Programming Project 3

Due Saturday, April 6 at 11:59pm

IMPORTANT: *Read the* ***Do's and Dont's*** *in the* ***Course Honor Policy*** *found on blackboard.*

I. Overview

The purpose of this project is to give you practice designing a class/type hierarchy. It is *important* that you spend time designing your class hierarchy *before* you start coding. If you properly organize your classes and/or interfaces, you can achieve the desired program behavior below with significantly less code than with a poorly organized hierarchy.

II. Code Readability (20% of your project grade)

New for this assignment: The comments above the class or interface and above each method must be written in JavaDoc format. You will be introduced to JavaDoc style commenting in the labs. You can also find a description in the *Java in a Nutshell* text. Be sure to run JavaDoc and view the webpage in order to verify that you have implemented the comments correctly.

To receive the full readability marks, your code must follow the following guideline:

- All variables (fields, parameters, local variables) must be given appropriate and descriptive names.
- All variable and method names must start with a lowercase letter. All class and interface names must start with an uppercase letter.
- The class body should be organized so that all the fields are at the top of the file, the constructors are next, the non-static methods next, and the static methods at the bottom.
- There should not be two statements on the same line.
- All code must be properly indented (see page 644 of the Lewis book for an example of good style). The amount of indentation is up to you, but it should be at least 2 spaces, and it must be used consistently throughout the code.
- You must be consistent in your use of {, }. The closing } must be on its own line and indented the same amount as the line containing the opening {.
- There must be an empty line between each method.
- There must be a space separating each operator from its operands as well as a space after each comma.
- There must be a comment at the top of the file that **is in proper JavaDoc format** and includes both your name and a description of what the class represents. The comment should include tags for the author.

- There must be a comment directly above each method (including constructors) that **is in proper JavaDoc format** and states *what* task the method is doing, not how it is doing it. The comment should include tags for any parameters, return values and exceptions, and the tags should include appropriate comments that indicate the purpose of the inputs, the value returned, and the meaning of the exceptions.
- There must be a comment directly above each field that, in one line, states what the field is storing.
- There must be a comment either above or to the right of each non-field variable indicating what the variable is storing. Any comments placed to the right should be aligned so they start on the same column.
- There must be a comment above each loop that indicates the purpose of the loop. Ideally, the comment would consist of any preconditions (if they exist) and the subgoal for the loop iteration.
- Any code that is complicated should have a short comment either above it or aligned to the right that explains the logic of the code.

III. Program Testing (20% of your project grade)

tcs94_CSDS132_Testing3

New for this assignment: The testing routines should be included in a JUnit test class.

You are to write a test report that indicates the *types* of tests needed to thoroughly test your project. The tests should demonstrate that all of your methods behave correctly. An conditional statements will need tests that go through each branch of the execution. Any loops will need tests that cover the "test 0, test 1, test many" and "test first, test middle, test last" guidelines. Your testing report should *not* list the actual tests and results.

You are to have a JUnit test class or classes that implement each of the needed tests. Comments and method names in your JUnit class should connect to your testing report. For example, if your testing report states "*method xxx must be tested with string inputs of different lengths*", then the reader should be able to go to the JUnit class and easily identify the tests that test that method on inputs of length 0, 1, and more than 1.

The testing report must be separate from the JUnit class. In most companies, the testing document will be written in a style that allows both programmers and non-programmers to read it and recognize whether all the needed test cases were included.

IV. Java Programming (55% of your grade)

You will program a function calculator. Each function can be a function of one variable (ex: *3x^2 - 6*) or zero variables (ex: *57^4 - 10*). The variable will always be "x". You can use the calculator to give the value of the function at any point or to compute the derivative of the function. You may even find this code useful with your calculus homework!

Design Rules: Your project must contain the following eight types and each type must contain the listed methods. The project *may* use any combination of classes, abstract classes, or interfaces that you feel is appropriate. The project *may* add additional types to the ones listed. The classes/types *may* contain additional methods to the ones listed if you feel they are needed. You may use any combination of inheritance, method overriding, and method overloading to achieve the needed behavior. Part of the coding grade will be the quality of the hierarchy you create.

The eight types your project *must* contain represent different kinds of functions and mathematical objects. The types are Variable, Number, BinaryOp, Polynomial, Log, Exp, Sin, and Cos.

Each of these types/functions needs to have the following methods:

1. value: returns a double and either takes a single double as input or takes no input. The method returns the value of the function at the given input. If value is called with no input, and input is expected, the method should throw a UnsupportedOperationException. For example, calling value(10) on the function *x^2-3x* is 70.0. Calling value() on the function *x^2-3x* should throw the UnsupportedOperationException. Calling either value() or value(10) on the function *45 + 6 - 10* should return 41.0
2. derivative: takes no input and returns the function that is the derivative of this function.

The following are specific properties for each of the function types that you need to create:

1. Variable: represents the variable *x*. The constructor should take no input. The toString method should just give *"x"* and the equals method should return true (if this object is compared to another Variable).
2. Number: represents a number as a double. The constructor should take a single double value. The toString method should give a String representation of the number, and the equals should compare the number values.
3. BinaryOp: represents a binary operator (+, -, , /) *and two function operands. The constructor should take three values: an enum that represents the operator and the left and right operands. (See below for a description of the enum type.) The BinaryOp type should have the getter methods getOperator, getLeftOperand. and getRightOperand. The equals method should return true if both BinaryOp instances have the same operator and both operands are equal.*
   The toString representation should be left-operand op right-operand where op is one of +, -, , /. (Note the single space between each operand and the operator.) The left operand should be placed inside parentheses if it is a BinaryOp. The right-operand should be places in parentheses if it is a BinaryOp that has a different operator.
4. Polynomial: represents a function raised to a power. The constructor should take two values: a function that is the operand and a double that is the power. The Polynomial type should have a getPower and getOperand methods. For the toString method, the string should use the ^ character. For example, x^5. If the operand is a BinaryOp, the string representation should place it inside parentheses. . Two polynomials are equal if their powers and their operands are equal.

5. Log: Represents the natural logarithm function. The constructor should take a single value, the function that is the operand of the logarithm function, and the type should have a getOperand method. The string representation should be *"Log[operand]"*. Two Log functions are equal if their parameters are equal.
6. Exp: Represents the natural exponential function. The constructor should take a single value, the function that is the operand of the exponential function, and the type should have a getOperand method. The string representation should be *"Exp[operand]"*. Two Exp functions are equal if their parameters are equal.
7. Sin: Represents the sine function. The constructor should take a single value, the function that is the operand of the sine function, and the type should have a getOperand method. The string representation should be *"Sin[operand]"*. Two Sin functions are equal if their parameters are equal.
8. Cos: Represents the cosine function. The constructor should take a single value, the function that is the operand of the cosine function, and the type should have a getOperand method. The string representation should be *"Cos[operand]"*. Two Cos functions are equal if their parameters are equal.

A Short Explanation of Enum

You will use the enum type in this project.

An *enum* is a shortcut for a class with a private constructor. The code

```
enum WeekDay {
Monday, Tuesday, Wednesday, Thursday, Friday;
you may add additional methods here
}
```

is identical to

```
public static class WeekDay {
public static final WeekDay Monday = new WeekDay();
public static final WeekDay Tuesday = new WeekDay();
public static final WeekDay Wednesday = new WeekDay();
public static final WeekDay Thursday = new WeekDay();
public static final WeekDay Friday = new WeekDay();

private WeekDay() {
}

some special helper methods provided for enums (see your text)

you may add additional methods here
}
```

So, WeekDay will be a static "inner" or "nested" class of whatever class you place the enum inside.
The Monday, Tuesday, etc. are fields set to instances of the WeekDay class. Because the constructor is private,

no other instances can be created than one instance for each of the listed fields. (Note that we do not need to override the equals method in an enum. Since no other instances can be created than those stored in the fields, you can use == to compare enum values.) As entered, you have a default private constructor for the enum, but you can create your own constructor if you wish. For example, we could create a constructor that takes a String that is the name of the day. If we do that, we would need to do the following:

```
enum WeekDay {
Monday("Monday"), Tuesday("Tuesday"), Wednesday("Wednesday"), Thursday("Thursday"),
Friday("Friday");

private String name;

private WeekDay(String name) {
this.name = name;
}

you may add additional methods here
}
```

For this project, you should create an *enum* called Op that contains four fields PLUS, SUB, MULT, and DIV to represent the +, -, *, and / operators. The Op enum should be a nested type inside of the BinaryOp type.