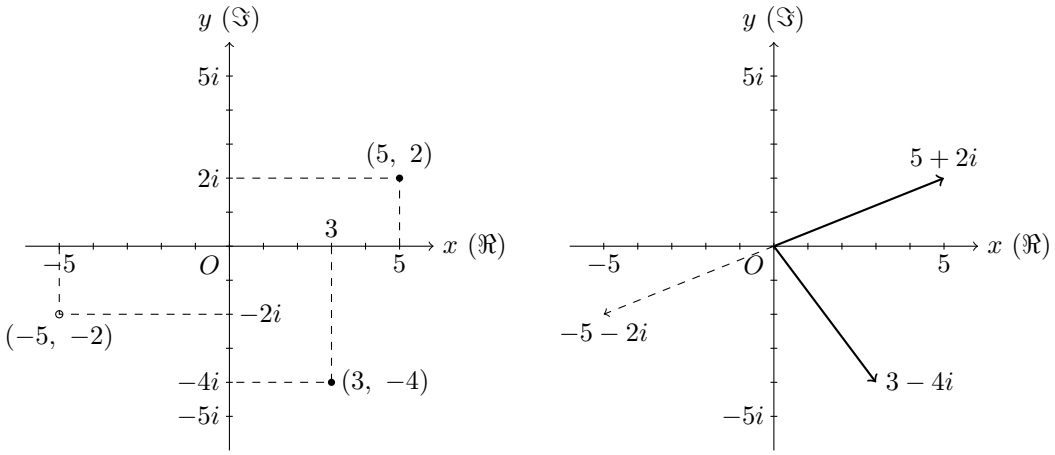
8.2 複素数

実数と虚数はそれぞれの数直線上に表せるが、複素数とは実際どんな数なのだろう。複素数 0 が両数直線に共通していることから、数直線と虚数直線を 0 で交差させた**複素平面**[1)]が作れる。複素数は複素平面上で表される数なのである。複素数 $a+bi$ を、実数単位の 1 と虚数単位の i を強調して書くと $a\cdot 1+b\cdot i$ であるから、複素数はそれぞれの単位の係数である $(a,\ b)$ の組で表してもよい。a を複素数の**実部**、b を複素数の**虚部**という。このことから、複素数は複素平面に自然な形で示すことができる。



実は、複素数は 2 通りの見方ができる。平面上の点として見るか、平面上の**ベクトル**として見るか、である。実数も数直線上で点やベクトルで表せたので似たようなものだ。図は $(3,\ -4)$ と $3-4i$ などを異なる見方で描いたが、$(3,\ -4)$ と $3-4i$ はまったく同じものを表していることを注意しておく。また実数は、数直線上で正反対の数は正負が異なっていたように、複素数も原点 O に対して正反対の点・ベクトルで正負が異なる。

それでは、**Java** で複素数はどのように扱えるだろう。とりあえず `Complex` クラスにそれらしいものを書いてみた。

programming list [Complex'.java]

```
1: public class Complex {
2:
3:     public double re;
4:     public double im;
```

1) ガウス平面とも言う。

```
 5:
 6:     public Complex(double x, double y) {
 7:         re = x;
 8:         im = y;
 9:     }
10:
11:     public static void main(String[] args) {
12:         Complex u = new Complex(3,-4);
13:         Complex v = new Complex(5, 2);
14:         v.re = -5; v.im = -2;
15:
16:         System.out.println(u.re + " + " + u.im + "i");
17:         System.out.println(v.re + " + " + v.im + "i");
18:     }
19: }
```

さすがの **Java** も `int c = 3;` みたいに、`complex c = 3-4i;` と認識してくれるほど気が利いているわけではない。やはり、実部と虚部をそれぞれ宣言しなくてはならないので、3:, 4:行目ではそうしている。ただ、この部分では単に、この `Complex` クラスに実部 `re` と虚部 `im` の**プロパティ**（特性・要素）を与えたに過ぎないので、`re`, `im` に値を設定するためのメソッドが必要となる。それが 6:–9:行目の `Complex()` メソッドである。これを使って複素数 `u`, `v` を初期化したのが 12:, 13:行目である。

この初期化の仕方— `= new` を用いる—は、何度も見ているね。`Scanner s = new ...` だとか `int[] p = new ...` なんかでだ。`int n = ...` なんかとの違いはというと、単に値を代入するだけでなく、プロパティ（要素）やメソッド（機能）まで与えていることである。だから、`s.nextInt()` という整数を取り込む "機能" が使えたり、`p.length` という配列の大きさを表す "要素" を得ることができるのだ。一方、整数 `n` からは値しか得られない。

そこで 12:行目の `Complex u = new Complex(3,-4);` である。これは変数 `u` に `Complex` クラスの要素である `(3, 4)` を与えて初期化しているのだ。`int n` で `int` 型の変数 `n` を宣言したように、`Complex u` で `complex` 型の変数 `u` を宣言し、同時に `Complex()` メソッドを用いて初期化したと思ってもらえばよい。

変数 `v` も同様である。`v` も `Complex` 型の変数なので、`Complex` クラスのプロパティを操作できる。`v` には（もちろん `u` にも）`.re` プロパティと `.im` プロパティがあるので、14:行目で再代入を試みた。16:行目と 17:行目の出力は、それぞれの操作に見合ったものになってるはずだ。

ただし、これだけでは複素数の計算ができない。Complex クラスに、計算のためのメソッドを加えよう。

programming list [Complex.java]

```
1: public class Complex {
2:
3:     public double re;
4:     public double im;
5:
6:     public Complex(double x, double y) {
7:         re = x;
8:         im = y;
9:     }
10:
11:    public static class Operator {
12:        Complex Add(Complex x, Complex y) {
13:            Complex z = new Complex(x.re + y.re, x.im + y.im);
14:            return z;
15:        }
16:
17:        Complex Sub(Complex x, Complex y) {
18:            Complex z = new Complex(x.re - y.re, x.im - y.im);
19:            return z;
20:        }
21:
22:        Complex Mul(Complex x, Complex y) {
23:            Complex z = new Complex(x.re * y.re - x.im * y.im,
                                                    x.re * y.im + x.im * y.re);
24:            return z;
25:        }
26:
27:        Complex Div(Complex x, Complex y) {
28:            Complex z = new Complex(
                       (x.re * y.re + x.im * y.im) / (y.re * y.re + y.im * y.im),
                       (-x.re * y.im + x.im * y.re) / (y.re * y.re + y.im * y.im));
29:            return z;
30:        }
31:    }
32:
33:    public static void main(String[] args) {
34:        Operator op = new Operator();
35:
36:        Complex u = new Complex(3,-4);
37:        Complex v = new Complex(5, 2);
38:
39:        Complex a = op.Add(u, v);
```

```
40:         Complex s = op.Sub(u, v);
41:         Complex m = op.Mul(u, v);
42:         Complex d = op.Div(u, v);
43:
44:         System.out.println(u.re + " + " + u.im + "i");
45:         System.out.println(v.re + " + " + v.im + "i");
46:         System.out.println();
47:         System.out.println(a.re + " + " + a.im + "i");
48:         System.out.println(s.re + " + " + s.im + "i");
49:         System.out.println(m.re + " + " + m.im + "i");
50:         System.out.println(d.re + " + " + d.im + "i");
51:     }
52: }
```

プログラムが妙に長いのは、複素数をいちいち設初期化し、ご丁寧に和・差・積・商を一つずつ計算および表示している `main()` メソッドのせいだ。

計算のためのメソッドは、和・差・積・商用にそれぞれ `Add()`、`Sub()`、`Mul()`、`Div()` とし、それらを `Operator` クラスにまとめてある。メソッドの中身は何のことはない、複素数の計算規則を `.re` と `.im` を用いて書き直しただけである。ちょっと、見づらいのは仕方ないけれど。で、計算ができるようにするためには、34:行目でインスタンス `op` を生成し、`op` の機能として 39:–42:行目のように四則計算を行うのである。

つまり **Java** では、自らクラスを作り、そのクラスでインスタンスの機能を利用できるってことだ。だから、何か特別な操作をしたくなったら、それ用のクラスを自分で作ればよい。いまは複素数の計算がしたいためにクラスを作ってみたのだが、もっとちゃんとした複素数クラスはすでに存在している。要するに無駄な寄り道をしたわけである。しかし豆旅とはいえ、少しぐらいの冒険をしてもかまわないだろう。