Homework 09: Concurrency

Release date: 3/18
Due date: 3/25 by 11:59 PM

Goals

- Learn how to apply synchronization techniques
- Learn how to create a thread-safe library
- Learn how objects can be compared

Introduction

Concurrency and parallelism are often seen as tools that allow parts of a program to execute faster, or even at the same time. Concurrent processes often deals with shared resources, such as fields, whereas parallel processes are independent of each other. The extent of these tools can vary from language to language, but The Java Programming Language has these tools baked in, in the form of threads and synchronization. Both will be covered in lab, but this homework will focus more on synchronization.

One area where synchronization is often used is in the creation of thread-safe APIs and/or methods, such as **`Collections#synchronizedList`**. Clients can use these APIs, and be sure that the code will be free of race conditions, behaving as expected. You will create your own thread-safe class below.

Description

Write a class, **`ComplexNumber`**, that represents a complex number, **`a + bi`**, in Mathematics. The required supertype, fields, constructors, and methods are listed below. Be sure to make each field **`private`**, each constructor **`public`**, and each method **`public`** and **`synchronized`**. A JavaDoc containing detailed descriptions of what is to be declared in the class can be found here.

Supertype

Name	Method to Override
`Comparable<ComplexNumber>`	**`compareTo`**

Fields

Name	Type
`realPart`	`double`
`imaginaryPart`	`double`

Constructors

Parameters
None
`double realPart, double imaginaryPart`
`ComplexNumber aComplexNumber`(*)

Methods

Name	Return Type	Parameters
`getRealPart`	`double`	*None*
`getImaginaryPart`	`double`	*None*
`setRealPart`	`void`	`double realPart`
`setImaginaryPart`	`void`	`double imaginaryPart`
`conjugate`	`ComplexNumber`	*None*
`reciprocal`	`ComplexNumber`	*None*
`add`(*)	`ComplexNumber`	`ComplexNumber aComplexNumber`
`subtract`(*)	`ComplexNumber`	`ComplexNumber aComplexNumber`
`multiply`(*)	`ComplexNumber`	`ComplexNumber aComplexNumber`
`divide`(*)	`ComplexNumber`	`ComplexNumber aComplexNumber`
`compareTo`	`int`	`ComplexNumber aComplexNumber`
`equals`	`boolean`	`Object anObject`

`toString`	**`String`**	*None*

The constructor/methods with (*) in their first column must throw an **`IllegalArgumentException`** if the argument passed is **`null`**. This is mentioned in the JavaDoc.

Sample usage

```
ComplexNumber operandOne = new ComplexNumber(5.0, 5.0);
ComplexNumber operandTwo = new ComplexNumber(operandOne);
ComplexNumber operandThree = new ComplexNumber(5.0, 7.5);

System.out.println(operandOne.add(operandTwo)); //displays 10.000000 + 10.000000i

System.out.println(operandOne.equals(operandTwo)); //displays true

System.out.println(operandOne.compareTo(operandThree)); //displays -1
```

Submission

You are required to declare one class, **`ComplexNumber`**, that follows the specifications outlined above and in the JavaDoc. It is to be held in a file called **`ComplexNumber.java`**.

Submit your file, **`ComplexNumber.java`**, to Vocareum ***through Blackboard***. Keep in mind that only your latest submission will be considered.

Grading Rubric

- **`ComplexNumber`** class — 100 points total
  - 0 points each
    - Field declarations are correct
    - Constructor declarations are correct
    - Method declarations are correct
  - 6.25 points for each constructor and method (16 total)