オブジェクト指向プログラミング　レポート課題

氏名：中村優希

学籍番号：212C1111

1. 設計

複素数を扱う電卓を作成した。

まず、Java には複素数を扱うクラスは無いようだったので Complex クラスを作成した。電卓として利用することを考えて、Matrix クラスのメソッドを参考に与えられた複素数をコピーするコンストラクタや toStrng()、read()メソッドを書いた。

Complex クラスでは以下のメソッドを作成した。

型	メソッド	説明
-	Complex()	引数を取らないコンストラクタ。実部、虚部ともに 0 の複素数を作成する。
-	Complex(double re, double im)	引数に実部と虚部の値をとるコンストラクタ。
void	setReal(double re)	自身の実部に値を設定する。
void	setImag(double im)	自身の虚部に値を設定する。
double	getReal()	自身の実部の値を返す。
double	getImag()	自身の実部の値を返す。
Complex	getConjugate()	自身の共役複素数を返す。
double	getAbs()	自身の複素数の大きさを返す。
double	getArg()	自身の複素数の偏角を返す。
String	toString()	複素数を表す文字列を返す。
Complex	add(Complex comp)	引数の複素数との和を計算して返す。
Complex	sub(Complex comp)	引数の複素数との差を計算して返す。
Complex	mul(Complex comp)	引数の複素数との積を計算して返す。
Complex	div(Complex comp)	引数の複素数との商を計算して返す。
Complex	Read(final List<String> block)	「ブロック」から複素数や実数、虚数を作成して返す。

複素数を扱うが、実数のみまたは虚数のみでも扱えるように、toString()メソッドでは値が 0 である実部または虚部は表示しないようにした。なおどちらも値が 0 の場合、実部のみが表示される。

read()メソッドでは block の 2 行目をトークンにして String の配列に入れ、その配列の長さによって異なる手続きを行う。長さが 1 のとき、入力されたのは実数または虚数のどちらかとなる。虚数は最後に i を加えて入力するようにして実数と虚数の判別をするので、read()メソッド内では i の有無を見て値を決定する。長さが 2 のとき、複素数が入力されたので 1 つ目の値を実部に、2 つ目の値を虚部に設定する。

コマンドは複素数を入力するコマンドと加減乗除を行うコマンド、共役複素数を求めるコマンド、複素数の大きさを比較するコマンドの計 7 個のコマンドを作成した。複素数を入力するコマンドを除く全てのコマンドは CommandWithMemory を継承している。

コマンドは以下のクラスを作成した。

クラス	説明
ComplexValue	「com」のあとに任意の複素数または実数、虚数を入力するとその値が「結果」となる。
ComplexAdd	「add」のあとに加えたい数を入力すると、現在の「結果」に値を足したものが新しい「結果」となる。 「add」の後が変数名でも実行できる。
ComplexSub	「sub」のあとに引きたい数を入力すると、現在の「結果」から値を引いたものが新しい「結果」となる。 「sub」の後が変数名でも実行できる。
ComplexMul	「mul」のあとに掛けたい数を入力すると、現在の「結果」に値を掛けたものが新しい「結果」となる。 「mul」の後が変数名でも実行できる。
ComplexDiv	「div」のあとに割りたい数を入力すると、現在の「結果」を値で割ったものが新しい「結果」となる。 「div」の後が変数名でも実行できる。
CompConju	「conj」で現在の「結果」の共役複素数が新しい「結果」となる。 「conj」の後が変数名のとき、その共役複素数が新しい「結果」となる。
CompCompare	「comp」のあとに比較したい数を入力すると、現在の「結果」と値を比較して大きい方が新しい「結果」となる。 「comp」の後が変数名のとき、現在の「結果」とその変数との比較を行う。 大きさが等しい場合、1.0 という「結果」になる。

2. 動かし方

以下のように動かすことが出来る。

1. javac Calculator.java ComplexCalc.java ComplexCalc.java MemoCalc.java IntCalc.java でコンパイルする。

2. java ComplexCalc で実行すると、初めに初期値の 0.0 とプロンプトが表示される。

```
powwa@DESKTOP-1JGV877:~/OP2022/11$ java ComplexCalc
0.000
>>
```

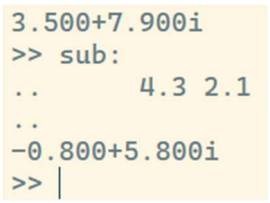
3. コマンド「com: (TAB)1.2(_)3.4」を入力すると、現在の結果が「1.2+3.4i」になる。(_は空白)

```
0.000
>> com:
..      1.2 3.4
..
1.200+3.400i
>>
```

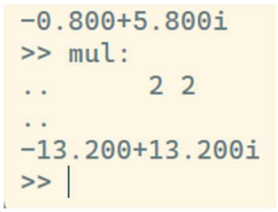
4. コマンド「add: (TAB)2.3(_)4.5」を入力すると、「1.2+3.4i」に「2.3+4.5i」を足した「3.5+7.9i」が結果になる。

```
1.200+3.400i
>> add:
..      2.3 4.5
..
3.500+7.900i
>>
```

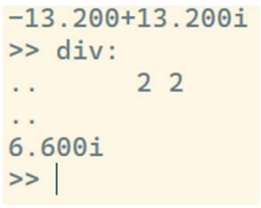
5. コマンド「sub: (TAB)4.3(_)2.1」を入力すると、「3.5+7.9i」から「4.3+2.1i」を引いた「-0.8+5.8i」が結果になる。



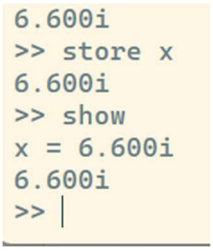
6. コマンド「mul: (TAB)2(_)2」を入力すると、「-0.8+5.8i」に「2.0+2.0i」を掛けた「-13.2+13.2i」が結果になる。



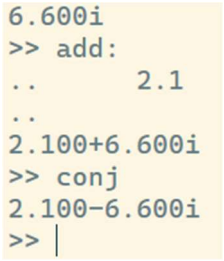
7. コマンド「div: (TAB)2(_)2」を入力すると、「-13.2+13.2i」を「2.0+2.0i」で割った「6.6i」が結果になる。



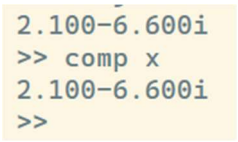
8. コマンド「store(_)x」で現在の結果である「6.6i」が変数 x に保存される。



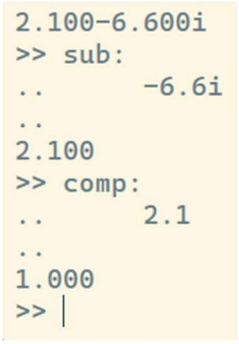
9. コマンド「add: (TAB)2.1」で結果を「2.1+6.6i」にしたのち、コマンド「conj」を入力すると「2.1+6.6i」の共役複素数である「2.1-6.6i」が結果になる。



10.　コマンド「comp x」で現在の結果と保存した変数 x の「6.6i」の大きさを比較し、大きい方である「2.1-6.6i」が結果になる。



11.　コマンド「add: (TAB)6.6i」で結果を「2.1」にしたのち、コマンド「comp: (TAB)2.1」で現在の結果と実数 2.1 と大きさを比較し、等しいため「1.0」が新しい結果となる。



3. 評価

当初は四則演算と大きさの比較ができるコマンドに加え、複素指数関数で何らかの簡単な計算ができるコマンドを実装する予定だった。しかし標準の複素数表記とは別に指数関数表記の結果をうまく出力する方法が考えつかなかったため断念した。

また、コマンドの add、sub、mul、div のクラスは、コマンドの種類を判定する「"add".equal()」や実際に計算する「res.add(v)」の部分以外のコードはほぼ変わらないため共通部分を取ったクラスを作成しようと考えたが、うまくいかなかった。振り返って考えると Method パターンをイメージして統合使用としていたためかもしれない。Generics で考えればうまくできたかもしれないと考える。

実装はほぼ想定通りに行えたが、Calculate.java で入力がどのようにトークナイズされるかについて気付くのが遅かったため、初めは小数の入力が出来ずその対処に長い時間を掛けてしまった点が良くなかったと考える。

計算の速さについて考える。以下のように大きい数を掛け続けてみたが、出力される速さはいつも同じように感じられた。いくら桁が大きくなっても、扱っているのは 2 つの double 型の数であるため計算速度は変わらず早いと考える。

```
powwa@DESKTOP-1JGV877:~/OP2022/11$ java ComplexCalc
0.00
>> com:
..      1234 5678
..
1234.00+5678.00i
>> store x
1234.00+5678.00i
>> mul x
-30716928.00
>> mul x
-37904689152.00+174410717184.00i
>> mul x
-1037078438584320.00+430445650010112.00i
>> mul x
-3723825193970466800.00+6419701306394247200.00i
>> mul x
-41046264307066090000000.00+29065790863454813000000.00i
>> mul x
-215686650677616000000000000.00+268927874661024500000000000.00i
>> mul x
-1793129799261470000000000000000.00+1556525799879208000000000000000.00i
>> mul x
-11050675664002802000000000000000000.00+12102143837257599000000000000000000.00i
>> store y
-11050675664002802000000000000000000.00+12102143837257599000000000000000000.00i
>> mul y
-243444528266883180000000000000000000000000000000000000000000000000000.00
>> mul y
2690226524053488200000000000000000000000000000000000000000000000000000000000000000000000000000000000000.00+29462006974791434000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000.00i
>> mul y
-6538416539433399000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000.00-651150166978017
1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000.00i
>> mul y
151057050337652720000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000.00+1510853504664804000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000.00i
```