Homework Assignment #4

CS5004 – Object-Oriented Design
Northeastern University – Silicon Valley
Summer 2020

Due Sunday 06/07 at 11:00pm PT

Grading: Each programming problem is graded as follows

- A submission which does not compile gets 0.
- A submission which compiles but does something completely irrelevant gets 0.
- A submission which works (partially) correctly, gets (up to) %80 of the total credit.
- %20 is reserved for the coding style. Follow the coding style described in the book.

Problem 1 [15pts]. Create a class `Employee` that includes the following instance variables

- First name (`String`) cannot be empty,
- Last name (`String`) cannot be empty,
- Monthly salary (`double`) must be non-negative.

Provide a `set` and a `get` method for each instance variable. The setters must return `boolean` and perform validation. If the monthly salary is not positive, set its value to zero. If a name is empty set it to "[empty]". Provide a constructor that initializes the three instance variables. No default constructor is required. Also, overwrite `equals()` and `toString()`. Write a test class named `EmployeeTest` that demonstrates class `Employee`'s capabilities. Create two different `Employee` objects. Use `println()` to display information about each employee (this will invoke `toString()`). Then compare the two using `equals()` and display the result.
Submission format: You must define two classes. One is `Employee` which must be declared as a non-public class. Two is `EmployeeTest` which is `public` and contains the `main()` method. So, you must submit one file `EmployeeTest.java` containing the above two classes. The tests must be performed in the `main()` method.

Problem 2 [35pts]. Define a class for rational numbers. A rational number is a number that can be represented as the quotient of two integers. For example, $\frac{1}{2}$, $\frac{3}{4}$, $\frac{64}{2}$, and so forth are all rational numbers. Represent rational numbers as two values of type `int`, one for the numerator and one for the denominator. Your class should have two instance variables of type `int`. Call the class `Rational`.
Include a constructor with two arguments that can be used to set the instance variables of an object to any values. Also include a constructor that has only a single parameter of type `int`; call this single parameter `whole` and define the constructor so that the object will be initialized to the rational number $\frac{\texttt{whole}}{1}$. Also include a no-argument constructor that initializes an object to 0 (that is, to $\frac{0}{0}$).[*]
Define methods `Add` addition, `Sub` subtraction, `Mul` multiplication, and `Div` division of objects of your class `Rational`. These methods should use a calling object and a single argument. For example, the `Add` method has a calling object and one argument. So `a.Add(b)` returns the result of adding the rationals `a` and `b`.
You should include a method, `Normalize()`, to normalize the sign of the rational number so that the denominator is positive and the numerator is either positive or negative. For example, after normalization, $\frac{4}{-8}$ would be represented the same as $\frac{-4}{8}$; similarly, $\frac{-1}{-2}$ would be represented the same as $\frac{1}{2}$.
Importantly, define accessor and mutator methods. Also provide methods `equals` and `toString`. These two methods must first perform normalization.
Extra points [10pts]: `Normalize()` performs full normalization. For example $\frac{4}{-8}$ is reduced to $\frac{-1}{2}$. You need to find the GCD (Greatest Common Divisor) of the numerator and the denominator and divide both by it. Use Euclid's recursive algorithm. We recommend that you use a private *helper* method to do the job.
Submission format: A file `RationalTest.java` which contains the non-public `Rational` class and a public test class `RationalTest` containing a `main()` method which instantiates and tests `Rationals`. Your tests should include the following corner cases[†]

- Operations involving zero,
- Operations involving positive and negative numbers,
- Various normalization possibilities.

We will, of course, test your code with our own tests too.

[*]Here $\frac{0}{0}$ is not the undefined division-by-zero operation. It is used to *represent* zero.

[†]The statements regarding tests are a bit vague because identifying the corner cases is a skill that you should develop. In programming interviews, you are usually asked to write tests for you code.