7차시 정리

```
/*
컬렉션
요소 객체들의 저장소로 요소의 개수에 따라 크기가 자동 조절이 된다. 요소의 삽입, 삭제에 따른 요소의 위치 자동 이동
고정 크기의 배열을 다루는 어려움을 가변 크기의 컬렉션이 해소
컬렉션은 제네릭 기법으로 구현된다.
컬렉션의 요소는 객체만 가능하며 기본 타입 사용 불가하다.(Wrapper로 변환 필요, JDK 1.5부터 자동 박싱, 언박싱을 사용한다.)

JDK 1.5 이전 : 자동 박싱/언박싱이 없으므로 기본 타입을 Wrapper로 다 변환하여 선언 해야 했었다.
v.add(Integer.valueOf(4));
Integer n = (Integer)v.get(0);
int k = n.intValue();

JDK 1.5 이후 : 자동 박싱/언박싱으로 기본 타입 값 사용가능
v.add(4);
int k = v.get(0);

인터페이스
        Collection<E>                    Map<K, V>

Set<E>    List<E>    Queue<E>

클래스
HashSet<E> ArrayList<E> Vector<E> LinkedList<E> HashMap<K, V>

                              Stack><E>
제네릭
JDK 1.5에서 도입을 했으며, 모든 종류의 데이터 타입을 다룰 수 있도록 일반화된 타입
특정 타입만 다루지 않고, 여러 종류의 타입으로 변신할 수 있도록 클래스나 메소드를 일반화시키는 방법
타입 매개 변수 표기방법 : <E>, <K>, <V>
제네틱 클래스 표기방법 : Stack<E>
나중에 제네틱 클래스를 구체화 한 경우 : Stack<Integer>, Stack<String>

Vector<E>

특성
java.util.Vector 에 존재
여러 객체들을 삽입, 삭제, 검색하는 컨테이너 클래스, 길이 제한이 없으며 크기가 커지면 자동으로 길이가 조절
Vector에 삽입 가능한 것 : 객체, null, Wrapper객체(기본타입을 변환)
Vector 맨뒤 혹은 중간에 객체를 삽입할 수 있으며, 객체를 삭제한다면 자동으로 자리가 이동된다.

선언
Vector<Integer> 변수이름 = new Vector<Integer>();

메서드
변수이름.add(E element) : 맨뒤에 추가
변수이름.add(int index, E element) : 인덱스 위치에 삽입
변수이름.capacity() : 백터의 현재 용량 리턴
변수이름.size() : 요소의 개수 리턴
변수이름.remove(int index) : 인덱스 요소 삭제
변수이름.remove(object o) : 객체 o와 같은 첫번째 요소 삭제
변수이름.clear() : 모든 요소 삭제
변수이름.isEmpty() : 벡터가 비어있으면 T
변수이름.get(int index) : 인덱스 값 반환
변수이름.elementAt(int index) : 인덱스 값 반환

오류
Vector<int> : <> 안에는 기본타입이 아닌 객체가 들어가야 한다.

*/
```

```
import java.util.Vector; // Vector<E>를 사용하기 위해서 불러온다.

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());
		System.out.println("벡터의 현재 용량 : " + v.capacity());

		// 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; } // v.elementAt(i) : i번째 인덱스 값을 반환

		System.out.println("벡터에 있는 정수 합 : " + sum);
	}
}

/* 출력

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

*/
```

```
import java.util.Vector; // Vector<E> 를 사용하기 위해 선언

class Point {
	private int x, y; // private로 선언

	public Point(int x, int y) { // 생성자를 통해 private 변수에 값을 저장
		this.x = x; this.y = y;
	}

	public String toString() { // 출력 설정
		return "(" + x + "," + y + ")";
	}
}

public class PointVectorEx {
	public static void main(String[] args) {

		Vector<Point> v = new Vector<Point>(); // Point 객체를 요소로만 가지는 벡터 생성

		// 3 개의 Point 객체 삽입
		v.add(new Point(2, 3));
		v.add(new Point(-5, 20));
		v.add(new Point(30, -8));
		v.remove(1); // 인덱스 1의 Point(-5, 20) 객체 삭제

		// 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
		}
	}
}

/* 출력

(2,3)
(30,-8)

*/
```

```
/*
컬렉션 매개변수로 받는 메소드 만들기
예시) public void 함수이름(Vector<Integer> v)

java 7 이전 : 모든 객체와 형식을 선언해야됨
java 7 이후 : 컴파일러에 타입추론 기능이 추가되어서 다이어몬드 연산자(<>)에 타입 매개변수 생략
java 10 이후 : var 키워드 도입. 지역변수 타입 추론 가능

ArrayList<E>

특성
java.util.ArrayList 에 존재
가변 크기의 배열을 구현한 클래스
삽입 가능한 것 : 객체, null, Wrapper 객체(기본타입을 변환)
리스트의 중간 혹은 맨 뒤에 객체 추가 가능, 임의의 위치에 있는 객체 삭제 가능
벡터와 달리 스레드 동기화의 기능이 없으므로 불안정한 형태를 유지, 개발자가 스레드 동기화 코드를 짜야한다.
즉, 하나의 객체를 사용하고 있는데, 다른 사용자가 들어와서 처리를 할 수 있는 상태.
벡터보단 빠르지만 불안정하다.
<-> 벡터는 느리지만 안정적이다. 즉 하나의 값은 처리가 끝나면 다른 스레드를 처리할 수 있다.
```

메소드
변수이름.add(E element) : 맨뒤에 추가
변수이름.add(int index, E element) : 인덱스 위치에 삽입
변수이름.size() : 요소의 개수 리턴
변수이름.remove(int index) : 인덱스 요소 삭제
변수이름.remove(object o) : 객체 o와 같은 첫번째 요소 삭제
변수이름.clear() : 모든 요소 삭제
변수이름.isEmpty() : 벡터가 비어있으면 T
변수이름.get(int index) : 인덱스 값 반환
변수이름.elementAt(int index) : 인덱스 값 반환
<-> 벡터와 달리 .capacity()메소드가 없다.

*/

```
import java.util.*; // ArrayList<E>를 사용하기 위해 선언

public class ArrayListEx {
	public static void main(String[] args) {

		// 문자열만 삽입가능한 ArrayList 컬렉션 생성
		ArrayList<String> a = new ArrayList<String>();

		// 키보드로부터 4개의 이름 입력받아 ArrayList에 삽입
		Scanner scanner = new Scanner(System.in);
		System.out.println("입력된 4명의 이름중 가장 긴 이름을 출력하겠습니다.");

		for(int i=0; i<4; i++) {
			System.out.print("이름을 입력하세요>>");
			String s = scanner.next(); // 키보드로부터 이름 입력
			a.add(s); // ArrayList 컬렉션에 삽입
		}

		// ArrayList에 들어 있는 모든 이름 출력
		for(int i=0; i<a.size(); i++) {
			// ArrayList의 i 번째 문자열 얻어오기
			String name = a.get(i); // 자동 언박싱
			System.out.print(name + " ");
		}

		// 가장 긴 이름 출력
		int longestIndex = 0;
		for(int i=1; i<a.size(); i++) {
			if(a.get(longestIndex).length() < a.get(i).length())
				longestIndex = i;
		}

		System.out.println("\n가장 긴 이름은 : " + a.get(longestIndex)); // a.get(idx) : idx번째 값을 반환
		scanner.close();
	}
}

/* 출력
입력된 4명의 이름중 가장 긴 이름을 출력하겠습니다.
이름을 입력하세요>>Mike
이름을 입력하세요>>Jane
이름을 입력하세요>>Ashley
이름을 입력하세요>>Helen
Mike Jane Ashley Helen
가장 긴 이름은 : Ashley
*/
```

```
/*

iterator<E> 인터페이스
컬렉션의 순차 검색을 위하여 iterator를 사용한다.
iterator 객체를 이용하여 인덱스 없이 순차적 검색이 가능하다.
Vector<E>, ArrayList<E>, LinkedList<E>가 상속받는 인터페이스

iterator<E> 인터페이스 메소드
객체이름.hasNext() : 다음 요소가 있다면 T
객체이름.next() : 다음 요소 리턴
객체이름.remove() : 마지막으로 리턴된 요소 제거

 */

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()) { // it에 다음 요소가 있다면 반복
			int n = it.next(); // it에 다음 요소를 반환
			System.out.println(n);
		}

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

// 출력
// 5
// 4
// 100
// -1
// 벡터에 있는 정수 합 : 108
```

```
/*
HashMap<K, V>
키(key)와 값(value)의 쌍으로 구정되는 요소를 다루는 컬렉션
java.util.HashMap 에 존재
키와 값이 한쌍으로 삽입, 값을 검색하기 위해서는 키를 이용해서 구해야 한다.
삽입, 검색, 삭제가 빠르다.

메소드
객체이름.clear() : 해시맵의 모든 요소 삭제
객체이름.get(object key) : 키에 대한 값을 반환하며, 값이 없다면 null을 반환
객체이름.isEmpty() : 해시맵이 비어 있으면 T리턴
객체이름.put(K key, V value) : key, value 쌍을 해시맵에 저장
객체이름.remove(Object key) : 입력된 키와 값을 찾아 삭제
객체이름.size() : 해시맵에 포함된 요소의 개수 리턴

 */

import java.util.*; // HashMap<K, V>를 사용하기 위해 선언

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");

		// 영어 단어를 입력받고 한글 단어 검색. "exit" 입력받으면 종료
		Scanner scanner = new Scanner(System.in);

		while(true) {
			System.out.print("찾고 싶은 단어는?");

			String eng = scanner.next();
			if(eng.equals("exit")) { // 문자열은 무조건 equals로 비교
				System.out.println("종료합니다...");
				break;
			}

			String kor = dic.get(eng); //해시맵에서 '키' eng의 '값' kor 검색
			if(kor == null) // key에 대한 값이 없으면 null 리턴
				System.out.println(eng + "는 없는 단어 입니다.");
			else
				System.out.println(kor);
		}
		scanner.close();
	}
}
// 출력
// 찾고 싶은 단어는?baby
// 아기
// 찾고 싶은 단어는?love
// 사랑
// 찾고 싶은 단어는?one
// one는 없는 단어 입니다.
// 찾고 싶은 단어는?exit
// 종료합니다...
```

```
import java.util.*; // // HashMap<K, V>, iterator를 사용하기 위해 선언

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("김성동", 97);
		javaScore.put("황기태", 88);
		javaScore.put("김남윤", 98);
		javaScore.put("이재문", 70);
		javaScore.put("한원선", 99);

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

		// key 문자열을 가진 집합 Set 컬렉션 리턴
		Set<String> keys = javaScore.keySet(); // .keySet() : 해시맵의 모든 키를 담은 Set(K) 컬렉션 리턴
		Iterator<String> it = keys.iterator(); // key 문자열을 순서대로 접근할 수 있는 Iterator 리턴

		// javaScore에 들어 있는 모든 (key, value) 쌍 출력
		while(it.hasNext()) { // it에 다음 요소가 있다면 반복
			String name = it.next(); // it에 다음 요소를 반환
			int score = javaScore.get(name); // name이라는 key값을 가지는 값을 리턴
			System.out.println(name + " : " + score);
		}
	}
}

// 출력
// HashMap의 요소 개수 :5
// 이재문 : 70
// 한원선 : 99
// 김남윤 : 98
// 김성동 : 97
// 황기태 : 88
```

```
// 학생 정보 관리 프로그램

import java.util.*;

class Student { // 학생을 표현하는 클래스
	int id;
	String tel;

	public Student(int id, String tel) { // 생성자
		this.id = id;
		this.tel = tel;
	}

	public int getId() { // id값을 리턴하는 메소드
		return id;
	}

	public String getTel() { // tel를 리턴하는 메소드
		return this.tel;
	}
}

public class HashMapStudentEx {
	public static void main(String[] args) {
		// 학생 이름과 Student 객체를 쌍으로 저장하는 HashMap 컬렉션 생성
		HashMap<String, Student> map = new HashMap<String, Student>();

		// 3 명의 학생 저장
		map.put("황기태", new Student(1, "010-111-1111"));
		map.put("이재문", new Student(2, "010-222-2222"));
		map.put("김남윤", new Student(3, "010-333-3333"));

		// 사용자 입력을 위한 객체 생성
		Scanner scanner = new Scanner(System.in);

		while(true) {
			System.out.print("검색할 이름?");
			String name = scanner.nextLine(); // 사용자로부터 이름 입력

			if(name.equals("exit"))
				break; // while 문을 벗어나 프로그램 종료

			Student student = map.get(name); // 이름에 해당하는 Student 객체 검색
			if(student == null) // 키에 대한 값이 없으면 null을 리턴
				System.out.println(name + "은 없는 사람입니다.");
			else
				System.out.println("id:" + student.getId() + ", 전화:" + student.getTel());
		}

		scanner.close();
	}
}
// 출력
// 검색할 이름?이재문
// id:2, 전화:010-222-2222
// 검색할 이름?김남윤
// id:3, 전화:010-333-3333
// 검색할 이름?오진석
// 오진석은 없는 사람입니다.
// 검색할 이름?exit
```

```
/*

LinkedLink<E>

특성
java.util.LinkedLink 에 존재
List 인터페이스를 구현한 클래스
요소 객체들을 양방향으로 연결되어 관리, 요소 객체는 맨 앞, 맨 뒤에, 혹은 인덱스를 이용하여 중간에 삽입 가능
맨 앞이나 맨 뒤에 요소를 추가하거나 삭제할 수 있어 스택이나 큐로 사용 가능

Collection 클래스

특성
java.util 패키지에 포함.
컬렉션에 대해 연산을 수행하고 결과로 컬렉션을 리턴
모든 메소드는 static 타입

메소드
.sort() .reverse() .max() .min() .binarySearch()

 */

import java.util.*; // Collections를 사용하기 위해 선언
public class CollectionsEx {
	static void printList(LinkedList<String> l) {

		Iterator<String> iterator = l.iterator(); // 인덱스 없이 요소를 차례로 확인하기 위해 선언

		while (iterator.hasNext()) {
			String e = iterator.next();
			String separator; // 문자를 이어주는 연결문자를 저장
			if (iterator.hasNext())
				separator = "->";
			else // 더 이상 값이 없으면 줄바꿈
				separator = "\n";
			System.out.print(e+separator);
		}
	}
	public static void main(String[] args) {
		LinkedList<String> myList = new LinkedList<String>();
		myList.add("트랜스포머");
		myList.add("스타워즈");
		myList.add("매트릭스");
		myList.add(0,"터미네이터");
		myList.add(2,"아바타");

		Collections.sort(myList); // Collection는 static이므로 따로 선언없이 사용, 요소 정렬
		printList(myList); // 정렬된 요소 출력

		Collections.reverse(myList); // Collection는 static이므로 따로 선언없이 사용, 요소의 순서를 반대로
		printList(myList); // 요소 출력

		int index = Collections.binarySearch(myList, "아바타") + 1;
		System.out.println("아바타는 " + index + "번째 요소입니다.");
	}
}
// 출력
// 매트릭스->스타워즈->아바타->터미네이터->트랜스포머
// 트랜스포머->터미네이터->아바타->스타워즈->매트릭스
// 아바타는 3번째 요소입니다.
```

```
/*

제네릭 클래스와 인터페이스
클래스나 인터페이스 선언부에 일반화된 타입 추가
public class MyClass<T>{
        T val; T 타입 변수 val
        void set(T a){ val = a; } T 타입값 a를 지정
        T get(){ return val; } T 타입값 val을 지정
}

구체화
public class MyClass<String>{
        String val; String 타입 변수 val
        void set(String a){ val = a; } String 타입값 a를 지정
        String get(){ return val; } String 타입값 val을 지정
}

구체화의 오류
Vector<int> vi = new Vector<int>(); : 타입 매개 변수에 기본 타입은 사용할 수 없다.
Vector<Integer> vi = new Vector<Integer>();

타입 매개변수
< 와 > 사이에 대문자를 타입 매개변수로 사용, 어떤 문자도 매개변수로 사용가능
E : Element를 의미하며 요소를 표시할때 많이 사용
T : Type를 의미하며 요소를 표시할때 많이 사용
V : Value를 의미하며 요소를 표시할때 많이 사용
K : Key를 의미하며 요소를 표시할때 많이 사용
타입 매개변수가 나타내는 타입의 객체 생성 불가
예) T a = new T();
타입 매개변수는 나중에 실제 타입으로 구체화
 */
```

```
class GStack<T> {
	int tos; Object [] stck;
	public GStack() {
		tos = 0;
		stck = new Object [10];
	}
	public void push(T item) {
		if(tos == 10)
			return;
		stck[tos] = item; tos++;
	}
	public T pop() {
		if(tos == 0)
			return null;
		tos--;
		return (T)stck[tos];
	}
}
public class MyStack {
	public static void main(String[] args) {
		GStack<String> stringStack = new GStack<String>();
		stringStack.push("seoul");
		stringStack.push("busan");
		stringStack.push("LA");

		for(int n=0; n<3; n++)
			System.out.println(stringStack.pop());

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

		for(int n=0; n<3; n++)
			System.out.println(intStack.pop());
	}
}

// 출력
// LA
// busan
// seoul
// 5
// 3
// 1
```

```
/*
제네릭에서 배열의 제한
제네릭 클래스 또는 인터페이스의 비열을 허용하지 않음.
예) GStack<String>[] gs = new GStack<String>[10]; 오류
제네릭 타입의 배열도 허용하지 않음.
예) T[] a = new T[10]; 오류
타입 매개변수의 배열에 레퍼런스는 허용
예) public void myArray(T[] a) {...} 허용
제네릭 메소드 선언 가능
예) static <T> void toStack(T[] a, GStack <T> gs)

제네릭의 장점
컴파일 시에 타입이 결정되어 보다 안전한 프로그래밍 가능
런타임 타입 충돌 문제 방지
ClassCastException 방지
 */
```

```
class GStack<T> {
	int tos;
	Object [] stck;

	public GStack() {
		tos = 0;
		stck = new Object [10];
	}

	public void push(T item) {
		if(tos == 10)
			return;
		stck[tos] = item;
		tos++;
	}

	public T pop() {
		if(tos == 0)
			return null;
		tos--;
		return (T)stck[tos]; // 타입 캐스팅
	}
}

public class GenericMethodExample {
	public static <T> GStack<T> reverse(GStack<T> a) { // T가 타입 매개 변수인 제네릭 메소드

		GStack<T> s = new GStack<T>();

		while (true) {
			T tmp = a.pop(); // 원래 스택에서 요소 하나를 꺼냄

			if (tmp==null) // 스택이 비었음
				break;
			else
				s.push(tmp); // 새 스택에 요소를 삽입
		}
		return s; // 새 스택을 반환
	}
	public static void main(String[] args) {

		// Double 타입의 GStack 생성
		GStack<Double> gs = new GStack<Double>();

		// 5개의 요소를 스택에 push
		for (int i=0; i<5; i++) {
			gs.push(new Double(i));
		}

		gs = reverse(gs);
		for (int i=0; i<5; i++) {
			System.out.println(gs.pop());
		}
	}
}
```

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

class Student{
	final String name;
	final int id;
	final int kor, eng, math; //final을 모두 붙임. private으로 안해도 외부에서 바꿀 수 없음

	public Student(String name, int id, int kor, int eng, int math){ //error: 생성자는 void 안붙임!!!!!!!
		this.name = name;
		this.id = id;
		this.kor = kor;
		this.eng = eng;
		this.math = math;
	}
}

class MMMM {
	public static void main(String[] args) throws Exception {

		Scanner scan = new Scanner(System.in); //학생입력

		ArrayList<Student> Slist = new ArrayList<Student>(); //Student 객체를 넣을 리스트 생성, 학생목록 LinkedList로
만들어도됨

		String _name;
		int _id, _kor, _eng, _math;

		while(true){

			System.out.println("이름 ");
			_name = scan.nextLine();

			if(_name.equals("q")){ //의미적으로 같으려면 equal를 호출해야함
				break;
			}

			System.out.println("학번	국 영 수 ");

			_id = scan.nextInt();
			_kor = scan.nextInt();
			_eng = scan.nextInt();
			_math = scan.nextInt(); // 입력받아서 변수에 저장하기
			scan.nextLine(); //다음 입력을 위해 띄워줘야함

			Slist.add(new Student(_name, _id, _kor, _eng, _math)); //객체를 생성하고 add.

		}
		System.out.println("학번	이름	국어성적 영어성적 수학성적 총점	평균");

		double sum = 0;
		double kor_sum = 0;
		double eng_sum = 0;
		double math_sum = 0;
		double count = 0;
		for(Student S: Slist){

			sum = S.kor + S.math + S.eng;

			kor_sum+=S.kor;
```

```
                        eng_sum+=S.eng;
                        math_sum+=S.math;

                        count+=1;
                        System.out.printf("%6d %4s %4d    %4d   %4d     %6.2f %6.2f\n", S.id, S.name, S.kor, S.eng, S.math,
sum, sum/3.0); //sum이 double 이니까 3.0으로 나눔

                }
                System.out.println("=======================================================");
                System.out.printf("전체평균              %6.2f  %6.2f  %6.2f               %6.2f ", kor_sum/count, eng_sum/count,
math_sum/count, (kor_sum + eng_sum + math_sum)/(count*3.0));

                System.out.println("\n찾고 싶은 학생의 이름을 입력홰 주세요.");
                String findname = scan.next();

                int index_num = 0;
                for(Student S: Slist){
                        if(S.name.equals(findname)) {
                                index_num = 1;
                                double tol_sum = S.kor + S.eng + S.math;
                                System.out.printf("%6d %4s %4d %4d %4d %6.2f %6.2f\n", S.id, S.name, S.kor, S.eng,
S.math, tol_sum, tol_sum/3.0);
                        }
                }
                if (index_num == 0) {
                        System.out.println("\n찾으시는 학생은 존재하지 않습니다.");
                }
        }
}

// 출력
// 이름
// 오진석
// 학번    국 영 수
// 201901 99 99 99
// 이름
// 박소영
// 학번    국 영 수
// 202002 100 100 95
// 이름
// 석진오
// 학번    국 영 수
// 201903 80 69 83
// 이름
// q
// 학번      이름    국어성적 영어성적 수학성적 총점      평균
// 201901   오진석    99      99     99     297.00   99.00
// 202002   박소영   100     100     95     295.00   98.33
// 201903   석진오    80      69     83     232.00   77.33
// =======================================================
// 전체평균          93.00   89.33   92.33            91.56
// 찾고 싶은 학생의 이름을 입력홰 주세요.
// 오진석
// 201901   오진석    99    99    99 297.00   99.00
```