BSM 262: Object Oriented Programming-II
Programming Project 2

Due date: 11:59PM, Wednesday, May 17

1 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.

Please submit your assignment to the ubs as a single zipfile containing: only .java files and your testing report.

2 Code Readability (20% of your project grade)

The comments above the class and above each method must be written in **JavaDoc** format. You are already introduced to JavaDoc style commenting in the lectures. You can also find a description in the Java in a Nutshell textbook.

JavaDoc is a tool used to generate documentation from Java source code. The JavaDoc tool reads specially formatted comments in Java source code and generates HTML pages that document the classes, interfaces, methods, and fields in the source code. Here are a couple of examples:

Example 1: Class and a method

```
/**
* This is a sample class that demonstrates how to use JavaDoc.
* @Author:  John DOE
*/
public class SampleClass {
     /**
     * This is a sample method that adds two numbers together.
     * @param a the first number to add
     * @param b the second number to add
     * @return the sum of a and b
     */
     public int add(int a, int b) {
```

Example 2: Method with return type and inputs

```
/**
* Return true of the given date/time is daylight savings.
* Daylight savings time begins 2am the second Sunday of March and ends 2am the first
Sunday of November.
*
```

```
* @param month - represents the month with 1 = January, 12 = December
* @param date - represents the day of the month, between 1 and 31
* @param day - represents the day of the week with 1 = Sunday, 7 = Saturday
* @param hour - represents the hour of the day with 0 = midnight, 12 = noon
* @return true/false - if the given date is daylight savings
*
* Precondition:  the month is between 1 and 12, the date is between 1 and 31, the day
is between 1 and 7 and the hour is between 0 and 23.
*/
public static boolean isDayLightSavings(int month, int date, int day, int hour) {
```

Example 3: Method with no return type and no inputs

```
/**
* Displays a welcome message on the console.
*/
public void displayWelcomeMessage() {
```

Example 4: Method with a return type and no inputs

```
/**
* Generates a random number between 1 and 10 (inclusive).
*
* @return the random number generated
*/
public static int generateRandomNumber() {
```

Example 5: Constructor

```
/**
* Constructs a new Employee object with the given name and salary.
*
* @param name the name of the employee
* @param salary the salary of the employee
*/
public Employee(String name, double salary) {
```

To receive the full readability scores, 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 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, and then the rest of the methods.
• Every statement of the program should be on it's own line and not sharing a line with another statement.
• All code must be properly indented. 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 method that, in one or two lines, states what task the method is doing, not how it is doing it. **Do not directly copy the homework instructions.**
• 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.

Here is the readability rubric: **General Program Structure**

Points	Metric
10	Excellent, readable code. Correct comments at top of class file, above each method and above each field. Each comment at most one or two sentences and is a good description. Class has fields first, then constructors, then the remaining methods. All code follows the proper indentation and naming style.
9	Very readable code. There are only a few places (at most 5) where there is inconsistent or missing indentation or poorly named fields/methods, or missing comments, or comments that are not good descriptions, or fields or constructors buried inside the code or missing whitespaces.
7	Reasonably readable code. Overall good commenting, naming, and indentation. However, there are more than 5 places with missing or inconsistent indentation, poor variable names, or missing, unhelpful, or very long comments OR many comments that are a direct copy of the homework description.
6	Poorly readable code. A significant number of missing comments or inconsistent and/or missing indentation or a large number of poorly named variables. However, at least half of the places that need comments have them.
4	Some commenting done (close to half), but missing majority of the comments.
3	There is at least five useful comment in the code OR there is some reasonable indentation of the code.
0	Code very unreadable. No useful comments in the code AND/OR no indentation in the code.

JavaDoc Comments:

Points	Metric
10	All comments above the class and methods follow the JavaDoc comment style. Correct tags are added for the author, all method parameters and return types and any explicitly thrown exceptions.
9	All comments above the class and methods follow the JavaDoc comment style, and most of the comments contain and have correct all required tags. (Possibly a few misspelled tags.)
7	Most comments above the class, fields, and methods follow the JavaDoc comment style, but a significant number of places are missing tags or the tags are misspelled so the comments are not formatted properly on the webpage.
5	Most comments in JavaDoc style of /** but missing most tags OR has most tags but the comments do not start with /** and so do not appear on the webpage.
0	No JavaDoc comments provided.

3 Program Testing Document (20% of your project grade)

The current softwares are very complex and one minor error might even cause a death. The software glitches starting to become a major issue and destroying companies. Now, standard practice is that all code must be thoroughly verified before a company is willing to release it. Even some companies are

requiring the programmers to write the test cases before even start programming. In this class, we will not be that strict, but you will need to test your code.

To receive full testing marks, you must write a testing report that shows that you thoroughly tested every method of the program. The report should be a short English or Turkish description for each test (what you are testing and what the expected result of the test is) followed by the actual result of the test. **If you are using DrJava, you can enter the test into the interactions pane and then copy and paste the test code plus the result to your report.** If you fail to complete the program (this will not result in point deduction on your testing or readability), your report should indicate how you would go about testing the incomplete methods.

Your grade on the testing report is how thoroughly you test your code, not how correctly your code runs. If your code is not 100% correct then your report should show an incorrect result to some test. Testing methods that do not have conditional statements should be pretty straightforward, but you need to put thought into testing methods with conditional statements so that each branch of the if-statement is tested.

Hint 1: You can test multiple methods with one test. For example, you can test each setter/getter method pair together or you can test constructors and getter methods together.
Hint 2: Do not put off testing to the end! est each method after you complete it. Many methods depend on other methods. Delaying testing could mean cascading errors that cause your whole project to collapse. Since you need to test anyway, copy the tests you do into a document, and you are most of the way to completing your report.

If you are not using DrJava, you are allowed (but not required) create a separate class that tests your program. You must still write a testing report that documents the tests you do in this class. Do not place testing code into a main method of the classes below. That is not the purpose of a main method.
Here is the testing rubric:

Points	Metric
20	Excellent demonstration of how to test the program. Completed tests for each working method of the student's submission, either on its own or as part of a related "unit" of the program. Good descriptions of how to test the parts of the submission that are not working. Tests of conditional statements cover the different possible execution paths. Each test gives the true output of the student's submission. A note on any test that gives an incorrect result.
18	Very good: Good descriptions, tests covers most of the program but a few tests were missed.
15	Reasonable demonstration of how to test the program. The tests performed and the testing description covers a majority of the program; however, multiple obvious cases were missed.
12	Poor demonstration of how to test the program. The testing report shows how to test some of the methods and conditional statements, but at least half of the needed tests are missing.
10	Some testing done (at least half), but nothing that demonstrates the proper way to test a conditional statement.
5	There is at least five proper testing of the methods/constructors in the code.
0	The testing report does not match the behavior of the student's code.

4 Programming (60% of your grade)

Guidelines for the programming part:

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.

Design Rules: Your project must contain the following five 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 five types your project must contain represent different kinds of functions and mathematical objects. The types are **Variable, Number, BinaryOp, Polynomial, and Log**.

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

• **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.

• **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, 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 placed 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 "**Exp[operand]**". Two Exp 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.

Programming part of the assignment will have two main parts including Good class hierarchy and performance. Here is the programming rubric:

Good class hierarchy and polymorphism: 30 points

Points	Metric
30	An excellent hierarchy and use of polymporphism. The class heirarchy is well organized and intuitive (every part of the hierarchy fits an "is-a" relation and all "is-a" relations in the project are included in the hierarchy), common methods are moved as high up the hierarchy as appropriate, and polymorphism is used to simplify the code everywhere that is appropriate.
26	A very good hierarchy and use of polymorphsim. The class heirarchy is well organized and intuitive. In every place that a class extends another class, the "is-a" relation makes sense, but there are one or two places where an "is-a" relation is not represented in the hierarchy. There is a lot of use of polymorphism to simplify the code. Most methods are moved as high up the hierarchy as appropriate.
23	A reasonable hierarchy and use of polymorphism. The class heirarchy has more correct "is-a" relations than incorrect ones. There are multiple places where methods are moved up the heirarchy and inherited, and there are multiple places where polymorphism is used. However, there are also several places where code is duplicated between classes or methods are unnecessarily overloaded or unnecessarily complicated.
15	A hierarchy is created and there is at least one use of polymorphism, but there are a significant number of "is-a" relations that are incorrect -OR- for the most part methods are unncessarily overloaded or duplicated between classes.
10	Some attempt at creating a hierarchy but no polymorphism is used.
0	No class hierarchy is created, and thus no polymorphism is used.

Performance correctness: 30 points

Points	Metric
30	All classes/methods are done and correct.
26	Most of the classes/methods are done and correct. Only minor errors are found.
23	Most of the classes/methods are done and correct, but possibly significant errors in a few of the methods.
18	At least half of the classes/methods are not correct, but all have reasonable code to solve the problems.
13	At least a few of the required classes have reasonable code for all of their require methods.
5	At least one of the required non-getter/setter methods has reasonable code.
0	Nothing works except getter/setter methods.

Penalties

Deduction	Cause
-3	The use of break or continue in loops (or any break-like loop organization). break is allowed in a switch statement.
-3	Unnecessarily long if statements or unnecessarily long methods.
-3	Warning messages when compiled.
-10	Using @SuppressWarnings to hide compiler warning messages.
-10	Submitting code that fails to compile.

5 Course Honor Policy

See the university policy on academic integrity for general rules that you should follow on tests and quizzes. Rules specific to programming in this class are listed here.

Programming is a collaborative enterprise. However, you cannot be an equal collaborator until you first build up your own programming skill set. Because of that, it is essential that you do programming project coding on your own. That does not mean that you can not seek help or give help to other students, but the assistance must be at a high level - for example English descriptions of how to solve a problem and not coding descriptions. The most important skill you will develop in this class is to translate a solution description into the Java commands needed to implement the solution. If you copy code from other sources, you are not developing this needed skill for yourself.

Here is a short list of things you may and may not do:
• **DO NOT** send an electronic copy of your code to another student. (Exception: You may share with your lab partner a copy of the code you wrote together during the recitation section.)

• **DO NOT** look at another person's code for the purpose of writing your own.

• **DO NOT** tell another student the Java code needed for an assignment.

• **DO** sit down with a student and go over the student's code together in order to help the student find a bug.

• **DO** describe in English (not code!) the steps needed to solve an assignment problem.

• **DO** show students how to do certain Java coding tasks as long as the task does not directly relate to a homework question.

• **DO** use coding examples from the lecture and textbook as a guide in your programming.

• **DO NOT** search the internet to find the exact solutions to assignment problems.

• **DO NOT** post code you write for this class onto an internet site such as StackExchange.

• **DO NOT** post the homework specific problem descriptions onto internet sites such as StackExchange.

• **DO NOT** use any online artificial intelligence tools.