자료구조 실습 보고서

[제07주] 성적처리 : 재귀(Recursion)

2021년 04월 19일

201702039 오명주

1. 프로그램 설명서

(1) 프로그램의 전체 설계 구조

➔ MVC (Model – View – Controller) 구조

Model : 프로그램이 "무엇"을 할 것인지 정의. 사용자의 요청에 맞는 알고리즘을 처리하고 DB와 상호작용하여 결과물을 산출하고 Controller에게 전달.

View : 화면에 무엇인가를 "보여주기 위한" 역할. 최종 사용자에게 "무엇"을 화면으로 보여줌.

Controller : 모델이 "어떻게" 처리할 지 알려주는 역할. 사용자로부터 입력을 받고 중개인 역할. Model과 View는 서로 직접 주고받을 수 없음. Controller을 통해 이야기함.

➔ 리스트 성능 비교 프로그램에서의 각 클래스 별 MVC 구조 역할

Model :

- UnsortedArrayList : ArrayList로 구현된 정렬되지 않은 리스트
- Ban : 학생 배열을 받을 학급을 의미하는 클래스. 학생 성적 통계를 담당한다.
- Student : 학생 객체 생성할 수 있는 클래스. 객체 비교를 위한 compareTo 함수가 존재한다.
- GradeCounter : 성적에 따른 학점을 정하고 학생 수를 관리한다.
- Iterator : 반복자 인터페이스. 배열에서 반복하여 처리할 때 사용한다.

View :

- AppView : 프로그램의 입/출력을 담당한다.

Controller :

- AppController : Model을 통해 생성된 결과물을 AppView를 통해 출력한다.

(2) 함수 설명서

➔ 주요 알고리즘

(1) 재귀를 이용하여 Lowest, Highest 값 반환

```
private Student lowestRecursively(int left, int right) {
    if (left == right) { // 모든 값을 다 비교한 경우
        return this.elementAt(left); // left를 반환
    } else {
        Student lowestFromRights = lowestRecursively(left + 1, right); // left와 left+1, left+2, left+3 .. 를 계속 비교
        if (lowestFromRights.compareTo(this.elementAt(left)) <= 0) { // 더 작은 값을 반환
            return lowestFromRights; // lowestFromRights가 작으면 이것을 반환
        } else {
            return this.elementAt(left); // left번째가 작다면 이것을 반환
        }
    }
}
```

학생 성적 배열에서 가장 낮은 성적의 값을 출력할 때 사용하는 lowestRecursively 함수.

- 재귀를 이용하여 구현하였다.
- Left와 Right가 같다면 해당 element를 반환한다. (탈출 조건)
- 재귀를 통해 Right와 Right-1을 비교한 후 더 작은 값을 Right-2와 비교, … 이를 반복하여 Left 값과 마지막으로 비교하여 가장 작은 값을 반환 받게 된다.
- 결국 (Left)와 (나머지 중 가장 작은 값)을 비교한 결과 더 작은 값을 반환하는 것이다.

```
private Student highestRecursively(int left, int right) {
    if (left == right) { // 모든 값을 다 비교한 경우
        return this.elementAt(left); // left를 반환
    } else {
        Student highestFromRights = highestRecursively(left + 1, right); // left를 left+1, left+2 .. 비교
        if (highestFromRights.compareTo(this.elementAt(left)) >= 0) { // 더 큰 값을 반환
            return highestFromRights; // highestFromRights 가 같거나 크다면 반환
        } else {
            return this.elementAt(left); // left 값이 더 크다면 반환
        }
    }
}
```

이는 성적 배열에서 가장 높은 성적을 구할 때도 동일하다. Right와 Right-1 인덱스의 값을 먼저 비교하여 큰 값을 Right-2와 비교, Right-3과 비교 …. 반복하여 Left 값과 비교하여 가장 큰 값을 반환한다.

(2) 재귀를 이용하여 Sum 값 반환

```
private int sumOfScoreRecursively(int left, int right) {
    int mid = (left + right) / 2; // 중간지점을 설정
    if (left == right) { // 처음과 끝이 같으면
        return this.elementAt(left).score(); // left의 스코어를 반환
    } else {
        int leftSum = this.sumOfScoreRecursively(left, mid); // leftSum은 left ~ mid 를 재귀적으로 합한 것
        int rightSum = this.sumOfScoreRecursively(mid + 1, right); // rightSum은 mid + 1 ~ right를 재귀적으로 합한 것
        return (leftSum + rightSum); // leftSum + rightSum을 반환한다
    }
}
```

학생 성적 배열에서 각각의 성적에 대한 Sum 값을 구하여 반환하는 sumOfScoreRecursively 함수.

- 중간 위치인 mid를 설정한다.
- Left와 Right가 같다면 해당 score를 반환한다. (탈출 조건)
- 배열을 반으로 나누어 각각의 배열의 합을 구하는 방식이다. 재귀를 통해 배열을 쪼개어 하나의 배열이 될 때까지 반복하여 해당 값을 반환하여 Left 값과 Right 값을 더한다.
- mid를 중심으로 leftSum은 left-mid까지 값의 합을, rightSum은 (mid+1)-right까지 값의 합을 구하여 더하는 형태이다.

(3) 재귀를 이용한 학생 성적 Sort

```
// 학급의 학생들을 성적 순으로 정렬한다
public void sortByScore() {
    if (this.size() > 1) { // 배열이 1개라도 있으면
        int maxLoc = 0; // 최대값 위치를 0으로 초기화
        for (int i = 1; i < this.size(); i++) { // 배열 끝까지 반복
            if (this.elementAt(i).score() > this.elementAt(maxLoc).score()) { // i번째 성적이 maxLoc의 성적보다 크다면
                maxLoc = i; // maxLoc은 i로 설정한다
            }
        }
        this.swap(maxLoc, this.size() - 1); // 최대값인 maxLoc을 맨 뒤로 보낸다.
        this.quicksortRecursively(0, this.size() - 2); // 최대값을 뺀 0 ~ this.size()-2 를 퀵정렬 한다.
    }
}
```

Quick Sort를 이용하여 학급의 학생들을 성적 순으로 정렬하는 sortByScore 함수.

- 학생 배열에 성적이 하나라도 존재할 경우 수행한다.
- 최대값 위치를 찾아 가장 마지막으로 보낸다.
- 인덱스 0부터 size-2까지 퀵 정렬을 수행한다.

```
private void quicksortRecursively(int left, int right) { // 퀵정렬
    if (left < right) { // left<right이면
        int mid = this.partition(left, right); // 파티션 후의 pivot 위치
        this.quicksortRecursively(left, mid - 1); // 나누어진 반을 다시 퀵정렬한다.
        this.quicksortRecursively(mid + 1, right); // 나누어진 반을 다시 퀵정렬한다.
    }
}
```

퀵 정렬은 다음과 같이 수행한다. 반을 나누어 퀵 정렬을 수행한다. Mid 변수에는 partition이 끝난 후 pivot의 위치가 반환된다.

```
private int partition(int left, int right) {
    int pivot = left; // pivot을 left로 설정
    int toRight = left; // 오른쪽으로 갈 toRight는 left 위치에 지정
    int toLeft = right + 1; // 왼쪽으로 갈 toLeft는 right+1 위치에 지정
    do {
```

Partition 함수를 확인하면 pivot을 기준으로 삼는데, 초기에는 left값으로 설정한다. 오른쪽으로 갈 toRigt는 Left 위치에 지정하고 왼쪽으로 갈 toLeft는 right+1 위치에 지정한다. (do-while을 통해 먼저 right--을 실행하므로 +1을 하여 위치를 설정한다)

```
do {
    toRight++;
} while (this.elementAt(toRight).score() < this.elementAt(pivot).score()); // Left에서 Right로 갈 score 위치 선정
do {
    toLeft--;
} while (this.elementAt(toLeft).score() > this.elementAt(pivot).score()); // Right에서 Left로 갈 score 위치 선정
if (toRight < toLeft) { // toRight < toLeft라면
    this.swap(toRight, toLeft); // 두개의 배열을 바꾼다.
```

- toRight 위치를 설정한다. pivot과 비교하여 pivot보다 작으면 반복해서 행하며 pivot보다 클 경우 반복문을 빠져나온다.
- toLeft 위치를 설정한다. pivot과 비교하여 pivot보다 크면 반복해서 행하며 pivot보다 작으면 반복문을 빠져나온다.
- swap 함수를 통해 두개의 값을 바꿔준다.

```
} while (toRight < toLeft); // toRight > toLeft가 되는 순간 탈출
this.swap(pivot, toLeft); // pivot과 toLeft를 바꾼다.
return toLeft; // pivot 위치가 toLeft 이다.
```

toRight 인덱스가 toLeft 인덱스보다 작아지면 (mid 부분에서 교차되면) 반복문을 빠져나온다. pivot을 중간위치로 설정해주고 pivot을 반환한다. 해당 함수는 mid를 기준으로 재귀적으로 반복되어 실행된다.

(4) iterator을 이용한 학생 리스트 출력

```
// 학생들의 성적을 순서대로 출력하는 함수
private void showStudentSortedByScore() {
    AppView.outputLine("");
    AppView.outputLine("[학생들의 성적순 목록]");
    this.ban().sortByScore(); // sortByScore를 통해 정렬

    Iterator<Student> iterator = this.ban().iterator(); // 반복자 생성
    Student student = null;
    while (iterator.hasNext()) { // hasnext()가 null이 아닌동안 반복
        student = iterator.next(); // 다음 student를 저장
        AppView.outputScore(student.score()); // student의 score 출력
    }
}
```

학생 성적을 sort 하여 iterator을 이용해 순서대로 출력하는 showStudentSortedByScore 함수.

- iterator 객체를 생성하고 출력에 이용할 student 객체도 생성하여 초기화한다.
- hasNext를 가져와서 있는 경우 반복하여 student에 해당 score을 저장하여 AppView의 출력 문을 통해 출력한다.

(3) 종합 설명서

➔ 프로그램 실행 순서대로 설명해보자.

```
public class _DS07_201702039_오명주 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        AppController appController = new AppController() ;
        // AppController가 실질적인 main class 이다
        appController.run() ;
        //여기 main() 에서는 앱 실행이 시작되도록 해주는 일이 전부이다
```

main에서 AppController 의 객체를 생성하여 run 한다. 프로그램 실행.

```
public void run() {
    AppView.outputLine("");
    AppView.outputLine("<<< 학급 성적 처리를 시작합니다 >>>");

    this.setBan(new Ban(AppController.BAN_CAPACITY)); // capacity 설정
    this.inputAndStoreStudents(); // 성적 입력받아서 Student 객체로 저장
    if (this.ban().isEmpty()) {
        AppView.outputLine("");
        AppView.outputLine("(경고) 입력된 성적이 없습니다.");
```

AppController 의 run 함수를 통해 capacity를 설정하고 inputAndStoreStudent 함수를 이용하여 사용자로부터 성적 입력 여부와, 학생 성적을 입력 받아 Student 객체에 저장한다. 입력한 성적이 없다면 없다는 경고를 출력한다.

```
private void inputAndStoreStudents() {
    AppView.outputLine("");
    boolean storingAStudentWasSuccessful = true; // 정상적으로 처리가 되었는지 확인하는 변수
    while (storingAStudentWasSuccessful && AppView.doesContinueToInputStudent()) { // 정상처리 &&'Y'입력확인
        Student student = AppController.inputStudent();
        if (!this.ban().add(student)) {
            AppView.outputLine("(경고) 입력에 오류가 있습니다. 학급에 더이상 학생을 넣을 공간이 없습니다.");
            storingAStudentWasSuccessful = false; // 제대로 입력안되면 false 저장
        }
    }
    AppView.outputLine("! 성적 입력을 마칩니다.");
}
```

정상 처리가 되는지 여부를 storingAStudentWasSuccessful 변수에 Boolean 타입으로 저장하고 입력된 문자가 유효한지 확인한다. 'Ban'에 저장공간이 존재하는지 확인한다. 유효한 문자이고 그 전 처리가 정상처리 되었으면 inputStudent 함수를 통해 정수를 입력 받아 Student 객체에 저장한다.

```
    } else {
        this.showStatistics();
        this.showGradeCounts();
        this.showStudentSortedByScore();
    }
    AppView.outputLine("");
    AppView.outputLine("<<< 학급 성적 처리를 종료합니다 >>>");
}
```

run 함수에서 프로그램 종료 시, 출력하는 함수들이다.

해당 함수들에 대한 설명은 하단에 존재.

```
// 학급 성적 통계를 출력해주는 함수
private void showStatistics() {
    AppView.outputLine("");
    AppView.outputLine("[학급 성적 통계]");

    AppView.outputNumberOfStudents(this.ban().size()); // 학급 학생 수 출력
    AppView.outputHighestScore(this.ban().highest().score()); // 학급 최고점 출력
    AppView.outputLowestScore(this.ban().lowest().score()); // 학급 최저점 출력
    AppView.outputAverageScore(this.ban().average()); // 학급 평균점수 출력
    AppView.outputNumberOfStudentsAboveAverage(this.ban().numberOfStudentsAboveAverage()); // 평균이상인 학생 수 출력
}
```

학급 성적의 각 통계들을 출력하는 함수이다.

- size를 출력하여 총 학생 수를 출력한다.
- 재귀를 이용하여 highest, lowest 성적을 뽑아 출력한다.
- 평균 점수와 평균 이상의 학생 수를 출력한다.
- int와 double 타입을 활용한다.

```
// 학점별로 학생수를 출력해주는 함수
private void showGradeCounts() {
    AppView.outputLine("");
    AppView.outputLine("[학점별 학생수]");

    this.setGradeCounter(this.ban().countGrades());
    AppView.outputNumberOfStudentsForGrade('A', this.gradeCounter().numberOfA()); // A 학생 수 출력
    AppView.outputNumberOfStudentsForGrade('B', this.gradeCounter().numberOfB()); // B 학생 수 출력
    AppView.outputNumberOfStudentsForGrade('C', this.gradeCounter().numberOfC()); // C 학생 수 출력
    AppView.outputNumberOfStudentsForGrade('D', this.gradeCounter().numberOfD()); // D 학생 수 출력
    AppView.outputNumberOfStudentsForGrade('F', this.gradeCounter().numberOfF()); // F 학생 수 출력

}
```

학생 성적을 학점으로 변환하는 gradeCounter 함수를 이용하여 학점별로 학생 수를 출력하는 함수이다.

⇨ 학생 성적을 순서대로 출력하는 함수에 대해서는 상단에 설명 되어있다.

2. 프로그램 장단점 / 특이점 분석

➔ 장점

- MVC 모델을 이용하여 가독성과 생산성이 뛰어나다. 각 클래스, 함수의 역할이 분명해서 코드와 프로그램을 잘 이해할 수 있다.
- 설계가 잘 되어있다. 학번 등 학생 정보를 추가하여 더 입력 할 것이 있는 경우 프로그램에서 Student 클래스에 학번변수만 생성하여 사용하면 된다.
- 재귀를 사용하여 for, while문과 같은 반복문에 비해 코드가 간결하다. 다양한 방법을 통해 구현할 수 있었다.
- iterator을 이용한 반복자 개념을 활용할 수 있다. 반복문도 하나의 객체처럼 인터페이스를 정의하고 프로그램을 구현하니 여러 군데 활용할 수 있다.
- 구현 되어있는 UnsortedArrayList에 대한 재사용으로 코드를 구현하여 편리하였다.

➔ 단점

- 반복문에 비해 이해하기가 어렵다. 탈출 조건과 어떻게 나누어 재귀적으로 구현할 지 설계하는데 시간이 많이 소요될 것 같다.
- UnsortedArrayList를 재사용하였는데 이번 과제와 관련이 없는 함수들도 그대로 구현되어 있어서 다소 지저분하다고 느낄 수 있다. 하지만 자료구조 특성상 감안해야 할 부분인 것 같다.

3. 실행 결과 분석

(1) 입력과 출력 (화면 capture하여 제출)

[입출력 결과]

```
<terminated> _DS07_201702039_오명주 [Java Application] C:₩Program Files₩Java₩jdk-12.0.1₩bin₩javaw.exe (2021. 4. 19. 오후 7:54:24)
<<< 학급 성적 처리를 시작합니다 >>>

? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: y
- 점수를 입력하시오 (0..100): 33
? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: Y
- 점수를 입력하시오 (0..100): 97
? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: y
- 점수를 입력하시오 (0..100): 89
? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: Y
- 점수를 입력하시오 (0..100): 64
? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: y
- 점수를 입력하시오 (0..100): 72
? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: n
! 성적 입력을 마칩니다.

[학급 성적 통계]
학급 학생 수: 5
학급 최고 점수: 97
학급 최저 점수: 33
학급 평균: 71.0
평균 이상인 학생 수: 3

[학점별 학생수]
A 학점의 학생 수는 1 입니다.
B 학점의 학생 수는 1 입니다.
C 학점의 학생 수는 1 입니다.
D 학점의 학생 수는 1 입니다.
F 학점의 학생 수는 1 입니다.

[학생들의 성적순 목록]
점수: 33
점수: 64
점수: 72
점수: 89
점수: 97

<<< 학급 성적 처리를 종료합니다 >>>
```

[예외 처리]

```
<terminated> _DS07_201702039_오명주 [Java Application] C:₩Program Files₩Java₩jdk-12.0.1₩bin₩javaw.exe (2021. 4. 19. 오후 7:56:33)

<<< 학급 성적 처리를 시작합니다 >>>

? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: n
! 성적 입력을 마칩니다.

(경고) 입력된 성적이 없습니다.

<<< 학급 성적 처리를 종료합니다 >>>
```

입력된 성적이 없는 경우

```
_DS07_201702039_오명주 [Java Application] C:₩Program Files₩Java₩jdk-12.0.1₩bin₩javaw.exe (2021. 4. 19. 오후 7:57:23)

<<< 학급 성적 처리를 시작합니다 >>>

? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: y
- 점수를 입력하시오 (0..100): -1
[오류]0 보다 작거나 100 보다 커서, 정상적인 점수가 아닙니다.
- 점수를 입력하시오 (0..100): 200
[오류]0 보다 작거나 100 보다 커서, 정상적인 점수가 아닙니다.
- 점수를 입력하시오 (0..100):
```

점수 범위가 0..100 사이가 아닌 경우

```
<terminated> _DS07_201702039_오명주 [Java Application] C:₩Program Files₩Java₩jdk-12.0.1₩bin₩javaw.exe (2021. 4. 19. 오후 7:58:52)
? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: y
- 점수를 입력하시오 (0..100): 11
? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: y
- 점수를 입력하시오 (0..100): 33
? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: y
- 점수를 입력하시오 (0..100): 23
(경고) 입력에 오류가 있습니다. 학급에 더이상 학생을 넣을 공간이 없습니다.
! 성적 입력을 마칩니다.
```

학생 수가 capacity(=10) 보다 많은 경우

(2) 결과 분석 (자신의 논리적 평가, 기타 느낀 점)

⇨ 학번 등 학생 정보를 추가하여 더 입력 할 것들이 있을 경우, 프로그램에서 바뀌어야 하는 부분은?

- Student 클래스에 학번을 나타내는 studentNumber 변수를 추가하고, 해당 정보에 해당하는 getter/setter을 추가하여 추가구현을 진행하면 된다.

⇨ 재귀적이지 않은 문제 풀이에 비해 재귀적 문제풀이가 항상 좋은 성능을 낸다고 할 수 있을까?

- 코드의 가독성, 그리고 구현 면에서 복잡한 문제를 단순하게 접근한다는 점은 재귀의 장점이나 항상 좋은 성능을 낸다고 말 할 수는 없다.

- 피보나치 수열 같은 경우, 반복문이 훨씬 효율적인 구현이라고 할 수 있다. 하노이의 탑 역시 N이 커지면 결과가 나오는데 까지 시간이 오래 걸린다.

- 재귀함수는 기본적으로 스택 메모리를 사용하는데 재귀의 깊이가 깊어지면 stack overflow가 발생하기 쉽다. 스택 메모리를 초과하여 사용하는 문제이다.

⇨ 느낀점

- 이해 면에서 반복문보다 재귀가 어려웠다. Lowest 값, Highest 값, Sum 등 재귀적으로 구할 때, 모두 3의 크기를 가진 배열이 있다고 가정하고 생각을 했던 것 같다.

- 주어진 코드를 이해하는 것도 쉽지 않았는데 구현해야 했다면 지금보다 더 오랜 시간을 투자해야 했을 것이라고 생각된다.

4. 소스코드

```
public class _DS07_201702039_오명주 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        AppController appController = new AppController() ;
        // AppController가 실질적인 main class 이다
        appController.run() ;
        //여기 main() 에서는 앱 실행이 시작되도록 해주는 일이 전부이다

    }
```

[_DS_201702039_오명주]

```
public class Student implements Comparable<Student> {
    // Constants
    private static final int DEFAULT_SCORE = 0;
    // Private Instance Variables
    private int _score; // 점수

    // Getters/Setters
    public int score() {
        return this._score;
    }

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

    // Constructor
    public Student() {
        this.setScore(Student.DEFAULT_SCORE);
    }

    public Student(int givenScore) {
        this.setScore(givenScore);
    }

    @Override
    public int compareTo(Student other) {
        if (this.score() < other.score()) {
            return -1;
        } else if (this.score() == other.score()) {
            return 0;
        } else {
            return +1;
        }
    }
}
```

[Student]

```
    // 상수
    private static final int VALID_MAX_SCORE = 100;
    private static final int VALID_MIN_SCORE = 0;
    private static final int BAN_CAPACITY = 10;

    // 비공개 인스턴스 변수들
    private Ban _ban;
    private GradeCounter _gradeCounter;

    // Getters/Setters
    private Ban ban() {
        return this._ban;
    }

    private void setBan(Ban newBan) {
        this._ban = newBan;
    }

    private GradeCounter gradeCounter() {
        return this._gradeCounter;
    }

    private void setGradeCounter(GradeCounter newGradeCounter) {
        this._gradeCounter = newGradeCounter;
    }

    // 생성자
    public AppController() {
    }

    // 입력받은 학생 점수가 유효한지 확인하는 함수
    private static boolean scoreIsValid(int aScore) {
        return (aScore >= AppController.VALID_MIN_SCORE && aScore <= AppController.VALID_MAX_SCORE);
    }

    // Student 생성하여 score 저장하는 함수
    private static Student inputStudent() {
        int score = AppView.inputScore(); // 학생 점수를 입력받는다.
        while (!AppController.scoreIsValid(score)) { // 유효한 점수가 입력될때까지 반복
            AppView.outputLine("[오류]" + AppController.VALID_MIN_SCORE + " 보다 작거나 " + AppController.VALID_MAX_SCORE
                    + " 보다 커서, 정상적인 점수가 아닙니다.");
            score = AppView.inputScore();
        }
        Student student = new Student(); // student 객체 생성
        student.setScore(score); // 점수 입력
        return student;
    }

    private void inputAndStoreStudents() {
        AppView.outputLine("");
        boolean storingAStudentWasSuccessful = true; // 정상적으로 저리가 되었는지 확인하는 변수
        while (storingAStudentWasSuccessful && AppView.doesContinueToInputStudent()) { // 정상처리 &&'Y'입력확인
            Student student = AppController.inputStudent();
            if (!this.ban().add(student)) {
                AppView.outputLine("(경고) 입력에 오류가 있습니다. 학급에 더이상 학생을 넣을 공간이 없습니다.");
                storingAStudentWasSuccessful = false; // 제대로 입력안되면 false 저장
            }
        }
        AppView.outputLine("! 성적 입력을 마칩니다.");
    }

    // 학급 성적 통계를 출력해주는 함수
    private void showStatistics() {
        AppView.outputLine("");
        AppView.outputLine("[학급 성적 통계]");

        AppView.outputNumberOfStudents(this.ban().size()); // 학급 학생 수 출력
        AppView.outputHighestScore(this.ban().highest().score()); // 학급 최고점 출력
        AppView.outputLowestScore(this.ban().lowest().score()); // 학급 최저점 출력
        AppView.outputAverageScore(this.ban().average()); // 학급 평균점수 출력
        AppView.outputNumberOfStudentsAboveAverage(this.ban().numberOfStudentsAboveAverage()); // 평균이상인 학생 수 출력
    }

    // 학점별로 학생수를 출력해주는 함수
    private void showGradeCounts() {
        AppView.outputLine("");
        AppView.outputLine("[학점별 학생수]");

        this.setGradeCounter(this.ban().countGrades());
        AppView.outputNumberOfStudentsForGrade('A', this.gradeCounter().numberOfA()); // A 학생 수 출력
        AppView.outputNumberOfStudentsForGrade('B', this.gradeCounter().numberOfB()); // B 학생 수 출력
        AppView.outputNumberOfStudentsForGrade('C', this.gradeCounter().numberOfC()); // C 학생 수 출력
        AppView.outputNumberOfStudentsForGrade('D', this.gradeCounter().numberOfD()); // D 학생 수 출력
        AppView.outputNumberOfStudentsForGrade('F', this.gradeCounter().numberOfF()); // F 학생 수 출력

    }

    // 학생들의 성적을 순서대로 출력하는 함수
    private void showStudentSortedByScore() {
        AppView.outputLine("");
        AppView.outputLine("[학생들의 성적순 목록]");
        this.ban().sortByScore(); // sortByScore를 통해 정렬

        Iterator<Student> iterator = this.ban().iterator(); // 반복자 생성
        Student student = null;
        while (iterator.hasNext()) { // hasnext()가 null이 아닌동안 반복
            student = iterator.next(); // 다음 student를 저장
            AppView.outputScore(student.score()); // student의 score 출력
        }
    }

    public void run() {
        AppView.outputLine("");
        AppView.outputLine("<<< 학급 성적 처리를 시작합니다 >>>");

        this.setBan(new Ban(AppController.BAN_CAPACITY)); // capacity 설정
        this.inputAndStoreStudents(); // 성적 입력받아서 Student 객체로 저장
        if (this.ban().isEmpty()) {
            AppView.outputLine("");
            AppView.outputLine("(경고) 입력된 성적이 없습니다.");
        } else {
            this.showStatistics();
            this.showGradeCounts();
            this.showStudentSortedByScore();
        }
        AppView.outputLine("");
        AppView.outputLine("<<< 학급 성적 처리를 종료합니다 >>>");
    }
}
```

[AppController]

```
    private static Scanner scanner = new Scanner(System.in);

    // 생성자
    public AppView() {

    }

    // 출력 관련 함수
    // 한줄을 출력하는 함수 (한줄이 띄워지지않는다)
    public static void output(String message) {
        System.out.print(message); // 입력받은 message를 출력한다
    }

    // 한줄을 출력하는 함수 (한줄이 띄워진다)
    public static void outputLine(String message) {
        System.out.println(message); // 입력받은 message를 출력한다
    }

    // 정수가 아닌 경우의 에러 처리를 보완할 것 : exception throws
    public static int inputInteger() throws NumberFormatException {
        return Integer.parseInt(AppView.scanner.next());
    }

    // 학급 학생 수 출력
    public static void outputNumberOfStudents(int aNumberOfStudents) {
        System.out.println("학급 학생 수: " + aNumberOfStudents);
    }

    // 학급 최고 점수 출력
    public static void outputHighestScore(int aScore) {
        System.out.println("학급 최고 점수: " + aScore);
    }

    // 학급 최저 점수 출력
    public static void outputLowestScore(int aScore) {
        System.out.println("학급 최저 점수: " + aScore);
    }

    // 평균값 출력
    public static void outputAverageScore(double anAverageScore) {
        System.out.println("학급 평균: " + anAverageScore);
    }

    // 평균 이상인 학생수 출력
    public static void outputNumberOfStudentsAboveAverage(int aNumberOfStudents) {
        System.out.println("평균 이상인 학생 수: " + aNumberOfStudents);
    }

    // 각 학점에 대한 학생 수 출력
    public static void outputNumberOfStudentsForGrade(char aGrade, int aNumberOfStudents) {
        System.out.println(aGrade + " 학점의 학생 수는 " + aNumberOfStudents + " 입니다.");
    }

    // 학생들의 점수 출력
    public static void outputScore(int aScore) {
        System.out.println("점수: " + aScore);
    }

    // 입력관련함수
    // 성적 입력받아서 예외처리
    public static int inputInt() throws NumberFormatException {
        // 입력값이 숫자가 아니면 예외처리
        return Integer.parseInt(AppView.scanner.nextLine());
    }

    // 성적 입력받아서 예외처리
    public static int inputScore() {
        while (true) {
            try {
                AppView.output("- 점수를 입력하시오 (0..100): ");
                int score = AppView.inputInt();
                return score;
            } catch (NumberFormatException e) {
                AppView.outputLine("(오류) 정수가 입력되지 않았습니다");
            }
        }
    }

    // Y또는y가 입력되었는지 확인
    public static boolean doesContinueToInputStudent() {
        AppView.output("? 성적을 입력하려면 'Y'또는 'y'를, 종료하려면 다른 아무 키나 치시오: ");
        String line = null;
        do { // 빈 줄이 아닐때까지 입력받는다
            line = AppView.scanner.nextLine();
        } while (line.equals(""));
        char answer = line.charAt(0);
        return ((answer == 'Y') || (answer == 'y'));
    }
}
```

[AppView]

```
    // Constructor
    public Ban() {
        super();
    }

    public Ban(int givenCapacity) {
        super(givenCapacity);
    }

    // 점수를 학점으로 환산하는 함수
    private static 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 Student lowestRecursively(int left, int right) {
        if (left == right) { // 모든 값을 다 비교한 경우
            return this.elementAt(left); // left를 반환
        } else {
            Student lowestFromRights = lowestRecursively(left + 1, right); // left와 left+1, left+2, left+3 .. 를 계속 비교
            if (lowestFromRights.compareTo(this.elementAt(left)) <= 0) { // 더 작은 값을 반환
                return lowestFromRights; // lowestFromRights가 작으면 이것을 반환
            } else {
                return this.elementAt(left); // left반해가 작다면 이것을 반환
            }
        }
    }

    // 성적이 가장 낮은 학생을 얻는다
    public Student lowest() {
        if (this.isEmpty()) { // 비어있으면
            return null; // null
        } else {
            return this.lowestRecursively(0, this.size() - 1); // 재귀를 이용하여 성적 가장 낮은 학생 반환
        }
    }

    private Student highestRecursively(int left, int right) {
        if (left == right) { // 모든 값을 다 비교한 경우
            return this.elementAt(left); // left를 반환
        } else {
            Student highestFromRights = highestRecursively(left + 1, right); // left와 left+1, left+2 .. 비교
            if (highestFromRights.compareTo(this.elementAt(left)) >= 0) { // 더 큰 값을 반환
                return highestFromRights; // highestFromRights 가 같거나 크다면 반환
            } else {
                return this.elementAt(left); // left 값이 더 크다면 반환
            }
        }
    }

    // 성적이 가장 높은 학생을 얻는다
    public Student highest() {
        if (this.isEmpty()) { // 비어있으면
            return null; // null
        } else {
            return this.highestRecursively(0, this.size() - 1); // 재귀를 이용하여 성적 가장 높은 학생 반환
        }
    }

    private int sumOfScoreRecursively(int left, int right) {
        int mid = (left + right) / 2; // 중간지점을 설정
        if (left == right) { // left와 right 같다면
            return this.elementAt(left).score(); // left의 스코어를 반환
        } else {
            int leftSum = this.sumOfScoreRecursively(left, mid); // leftSum은 left ~ mid 를 재귀적으로 합한 것
            int rightSum = this.sumOfScoreRecursively(mid + 1, right); // rightSum은 mid + 1 ~ right를 재귀적으로 합한 것
            return (leftSum + rightSum); // leftSum + rightSum을 반환한다
        }
    }

    // 학급의 학생들의 성적의 합계를 얻는다
    public int sum() {
        if (this.isEmpty()) { // 비어있다면
            return 0; // 합은 0
        } else {
            return this.sumOfScoreRecursively(0, this.size() - 1); // 재귀를 이용하여 0~(size-1)합 반환
        }
    }

    // 학급의 성적 평균 성적을 얻는다
    public double average() {
        if (this.isEmpty()) {
            return 0;
        } else {
            return ((double) this.sum()) / ((double) this.size());
        }
    }

    // 평균보다 높은 학생 혹은 반환하는 함수
    public int numberOfStudentsAboveAverage() {
        double average = this.average(); // average()를 통해 평균값을 구한다
        int numberOfStudentsAboveAverage = 0; // 평균이상학생수를 0으로 초기화
        Iterator<Student> iterator = this.iterator(); // 반복자 객체 생성
        while (iterator.hasNext()) { // 다음 반복자가 있으면
            Student student = iterator.next(); // student가 반복자의 다음이 된다.
            if (student.score() >= average) { // 평균 이상이면
                numberOfStudentsAboveAverage++; // 학생수 + 1
            }
        }
        return numberOfStudentsAboveAverage; // 카운트 된 학생수를 반환
    }

    // p번째와 q번째 Student element를 교체
    private void swap(int p, int q) {
        Student temp = this.elementAt(p); // Student temp는 p번째 배열
        this.setElementsAt(p, this.elementAt(q)); // p번째 배열에 q번째 학생을 넣고
        this.setElementsAt(q, temp); // q번째 배열에 p번째 배열이었던 temp를 넣는다.
    }

    private int partition(int left, int right) {
        int pivot = left; // pivot을 left로 설정
        int toRight = left; // 오른쪽으로 갈 toRight는 left 위치에 지정
        int toLeft = right + 1; // 왼쪽으로 갈 toLeft는 right+1 위치에 지정
        do {
            do {
                toRight++;
            } while (this.elementAt(toRight).score() < this.elementAt(pivot).score()); // Left에서 Right로 갈 score 위치 선정
            do {
                toLeft--;
            } while (this.elementAt(toLeft).score() > this.elementAt(pivot).score()); // Right에서 Left로 갈 score 위치 선정
            if (toRight < toLeft) { // toRight < toLeft라면
                this.swap(toRight, toLeft); // 우가의 배열을 바꾼다.
            }
        } while (toRight < toLeft); // toRight > toLeft가 되는 순간 탈출
        this.swap(pivot, toLeft); // pivot과 toLeft를 바꾼다.
        return toLeft; // pivot 위치가 toLeft 이다.
    }

    private void quicksortRecursively(int left, int right) { // 확장형
        if (left < right) { // left<right이면
            int mid = this.partition(left, right); // 파티션 후의 pivot 위치
            this.quicksortRecursively(left, mid - 1); // 나누어진 반을 다시 확장할한다.
            this.quicksortRecursively(mid + 1, right); // 나누어진 반을 다시 확장할한다.
        }
    }

    // 학급의 학생들을 성적 순으로 정렬한다
    public void sortByScore() {
        if (this.size() > 1) { // 배열이 1개이도 있으면
            int maxLoc = 0; // 최대값 위치를 0으로 초기화
            for (int i = 1; i < this.size(); i++) { // 배열 끝까지 반복
                if (this.elementAt(i).score() > this.elementAt(maxLoc).score()) { // i번째 성적이 maxLoc의 성적보다 크다면
                    maxLoc = i; // maxLoc을 i로 설정한다
                }
            }
            this.swap(maxLoc, this.size() - 1); // 최대값인 maxLoc을 맨 뒤로 보낸다.
            this.quicksortRecursively(0, this.size() - 2); // 최대값을 뺀 0 ~ this.size()-2 를 확장할 한다.
        }
    }

    // 학급 학점별로 학생수를 세거하고, 그 결과를 가지는 GradeCounter 객체 얻는다
    public GradeCounter countGrades() {
        // step 1 : GradeCounter 객체 생성
        // 학점별 counter는 모두 0으로 초기화한다.
        GradeCounter gradeCounter = new GradeCounter(); // gradeCounter 선언
        // step 2 : 학생들의 학점을 센다
        // 학생들은 Iterator를 사용하여 자체로 얻는다.
        Iterator<Student> iterator = this.iterator(); // 반복자 선언
        while (iterator.hasNext()) { // 반복자의 다음이 있다면
            char grade; // grade 문자열 선언
            Student student = iterator.next(); // student는 반복자의 next가 됨
            grade = Ban.scoreToGrade(student.score()); // 학생의 성적을 학점으로 반환하고 grade에 넣는다.
            gradeCounter.count(grade); // grade를 받아 카운트를 한다.
        }
        return gradeCounter; // gradeCounter를 반환
    }
}
```

[Ban]

```
    public int numberOfA() {
        return _numberOfA;
    }

    public int numberOfB() {
        return _numberOfB;
    }

    public int numberOfC() {
        return _numberOfC;
    }

    public int numberOfD() {
        return _numberOfD;
    }

    public int numberOfF() {
        return _numberOfF;
    }

    private void setNumberOfA(int newA) {
        this._numberOfA = newA;
    }

    private void setNumberOfB(int newB) {
        this._numberOfB = newB;
    }

    private void setNumberOfC(int newC) {
        this._numberOfC = newC;
    }

    private void setNumberOfD(int newD) {
        this._numberOfD = newD;
    }

    private void setNumberOfF(int newF) {
        this._numberOfF = newF;
    }

    // Constructor
    public GradeCounter() {
        this.setNumberOfA(0);
        this.setNumberOfB(0);
        this.setNumberOfC(0);
        this.setNumberOfD(0);
        this.setNumberOfF(0);
    }

    public void count(char aGrade) {
        switch (aGrade) {
        case 'A':
            this.setNumberOfA(this.numberOfA() + 1);
            break;
        case 'B':
            this.setNumberOfB(this.numberOfB() + 1);
            break;
        case 'C':
            this.setNumberOfC(this.numberOfC() + 1);
            break;
        case 'D':
            this.setNumberOfD(this.numberOfD() + 1);
            break;
        case 'F':
            this.setNumberOfF(this.numberOfF() + 1);
            break;
        }
    }

}
```

[GradeCounter]

```
    private static final int DEFAULT_CAPACITY = 100;

    private int _capacity;
    private int _size;
    private E[] _elements;

    // Getter/Setter
    private int capacity() { // getter of capacity
        return this._capacity;
    }

    private void setCapacity(int newCapacity) { // setter of capacity
        this._capacity = newCapacity;
    }

    public int size() { // getter of size
        return this._size;
    }

    private void setSize(int newSize) { // setter of size
        this._size = newSize;
    }

    private E[] elements() { // getter of elements
        return this._elements;
    }

    private void setElements(E[] newElements) { // setter of elements
        this._elements = newElements;
    }

    // 생성자
    public UnsortedArrayList() {
        this.setCapacity(DEFAULT_CAPACITY);
    }

    @SuppressWarnings("unchecked")
    public UnsortedArrayList(int givenCapacity) {
        this.setCapacity(givenCapacity);
        this.setElements((E[]) new Comparable[this.capacity()]);
    }

    public boolean isEmpty() { // 비어있으면 true
        return (this.size() == 0);
    }

    public boolean isFull() { // 꽉찼으면 true
        return (this.size() == this._capacity);
    }

    public E elementAt(int anOrder) { // anOrder번째 배열 반환
        if (anOrder < 0 || anOrder >= this.size()) { // anOrder이 범위 밖이면
            return null; // null 반환
        } else {
            return this.elements()[anOrder]; // anOrder번째 배열 반환
        }
    }

    protected void setElementsAt(int anOrder, E anElement) {
        if (anOrder < 0 || anOrder >= this.size()) { // anOrder 범위 밖이라면
            return; // 진행
        } else { // 범위 안이라면
            this.elements()[anOrder] = anElement; // anOrder번째 배열 anElement로 설정
        }
    }

    public boolean doesContain(E anElement) { // 존재 유무 확인
        return (this.orderOf(anElement) >= 0); // anElement가 orderOf로 인해 있다는게 확인되면
                                               // true를 반환
    }

    public int orderOf(E anElement) {
        int order = -1; // 순서 변수 -1로 선언
        for (int index = 0; index < this.size() && order < 0; index++) { // 0~사이즈 &
            // order의 -1일때 반복
            if (this.elements()[index].equals(anElement)) { // index번째 배열과 anElement가 같으면
                order = index; // order 를 index로 설정
            }
        }
        return order; // 같은 배열을 찾은 순서인 order를 반환
    }

    private void makeRoomAt(int aPosition) {
        for (int i = this.size(); i > aPosition; i--) { // this.size()부터 aPosition까지 반복
            this.elements()[i] = this.elements()[i - 1]; // i번째 배열을 i - 1 번째 배열로 설정
                                                         // 한 칸씩 미는 것
        }
    }

    public boolean addToFirst(E anElement) {
        if (this.isFull()) { // 꽉차면 추가를 못하니
            return false; // false 반환
        } else {
            this.makeRoomAt(0); // 0번째부터 한칸씩 민다
            this.elements()[0] = anElement; // 0번째 배열에 anElement 대입
            this.setSize(this.size() + 1); // 사이즈 + 1
            return true; // true반환
        }
    }

    public boolean addToLast(E anElement) {
        if (this.isFull()) { // 꽉차면 추가를 못하니
            return false; // false 반환
        } else {
            this.elements()[this.size()] = anElement; // this.size()번째 배열에 anElement 대입
            this.setSize(this.size() + 1); // 사이즈 + 1
            return true; // true 반환
        }
    }

    public boolean add(E anElement) { // 용이한 위치에 add
        return this.addToLast(anElement); // 마지막 위치에 추가하는 것이 용이함
    }

    public Iterator<E> iterator() {
        return (new ListIterator());
    }

    private class ListIterator implements Iterator<E> {
        private int _nextPosition;

        private int nextPosition() {
            return this._nextPosition;
        }

        private void setNextPosition(int newNextPosition) {
            this._nextPosition = newNextPosition;
        }

        private ListIterator() {
            this.setNextPosition(0);
        }

        @Override
        public boolean hasNext() {
            return (this.nextPosition() < UnsortedArrayList.this.size());
        }

        @Override
        public E next() {
            E nextElement = null;
            if (this.hasNext()) {
                nextElement = UnsortedArrayList.this.elements()[this.nextPosition()];
                this.setNextPosition(this.nextPosition() + 1);
            }
            return nextElement;
        }
    }
}
```

[UnsortedArrayList]