Assignment 2 Comp 1510

Section 1
Due Sunday, November 13, 2022, 8:59 PM in the learning hub.

Read the entire assignment carefully, especially the sections about assignment preparation and Plagiarism / Collaboration.

This is an individual assignment, no partners allowed and no sharing of any code whatsoever.

There are **four** programming projects, each is worth 10 marks. The projects can and should be solved using the material through chapter 6 of the text.

Project 1: (package q1) Cylinder.

Design and implement a class `Cylinder` that contains instance data that represents the cylinder's radius and height. Define a `Cylinder` constructor to accept and initialize the instance variables for radius and height. Include getter and setter methods for all instance variables. Include methods that calculate and return the volume and surface area of the cylinder

> Surface area: $2\pi r(r + h)$
>
> Volume: $\pi r^2 h$

Include a `toString` method that returns a two-line description of the cylinder (radius and height on separate lines). Create a driver class, `MultiCylinder`, that instantiates and updates two `Cylinder` objects (no user input required), printing their radius and height before and after modification, and prints the final volume and surface area of each cylinder.

Project 2: (package q2) Word Counter.

Project Gutenberg has been providing free copies of books on the Internet since at least 1989. For your next exercise, we would like you to tell us how many distinct words there are in the enclosed text file which is a plaintext copy of the King James Bible, available from Project Gutenberg. The file is provided in the src directory (assume this is the working directory). If you run your program with eclipse, you will need to set the working directory to this src directory.

Please write a program that meets the following requirements:

1. Create a Word class that contains two instance variables, a String for the word and an <u>int</u> representing its frequency.

2. Word must contain a constructor that accepts one parameter, a String (which is the word). The constructor should set the frequency to one.
3. Make the String in Word final. Its value will be assigned in the constructor and never changed. Provide an <u>accessor</u> for the String. Provide an <u>accessor</u> for the frequency. Also implement a method that increments the frequency by 1.
4. Word should have a toString method that returns the word and its frequency.
5. Word should also implement the Comparable<Word> interface. By implementing the interface, Word must contain a method called compareTo which accepts another Word as the parameter and returns an <u>int</u>. When comparing Words, the word with the higher frequency should come first.
6. Create a WordCounter class with a List of Word as an instance variable. Initialize this with an ArrayList.
7. WordCounter should contain a method <u>called</u> parseBook which accepts a String as a parameter which represents the name of the file to open. The parseBook method will open the file using a Scanner and use a while loop to acquire each word. Your program should check if the word is already in the list. If it is, its frequency should be incremented. If the word is not in the word list, then it should be added. When parseBook finishes, it should return the total number of unique Words stored in the list.
8. WordCounter should have a method called printTopWords which accepts an <u>int</u> n and then prints the n most frequent words in order (most frequent ones first). Make sure you sort the <u>List</u> first. Note that calling the List sort method with a null parameter will sort the list according to Word's compareTo method.

9. You will need a main method to drive the program. Put it at the bottom of the WordCounter class (inside the class, but after all the other methods). Inside the main method, instantiate a WordCounter. Invoke the Word Counter object's parseBook method passing it the name of a file, and then print the result (which will be the number of words in the text). Finally, print out the top 10 most frequent words.

Note that you may need to define more methods to decompose the ones mentioned above in case they are too large.

Project 3: (package q3) Primes

Eratosthenes (Air-uh-TOSS-the-knees) was a Greek mathematician who developed an efficient method for identifying prime numbers. A prime number is a number greater than 1 that is only divisible by 1 and itself, i.e. 7 is prime because it is only divisible by 1 and 7, but 8 is not prime because it is divisible by 1, 2, 4, and 8.

His method, called the Sieve of Eratosthenes, works like this:

1. Make a list of numbers: 0, 1, 2, 3, ..., N

2. 0 is not prime, so cross it out

3. 1 is not prime, so cross it out

4. 2 is prime, but its multiples are not prime, so cross out 4, 6, 8, 10, ...

5. 3 is prime, but its multiples are not prime, so cross out 6, 9, 12, 15, ...

6. (4 is crossed out so next is 5) 5 is prime, but its multiples are not primes, so cross out 10, 15, 20, 25, ...

7. Continue until you have reached √N (can you explain why it is sufficient to stop here?)

What has not been crossed out is a list of prime numbers!

Create a class called Primes. Primes should contain the following:

1. a Boolean List called primes (initialized with an ArrayList).

2. a constructor that accepts a single integer, N, which is used to initialize the primes ArrayList to have maximum index N.

3. the constructor should pass N to a private method called calculatePrimes, which uses the Sieve of Eratosthenes to determine which indices are prime (true) and which are not (false)

4. a method called printPrimes() which prints out the list of primes you calculated

5. a method called countPrimes() which returns an int representing the number of primes in the range [0, N]

6. a method called isPrime(int x) which returns true if x is prime and false if it is not. The parameter must be in the range [0, N]

7. a main method which:

  a) uses a Scanner object to ask the user for an upper bound, N

  b) uses the input value as the parameter for the Primes constructor to create a new Primes object

  c) invokes countPrimes() and prints the result

  d) invokes printPrimes() to print a list of prime numbers.

Here is how your program should work:

```
This program uses the sieve of Eratosthenes to determine
which numbers are prime.

Enter an upper bound:

30

There are 10 primes:

The prime numbers between 0 and 30 are:

2 3 5 7 11 13 17 19 23 29
```

The program should validate that the upper bound is a number greater than one.

Project 4: (package q4) Exponential.

The usual definition of the exponential function, e^x, is an infinite sum:

$$\exp x := \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} + \cdots$$

Fortunately, the terms get smaller very rapidly, so we can calculate the answer quite rapidly with a loop. To calculate the next term x_{k+1} from the previous term x_k, we can multiply $x_{k+1} = x_k * (x / (k+1))$.

Write a class, Exponential, which defines a static method, `exp`, that has a parameter and return type of `double`. Write a main method that evaluates and prints a table of 21 exponentials for x = -10.0 .. 10.0 with one value per row, showing x and e^x.

Each program is required to conform to the style guidelines in **Section 3**. Each class must have Javadoc comments, as in **Section 3**. For this assignment, the Javadoc comments are required for classes, methods and variables but see **Section 3** for details.

For this assignment, programs may consist of multiple classes, with the names as given above. The classes for project 1 should be in package q1, project 2 in package q2, etc. The assignment template provides an ant script to compile, run, and package your code for submission.

Section 2

Assignment Preparation

In the corresponding assignment template zip file, there is a starting project layout for building and packaging your assignment. You will need to **fill in your name** in the build.xml file. You then need to fill in the code for each of the classes (we give you a template for the main class, you may need to write the others) and run the ant script to compile / package your assignment.

In developing each programming project, you will follow the program development steps as discussed in the text book. Specifically, when the problem requires it, you need to perform all of the following steps:

1. establishing the requirements,
2. creating a design,
3. implementing the code, and
4. testing the implementation.

Assignments are to be submitted in *soft copy* (in the Learning Hub). See below for details on the requirements for your submission.

Section 3

Comments and documentation

Comment Item: All Classes
Details:
Do not create comments for the sake of creating comments. Focus is on quality not on quantity. Comments should be succinct and to the point. If you can be brief, then do so. Please use English that is grammatically correct.

Each class must have a Javadoc header block comment immediately before the class statement. This must have the following format:

```
/**
 * Introductory summary sentence describing the class.
 * More complete description of everything the class is supposed
 * to do(may be several lines long.
 *
 * @author name of author of the code and set
 * @version version number, such as 1.0
*/
```

Each paragraph should be separated from the next by a <p> tag. If you need to use lists or tables in the description, you should use HTML tags.

Example header block:

```
/**
 * Graphics is the abstract base class for all graphics contexts
 * which allow an application to draw onto components realized on
 * various devices or onto off-screen images.
 * A Graphics object encapsulates the state information needed
 * for the various rendering operations that Java supports.  This
 * state information includes:
 * <ul>
 * <li>The Component to draw on
 * <li>A translation origin for rendering and clipping coordinates
 * <li>The current clip
 * <li>The current color
 * <li>The current font
 * <li>The current logical pixel operation function (XOR or Paint)
 * <li>The current XOR alternation color
 *     (see <a href="#setXORMode">setXORMode</a>)
 * </ul>
 * <p>
 * Coordinates are infinitely thin and lie between the pixels of the
 * output device.
 * Operations which draw the outline of a figure operate by traversing
 * along the infinitely thin path with a pixel-sized pen that hangs
 * down and to the right of the anchor point on the path.
 * Operations which fill a figure operate by filling the interior
 * of the infinitely thin path.
 * Operations which render horizontal text render the ascending
 * portion of the characters entirely above the baseline coordinate.
 * <p>
 * Some important points to consider are that drawing a figure that
 * covers a given rectangle will occupy one extra row of pixels on
 * the right and bottom edges compared to filling a figure that is
 * bounded by that same rectangle.
 * Also, drawing a horizontal line along the same y coordinate as
 * the baseline of a line of text will draw the line entirely below
 * the text except for any descenders.
 * Both of these properties are due to the pen hanging down and to
 * the right from the path that it traverses.
 * <p>
 * All coordinates which appear as arguments to the methods of this
 * Graphics object are considered relative to the translation origin
 * of this Graphics object prior to the invocation of the method.
 * All rendering operations modify only pixels which lie within the
 * area bounded by both the current clip of the graphics context
 * and the extents of the Component used to create the Graphics object.
 *
 * @author      Sami Shaio, Set K
 * @author      Arthur van Hoff, Set J
 * @version     %I%, %G%
 */
```

What Javadoc tags to use:

```
@author
@link (where applicable) - for example when you want to link to superclasses
@literal (where applicable)
@version
<b> ... </b>
<code> ... </code>
<i> ... </i>
<p> ... </p>
<ul> <li> ... </li> ... </ul>
```

Comment Item: Class Variable, Instance Variable (including private, protected, public & default)

What Javadoc tags to use:
```
@literal (where applicable)
<b> ... </b>
<code> ... </code>
<i> ... </i>
<p> ... </p>
<ul> <li> ... </li> ... </ul>
```
Details:

The only variables that you will not comment using Javadoc comments are local variables. Local variables can be commented by using the slash-slash style of commenting.

Comment Item: Method (including private, protected, public & default)

What Javadoc tags to use:
```
@literal (where applicable)
@param (where applicable) - when you have parameters passed to a method.
@return (where applicable) - when you have a return value.
@see (where applicable)
@throws (where applicable) - when you have a method that throws an exception -
Any exception!
<b> ... </b>
<code> ... </code>
<i> ... </i>
<p> ... </p>
<ul> <li> ... </li> ... </ul>
```

Details:

Don't forget to comment all methods including constructors.

Lastly:

Tutorial on javadoc: http://java.sun.com/j2se/javadoc/writingdoccomments/

As well, for coding style in Java you are required to conform to the document: http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html. Note that some of the style items in this document refer to things not yet covered. You can safely ignore these until they become applicable after we have covered the corresponding constructs (loops, for example).

Quick checklist for your code conventions:

- proper indenting (4 spaces for each indentation level)
- proper Javadoc documentation (as per above)
- proper variable naming conventions
- one Javadoc comment per variable, class and method
- comments go before the thing commented. .
- only one declaration per line.
- adhere to the 80 column rule <= 80 columns

Section 4

Assignment Grading

The grade for this assignment will be assigned on the basis of 10 points for each part:

- up to **3 points** for commenting and following the style guide, and
- up to **5 points** for class' correctness
- up to **2 points** for overall program correctness

Section 5

Schedule

Your project is due on the assigned date at the assigned time. Late assignments will count as zero – absolutely no exceptions. It will be your responsibility to ensure that you've submitted the appropriate files and that you've done so on time. Do not wait until an hour before the assignment is due – you never know if you'll have network trouble.

This policy will be waived only for documented medical situations (not including cold, flu, or simply not feeling well) or other extraordinary circumstances (e.g. war, natural disaster). "The computer or the network was down" is not an unusual circumstance; our response will always be "the computer/network often goes down; you should have allowed yourself more time."

How to hand in COMP 1510 Assignments

1) You must ensure to fill out the readme.txt file found in the `src` (source) directory. This file will be bundled up (in your zip file) with your included source code. The readme.txt file should contain the status of your assignment. Please fill it out accurately for each of the questions (e.g., Q1 is 100% complete, Q2 is 95% complete and so on). Note: *You must ensure that your submitted readme.txt file accurately reflects what you've submitted in your assignment. If the assignment is not complete but your readme.txt file states that it is, you've misrepresented yourself and therefore you will lose marks for this.*
2) Ensure that you've checked over the report generated by ***Checkstyle***. If it contains errors and you do not fix them, you will lose marks for style (i.e., 1-2 out of 10). **If you do not understand any of the code conventions, please see your lab instructor well before the assignment is due**.
3) Remember, you are only submitting the zip file build by the ant script, which contains the source code files and the readme.txt file. You should run the *ant* command to produce this zip file for submission.
4) *Upload your zip file into your **Learning Hub assignment folder** for the assignment. This must be before the due date and time.*

Section 6

Plagiarism and Collaboration on Programming Projects

The assignment you turn in *must* represent your own work and not the work of anyone else. On the other hand, it is unreasonable to expect that you will work in a complete vacuum, without ever speaking to a classmate. The purpose of this section is to give you guidance about the areas in which it is appropriate to discuss assignment topics with your classmates. Violating these guidelines may result in a charge of academic dishonesty.

Plagiarism

The term plagiarism describes an attempt to claim work as your own, which you have copied from another person, whether that other person knows about it or not. In a class like this, plagiarism includes copying program code, data, documentation, etc. Plagiarism is simply not allowed. If you submit another student's work as your own, you will be charged with a violation of the BCIT Academic Integrity Code.

Collaboration

Collaboration is defined as two or more students working together on a phase of a project. Working together does not mean that one student does the work and the other student just copies it! Collaboration is allowed under certain conditions, as long as you are honest about it.

You are taking this class to learn important fundamental things about computing, and we must give you a grade that fairly represents what we think you've learned. Therefore, we need to know that your work is your work, so we need to limit the collaboration somewhat. For purposes of projects in this class, here are some guidelines as to which phases of a project are appropriate for collaboration, and which are inappropriate. This may change from assignment to assignment.

OK	Preliminary analysis of problem
OK	Developing an algorithm
OK	Developing a plan for testing
NO	Coding
NO	Proof-reading the program before compiling
OK	Interpreting errors at compilation time
OK	Interpreting errors at execution time
NO	Testing

Working in pairs

If the assignment explicitly allows or requires people to work in pairs, then both names must appear on the one assignment, which is handed in for the pair. In this case, the rules on collaboration apply to the students in that pair collaborating with anyone else.

Save Your Projects!

You are required to save a copy of all your projects until the end of the semester, after grades have been reported. Be prepared to re-submit these to the instructor if he or she asks you to do so.

Protect Yourself

If you suspect that another student is misusing your work (for example, one of your printouts disappeared), report this immediately to the instructor, to protect yourself against a charge of plagiarism if another student copies your work. If there is ever any confusion as to who copied from who, it is institute policy to charge both with plagiarism and punish both (or all) parties.

Read the BCIT Policy on Conduct carefully.