DataStructures COSE213-01 Spring 2019 Prof. Heung-Il Suk _ #2017320233 김채령

[Assignment #1] Polynomial Addition

1. 해결 과제

<수업 시간에 배운 내용을 기반으로 두 다항식의 덧셈을 하는 프로그램 구현하기>

- ADT에 따라 계수는 실수, 지수는 음이 아닌 정수로 정의한다.
- Data structure는 polynomial이라는 Class객체(C에서 구조체)의 배열에 계수와 지수 쌍을 내림차순으로 저장하는데, 이때 2개의 다항식과 결과 다항식을 모두 하나의 공유하는 static 배열(C에서 global array)에 저장한다.

2. 코드 설명

```
196 /* C언어의 typedef struct 역할
197   계수와 차수를 하나의 객체로 만들어서 배열에 저장하기 위함 */
198 class Poly{
199     float coef; // ADT에 따라 계수는 실수, 차수는 정수로 선언함
200     int expon;
201 }
```

먼저, C언어의 typedef struct를 새로운 class를 만들어주어 대체했다. Poly라는 하나의 class에서 계수(coef)와 지수(expon)을 동시에 관리할 수 있게 되었고, ADT에 따라 각 변수의 데이터타입을 선언해주었다.

```
 8 public class Padd {
 9
10     /*이 class의 모든 method들이 공유하는 field*/
11
12     static int MAX_TERM = 100; /* 배열이 너무 커지는 것을 막기 위함, 항의 개수에 제한 */
13     static Poly poly[] = new Poly[MAX_TERM]; /* MAX_TERM개의 Poly 데이터 타입을 가지는 배열 선언 - 두 다항식과 결과 다항식 함께 저장 */
14     static int avail = 0; /* 배열 poly에서 다음 값을 저장할 수 있는 사용 가능한 인덱스 */
15     static int startA, startB, finishA, finishB; /* 각 다항식을 공유되는 한 배열에 저장하기 위해서, 각 다항식의 시작과 끝 인덱스를 변수로 둠 */
16     static int startD, finishD; /* A(x)와 B(x)를 더한 결과 다항식인 D(x)가 poly 배열에 저장되는 시작과 끝 인덱스 */
```

Public class에서는 각 method 밖의 필드에서 static 변수들을 선언해주었다. 이 public class, Padd에 있는 모든 메소드들이 객체를 따로 만들지 않고 공유의 개념으로 쉽게 접근할 수 있는 static 변수들은 C언어의 public을 대신한다. MAX_TERM은 다항식들을 저장하는 배열의 요소 개수를 제한해주기 위함이다. 코드 상으로 public class 밖에서 새로운 class로 선언해준 Poly 타입의 배열 poly를 선언해주었고, 각 index 당 coef, expon 변수를 지니게 되었다. 이 배열에 덧셈을 위한 두 다항식과 결과 다항식이 모두 저장될 것이기 때문에, 여러 메소드를 오갈 때 필요한 인덱스들 역시 static으로 선언해주었다.

```
19    public static void main(String[] args) {
20
21    /* main 메서드에서는 두 다항식을 입력받아 공유하는 배열에 저장하고, 덧셈을 해주는 padder 메서드를 호출함 */
22
23        /* NullPointerEXCEPTION을 방지하기위해 배열 안의 각 object를 초기화 */
24        for(int i = 0; i<poly.length; i++) {
25            poly[i] = new Poly();
26        }
```

main 메소드에서는 위 주석이 설명하듯 두 다항식을 입력 받아 poly라는 배열에 저장하고 덧셈을 실제로 담당하는 메소드를 호출하는 역할을 담당하도록 설계하였다. Poly class를 만들고 Poly 타입의 배열을 선언했기 때문에, 위 코드의 23-26행이 과정을 거치치 않으면 후에 poly 배열에 값을 대입할 때 NullPointerException이 발생한다. 따라서 poly 배열 안의 각 class object를 초기화해주는 작업을 해주었다.

```
36          /* 아무것도 입력하지 않고 엔터(개행)를 누를 경우에 대한 예외 처리 */
37          if(temp.equals("")) {
38              System.out.println("다항식을 입력하지 않아서 종료합니다.");
39              System.exit(1);
40          }
```

기본적으로 두 다항식을 입력해야 덧셈을 할 수 있다. 따라서 입력 실수 등으로 아무 것도 입력하지 않고 엔터를 누를 경우에 대한 예외 처리를 해주었다.

```
42          /* string으로 입력받은 다항식을 배열에 저장하기 위해서 공백을 구분자로 잘라서 임시 배열에 저장 */
43          String arr[] = temp.split(" ");
44
45          startA = avail; /* 첫번째 다항식을 poly 배열에 저장을 시작할 수 있는 위치 */
46
47          /* arr배열에서 계수와 차수를 poly배열에 저장, arr배열에는 계수와 차수가 입력 순서대로 저장되어 있음 */
48          for(int i = 0; i<arr.length ;avail++) {
49              poly[avail].coef = Float.parseFloat(arr[i]);
50              i++;
51              poly[avail].expon = Integer.parseInt(arr[i]);
52              i++;
53              }
```

위는 입력을 받아 각 다항식을 어떻게 poly 배열에 저장하는지에 대한 코드다. 엔터를 기준으로 한 줄의 string을 입력 받은 후, 공백(스페이스)를 구분자로 입력 string을 token으로 잘라서 임시로 선언한 arr 배열에 차례대로 저장한다. 이때 계수와 차수가 번갈아 차례대로 저장된다. 프로그램을 시작하면, avail은 초깃값이 0이므로, 첫 다항식이 poly 배열에 저장되기 시작하는 인덱스인 startA는 0이 된다. arr 배열에 임시로 저장되어 있던 계수와 차수는 차례대로 for문을 통해 인덱스 i를 하나씩 올려가며 차례로 poly 배열의 계수와 차수로 같은 poly 배열의 인덱스에 쌍으로 저장된다. poly 배열의 한 인덱스에는 coef와 expon이 쌍으로 있으므로, 이 둘을 저장한 후에 avail 인덱스를 1씩 증가시킨다.

```
45          /* 계수와 지수 짝이 맞지 않는 경우에 대한 예외처리 */
46          if(arr.length%2 != 0) {
47              System.out.println("계수와 지수의 짝이 맞지 않아서 종료합니다.");
48              System.exit(1);
49          }
```

(추가**) 입력이 계수 지수 짝이 맞지 않게 홀수로 들어온 경우에 대한 예외처리를 추가했다.

```
57          /* poly 배열에 최종적으로 값을 저장하기위해 사용한 string과 배열의 reference를 clear해줌
58           나중에 garbage로 처리가 됨
59             => C언어의 free()를 대체하기 위함임 */
60          temp = null;
61          arr = null;
```

C언어의 free()를 대체하려고 입력을 받아 저장하는데 사용된 string 변수와 배열의 참조값을 clear해주었다. 이는 후에 garbage collector가 작동할 때, garbage로 처리된다.

```
88          /* 두개의 다항식 덧셈 함수 호출 */
89          padder();
90
91          /* garbage collector로 free space를 reclaim해줌 */
92          System.gc();
93
94      } // main 메소드의 끝
```

같은 방식으로 두번째 다항식도 입력을 받아 저장한 후, 두 다항식을 덧셈해줄 padder 메소드를 호출한다. 이때, 각 다항식이 저장된 위치를 알려주는 시작,끝 인덱스나 다음 사용가능한 위치를 알려주는 인덱스를 parameter로 넘겨주지 않아도 되는 이유는 앞서 이들을 static 변수로 선언해 public class 내의 모든 메소드가 공유하고 쉽게 접근할 수 있도록 해주었기 때문이다. garbage collector인 System.gc()는 앞에서 언급한 것처럼 free space를 reclaim해준다.

```
157     static int compare(int x, int y) {
158
159     /* compare 메소드에서는 두 다항식의 항의 계수를 비교해서 결과를 정수로 반환함 */
160
161         /* case analysis
162          x<y -> z = -1
163          x==y -> z = 0
164          x>y -> z = 1
165          */
166         int z = (((x) < (y))? -1: ((x)==(y)) ? 0 : 1);
167
168         return z;
169     } // compare 메소드의 끝
```

padder()로 흐름을 넘기기 전에, padder() 메소드 내에서 호출 되는 두가지 메소드, compare과 attach를 먼저 살펴보면, compare는 수업 자료에서 C언어의 #define으로 매크로로 선언된 함수였다. JAVA로 구현하면서, 이를 static 메소드로 구현해주었고, 기능은 두 다항식의 각 항들의 차수를 비교하기 위함으로 그 결과를 -1, 0, 1로 반환하여 padder() 메소드의 구현을 간단케 해준다.

```
172    static void attach(float coefficient, int exponent) {
173
174    /* attach 메소드에서는 padder 메소드로부터 결과 다항식인 D(x)에 넣을 각 항의 계수와 차수를 넘겨 받아 poly 배열에 저장함 */
175
176        /* 두 항의 연산 결과를 저장할 시, poly 배열의 최대 저장 가능한 항 개수를 넘으면 프로그램을 종료함
177           ArrayOutOfBoundEXCEPTION을 미리 방지하기 위함 */
178        if(avail>=MAX_TERM) {
179            System.out.println("Too many terms in the polynomial!");
180            System.exit(1);
181        }
182
183        /* 계수가 0이 아닐 때만 저장하고, 0일 때는 저장하지 않고 padder 메소드로 돌아감*/
184        if(coefficient!=0) {
185        poly[avail].coef = coefficient;
186        poly[avail].expon = exponent;
187        avail++;
188        }
189
190    } // attach 메소드의 끝
```

attach 메소드에서는 두 다항식의 연산 결과를 poly 배열에 저장하는 역할을 수행한다. 이때, 연산을 항별로 함으로 저장도 한 항씩 한다. 178-181행은 저장가능한 위치가 이미 선언된 poly 배열의 크기(요소 개수)를 넘었을 때에 대한 예외처리다. 그 아래쪽에서 계수가 0일 때는 저장하지 않는 조건을 달아준 것은, 입력 받을 때 0이 계수로 들어와 다른 다항식과 동차항이 없어 그대로 계수가 0인 항이 결과 다항식의 항으로 저장되지 않게 하기 위함이다.

```
104        /* 두 다항식의 차수를 비교하며 연산하는 부분 */
105        while(startA<=finishA && startB<=finishB) {
106            switch(compare(poly[startA].expon, poly[startB].expon)) { // A(x)와 B(x)의 항의 차수를 비교
107                case -1: // A expon < b expon 때, B(x) 항을 D(x)에 넣음
108                    attach(poly[startB].coef, poly[startB].expon);
109                    startB++;
110                    break;
111                case 0: // A expon == b expon 때, 두 다항식의 동차항의 계수를 더해서 D(x)에 넣음
112                    coefficient = poly[startA].coef + poly[startB].coef;
113                    if (coefficient !=0 ) { // 두 계수의 합이 0이라면 D(x)에 굳이 저장하지 않아도 됨
114                        attach(coefficient, poly[startA].expon);
115                    }
116                    startA++;
117                    startB++;
118                    break;
119                case 1: //// A expon > b expon 때, A(x)의 항을 D(x)에 넣음
120                    attach(poly[startA].coef, poly[startA].expon);
121                    startA++;
122            } // switch 문의 끝
123        } // while 문의 끝
```

padder() 메소드는 실제 다항식 덧셈을 해서 attach 메소드로 결과 다항식의 항으로 저장할 계수와 차수를 넘겨주는 역할을 한다. 이때 결과 다항식의 항을 연산하는 것은 크게 2가지 부분으로 나뉜다. 위의 while문에 해당하는 부분이 첫번째 부분인데, 두 다항식을 한 항씩 각자의 인덱스 범위 내에서 비교하며 연산하는 부분이다. 이때 각 다항식의 항의 차수를 compare 메소드로 넘겨 두 다항식이 동차항이면 계수 덧셈을 하고, 어느 한 항이 더 큰 차수를 가지면 그 항을 결과 다항식으로 먼저 넘긴다. 그리고 이때 결과 다항식으로 넘어가지 못한 차수가 작은 항은 상대 다항식의 다음 항과 또 비교를 하게 된다. 이때 compare 메소드에서 int 형태로 차수의 비교 결과를 반환하기 때문에 switch문으로 쉽게 구현이 가능하다.

```
125         /* 두 다항식의 차수를 비교하면서 D(x)에 값을 넣는 연산이 끝난 후, 남은 한 다항식의 항들을 D(x)에 넣어줌 */
126             /* A(x)의 남은 항을 D(x)에 넣음 */
127         for(; startA<=finishA; startA++) {
128             attach(poly[startA].coef, poly[startA].expon);
129         }
130             /* B(x)의 남은 항을 D(x)에 넣음 */
131         for(; startB<=finishB; startB++) {
132             attach(poly[startB].coef, poly[startB].expon);
133         }
```

padder() 메소드의 두번째 부분은 어느 한 다항식이 모두 결과 다항식에 반영이 되고, 다른 다항식의 남은 항을 처리하는 위의 코드다. 즉, 두 다항식의 항끼리 상호 비교가 끝난 후의 연산이며, 각 다항식은 start인덱스부터 결과 다항식에 마저 반영을 해줘야한다. 각 다항식이 저장된 인덱스 범위 내에서 남은 항들을 연산한다.

이후의 코드는 결과 다항식을 출력하기 위한 부분이며, 결과 다항식은 D(x)로 시작과 끝 인덱스를 startD, finishD로 앞쪽에서 선언하고 값이 할당되었다. 따라서 이 범위 내의 저장된 계수와 지수 요소를 출력하고 실수인 계수(coef)는 과제 예시에 따라 소수점 아래 3자리까지 출력되도록 했다.

따라서 출력까지 padder() 메소드에서 담당하며, 이 과정이 끝나면 이 메소드를 호출한 main 메소드로 돌아가 garbage collector가 작동하고 전체 프로그램 작동이 끝나게 된다.

3. 예시 결과

- 몇가지 경우에 대해서 임의의 입력을 주어 구현한 프로그램이 제대로 작동하는지 확인해보았다.

```
첫번째 다항식을 입력하세요.
4 4 2 2 5 1 9 0
두번째 다항식을 입력하세요.
7 4 3 3 5 2 4 1 10 0
11.000x^4 + 3.000x^3 + 7.000x^2 + 9.000x^1 + 19.000
```

(과제 예시 아웃풋과 동일함)

```
첫번째 다항식을 입력하세요.

다항식을 입력하지 않아서 종료합니다.
```

(아무것도 입력 않고 엔터를 누를 경우에 대한 예외처리가 잘 작동하고 있음)

```
첫번째 다항식을 입력하세요.
1 3 2
계수와 지수의 짝이 맞지 않아서 종료합니다.
```

(계수 지수 짝이 맞지 않게 들어온 경우에 대한 예외처리의 작동)

```
첫번째 다항식을 입력하세요.
0 8 9 0
두번째 다항식을 입력하세요.
3 4 11 0
3.000x^4 + 20.000
```

(계수가 0으로 입력된 동차항이 없는 항은 결과에서 제외된 것을 확인할 수 있음)

```
첫번째 다항식을 입력하세요.
7 6 5 4 4 3
두번째 다항식을 입력하세요.
3 2 4 1 7 0
7.000x^6 + 5.000x^4 + 4.000x^3 + 3.000x^2 + 4.000x^1 + 7.000
```

(두 다항식의 연산 결과가 내림차순으로 잘 출력됨)

-THE END☺

고맙습니다☺☺☺