**public** **class** PolyTest {

**public** **static** **void** main(String[] args) {

System.***out***.println("학번 : 20145165, 이름 : 정균모");

Polynomial1 p1 = **new** Polynomial1();

Polynomial1 p2 = **new** Polynomial1();

//p1(x) = 3x^4 + 4x^3 + 6x^a +8

p1.addTerm(8,0);

p1.addTerm(6,1);

p1.addTerm(4,3);

p1.addTerm(3,4);

System.***out***.println("p1(x) = " + p1);

//p2(x) = 9s^7 + 3x^6 + 2x^3 + 4x^2

p2.addTerm(4,2);

p2.addTerm(2,3);

p2.addTerm(3,6);

p2.addTerm(9,7);

System.***out***.println("p2(x) = "+p2);

System.***out***.println();

p2.delTerm(7);

System.***out***.println("p2(x) = "+p2);

System.***out***.println();

System.***out***.println("p1(x) \* 2x^3) =" + p1.sMult(2,3));

System.***out***.println("p1(x) + p2(x) =" + p1.polyAdd(p2));

System.***out***.println("p1(x) \* p2(x) =" + p1.polyMult(p2));

}

}

**public** **class** Polynomial1 {

**private** **int** coef[];//coef배열의 index는 항의 지수를 의미, 원소는 항의 계수를 의미

**private** **int** degree;//다항식의 제일 큰 지수

//degree는 0으로 초기화

//계수 배열은 크기 50으로 할당

**public** Polynomial1(){

**int** degree =0;

**int** [] coef = **new** **int**[50];

}

//다항식이 빈다항식인지 아닌지 판단

**public** **boolean** ispZero(){

**for**(**int** i=0; i<coef.length; i++){

**if**(coef[i]==0)

**return** **true**;

**else**

**return** **false**;

}**return** **false**;

}

//지정한 지수를 갖고있는 항의 계수를 리턴

**public** **int** coef(**int** e){

**for**(**int** i =0; i<coef.length;i++){

**if**(i==e) **return** coef[i];

}**return** 0;

}

//다항식의 제일 큰 지수를 리턴

**public** **int** maxExp(){

**return** degree;

}

//다항식에서 항을 추가하는 메소드

//c는 추가하고 싶은 항의 계수

//e는 추가하고 싶은 항의 지수

**public** **void** addTerm(**int** c, **int** e){

**if**(coef[e] ==0){

coef[e] =c; //지수가 e인 항 값이 0일 때만 추가 가능

degree=e;

}**else**

**if**(degree>e)

coef[e] = c;

}

//다항식에서 지정한 항을 삭제하는 메소드

//e는 삭제하고 싶은 항의 지수

**public** **void** delTerm(**int** e){

coef[e] =0;

**if**(degree ==0)

**for**(**int** i =0; i<degree; i--){

**if**(coef[i]==0)

degree = i;

}

}

//다항식은 사용자가 지정한 항과 곱하는 메소드

//c는 연산자인 항의 계수

//e는 연산자닌 항의 지수

**public** Polynomial1 sMult(**int** c, **int** e){

Polynomial1 p = **new** Polynomial1();

**for**(**int** i =0; i<=degree; i++){

p.coef[i+e] = **this**.coef[i]\*c;

}

p.degree = **this**.degree+e;

**return** p;

}

//현재 다항식과 지정한 p다항식과 더하는 메소드

**public** Polynomial1 polyAdd(Polynomial1 p){

Polynomial1 newp = **new** Polynomial1();

**int** bigger=0;

**if**(newp.degree > **this**.degree){

bigger=newp.degree ;

}**else** **if** (newp.degree < **this**.degree){

bigger=**this**.degree ;

}**else** bigger=**this**.degree;

**for**(**int** i=0; i<= bigger;i++){

newp.coef[i] = **this**.coef[i] + newp.coef[i];

}newp.degree = bigger;

**return** newp;

}

//현재 다항식과 곱함

//sMult 메소드 호출하여 사용

**public** Polynomial1 polyMult(Polynomial1 p){

Polynomial1 s = **new** Polynomial1();

**int** bigger = 0;

**for**(**int** i =0; i<=degree; i++){

**if**(coef[i] !=0)

Polynomial1 t = **this**.sMult(p.coef[i],i);

s= t.polyAdd(t);

} //sMult로 곱한 후 polyAdd로 각항을 더해줌

}**return** s;

}

**public** String toString(){

String result =" ";

**for**(**int** i = degree; i>=0; i--){

**if**(coef[i] !=0)

**if**(i<degree)

result += "+";

**if**(i>0)

result += coef[i] + "x^" +i;

**else** **if**(i==0)

result +="0";

**return** result;

}

}

}

**public** **class** Term {

**private** **int** coef;

**private** **int** exp;

**public** Term(**int** c, **int** e){

coef = c;

exp = e;

}

**public** **int** getCoef(){

**return** coef;

}

**public** **int** getExp(){

**return** exp;

}

}

**public** **class** Polynomial2 {

**private** Term terms[]; //terms배열

**private** **int** noOfTerms = 0; //noOfTerms의 초기값은 0

**public** Polynomial2(){

terms = **new** Term[100]; //terms 배열 크기 100

noOfTerms = 0; //noOfTerms 0으로 초기화

}

**public** **boolean** ispZero(){ //빈 다항식인지 아닌지를 판별하는 메소드

**if**(noOfTerms ==0) //만약 noOfTerms(인덱스 값)가 0이라면

**return** **true**;

**else**

**return** **false**;

}

**public** **int** coef(**int** e){ //지수가 e인 항의 계수를 리턴하는 메소드

**for**(**int** i =0; i<noOfTerms; i++){ //i가 noOfTerms보다 작을 때 까지, i를 하나씩 증가

**if**(terms[i].getExp()==e) //만약 terms배열 i번째 항의 지수가 e랑 같을 때

**return** terms[i].getCoef(); //terms배열의 i번째 계수 리턴

}

**return** 0;

}

**public** **int** maxExp(){ //제일 큰 지수가 무엇인지 찾는 메소드

**if**(noOfTerms ==0) //만약 noOfTerms(인덱스 값)가 0이면

**return** -1; //리턴 -1

**else**

**return** terms[0].getExp(); //terms의 0번째 항의 지수 리턴

}

**public** **void** addTerm(**int** c, **int** e){ //항 덧셈(c:항의 계수, e:항의 지수) 메소드

Term newTerm = **new** Term(c,e); //매개변수를 c와 e로 받는객체 newTerm 생성

**int** index = findNextIndex(e); //인덱스 값은 findNextIndex의 e매개변수

**if**(index>=0 && noOfTerms<100){ //만약 인덱스가 0보다 크거나 같고 noOfTerms가 100보다 작다면

**for**(**int** i =noOfTerms; i>index; i--) //i가 인덱스보다 클 때 까지, i를 하나씩 감소

terms[i] =terms[i-1]; //배열의 원소들을 하나씩 뒤로 미룬다

terms[index] = newTerm; //terms의 인덱스는 객체 newTerm과 같다

noOfTerms++; //noOfTerms를 하나씩 증가

}

}

**public** **void** delTerm(**int** e){ //지수(e)값에 해당하는 항을 삭제하는 메소드

**int** index = findIndex(e); //인덱스 값은 찾는 인덱스 값(e)

**if**(index>0){ //만약 인덱스 값이 0보다 크면

**for**(**int** i =index; i<noOfTerms-1; i++) //i가 noOfTerms-1보다 작을 때 까지 , i를 하나씩 증가

terms[i] = terms[i+1]; //terms 배열의 원소를 하나씩 앞으로 당긴다

terms[--noOfTerms]=**null**; //noOfTerms를 감소시킬 값이 없어진다

}

}

**public** Polynomial2 sMult(**int** c, **int** e){ //다항식의 곱(c:항의 계수, e:항의 지수)을 하는 메소드

Polynomial2 p = **new** Polynomial2(); //매개변수를 c와 e로 받는 p라는 객체 생성

**int** exp, coef; //exp(지수)와 coef(계수)를 정수형 변수로 선언

**for**(**int** i=0; i<noOfTerms; i++){ //i가 noOfTerms보다 작을 때 까지, i를 하나씩 증가

exp = terms[i].getExp()+e; //지수끼리 더하기

coef = terms[i].getCoef()\*c; //계수끼리 곱하기

p.terms[i] = **new** Term(coef,exp); //새로 계산 된 exp과 coef

}

p.noOfTerms = noOfTerms; //p객체의 noOfTerms와 noOfTerms가 같으면

**return** p; //리턴 p

}

**public** Polynomial2 polyAdd(Polynomial2 p){ //다항식과 다항식을 더하는 메소드

Polynomial2 newP = **new** Polynomial2(); //newP라는 객체 생성

**int** thisIndex = 0; // 초기값 =0

**int** pIndex = 0; // 초기값 =0

**while**(thisIndex<**this**.noOfTerms && pIndex<p.noOfTerms){ //thisIndex가 this.noOfTerms보다 작고 pIndex가 p.noOfTerms보다 작으면

**if**(**this**.terms[thisIndex].getExp()==p.terms[pIndex].getExp()){

//현재 terms에 있는 인덱스의 지수와 p.terms에 있는 인덱스의 지수가 같으면

newP.addTerm(**this**.terms[thisIndex].getCoef()+p.terms[pIndex].getCoef(), p.terms[pIndex].getExp());

//newP의 addTerm은 현재 terms에 있는 인텍스의 계수와 p.terms에 있는 인덱스의 계수를 더하고, p객체의 terms에 있는 인덱스의 지수이다

thisIndex++; //thisIndex값 하나씩 증가

pIndex++; //pIndex값 하나씩 증가

}

**else** **if**(**this**.terms[thisIndex].getExp()<p.terms[pIndex].getExp()){

//현재 terms에 있는 인텍스의 지수가 p.terms에 있는 인덱스의 지수보다 작으면

newP.addTerm(p.terms[thisIndex].getCoef(), p.terms[pIndex].getExp());

//newP의 addTerm은 p객체의 terms에 있는 인덱스의 계수와 p객체의 terms에 있는 인덱스의 지수이다

pIndex++; //pIndex값 하나씩 증가

}

**else**{ //그렇지 않으면

newP.addTerm(**this**.terms[thisIndex].getCoef(), **this**.terms[thisIndex].getExp());

//newP의 addTerm은 현재 terms에 있는 인덱스의 계수와 현재 terms에 있는 인덱스의 지수이다

thisIndex++; //thisIndex값 하나씩 증가

}

}

**while**(thisIndex<**this**.noOfTerms){ //thisIndex가 noOfTerms보다 작을때 까지 반복

newP.addTerm(**this**.terms[thisIndex].getCoef(), **this**.terms[thisIndex].getExp());

thisIndex++; //thisIndex값 하나씩 증가

}

**while**(pIndex<p.noOfTerms){ //반복문 -pIndex가 p객체의 noOfTerms보다 작을때까지

newP.addTerm(p.terms[pIndex].getCoef(), p.terms[pIndex].getExp());

//newP의 addTerm은 p객체의 terms에 있는 pIndex의 계수와 p객체의 term에 있는 pIndex의 지수이다

pIndex++; //pIndex값 하나씩 증가

}

**return** newP; //리턴 newP

}

**public** **void** print(){ //출력 하는 메소드

**for**(**int** i=0; i<noOfTerms;i++){ //반복문 -초기값:i=0, 조건:i가 noOfTerms보다 작을 때 까지, i를 하나씩 증가

**if**(i==noOfTerms-1) //만약 i가 noOfTerms-1과 같으면

System.*out*.println(terms[i].getCoef()+"x^"+terms[i].getExp());

**else** //같지 않으면

System.*out*.println(terms[i].getCoef()+"x^"+terms[i].getExp()+"+");

}

}

**private** **int** findNextIndex(**int** e){ //지수(e)값에 해당하는 다음번째 인덱스를 찾는 메소드

**int** i;

**for**(i=0; i<noOfTerms;i++){ //조건:i가 noOfTerms보다 작을 때 까지, i를 하나씩 증가

**if**(terms[i].getExp()==e) //만약 terms배열의 i번째 지수가 다음 번째 찾는 지수라면

**return** -0; //리턴 -0

**if**(terms[i].getExp()<e) //만약 terms배열의 i번째 지수가 다음 번째 찾는 지수보다 작다면

**return** i; //리턴 i

}

**return** noOfTerms; //for문의 조건을 만족시키지 못하면 리턴 noOfTerms

}

**private** **int** findIndex(**int** e){ //지수(e)값에 해당하는 인덱스를 찾는 메소드

**for**(**int** i=0; i<noOfTerms; i++){ ////반복문 -초기값:i=0, 조건:i가 noOfTerms보다 작을 때 까지, i를 하나씩 증가

**if**(terms[i].getExp()==e) //만약 terms의 i번째 지수가 찾는 지수라면

**return** i; //리턴 i

}

**return** -1; //for문의 조건을 만족시키지 못하면 리턴 -1

}

}