C490 Homework #2

Points:	: 50 points
Due Date:	: July 15th (8:00am)
Submissions:	: Canvas and hardcopy

PART I (20 POINTS)

Define a class for rational numbers. A rational number is a number that can be represented as the quotient of two integers. For example, 1/2, 3/4, 64/2, and so forth are all rational numbers. (By ½ and so forth, we mean the everyday meaning of the fraction, not the integer division this expression would produce in a Java program.) Represent rational numbers as two values of type `int`, one for the numerator and one for 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 variable of an object to any values.
- Also include a constructor that has only a single parameter of type `int`; call this single parameter `wholeNumber` and define the constructor so that the object will be initialized to the rational number `wholeNumber/1`.
- Also include a no-argument constructor that initializes an object to 0 (that is, to 0/1).
- Define methods for addition, subtraction, multiplication, and division of objects of your class `Rational`. These methods should be **static** methods that each has two parameters of type Rational and return a value of type `Rational`. For example, `Rational.add(r1, r2)` will return the result of adding the two rational numbers (two objects of the class `Rational`, `r1` and `r2`).
- Define accessor and mutator methods as well as the methods `equals` and `toString`.

*Hints: Two rational numbers a/b and c/d are equal if a*d equals c*b.*

A test program RationalTest.java has been uploaded to Canvas, which you can use to test your implementation of `Rational` class.

PART II (10 POINTS)

You are interested in keeping track of the team members and competition information for your school's annual entries in computer programming competitions. Each team consists of exactly four team members. Every year your team competes in two competitions. You decide to develop an application that consists of two classes, namely, Team and Competition.

The **Team** class has data members recording the name of the team, the names for each team member, and info on each competition. The **Competition** class contains variables to track the following: the name of the competition, the name of the winning team, the name of the runner-up, and the year of the competition.

Now assume you have implemented the two classes with appropriate constructor, accessor, and mutator methods. In entering data for past competitions, you note that an entry is usually very similar to the previous year's entry. To help with the data entry, create a ***deep copy*** constructor for the Team class.

The complete version of the Competition class and an incomplete implementation of Team class have been uploaded to Canvas. Read the code carefully and make sure you understand it. Now you just need to complete the deep copy constructor for the Team class. A main() method containing code to test the copy constructor is given in the Team class, you just need to make sure that your implementation of the copy constructor works correctly.

PROGRAMMING PROJECT (20 POINTS)

Define a class called `Administrator`, which is a derived class of the class `SalariedEmployee` (this code for this and two other dependent classes, `Employee` and `Date`, have been uploaded to Canvas). You are to supply the following additional instance variables and methods:

- An instance variable of type `String` that contains the administrator's title (such as "`Director`" or "`Vice President`").
- An instance variable of type `String` that contains the administrator's area of responsibility (such as "`Production`", "`Accounting`", or "`Personnel`").
- An instance variable of type `String` that contains the name of this administrator's immediate supervisor.
- Suitable constructors (including a copy constructor), and suitable accessor and mutator methods.
- A method for reading in an administrator's data from the keyboard (name, hiring date, salary, title, area, supervisor). ***Hints:*** *after you read each value, you need then use accessor methods (including those accessor methods from ancestor classes:* `SalariedEmployee` *and* `Employee`*) to set those values for the local object.*
- Override the definitions for the methods `equals` and `toString` so they are appropriate to the class `Administrator`.

- When you implement the above methods, use "`super`" whenever possible to reuse the code in superclass.

A test program AdministratorDemo.java has been uploaded to Canvas, which you can use to test your implementation of `Administrator` class.

WHAT TO SUBMIT:

- Submit your code for all the parts to Canvas (using the "Assignments" function).
- Submit a hard copy of your code and test-run output (or screenshot).
- Make sure that you follow the "Assignment_style-guideline_C490" or you might lose credits.