## 8.4 文字列操作

前節のやり残しを解決しよう。それは、たとえば複素数 $4-3i$ がガウス素数かどうか判定する際、入力要求 `input:` に対して `input: 4 -3` ではなく `input: 4-3i` とできないだろうかということだ。ちょっと頑張ってみよう。

programming list [GPcheck3.java]

```
 1: import java.util.Scanner;
 2:
 3: public class GPcheck3 {
 4:
 5:     public static void main(String[] args) {
 6:         Scanner s = new Scanner(System.in);
 7:
 8:         System.out.print("input 'a+bi': ");
 9:         String str = s.next();
10:
11:         int re = Integer.parseInt(str.substring(0, SeparatePos(str)));
12:         int im = Integer.parseInt(str.substring(SeparatePos(str),
                                                    str.length()-1));
13:
14:         int n = re * re + im * im;
15:
16:         String msg = "a Gauss prime.";
17:         for(int i = 2; i < n; i++) {
18:             if(n % i == 0) {
19:                 msg = "NOT a Gauss prime.";
20:             }
21:         }
22:         System.out.println(re + " + " + im + "i"  + " is " + msg);
23:     }
24:
25:     static int SeparatePos(String str) {
26:         int p = str.lastIndexOf('+');
27:         if(p == -1) {
28:             p = str.lastIndexOf('-');
29:         }
30:         return p;
31:     }
32: }
```

問題は、入力された文字列を複素数の実部と虚部に分けることである。そこで、文字列を分割する必要があるのだが、**Java** はたとえば文字列 "`abcdef`" や "`-12-34i`" に対してインデックス（位

置）を

| 文字列 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| インデックス | 0 | 1 | 2 | 3 | 4 | 5 |

| 文字複素数 | - | 1 | 2 | - | 3 | 4 | i |
|---|---|---|---|---|---|---|---|
| インデックス | 0 | 1 | 2 | 3 | 4 | 5 | 6 |

のように決めている。

この地で入力する複素数は、実部・虚部ともに整数値を想定しているので、それほど複雑な区切りを考える必要がない。確実なのは、実部と虚部の間には必ず + か - があることである。そこで、その位置を知るためのメソッド `SeparatePos()` を使っている。25:–31:行目にかけてである。

`SeparatePos()` メソッドは文字列を受け取り、その位置を返す。26:行目の`.lastIndex()` メソッドが、指定した文字列の "末尾からの" 位置を返す。`.lastIndex()` メソッドは、目的の文字列が見つからなければ `-1` を返す仕様である。そのため、`+` が見つからなければ再度 `-` を探す。これが 27:–29:行目の処理だ。

`.indexOf()` メソッドを使えば先頭からの位置を返すのだが、これを使わなかったのには理由がある。上の例の複素数 `-12-34i` の場合は、`+` が見つからないので `-` を探すのだが、先頭から探してしまうと `-12` の `-` が区切りと判断されてしまうからだ。

さて、区切り位置が分かれば`.substring()` メソッドを使って文字列を切り出せる。`.substring(i, j)` と書いたら、インデックス `i` から `j-1` までが切り出されることに注意。だから、インデックスは

| 文字列 | a | b | c | d | e | f |
|---|---|---|---|---|---|---|
| インデックス | 0 | 1 | 2 | 3 | 4 | 5 |

のように振られていると思った方がよいかもしれない。

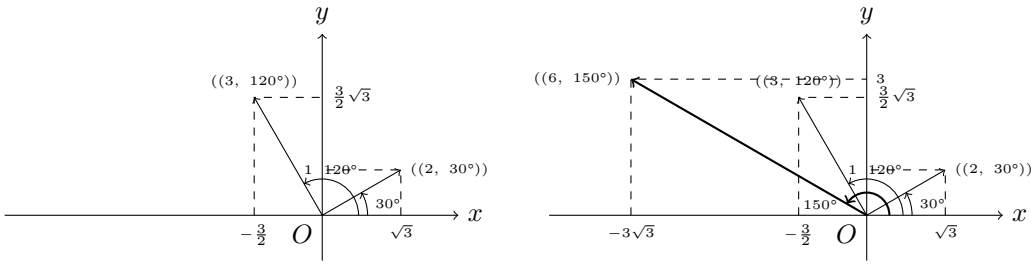
切り出した文字列は整数値にしたいので`.parseInt()` メソッドを用いる。それが 11:, 12:行目である。ちなみに、入力する複素数は末尾が必ず `i` なので、12:行目は末尾の前までが切り出す範囲である。

ただし、プログラムには欠陥がある。それは、`3-i` のように `i` の係数 `1` を省略して書くとエラーを引き起こす。`3-1i` と書けば済むのだが、数学的にはあまりよい眺めではない。

**TRY!** プログラムを `3-i` のような入力も受け付けるように改良してみよ。

さてと。眺めのよくない場所に長居しても仕方ないね。目の覚めるような景色を見に行こう。

複素数の表し方には別の考え方もある。これまで複素数は、ガウス平面の $x$ 座標と $y$ 座標の組で表してきたが、原点 $O$ からの距離 $r$（**大きさ** $r$）と $x$ 軸の正の方向となす角 $\theta$（**偏角** $\theta$）で表すこともできる。たとえば、複素数 $(x_1,\ y_1)=(\sqrt{3},\ 1)$ は複素数 $((r_1,\ \theta_1))=((2,\ 30°))$ と表すことができ、複素数 $(x_2,\ y_2)=\left(-\dfrac{3}{2},\ \dfrac{3}{2}\sqrt{3}\right)$ は複素数 $((r_2,\ \theta_2))=((3,\ 120°))$ と表すことができる。このように複素数を、大きさ $r$ となす角 $\theta$ を用いて表す形式を**極形式**と呼ぶ[1)]。




その上で、複素数の一般形式の積と極形式の積を較べてもらいたい。

$$(\sqrt{3},\ 1)\left(-\frac{3}{2},\ \frac{3}{2}\sqrt{3}\right)=(-3\sqrt{3},\ 3)\quad\Leftrightarrow\quad((2,\ 30°))((3,\ 120°))=((6,\ 150°))$$

**EX.** 上の計算が正しいことを確かめよ。

この豆旅では込み入った理論は述べないが、上の極形式の積が

**大きさは 2 数の大きさ $r_1$, $r_2$ の "積"、偏角は 2 数の偏角 $\theta_1$, $\theta_2$ の "和"**

となっていることに気づくだろうか。これは複素数の積において常に成り立つことである。ということは、同じ複素数の積、すなわち複素数の 2 乗なら

$$\underline{((r,\ \theta))^2}=((r,\ \theta))((r,\ \theta))=((r\cdot r,\ \theta+\theta))=\underline{((r^2,\ 2\theta))}$$

だね。ならば、この式の両端（下線部）だけ見て平方根をとると

$$((r,\ \theta))=\sqrt{((r^2,\ 2\theta))}$$

という関係が見える。この式を「左辺 $\leftarrow$ 右辺」の方向に見ると複素数の平方根は、大きさは平方根倍に（$r\leftarrow r^2$）、偏角は 1/2 倍に（$\theta\leftarrow 2\theta$）なるということだ。だから

1) この豆旅だけの記法だが、極形式の複素数は二重括弧 (( )) を用いて表すことにする。

$$
\begin{array}{ccccc}
i & \Leftrightarrow & (0,\ 1) & \Leftrightarrow & ((1,\ 90°)) \\
\boxed{\downarrow 2\text{ 乗}} & & & & \\
-1 & \Leftrightarrow & (-1,\ 0) & \Leftrightarrow & ((1,\ 180°)) \\
\boxed{\downarrow 2\text{ 乗}} & & & & \\
1 & \Leftrightarrow & (1,\ 0) & \Leftrightarrow & ((1,\ 360°))
\end{array}
$$

という関係がある。(負) × (負) = (正) であることは、複素数の目で見れば至極自然なことなのである。

そして驚くことに、$i$ から逆方向へ進むと ...（次の図は、下から上へ見てほしい)。

$$
\begin{array}{ccccc}
1 & \Leftrightarrow & (1,\ 0) & \Leftrightarrow & ((1,\ 0°)) \\
\boxed{\uparrow \text{平方根}} & & & & \\
\vdots & \Leftrightarrow & \vdots & \Leftrightarrow & \vdots \\
\boxed{\uparrow \text{平方根}} & & & & \\
\sqrt{\frac{\sqrt{2}}{2}+\frac{\sqrt{2}}{2}i} & \Leftrightarrow & (\frac{\sqrt{2+\sqrt{2}}}{2},\ \frac{\sqrt{2-\sqrt{2}}}{2}) & \Leftrightarrow & ((1,\ 22.5°)) \\
\boxed{\uparrow \text{平方根}} & & & & \\
\sqrt{i} & \Leftrightarrow & (\frac{\sqrt{2}}{2},\ \frac{\sqrt{2}}{2}) & \Leftrightarrow & ((1,\ 45°)) \\
\boxed{\uparrow \text{平方根}} & & & & \\
i & \Leftrightarrow & (0,\ 1) & \Leftrightarrow & ((1,\ 90°))
\end{array}
$$

なんと、1 だ！　そして、これは 一番目の図の下につながって（$360° = 0°$ に注意)、$\boxed{\downarrow 2\text{ 乗}}$ は $\boxed{\uparrow \text{平方根}}$ に置き換えられるから、$i$ は一周回って $i$ に戻ってしまった。うーむ。複素数ってなんだか不思議。と言っても豆旅ではほんの入口をのぞいたにすぎない。複素数の頂上はおそらくリーマン[2)]予想

$$\zeta\text{（ゼータ）関数の自明でない零点の実部は } \frac{1}{2} \text{ である}$$

じゃなかろうか。パッと読むだけでは意味がつかみづらいだろう。だったら、頂上なんか目指さなくても、複素数の麓（ふもと）を散策しても見どころ満載である。まずそっちから訪れてみよう。

2)　ゲオルク・フリードリヒ・ベルンハルト・リーマン (1826–1866) : ドイツの数学者。