CSE 8B: Introduction to Programming and Computational Problem Solving - 2

Assignment 2

Numbers, Mathematics, Characters, and Strings in Java

Due: Wednesday, April 19, 11:59 PM

Learning goals:

- Write Java code that includes:
  - Numbers
  - Mathematical operations
  - String and character parsing
- Write your own test cases to test the correctness of your methods

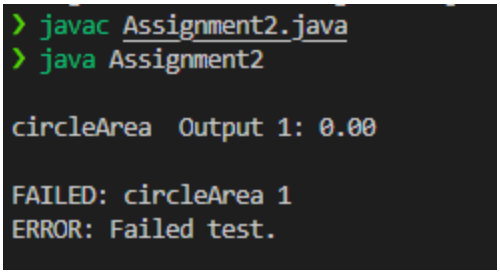
NOTE: This assignment should be completed INDIVIDUALLY.

Coding Style (10 points)

For this programming assignment, we will be enforcing the CSE 8B Coding Style Guidelines. These guidelines can also be found on Canvas. Please ensure to have *COMPLETE* file headers, class headers, and method headers, to use descriptive variable names and proper indentation, and to avoid using magic numbers.

Part 0: Getting started with the starter code (0 points)

1. If using a personal computer, then ensure your Java software development environment does not have any issues. If there are any issues, then review Assignment 1, or come to the office/lab hours before you start Assignment 2.
2. First, navigate to the `cse8b` folder that you created in Assignment 1 and create a new directory named `assignment2`
3. Download the starter code.
   You can download the starter code from Piazza → Resources → Homework → `Assignment2.java`
   Place the starter code in the `assignment2` folder you just created
4. Compile and run the starter code, and you should expect the following output:



Although we have not covered Java methods in lecture yet, you can think of a Java method as a math function. A method takes in zero or more parameters (input) and returns a value (output). The starter code is set up so you need not know anything about the details of creating a method from scratch. Your job is to just fill in the starter code.

Part 1: Implement seven methods (75 points)

In class `Assignment2` of the starter code, 9 methods are already declared for you: `circleArea`, `triangularPrismVolume`, `minCorrectDivision`, `concatSubIntegers`, `concatAndModify`, `substringIndex`, `createStarHourglass`, `unitTests`, and `main`. **For this part of the assignment, your task is to implement the first seven methods (`circleArea`, `triangularPrismVolume`, `minCorrectDivision`, `concatSubIntegers`, `concatAndModify`, `substringIndex`, and `createStarHourglass`).**

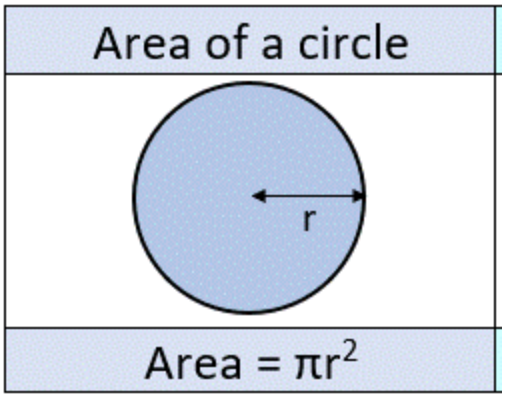
Before you implement your methods, please take a look at the constants at the top of the class. All of the error messages as well as method-specific variables are already defined for you with the keywords `private final static`. **IMPORTANT: You should use these constants when developing your methods to ensure that your code will always give the correct output. You technically do not have to but using them will help you remain consistent.**

`circleArea` (10 points):

This method takes a `double` variable, `radius`, as the input which is the radius of a circle, and the method must calculate the area of the circle with radius `radius`. Assign your result to the variable `area`.

You can assume that the radius will always be greater than or equal to 0.

The formula to calculate the area of a circle is shown below.

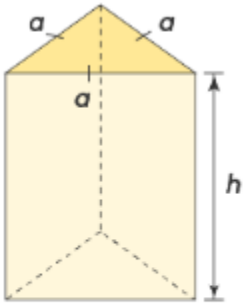


Sample:

`circleArea(3.0)` → `28.27`

`triangularPrismVolume` (10 points):

This method takes two `double` variables, `triangleSideLength` and `height` as input, and calculates the volume of an equilateral triangular prism with that side length and height. Assign your result to the variable `volume`.



$$V = \frac{\sqrt{3}}{4} \times a^2 \times h$$

IMPORTANT: **We are only concerned with equilateral triangular prisms in this assignment**. So `triangleSideLength` will be the same on all three sides.

The volume of an equilateral triangular prism is shown below.

Sample:
`triangularPrismVolume(3.5, 10)` → `53.04`

minCorrectDivison (15 points):

This method takes in four integer variables `a`, `b`, `c`, and `d`. You must perform floating point division between `a` and `b` (`a`/`b`) and likewise for `c` and `d` (`c`/`d`). Then you must return the minimum of these two quantities as a `double` type. Note that because the variables are passed in as integers, truncation may happen and yield a wrong result. Hence, you must find a way around this issue. Calculate the result, and assign it to the variable `ans`.

Sample:
`minCorrectDivision(10, 7, 2, 3)` → `0.67`

concatSubIntegers (10 points)

This method takes in six variables where `x` and `y` represents an integer but in `String` form:

1. `String x`
2. `int a`
3. `int b`
4. `String y`
5. `int c`
6. `int d`

Your job for this method is to extract the substring from `x` starting at index `a` and up to but not including index `b` and combine it with the substring from `y` starting at index `c` and up to but not including index `d`. Then convert this combined result into an integer and assign it to the variable `ans`. You may assume that the final answer will not have any leading zeros and that the indexes are valid for the corresponding strings.

Sample:
`concatSubIntegers("56910", 0, 2, "12345", 3, 4)` → `564`

concatAndModify (15 points):

This method takes two string variables `stringOne`, and `stringTwo` as input. The method does two things:

- combines the two strings

- convert the first and last letter of the combined string into uppercase letters
- it must leave the rest of the characters unchanged

You can assume that the strings `x` and `y` will only contain characters from the english alphabet (26 letters), though the characters can either be lowercase or uppercase. You may also assume that the strings will not be empty. Assign your answer to the variable `ans`.

Samples:

`concatAndModify("hello", "world")` → `"HelloworlD"`
`concatAndModify("Mouse", "Trap")` → `"MouseTraP"`
`concatAndModify("Boxed", "milK")` → `"BoxedmilK"`

substringIndex (10 points):

This method takes in three `String` variables `a`, `b`, and `c`. You must find the first occurrence of `c` in `b` and keep track of the indexes in `b` where `c` begins and ends. Then return the corresponding substring in `a` with the same start and end indexes. You may assume that `c` will always be contained in `b` at least once and that the start and end indexes will be valid ranges within `a`. Extract the result and assign your answer to the variable `result`.

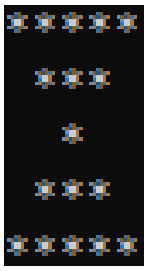
Sample:
`substringIndex("welcometocse8b", "helloworldello", "ello")` → `"elco"`

Explanation:
Because `c` (`"ello"`) occurs twice in `String` `b` (`"helloworldello"`), we keep track of the first occurrence which starts at index 1 and ends at index 4. With these start and end indices, we will extract the corresponding substring within `String` `a` (`"welcometocse8b"`) which results in `"elco"`.

`createStarHourglass` (5 points):

This method does not take in any inputs. Construct and return a String that uses the asterisk character `'*'` to create an hourglass with the number of rows equal to **seven**.

For example, if we creating an hourglass of height **five** then the result string should be: `"*****\n *** \n  *  \n *** \n*****\n"` (where `\n` is the new line character). When this string is printed out, it should look like this:



NOTE: Notice how each row has one extra space in the **beginning and end** until you reach the middle of the hourglass which is just a single `'*'` which then starts reversing. In this example, there are 5 rows. **In your method, you must construct and return an hourglass whose height (number of rows) is 7.**

Your job is to construct the hourglass and assign it to the variable `result`.

IMPORTANT (Will lose points if not followed):

1. There must be the right amount of spaces **before and after** each row. More importantly, a common mistake is if you forget the spaces **after** the asterisks and print out the string, it may look correct in your terminal but will fail the autograder.

2. Append the newline character `'\n'` after the last space (if any) in every row **including the last row**. Do not append anything else. Any extra characters will fail the autograder.

3. As long as your output matches the format of the result string literal shown in quotation marks above (for a height 7 hourglass), you should pass.

Part 2: Test the correctness of your seven methods (10 points)

Testing is a very important part in programming. In this course, we will get you familiar with unit testing. For this assignment and all future assignments, you will be asked to create your own tests to check whether your code works as expected. **In this part of the assignment, you need to implement your own test cases in the method called `unitTests`.**

In the starter code, several test cases are already implemented for you. You can regard it as an example to implement other cases. The general approach is to come up with different inputs and manually give the expected output, then call the method with that input and compare the result with expected output.

You are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. The `unitTests` method should `return true` only when all the test cases are passed. Otherwise, it should `return false`. **To get full credit for this section, for each of**

your seven methods above (except for `createStarHourGlass`), you should have at least 3 total test cases that cover different situations (including the ones we have provided). In other words, you will need to create two more tests for `circleArea`, two more tests for `triangularPrismVolume`, two more tests for `minCorrectDivision`, two more tests for `concatSubIntegers`, two more tests for `concatAndModify`, and two more tests for `substringIndex`.

For methods that potentially return a `double` type which can be very long, you will round the output to two decimal places by using `String.format()` then comparing it with the expected result. An example is shown below and the starting code includes more examples:

An easy way to round the volume to 2 digits of accuracy is to use `String.format()`. To round to 2 digits, the format string should be `"%.2f"`. Here is an example of how `String.format()` works:

```
String myStr = String.format("The value is: %.2f", 12.478223);
System.out.println(myStr);
// This will output: "The value is: 12.48"
```

Part 3: Complete `main` (5 points)

After completing the three methods and creating several unit tests, compile and run `Assignment2`. You should see the message `"All unit tests passed"`. If not, then it is very likely that you have bugs in your code. Read the previous parts carefully while inspecting your code to fix your bugs. We call this process debugging.

The main method is the method that will be called when running program `Assignment2`. Here is the main method given to you in the starter method:

```java
// TODO: Complete the method header and the method body
Run | Debug
public static void main(String[] args) {

    if (unitTests()) {
        System.out.println("All unit tests passed.\n");
    } else {
        System.out.println("ERROR: Failed test.\n");
        return;
    }
    Scanner scanner = new Scanner(System.in);
    System.out.println(PROMPT_MSG_NAME);

    scanner.close();
}
```

The code to run `unitTests` is already given to you. **Do NOT change any code that is provided to you in main. You should only add more lines of code.**

First, your program should print the prompt `"Please enter your name."` (which is already done in the starter code) and wait for the user to enter feedback.

1. The user will enter their name.
2. You should then print out “Hello [name]! Nice to meet you and welcome to CSE8B.” on a new line where [name] is the string that the user inputted.

Example: you should be able to *exactly* reproduce this output below with your program after the unit testing. If you cannot, then you will lose points on this portion of the assignment.

```
All unit tests passed.

Please enter your name.
Darren
Hello Darren! Nice to meet you and welcome to CSE8B.
```

Submission

VERY IMPORTANT: Please follow the instructions below carefully and make the exact submission format.

1. Go to Gradescope via Canvas and click on Assignment 2.

2. Click the DRAG & DROP section and directly select the required file (`Assignment2.java`). Drag & drop is fine. **Please make sure you don't submit a zip. Make sure the filename is correct.**
3. **You can resubmit an unlimited number of times before the due date.** Your score will depend on your final submission, even if your former submissions have a higher score.
4. The autograder is for grading your uploaded files automatically. Make sure your code can compile on Gradescope.

   NOTE: The Gradescope Autograder you see is a minimal autograder. For this particular assignment, it will only show the compilation results and the results of basic tests. After the assignment deadline, a thorough Autograder will be used to determine the final grade of the assignment. **Thus, to ensure that you would receive full points from the thorough Autograder, it is your job to extensively test your code for correctness via `unitTests`.**

5. Your submission should look like the screenshot below. **If you have any questions, then feel free to post on Piazza!**

Submit Programming Assignment

Upload all files for your submission

SUBMISSION METHOD

Upload GitHub Bitbucket

Add files via Drag & Drop or Browse Files.

NAME	SIZE	PROGRESS
Assignment2.java	**7.5** KB	

STUDENT NAME (OPTIONAL)

Enter student name

Upload Cancel