제2장
클래스와 객체

2.1 클래스, 객체, 참조변수

클래스

- 한 사람의 “이름”과 “전화번호”는 항상 같이 붙어다녀야 하는 데이터이다.
- 이 두 가지 데이터를 서로 별개의 변수에 저장하면 우리가 이름 데이터를 옮길 때 마다 전화번호 데이터도 따로 옮겨줘야 한다.
- 만약 각 사람에 대해서 이름과 전화번호 뿐만 아니라 주소, 이메일 등 여러 가지 데이터를 저장한다면 이 불편은 더 심해질 것이다.
- 인덱스 메이커 프로그램에서도 “하나의 단어”와 그 단어의 “등장 횟수”는 항상 같이 붙어다녀야 하는 데이터이다.
- 이렇게 서로 관련있는 데이터들을 하나의 단위로 묶어두면 편할 것이다.
- 이것이 클래스라는 개념이 등장하는 가장 기본적인 이유이다.

Person1.java

```
public class Person1 {
    public String name;
    public String number;
}
```

Java에서 각각의 클래스는 별개의 파일에 저장되어야 하고 파일의 이름은 클래스의 이름과 일치해야 한다. 즉 클래스 Person1은 Person1.java라는 이름의 파일로 저장해야한다.

클래스의 이름은 항상 대문자로 시작하는 것이 관습이다.

이 클래스의 주 목적은 한 사람의 이름(name)과 그 사람의 전화번호(number)를 하나의 단위로 묶어두는 것이다. 여기서 name과 number를 클래스 Person1의 **필드(field)**라고 부른다. 즉 클래스 Person1은 name과 number라는 두 개의 필드를 가지고 있다.

Code01.java

```
public class Code01 {

    public static void main(String[] args) {
        Person1 first;
        first = new Person1();
        first.name = "John";
        first.number = "0511233456";
        System.out.println("First is " + first.name + " with number " + first.number);

        Person1 [] members = new Person1 [100];
        members[0] = first;
        members[1] = new Person1();
        members[1].name = "David";
        members[1].number = "024343432";

        System.out.println("Second is " + members[1].name + " with number " + members[1].number);
    }
}
```

하나의 Person1 타입의 객체(object)를 생성하고 그것을 first라고 한다.

first객체의 name필드와 number 필드에 이름과 전화번호를 각각 저정한다.

first.number는 first가 가리키는 Person1 객체의 number 필드를 의미한다.

Person1 타입의 배열도 만들 수 있다.

클래스

- 클래스는 결국 하나의 **"타입"**이다. 마치 int, double 등 처럼.
- 다만 int, double 처럼 Java가 미리 정해놓은 타입이 아니라 사용자가 정의한 새로운 타입이라는 의미에서 **"사용자 정의 타입"**이라고 부르기도 한다.
- int 혹은 double 형 변수를 선언하고 사용하는 것처럼 Person1형 변수를 선언하고 사용한다.

```
int count = 0;
Person1 first = new Person1();
```

프리미티브 타입과 클래스: 차이점

- int 혹은 double 형 변수를 선언하고 사용하는 것처럼 Person1형 변수를 선언하고 사용한다.

```
int count = 0;
Person1 first = new Person1();
```

클래스와 객체

Person1 first;

first = new Person1();

first.name = “John”; first.number = “010234546”;

The rule is very simple.

모든 프리미티브 타입의 변수는 보통 변수이다.
즉 변수 자체에 값이 저장된다.

프리미티브 타입이 아닌 모든 변수는
참조 변수이다.

즉 실제 데이터가 저장될 “객체”는 new 명령으로 따로 만들어야 하고, 참조변수에는 그 객체의 주소를 저장한다.

Code01_2.java

```
public class Code01_2 {

    public static void main(String[] args) {
        Person1 first = new Person1();
        first.name = "John";
        first.number = "0511233456";

        Person1 second = first;
        second.name = “Tom”;
        System.out.println(first.name + “, " + first.number);

        Person1 [] members = new Person1 [100];
        members[0] = first;
        members[1] = second;
        System.out.println(members[0].name + “, ” + members[0].number);
        System.out.println(members[1].name + “, ” + members[1].number);
    }
}
```

클래스와 객체

int [] numbers = new int [8];

클래스와 객체

Person1 [] members = new Person1 [8];

members는 배열의 이름이므로 당연히 참조 변수이다.

배열의 각 칸은 Person1 타입이다. 그런데 Person1은 프리미티브타입이 아니다. 따라서 배열의 각 칸도 참조 변수이다.

즉 이 상태에서 배열의 각 칸에 다음과 같이 바로 이름과 번호를 저장할 수는 없다.

members[2].name = "John";

members[2].number = "010232432";

클래스와 객체

```
Person1 first = new Person1();
first.name = “John”;      first.number = 01023456”;
Person1 [] members = new Person1 [8];
members[0] = first;
members[1] = new Person1();
members[1].name = “David";       members[1].number = "024343432";
```

값에 의한 호출:배열

호출문:

```
bubbleSort(data, count);
...
```

1. 호출하는 순간 변수 data 의 값이 호출된 메서드의 매개변수 data2로 복사된다.

2. 그런데 복사된 값은 배열의 주소이다. 그러므로 data와 data2는 실제로는 동일한 배열을 가리키게 된다.

호출된 메서드:

```
public static void bubbleSort(int [] data2, int n) {
    for ( int i=n-1; i>0; i--) {
        for ( int j=0; j<i; j++ ) {
            if (data2[j] > data2[j+1]) {
                int tmp = data2[j];
                data2[j] = data2[j+1];
                data2[j+1] = tmp;
            }
        }
    }
}
```

3. 따라서 배열 data2의 내용을 변경하면 배열 data의 내용도 변경된다.

즉 배열을 매개변수로 넘겨주고 호출된 메서드에서 배열의 값을 수정하면 원본 배열의 값도 수정되는 것은 “값에 의한 호출”의 예외가 아니라 배열의 이름이 참조변수이기 때문에 벌어진 일이다.

Code02.java

```
public class Code02 {
    static Person1 [] members;
    static int count = 0;
    public static void main(String[] args) {
        Scanner in;
        try {
            in = new Scanner(new File("data.txt"));
            while (in.hasNext()) {
                String str1 = in.next();
                String str2 = in.next();
                members[count].name = str1;
                members[count].number = str2;
                count++;
            }
            in.close();
        } catch (FileNotFoundException e) {
            System.out.println("No data file exists."):
            System.exit(1);
        }
        for ( int i=0; i<count; i++ )
            System.out.println(members[i].name + "'s phone number is " + members[i].number);
    }
}
```

데이터 파일로 부터 사람들의 이름과 전화번호를 입력받아 배열 members에 저장한 후, 입력된 순서대로 출력하려고 한다. 이 코드에서 잘못된 점들을 모두 찾아서 수정하라.

Code03.java

이름과 번호를 입력받고
사전순으로 정렬하여 출력한다.

```
public class Code03 {
    static Person1 [] members;
    static int count=0;
    public static void main(String[] args) {
        Scanner in;
        try {
            // Code02와 동일
        } catch (FileNotFoundException e) {
            System.out.println("No data file exists."):
            System.exit(1);
        }
        bubbleSort();
        for ( int i=0; i<count; i++ )
            System.out.println(members[i].name + " " + members[i].number);
}
```

Code03.java (continued)

```
public static void bubbleSort() {
    for ( int i=count-1; i>0; i--) {
        for ( int j=0; j<i; j++ ) {
            if (members[j].name.compareToIgnoreCase(members[j+1].name)>0) {
                Person1 tmpPerson = members[j];
                members[j] = members[j+1];
                members[j+1] = tmpPerson;
            }
        }
    }
  }
}
```

compareToIgnoreCase()는 대소문자 구분을 무시한다.

클래스, 참조변수, 객체

```
public class Person {
    String name;
    String number;
}
```

```
Person a = new Person();
a.name = "John";
a.number = "0108978811";
```

C vs. Java

구분		C	Java
프리미티브 타입 (int, double, char…)	보통 변수	int a = 10; char ch = ‘x’;	int a = 10;; char ch = ‘x’;
	참조변수 (포인터)	int *p; char *q;	
사용자 정의 타입 (struct, class)	보통 변수	struct person a; a.name = “John”; a.number = “01034577”;	
	참조변수 (포인터)	struct person *a; a = malloc(sizeof(…)); a->name = “John”; a->number = “0103428”;	Person b; b = new Person(); b.name = “David”; b.number = “4563434”;

인덱스 메이커 프로그램의 수정

하나의 단어와 그 단어의 등장 빈도를
저장하기 위한 클래스이다.

```
public class Item {
    public String word;
    public int count;
}
```

1장의 Code23을 class Item을 이용하도록
수정하라.

사각형의 면적

- 평면상에 좌표축에 평행한 n개의 직사각형에 관한 데이터를 입력 받은 후 면적이 작은 것부터 큰 것 순으로 정렬하여 출력하는 프로그램을 작성하라.
- 입력 파일의 형식:

```
0 1 2 4          // 왼쪽-위쪽 꼭지점의 좌표가 (0,1)이고, 너비가 2, 높이가 4
1 4 7 8
4 3 12 9
8 18 11 30
5 10 6 11
```

MyPoint1.java

```
public class MyPoint1 {
    public int x;
    public int y;
}
```

2차원 평면에서 정수 좌표를 가지는 하나의 점을 표현하기 위한 클래스. 한 점의 x-좌표와 y-좌표는 항상 붙어다녀야 하므로 하나의 데이터 단위로 취급하기 위해서 이 클래스를 작성

MyRectangle1.java

```
public class MyRectangle1 {
    public MyPoint1 lu;
    public int width;
    public int height;
}
```

좌표축에 평행한 하나의 사각형은 왼쪽-위쪽 꼭지점과 너비, 그리고 높이에 의해서 정의된다.

Code05.java

```
public class Code05        {
    static MyRectangle1 [] rects;;
    static int n=0;
    public static void main(String[] args) {
        try {
            Scanner  source = new Scanner(new File("data.txt"));
            while (source.hasNext())     {
                rects[n].lu.x = source.nextInt();
                rects[n].lu.y = source.nextInt();
                rects[n].wdith = source.nextInt();
                rects[n++].height = source.nextInt();
            }
            source.close();
        } catch (FileNotFoundException e) {
            System.out.println("No data file exists.");
            System.exit(1);
        }
    }
}
```

데이터 파일로 부터 사각형에 관한 정보를 읽어 배열 **rects**에 저장한다. n은 읽은 사각형의 개수이다.

모든 오류를 찾아 수정하라.

객체와 참조변수

참조변수는 **new** 명령으로 생성될 객체의 주소를 저장할 자리일 뿐,
객체 그 자체가 아니다.

Code05.java (Continued)

```
public class Code05      {
    static MyRectangle1 [] rects;;
    static int n=0;
    public static void main(String[] args) {

        // 데이터 파일로 부터 읽는다.

        bubbleSort();
        for (int i=0; i<n; i++) {
            System.out.println(rects[i].lu.x + " " + rects[i].lu.y + " "
                                + rects[i].width + " " + rects[i].height);
    }

    public static void bubbleSort() {
        // exercise
    }
}
```

다항함수

- 다항함수(polynomial)는 항(term)들의 합이며, 항(term)은 계수(coefficient)와 지수(exponent)에 의해서 정의된다. 계수는 0이 아닌 정수이고 지수는 음이 아닌 정수라고 가정한다. 예를 들면,

$$f(x) = -x^5 + 2x^4 - 10x - 3$$

프로그램 실행 예

```
$ create f          // 다항함수 f = 0을 정의한다.
$ add f 2 3         // f(x)에 2x^3을 더한다. 따라서 f(x) = 2x^3 이 된다.
$ add f -1 1        // f(x) = 2x^3 - x 이 된다.
$ add f 5 0         // f(x) = 2x^3 - x + 5 이 된다.
$ add f 2 1         // f(x) = 2x^3 - x + 5 + 2x = 2x^3 + x + 5 이 된다.
$ calc f 2          // x=2일 때 다항함수 f의 값, 즉 f(2) = 23을 계산하여 출력한다.
23
$ print f           // 차수에 관한 내림차순으로 정렬하여 다음과 같이 출력한다.
2x^3 + x + 5        // 동일한 차수의 항은 하나로 합쳐져야 한다.
$ create g          // 다른 다항 함수 g를 정의한다.
....
$ exit
```

class Term

```
public class Term {
    public int coef;
    public int expo;
}
```

class Term은 다항식을 구성하는 하나의 항을 표현하기 위한 클래스이다. 계수(coef)와 차수(expo)를 가진다.

class Polynomial

class Polynomial은 하나의 다항식을 표현하기 위한 클래스이다.

```
public class Polynomial {
    public char name;
    public int n;
    public Term [] terms;
}
```

다항식의 이름이다.

다항식을 구성하는 항의 개수이다.

다항식을 구성하는 항들을 저장할 배열이다.

Code06.java

Watch videos !!!

Task Set #1

1. Code05.java에서 처럼 평면상의 n개의 직사각형을 입력 파일로 부터 읽은 후, 또 다른 한 점의 좌표를 키보드로 부터 입력 받아, 그 점을 포함하는 사각형의 개수를 출력하는 프로그램을 작성하라. 점이 사각형의 가장자리에 놓일 때도 포함되는 것으로 간주한다.

2. Code05.java에서 처럼 평면상의 n개의 사각형을 입력 파일로 부터 입력받 후, 서로 교차하는 사각형쌍들 중 그 교집합의 면적이 최대인 쌍을 찾아서 그 면적을 출력하는 프로그램을 작성하라.

3. 전화번호부 프로그램을 수정하여 다음과 같이 수정하라. 우선 프로그램을 실행하면 data.txt 파일로부터 전화번호부를 읽는다. 데이터 파일에는 사람 이름과 전화번호의 쌍들이 한 줄에 한 명씩 저장되어 있다. 그런 다음 다음과 같이 find, add, remove, list, exit 명령을 처리한다.

```
$ find Kim                    // Kim의 전화번호를 찾아 출력한다.
010-3423-2234
$ add Park 0103457235         // 새로운 사람을 추가한다.
$ remove Lee                  // Lee를 삭제한다.
$ list                        // 모든 사람의 이름과 전화번호를 출력한다.
...
$ exit
```

2.2 메서드와 생성자

메서드와 생성자

- 클래스는 서로 관련있는 데이터들을 하나의 단위로 묶어두기 위한 것이다.
- 하지만 이것이 전부가 아니다.
- 서로 관련있는 데이터들 뿐 아니라, 그 데이터와 관련이 깊은 메서드도 함께 묶어 둘 수 있다.
- 이렇게 함으로써 코드의 응집도(cohesion)를 높이고 결합도(coupling)를 낮출 수 있다.

다항식 프로그램에서

```
class Term {
    int coef;
    int expo;
}

class Polynomial  {
    int n;
    int Term [] terms;
}

class Code06 {
...

    static void printTerm( Term t )
    {
        System.out.println(t.coef + "x^" + t.expo);
    }

    static int calcTerm( Term t, int x )
    {
        return t.coef * Math.pow(x, t.expo);
    }
...
}
```

printTerm과 calcTerm은 class Term과 매우 긴밀하게 연관되어 있는 메서드들이다. 이렇게 서로 관련성이 깊은 데이터와 메서드들을 하나의 클래스로 묶어둘 수 있다.

class Term2

```
class Term2  {
   int coef;
   int expo;
```

calcTerm은 어떤 Term의 일부분이며 "자기 자신"의
값을 계산하여 반환한다.따라서 매개변수로 어떤 항을 받을 필요가 없다.

```
   int calcTerm( int x )
   {
      return coef * Math.pow(x, expo);
   }
```

printTerm은 어떤 Term의 일부분이며
"자기 자신"을 출력하는 기능을 수행한다.
따라서 역시 매개변수로 어떤 항을 받을 필요가 없다.

```
   void printTerm( )
   {
      System.out.println( coef + "x^" + expo);
   }

}
```

클래스, 참조변수, 객체

```
class Term2  {
    int coef;
    int expo;
    int calcTerm( int x ) {
        return coef * Math.pow(x, expo);
    }
    void printTerm( ) {
        System.out.println( coef + "x^" + expo);
    }
}
```

class는 여전히
설계도일 뿐이다.
즉 실체가 아니다.

```
Term2 t = new Term2();
t.coef = 3;
t.expo = 2;
int result = t.calcTerm( 2 );
t.printTerm();
```

우리가 실행하는 것은 클래스가 아니라
각각의 객체 안에 있는 메서드들이다.

각각의 객체 안에
calcTerm과 printTerm 메서드가
존재한다.

객체란 ?

- 객체지향 프로그래밍에서 객체란 "데이터" + "메서드"이다. 데이터는 객체의 "정적 속성"을 표현하며, 메서드는 객체의 "기능 (동적 속성)"을 표현한다.

  자전거는 "모양, 무게, 크기, 브랜드" 등의 정적 속성(데이터)과 "달린다, 정지한다, 뒤로 간다" 등의 기능을 가진다.

  하나의 항(term)은 "계수와 차수"라는 정적 속성과 "x의 값을 주면 자신의 값을 계산해 준다", "화면에 출력해 준다" 등의 기능을 가진다

class Polynomial

마찬가지로 매개변수로 어떤 다항식을 받을 필요가 없다.

```
class Polynomial  {
    int n;
    int Term []  terms;
}
```

```
class Code06 {
    ....
    int calcPolynomial( Polynomial p, int x)
    {
        int result = 0;
        for (int i=0; i<p.n; i++)
            result += calcTerm( p.terms[i], x );
        return result;
    }

    void printPolynomial( Polynomial p) {
    ...
    }
...
}
```

```
class Polynomial2 {
    int n;
    int Term2 []  terms;
    int calcPolynomial( int x )
    {
        int result = 0;
        for (int i=0; i<n; i++)
            result += terms[i].calcTerm( x );
        return result;
    }
    void printPolynomial( ) {
        ...
    }
}
```

다항식 프로그램 (Code07.java)

- 2.1에서 작성한 다항식 프로그램을 수정된 class Term2와 class Polynomial2를 사용하도록 변경하라.

생성자

- 클래스 안에 그 클래스와 동일한 이름을 가지며 return 타입이 없는 특별한 메서드를 둘 수 있다. 이것을 생성자(constructor)라고 부른다.
- 생성자는 new 명령으로 객체가 생성될 때 자동으로 실행된다. 주 목적은 객체의 데이터 필드의 값을 초기화하는 것이다.

class Term3

```
class Term3  {
   int coef;
   int expo;

   public Term3(int c, int e)
   {
      coef = c;
      expo = e;
   }

   int calcTerm( int x )
   {
      return coef * Math.pow(x, expo);
   }

   void printTerm( )
   {
      System.out.println( coef + "x^" + expo);
   }
}
```

클래스와 동일한 이름을 가지며 return 타입이 없는 메서드를 생성자(constructor)라고 부른다.
생성자는 객체를 생성할 때 자동으로 실행된다.

생성자가 없는 Term의 경우 다음과 같이 객체를 생성한 후 따로 데이터 멤버의 값을 초기화하였다.

```
Term2 t = new Term2();
t.coef = 3;
t.expo = 2;
```

생성자가 있는 경우 다음과 같이 객체의 생성과 초기화를 한번에 할 수 있다.

```
Term3 t2 = new Term3(3, 2);
```

class Polynomial3

```
class Polynomial3  {
   int nTerms;
   int Term3 [] terms;

   public Polynomial3() {
      nTerms = 0;
      terms = new Term3 [100];
   }

   int calcPolynomial( int x )
   {
      int result = 0;
      for (int i=0; i<n; i++)
         result += terms[i].calcTerm( x );
      return result;
   }

   void printPolynomial( ) {
      ...
   }
   ....
}
```

생성자가 반드시 매개변수를 받아야하는 것은 아니다.

생성자는 객체에게 필요한 초기화 작업을 하기에 적절한 장소이다. 이 예에서는 배열 terms를 생성하였다.

다항식 프로그램 (Code08.java)

- 앞에서 작성한 다항식 프로그램을 수정된 class Term3과 class Polynomial3을 사용하도록 변경하라.

생성자: MyPoint2.java

```
public class MyPoint2 {
    public int x;
    public int y;

    public MyPoint2(int a, int b) {
        x = a;
        y = b;
    }
}
```

클래스 안에 그 클래스와 동일한 이름을 가지며 return 타입이 없는 메서드를 둘 수 있다. 이것을 생성자(constructor)라고 부른다. 생성자는 객체를 생성할 때 자동으로 실행된다. 주 목적은 객체의 데이터 필드의 값을 초기화하는 것이다.

생성자가 없는 MyPoint1의 경우 다음과 같이 객체를 생성한 후 따로 좌표값을 초기화하였다.

```
MyPoint1 pt1 = new MyPoint1();
pt1.x = 3;
pt1.y = 2;
```

MyPoint2의 경우 다음과 같이 객체의 생성과 초기화를 한번에 할 수 있다.

```
MyPoint2 pt2 = new MyPoint2(3, 2);
```

MyRectangle2.java

```
public class MyRectangle2 {
    public MyPoint2 lu;
    public int width;
    public int height;

    public MyRectangle2(MyPoint2 p, int w, int h) {
        lu = p;
        width = w;
        height = h;
    }

    public MyRectangle2( int x, int y, int w, int h )  {
        lu = new MyPoint2(x,y);
        width = w;
        height = h;
    }

    public int calcArea() {
        return width*height;
    }

    public String toString()        {
        return "LU = (" + lu.x + ", " + luny + "), width=" + width + ", height=" + height;
    }
}
```

하나의 클래스가 2개 이상의 생성자를 가질수 있다. 생성자들은 매개변수의 개수나 타입에 있어서 차이가 있어야 한다. 객체를 생성할 때 어떤 매개변수를 제공하느냐에 따라서 해당되는 생성자가 선택되어 실행된다.

```
MyPoint2 p = new MyPoint2(12, 2);
MyRectangle2 r1 = new MyRectangle2(p, 4, 5);
MyRectangle2 r2 = new MyRectangle2(
    new MyPoint2(1,2),  6, 7);
MyRectangle2 r3 = new MyRectangle2(10, 5, 7, 3);
```

사각형에 관한 정보를 화면에 출력할 일이 자주 있다. 그래서 자신의 정보(좌표, 넓이, 높이 등)을 하나의 String으로 만들어 주는 메서드를 만들었다.

Code09.java

```
public class Code09      {
    static MyRectangle2 [] rects = new MyRectangle2 [100];
    static int n=0;

    public static void main(String[] args) {
        try {
            Scanner  in = new Scanner(new File("data.txt"));
            while (in.hasNext())
                rects[n++] = new MyRectangle2( in.nextInt(), in.nextInt(), in.nextInt(), in.nextInt() );
            in.close();
        } catch (FileNotFoundException e) {
            System.out.println("No data file exists.");
            System.exit(1);
        }
        bubbleSort();
        for (int i=0; i<n; i++)
            System.out.println( rects[i].toString() );
}
```

Code09.java

```
public static void bubbleSort() {
    for (int i=n-1; i>0; i—)  {
        for (int j=0; j<i; j++)    {
            if ( rects[j].calcArea() > rects[j+1].calcArea() )  {
                MyRectangle2 tmp = rects[j+1];
                rects[j+1] = rects[j];
                rects[j] = tmp;
            }
        }
    }
  }
}
```

어떤 이유로 하나의 사각형을 왼쪽 위쪽 꼭지점의 좌표와 너비, 높이 대신 "대각 방향의 두 꼭지점의 좌표"로 표현해야 한다고 가정해보자. class Rectangle2의 경우 프로그램의 다른 부분 즉 class Code09에 전혀 영향을 주지 않고 이런 일을 할 수 있다.

다각형과 점

- 입력으로 하나의 직교 다각형(모든 변이 x-축 혹은 y-축과 평행한 다각형)과 또 하나의 점 p가 주어질 때 그 점이 다각형의 내부에 있는지 외부에 있는지 판단하는 프로그램을 작성하라.
- 입력 형식의 예: 시계방향으로 꼭지점의 좌표가 주어짐

```
8          // 꼭지점의 개수
0 0        // 첫 번째 꼭지점의 x및 y좌표
16 0       // 두 번째 꼭지점의 x및 y좌표
16 18      // ...
10 18
10 6       // 꼭지점들은 시계방향 순서로
6 6        // 제공된다.
6 12
0 12       // 마지막 꼭지점의 x및 y좌표
7 7        // 테스트할 점 p의 좌표
```

내부/외부 검사

- 점에서 시작하여 한 방향으로 무한히 뻗어가는 아무 직선이나 하나 그어서 그것이 다각형의 변과 짝수 번 교차하면 외부, 홀수 번 교차하면 내부에 있다.

자바는 Bottom Up 방식

겹치면 ? 문제를 조금 단순화하기 위해서
여기서는 점 p는 다각형의 경계상에 있지 않고, 또한 다각형에는 p와 동일한 x혹은 y좌표를 가진 꼭지점은 없다고 가정한다.
(꼭지점의 좌표값은 모두 짝수, 검사할 점 p의 좌표값은 홀수로 가정한다.)

OrthoLine.java

```
public class OrthoLine {
    public MyPoint2 u;              // 한쪽 끝점
    public MyPoint2 v;              // 다른쪽 끝점

    public OrthoLine(MyPoint2 p, MyPoint2 q)  {
        u = p;
        v = q;
        if (p.x>q.x || p.x==q.x && p.y>q.y)
            swap();
}

private void swap() {
    MyPoint2 tmp = u;
    u = v;
    v = tmp;
}
```

수직 선분의 경우 위쪽이 u, 아래쪽이 v, 수평선분의 경우 왼쪽이 u, 오른쪽이 v가 되도록 해주었다.

OrthoLine.java

수직선분인지 검사하는 메서드를 만들었다.

두 선분이 교차하는지 검사한다.

```
public boolean isVertical() {
    return u.x==v.x;
}

public boolean intersects(OrthoLine other) {
    if (isVertical() && !other.isVertical())
        return (other.u.x < u.x && other.v.x > u.x && other.u.y > u.y && other.u.y < v.y);
    else if (!isVertical() && other.isVertical())
        return (other.u.y < u.y && other.v.y > u.y && other.u.x > u.x && other.u.x < v.x);
    else
        return false;
  }
}       // end of class OrthoLine
```

OrthoPolygon.java

```
public class OrthoPolygon {
    public int n = 0;                    // 저장된 꼭지점의 개수
    public MyPoint2 [] vertices;         // 꼭지점 들

    public OrthoPolygon(int k) {
        vertices = new MyPoint2 [k];
    }
```

꼭지점의 개수가 k개인 다각형을 만든다.

OrthoPolygon.java (Continued)

```
public void addVertex(int x, int y) {
    vertices[n++] = new MyPoint2(x, y);
}

public int maxX() {
    int max = vertices[0].x;
    for (int i=1; i<n; i++)
        if (vertices[i].x > max)
            max = vertices[i].x;
    return max;
}
```

새로운 꼭지점을 하나 추가하는 메서드이다.

꼭지점들의 x-좌표의 최대값을 찾아서 반환한다.

OrthoPolygon.java (Continued)

점 p가 다각형 내에 포함되는지 검사한다.

점 p에서 왼쪽 방향으로 다각형 외부까지 뻗어가는 하나의 선분 arrow를 만든다. p의 반대쪽 점의 x좌표가 꼭지점들의 x좌표의 최대값보다 1이 크므로 이 점은 반드시 다각형의 외부에 있다.

```
public boolean contains(MyPoint2 p)  {
    OrthoLine arrow = new OrthoLine( p, new MyPoint2(maxX()+1, p.y));
    int count = 0;
    for (int i=0; i<n; i++)  {
        OrthoLine edge = new OrthoLine(vertices[i], vertices[(i+1)%n]);
        if (edge.intersects(arrow))
            count++;
    }
    return count%2 != 0;
  }
}        // end of class OrthoPolygon
```

n-1번째 선분은 마지막 꼭지점(vertex[n-1])과 첫번째 꼭지점(vertex[0])을 연결하는 선분이다.

arrow와 교차하는 다각형의 변의 개수를 카운트한다. 그 결과가 짝수이면 내부, 홀수이면 외부이다.

Code10.java

```
public class Code10  {
    public static void main(String [] args)  {
        try {
            Scanner  in = new Scanner(new File("data.txt"));
            int n = in.nextInt();
            OrthoPolygon thePolygon = new OrthoPolygon(n);
            for (int i=0; i<n; i++)
                thePolygon.addVertex( in.nextInt(), in.nextInt() );
            MyPoint2 p = new MyPoint2(in.nextInt(), in.nextInt());
            in.close();
            if (thePolygon.contains(p))
                System.out.println("Yes");
            else
                System.out.println("No");
        } catch (FileNotFoundException e) {
            System.out.println("No data file exists.");
            System.exit(1);
        }
    }
}
```

꼭지점의 개수를 먼저 입력받아 다각형을 생성한다.

꼭지점의 좌표를 시계방향 순서로 입력받아 다각형에 추가한다.

테스트할 점의 좌표를 입력받아 점 p를 생성한다.

내부인지 외부인지 검사한다.

Task Set #2

1. 입력으로 n개의 수평 혹은 수직 선분이 주어진다. 주어진 선분들에 의해서 만들어지는 사각형 중 가장 면적이 큰 사각형의 면적을 구하여 출력하는 프로그램을 작성하라. 단 그 사각형의 내부에 어떤 다른 선분도 침범하지 않아야 한다. 그런 사각형이 존재하지 않으면 0을 출력한다. 단, “두 선분이 겹치는 경우”, “두 선분의 끝점이 서로 일치하는 경우”, “한 선분의 끝점이 다른 선분 내에 포함되는 경우”는 없다고 가정한다. 입력은 input.txt파일로부터 읽는다. 파일의 첫 줄에는 먼저 n의 값이 주어지고, 이어지는 n줄에 각 줄마다 한 선분의 양 끝점의 좌표인 4개의 정수가 주어진다. 예를 들어 1 2 1 4는 시작점이 (1,2), 끝점이 (1,4)인 수직 선분을 의미한다. 모든 좌표값은 정수이다.

내부에 아무것도 포함하지 않으면서 가장 큰 사각형

두 선분이 겹치는 경우들

두 선분의 끝점이 일치하는 경우들

한 선분의 끝점이 다른 선분에 포함되는 경우

2.3 static 그리고 visibility

클래스와 객체

- 클래스는 타입이다. 집이 아니라 집의 설계도이다. 즉 실체가 아니다.
- 따라서 클래스의 데이터 필드에 데이터를 저장할 수는 없고, 클래스의 멤버 메서드를 실행할 수도 없다. 왜냐하면 실체가 아니므로 !!
- new 명령으로 해당 클래스 타입의 객체를 만든 후, 그 객체에 데이터를 저장하고, 그 객체의 멤버 메서드를 실행하는 것이다.
- 여기에는 하나의 예외가 존재하는데 그것이 static 멤버이다.
- static 멤버는 클래스 안에 실재로 존재하며 객체에는 존재하지 않는다.

Static vs. Non-static

```
public class Test {
    public static int s = 0;
    public int t = 0;

    public static void print1()   {
        System.out.println("s = " + s);
    }

    public void print2()      {
        System.out.println("t = " + t);
    }
}
```

```
public class Test {
    public static int s = 0;
    public int t = 0;
    public static void print1()  {
        System.out.println("s = " + s);
    }
    public void print2()      {
        System.out.println("t = " + t);
    }
}
```

static 멤버는
class 멤버이고

Test test1 = new Test();

test1

```
public static int s = 0;
public int t = 0;
public static void print1() {
    System.out.println("s = " + s);
}
public void print2() {
    System.out.println("t = " + t);
}
```

non-static 멤버는
object 멤버이다.

Test test2 = new Test();

test2

```
public static int s = 0;
public int t = 0;
public static void print1() {
    System.out.println("s = " + s);
}
public void print2() {
    System.out.println("t = " + t);
}
```

Questions

- 왜 main 메서드는 반드시 static이어야 하는가?
- 왜 static 메서드에서 같은 클래스의 non-static 멤버를 엑세스 할 수 없는가?
- 다른 클래스에 속한 static 멤버는 어떻게 엑세스하는가?
- static 메서드/필드의 용도는?

static 멤버의 용도

- main 메서드
- 상수 혹은 클래스 당 하나만 유지하고 있으면 되는 값(혹은 객체)

  예: Math.PI, System.out
- 순수하게 기능만으로 정의되는 메서드. 대표적인 예로는 수학 함수들

  예: Math.abs(k), Math.sqrt(n), Math.min(a,b)

다항식 프로그램

- 앞에서 작성했던 다항식 프로그램들을 오직 main 메서드만 static이 되도록 수정해보자.

접근 제어: public, private, default, protected

- public: 클래스 외부에서 접근이 가능하다.
- private: 클래스 내부에서만 접근이 가능하다.
- default: 동일 패키지에 있는 다른 클래스에서 접근 가능하다.
- protected: 동일 패키지의 다른 클래스와 다른 패키지의 하위클래스에서도 접근 가능하다.

데이터 캡슐화

- 모든 데이터 멤버를 private으로 만들고 필요한 경우에 public한 get/set 메서드를 제공한다.
- 이렇게 하면 객체가 제공해주는 메서드를 통하지 않고서는 객체 내부의 데이터에 접근할 수가 없다.
- 이것을 data encapsulation 혹은 information hiding이라고 부른다.

다항식 프로그램

- 모든 데이터 멤버를 private로 바꾸어보고, public 멤버를 최소화해보자.

전화번호부

```
// 프로그램을 시작하면 자동으로 data.txt파일을 읽어온다.
$ add Lee 348756384          // 새로운 사람을 추가할 수 있다.
$ find Park
Park's number is 32864283
$ find Kwon
No such person exists.
$ list
John 0104623482
Park 32864283
Kim 738568345
David 365482356
Lee 348756384
$ save                       // 변경된 사항이 data.txt에 저장된다.
$ exit
```

data.txt의 예

```
John 0104623482
Park 32864283
Kim 738568345
David 365482356
```

ConsoleUI
PhoneBook theBook;
void processCommand();
MyApp
static void main()
PhoneBook
Person [] members;
int n;
void loadData(String fileName);
String findPerson(String name);
void addPerson(String name,
String number);
void save();
String toString();
Person
String name;
String number;
String toString();

Person.java

```
public class Person {
    private String name;
    private String number;

    public Person(String name, String number)
        this.name = name;
        this.number = number;
    }

    public String getName() {
        return name;
    }

    public String getNumber() {
        return number;
    }

    public void setNumber(String number) {
        this.number = number;
    }

    public String toString()  {
        return "Name: " + name + ", Number: " + number;
    }
}
```

name과 number는 private 필드이다. private 멤버는 이 클래스 내부에서만 엑세스할 수 있고 외부에서는 사용할 수 없다.

class Person의 데이터 멤버인 name, number와 생성자의 매개변수인 name, number가 "우연히" 같은 이름을 가지고 있다. 이런 경우 클래스의 데이터 멤버임을 표시하기 위해서 this.name 과 같이 표현한다.

name과 number를 private 멤버로 만드는 대신 각각을 읽고/쓰기 위한 public 메서드들을 제공한다. 이제 이 클래스의 외부에서 name이나 number를 읽거나 쓰려면 반드시 이 메서드들을 이용해야 한다. 이 메서드들은 보통 getter/setter혹은 accessor/mutator라고 불린다.

setNumber 메서드는 있지만 setName 메서드는 제공하지 않는다. 따라서 일단 Person 객체가 만들어지고 나면 번호는 수정 가능하지만 이름은 수정가능하지 않다. 필요하지 않은 일은 아예 할 수 없도록 만드는 예이다.

이렇게 클래스의 필드들을 private멤버로 만들고 클래스가 제공하는 public 메서드들을 통해서만 접근하게 하는 기법을 information hiding, 혹은 data encapsulation이라고 부른다.

PhoneBook.java

마찬가지로 모든 데이터 멤버들을 private 멤버로 만들었다.
이 클래스 외부에서는 원래 아무도 members와 count를 직접 엑세스 하지 않으므로 getter/setter 메서드는 불필요하다.

```
public class PhoneBook {
    private static final int INITIAL_CAPACITY = 100;
    private int capacity = INITIAL_CAPACITY;
    private Person [] members;
    private int n = 0;
    private String dataFile;
    private boolean modified = false;

    public PhoneBook() {
        members = new Person [capacity];
    }
```

읽어온 데이터 파일의 이름을 저장해 둔다.

데이터 파일로 부터 읽어온 후 변경사항이 있는지를 표시한다. 변경 사항이 없다면 굳이 다시 파일로 저장할 필요가 없다.

PhoneBook.java

```
public void loadData(String fileName)
{
    dataFile = fileName;
    Scanner source;
    try {
        source = new Scanner(new File(fileName));
        while (source.hasNext())
            addPerson(source.next(), source.next());
        source.close();
    } catch (FileNotFoundException e) {
        System.out.println("No data file exists.");
        System.exit(1);
    }
}
```

data file로 부터 전화번호부 데이터를 읽어온다.

PhoneBook.java (continued)

```
public String findPerson(String targetName) {
    for (int i=0; i<n; i++) {
        if (members[i].getName().equalsIgnoreCase(targetName))
            return members[i].getNumber();
    }
    return null;
}

public boolean addPerson(String name, String number) {
    Person newPerson = new Person(name, number);
    if(n>=members.length)
        reallocate();
    members[n++] = newPerson;
    modified = true;
    return true;
}
```

name을 직접 엑세스하는 대신 getName() 메서드를 이용한다.

찾은 경우 전화번호를 반환한다.

배열의 용량이 꽉 찬 경우 배열의 크기를 늘려준다.

PhoneBook.java (continued)

데이터 파일에 저장한다.

```
public void save()  {
    if (!modified)
        return;
    try {
        PrintWriter pw = new PrintWriter(new FileWriter(dataFile));
        for (int i=0; i<n; i++)
            pw.println(members[i].getName() + " " + members[i].getNumber());
        pw.close();
    }
    catch (IOException e) {
        System.out.println("Save failed.");
        return;
    }
    modified = false;
}
```

저장했으므로 데이터 파일과 프로그램이 가진 데이터의 차이가 없다.

PhoneBook.java (continued)

```
private void reallocate() {
    Person [] tmpArray = new Person [capacity * 2];
    for (int i=0; i<n; i++)
        tmpArray[i] = members[i];
    members = tmpArray;
    capacity *= 2;
}
```

reallocate 메서드는 오직 addPerson 메서드에서만 호출하므로 private 멤버로 만들었다.

PhoneBook.java (continued)

```
public String toString()  {
        StringBuilder sb = new StringBuilder();
        for (int i=0; i<n; i++)
            sb.append(members[i].toString() + "\n");
        return sb.toString();
    }
}
```

전화번호부에 있는 모든 사람의 이름과 번호를 하나의 String으로 만들어 주는 toString() 메서드를 추가하였다.

StringBuilder는 스트링들을 합쳐서 큰 스트링을 만들때 사용한다.

ConsoleUI.java

```
public class ConsoleUI
{
    private PhoneBook theBook;
    private Scanner keyboard;

    public ConsoleUI(PhoneBook book) {
        theBook = book;
    }

    public void processCommand()
    {
        keyboard = new Scanner(System.in);
        boolean finished = false;
        while (!finished) {
            System.out.print("$ ");
            String command = keyboard.next();
            switch (command){
            case "find":
                handleFind();
                break;
```

생성자를 통해 전달된 하나의 PhoneBook 객체를 알고 있다.

ConsoleUI.java (continued)

```
case "add":
        handleAdd();
        break;
case "list":
        System.out.println(theBook.toString());
        break;
case “save”:
        theBook.save();
        case "exit":
                System.out.println("Goodbye !");
                finished = true;
                break;
        default:
                System.out.println("Unsupported command !");
        }
    }
    keyboard.close();
  }
}
```

전화번호부에 있는 모든 사람들을 출력하는 명령이다.

ConsoleUI.java (continued)

```
private void handleAdd() {
    String name = keyboard.next();
    String number = keyboard.next();
    theBook.addPerson(name, number);
}

private void handleFind() {
    String who = keyboard.next();
    String theNumber = theBook.findPerson(who);
    if (theNumber != null)
        System.out.println(who + "'s number is " + theNumber);
    else
        System.out.println("No such person exists.");
  }
}
```

findPerson()은 찾은 사람의 전화번호를 넘겨준다.

MyPDApp.java

```
public class MyPDApp {
    public static void main(String[] args) {
        PhoneBook theBook = new PhoneBook();
        theBook.loadData("data.txt");
        ConsoleUI theConsoleUI = new ConsoleUI(theBook);
        theConsoleUI.processCommand();
    }
}
```

클래스(허상)의 세계

```
public class Person {
  private String name;
  private String number;
}
```

```
public class PhoneBook {
  static int INITIAL_CAPACITY = 100;
  public int capacity = INITIAL_CAPACITY;
  public Person [] members;
  public int n = 0;
  public void loadData(String sourceName)  { ... }
  public String addPerson() { ... }
  public String findPerson(String name) { ... }
}
```

```
public class PDConsoleUI {
  private PhoneBook theBook;
  public void processCommands() { ... }
}
```

```
public class MyPDApp {
  public static void main(String args[])  {
    PhoneBook1 theBook = new PhoneBook();
    theBook.loadData("data.txt");
    PDConsoleUI theConsoleUI =
                    new PDConsoleUI(theBook);
    theConsoleUI.processCommand();
  }
}
```

객체(실체)의 세계

아직 아무 객체도 존재하지 않는다.
다만 static member인 INITIAL_CAPACITY와 main 메서드만이
예외로 실제로 존재한다.

클래스(허상)의 세계

```
public class Person {
  private String name;
  private String number;
}
```

```
public class PhoneBook {
  static int INITIAL_CAPACITY = 100;
  public int capacity = INITIAL_CAPACITY;
  public Person [] members;
  public int n = 0;
  public void loadData(String sourceName)  { ... }
  public String addPerson() { ... }
  public String findPerson(String name) { ... }
}
```

```
public class PDConsoleUI {
  private PhoneBook theBook;
  public void processCommands() { ... }
}
```

```
public class MyPDApp {
  public static void main(String args[])  {
    PhoneBook theBook = new PhoneBook();
    theBook.loadData("data.txt");
    PDConsoleUI theConsoleUI =
                    new PDConsoleUI(theBook);
    theConsoleUI.processCommand();
  }
}
```

객체(실체)의 세계

1 PhoneBook theBook = new PhoneBook();

클래스(허상)의 세계

```
public class Person {
  private String name;
  private String number;
}
```

```
public class PhoneBook {
  static int INITIAL_CAPACITY = 100;
  public int capacity = INITIAL_CAPACITY;
  public Person [] members;
  public int n = 0;
  public void loadData(String sourceName)  { ... }
  public String addPerson() { ... }
  public String findPerson(String name) { ... }
}
```

```
public class PDConsoleUI {
  private PhoneBook theBook;
  public void processCommands() { ... }
}
```

```
public class MyPDApp {
  public static void main(String args[])  {
    PhoneBook theBook = new PhoneBook();
    theBook.loadData("data.txt");
    PDConsoleUI theConsoleUI =
                    new PDConsoleUI(theBook);
    theConsoleUI.processCommand();
  }
}
```

객체(실체)의 세계

2 theBook.loadData("data.txt");

클래스(허상)의 세계

```
public class Person {
  private String name;
  private String number;
}
```

```
public class PhoneBook {
  static int INITIAL_CAPACITY = 100;
  public int capacity = INITIAL_CAPACITY;
  public Person [] members;
  public int n = 0;
  public void loadData(String sourceName)  { ... }
  public String addPerson() { ... }
  public String findPerson(String name) { ... }
}
```

```
public class PDConsoleUI {
  private PhoneBook theBook;
  public void processCommands() { ... }
}
```

```
public class MyPDApp {
  public static void main(String args[])  {
    PhoneBook theBook = new PhoneBook();
    theBook.loadData("data.txt");
    PDConsoleUI theConsoleUI =
                    new PDConsoleUI(theBook);
    theConsoleUI.processCommand();
  }
}
```

객체(실체)의 세계

3 PDConsoleUI1 theConsoleUI = new PDConsoleUI1();

theBook

```
public int capacity = 100;
public Person [] members;
public int n = 0;

public void loadData(String sourceName)  { ... }
public String addPerson() { ... }
public String findPerson(String name) { ... }
```

theConsoleUI 객체는 이미 생성된 PhoneBook 객체의 주소를 참조변수 theBook 에 가지고 있다.

theConsoleUI

```
private PhoneBook theBook;
public void processCommands() { ... }
```

하나의 ConsoleUI 객체가 생성되고, main 메서드 내에서 그 이름은 theConsoleUI이다.

클래스(허상)의 세계

```
public class Person {
  private String name;
  private String number;
}
```

```
public class PhoneBook {
  static int INITIAL_CAPACITY = 100;
  public int capacity = INITIAL_CAPACITY;
  public Person [] members;
  public int n = 0;
  public void loadData(String sourceName)  { ... }
  public String addPerson() { ... }
  public String findPerson(String name) { ... }
}
```

```
public class PDConsoleUI {
  private PhoneBook theBook;
  public void processCommands() { ... }
}
```

```
public class MyPDApp {
  public static void main(String args[])  {
    PhoneBook theBook = new PhoneBook();
    theBook.loadData("data.txt");
    PDConsoleUI theConsoleUI =
                    new PDConsoleUI(theBook);
    theConsoleUI.processCommand();
  }
}
```

객체(실체)의 세계

4 theConsoleUI.processCommand(theBook);