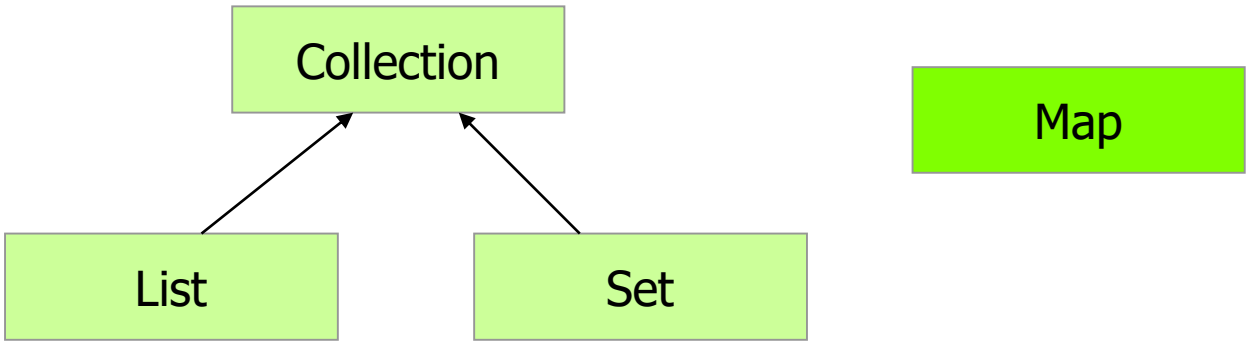
컬렉션 프레임워크의 핵심 인터페이스

- 컬렉션 프레임워크의 핵심 인터페이스
  - List, Set, Map

인터페이스	특 징
List	순서가 있는 데이터의 집합. 데이터의 중복을 허용함 예) 대기자 명단 구현 클래스 : ArrayList, LinkedList, Stack, Vector 등
Set	순서를 유지하지 않는 데이터의 집합. 데이터의 중복을 허용하지 않음 예) 양의 정수집합, 소수의 집합 구현 클래스 : HashSet, TreeSet 등
Map	키(key)와 값(value)의 쌍으로 이루어진 데이터의 집합 순서는 유지되지 않으며, 키는 중복을 허용하지 않고, 값은 중복을 허용함 예) 우편번호, 지역번호(전화번호) 구현 클래스: HashMap, TreeMap, Hashtable, Properties 등

컬렉션 프레임워크의 핵심 인터페이스

- 컬렉션 프레임워크의 모든 컬렉션 클래스들은 List, Set, Map 중의 하나를 구현하고 있으며, 구현한 인터페이스의 이름이 클래스의 이름에 포함되어 있음
- 컬렉션 클래스에 저장된 데이터를 읽고, 추가하고, 삭제하는 등 컬렉션을 다루는데 필요한 기능이 메서드로 정의되어 있음
- JDK 1.2 부터 컬렉션 프레임워크 등장
  - 이전부터 존재하던 클래스 – Vector, Stack, Hashtable, Properties



컬렉션 프레임워크의 핵심 인터페이스간의 상속 계층도

Vector와 ArrayList

길이를 늘릴 때 두 배로 늘려서 알아서 값 복사해옴
근데 이 과정 자체가 시간이 좀 걸리는 편

- Vector와 ArrayList – 컬렉션 프레임워크에서 가장 많이 사용되는 컬렉션 클래스
  - List<E> 인터페이스를 구현하는 컬렉션 클래스
  - ArrayList – 기존의 Vector를 개선한 것
  - Object 배열을 이용해서 데이터를 순차적으로 저장함
  - 데이터의 저장을 위해서 인덱스 정보를 별도로 관리할 필요가 없고, 데이터의 삭제를 위한 추가적인 코드의 작성이 필요 없음
  - 저장되는 인스턴스의 수에 따라서 그 크기도 자동으로 늘어나지만, 효율은 떨어짐
- 특성
  - 동일한 인스턴스의 중복 저장을 허용한다.
  - 인스턴스의 저장 순서가 유지된다.

Vector와 ArrayList

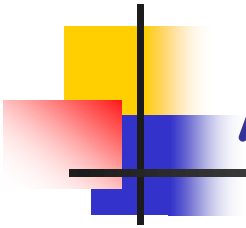
Object 배열을 이용해서 데이터를 순차적으로 저장함

- 첫번째로 저장한 객체는 Object 배열의 0번째 위치에 저장되고 그 다음에 저장하는 객체는 1번째 위치에 저장됨

- 이런 식으로 계속 배열에 순서대로 저장되며, 배열에 더 이상 저장할 공간이 없으면 보다 큰 새로운 배열을 생성해서 기존의 배열에 저장된 내용을 새로운 배열로 복사한 다음에 저장됨

- Vector와 ArrayList 같이 배열을 이용한 자료구조는 데이터를 읽어오고 저장하는 데는 효율이 좋지만,
- 용량을 변경해야 할 때는 새로운 배열을 생성한 후 기존의 배열로부터 새로 생성된 배열로 데이터를 복사해야 하기 때문에 효율이 떨어진다는 단점이 있다

처음에 인스턴스를 생성할 때, 저장할 데이터의 개수를 고려하여 **충분한 용량의 인스턴스를 생성**하는 것이 좋다



ArrayList의 장단점

- 장점
  - 데이터의 참조가 용이해서 빠른 참조가 가능하다
    - 예) 13번째 위치의 데이터를 참조하고 싶다면 인덱스 값 12를 이용해서 바로 접근이 가능
    - alist.get(12)
- 단점
  - 저장소의 용량을 늘리는 과정에서 많은 시간이 소요된다
  - 데이터의 삭제에 필요한 연산과정이 매우 길다
- ArrayList
  - => 저장하게 되는 데이터의 수가 예측 가능하며, 빈번한 데이터의 참조가 일어나는 상황에서 유용하게 사용할 수 있는 컬렉션 클래스

제네릭스(Generics)

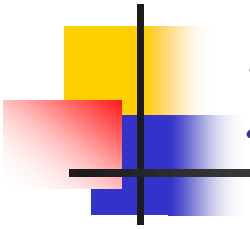
- 제네릭스(Generics)
  - JDK 1.5에 추가된 기능
  - 컬렉션에 저장하는 객체의 타입을 컴파일시에 체크하기 때문에 객체의 타입 안정성을 높이고, **꺼낼 때는 자동으로 형변환해주**기 때문에 편리함
  - **컬렉션에 저장할 객체의 타입을 지정**
  - 지정한 타입의 객체만 해당 컬렉션에 저장할 수 있다

저장객체 타입 저장하면 자동으로 형변환해줌

컬렉션 클래스<저장할 객체의 타입> 변수명= new 컬렉션 클래스<저장할 객체의 타입>;

```
ArrayList<Integer> list =  new ArrayList<Integer>();
list.add(10);
list.add(3.14);  //에러
int num = list.get(0);
```

```
ArrayList list =  new ArrayList();
list.add(new Integer(10));
list.add(3.14);
Integer num = (Integer)list.get(0);
Double d = (Double)list.get(1);
```



제네릭스(Generics)

- 일반 컬렉션의 요소 타입은 Object 이므로 임의의 요소를 저장할 수 있음
  - 어떤 객체든지 컬렉션에 넣을 수 있으며, 이를 막을 수 있는 문법적인 방법이 전혀 없음
- 문제점
  - 컬렉션에 저장된 정보를 읽을 때 Object 타입으로 리턴되므로 원하는 타입으로 캐스팅해야 함
  - 값 타입을 컬렉션에 저장할 때는 Object 타입으로 변환하는 박싱이 필요, 꺼낼 때는 언박싱이 필요 – 성능상의 불이익
    - 정수값 하나를 넣어도 Object로 바꾼 후에 넣기 때문에 메모리도 많이 소모, 속도도 느림
- 제네릭을 사용하면 형매개변수(타입 인수)로 처리 대상을 지정할 수 있으므로 이런 문제 해결

예제

• Boxing - 스택에 저장된 기본자료형 데이터를 힙영역의 참조형으로 변환하는것
박싱을 해주는 클래스가 Wrapper 클래스이다

• Unboxing - 참조형의 데이터를 기본형으로 변환

```
import java.util.ArrayList;
class ArrayListTest1{
	public static void main(String[] args){
		ArrayList list = new ArrayList(3);

		/* 데이터의 저장 */
		list.add(new Integer(10)); // JDK 5.0 이전 - 컬렉션에 값을 저장할 때 객체로 저장해야 하므로, Wrapper클래스 사용
		list.add(new Integer(20));
		list.add(30); //JDK 5.0-autoboxing : 기본 자료형 값이 컴파일러에 의해서 자동으로 Wrapper 클래스로 변환되어 저장

		System.out.println("ArrayList의 크기 : " + list.size());

		/* 데이터의 참조 */
		System.out.println("-----------------");
		for(int i=0; i<list.size(); i++){
			Object obj=list.get(i); //ArrayList의 요소 꺼내기
			Integer num=(Integer)obj;
			System.out.println(num);
		}

		list.add(new Double(3.14));
		list.add("java!!");
		Object obj=list.get(3);
		Double num2=(Double)obj;  //3.14
		System.out.println(num2);
```

예제

```
String str=list.contains(new Double(3.14))?"포함함": "포함하지 않음";
System.out.println(str);

//JDK 5.0 이후 - Generic : 컬렉션에 저장할 객체의 타입을 지정
ArrayList<Integer> list2=new ArrayList<Integer>();

/*list2.add(1);  //autoboxing
list2.add(2);
list2.add(3);*/

for (int i=0;i<3 ;i++ )
{
        list2.add(i+1);
}
System.out.println("--------삭제 전-----------");
for(int i=0; i<list2.size(); i++)
{
        int num = list2.get(i); //형변환 필요없다, unboxing:저장된 값을 꺼낼 때도 기본
형 형태의 값을 바로 얻는 것

        System.out.println(num);
}

/* 데이터의 삭제 */
list2.remove(0);
System.out.println("--------삭제 후-----------");
for(int i=0; i<list2.size(); i++)
        System.out.println(list2.get(i));
```

```
System.out.println("확장 for 이용");
for(int k: list2)
{
        System.out.println(k);
}
```

오토박싱(autoboxing)

- jdk 5.0 이전 버전
  - 컬렉션에 값을 저장할 때 객체로 저장해야 하므로, 기본형 값을 저장하기 위해서는 Integer나 Long 과 같은 Wrapper 클래스를 사용해야 했음(boxing)
- JDK 5.0 이후
  - 기본 자료형 값을 직접 컬렉션에 저장할 수 있음(autoboxing)

- 오토박싱(autoboxing)
  - 기본 자료형 값이 컴파일러에 의해서 자동으로 Wrapper 클래스로 변환되어 저장되는 것
- 언박싱(unboxing)
  - 저장된 값을 꺼낼 때도 변환과정을 거치지 않고 기본형 형태의 값을 바로 얻는 것

예제

```
import java.util.*;
class AutoBoxingTest{
    public static void main(String[] args) {
        ArrayList list = new ArrayList();
        list.add(new Integer(10));
        list.add(new Integer(20));

        Integer i = (Integer)list.get(0);
        int value = i.intValue();

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

        //JDK 5.0 이후
        ArrayList<Integer> list2 = new ArrayList<Integer>();
        list2.add(100);  //오토박싱 int => Integer
        list2.add(500);

        int val = list2.get(0); //언박싱  Integer => int
        System.out.println("₩nval = " + val);
    }
}
```

Integer 클래스
public int intValue()
- Returns the value of this Integer as an int.

실습

- Generic ArrayList에 double형 값을 3개 저장하고, 꺼내오기
  - 출력시에만 for 문 이용

- Generic ArrayList에 String 객체 5개 저장하고, 꺼내오기
  - 입력, 참조 모두 for 문 이용

```
3.14
5.87
2.476

1 : Hello Java
2 : Hello Java
3 : Hello Java
4 : Hello Java
5 : Hello Java
```

Object 클래스의 toString()메서드

```
p객체를 표현하는 기본 문자열은? Person@61de33
p객체를 표현하는 기본 문자열 약식은? Person@61de33
Person@61de33
this ? Person@61de33
```

```
class Person {
    public void display(){
        System.out.println("this ? " + this);
    }
}

public class ObjectTest {
    public static void main(String[] ar) {
        Person p = new Person();
        System.out.println("p객체를 표현하는 기본 문자열은? " + p.toString());
        System.out.println("p객체를 표현하는 기본 문자열 약식은? " + p);
        System.out.println(p);
        p.display();
    }
}
```

toString() 메서드의 결과 : 클래스명@16진수 해시코드

Person 클래스에는 toString() 메서드가 선언되어 있지 않지만, Object 클래스를 자동으로 상속받으므로, Object 클래스의 메서드인 toString() 메서드를 사용할 수 있다

hashCode() – 객체의 메모리 주소를 16진수로 리턴

public void println(Object x)

public String toString()

toString() 메서드

- Object클래스의 toString() 메서드
  - 해당 클래스가 어떤 객체인지를 쉽게 나타낼 수 있는 메서드
  - **객체를 문자열로 표현하는 값을 리턴**
- p.toString() 과 p 를 출력한 결과값이 동일
  - 자바에서는 클래스의 멤버가 할당되어 있는 곳의 주소를 숨기려는 속성이 있음
  - 자바에서는 c언어에서의 포인터를 내부 포인터로 바꾸고, 주소를 출력해 볼 수 있는 예약어를 사용하지 못하도록 만들었음
  - => 직접 주소를 출력해보려고 객체를 출력하면 자동으로 출력 형식의 메서드로 연결해 버림
    - 그 메서드가 toString() 메서드
- toString() 메서드가 자동으로 호출되는 경우
  - [1] System.out.println() 메서드에 매개변수로 들어가는 경우
  - [2] 객체에 대하여 더하기(+) 연산을 하는 경우

예제

```
객체 출력 : Person2 [name=null, age=0]
Person2 [name=null, age=0]
```

```
class Person2 {
	private String name;
	private int age;

	//Object 클래스의 toString() 메서드를 오버라이딩
	public String toString() {
		return "Person2 [name=" + name + ", age=" + age + "]";
	}
}

public class ObjectTest2 {
	public static void main(String[] ar) {
		Person2 p = new Person2();
		System.out.println("객체 출력 : " + p);
		System.out.println(p.toString());
	}
}
```

예제2

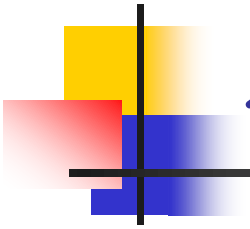
```
import java.util.*;
class MyMember
{
	private String id;
	private String name;

	MyMember(String id, String name)
	{
		this.id = id;
		this.name = name;
	}

	public String getId()
	{
		return id;
	}
	public void setId(String id)
	{
		this.id = id;
	}
	public String getName()
	{
		return name;
	}
	public void setName(String name)
	{
		this.name = name;
	}
}//
```

```
public String toString() {
   return "MyMember [id=" + id + ", name=" + name+ "]";
}
```

예제2

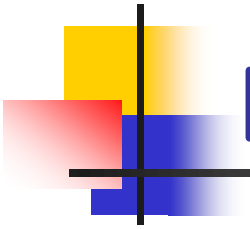
```
class ArrayListTest2{
	public static void main(String[] args){
		ArrayList<MyMember> list=new ArrayList<MyMember>();

		MyMember m = new MyMember("hong","홍길동");
		list.add(m);
		list.add(new MyMember("kim","김연아"));
		list.add(new MyMember("lim","윤아"));

		System.out.println("-----------------------------");
		System.out.println("아이디₩t이름");
		System.out.println("-----------------------------");
		for(int i=0; i<list.size(); i++){
			MyMember mem = list.get(i);
			System.out.println(mem.getId() + "₩t" + mem.getName());
		}

		System.out.println("----------------------₩n");
		for(MyMember mem: list){
			System.out.println(mem);
		}

	}//main
}
```

```
<회원 리스트>

----------------------
아이디  이름
----------------------
hong    홍길동
kim     김연아
lim     윤아
----------------------

MyMember [id=hong, name=홍길동]
MyMember [id=kim, name=김연아]
MyMember [id=lim, name=윤아]
```



실습

- Product 클래스를 정의하고
  - 멤버변수
    - 상품코드(code), 상품명(pdName), 상품가격(price)
  - 생성자, getter/setter, toStirng() 오버라이딩
- ArrayList에 Product 클래스를 3개 저장한 후, 읽어와서 toString() 을 이용하여 화면 출력하기



HashSet

- HashSet
  - Set<E> 인터페이스를 구현하는 컬렉션 클래스
- 특성
  - 데이터의 저장 순서를 유지하지 않는다.
  - 데이터의 중복 저장을 허용하지 않는다.
    - 집합의 특성

예제

```
저장된 데이터 수: 3
Third
Second
First
```

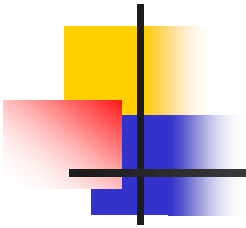
- 두번 저장된 "First"가 반복해서 저장되지 않음
- 문자열의 출력순서가 저장된 순서에 상관없이 출력됨

```
import java.util.Iterator;
import java.util.HashSet;

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

		hSet.add("First");
		hSet.add("Second");
		hSet.add("Third");
		hSet.add("First");

		System.out.println("저장된 데이터 수: "+hSet.size());

		Iterator<String> itr=hSet.iterator();
		while(itr.hasNext()){
			String str = itr.next();
			System.out.println(str);
		}//while

		for (String str : hSet)
		{
			System.out.println(str);
		}//for each
	}
}
```

Enumeration, Iterator

- Enumeration, Iterator
  - 컬렉션에 저장된 요소를 접근하는데 사용되는 인터페이스
  - 저장된 데이터 전부를 참조할 때 유용함
- Iterator
  - 컬렉션 프레임웍에서는 컬렉션에 저장된 각 요소에 접근하는 기능을 가진 Iterator 인터페이스를 정의
  - Collection 인터페이스에는 Iterator(Iterator를 구현한 클래스의 인스턴스)를 반환하는 iterator()를 정의
  - 컬렉션 클래스에 대해 iterator()를 호출하여 Iterator를 얻은 다음 반복문을 사용해서 컬렉션 클래스의 요소들을 읽어 올 수 있음

```
Iterator<E>  iterator()
```

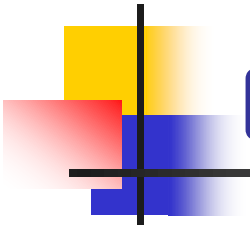


Iterator

```
Set set = new HashSet();
Iterator<E> it = set.iterator();
while(it.hasNext())
{
        System.out.println(it.next());
}
```

메서드	설 명
boolean hasNext()	읽어 올 요소가 남아있는지 확인함. 있으면 true, 없으면 false를 반환함
Object next()	다음 요소를 읽어옴. next()를 호출하기 전에 hasNext()를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전함

Iterator 인터페이스의 메서드



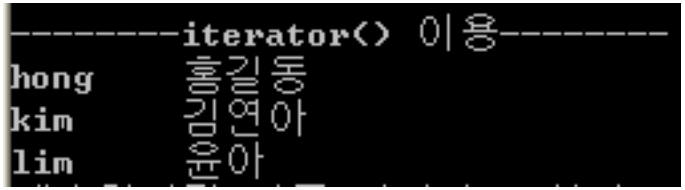
Enumeration

- Enumeration
  - 컬렉션 프레임워크가 만들어지기 이전에 사용하던 것
  - Iterator의 구버전

메서드	설 명
boolean hasMoreElements()	읽어 올 요소가 남아있는지 확인함. 있으면 true, 없으면 false를 반환함
Object nextElement()	다음 요소를 읽어옴. nextElement()를 호출하기 전에 hasMoreElement()를 호출해서 읽어 올 요소가 있는지 확인하는 것이 안전함

Enumeration 인터페이스의 메서드

예제




```
class ArrayListTest2{
	public static void main(String[] args)	{
		ArrayList<MyMember> list=new ArrayList<MyMember>();

		MyMember m = new MyMember("hong","홍길동");
		list.add(m);
		list.add(new MyMember("kim","김연아"));
		list.add(new MyMember("lim","윤아"));

		//Iterator 이용 - ArrayList에 저장된 객체들을 한꺼번에 꺼내오기
		System.out.println("₩n--------iterator() 이용--------");
		Iterator<MyMember> it=list.iterator();
		while(it.hasNext()){
			MyMember mem =it.next();
			System.out.println(mem.getId() + "₩t" + mem.getName());
		}
	}//main
}
```

예제2

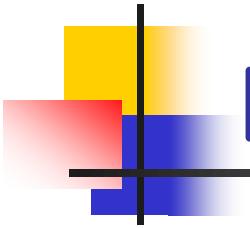
```
아이디: hong, 이름: 홍길동
아이디: kim, 이름: 김연아
아이디: son, 이름: 손연재
```

```
class EnumerationTest
{
    public static void main(String[] args)
    {
        Vector<MyMember> v=new Vector<MyMember>();
        MyMember m1=new MyMember("hong", "홍길동");
        v.add(m1);
        v.add(new MyMember("kim","김연아"));
        v.add(new MyMember("son","손연재"));

        Enumeration<MyMember> en =v.elements();
        while(en.hasMoreElements()){
            MyMember m=en.nextElement();
            System.out.print("아이디: "+m.getId());
            System.out.println(", 이름: "+m.getName());
        }//while
    }
}
```

실습

- 규칙이 있는 4개의 실수를 컬렉션에 저장하고 출력하기
  - 1. 배열 이용
  - 2. ArrayList 컬렉션 이용
    - 입력, 출력 모두 for문
  - 3. HashSet 이용
    - for문 이용해서 입력
    - iterator() 이용하여 출력

```
--------ArrayList 이용-------
1.5
2.5
3.5
4.5

--------배열 이용-------
1.5
2.5
3.5
4.5

--------HashSet iterator() 이용--------
1.5
3.5
4.5
2.5

--------HashSet foreach 이용-------
1.5
3.5
4.5
2.5
```



HashMap

- HashMap
  - Map<K, V> 인터페이스를 구현하는 컬렉션 클래스
  - 데이터 저장방식 : key-value 방식
    - key : 데이터를 찾는 열쇠(이름)
    - value : 찾고자 하는 실질적인 데이터
  - 키와 값을 한 쌍으로 하여 저장하는 자료 구조
    - value를 저장할 때, 이를 찾는데 사용되는 key를 함께 저장하는 구조
  - 특성
    - value에 상관없이 중복된 key의 저장은 불가능하다
    - value는 같더라도 key가 다르면 둘 이상의 데이터 저장도 가능하다.

예제

```
10번 학생: 아이유
7번 학생: 김연아
3번 학생: 홍길동
7번 학생: null


key: 3, value: 홍길동
key: 10, value: 아이유
```

```
import java.util.HashMap;
class IntroHashMap
{
    public static void main(String[] args)
    {
        HashMap<Integer, String> hMap=new HashMap<Integer, String>();

        hMap.put(new Integer(3), "홍길동");
        hMap.put(7, "김연아");
        hMap.put(10, "아이유");

        System.out.println("10번 학생: "+hMap.get(new Integer(10)));
        System.out.println("7번 학생: "+hMap.get(7));
        System.out.println("3번 학생: "+hMap.get(3));

        hMap.remove(7);                    // 7번 학생 전학 감
        System.out.println("7번 학생: "+hMap.get(7)+ "₩n₩n");

        Iterator<Integer> keyIter = hMap.keySet().iterator();
        while( keyIter.hasNext() ) {
            int key = keyIter.next();
            System.out.println("key: "+ key + ", value: "+hMap.get(key));
        }
    }
}
```

실습

```
boy => 소년
girl => 소녀
school => 학교
```

- HashMap 을 이용하여 영어사전 만들기
  - key : 영어단어
  - value : 단어의 뜻

실습

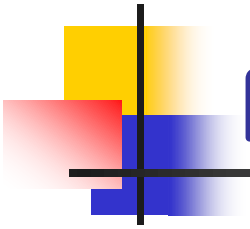
- 알파벳을 각 컬렉션에 넣고, 출력하기
  - for문 이용하여 저장하고, 출력
    - ArrayList – 출력시 for
    - HashMap – 출력시 Iterator, for
    - HashSet 이용 – 출력시 Iterator

```
-----ArrayList 이용-----
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

-----HashMap 이용-----
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

--------HashSet 이용--------
D E F G A B C L M N O H I J K U T W V Q P S R Y X Z
```

실습 – 컬렉션(ArrayList) 이용

```
국어, 영어, 수학 점수를 입력하세요
70
80
90
국어, 영어, 수학 점수: 70 80 90
총점=240, 평균=80.00
```

- 성적 클래스
  - 필드
    - 총점
    - 평균
- getter/setter – 총점, 평균
- 메서드
  - 총점 구하는 메서드
    - 매개변수가 ArrayList<Integer> (과목)
    - 총점 필드에 결과값 넣기
  - 평균 구하여 평균 필드에 결과값 넣기
- main()에서
  - 사용자로부터 국어, 영어, 수학 점수를 입력 받는데, ArrayList에 입력 받은 점수를 넣고, 총점, 평균 메서드 호출한 후, 화면 출력
    - 화면 출력시 세 과목의 점수는 ArrayList의 내용을 출력
    - 총점, 평균은 getter/setter를 이용하여 화면 출력



Properties

- Properties – HashMap의 구버전인 Hashtable을 상속받아 구현한 것
- Hashtable – 키와 값을 (Object, Object)의 형태로 저장
- Properties – **(String, String)의 형태**로 저장하는 보다 단순화된 컬렉션 클래스
  - 주로 **어플리케이션의 환경설정과 관련된 속성(property)을 저장**하는데 사용되며 데이터를 파일로부터 읽고 쓰는 편리한 기능을 제공함
  - 간단한 입출력은 Properties 를 활용하면 몇 줄의 코드로 쉽게 해결될 수 있음

메서드	설명
Properties()	Properties()객체를 생성한다.
Properties(Properties defaults)	지정된 Properties에 저장된 목록을 가진Properties()객체를 생성한다.
String getProperty(String key)	지정된 키(key)의 값(value)을 반환한다.
String getProperty(String key, String defaultValue)	지정된 키(key)의 값(value)을 반환한다. 키를 못찾으면 defaultValue를 반환한다.
void list(PrintStream out)	지정된 PrintStream에 저장된 목록을 출력한다.
void list(PrintWriter out)	지정된 PrintWriter에 저장된 목록을 출력한다.
void load(InputStream inStream)	지정된 InputStream으로부터 목록을 읽어서 저장한다.
void loadFromXML(InputStream in) *	지정된 InputStream으로부터 XML문서를 읽어서, XML문서에 저장된 목록을 읽어다 담는다.(load & store)
Enumeration propertyNames()	목록의 모든 키(key)가 담긴 Enumeration을 반환한다.
void save(OutputStream out, String header)	deprecated되었으므로 store()를 사용하자.
Object setProperty(String key, String value)	지정된 키와 값을 저장한다. 이미 존재하는 키(key)은 새로운 값(value)로 바꾼다.
void store(OutputStream out, String header)	저장된 목록을 지정된 출력스트림에 출력(저장)한다. header는 목록에 대한 설명(주석)으로 저장된다.
void storeToXML(OutputStream os, String comment)*	저장된 목록을 지정된 출력스트림에 XML문서로 출력(저장)한다. comment는 목록에 대한 설명(주석)으로 저장된다.
void storeToXML(OutputStream os, String comment, String encoding) *	저장된 목록을 지정된 출력스트림에 해당 인코딩의 XML문서로 출력(저장)한다. comment는 목록에 대한 설명(주석)으로 저장된다.

```
capacity=10
size=10
timeout=30
language=kr

size=20
capacity=10
loadfactor=0.75
{capacity=10, size=20, timeout=30, language=kr}
-- listing properties --
capacity=10
size=20
timeout=30
language=kr
```

```
import java.util.*;
import java.io.*;
class PropertiesEx1 {
	public static void main(String[] args) throws Exception {
		Properties prop = new Properties();

		// prop에 키와 값(key, value)을 저장한다.
		prop.setProperty("timeout","30");
		prop.setProperty("language","kr");
		prop.setProperty("size","10");
		prop.setProperty("capacity","10");

		// prop에 저장된 요소들을 Enumeration을 이용해서 출력한다.
		Enumeration e = prop.propertyNames();

		while(e.hasMoreElements()) {
			String element = (String)e.nextElement();
			System.out.println(element + "="+ prop.getProperty(element));
		}
		System.out.println();
		prop.setProperty("size","20");		// size의 값을 20으로 변경한다.
		System.out.println("size=" + prop.getProperty("size"));
		System.out.println("capacity=" + prop.getProperty("capacity", "20"));
		System.out.println("loadfactor=" + prop.getProperty("loadfactor", "0.75"));

		System.out.println(prop);		// prop에 저장된 요소들을 출력한다.
		prop.list(System.out);	// prop에 저장된 요소들을 화면(System.out)에 출력한다.
	}
}
```

예제 1

- Properties의 기본적인 메서드를 이용해서 저장하고 읽어오고 출력하는 예제
- setProperty() - 데이터를 저장하는데 사용되는 메서드
  - 단순히 Hashtable의 put 메서드를 호출
  - 기존에 같은 키로 저장된 값이 있는 경우 그 값을 Object 타입으로 반환하며, 그렇지 않을 때는 null을 반환

```
public Object setProperty(String key, String value)
```

- getProperty() - Properties에 저장된 값을 읽어오는 일을 하는데, 만일 읽어오려는 키가 존재하지 않으면 지정된 기본값(defaultValue)을 반환함

```
public String getProperty(String key)
public String getProperty(String key, String defaultValue)
```

예제 1

- Properties
  - Hashtable을 상속받아 구현한 것이라 Map의 특성상 저장순서를 유지하지 않는다
    - 예제의 결과에 출력된 순서가 저장순서와는 무관
  - 컬렉션 프레임워크 이전의 구버전이므로 Iterator가 아닌 Enumeration을 사용함
  - list() 메서드를 이용하면 Properties에 저장된 모든 데이터를 화면 또는 파일에 편리하게 출력할 수 있음

```
public void list(PrintStream out)
public void list(PrintWriter out)
```

```
이름 :Hong gil dong
최대값 :29
최소값 :3
합계 :74
평균 :9.25
```

```
import java.io.*;
import java.util.*;
class PropertiesEx2{
	public static void main(String[] args) {
		// commandline에서 inputfile을 지정해주지 않으면 프로그램을 종료한다.
		if(args.length != 1) {
			System.out.println("USAGE: java PropertiesEx2 INPUTFILENAME");
			System.exit(0);
		}

		Properties prop = new Properties();

		String inputFile = args[0];

		try {
			prop.load(new FileInputStream(inputFile));
		} catch(IOException e) {
			System.out.println("지정된 파일을 찾을 수 없습니다.");
			System.exit(0);
		}

		String name = prop.getProperty("name");
		String[] data = prop.getProperty("data").split(",");
		int max = 0;
		int min = 0;
		int sum = 0;
```

input.txt

```
# 이것은 주석입니다
# 여러 줄도 가능합니다
name=Hong gil dong
data=9,3,6,29,3,7,6,11
```

```
        for(int i=0; i < data.length; i++) {
                int intValue = Integer.parseInt(data[i]);
                if (i==0) max = min = intValue;

                if (max < intValue) {
                        max = intValue;
                } else if (min > intValue) {
                        min = intValue;
                }

                sum += intValue;
        }

        double avg = (double)sum/data.length;

        System.out.println("이름 :" + name);
        System.out.println("최대값 :" + max);
        System.out.println("최소값 :" + min);
        System.out.println("합계 :" + sum);
        System.out.println("평균 :" + Math.round(avg*100)/100.0);          }
}
```

예제2

- 외부파일(input.txt)로부터 데이터를 입력 받아서 계산결과를 보여주는 예제
- 외부 파일의 형식은 라인단위로 키와 값이 '='로 연결된 형태이어야 하며 주석라인은 첫 번째 문자가 '#'이어야 함
- 정해진 규칙대로만 파일을 작성하면 load() 메서드를 호출하는 것만으로 쉽게 데이터를 읽어 올 수 있음
- 인코딩 문제로 한글이 깨질 수 있기 때문에 한글을 입력 받으려면 아래와 같이 코드를 변경해야 함

```
String name = prop.getProperty("name");
try{
        name = new String(name.getBytes("8859_1"), "EUC-KR");
}catch (Exception e){}
```

파일로부터 읽어온 데이터의 인코딩을 라틴문자 집합(8859_1)에서 한글 완성형(euc-kr 또는 KSC5601)으로 변환해주는 과정을 추가

예제3

Properties에 저장된 데이터를 store(), storeToXML()을 이용해서 파일로 저장하는 예제

```
import java.util.*;
import java.io.*;
class PropertiesEx3{
	public static void main(String[] args){
		Properties prop = new Properties();

		prop.setProperty("timeout","30");
		prop.setProperty("language","한글");
		prop.setProperty("size","10");
		prop.setProperty("capacity","10");

		try {
			prop.store(new FileOutputStream("output.txt"), "Properties Example");
			prop.storeToXML(new FileOutputStream("output.xml"), "Properties Example");
		} catch(IOException e) {
			e.printStackTrace();
		}
	}
}
```

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
<comment>Properties Example</comment>
<entry key="capacity">10</entry>
<entry key="size">10</entry>
<entry key="timeout">30</entry>
<entry key="language">한글</entry>
</properties>
```

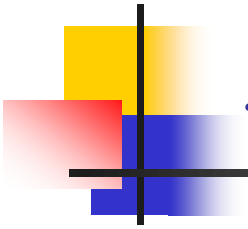
```
#Properties Example
#Sun Mar 08 20:57:08 KST 2015
capacity=10
size=10
timeout=30
language=\uD55C\uAE00
```

예제4

• 시스템 속성을 가져오는 예제
• System 클래스의 getProperties()를 호출하면 시스템과 관련된 속성이 저장된 Properties를 가져올 수 있다
• 이 중에서 원하는 속성을 getProperty()를 통해 얻을 수 있다

```
import java.util.*;

class PropertiesEx4 {
	public static void main(String[] args)
	{
		Properties sysProp = System.getProperties();
		System.out.println("java.version :" + sysProp.getProperty("java.version"));
		System.out.println("user.language :" + sysProp.getProperty("user.language"));
		sysProp.list(System.out);
	}
}
```

```
java.version :1.7.0_75
user.languag :ko
-- listing properties --
java.runtime.name=Java(TM) SE Runtime Environment
sun.boot.library.path=C:₩java₩jdk1.7.0_75₩jre₩bin
java.vm.version=24.75-b04
java.vm.vendor=Oracle Corporation
java.vendor.url=http://java.oracle.com/
path.separator=;
java.vm.name=Java HotSpot(TM) 64-Bit Server VM
file.encoding.pkg=sun.io
user.script=
user.country=KR
sun.java.launcher=SUN_STANDARD
sun.os.patch.level=
java.vm.specification.name=Java Virtual Machine Specification
user.dir=D:₩yang₩강의안₩KH₩웹개발자1₩java₩0309₩lec11
java.runtime.version=1.7.0_75-b13
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
java.endorsed.dirs=C:₩java₩jdk1.7.0_75₩jre₩lib₩endorsed
os.arch=amd64
java.io.tmpdir=C:₩Users₩yang-hp1₩AppData₩Local₩Temp₩
line.separator=

java.vm.specification.vendor=Oracle Corporation
```



동기화(Synchronization)

- 멀티 쓰레드 프로그래밍에서는 하나의 객체를 여러 쓰레드가 동시에 접근할 수 있기 때문에 데이터의 일관성을 유지하기 위해서는 동기화가 필요함
- Vector, Hashtable과 같은 구버전(JDK 1.2 이전) 의 클래스들은 자체적으로 동기화 처리가 되어 있는데, 멀티쓰레드 프로그래밍이 아닌 경우에는 불필요한 기능이 되어 성능을 떨어뜨림
- 새로 추가된 ArrayList, HashMap 과 같은 컬렉션은 동기화를 자체적으로 처리하지 않고, 필요한 경우에만 Collections 클래스의 동기화 메서드를 이용해서 동기화 처리가 가능하도록 변경하였음

동기화(Synchronization)

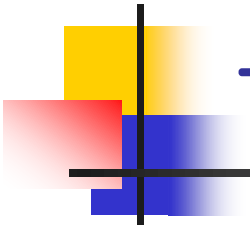
- Collections 클래스에는 다음과 같은 동기화 메서드를 제공
  - 동기화가 필요할 때 해당하는 것을 사용하면 됨

```
static Collection synchronizedCollection(Collection c)
static List synchronizedList(List list)
static Map synchronizedMap(Map m)
static Set synchronizedSet(Set s)
```

  - 사용하는 방법

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

```
List<Integer> list = Collections.synchronizedList(new ArrayList<Integer>());
list.add(50);
list.add(80);

for (int n : list) System.out.println(n);
```



TreeSet<E> 클래스

- Set<E> 인터페이스를 구현하는 TreeSet<E> 클래스
  - TreeSet<E> 클래스는 트리(Tree)라는 자료구조를 기반으로 구현
  - **트리 – 데이터를 정렬된 상태로 저장**하는 자료구조
  - TreeSet<E> 클래스도 데이터를 정렬된 상태로 유지함 (저장 순서를 유지하는 것과 다름)

예제

```
저장된 데이터 수: 4
1
2
3
4
```

정렬 기준 – 숫자의 크고 작음을 기준으로 정렬이 이뤄졌음

```
import java.util.Iterator;
import java.util.TreeSet;

class SortTreeSet{
	public static void main(String[] args){
		TreeSet<Integer> sTree=new TreeSet<Integer>();
		sTree.add(1);
		sTree.add(2);
		sTree.add(4);
		sTree.add(3);
		sTree.add(2);  //저장될때마다 데이터가 정렬됨

		System.out.println("저장된 데이터 수: "+sTree.size());

		Iterator<Integer> itr=sTree.iterator();  //Iterator는 정렬된 데이터를 오름차순으로 참조함
		while(itr.hasNext())
		{
			System.out.println(itr.next());
		}//while
	}
}
```

예제2

```
[7, 10, 14, 23, 27, 33]
```

public final class Integer
  extends Number
  implements **Comparable<Integer>**

```
import java.util.*;
class TreeSetTest2{
	public static void main(String[] args) {
		Set<Integer> set = new TreeSet<Integer>();

		while (set.size() < 6) {
			int num = (int)(Math.random()*45) + 1;  //1~45
			set.add(num);
		}
		System.out.println(set);
	}
}
```

```
TreeSet<String> sTree=new TreeSet<String>();
sTree.add("홍길동");
sTree.add("김연아");
sTree.add("손연재");
sTree.add("홍길동");

Iterator<String> itr=sTree.iterator();
while(itr.hasNext()){
		String s = itr.next();
		System.out.println(s);
}//while
```

```
김연아
손연재
홍길동
```

문자열의 경우 정렬순서는
문자의 코드값이 기준이 됨

예제3

```
import java.util.Iterator;
import java.util.TreeSet;
class Person {
    private String name;
    private int age;
    public Person(String name, int age) {
        this.name=name;
        this.age=age;
    }
    public String toString()
    {
     return "[Person : name=" + name +", age="+age+"]";
    }
}
class ComparablePerson {
     public static void main(String[] args){
            TreeSet<Person> sTree=new TreeSet<Person>();
            sTree.add(new Person("홍길동", 20));  //실행 에러
            sTree.add(new Person("김연아", 17));
            sTree.add(new Person("손연재", 31));

            Iterator<Person> itr=sTree.iterator();
            while(itr.hasNext()){
                        Person p = itr.next();
                        System.out.println(p);
            }//while
     }
```

- 실행 중 예외 발생함

```
Exception in thread "main"
java.lang.ClassCastException: Person cannot be
cast to java.lang.Comparable
```

인스턴스의 비교 기준을 정의하는 Comparable<T> 인터페이스의 구현

```
class Person
{
        private String name;
        private int age;
        Person(String name, int age)
        {
                this.name=name;
                this.age=age;
        }
}
```

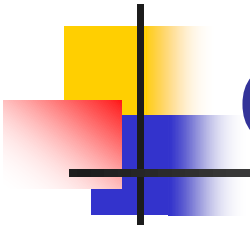
```
interface Comparable<T>
{
        int compareTo(T  obj);
}
```

- Person 클래스의 인스턴스가 저장 대상이라면
  - Person p1 = new Person("홍길동", 20);
  - Person p2 = new Person("김연아", 17);
  - Person p3 = new Person("손연재", 31);
- 이 셋을 정렬하고자 한다면 **정렬의 기준은** 무엇일까?
  - 인스턴스의 정렬기준은 개발자가 직접 정의해야 함
  - 자바에서는 Comparable<T> 인터페이스 구현을 통해 정렬의 기준을 개발자가 직접 정의할 것을 요구함

Comparable<T> 인터페이스의 compareTo() 메서드는 다음과 같이 정의되어야 함

[1] 인자로 전달된 obj가 작다면 양의 정수를 반환해라
[2] 인자로 전달된 obj가 크다면 음의 정수를 반환
[3] 인자로 전달된 obj가 같다면 0 을 반환해라

compareTo() 메서드
- 매개변수로 넘어가는 객체와 현재 객체가 같은지를 비교하는 데 사용됨
-리턴 타입은 int
- 같으면 0, 순서 상으로 앞에 있으면 -1, 뒤에 있으면 1 을 리턴함
- 객체의 순서를 처리할 때 유용하게 사용됨



Comparable<T>

- Person 클래스가 Comparable<T> 인터페이스를 구현한다면, Person 인스턴스를 TreeSet<E>에 저장될 수 있게 됨
  - TreeSet<E> 도 compareTo() 메서드의 호출 결과를 참조하여 정렬하기 때문에, TreeSet<E>에 저장되는 인스턴스는 반드시 Comparable<T> 인터페이스를 구현하고 있어야 함

Integer 인스턴스를 TreeSet<E>에 저장할 수 있었던 것은 Integer 클래스가 Comparable<T> 인터페이스를 구현하고 있기 때문에 가능했음

```
public final class Integer
                extends Number
                implements Comparable<Integer>
```

```
[Person : name=김연아, age=17]
[Person : name=홍길동, age=20]
[Person : name=손연재, age=31]
```

Comparable<T>

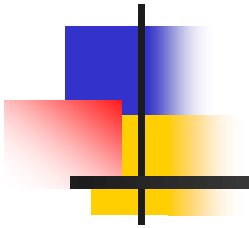
- Person 클래스의 정렬기준을 나이로 정하고, Comparable<T> 를 구현해보자

```
import java.util.Iterator;
import java.util.TreeSet;

class Person implements Comparable<Person>
{
    private String name;
    private int age;

    public Person(String name, int age)
    {
        this.name=name;
        this.age=age;
    }
    public String toString()
    {
            return "[Person : name=" + name +", age="+age+"]";
    }
```

```
    public int compareTo(Person p)
    {
        if(age>p.age)    //인자로 전달된 p의 나이가 작다면 양수 반환
            return 1;
        else if(age<p.age) //인자로 전달된 p의 나이가 크다면 음수 반환
            return -1;
        else
            return 0;        //인자로 전달된 p의 나이와 같다면 0 반환
    }
}

class ComparablePerson
{
        public static void main(String[] args)
        {
                TreeSet<Person> sTree=new TreeSet<Person>();
                sTree.add(new Person("홍길동", 20));
                sTree.add(new Person("김연아", 17));
                sTree.add(new Person("손연재", 31));

                Iterator<Person> itr=sTree.iterator();
                while(itr.hasNext()){
                        Person p = itr.next();
                        System.out.println(p);
                }//while
        }
}
```

Person 인스턴스가 TreeSet<E>에 저장될 때, 나이를 기준으로 정렬됨

- Person 인스턴스의 크고 작음을 비교할 수 있게 됨
- TreeSet<E> 는 Person인스턴스가 저장될 때마다 기존에 저장된 인스턴스와의 비교를 위해서 compareTo() 메서드를 빈번히 호출하여, 이때 반환되는 값을 기반으로 정렬을 진행



Comparator<T>

Comparator<T> 인터페이스를 기반으로 TreeSet<E>의 정렬기준 제시하기

- 예제 – 문자열을 사전편찬 순서가 아닌, 길이 순으로 정렬해서 TreeSet<String> 에 저장해보자
  - Comparator<T> 인터페이스 이용

```
interface Comparator<T>
{
        int compare(T  obj1, T obj2);
        boolean equals(Object obj);
}
```

  - compare()메서드만 구현하면 됨
    - obj1 이 크면 양수를,
    - obj2 가 크면 음수를
    - obj1과 obj2 가 같으면 0 을 반환하면 됨

예제

```
Dog
Apple
Orange
Individual
```

```
import java.util.TreeSet;
import java.util.Iterator;
import java.util.Comparator;

class StrLenComparator implements Comparator<String>
{
    public int compare(String str1, String str2)
    {
        if(str1.length()> str2.length())
            return 1;
        else if(str1.length()< str2.length())
            return -1;
        else
            return 0;

        /*
         * return str1.length()-str2.length();
         */
    }
}
```

예제

```
Dog
Apple
Orange
Individual
```

```
class IntroComparator
{
    public static void main(String[] args)
    {
        TreeSet<String> tSet=new TreeSet<String>(new StrLenComparator());
        tSet.add("Orange");
        tSet.add("Apple");
        tSet.add("Dog");
        tSet.add("Individual");

        Iterator<String> itr=tSet.iterator();
        while(itr.hasNext())
            System.out.println(itr.next());
    }
}
```

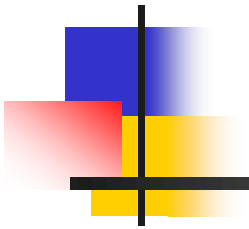
TreeSet(Comparator<? super E> comparator)

```
[10, 35, 45, 50, 65, 85, 95, 100]
50보다 작은 값 :[10, 35, 45]
50보다 큰 값 :[50, 65, 85, 95, 100]
범위 검색(range search) : from 60 to 95
result1 : [65, 85]
```

```
import java.util.*;

class TreeSetEx2
{
    public static void main(String[] args)
    {
        TreeSet<Integer> set = new TreeSet<Integer>();
        int[] score = {85, 95, 50, 35, 45, 65, 10, 100};

        for(int i=0; i < score.length; i++){
            set.add(score[i]);
        }

        System.out.println(set);
        System.out.println("50보다 작은 값 :" + set.headSet(50));
        System.out.println("50보다 큰 값 :" + set.tailSet(50));

        int from = 60;
        int to = 95;
        System.out.println("범위 검색(range search) : from " + from  +" to "+ to);
        System.out.println("result1 : " + set.subSet(from, to));
    }
}
```

- headSet() – TreeSet 에 저장된 객체 중 지정된 기준값 보다 작은 값의 객체들을 얻을 수 있다
- tailSet() – 큰 값의 객체들을 얻음(같은 값도)

- subSet() – 범위 검색할 때 시작범위는 포함되지만, 끝 범위는 포함되지 않음
  문자열의 경우 정렬순서는 문자의 코드값이 기준이 됨



예제- 여러 종류의 객체를 배열이나 컬렉션으로 다루기

여러 종류의 객체를 배열이나 컬렉션으로 다루기

- 멤버 변수 추가 – Buyer 클래스에 구입한 제품을 저장하기 위한 Product 컬렉션(ArrayList)을 추가하자
  - buy() 메서드 – 물건을 구입하면, 컬렉션에 저장되도록 한다
    - => 구입한 제품을 하나의 컬렉션으로 간단하게 다룰 수 있다
  - ArrayList<Product> item = new ArrayList<Product>();
- 메서드 추가
  - refund(Product p)
    - **구입한 물건을 다시 반환**할 수 있도록 refund(Product p)를 추가하자
    - 이 메서드가 호출되면 구입 물품이 저장되어 있는 item에서 해당 제품을 제거한다
  - summary()
    - 구매한 물품에 대한 정보를 요약해서 보여 준다.
    - 반복문을 이용해서 구입한 물품의 총 가격과 목록을 만든다.

```
class Buyer {           // 고객, 물건을 사는 사람
    private int money = 1000;                   // 소유금액-1000만원
    private int bonusPoint = 0;                 // 보너스점수
    private ArrayList<Product> item = new ArrayList<Product>();// 구입한 제품을 저장하는데 사용될 ArrayList객체
...
    public void buy(Product p) {
        if(money < p.getPrice()) {
            System.out.println("잔액이 부족하여 물건을 살수 없습니다.");
            return;
        }
        money -= p.getPrice();              // 가진 돈에서 구입한 제품의 가격을 뺀다.
        bonusPoint += p.getBonusPoint();    // 제품의 보너스 점수를 추가한다.
        item.add(p);                        // 구입한 제품을 ArrayList에 저장한다.
        System.out.println(p + "을/를 구입하셨습니다.");
    }

    public void refund(Product p) {// 구입한 제품을 환불한다.
        if(item.remove(p)) {    // 제품을 ArrayList에서 제거한다.
            money += p.getPrice();
            bonusPoint -= p.getBonusPoint();
            System.out.println(p + "을/를 반품하셨습니다.");
        } else {                            //  제거에 실패한 경우
            System.out.println("구입하신 제품 중 해당 제품이 없습니다.");
        }
    }
```

```
Tv을/를 구입하셨습니다.
Computer을/를 구입하셨습니다.
구입하신 물품의 총금액은 300만원입니다.
구입하신 제품은 Tv, Computer입니다.

Computer을/를 반품하셨습니다.
구입하신 물품의 총금액은 100만원입니다.
구입하신 제품은 Tv입니다.
```

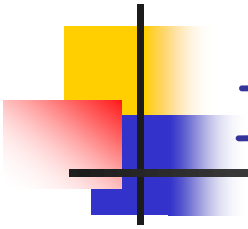
public boolean remove(Object o)

```
    public void summary() {     // 구매한 물품에 대한 정보를 요약해서 보여준다.
        int sum = 0;                // 구입한 물품의 가격합계
        String itemList ="";        // 구입한 물품목록
        // 반복문을 이용해서 구입한 물품의 총 가격과 목록을 만든다.
        if(item.isEmpty()) {        // ArrayList가 비어있는지 확인한다.
            System.out.println("구입하신 제품이 없습니다.");
            return;
        }
        //ArrayList의 i번째에 있는 객체를 얻어 온다.
        for(int i=0; i<item.size();i++) {
            Product p = item.get(i);
            sum += p.getPrice();
            itemList += (i==0) ? "" + p : ", " + p;
        }
        System.out.println("구입하신 물품의 총금액은 " + sum + "만원입니다.");
        System.out.println("구입하신 제품은 " + itemList + "입니다.");
    }
}
class PolyArgumentTest3 {
    public static void main(String args[]) {
        Buyer b = new Buyer();
        Tv tv = new Tv();
        Computer com = new Computer();
        b.buy(tv);
        b.buy(com);
        b.summary();
        System.out.println();
        b.refund(com);
        b.summary();
    }
}
```

과제-여러 종류의 객체를 하나의 컬렉션으로 다루기

- 도형을 입력 받는다 (무한루프)
  - 그 도형은 원과 사각형 중 어느 것이어도 됨
  - 사용자가 원하는 도형을 입력할 수 있도록 함
- 입력 받는 도중에 사용자가 현재까지 입력된 도형을 보려는 경우 보여 주어야 함
- 언제든 프로그램은 종료될 수 있어야 함

```
1.원 2.사각형 3.보기 4.종료 ==> 1
반지름은 ? 10
1.원 2.사각형 3.보기 4.종료 ==> 2
가로는? 5
세로는? 6
1.원 2.사각형 3.보기 4.종료 ==> 1
반지름은 ? 20
1.원 2.사각형 3.보기 4.종료 ==> 3

-------------- 보기 ---------------
[Circle, r=10]
원의 둘레 : 63
[Rect, w=5, h=6]
사각형의 둘레 : 22
[Circle, r=20]
원의 둘레 : 126
-----------------------------------
1.원 2.사각형 3.보기 4.종료 ==> 4
프로그램을 종료합니다.
```



과제

- Shape 클래스-부모, 추상 클래스
  - 메서드
    - findGirth(); => 추상 메서드
- Circle 클래스 - 자식
  - 필드 : 반지름
  - 상수 : 3.14
  - 생성자
    - 반지름 초기화
  - 메서드
    - findGirth() – 원의 둘레 구한 후 출력 => 2*반지름*3.14
    - toString() 오버라이드
- Rect 클래스 - 자식
  - 필드 : 가로, 세로
  - 생성자
    - 가로, 세로값 초기화
  - 메서드
    - findGirth() – 사각형 둘레 구한 후 출력=> 2*(가로+세로)
    - toString() 오버라이드

과제-main()

- 1단계
  - main()에서 사용자로부터 원, 사각형, 종료 중 선택하게 하고,
  - 원을 선택하면 반지름을 입력 받아서 원의 객체 생성
  - 사각형을 선택하면 가로,세로를 입력 받아서 사각형의 객체 생성
  - 보기를 선택하면, 둘레를 구하는 메서드를 호출하여 모든 객체의 둘레 출력
  - 종료를 선택하면 메서드 종료(return)
  - 단, 다형성을 이용하여 구현할 것
- 2단계
  - 무한루프- 계속 반복하여 사용자로부터 입력 받을 것
- 3단계
  - ArrayList를 이용하여 부모ArrayList에 자식 객체들을 저장
  - ArrayList<Shape> alist = new ArrayList<Shape>();
    - 1. 원을 선택하면 Circle 객체 생성 후 alist에 add
    - 2. 사각형을 선택하면 Rect 객체 생성 후 alist에 add