JCF 활용

egyou@induk.ac.kr

학습 개요

- **학습 배경**
  - 동일한 유형의 자료들을 관리하는 경우 배열 객체를 이용한 처리는 빠른 참조가 가능하다는 장점을 제공한다.
  - 그러나 다양한 유형의 자료들을 관리해야 하는 경우 배열 객체로 처리하면 오류 발생이나 자원 낭비를 유발할 수 있다.
  - 이를 해결하기 위한 방법으로 다양한 컬렉션 객체들이 사용되어 왔다.

학습 개요

- **학습 목표**
  - 실습을 통해 자바 컬렉션 프레임워크 기반으로 다양한 유형의 자료들을 효율적으로 다룰 수 있는 능력을 배양한다.
- **주요 용어**
  - Data Structures, Implementations, Algorithms, Generics, Framework

학습 목차

- **JCF를 구성하는 인터페이스들, 구현요소들과 알고리즘에 대한 실습을 한다.**
  - List, AbstractList, AbstractSequentialList, ArrayList, Vector, Stack, LinkedList
  - HashMap, HashTable, TreeMap, LinkedHashMap
  - Queue, ArrayQueue, LinkedList
  - HashSet, TreeSet, LinkedHashSet

실습

- 참고
  - http://docs.oracle.com/javase/tutorial/collections/index.html
  - http://beginnersbook.com/category/java-collections/
  - https://www.tutorialspoint.com/java/util/

java.util.ArrayList

주요 특징

- 순서를 가지는 요소들의 모임
- 중복을 허용하고, 인덱스를 이용한 접근을 허용함
- 저장되는 요소들의 개수에 따라 자동적으로 크기가 변경됨
- 추가 연산을 실행하는 경우 오버헤드가 발생함 vs. LinkedList
- 스레드 안전 하지 않음(thread-unsafe) vs. Vector

상속 관계

- public class ArrayList<E>
  extends AbstractList<E>
  implements LIst<E>, RandomAccess, Cloneable, Serializable

java.util.ArrayList

- **주요 메소드**
  - public boolean add(Element E) : 리스트의 마지막에 지정한 요소를 추가
  - public E set(int index, Element E) : 리스트의 지정한 위치에 지정한 요소를 추가
  - public E get(int index) : 리스트의 지정한 위치에 있는 요소를 반환
  - public boolean remove(Object o) : 리스트에서 지정한 요소가 첫 번째로 발견되는 위치의 요소를 제거
  - public E remove(int index) : 리스트의 지정한 위치에 있는 요소를 제거
    - IndexOutOfBoundsException - if the index is out of range (index < 0 || index >= size())

ArrayListLoopTest

```java
package iducs.java.jcf;
import java.util.*;
public class ArrayListLoopTest{
    public static void main(String[] args) {
        ArrayList<Integer> arrlist = new ArrayList<Integer>();
        arrlist.add(11);      arrlist.add(15);      arrlist.add(12);      arrlist.add(17);
        for (int counter = 0; counter < arrlist.size(); counter++) {
            System.out.print(arrlist.get(counter) + ", ");
        }
        System.out.print("\n");
        int count = 0;
        while (arrlist.size() > count) {     System.out.print(arrlist.get(count++) + ", ");}
        System.out.print("\n");
        // Enhanced Loop Statement 사용
        for (Integer num : arrlist) {   System.out.print(num + ", ");     }
        System.out.print("\n");
        // Iterator 사용
        Iterator iter = arrlist.iterator();
        while (iter.hasNext()) {        System.out.print(iter.next() + ", "); }

    }
}
```

ArrayListAddRemoveTest.java

```java
package iducs.java.jcf;

import java.util.*;
public class ArrayListAddRemoveTest {
    public static void main(String args[]) {
        ArrayList<String> arrayList = new ArrayList<String>();
        System.out.println("Initial size of al: " + arrayList.size());

        arrayList.add("C"); // index is 0
        arrayList.add("A");          arrayList.add("E");      arrayList.add("B");
        arrayList.add("D");      arrayList.add("F");      arrayList.add( index: 1,  element: "A2");
        System.out.println("추가 후 ArrayList 사이즈 : " + arrayList.size());
        System.out.println("ArrayList 내용 : " + arrayList);

        arrayList.remove( o: "F");
        arrayList.remove( index: 2);
        System.out.println("삭제 후 ArrayList 사이즈 : " + arrayList.size());
        System.out.println("ArrayList 내용 : " + arrayList);
    }
}
```

계속

- **크기 비교**
  - ArrayList 형 객체의 크기 : arrayList.size()
  - 배열형 객체의 크기 : arrayName.length
  - String형 객체의 크기 : strName.length()

ArrayListSizeTester.java

```java
package iducs.java.jcf;
import java.util.ArrayList;

public class ArrayListSizeTester {
    public static void main(String [] args)    {
        ArrayList<Integer> arrayList=new ArrayList<Integer>();
        System.out.println("최초 ArrayList 사이즈 : "+arrayList.size());
        arrayList.add(1);
        arrayList.add(2); // index 1
        arrayList.add(3);
        arrayList.add(4); // index  3
        arrayList.add(5);
        System.out.println("요소 추가 후 ArrayList 크기 : "+arrayList.size());
        arrayList.remove( index: 1); // 현재 ArrayList중 2번째 인덱스 요소 제거, ArrayList의 크기가 5에서 4로 작아짐
        arrayList.remove( index: 3); // 현재 ArrayList중 4번째 인덱스 요소 제거, 최초 ArrayList의 경우 마지막 요소
        System.out.println("요소 제거 후 ArrayList 크기: "+arrayList.size());
        System.out.println("ArrayList 요소 출력 : ");
        for(int num: arrayList){
            System.out.println(num);
        }
    }
}
```

java.util.Collections 활용

- **정의**
  - 컬렉션들에 대한 다양한 연산을 수행하거나 반환해주는 정적 메소드를 제공하는 클래스
- **상속 관계**
  - public class Collections extends Object

계속

- **주요 메소드**
  - public static <T extends Comparable<? super T>> void sort(List<T> list) : 대상 리스트를 요소들의 자연스러운 순서 매기기에 따라 오름차순 정렬
  - public static <T> void sort(List<T> list, Comparator<? super T> c) : 대상 리스트를 지정한 비교자에 의해 유도된 오름차순으로 정렬

ArrayListStringSort.java

```java
package iducs.java.jcf;

import java.util.*;
public class ArrayListStringSort {
    public static void main(String args[]){
        ArrayList<String> arrayList= new ArrayList<String>();
        arrayList.add("Korea");
        arrayList.add("Denmark");
        arrayList.add("France");
        arrayList.add("India");
        System.out.println("정렬 전 :");
        for(String countryName: arrayList){
            System.out.println(countryName);
        }
        Collections.sort(arrayList);
        System.out.println("정렬 후:");
        for(String countryName: arrayList){
            System.out.println(countryName);
        }
    }
}
```

ArrayListSortDescending.java

```java
package iducs.java.jcf;

import java.util.*;
public class ArrayListStringSortDesc  {
    public static void main(String args[]){
        ArrayList<String> arrayList = new ArrayList<String>();
        arrayList.add("Korea");
        arrayList.add("Denmark");
        arrayList.add("France");
        arrayList.add("India");
        System.out.println("정렬 전:");
        for(String countryName: arrayList){
            System.out.println(countryName);
        }
        Collections.sort(arrayList, Collections.reverseOrder());
        System.out.println("정렬 후 (내림차순):");
        for(String countryName: arrayList){
            System.out.println(countryName);
        }
    }
}
```

ArrayListIntergerSort.java

```java
package iducs.java.jcf;
import java.util.*;
public class ArrayListIntergerSort  {
    public static void main(String args[]){
        ArrayList<Integer> arraylist = new ArrayList<Integer>();
        arraylist.add(11);
        arraylist.add(5);
        arraylist.add(7);
        arraylist.add(3);
        System.out.println("정렬 전:");
        for(int counter: arraylist){
            System.out.println(counter);
        }
        Collections.sort(arraylist);
        System.out.println("정렬 후:");
        for(int counter: arraylist){
            System.out.println(counter);
        }
    }
}
```

비교를 위한 인터페이스

- **Comparable<T> interface**
  - int compareTo(T o) : 현재 객체 c와 o를 비교
    - negative : 오름 차순 정렬 시 c, o2 순서
    - zero : 오름 차순 정렬 시 c, o2 순서 유지
    - positive : 오름 차순 정렬 시 o2, c 순서
- **Comparator<T> interface**
  - int compare(T o1, T o2) : 순서를 정하기 위해서 o1과 o2를 비교
    - negative : 오름 차순 정렬 시 o1, o2 순서
    - zero : 오름 차순 정렬 시 o1, o2 순서 유지
    - positive : 오름 차순 정렬 시 o2, o1 순서
  - boolean equals(Object obj)

계속

- **활용**
  - comparable 인터페이스는 객체의 특정 속성을 기준으로 정렬이 가능하도록 하는 클래스 정의에 활용
  - comparator 인터페이스는 객체에 대하여 다양한 속성기준으로 정렬이 가능하도록

ComparableMember.java 1

```
public class ComparableMember<Object> implements Comparable<Object> {
	private String memberName;
	private int rollno;
	private int memberPhone;
	public ComparableMember(int rollno, String memberName, int memberPhone) {
		this.rollno = rollno;
		this.memberName = memberName;
		this.memberPhone = memberPhone;
	}
	public String getmemberName() {
		return memberName;
	}
	public void setmemberName(String memberName) {
		this.memberName = memberName;
	}
	public int getRollno() {
		return rollno;
	}
	public void setRollno(int rollno) {
		this.rollno = rollno;
	}
```

ComparableMember.java 2

```
    public int getmemberPhone() {
        return memberPhone;
    }
    public void setmemberPhone(int memberPhone) {
        this.memberPhone = memberPhone;
    }
    public String toString() {
        return "[ 번호 =" + rollno + ", name=" + memberName + ", 전화번호 =" + memberPhone + "]";
    }

    @Override
    public int compareTo(Object arg0) {
        int comparePhone = ((ComparableMember<Object>) arg0).getmemberPhone();
        // 오름차순
        return this.memberPhone - comparePhone;
        // 내림차순
        // return comparePhone - this.memberPhone;
    }
}
```

ComparableMemberTest.java

```
import java.util.*;


public class ComparableMemberTest  {
	@SuppressWarnings({ "rawtypes", "unchecked" })
	public static void main(String args[]){
		ArrayList<ComparableMember> arraylist = new ArrayList<ComparableMember>();
		arraylist.add(new ComparableMember(223, "apple", 31));
		arraylist.add(new ComparableMember(245, "mango", 30));
		arraylist.add(new ComparableMember(209, "banana", 32));
		arraylist.add(new ComparableMember(215, "kiwi", 28));
		arraylist.add(new ComparableMember(233, "melon", 29));
		// 전화번호로 정렬
		Collections.sort(arraylist);

		for(ComparableMember str: arraylist){
			System.out.println(str);
		}
	}
}
```

ComparatorMember.java 1

```
package induk.soft.collection;

import java.util.Comparator;
public class ComparatorMember  {
	private String memberName;
	private int rollno;
	private int memberPhone;

	public ComparatorMember(int rollno, String memberName, int memberPhone) {
		this.rollno = rollno;
		this.memberName = memberName;
		this.memberPhone = memberPhone;
	}
	public String getmemberName() {
		return memberName;
	}
	public void setmemberName(String memberName) {
		this.memberName = memberName;
	}
	public int getRollno() {
		return rollno;
	}
	public void setRollno(int rollno) {
		this.rollno = rollno;
	}
```

ComparatorMember.java 2

```
public int getmemberPhone() {
        return memberPhone;
}
public void setmemberPhone(int memberPhone) {
        this.memberPhone = memberPhone;
}

public static Comparator<ComparatorMember> MemNameComparator =
        new Comparator<ComparatorMember>() {
        public int compare(ComparatorMember s1, ComparatorMember s2) {
                String memberName1 = s1.getmemberName().toUpperCase();
                String memberName2 = s2.getmemberName().toUpperCase();
                //오름 차순(ascending order)
                return memberName1.compareTo(memberName2);

                //내림 차순 (descending order)
                //return memberName2.compareTo(memberName1);
        }
};
```

ComparatorMember.java 3

```
    public static Comparator<ComparatorMember> MemNoComparator =
        new Comparator<ComparatorMember>() {

        public int compare(ComparatorMember s1, ComparatorMember s2) {

            int rollno1 = s1.getRollno();

            int rollno2 = s2.getRollno();

            //오름 차순(ascending order)

            return rollno1-rollno2;

            //내림 차순 (descending order)

            //rollno2-rollno1;

        }

    };

    public String toString() {

        return "[ 번호=" + rollno + ", name=" + memberName + ", 전화번호=" + memberPhone + "]";

    }


}
```

ComparatorMemberTest.java

```
import java.util.ArrayList;
import java.util.Collections;
public class ComparatorMemberTest {
	public static void main(String args[]){
		ArrayList<ComparatorMember> arraylist = new ArrayList<ComparatorMember>();
		arraylist.add(new ComparatorMember(11, "yeyeong", 14));
		arraylist.add(new ComparatorMember(12, "junyeong", 10));
		arraylist.add(new ComparatorMember(23, "jeongwon", 18));
		arraylist.add(new ComparatorMember(24, "joowon", 17));
		arraylist.add(new ComparatorMember(35, "soogyeum", 19));
		arraylist.add(new ComparatorMember(36, "minha", 15));

		System.out.println("이름으로 오름 차순:");
		Collections.sort(arraylist, ComparatorMember.MemNameComparator);
		for(ComparatorMember str: arraylist){
			System.out.println(str);
		}
		System.out.println("번호로 오름 차순:");
		Collections.sort(arraylist, ComparatorMember.MemNoComparator);
		for(ComparatorMember str: arraylist){
			System.out.println(str);
		}
	}
}
```

java.util.Vector

- **정의 및 특징**
  - 순서를 가지는 요소들의 모임
  - 중복을 허용하고, 인덱스를 이용한 접근을 허용함.
  - 저장되는 요소들의 개수에 따라 자동적으로 크기가 변경됨 vs. 배열
  - 추가 연산을 실행하는 경우 오버헤드가 발생함 vs. LinkedList
  - 스레드 안전함(thread-safe) vs. ArrayList
- **상속 관계**
  - public class Vector<E>
    extends AbstractList<E>
    implements List<E>, RandomAccess, Cloneable, Serializable

계속

- **생성자**
  - Vector()
    - 초기 용량은 10, 증가 용량 10인 Vector 생성
  - Vector(Collection<? extends E> c)
    - 컬렉션의 반복자가 반환하는 순서로 특정 컬렉션의 요소를 포함하는 Vector를 생성
  - Vector(int initialCapacity)
    - 초기 용량은 initialCapacity, 증가 용량도 현재 용량인 Vector 생성
  - Vector(int initialCapacity, int capacityIncrement)
    - 초기 용량은 initialCapacity, 증가 용량은 capacityIncrement 인 Vector 생성

VectorContructorTest.java

```
import java.util.*;

public class VectorContructorTest {

    public static void main(String args[]) {
        /* Vector of initial capacity(size) : 3 */
        Vector<String> vec = new Vector<String>(3);

        vec.addElement("Apple");
        vec.addElement("Orange");
        vec.addElement("Mango");
        System.out.println("Size is: " + vec.size() + " Capacity is : " + vec.capacity());

        vec.addElement("Fig");
        // current capacity 단위로 용량 증가 : 3 + 3 = 6
        System.out.println("Size is: " + vec.size() + " Capacity is : " + vec.capacity());
```

VectorContructorTest.java

```
        vec.addElement("fruit1");
        vec.addElement("fruit2");
        vec.addElement("fruit3");
        // current capacity 단위로 용량 증가 : 6 + 6 = 12
        System.out.println("Size is: " + vec.size() + " Capacity is : " + vec.capacity());

        vec.addElement("fruit3");
        vec.addElement("fruit4");
        // 현재 size 9이므로 용량을 증가하지 않음
        System.out.println("Size is: " + vec.size() + " Capacity is : " + vec.capacity());

        Enumeration<String> en = vec.elements();
        System.out.println("₩nElements are:");
        while(en.hasMoreElements())
            System.out.print(en.nextElement() + " ");
    }
}
```

계속

- **주요 메소드**
  - void addElement(Object element): 벡터의 마지막에 요소를 추가
  - int capacity(): 벡터의 현재 용량을 반환
  - int size(): 백터의 현재 사이즈(채워진 양)를 반환
  - boolean contains(Object element): 지정한 요소가 벡터에 현재 존재하는 경우 true 반환
  - boolean containsAll(Collection c): 컬렉션 c의 모든 요소들이 벡터에 현재 존재하는 우 true 반환
  - Object elementAt(int index): 벡터의 지정한 위치에 존재하는 요소를 반환
  - Object firstElement(): 벡터의 첫 번째 위치의 요소를 반환
  - Object lastElement(): 벡터의 마지막 위치의 요소를 반환
  - Object get(int index): 지정한 위치에 존재하는 요소를 반환
  - boolean isEmpty(): 벡터에 요소들이 존재하지 않는 경우 true 반환
  - boolean removeElement(Object element): 벡터에서 지정한 요소를 제거
  - boolean removeAll(Collection c): 컬렉션 c의 모든 요소들을 벡터에서 제거
  - void setElementAt(Object element, int index): It updates the element of specifed index with the given element.

VectorTraverseTest.java

```
import java.util.*;

public class VectorTraverseTest {
	public static void main(String[] args) {
		Vector<String> vector = new Vector<String>();
		vector.add("yeyeong");
		vector.add("junyeong");
		vector.add("jeongwon");
		vector.add("joowon");
		vector.add("soogyeom");
		vector.add("minha");
		// Returns an enumeration of the components of this vector
		Enumeration<String> en = vector.elements();
		// Display Vector elements using hashMoreElements() and nextElement() methods.
		System.out.println("Vector using Enueration : ");
		while(en.hasMoreElements())
			System.out.println(en.nextElement());
```

VectorTraverseTest.java

```
        //Obtaining an iterator
        Iterator<String> it = vector.iterator();

        System.out.println("Vector using Iterator : ");
        while(it.hasNext())
            System.out.println(it.next());

        ListIterator<String> litr = vector.listIterator();
        System.out.println("Vector using ListIterator-traversing in Forward Direction:");
        while(litr.hasNext())
            System.out.println(litr.next());

        System.out.println("Vector using ListIterator-traversing in Backward Direction:");
        while(litr.hasPrevious())
            System.out.println(litr.previous());
    }

}
```

ArrayList vs. Vector

- **공통점**
  - 자동 크기 조정가능한 배열 자료 구조
    - 용량을 넘치거나 삭제가 발생하면 크기가 자동적으로 커지거나 줄어듦
  - Iterator, ListIterator 반환하여 순차 탐색하는 경우 : fail-fast 지원
  - 입력 순서를 유지하는 순서있는 컬렉션
  - 값의 중복과 널을 허용함
- **차이점**
  - 사이즈의 절반 크기로 증감 vs. 사이즈와 동일한 크기씩 증감
  - 동기화 지원하지 않음 - 성능 우수 vs. 동기화 지원
  - Vector가 elements()를 이용하여 Enumeration를 반환한 경우 해당 순차 탐색은 fail-fast 지원하지 않음

java.util.LinkedList

- **정의 및 기능**
  - List와 Deque 인터페이스를 구현한 이중 연결 리스트 (doubly-linked list) 구현체
  - 연속적인 공간에 요소를 저장하지 않음
  - 삽입, 삭제가 용이하고, 이로 인한 내부 요소의 이동이 필요 없음
  - 임의 접근을 제공하지 않고 순차 접근을 지원함
- **상속 관계**
  - public class LinkedList<E>
    extends AbstractSequentialList<E>
    implements List<E>, Deque<E>, Cloneable, Serializable

ArrayList vs. LinkedList

- **ArrayList vs. LinkedList 공통점**
  - List 인터페이스의 구현체
  - 입력 순서대로 요소들을 유지함
  - 동기화를 지원하지 않음
    - 필요한 경우 Collections.synchronizedList 메소드 사용
  - fail-fast 지원
    - 순차적 접근이 모두 끝나기 전에 컬렉션 객체에 변경이 일어날 경우 순차적 접근이 실패되면서 ConcurrentModificationException 예외 발생

계속

- **ArrayList vs. LinkedList 차이점**
  - 검색 속도 : O(1) vs O(n)
  - 삭제/삭제 연산 성능 : O(n) - worst case, O(1) - best case vs. O(1) - 시작요소, 마지막 요소, 검색 + O(1) - 중간 요소
  - 메모리 오버헤드 : 낮음 vs. 상대적으로 높음

java.util.HashMap

- **정의 및 기능**
  - 키와 값의 쌍으로 사용하는 컬렉션
  - 동기화를 지원하지 않음, 스레드 안전하지 않음(thread-unsafe)
  - 값의 중복은 허용하지만, 키의 중복은 허용하지 않음
  - 키의 경우는 하나, 값의 경우 다수의 널 객체도 허용
- **상속 관계**
  - public class HashMap<K,V>
    extends AbstractMap<K,V>
    implements Map<K,V>, Cloneable, Serializable

HashMapTest.java

```
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;
import java.util.Set;
import java.util.Iterator;
public class HashMapTest {
	public static void main(String[] args) {
		HashMap<Integer, String> hmap = new HashMap<Integer, String>();
		hmap.put(85, "재훈");
		hmap.put(871, "대열");
		hmap.put(86, "영준");
		hmap.put(88, "대양");
		hmap.put(872, "명조");
		hmap.put(83, "창용");
		hmap.put(84, "재교");
		System.out.println("HashMap은 순서를 유지하지 않음 : ");
		Iterator<Map.Entry<Integer, String>>  iterator = hmap.entrySet().iterator();
		while(iterator.hasNext()) {
			Map.Entry<Integer, String> me = (Map.Entry<Integer, String>) iterator.next();
			System.out.print(me.getKey() + ": ");
			System.out.println(me.getValue());
		}
	}
}
```

LinkedHashMapTest.java

```
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.Map;
public class LinkedHashMapTest {
	public static void main(String[] args) {
		LinkedHashMap<Integer, String> lhmap = new LinkedHashMap<Integer, String>();
		lhmap.put(85, "재훈");
		lhmap.put(871, "대열");
		lhmap.put(86, "영준");
		lhmap.put(88, "대양");
		lhmap.put(872, "명조");
		lhmap.put(83, "창용");
		lhmap.put(84, "재교");
		System.out.println("LinkedHashMap 삽입 순서를 유지함 : ");
		Iterator<Map.Entry<Integer, String>>  iterator = lhmap.entrySet().iterator();
		while(iterator.hasNext()) {
			Map.Entry<Integer, String> me = (Map.Entry<Integer, String>) iterator.next();
			System.out.print(me.getKey() + ": ");
			System.out.println(me.getValue());
		}
	}
}
```

TreeMapTest.java

```
import java.util.Iterator;
import java.util.Map;
import java.util.TreeMap;


public class TreeMapTest {
	public static void main(String[] args) {
		TreeMap<Integer, String> tmap = new TreeMap<Integer, String>();
		tmap.put(85, "재훈");
		tmap.put(871, "대열");
		tmap.put(86, "영준");
		tmap.put(88, "대양");
		tmap.put(872, "명조");
		tmap.put(83, "창용");
		tmap.put(84, "재교");
		System.out.println("TreeMap 키 기준 정렬 : ");
		Iterator<Map.Entry<Integer, String>>  iterator2 = tmap.entrySet().iterator();
		while(iterator2.hasNext()) {
			Map.Entry<Integer, String> me = (Map.Entry<Integer, String>) iterator2.next();
			System.out.print(me.getKey() + ": ");
			System.out.println(me.getValue());
		}
	}
}
```

비교

- **HashMap, TreeMap, LinkedHashMap ordering**
  - HashMap : 순서를 유지하지 않음
  - TreeMap : 키의 오름차순으로 엔트리를 정렬
  - LinkedHashMap : 삽입 순서를 유지함

HashMapSortTest.java

```
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;
import java.util.Set;
import java.util.Iterator;
public class HashMapSortTest {
	public static void main(String[] args) {
		HashMap<Integer, String> hmap = new HashMap<Integer, String>();
		hmap.put(85, "재훈");
		hmap.put(871, "대열");
		hmap.put(86, "영준");
		hmap.put(88, "대양");
		hmap.put(872, "명조");
		hmap.put(83, "창용");
		hmap.put(84, "재교");

		Map<Integer, String> map = new TreeMap<Integer, String>(hmap);
		System.out.println("HashMap을 TreeMap을 이용하여 키기준으로 정렬 :");
		Iterator<Map.Entry<Integer, String>> iterator = map.entrySet().iterator();
		while(iterator.hasNext()) {
			Map.Entry<Integer, String> me = (Map.Entry<Integer, String>) iterator.next();
			System.out.print(me.getKey() + ": ");
			System.out.println(me2getValue());
		}
	}
}
```

HashMapSortByValue.java

```
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedHashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

public class HashMapSortByValue {
	public static void main(String[] args) {
		HashMap<Integer, String> hmap = new HashMap<Integer, String>();
		hmap.put(85, "재훈");		hmap.put(871, "대열");		hmap.put(86, "영준");
		hmap.put(88, "대양");		hmap.put(872, "명조");		hmap.put(83, "창용");
		hmap.put(84, "재교");
		Map<Integer, String> map = sortByValues(hmap);
		System.out.println("값 기준으로 정렬 후:");
		Iterator<Map.Entry<Integer, String>> iterator2 = map.entrySet().iterator();
		while(iterator2.hasNext()) {
			Map.Entry<Integer, String> me2 = (Map.Entry<Integer, String>) iterator2.next();
			System.out.print(me2.getKey() + ": ");
			System.out.println(me2.getValue());
		}
	}
```

HashMapSortByValue.java

```
    private static HashMap<Integer, String> sortByValues(HashMap<Integer, String> map) {
        List list = new LinkedList(map.entrySet());
        // 키가 아닌 값을 비교하는 Comparator 정의
        Collections.sort(list, new Comparator() {
            public int compare(Object o1, Object o2) {
                return ((Comparable) ((Map.Entry) (o1)).getValue())
                  .compareTo(((Map.Entry) (o2)).getValue());
            }
        });
        // Collections.reverse(list); // 내림차순, 즉 오름차순의 역순으로 리스트 객체 생성
        // 정렬된 리스트를 삽입 순서를 유지하는 LinkedHashMap을 이용하여 HashMap에 반환
        HashMap sortedHashMap = new LinkedHashMap();
        for (Iterator it = list.iterator(); it.hasNext();) {
            Map.Entry entry = (Map.Entry) it.next();
            sortedHashMap.put(entry.getKey(), entry.getValue());
        }
        return sortedHashMap;
    }
}
```

계속

- **주요 메소드**
  - boolean containsKey(Object key)
    - 지정한 키에 매핑되는 값이 존재하는 경우 true 반환
  - boolean containsValue(Object value)
    - 지정한 값에 매핑되는 키가 하나 이상 존재하는 경우 true 반환

java.util.HashTable

- **정의 및 기능**
  - 키들을 값에 매핑하는 해쉬 테이블을 구현한 클래스
  - 널 아닌 객체만이 키나 값으로 사용될 수 있음
  - 키로 사용되는 객체는 hashCode(), equals() 메소드를 구현해야만 함
- **성능에 영향을 주는 파라미터**
  - initial capacity : 해쉬 테이블이 생성된 시점에서의 버킷들의 수
    - hash collision : 하나의 버킷에 다수의 엔트리들이 저장되는 것으로 순차적으로 검색됨.
  - load factor : 용량이 자동적으로 증가하기 전에 해쉬 테이블이 얼마나 채워졌는지를 나타내는 값

계속

- **상속 관계**
  - public class Hashtable<K,V>
    extends Dictionary<K,V>
    implements Map<K,V>, Cloneable, Serializable

HashMap vs. Hashtable

- **차이점**
  - 동기화 지원하지 않음, 스레드 안전하지 않음 vs. 동기화 지원함, 스레드 안전함
  - null key, null values 허용함 vs. any null key or value 허용하지 않음
  - 속도가 빠름 vs. 속도가 느림
  - Traverse : Iterator vs. Iterator or Enumerator
  - fail-fast 지원함 vs. fail-fast 지원하지 않음
  - AbstractMap 상속 vs. Dictionary 상속

HashSet

- **주요 특징**
  - Set 인터페이스의 구현체, 중복 요소를 허용하지 않음
  - 널 값은 허용함
  - iterator 를 이용한 순차접근시 fail-fast 지원
  - 순서를 유지하지 않음
    - LinkedHashSet은 삽입 순서 유지
  - 동기화를 지원하지 않음
    - 명시적 방법 지원 방법 : Set set = Collections.synchronizedSet(new HashSet(...))
- **상속관계**
  - public class HashSet<E>
    extends AbstractSet<E>
    implements Set<E>, Cloneable, Serializable

HashSetTest.java

```
import java.util.HashSet;
import java.util.LinkedHashSet;

public class HashSetTest {

	public static void main(String[] args) {
		// TODO Auto-generated method stub
		String[] strArray = { "apple", "orange", "melon", "apple", "tomato", "banana"};
		HashSet<String> hashSet = new HashSet<String>();
		LinkedHashSet<String> linkedHashSet = new LinkedHashSet<String>();

		for(int i = 0; i < strArray.length; i++)
			hashSet.add(strArray[i]);
		System.out.println(hashSet);
		for(String str : strArray)
			linkedHashSet.add(str);
		System.out.println(linkedHashSet);

	}
}
```

[실행결과]
[orange, banana, apple, tomato, melon]
[apple, orange, melon, tomato, banana]

HashSet vs. TreeSet

- **차이점**
  - 구현 : hashing 사용 vs. red-black tree 사용
  - 속도 : 빠름 vs. 느림 - log(n)
  - 순서 : 유지하지 않음 vs. 삽입 순서와 관계없지만 값 기준 오름차순 정렬
- **공통점**
  - Set 인터페이스 구현체
    - 순서를 유지하지 않음
  - 중복을 허용하지 않음

TreeSetTest.java

```
import java.util.TreeSet;

public class TreeSetTest {
    public static void main(String args[]) {
        TreeSet<String> strTreeSet = new TreeSet<String>();
        strTreeSet.add("Apple");
        strTreeSet.add("Orange");
        strTreeSet.add("Melon");
        strTreeSet.add("Tomato");
        strTreeSet.add("Banana");

        for(String element : strTreeSet)
            System.out.print(element + "₩n");  }
}
```

[실행결과]
Apple
Banana
Melon
Orange
Tomato

HashSet을 배열로 변경

```
class ConvertHashSettoArray{
  public static void main(String[] args) {
     HashSet<String> hset = new HashSet<String>();

     hset.add("Element1");
     hset.add("Element2");
     hset.add("Element3");
     hset.add("Element4");

     // Displaying HashSet elements
     System.out.println("HashSet contains: "+ hset);

     // Creating an Array
     String[] array = new String[hset.size()];
     hset.toArray(array);

     // Displaying Array elements
     System.out.println("Array elements: ");
     for(String temp : array){
        System.out.println(temp);
     }
  }
}
```

java.util.Arrays

- **정의 및 기능**
  - 배열들을 조작(정렬, 찾기 등)하기 위한 다양한 메소드를 제공하는 클래스
- **상속 관계**
  - public class Arrays extends Object
- **주요 메소드**
  - public static <T> List<T> asList(T... a)
  - binarySearch(), copyOf(), copyOfRange(), equals(), fill(), hashCode(), sort(), toString()

Collections.addAll()를 활용하여 배열 List 생성

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class ArrayToListTest {
    public static void main(String[] args) {
        String[] strArray = {"Apple", "Orange", "Melon", "Tomato", "Banana"};
        List<String> strList = new ArrayList<String>();
        Collections.addAll(strList, strArray); // List 객체 생성
        System.out.println(strList);
        List<String> strList2 = Arrays.asList(strArray);    // 배열을 List로 변환
        System.out.println(strList2);

    }
}
```

sort(T[] elements), binarySearch(T[] elements, T key)

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

public class ArraysSortSearchTest {
	public static void main(String[] args) {
		String[] strArray = {"Apple", "Orange", "Melon", "Tomato", "Banana"};
		Arrays.sort(strArray); // 정렬

		System.out.println(Arrays.binarySearch(strArray, "Apple"));
		System.out.println(Arrays.binarySearch(strArray, "Orange"));
		System.out.println(Arrays.binarySearch(strArray, "Melon"));
		System.out.println(Arrays.binarySearch(strArray, "Tomato"));
		System.out.println(Arrays.binarySearch(strArray, "Banana"));
	}
}
```

ArrayList 를 배열로 변환 : toArray()

```
import java.util.ArrayList;

public class ArrayListToArray {

	public static void main(String[] args) {
		ArrayList<String> list = new ArrayList<String>();
		list.add("yeyeong");
		list.add("junyeong");
		list.add("jeongwon");
		list.add("joowon");
		list.add("soogyeom");
		list.add("minha");

		String[] strArray = list.toArray(new String[list.size()]);

		for(String str : strArray)
			System.out.println(str);
	}
}
```

Arrays.copyOf() 메소드 활용

```
import java.util.Arrays;

public class ArraysCopyOfTest {

    public static void main(String[] args) {
            int[] arr1 = {15, 10, 45, 13, 27};
            for(int i : arr1)
                    System.out.print(i + ", ");

            System.out.print("₩nCopyOf 5 -> 2 : ");
            int[] copyOfArr1 = Arrays.copyOf(arr1, 2);
            for(int j : copyOfArr1)
                    System.out.print(j + ", ");

            System.out.print("₩nCopyOf 5 -> 7 : ");
            copyOfArr1 = Arrays.copyOf(arr1, 7);
            for(int k : copyOfArr1)
                    System.out.print(k + ", ");
    }
}
```

학습 후 기대 효과

- **JCF를 구성하는 인터페이스들, 구현요소들과 알고리즘을 프로그램 작성에 활용할 수 있다.**
  - List, ArrayList, Vector, Stack, LinkedList
  - HashMap, HashTable, TreeMap, LinkedHashMap
  - HashSet, TreeSet, LinkedHashSet
  - Collections, Comparable, Comparator, Arrays