자료구조 실습 보고서

[제 6주]

제출일: 19.05.01
학번/이름: 201603867/조성환

1. 프로그램 설명서

1) 주요 알고리즘/ 자료구조/ 기타

성적을 입력하면 저장하는 함수

2) 함수 설명서

AppController

void run()

시작한다는 출력문을 나타낸다

this.setBan(new Ban(Appcontroller.BAN_CAPACITY)를 통해 Ban을 set한다. 그 대상은 새로운 Ban 객체이며, BAN_CAPACITY를 전달한다.

this.inputAndStoreStudents()를 통해 성적을 입력받고, 성적을 저장한다.

this.ban().isEmpty()를 통해 ban이 비어있는지 확인한다. 만약 비어있다면 경고를 날리고, 비어있지 않다면 showStatistics(), showGradeCounts(), ShowStudentSortedByScore() 함수를 실행한다. 각 함수는 현재 상황, 학점별 인원수, 성적순 정렬된 모습을 보여준다.

끝낸다는 출력문을 나타낸다.

inputAndStoreStudents()

boolean storingAStudentSuccessful이라는 boolean형 변수를 사용하며 이 변수와 doesCotinueToInputStudent()함수가 모두 true일 때 while문을 계속 돌린다.

while문은 Student객체를 만들고, Appcontorller.inputSutdent()를 통해 성적을 입력받는다.

만약 입력이 이상하거나, 공간이 없으면 성적 storingAStudentSuccesfful을 false로 만들고 함수를 멈춘다.

마지막으로 성적 입력을 마치는 출력문과 함께 함수를 종료한다.

inputStudent()

score변수를 AppView.inputScore()함수를 통해 값을 입력받아 넣어준다.

AppController.scoreIsValid(score)가 true이면 Student객체를 만들고 student.setScore(score)함수를 통해 값을 입력한다. 이 입력한 student를 리턴한다.

만약 scoreIsValid(score)가 false이면 오류 메시지를 출력한다.

boolean scoreIsValid(int aScore)

이 함수는 score가 유효한지 검사한다.

AppController는 VALID_MIN_SCORE와 MAX를 지정해뒀으며, 입력된 score가 최소와 최대 사이에 있으면 true 없으면 false를 반환한다.

ShowStatistics()

이 함수는 성적 통계를 보여준다.

Appview의 학생수, 최고 성적, 최저 성적, 평균, 평균을 넘어서 학생의 수를 this.ban().size, 최고점.score, 최저점.score, 평균, 평균 학생수의 함수의 리턴값을 appview에 넣는 형식으로 보여준다.

showGradeCounts()
이 함수는 점수를 학점으로 보여준다.
this.setGradeCounter(this.ban().countGrades())를통해 학점 별 학생의 수를 계산한다.
appView.outputNumberOfStudentsForGrade()함수를 통해 학점별 학생 수를 출력한다.

show StudentSortedByScore()
이 함수는 성적을 낮은 순으로 정렬해서 보여준다.
this.ban().sortedScore()함수를 통해 성적을 정렬하낟.
Iterator<Student> iterator= this.ban().iterator();를 통해 student를 담는 iterator를 생성하고
while(iterator.hasNext)로 ban의 내부를 계속 돌린다.
Student student= null;을 통해 student를 만들고 iterator.next()를 통해 ban 전체를 조회한다. 각 요소마다 AppView.outputStudentInfo(student.score())를 통해 점수를 출력한다.

Ban

Ban(int givenCapacity)
이 함수를 통해 최대 길이를 지정한다.
Ban은 UnsortedArrayList<Student>를 상속하기 때문에, Student를 담는 정렬되지 않은 배열이다. 그러므로 Ban 생성자는 Ban의 길이를 정해줘야한다.

isEmpty(), isFull()
size를 0과 capacity와 비교해 비어있는지, 꽉 차있는지를 검사한다.

size()
Ban의 size를 나타낸다.

sortByScore()
size가 1보다 크면
maxLoc변수를 0으로 초기화 한다.
for문을 size만큼 돌린다. 현재 요소가 maxLoc위치의 요소보다 크면 현재 위치를 maxLoc로 바꾼다.
this.swap(maxLoc, this.size()-1)을 통해 maxLoc를 맨 마지막 요소와 바꾼다.
this.quicksortRecursively(0, size()-2)를 통해 맨 마지막 요소를 제외한 나머지를 quicksort로 재귀적으로 정렬한다.

void quicksortRecursively(int left, int right)
left가 right보다 작으면 mid를 this.partition(left, right)로 초기화한다.
quicksortRecursively(left, mid-1), quicksortRecursively(mid+1, right)로 두 개로 나눠서 정렬을 시킨다.

int partition(int left, int right)
pivot 변수를 left로 초기화한다. toRight= left, toLeft= right로 초기화한다.
toRight< toLeft가 false가 될 때까지 do while문을 돌린다.
this.elementAt(toRight).score()가 this.elementAt(pivot).score()보다 작으면 toRight+1
this.elementAt(toLeft).score()가 this.elementAt(pivot).score()보다 크면 toLeft+1를 한다
만약 toRight가 toLeft보다 작으면 toRight, toLeft의 위치를 this.swap()을 통해 바꿔준다.
마지막으로 while문을 벗어나면 this.swap(left, toLeft)를 통해 서로 바꿔준고 toLeft를 반환한다. 이게 바로 mid가 된다.

void swap(int p, int q)
Student temp= this.elementAt(p);로 초기화 한 뒤
this.setElementAt(해당 위치)를 통해 서로 바꿔준다.

Student lowest()
lowestRecursively를 통해 가장 작은 점수를 가져온다.
가장 작은 점수는 Student에 담고, 이 student를 반환한다.

lowestRecursively(left, right)
left== right일 때 this.elementAt(left)를 통해 해당 값을 반환한다.
만약 같지 않으면 Student에 lowestRecursively(left+1, right)를 통해 맨 왼쪽 값만 남기고 나머지를 재귀적으로 계속 호출하고, 마지막에 리턴 값과 highestFromLefts와 비교해 큰 값을 반환한다.

GradCounter
A,B,C,D,F의 numberOf에 관한 변수를 선언한다.

GradeCounter()
생성자를 통해 각 변수의 값을 0으로 초기화한다.

void Count(char aGrade)
를 통해 입력 받은 값이 switch로 들어간다. a,b,c,d,f에 해당하면 해당하는 변수를 1 늘려준다.
이 함수는 ban에서 호출되어 scoreToGrade를 통해 학점으로 변환한 후 모든 요소를 학점수만큼 증가시킨다.

3) 종합 설명서

전체적으로 입력을 받고 저장을 한 뒤 통계를 보여준다.

2. 프로그램 장단점/ 특이점 분석

프로그램을 재귀적으로 분리해서 최대값이나 원하는 값을 가져오기 때문에 그냥 앞에서부터

찾는 unsortedList에서 max를 찾는거에 비해 훨씬 빠르게 찾을 수 있다. 내부의 동작방식은 iterator, student, ban, unsortedArrayList를 통해 그래도 진행이 되며, Appcontroller에 의해 원할 때 불러오면 돼서 stack뿐 만 아니라 다른 곳에서도 사용이 가능하다는 장점이있다.

3. 실행 결과 분석

1) 입력과 출력

```
성적을 입력하려면 'Y' 또는 'y'를, 종료하려면 다른 아무 키나 치시오: Y
- 점수를 입력하시오 (0..100): 60
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): 102
[오류] 0 보다 작거나 100 보다 커서, 정상적인 점수가 아닙니다.
- 점수를 입력하시오 (0..100): 88
성적을 입력하려면 'Y' 또는 'y'를, 종료하려면 다른 아무 키나 치시오: n
! 성적 입력을 마칩니다.

[학급 성적 통계]
학급 학생 수: 4
학급 최고 점수: 90
학급 최저 점수: 50
학급 평균: 72.0
평균 이상인 학생 수: 2.0

[학점별 학생수]
A학점의 학생 수는 1 입니다.
B학점의 학생 수는 1 입니다.
C학점의 학생 수는 0 입니다.
D학점의 학생 수는 1 입니다.
F학점의 학생 수는 1 입니다.

[학생들의 성적순 목록]
점수: 50
점수: 60
점수: 88
점수: 90

<<< 학급 성적 처리를 종료합니다 >>>
```

2) 결과 분석

오류 처리도 잘되고 결과도 잘 나온다.

4. 소스 코드

Appcontroller

```
        AppView.outputNumberOfStudentsAboveAverage(this.ban().numberOfStudentsAboveAverage());
    }
    private void showGradeCounts(){
        AppView.outputLine( aMessage: "");
        AppView.outputLine( aMessage: "[학점별 학생수]");

        this.setGradeCounter(this.ban().countGrades());
        AppView.outputNumberOfStudentsForGrade( aGrade: 'A', this.gradeCounter().numberOfA());
        AppView.outputNumberOfStudentsForGrade( aGrade: 'B', this.gradeCounter().numberOfB());
        AppView.outputNumberOfStudentsForGrade( aGrade: 'C', this.gradeCounter().numberOfC());
        AppView.outputNumberOfStudentsForGrade( aGrade: 'D', this.gradeCounter().numberOfD());
        AppView.outputNumberOfStudentsForGrade( aGrade: 'F', this.gradeCounter().numberOfF());
    }
    private void showStudentsSortedByScore(){
        AppView.outputLine( aMessage: "");
        AppView.outputLine( aMessage: "[학생들의 성적순 목록]");

        this.ban().sortByScore();

        Iterator<Student> iterator= this.ban().iterator();
        Student student= null;
        while(iterator.hasNext()){
            student= iterator.next();
            AppView.outputStudentInfo(student.score());
        }
    }

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

        this.setBan(new Ban(AppController.BAN_CAPACITY));
        this.InputAndStoreStudents();
        if (this.ban().isEmpty()){
```

```
private static Student inputStudent(){
    int score= AppView.inputScore();
    while (! AppController.scoreIsValid(score)){
        AppView.outputLine( aMessage: "[오류] "+
                AppController.VALID_MIN_SCORE+" 보다 작거나 "+
                AppController.VALID_MAX_SCORE+ " 보다 커서, 정상적인 점수가 아닙니다.");
        score= AppView.inputScore();
    }
    Student student= new Student();
    student.setScore(score);
    return student;
}
private void inputAndStoreStudents(){
    AppView.outputLine( aMessage: "");
    boolean storingAStudentSuccessful= true;
    while ( storingAStudentSuccessful && AppView.doesContinueToInputStudent()){
        Student student= AppController.inputStudent();
        if(! this.ban().add(student)){
            AppView.outputLine( aMessage: "(경고 입력에 오류가 있습니다. 학급에 더이상 학생을 넣을 공간이 없습니다.");
            storingAStudentSuccessful= false;
        }
    }
    AppView.outputLine( aMessage: "| 성적 입력을 마칩니다.");
}
private void showStatistics(){
    AppView.outputLine( aMessage: "");
    AppView.outputLine( aMessage: "[학급 성적 통계]");

    AppView.outputNumberOfStudets(this.ban().size());
    AppView.outputHighestScore(this.ban().highest().score());
    AppView.outputLowestScore(this.ban().lowest().score());
    AppView.outputAverageScore(this.ban().average());
    AppView.outputNumberOfStudentsAboveAverage(this.ban().numberOfStudentsAboveAverage());
```

```java
package controller;

import model.Ban;
import model.GradeCounter;
import model.Iterator;
import model.Student;
import view.AppView;

public class AppController {

    //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 변수
    private Ban _ban;
    private GradeCounter _gradeCounter;

    public Ban ban() { return _ban; }
    public void setBan(Ban newBan) { this._ban = newBan; }

    public GradeCounter gradeCounter() { return _gradeCounter; }
    public void setGradeCounter(GradeCounter newGradeCounter) { this._gradeCounter = newGradeCounter; }

    public AppController(){}

    private static boolean scoreIsValid(int aScore){
        return ( aScore>= AppController.VALID_MIN_SCORE&&
                aScore<= AppController.VALID_MAX_SCORE);
    }
    private static Student inputStudent(){
```

```java
            AppView.outputLine( aMessage: "");
            AppView.outputLine( aMessage: "(경고) 입력된 성적이 없습니다.");
        } else {
            this.showStatistics();
            this.showGradeCounts();
            this.showStudentsSortedByScore();
        }
        AppView.outputLine( aMessage: "");
        AppView.outputLine( aMessage: "<<< 학급 성적 처리를 종료합니다 >>>");

    }
```

Ban

```
                numberOfStudentsAboveAverage++;
            }
        }
        return numberOfStudentsAboveAverage;
    }
    public void sortByScore(){
        if (this.size()> 1){
            int maxLoc= 0;
            for (int i= 1; i< this.size(); i++){
                if (this.elementAt(i).score()> this.elementAt(maxLoc).score()){
                    maxLoc= i;
                }
            }
            this.swap(maxLoc,  q: this.size()-1);
            this.quicksortRecursively( left: 0,  right: this.size()-2);
        }
    }
    private void swap(int p, int q){
        Student temp= this.elementAt(p);
        this.setElementAt(p, this.elementAt(q));
        this.setElementAt(q, temp);
    }
    private int partition(int left, int right){
        int pivot= left;
        int toRight= left;
        int toLeft= right+1;
        do {
            do {toRight++;} while (this.elementAt(toRight).score()< this.elementAt(pivot).score());
            do {toLeft--;} while (this.elementAt(toLeft).score()> this.elementAt(pivot).score());
            if (toRight< toLeft){
                this.swap(toRight, toLeft);
            }
        } while (toRight< toLeft);
        this.swap(left, toLeft);
        return toLeft;
```

```
        } else {
            return this.lowestRecursively( left: 0,  right: this.size() - 1);
        }
    }
    public Student highest() {
        if (this.isEmpty()) {
            return null;
        } else {
            return this.highestRecursively( left: 0,  right: this.size() - 1);
        }
    }
    public int sum(){
        if (this.isEmpty()){
            return 0;
        } else {
            return this.sumOfScoresRecursively( left: 0,  right: this.size()-1);
        }
    }
    public double average(){
        if (this.isEmpty()){
            return 0;
        } else {
            return ((double)this.sum())/ ((double) this.size());
        }
    }
    public int numberOfStudentsAboveAverage(){
        double average= this.average();
        int numberOfStudentsAboveAverage= 0;
        Iterator<Student> iterator= this.iterator();
        while (iterator.hasNext()){
            Student student= iterator.next();
            if (student.score()>= average){
                numberOfStudentsAboveAverage++;
            }
```

```
private Student highestRecursively(int left, int right) {
    if (left == right) {
        return this.elementAt(left);
    } else {
        Student highestFromLefts = highestRecursively( left: left + 1, right);
        if (highestFromLefts.compareTo(this.elementAt(left)) >= 0) {
            return highestFromLefts;
        } else {
            return this.elementAt(left);
        }
    }
}
private int sumOfScoresRecursively(int left, int right){
    int mid= (left+ right)/ 2;
    if(left== right){
        return this.elementAt(left).score();
    } else {
        int leftSum= this.sumOfScoresRecursively(left, mid);
        int rightSum= this.sumOfScoresRecursively( left: mid+1, right);
        return (leftSum+ rightSum);
    }
}
private void quicksortRecursively(int left, int right){
    if (left< right){
        int mid= this.partition(left, right);
        this.quicksortRecursively(left,  right: mid-1);
        this.quicksortRecursively( left: mid+1, right);
    }
}

public Student lowest() {
    if (this.isEmpty()) {
        return null;
    } else {
```

```java
package model;

public class Ban extends UnsortedArrayList<Student> {

    //static method
    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';
        }
    }

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

    private Student lowestRecursively(int left, int right) {
        if (left == right) {
            return this.elementAt(left);
        } else {
            Student lowestFromRights = lowestRecursively( left: left + 1, right);
            if (lowestFromRights.compareTo(this.elementAt(left)) <= 0) {
                return lowestFromRights;
            } else {
                return this.elementAt(left);
            }
        }
    }
```

```
private int partition(int left, int right){
    int pivot= left;
    int toRight= left;
    int toLeft= right+1;
    do {
        do {toRight++;} while (this.elementAt(toRight).score()< this.elementAt(pivot).score());
        do {toLeft--;} while (this.elementAt(toLeft).score()> this.elementAt(pivot).score());
        if (toRight< toLeft){
            this.swap(toRight, toLeft);
        }
    } while (toRight< toLeft);
    this.swap(left, toLeft);
    return toLeft;
}
public GradeCounter countGrades(){
    char currentGrade;
    GradeCounter gradeCounter = new GradeCounter();
    for(int i = 0; i<this.size() ; i++) {
        currentGrade = Ban.scoreToGrade(this.elementAt(i).score());
        gradeCounter.count(currentGrade);
    }
    return gradeCounter;
}
```

GradeCounter

```java
package model;

public class GradeCounter {

    //비공개 변수
    private int _numberOfA;
    private int _numberOfB;
    private int _numberOfC;
    private int _numberOfD;
    private int _numberOfF;

    public int numberOfA() {return _numberOfA;}
    public void setNumberOfA(int newNumberOfA) { this._numberOfA = newNumberOfA; }

    public int numberOfB() { return _numberOfB;}
    public void setNumberOfB(int newNumberOfB) { this._numberOfB = newNumberOfB; }

    public int numberOfC() { return _numberOfC; }
    public void setNumberOfC(int newNumberOfC) { this._numberOfC = newNumberOfC; }

    public int numberOfD() { return _numberOfD; }
    public void setNumberOfD(int newNumberOfD) { this._numberOfD = newNumberOfD; }

    public int numberOfF() { return _numberOfF; }
    public void setNumberOfF(int newNumberOfF) { this._numberOfF = newNumberOfF; }

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

    }
```

```java
//공개 함수
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;
    }
}
```