자료구조 실습 보고서

[제 06 주] 재귀 – 성적처리

제출일	2017/04/17
학 번	201000287
소 속	일어일문학과
이 름	유다훈

201000287 일어일문학과 유다훈

1 프로그램 설명서

1 주요 알고리즘 및 자료구조

- 알고리즘
  - 퀵 정렬(Quick Sort)의 개념 이해
    - ◆ 재귀적인 문제 해결 방식을 퀵 정렬을 통해서 알아본다.
    - ◆ 재귀적 표현 방법을 JAVA 에서 구현하는 방법을 알아본다.
  - 파티션(Partition)나누기 의 이해
    - ◆ 퀵 정렬의 핵심인 파티션 나누기를 알아본다.
  - 입력
    - ◆ 학번과 성적을 입력할지 말지의 여부(Yes/No)
      - i Yes 이면 한 학생의 학번과 점수를 입력 받는다.
      - ii 성적이 0 보다 작거나 100 보다 크면 오류 메시지 출력. 입력 무시.
      - iii No 이면 입력 종료.
    - ◆ 최대 학생 수 이상 입력되면 공간 부족 메세지를 내보내고 종료한다.
  - 출력
    - ◆ 성적이 하나도 입력되지 않았으면 아무 일도 하지 않고 종료한다.
    - ◆ 성적이 입력되었으면,
      - i 입력된 한번과 점수와 그에 해당하는 학점을 출력.
      - ii 평균점과 평균 이상인 학생의 학생 정보를 출력
      - iii 마지막에 학점 별 학생수 출력

- 자료구조
  - 학생들을 저장하고 정렬하기 위한 Student 형 배열
  - 알림 및 오류 등의 메세지를 처리하는 enum

1 함수 설명서

Class	AppController			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppController()	없음	없음	_appView 변수를 초기화하는 생성자 메소드
	run()	없음	없음	학생들의 성적순 정렬 프로그램을 실행시키는 메소드
	showMessage(MessageID aMessageID)	에러 메세지 혹은 알림	없음	에러 메세지 혹은 알림에 따라 메세지를 출력하는 메소드
	void inputAndStoreStudents()	없음	boolean	입력받는 점수가 0 점이상이고 100 점이하 일때 Student 객체를 생성하여 ban 클래스에 저장하는 메소드
	void showStatics()	없음	없음	성적 입력이 종료되면 입력 결과와 각 학점 당 학생수를 출력
	void showStudentsSortedByScore	없음	없음	성적 순으로 정렬된 학생들의 점수를

				출력한다.

Class	AppView			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppView()	없음	없음	생성자 메소드 값을 입력받는 스캐너 생성
	int inputInt()	없음	정수값	입력 받은 정수를 리턴하는 메소드
	String inputString()	없음	문자열값	입력 받은 문자열을 리턴하는 메소드
	void outputMessage(String aMessage)	문자열	없음	전달 받은 문자열을 출력하는 메소드
	void outputAverageScore(float anAverageScore)	평균 점수	없음	전달 받은 학생의 평균 점수 값을 출력
	void outputNumberOfStudnetsAboveAverage(int aNumber)	평균 점수 이상 점수를 가진 학생 수	없음	전달 받은 학생수를 출력
	void outputMaxScore(int aMaxScore)	최고점	없음	전달 받은 성적의 최고점을 출력
	void outputMinScore(int aMinScore)	최저점	없음	전달 받은 성적의 최저점을 출력
	void outputGradeCountFor(char aGrade, int aCount)	학점, 인원수	없음	전달받은 학점 당 학생 수를 출력
	voidt outputStudentInfo(int aScore)	점수	없음	전달받은 학생의 점수를 출력
	boolean inputDoesContinueToInputNextStudent()	없음	boolean	성적 입력 여부를 묻는 메소드. y 를 입력하면 값은 true
	int inputScore()	없음	정수 값	점수를 입력받아 리턴하는 메소드

Class	Student			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Student(int givenScore)	점수 값	없음	주어지는 점수값으로 Student 객체를 생성하는 생성자 메소드
	int score()	없음	정수 값	학생의 점수를 리턴하는 메소드
	void setScore(int newScore)	새로운 점수값	없음	주어지는 값으로 학생의 점수를 설정하는 메소드

Class	GradeCounter			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	void count(char aGrade)	캐릭터형 등급	없음	입력받은 성적 등급에 따라 해당 학생수를 카운트하는 메소드
	int numberOfA()	없음	정수 값	A 등급의 학생 수 리턴
	int numberOfB()	없음	정수 값	B 등급의 학생 수 리턴

	int numberOfC()	없음	정수 값	C 등급의 학생 수 리턴
	int numberOfD()	없음	정수 값	D 등급의 학생 수 리턴
	int numberOfF()	없음	정수 값	F 등급의 학생 수 리턴

Class	Ban			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Ban(int givenMaxSize)	배열의 최대크기	없음	주어진 값으로 배열의 최대 크기를 설정하는 생성자 메소드
	int size()	없음	배열 내의 원소의 갯수	배열 안에 가지고 있는 원소의 갯수를 리턴하는 메소드
	int maxSize()	없음	배열이 가질 수 있는 원소의 최대 갯수	배열이 가질 수 있는 원소의 최대 갯수를 리턴하는 메소드
	boolean isEmpty()	없음	배열 내부가 비었으면 true 안 비었으면 false	배열 내부가 비어 있는지 여부 리턴
	boolean isFull()	없음	배열 내부가 꽉 차있으면 true 꽉 차있지 않으면 false	배열 내부가 꽉 차 있는지의 여부 리턴
	boolean doesContain(E anElement)	특정 타입의 원소	존재하면 true 없으면 false	집합 내부에 입력받은 특정 타입의 원소가 존재하는지 여부를 검사하여 리턴
	boolean add(Student aScore)	특정 타입의 원소	true	배열 내부가 꽉 차 있지 않으면 배열에 원소를 추가하고, 배열 내 원소카운터를 증가시키는 메소드
	Student elementAt(int aPosition)	배열 내 특정 위치	Student 타입의 원소	입력받은 정수값을 인덱스로 하여 배열의 인덱스값에 존재하는 Student 형 원소를 리턴한다.
	void sortStudentByScore()	없음	없음	배열 내에서 최소값 위치를 찾아 배열의 맨 끝으로 보낸 후 퀵정렬을 실행하는 메소드
	int minScore()	없음	최소값	재귀적 최소값 구하기를 실행하는 메소드
	int maxScore()	없음	최대값	재귀적 최대값 구하기를 실행하는 메소드
	float averageScore()	없음	평균값	학생들의 평균 점수값을 리턴하는 메소드
	int numberOfStudentAboveAverage()	없음	평균 점수 이상의 학생 수	평균 이상의 점수를 가진 학생 수를 리턴하는 메소드
	GradeCounter countGrades()	없음	등급 별 학생수를 저장한 GradeCounter 형	등급 별 학생 수를 계산하고 그 계산한 값이 담긴 GradeCounter 형 자료를 리턴하는 메소드
	char scoreToGrade(int aScore)	없음	등급	점수를 등급화하는 메소드
	void swap(int postitionA, int position)	배열 내 자리를 바꿀	없음	전달받은 두 값에 해당하는 배열 내 원소들의 자리를 바꾼다.

		원소들의 위치		
	void quickSortRecursively(int left, int right)	퀵정렬을 실행할 배열 원소의 범위	없음	전달 받은 두 값을 이용하여 중간 값을 생성한 후, 자신(메소드)를 다시 호출하여 반복적으로 중간값을 생성하는 메소드
	int partition(int left, int right)	원소의 범위를 지정하는 값	중간 값	전달 받은 두 값을 이용하여 배열 내에서 범위를 지정하고 오른쪽값이 왼쪽값보다 크다면 자리를 바꿔주어 정렬을 실행하는 메소드
	float sumOfScoresRecursively(int left, int right)	값을 더할 배열의 위치	배열의 총점	자기 자신을 이용하여 배열 내 원소의 합을 구하는 메소드
	int maxScoreRecursively(int left, int right)	최대값을 찾기 위해 배열의 원소 위치를 지정해줄 값	최대값	왼쪽과 오른쪽이 같은 값이라면 배열의 가장 아래의 점수값을 리턴. 그렇지 않으면 중간값을 찾아 자기 자신을 이용해 재귀적으로 최대값을 찾는 메소드
	int minScoreRecursively(int left, int right)	최소값을 찾기 위해 배열의 원소 위치를 지정해줄 값	최소값	왼쪽과 오른쪽이 같은 값이라면 배열의 가장 아래의 점수 값을 리턴. 그렇지 않으면 자기 자신을 이용해 재귀적으로 최소값을 찾는 메소드.

2 종합 설명서

- 재귀적인 방법으로 최대값과 최소값, 정렬하는 기능을 제공하는 메소드를 구현할 수 있다.
- 학생들의 점수를 입력하여 저장한다.
- 입력된 점수들을 바탕으로 각 등급별 학생 수를 계산한다.
- 학생들의 점수를 높은 점수순으로 정렬하여 출력한다.

201000287 일어일문학과 유다훈

2 프로그램 장단점 분석

- 장점
  - 퀵 정렬 방법을 이용하여 입력 받은 데이터가 많으면 많아질수록 다른 정렬 방법에 비하여 빠르게 정렬할 수 있다.
  - 입력 받은 점수를 바탕으로 각 등급별 학생 수가 몇명인지 계산해볼 수 있다.
  - 입력 받은 점수를 높은 점수 순으로 정렬할 수 있다.

- 단점
  - 재귀적 개념에 대해 익숙하지 않으면 프로그램 내부 알고리즘을 이해하기 어려울 수 있다.
  - 최대값, 최소값, 배열 내 원소의 합계 구하기는 재귀적인 방법을 쓰지않더라도 이해하기 쉬운 방법으로 설계할 수 있다.

3 실행 결과 분석

1 입력과 출력

프로그램 실행 및 성적 입력

```
<<성적 처리를 시작합니다.>>
성적을 입력하려면 'Y' 또는 'y'를, 종료하려면 다른 아무 키나 치시오: y
점수를 입력하시오: 82
```

입력 - 100 보다 큰 수 입력 시 오류문 출력

```
성적을 입력하려면 'Y' 또는 'y'를, 종료하려면 다른 아무 키나 치시오: y
점수를 입력하시오: 102
ERROR : 0보다 작거나 100보다 커서, 정상적인 정수가 아닙니다.
```

입력 - 0 보다 작은 수 입력시 오류문 출력

```
성적을 입력하려면 'Y' 또는 'y'를, 종료하려면 다른 아무 키나 치시오: y
점수를 입력하시오: -1
ERROR : 0보다 작거나 100보다 커서, 정상적인 정수가 아닙니다.
```

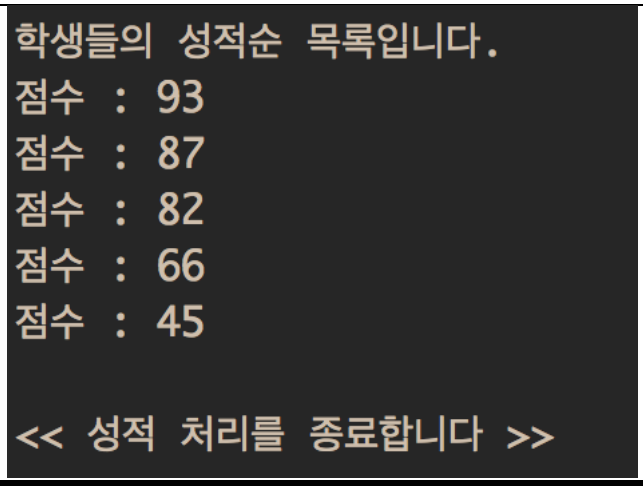
성적 입력 종료 및 안내문 출력

```
성적을 입력하려면 'Y' 또는 'y'를, 종료하려면 다른 아무 키나 치시오: n
[성적 입력을 종료합니다.]
```

출력 - 입력받은 점수들의 평균 점수, 평균 이상 학생 수, 최고점, 최저점, 각 등급 당 학생 수

```
평균 점수는 74.6 입니다.
평균 이상인 학생은 모두 3 명 입니다.
최고점은 93 점 입니다.
최저점은 45 점 입니다.
A 학점은 모두 1 명 입니다.
B 학점은 모두 2 명 입니다.
C 학점은 모두 0 명 입니다.
D 학점은 모두 1 명 입니다.
F 학점은 모두 1 명 입니다.
```

출력 - 학생들의 성적 순 출력 및 성적 입력/정렬 프로그램 종료 안내



2 결과 분석

- 학생들의 점수를 받아 평균 점수 및 평균보다 높은 점수를 받은 학생의 수, 최고점, 최저점, 학점(등급) 당 학생 수 및 높은 성적 순으로의 점수 출력을 할 수 있다.
- 입력한 학생 숫자가 매우 큰 수가 아니므로 퀵 정렬과 다른 정렬 방법간의 차이점을 알 순 없다.
- 만일, 해당 학급으로 학생이 전학을 와서 배열의 최대값보다 학생수가 많게 된다면 배열을 이용한 방법은 적절하진 않다.

3 생각해보기

- 학번 등 학생 정보를 추가하여 더 입력할 것들이 있을 경우, 프로그램에서는 Student 형에서의 클래스 변수의 추가가 이루어져야 한다. (학번, 이름, 주소 등등)
- 이번 과제를 풀면서 오히려 최대값, 최소값 찾기 및 합계구하기는 재귀적인 방법을 통하지 않고 설계하는 것이 이해하기 좀 더 수월한 것 같았다. 즉 방법 혹은 상황에 따라 재귀적이지 않은 방법과 재귀적인 방법을 적절하게 구현해야한다.
  - 무조건 재귀적 문제해결 방법이 좋은 것은 아니다.
- minScoreRecursively 의 내부에서 minScore = minScoreRecursively(left+1, right) 방식으로 재귀적인 문제해결 방법을 사용하였다.
  - 문제에서 크기를 (N-1)로 줄이는 재귀함수를 작성하라고 하였는데, 배열이 왼쪽에서 오른쪽으로 나열된 평행한 구조라고 가정하였을 때, 왼쪽에서 1 증가한 값은 왼쪽에서 오른쪽으로 한 칸 이동한 값이고, 이것은 결국 배열 N 에서 다시 N-1 로 크기를 줄이는 것과 같기 때문이다.
- MaxScoreRecursively 의 내부에서

```
mid = (left + right) /2;
maxScoreOfLeft = maxScoreRecursively(left, mid);
maxScoreOfRight = maxScoreRecursively(mid+1, right);
```

로 구현하였는데, 이것은 중간 점을 구해서 배열을 둘로 쪼갠 후 다시 재귀적인 방법으로 최대값을 찾는 방법이다. 계속해서 중간 값을 구하고, 중간 값의 양 옆의 값을 swap 해줌으로 인해 파티션을 나누어 정렬한다.

- 결론

- 모든 경우에서 재귀적인 문제해결 방법이 좋은 것은 아니다.
- 값이 큰 경우 퀵 정렬을 하는 것이 일정한 배열을 빠르게 정렬한다.
- 재귀적인 방법은 n의 크기가 커질수록 코드를 이해하기 어렵다.

4 소스 코드

Class	AppController

```
public class AppController {
	private AppView _appView ;
	private Ban _ban;

	public AppController() {
		this._appView = new AppView();
	}
	public void run() {
		this.showMessage(MessageID.Notice_StartProgram);

		this.inputAndStoreStudents();

		if(this._ban.isEmpty()) {
			this.showMessage(MessageID.Error_NoInputScores);
		} else {
			this.showStatics();

			this._ban.sortStudentsByScore();
			this.showStudentsSortedByScore();
		}
		this.showMessage(MessageID.Notice_EndProgram);
	}


	private boolean inputAndStoreStudents() { //Ban 객체에 저장
		this.showMessage(MessageID.Notice_StartMenu);

		int score;
		boolean storingAStudentWasSuccessful = true;
		this._ban = new Ban();

		while(storingAStudentWasSuccessful &&
this._appView.inputDoesContinueToInputNextStudent()) {
			score = this._appView.inputScore();

			if(score < 0 || score > 100) {
				this.showMessage(MessageID.Error_InvalidScore);
			} else {
				Student aStudent = new Student(score);
				this._ban.add(aStudent);
			}
		}
		this.showMessage(MessageID.Notice_EndMenu);
		return storingAStudentWasSuccessful;

	}
	private void showStatics() { //Ban 이 성적을 처리한 후 그 결과를 얻어서 출력
		this._appView.outputAverageScore(this._ban.averageScore());

	this._appView.outputNumberOfStudentsAboveAverage(this._ban.numberOfStudentAboveAverage());
		this._appView.outputMaxScore(this._ban.maxScore());
		this._appView.outputMinScore(this._ban.minScore());
```

```
                GradeCounter gradeCounter = this._ban.countGrades();
                //학점 별 학생수는 Ban 객체로 부터 GradeCounter 객체형태로 얻는다
                this._appView.outputGradeCountFor('A', gradeCounter.numberOfA());
                this._appView.outputGradeCountFor('B', gradeCounter.numberOfB());
                this._appView.outputGradeCountFor('C', gradeCounter.numberOfC());
                this._appView.outputGradeCountFor('D', gradeCounter.numberOfD());
                this._appView.outputGradeCountFor('F', gradeCounter.numberOfF());

        }

        private void showStudentsSortedByScore() { //성적순으로 정렬된 학생 정보를 출력한다.
                this.showMessage(MessageID.Show_SortedStudentList);

                for(int position = 0; position < this._ban.size(); position++) {
                        this._appView.outputStudentInfo(this._ban.elementAt(position).score());
                }

        }



        private void showMessage(MessageID aMessage) {
                switch(aMessage) {
                case Notice_StartProgram:
                        this._appView.outputMessage("<<성적 처리를 시작합니다.>>₩n");
                        break;
                case Notice_StartMenu:

                        break;
                case Notice_EndMenu:
                        this._appView.outputMessage("[성적 입력을 종료합니다.]₩n₩n");
                        break;
                case Notice_EndProgram:
                        this._appView.outputMessage("₩n<< 성적 처리를 종료합니다 >>");
                        break;
                case Error_InvalidScore:
                        this._appView.outputMessage("ERROR : 0 보다 작거나 100 보다 커서, 정상적인
정수가 아닙니다.₩n");
                        break;
                case Error_NoInputScores:
                        this._appView.outputMessage("성적을 입력해주세요.");
                        break;
                case Error_WrongMenu:
                        this._appView.outputMessage("잘못된 메뉴입니다.");
                        break;
                case Show_SortedStudentList:
                        this._appView.outputMessage("₩n 학생들의 성적순 목록입니다.₩n");
                        break;
                        default:
                                break;
                }
        }
}
```

Class	AppView

```
import java.util.Scanner;

public class AppView {
	private Scanner _scanner;

	public AppView(){
		this._scanner = new Scanner(System.in);
	}
	public int inputInt() { //점수를 입력받아 정수형으로 변환하여 리턴
		return Integer.parseInt(this._scanner.nextLine());
	}
	public String inputString() { //문자열 입력을 리턴
		return this._scanner.nextLine();
	}
	public boolean inputDoesContinueToInputNextStudent() {
		//다음 학생을 계속 입력받는 메소드

		char answer;
		System.out.print("성적을 입력하려면 'Y' 또는 'y'를, 종료하려면 다른 아무 키나 치시오: ");
		answer = this.inputString().charAt(0); //문자열중 맨 첫번째 알파벳 저장

		if((answer == 'Y') || (answer =='y')) {
			return true;
		} else {
			return false;
		}
	}

	public int inputScore() { //점수 입력받는 메소드
		int score;
		System.out.print("점수를 입력하시오: ");
		score = this.inputInt();
		return score;
	}

	public void outputMessage(String aMessageString){
		System.out.print(aMessageString);

	}
	public void outputAverageScore(float anAverageScore){ //평균 점수 출력
		System.out.println("평균 점수는 "+ anAverageScore + " 입니다.");

	}
	public void outputNumberOfStudentsAboveAverage(int aNumber){ //평균이상 학생수
		System.out.println("평균 이상인 학생은 모두 " + aNumber + " 명 입니다.");

	}
	public void outputMaxScore(int aMaxScore){ //최고점
		System.out.println("최고점은 " + aMaxScore + " 점 입니다.");

	}
	public void outputMinScore(int aMinScore){ //최저점
		System.out.println("최저점은 " + aMinScore + " 점 입니다.");

	}
	public void outputGradeCountFor(char aGrade, int aCount){
```

```
                System.out.println(aGrade + " 학점은 모두 " + aCount + " 명 입니다.");
                //학점 당 학생수
        }
        public void outputStudentInfo(int aScore) { //성적순 목록의 점수반영.
                System.out.println("점수 : " + aScore);

        }

}
```

Class	Ban

```
public class Ban { //학생들의 점수를 계산할 클래스
        private static final int DEFAULT_MAX_SIZE = 100;

        private int _maxSize;
        private int _size;
        private Student[] _elements;

        public Ban() {
                this (DEFAULT_MAX_SIZE);
        }

        public Ban(int givenMaxSize) {
                this._maxSize = givenMaxSize;
                this._elements = new Student[givenMaxSize]; //주어진 수 만큼 배열생성
        }

        public int maxSize() {
                return this._maxSize;
        }
        public int size() {
                return this._size;
        }

        public boolean isEmpty() {
                return (this._size == 0);
        }
        public boolean isFull() {
                return (this._size == this._maxSize);
        }
        public boolean add(Student aScore) {
                if(this.isFull()) { //만약 꽉 찼으면
                        return false; //넣지 말기

                } else{//꽉안찼으면
                        this._elements[this._size] = aScore; //size 번째 공간에 코인을 넣기
                        this._size++; //사이즈 증가
                        return true;
                }
        }
        public Student elementAt(int aPosition) {
                return this._elements[aPosition]; //해당 번호의 배열 내부 학생을 반환
        }
```

```
	public void sortStudentsByScore() { //정렬하기
		int size = this._size;

		if( size >= 2) {
			//최소값 위치찾기
			int minLoc = 0;
			for( int i = 1; i < size; i++) {
				if(this._elements[i].score() < this._elements[minLoc].score()) {
					minLoc = i;
				}
			}
			this.swap(minLoc, size-1); //최소값을 원소구간의 맨 끝으로 보냄.
			quickSortRecursively(0, size-2);//정렬시작
		}

	}
	public int minScore() {
		int left = 0;
		int right = this._size-1;
		return this.minScoreRecursively(left, right);

	}
	public int maxScore() {
		int left = 0;
		int right = this._size-1;
		return this.maxScoreRecursively(left, right);
	}
	public float averageScore() {
		float sumOfScore = (float) sumOfScoresRecursively(0, this._size-1);
		float average = sumOfScore / (float) this._size;

		return average;

	}
	public int numberOfStudentAboveAverage() { //평균 이상인 학생수
		float average = averageScore();
		float score;
		int numberOfStudentsAboveAverage = 0;

		for(int i = 0; i< this._size; i++) {
			score = (float) this._elements[i].score();
			if(score >= average) {
				numberOfStudentsAboveAverage++;
			}
		}
		return numberOfStudentsAboveAverage;

	}
	public GradeCounter countGrades() { //학점별 학생수 계산
		char currentGrade;
		GradeCounter gradeCounter = new GradeCounter();

		for (int i = 0; i < this._size; i++) {
			currentGrade = this.scoreToGrade(this._elements[i].score()); //점수를 등급화한
값을 저장
			gradeCounter.count(currentGrade);
		}
```

```
                return gradeCounter;

        }
        private char scoreToGrade(int aScore) {
                if(aScore >= 90) {
                        return 'A';
                } else if (aScore >= 80) {
                        return 'B';
                } else if (aScore >= 70) {
                        return 'C';
                } else if (aScore >= 60) {
                        return 'D';
                } else {
                        return 'F';
                }

        }
        private void swap(int positionA, int positionB) {
                Student temp = this._elements[positionA];
                this._elements[positionA] = this._elements[positionB];
                this._elements[positionB] = temp;

        }
        private void quickSortRecursively(int left, int right) {
                if (left< right) {
                        int mid = this.partition(left, right);
                        this.quickSortRecursively(left, mid-1);
                        this.quickSortRecursively(mid+1, right);
                }

        }
        private int partition (int left, int right) {
                int pivot = left;
                int toRight = left;
                int toLeft = right+1;
                do {
                        do{toRight++;} while (this._elements[pivot].score() >
this._elements[toRight].score());
                        do{toLeft--;} while(this._elements[pivot].score() < this._elements[pivot].score());
                        if(toRight < toLeft) {
                                swap(toRight, toLeft);
                        }

                } while (toRight <toLeft);
                swap(pivot, toLeft);
                return pivot;

        }
        private float sumOfScoresRecursively(int left, int right) {
                if (left > right) {
                        return 0;
                } else {
                        return (this._elements[left].score() + this.sumOfScoresRecursively(left+1, right));
                }

        }
        private int maxScoreRecursively(int left, int right) {
                int maxScoreOfLeft;
```

```
                int maxScoreOfRight;
                int mid;
                if(left == right) {
                        return this._elements[left].score();
                } else {
                        mid = (left + right) /2;
                        maxScoreOfLeft = maxScoreRecursively(left, mid);
                        maxScoreOfRight = maxScoreRecursively(mid+1, right);

                        if(maxScoreOfLeft >= maxScoreOfRight) {
                                return maxScoreOfLeft;
                        } else
                                return maxScoreOfRight;
                }

        }
        private int minScoreRecursively(int left, int right) {
                int minScore ;
                if (left == right) {
                        return this._elements[left].score();
                } else {
                        minScore = minScoreRecursively(left+1, right);
                        if(this._elements[left].score() <= minScore) {
                                return this._elements[left].score();
                        } else {
                                return minScore;
                        }
                }


        }


        }
```

Class	GradeCounter

```
public class GradeCounter {
        private int _numberOfA;
        private int _numberOfB;
        private int _numberOfC;
        private int _numberOfD;
        private int _numberOfF;

        public void count(char aGrade) {
                switch (aGrade) {
                case 'A':
                        this._numberOfA++;
                        break;
                case 'B':
                        this._numberOfB++;
                        break;
                case 'C':
```

```
                                this._numberOfC++;
                                break;
                        case 'D':
                                this._numberOfD++;
                                break;
                        default:
                                this._numberOfF++;
                                break;

                        }

        }

        public int numberOfA() {
                return this._numberOfA;
        }
        public int numberOfB() {
                return this._numberOfB;

        }
        public int numberOfC() {
                return this._numberOfC;

        }
        public int numberOfD() {
                return this._numberOfD;
        }
        public int numberOfF() {
                return this._numberOfF;
        }

}
```

Class	Student

```
public class Student {
        private int _score;

        public Student(int givenScore) {
                this._score = givenScore;
        }

        public int score() {
                return this._score;
        }
        public void setScore(int newScore) {
                this._score = newScore;
        }
}
```