CISC 181 Spring 2014
Practice Set 3
Assigned: Mar 10
Due: Mar 16 at 11:55PM on Sakai

Practice sets are to be completed individually. You are free to consult other students to help complete the practice sets (see syllabus for collaboration policy). However, keep in mind that each practice set is designed to cover basic material on which you will be quizzed and tested.

This practice set is intended to cover the following major topics:

- Implementing pre-defined patterns common for Objects (equals, toString)
- Overriding methods for polymorphism
- Creating object hierarchies with polymorphism
- Custom Linked Lists

---

1) One way to represent precise numbers (as opposed to floating point numbers) is to store a fixed number of decimal places. Create a class, ExactNumber, that uses two long properties named left and right (representing the portion of the number that is to the left and right of the decimal point respectively). Longs in Java can store just over 18 base10 digits, so our ExactNumber will allow up to 16 exact digits before and after the decimal.

   For example, 3.75 would be represented by new ExactNumber(3, 7500000000000000L). Note the L on the end which tells Java the large number is a long. This translates to:

$$3 + \frac{7500000000000000}{10000000000000000} = 3.75$$

   We will be making the ExactNumber immutable -- none of your methods should mutate the properties of an ExactNumber. Instead, some methods will return a new ExactNumber.

   a. Write the ExactNumber class and override the default Object toString method (have it use String.valueOf(doubleValue())).
   b. Add a method, doubleValue(), that returns a double that is a close approximation of the ExactNumber.
   c. Make a method, compareTo, that takes as a parameter another ExactNumber and returns -1 if this ExactNumber is less than the parameter, 0 if they are equal, and 1 if this ExactNumber is greater than the parameter.
   d. Override the default Object equals method (the parameter is another Object). Check to see if the parameter is an instanceof ExactNumber, and if it is use your compareTo method to return whether this ExactNumber is equal in value to the parameter (or false if the parameter is not an ExactNumber).
   e. Make a method, add, that takes as a parameter another ExactNumber and returns a new ExactNumber that represents the sum of the values of this ExactNumber and other ExactNumber. Do not mutate this ExactNumber or the parameter.
   f. Your class should pass all of the included JUnit tests.

2) In lecture we created a custom linked list structure with a strand of lights. Complete each of the four methods in a similar custom linked list found in Pipeline.java using structural recursion (call the same method on a different instance of the same type, in this case the "next" Pipeline). Do not write loops to solve these methods. Your code must pass the included tests.

3) One of the most common uses for abstraction is to take two existing classes, find what is similar between them, and create a superclass for both of them that includes the similarities. However, any differences between the class behaviors should be marked as abstract and the subclasses should have specific code for their own behavior in the subclass.

   You have been given two classes, Dog and Cat. Read through the given test and code to figure out what the classes are doing.

   You must create an abstract superclass named Pet that both Dog and Cat will extend. A Pet should have everything (properties and methods) that are common between Dogs and Cats. Anything that is different must be placed in the Dog or Cat class. Use abstract methods to refactor the move method so that most of its code is in the Pet class. Your modified code must still pass the same test. A fully correct submission for this problem will leave only the necessary code in Dog and Cat. Do not use instanceof or a similar type check in Pet, the code must be polymorphic.

Submit your PS3 project to Sakai by exporting it from Eclipse as an archive.