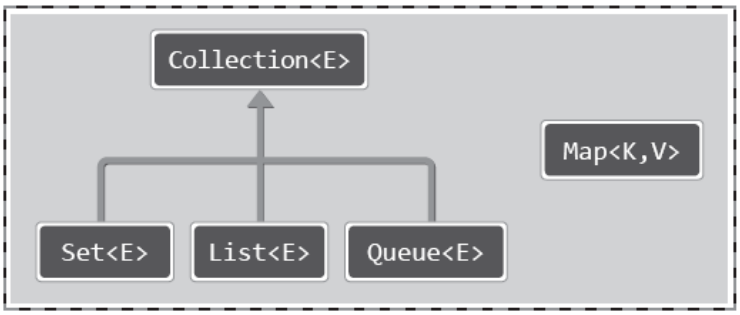
컬렉션 프레임워크

프레임 워크가 의미하는 바는 다음과 같다.
  잘 정의된, 약속된 구조와 골격

컬렉션 프레임워크가 제공하는 기능의 영역
  자료구조와 알고리즘

자바의 컬렉션 프레임워크는 별도의 구현과 이해 없이 자료구조와 알고리즘을 적용할 수 있도록 설계된 클래스들의 집합이다. 그러나 자료구조의 이론적인 특성을 안다면, 보다 적절하고 합리적인 활용이 가능하다.



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

[표11-1] 컬렉션 프레임웍의 핵심 인터페이스와 특징

제네릭

컬렉션 프레임워크에 저장할 데이터 타입을 정하지 않고 객체를 생성할 때(new) 저장할 데이터 타입을 정하도록 한것. 구조는<E> 형태로 사용한다. 아래 예제에서 많이 사용하고 있다.

Vector

배열과 같은 구조를 가지며 저장 공간이 고정된 것이 아니고 데이터를 추가할 때마다 공간이 늘어난다.

실습예제1

```
import java.util.Vector;

public class VectorEx {
	public static void main(String[] args) {
		// 정수 값만 다루는 제네릭 벡터 생성
		Vector<Integer> v = new Vector<Integer>();
		v.add(5); // 5 삽입
		v.add(4); // 4 삽입
		v.add(-1); // -1 삽입

		// 벡터 중간에 삽입하기
		v.add(2, 100); // 4와 -1 사이에 정수 100 삽입

		System.out.println("벡터 내의 요소 객체 수 : " + v.size()); // 크기 3
		System.out.println("벡터의 현재 용량 : " + v.capacity()); // 벡터의 디폴트 용량 10

		// 모든 요소 정수 출력하기
		for(int i=0; i<v.size(); i++) {
			int n = v.get(i);
			System.out.println(n);
		}

		// 벡터 속의 모든 정수 더하기
		int sum = 0;
		for(int i=0; i<v.size(); i++) {
			int n = v.elementAt(i);
			sum += n;
		}
		System.out.println("벡터에 있는 정수 합 : " + sum);
	}
}
```

```
벡터 내의 요소 객체 수 : 4
벡터의 현재 용량 : 10
5
4
100
-1
벡터에 있는 정수 합 : 108
```

아래 코드는 정수, 실수 ,문자, 문자열이 아닌 객체를 Vector에 저장하는 예제이다. 객체를 저장한다는 의미는 클래스의 멤버변수의 값들을 저장한다는 의미이다.

실습예제2

```
import java.util.Vector;

class Point {
	private int x, y;
	public Point(int x, int y) {
		this.x = x;
		this.y = y;
	}
	public String toString() {
		return "(" + x + "," + y + ")";
	}
}

public class PointVectorEx {
	public static void main(String[] args) {
		// Point 객체를 요소로만 가지는 벡터 생성
		Vector<Point> v = new Vector<Point>();

		// 3 개의 Point 객체 삽입
		v.add(new Point(2, 3));
		v.add(new Point(-5, 20));
		v.add(new Point(30, -8));

		// 벡터에 있는 Point 객체 모두 검색하여 출력
		for(int i=0; i<v.size(); i++) {
			Point p = v.get(i); // 벡터에서 i 번째 Point 객체 얻어내기
			System.out.println(p); // p.toString()을 이용하여 객체 p 출력
		}
	}
}
```

```
(2,3)
(-5,20)
(30,-8)
```

아래에서 5개의 실수를 입력 받아 가장 큰 수를 구하는 코드 실습하시오

```
import java.util.*;
public class VectorBig {
	public static void main(String[] args) {
		Vector<Double> v = new Vector<Double>();
		Scanner scanner = new Scanner(System.in);
		for(int i=0; i<5; i++) {
			double d = scanner.nextDouble();
			v.add(d);
		}

		double big = v.get(0);
		for(int i=1; i<v.size(); i++) {
			if(big < v.get(i))
				big = v.get(i);
		}

		System.out.println("가장 큰 수는 " + big);
	}
}
```

```
0.1 0.2 0.3 0.4 0.41
가장 큰 수는 0.41
```

컬렉션 클래스를 이용한 정수의 저장

```
ArrayList<int> arr=new ArrayList<int>( ); error
LinkedList<int> link=new LinkedList<int>( ); error
```

기본 자료형 정보를 이용해서 제네릭 인스턴스 생성 불가능! 따라서 Wraper 클래스를 기반으로 컬렉션 인스턴스를 생성한다.

실습예제3

```
import java.util.Iterator;
import java.util.LinkedList;

class ListTest {
	public static void main(String[] args) {
		LinkedList<Integer> list = new LinkedList<Integer>();
		list.add(10);	// Auto Boxing
		list.add(20);	// Auto Boxing
		list.add(30);	// Auto Boxing

		Iterator<Integer>	it = list.iterator();

		while(it.hasNext())
		{
			int num = it.next();	// Auto Unboxing
			System.out.println(num);
		}
	}
}
```

```
10
20
30
```

Auto Boxing과 Auto Unboxing의 도움으로 정수 단위의 데이터 입출력이 매우 자연스럽다!

ArrayList<E>, LinkedList<E>

List<E> 인터페이스를 구현하는 대표적인 제네릭 클래스

인터페이스를 구현 클래스의 인스턴스 저장 특징
동일한 인스턴스의 중복 저장을 허용한다.
인스턴스의 저장 순서가 유지된다.

실습예제4

```
import java.util.Iterator;
import java.util.LinkedList;

class ListTest {
	public static void main(String[] args) {
		LinkedList<Integer> list = new LinkedList<Integer>();

		// 데이터의 저장
		list.add(11);
		list.add(22);
```

```
                list.add(33);

                // 데이터의 출력
                System.out.println("삭제전");
                for(int i=0; i<list.size(); i++) {
                        System.out.println(list.get(i));
                }
                // 데이터의 삭제
                list.remove(0);
                System.out.println("삭제후");
                for(int i=0; i<list.size(); i++) {
                        System.out.println(list.get(i));
                }
        }
}
```

```
삭제전
11
22
33
삭제후
22
33
```

ArrayList<E>와 LinkedList<E>의 차이점

저장소의 용량을 늘리는 과정에서 많은 시간이 소요된다. ArrayList<E>의 단점 데이터의 삭제에 필요한 연산 과정이 매우 길다. ArrayList<E>의 단점 데이터의 참조가 용이해서 빠른 참조가 가능하다. ArrayList<E>의 장점
저장소의 용량을 늘리는 과정이 간단하다. LinkedList<E>의 장점 데이터의 삭제가 매우 간단하다. LinkedList<E>의 장점 데이터의 참조가 다소 불편하다.LinkedList<E>의 단점

Iterator를 이용한 인스턴스의 순차적 접근

Collection<E> 인터페이스에는 iterator라는 이름의 메소드가 다음의 형태로 정의
iterator 메소드가 반환하는 참조 값의 인스턴스는 Iterator<E> 인터페이스를 구현하고 있다.
iterator 메소드가 반환한 참조값의 인스턴스를 이용하면, 컬렉션 인스턴스에 저장된 인스턴스의 순차적 접근이 가능함.
iterator 메소드의 반환형이 Iterator<E>이니, 반환된 참조값을 이용해서 Iterator<E>에 선언된 함수들만 호출하면 된다.

Iterator<E>	인터페이스에 정의된 메소드
boolean hasNext( )	참조할다음번요소(element)가존재하면true를반환
E next()	다음번요소를반환
void remove()	현재위치의요소를삭제

실습예제5

```
import java.util.*;

public class IteratorEx {

	public static void main(String[] args) {
		// 정수 값만 다루는 제네릭 벡터 생성
		Vector<Integer> v = new Vector<Integer>();
		v.add(5); // 5 삽입
		v.add(4); // 4 삽입
		v.add(-1); // -1 삽입
		v.add(2, 100); // 4와 -1 사이에 정수 100 삽입

		// Iterator를 이용한 모든 정수 출력하기
		Iterator<Integer> it = v.iterator(); // Iterator 객체 얻기
		while(it.hasNext()) {
			int n = it.next();
			System.out.println(n);
		}

		// Iterator를 이용하여 모든 정수 더하기
		int sum = 0;
		it = v.iterator(); // Iterator 객체 얻기
		while(it.hasNext()) {
			int n = it.next();
			sum += n;
		}
		System.out.println("벡터에 있는 정수 합 : " + sum);
	}
}
```

```
5
4
100
-1
벡터에 있는 정수 합 : 108
```

실습예제6

```
import java.util.Iterator;
import java.util.LinkedList;

class ListTest {
	public static void main(String[] args) {
		LinkedList<String> list = new LinkedList<String>();
		list.add("First");
		list.add("Second");
		list.add("Third");
		list.add("Fourth");

		Iterator<String> it = list.iterator();

		System.out.println("반복자를 이용한 1차 출력과  Third 삭제");

		while(it.hasNext())
		{
			String str = it.next();
			System.out.println(str);
```

```
                                if(str.compareTo("Third") == 0) {
                                        it.remove();
                                }
                        }

                        System.out.println("\nThird 삭제 후 반복자를 이용한 2차 출력");
                        it = list.iterator();
                        while(it.hasNext())
                        {
                                System.out.println(it.next());
                        }
                }
}
```

```
반복자를 이용한 1차 출력과  Third 삭제
First
Second
Third
Fourth

Third 삭제 후 반복자를 이용한 2차 출력
First
Second
Fourth
```

'반복자'를 사용하는 이유

반복자를 사용하면, 컬렉션 클래스의 종류에 상관없이 동일한 형태의 데이터 참조 방식을 유지할 수 있다. 따라서 컬렉션 클래스의 교체에 큰 영향이 없다.
컬렉션 클래스 별 데이터 참조 방식을 별도로 확인할 필요가 없다.

```
LinkedList<String> list
=new LinkedList<String>( );
```

위의 코드에서 반복자를 사용했기 때문에, LinkedList<E>가어울리지 않아서, 컬렉션클래스를 HashSet<E>로 변경해야 할때, 다음과 같이 변경이 매우 용이하다.

```
HashSet<String> set
=new HashSet<String>( );
```

Set<E>

List<E>를 구현하는 클래스들과 달리Set<E>를 구현하는 클래스들은 데이터의 저장순서를 유지하지 않는다. List<E>를 구현하는 클래스들과 달리Set<E>를 구현하는 클래스들은 데이터의 중복 저장을 허용하지 않는다. 단, 동일 데이터에 대한 기준은 프로그래머가 정의
즉, Set<E>를 구현하는 클래스는 '집합'의 성격을 지닌다.

실습예제7

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

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

```
                set.add("First");
                set.add("Second");
                set.add("Third");
                set.add("First");

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

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

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

동일 인스턴스의 판단 기준 관찰을 위한 예

Object 클래스에 정의되어 있는 equals 메소드의 호출 결과와 hashCode 메소드의 호출 결과를 참조하여 인스턴스의 동등 비교를 진행. 즉, new를 하면 각각의 인스턴스의 주소값이 다르기 때문에 서로 다른 값을 가진다.

실습예제8

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

class Number {
        int num;
        public Number(int n)
        {
                num = n;
        }

        public String toString()
        {
                return String.valueOf(num);
        }

        public static void main(String[] args) {
                HashSet<Number> set = new HashSet<Number>();
                set.add(new Number(10));
                set.add(new Number(20));
                set.add(new Number(20));

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

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

```
저장된 데이터 수 : 3
```

```
10
20
20
```

TreeSet<E> 클래스의 이해와 활용

TreeSet<E> 클래스는 트리라는 자료구조 를기반으로 데이터를 저장한다.
데이터를 정렬된 순서로 저장하며, HashSet<E>와 마찬가지로 데이터의 중복 저장 않는다.
정렬의 기준은 프로그래머가 직접 정의한다.

데이터는 정렬 되어 저장이 되며, 때문에 iterator 메소드의 호출로 생성된 반복자는 오름차순의 데이터참조를 진행한다.

실습예제9

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

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

		tree.add(1);
		tree.add(2);
		tree.add(4);
		tree.add(3);
		tree.add(2);

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

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

```
저장된 데이터 수 : 4
1
2
3
4
```

출력 순서가 정렬 되어 있음에 주목해야 한다! 이것이 TreeSet<E>의 특징이다.

Map<K, V> 인터페이스와 HashMap<K, V> 클래스

Map<K, V> 인터페이스를 구현하는 컬렉션 클래스는 key-value 방식의 데이터 저장을 한다.
value는 저장할 데이터를 의미하고, key는 value를 찾는 열쇠를 의미한다.
Map<K, V>를 구현하는 대표적인 클래스로는 HashMap<K, V>와 TreeMap<K, V>가 있다.

TreeMap<K, V>는 정렬된 형태로 데이터가 저장된다.

실습예제10

```
import java.util.HashMap;

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

		map.put(new Integer(3),   "나삼번");
		map.put(5, "윤오번");
		map.put(8, "박팔번");

		System.out.println("6학년 3반 8번 학생 : " + map.get(new Integer(8)));
		System.out.println("6학년 3반 5번 학생 : " + map.get(5));
		System.out.println("6학년 3반 3번 학생: " + map.get(3));

		map.remove(5);   // 5번 학생 전학 감
		System.out.println("6학년 3반 5번 학생 :" + map.get(5));
	}
}
```

```
6학년 3반 8번 학생 : 박팔번
6학년 3반 5번 학생 : 윤오번
6학년 3반 3번 학생: 나삼번
6학년 3반 5번 학생 :null
```

TreeMap<K, V> 클래스의 활용 예

실습예제11

아래 코드에서 TreeMap의 자동 정렬 기능을 실습하시오

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

class MapTest {
	public static void main(String[] args) {
		TreeMap<Integer, String> map = new TreeMap<Integer, String>();

		map.put(1,   "one");
		map.put(3,   "three");
		map.put(5,   "five");
		map.put(2,   "two");
		map.put(4,   "four");

		NavigableSet<Integer> navi = map.navigableKeySet();

		System.out.println("오름차순 출력 ...");
		Iterator<Integer> it = navi.iterator();
		while(it.hasNext())
		{
			System.out.println(map.get(it.next()));
```

```
                }

                System.out.println("내림차순 출력 ...");
                it = navi.descendingIterator();
                while(it.hasNext())
                {
                        System.out.println(map.get(it.next()));
                }
        }
}
```

```
오름차순 출력 ...
one
two
three
four
five
내림차순 출력 ...
five
four
three
two
one
```

실습예제12

```
import java.util.*;

public class HashMapDicEx {
        public static void main(String[] args) {
            // 영어 단어와 한글 단어의 쌍을 저장하는 HashMap 컬렉션 생성
            HashMap<String, String> dic = new HashMap<String, String>();

            // 3 개의 (key, value) 쌍을 dic에 저장
            dic.put("baby", "아기"); // "baby"는 key, "아기"은 value
            dic.put("love", "사랑");
            dic.put("apple", "사과");

            // dic 컬렉션에 들어 있는 모든 (key, value) 쌍 출력
            Set<String> keys = dic.keySet(); // key 문자열을 가진 집합 Set 컬렉션 리턴
            Iterator<String> it = keys.iterator(); // key 문자열을 순서대로 접근하는 Iterator 리턴
            while(it.hasNext()) {
                    String key = it.next();
                    String value = dic.get(key);
                    System.out.println("(" + key + "," + value + ")");
            }
            // 사용자로부터 영어 단어를 입력받고 한글 단어 검색
            Scanner scanner = new Scanner(System.in);
            for(int i=0; i<3; i++) {
                    System.out.print("찾고 싶은 단어는?");
                    String eng = scanner.next();
                    System.out.println(dic.get(eng));
            }
    }
}
```

```
(love,사랑)
(apple,사과)
(baby,아기)
찾고 싶은 단어는? baby
```

```
아기
찾고 싶은 단어는? lll
null
찾고 싶은 단어는? love
사랑
```

실습예제13

```
import java.util.*;

public class HashMapScoreEx {
	public static void main(String[] args) {
		// 사용자 이름과 점수를 기록하는 HashMap 컬렉션 생성
		HashMap<String, Integer> javaScore = new HashMap<String, Integer>();

		// 5 개의 점수 저장
		javaScore.put("한홍진", 97);
		javaScore.put("황기태", 34);
		javaScore.put("이영희", 98);
		javaScore.put("정원석", 70);
		javaScore.put("한원선", 99);

		System.out.println("HashMap의 요소 개수 :" + javaScore.size());

		// 모든 사람의 점수 출력. javaScore에 들어 있는 모든 (key, value) 쌍 출력
		Set<String> keys = javaScore.keySet(); // key 문자열을 가진 집합 Set 컬렉션 리턴
		Iterator<String> it = keys.iterator(); // key 문자열을 순서대로 접근할 수 있는 Iterator 리턴
		while(it.hasNext()) {
			String name = it.next();
			int score = javaScore.get(name);
			System.out.println(name + " : " + score);
		}
	}
}
```

```
HashMap의 요소 개수 :5
한원선 : 99
한홍진 : 97
황기태 : 34
이영희 : 98
정원석 : 70
```

실습예제14 – HashMap 이용

나라이름과 인구를 각각 5개 입력하고 나라이름으로 인구를 검색할 수 있는 코드를 작성하시오.

```
import java.util.*;
public class HashMapNation {
	public static void main(String[] args) {
		HashMap<String, Integer> nations = new HashMap<String, Integer>();
		Scanner scanner = new Scanner(System.in);

		System.out.println("나라 이름과 인구를 5개 입력하세요.(예: Korea 5000)");
		for(int i=0; i<5; i++) {
			System.out.print("나라 이름, 인구 >> ");
			String nation = scanner.next();
			int population = scanner.nextInt();
```

```
                        nations.put(nation, population);
                }

                System.out.println("나라 이름을 입력하면 인구를 검색할 수 있습니다. 끝을 입력하면 종료합니다.");

                while(true) {
                        System.out.print("나라 이름 >> ");
                        String nation = scanner.next();
                        if(nation.equals("끝"))
                                break;
                        Integer n = nations.get(nation);
                        if(n == null)
                                System.out.println(nation + " 나라는 없습니다.");
                        else
                                System.out.println(nation + "의 인구는 " + n);
                }
        }
}
```

```
나라 이름과 인구를 5개 입력하세요.(예: Korea 5000)
나라 이름, 인구 >> korea 1000
나라 이름, 인구 >> japan 2000
나라 이름, 인구 >> china 3000
나라 이름, 인구 >> india 400
나라 이름, 인구 >> usa 300
나라 이름을 입력하면 인구를 검색할 수 있습니다. 끝을 입력하면 종료합니다.
나라 이름 >> india
india의 인구는 400
```

실습예제15 - ArrayList로 학생정보를 저장하시오

학생 class를 만들고 3명의 정보를 입력 한 후 출력하는 코드를 작성하시오

멤버변수는 모두 private로 하고 (name, department, id, grade) 입니다.

```
<Student.java>
public class Student {
        private String name;
        private String department;
        private String id;
        private double grade;

        public Student(String name, String department, String id, double grade) {
                this.name = name;
                this.department = department;
                this.id = id;
                this.grade = grade;
        }
        public void setName(String name) {
                this.name = name;
        }
        public String getName() {
                return name;
        }
        public void setDepartment(String department) {
                this.department = department;
        }
```

```
	public String getDepartment() {
		return department;
	}
	public void setId(String id) {
		this.id = id;
	}
	public String getId() {
		return id;
	}
	public void setGrade(double grade) {
		this.grade = grade;
	}
	public double getGrade() {
		return grade;
	}
}
```

<StudentManager.java>

```
import java.util.*;
public class StudentManager {
	public static void main (String[] args) {
		ArrayList<Student> dept = new ArrayList<Student>();
		Scanner sin = new Scanner(System.in);
		System.out.println("학생 정보를 입력하세요.");
		for (int i=0; i<3; i++) {
			System.out.print(i + "이름>>");
			String name = sin.next();
			System.out.print(i + "학과>>");
			String department = sin.next();
			System.out.print(i + "학번>>");
			String id = sin.next();
			System.out.print(i + "학점평균>>");
			double grade = sin.nextDouble();
			Student st = new Student(name, department, id, grade);
			dept.add(st);
		}
		Iterator<Student> it = dept.iterator();
		while (it.hasNext()) {
			Student st = it.next();
			if (st != null) {
				System.out.println("---------------------------");
				System.out.println("이름:" + st.getName());
				System.out.println("학과:" + st.getDepartment());
				System.out.println("학번:" + st.getId());
				System.out.println("학점평균:" + st.getGrade());
				System.out.println("---------------------------");
			}
		}
	}
}
```

```
학생 정보를 입력하세요.
0이름>>홍길동
0학과>>컴공과
0학번>>911111
0학점평균>>3.0
1이름>>강감찬
```

```
1학과>>물리학과
1학번>>201111
1학점평균>>4.1
2이름>>이순신
2학과>>조선공학과
2학번>>980101
2학점평균>>4.5
----------
이름:홍길동
학과:컴공과
학번:911111
학점평균:3.0
----------
----------
이름:강감찬
학과:물리학과
학번:201111
학점평균:4.1
----------
----------
이름:이순신
학과:조선공학과
학번:980101
학점평균:4.5
----------
```

실습예제16

지금까지 배운 문법과 개념을 기반으로 “전화번호 관리프로그램”을 만들어 보시오.

또한, 입력한 정보를 파일(.dat)에 저장하시오.

아래 코드를 기반으로 실습해 보시오.

<PhoneBookApp.java>

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

public class PhoneBookApp {
        // 메뉴 번호를 위한 상수를 정의한다.
        final int INSERT = 0;
        final int DELETE = 1;
        final int SEARCH = 2;
        final int SHOWALL = 3;
        final int FILESAVE = 4;
        final int EXIT = 5;

        private Scanner scanner;
        private BufferedReader br = null;
        private BufferedWriter bw = null;
        private Hashtable<String, Phone> table = new Hashtable<String, Phone>();

        public PhoneBookApp() {
                scanner = new Scanner(System.in);
```

```
}

 // 전화번호 관리 메소드
public void run() {
   readPhoneBook();
   System.out.println("-------------------------------------------------");
   System.out.println("전화번호 관리 프로그램을 시작합니다. 파일로 저장합니다.");
   System.out.println("-------------------------------------------------");
   while(true) {
         System.out.print("삽입:0. 삭제:1, 찾기:2, 전체보기:3, 파일 저장:4, 종료:5>>");
         int menu = readNumber();
         switch(menu) {
               case INSERT: insert(); break;
               case DELETE: delete(); break;
               case SEARCH: search(); break;
               case SHOWALL: showAll(); break;
               case FILESAVE: saveFile(); break;
               case EXIT:          System.out.println("프로그램을 종료합니다..."); return;

               default: System.out.println("입력이 틀렸습니다. 다시 입력하세요."); continue;
         }
    }
}


private void readPhoneBook() {
    try {
         br = new BufferedReader(new FileReader("phonebook.dat"));
    } catch (FileNotFoundException e) {
         return;
    }
    while (true) {
         String name = null;
         String address = null;
         String telNum = null;
           try {
               name = (String)br.readLine();
               address = (String)br.readLine();
               telNum = (String)br.readLine();
                  if (name != null && address != null && telNum != null)
                        table.put(name, new Phone(name, address, telNum));
                  else {
                        br.close();
                        break; // 파일 끝에 도달한 경우
                  }
         }
         catch (IOException e) {
               System.exit(1);
         }
      }
}

private void saveFile() {
         try {
               bw = new BufferedWriter(new FileWriter("phonebook.dat", false));
         } catch (FileNotFoundException e) {
               return;
         } catch (IOException e) {
               return;
         }

         Enumeration<String> e = table.keys();
```

```
                while(e.hasMoreElements()) {
                        String name = e.nextElement();
                        Phone p = table.get(name);
                        try {
                                bw.write(name+"\n");
                                bw.write(p.getAddress()+"\n");
                                bw.write(p.getTelNum()+"\n");
                        } catch (IOException e1) {
                                System.exit(1);
                        }
                }
                try {
                        bw.close();
                } catch (IOException e1) {
                }
        }

        // 삽입 메뉴를 구현하며 하나의 전화 번호 레코드를 저장한다.
        private void insert() {
                System.out.print("이름>>");
                String name = scanner.next();
                if(table.get(name)!=null) {
                        System.out.println("이미 존재하는 사람입니다.");
                        return;
                }

                System.out.print("주소>>");
                String address = scanner.next();
                System.out.print("전화번호>>");
                String telNum = scanner.next();

                // 해쉬 테이블에 삽입
                table.put(name, new Phone(name, address, telNum));
        }

        // 삭제 메뉴를 구현하며 하나의 전화 번호 레코드를 삭제한다.
        private void delete() {
                System.out.print("이름>>");
                String name = scanner.next();
                Phone p = table.remove(name); // 해쉬 테이블에서 삭제
                if(p == null)
                    System.out.println(name +"는 등록되지 않은 사람입니다.");
                else {
                    System.out.println(name+"은 삭제되었습니다.");
                }
        }

        // 찾기 메뉴를 구현하며 하나의 전화 번호 레코드를 검색한다.
        private void search() {
                System.out.print("이름>>");
                String name = scanner.next();
                Phone p = table.get(name); // 해쉬테이블에서 검색
                if(p == null)
                        System.out.println(name +"는 등록되지 않은 사람입니다.");
                else
                        System.out.println(p.toString());
        }
```

```
	// 전체보기 메뉴를 구현한다.
	private void showAll() {
		// 다음의 한 줄로 만으로도 가능하다. 그러나 다시 작성함
		//System.out.println(table.toString());

		Enumeration<String> e = table.keys();
		while(e.hasMoreElements()) {
			String name = e.nextElement();
			Phone p = table.get(name);
			System.out.println(p.toString());
		}
	}

	private int readNumber() {
		int n=-1;
		try {
		    n = scanner.nextInt();
		}
		catch(InputMismatchException e) {
		    return -1;
		}
		return n;
	}

	public static void main(String args[]) {
		PhoneBookApp pb = new PhoneBookApp();
		pb.run();
	}
}

// 하나의 전화 번호 정보를 가지는 클래스
class Phone {
	private String name; // 이름
	private String address; // 주소
	private String telNum; // 전화 번호

	public Phone(String name, String address, String telNum) {
		this.name = name;
		this.address = address;
		this.telNum = telNum;
	}

	public String getName() {return name;}
	public String getAddress() {return address;}
	public String getTelNum() {return telNum;}

	public String toString() {
		return name + " " + address + " " + telNum;
	}
}
```

```
삽입:0. 삭제:1, 찾기:2, 전체보기:3, 파일 저장:4, 종료:5>>0
이름>>이순신
주소>>목포시
전화번호>> 111-111-1111
```

이제 마지막으로 제네릭 클래스를 직접 만들어 보자. 아래 코드는 제네릭 스택을 구현한 것이다.

```
class GStack<T> { // 제네릭 스택 선언. 제네릭 타입 T
        int tos;
        Object [] stck; // 스택에 저장된 요소의 개수
        public GStack() {
                tos = 0;
                stck = new Object [10];
        }
        public void push(T item) {
                if(tos == 10) // 스택이 꽉 차서 더 이상 요소를 삽입할 수 없음
                        return;
                stck[tos] = item;
                tos++;
        }
        public T pop() {
                if(tos == 0) // 스택이 비어 있어 꺼낼 요소가 없음
                        return null;
                tos--;
                return (T)stck[tos]; // 타입 매개 변수 타입으로 캐스팅
        }
}

public class MyStack {
    public static void main(String[] args) {
        GStack<String> stringStack = new GStack<String>(); // String 타입의 GStack 생성
        stringStack.push("seoul");
        stringStack.push("busan");
        stringStack.push("LA");

        for(int n=0; n<3; n++)
                System.out.println(stringStack.pop()); // stringStack 스택에 있는 3 개의 문자열 팝

        GStack<Integer> intStack = new GStack<Integer>(); // Integer 타입의 GStack 생성
        intStack.push(1);
        intStack.push(3);
        intStack.push(5);

        for(int n=0; n<3; n++)
                System.out.println(intStack.pop()); // intStack 스택에 있는 3 개의 정수 팝
    }
}
```

```
LA
busan
seoul
5
3
1
```

수고하셨습니다. 컬렉션 프레임워크는 실제 코딩 할 때 많이 사용이 되고 있으므로 각각의 특징과 용도를 잘 기억해 두시기 바랍니다. 아래 문제를 배열을 이용하지 않고 컬렉션 프레임워크를 이용해서 코딩 해 보세요.

문제풀기: 1038, 1261, 1420, 1753, 2025, 3004, 4501 실습문제 1번, 3번, 5번, 9번

1753: 코드네임 풀이

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;
public class Main {
 public static void main(String[] args) {
  Scanner scan = new Scanner(System.in);
  ArrayList<String> arr=new ArrayList<String>();

  int a=scan.nextInt();

  for(int i=0;i<a;i++) {
   String s=scan.next();
   arr.add(s);
  }
  arr.iterator();
  for(int i=0;i<a;i++) {
   String st=arr.get(i);
   StringBuffer st2=new StringBuffer(st).reverse();
   System.out.println(st2);
  }
 }
}
```