CSE 11: Introduction to Programming and Computational Problem Solving: Accelerated Pace

Programming Assignment 2

Numbers, Mathematics, Characters, and Strings in Java

Due: Wednesday, April 17, 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 must be completed INDIVIDUALLY.

Coding Style (10 points)

For this programming assignment, we will be enforcing the CSE 11 Coding Style Guidelines. These guidelines can also be found on Canvas and the class website. Please ensure to have *COMPLETE* file headers, class headers, and method headers, use descriptive variable names and proper indentation, and 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 `cse11` 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:

```
sphere volume  Output 1: 0.0

FAILED: sphereVolume 1
ERROR: Failed test.
```

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: `sphereVolume`, `euclideanDistance`, `divisionDifference`, `concatSubIntegers`, `concatAndModify`, `replaceSubstring`, `createDiamond`, `unitTests`, and `main`. **For this part of the assignment, your task is to implement the first seven methods (**`sphereVolume`, `euclideanDistance`, `divisionDifference`, `concatSubIntegers`, `concatAndModify`, `replaceSubstring`, and `createDiamond`**).**

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 your code will always give the correct output. You technically do not have to but using them will help you remain consistent.**

1- `sphereVolume` (10 points):

Write a Java method called `sphereVolume` that takes a `double` variable, `radius`, as input. The method must calculate the volume of a sphere with the given radius. Assign your result to the variable `volume`. You can assume that the radius will always be greater than or equal to 0.

The formula to calculate the volume of a sphere is:

$$Volume = \frac{4}{3}\pi r^3$$

Sample: `sphereVolume(3.0)` → `113.10`

2- `euclideanDistance` (10 points):

This method, `euclideanDistance`, takes four `double` variables, `x1`, `x2`, `y1`, and `y2`, as inputs. These variables represent the coordinates of two points in a 2-dimensional plane: (x_1, y_1) and (x_2, y_2) respectively. The method must calculate the Euclidean distance between these two points and return the result as a double.

The Euclidean distance is calculated by taking the square root of the sum of the squares of the differences in the coordinates:

$$distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Calculate the result and assign it to the variable `distance`.

Sample:

`euclideanDistance(2.5, 5.2, 3.7, 7.8)` → `4.91`

3- `divisionDifference` (15 points):

Method `divisionDifference` takes four `int` variables `a`, `b`, `c`, and `d` as inputs. The method must calculate the floating-point division of `a` by `b` and `c` by `d`, then return the absolute difference between these two results as a `double` value.

Note that because the variables are passed in as `int`s, truncation may happen and yield an incorrect result. Hence, you must find a way around this issue. Calculate the result, and assign it to the variable `diff`.

Sample:
`divisionDifference(10, 7, 2, 3)` → `0.76`

4- `concatSubIntegers` (10 points)

This method takes in three variables where `num` represents an integer but in `String` form:

1. `String num`
2. `int a`
3. `int b`

Your job for this method is to extract the substring from `num` starting at index `a` and up to but not including index `b`. Then append this substring to the end of string `num`. Convert both `num` and the newly created concatenated string to `int`s. Assign the absolute difference of the two `int`s to variable `ans`.
Sample:
`concatSubIntegers("56910", 0, 2)` → `5634146`

5- `concatAndModify` (15 points):

This method takes three `String` variables `word1, word2,` and `i` as input where `i` represents an integer value in a string form. Here is what the method must do:

- Append `word2` to the end of `word1`
- Find the length of the concatenated string and assign it to `len`
- Convert the `i`-th letter and (`len - i`)-th letter of the concatenated string into uppercase letters
- The rest of the characters must remain unchanged

You can assume the `Strings` `word1` and `word2` 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. And it is safe to assume that `i` is smaller than the length of the concatenated string. Assign your answer to the variable `result`.

Samples:

`concatAndModify("hello", "world", "3")` → `"helLowoRld"`

6- `replaceSubstring` (10 points):

`replaceSubstring` takes in three `String` variables: `a`, `b`, and `c`. Your task is to find the first occurrence of `b` within `a`. Once found, extract the substring from `a` starting at the index of the first occurrence of `b` and ending at the index of the last character of `b`. Then, replace that substring in `a` with `c`. Assign the updated string to `answer`. You may assume that `b` will always be contained in `a` at least once.

Sample:

`replaceSubstring("helloworld", "world", "Sara")` → `"helloSara"`

7- `createDiamond` (5 points):

Write a Java method named `createDiamond` that constructs and returns a `String` representing a diamond shape using the asterisk character `'*'`. The diamond must have the number of rows equal to **seven**.

Your task is to generate a diamond shape with seven rows, where the width of the diamond increases from the top row to the middle row, and then decreases from the middle row to the bottom row.

For example, if we create a diamond of height **five** then the resulting string would be: `"  *\n ***\n*****\n ***\n  *\n"` (where `\n` is the new line character). When this string is printed out, it would look like this:

Your job is to construct the diamond and assign it to the variable `res`.

IMPORTANT (Will lose points if not followed):

1. There must be the right amount of spaces **before** each row.
2. Append the newline character `'\n'` after the last asterisk(*) 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 (but for a height **seven** diamond), 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 these as examples 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 the expected output.

You are encouraged to create as many test cases as you think are necessary to cover all the edge cases. The `unitTests` method must `return true` only when all the test cases are passed. Otherwise, it must `return false`. **To get full credit for this section, for each of your seven methods above (except for `createDiamond`), you must 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 `sphereVolume`, two more tests for `euclideanDistance`, two more tests for `divisionDifference`, two more tests for `concatSubIntegers`, two more tests for `concatAndModify`, and two more tests for `replaceSubstring`.

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()` and then compare it with the expected result. An example is shown below and the starter 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 is `"%.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"
```

As you work on implementing methods for this assignment, it's crucial that you test your code thoroughly by running the provided test suite after implementing each method. Regular testing ensures your code works as expected and catches bugs early, saving time and effort.

Part 3: Complete `main` (5 points)

After completing the seven 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:

```
// 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 must 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 must then print out “Hello [name]! Nice to meet you and welcome to CSE11.” on a new line where [name] is the string that the user inputted.

Example: you must 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.

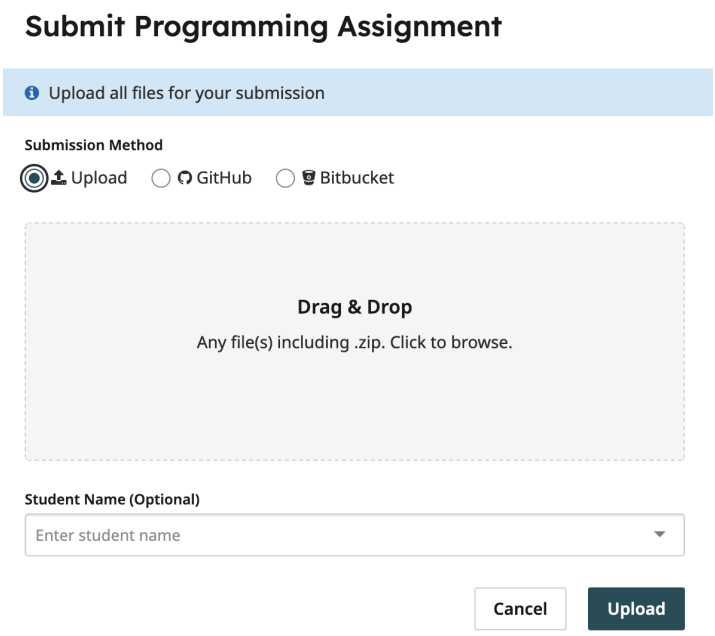
```
All unit tests passed.

Please enter your name.
Sara McCoy
Hello Sara McCoy! Nice to meet you and welcome to CSE11.
```

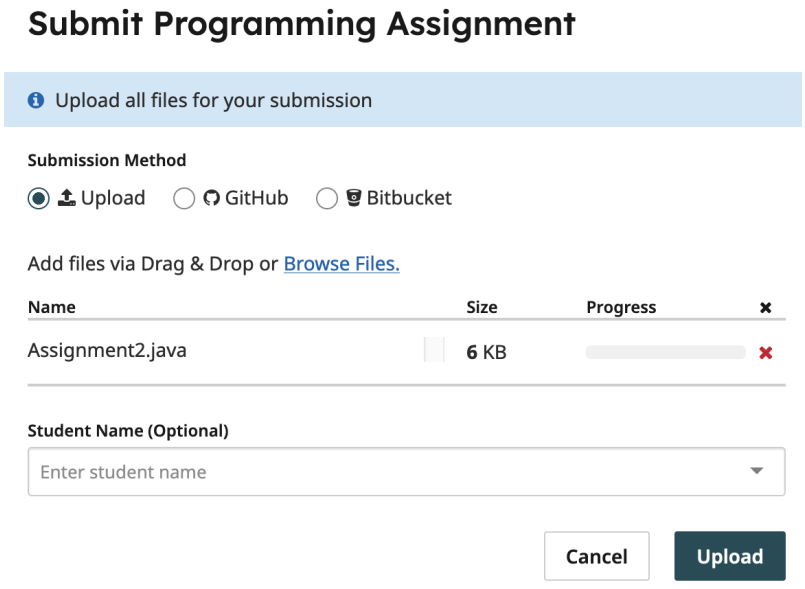
Submission

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

1. Go to Gradescope via Canvas and click on PA2.
2. To upload your Assignment2.java code, you can either drag the file from your computer to the “Drag & Drop” section[shown below]. Or, you can click on “Drag & Drop” section which prompts you to directly select the required file (Assignmet2.java in this case). **Please make sure you do not submit a zip. Make sure the filename is correct.**



3. Following the previous step, our submission should look like the screenshot below. Click upload to submit your file.



4. **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.
5. 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`.**

6. **If you have any questions, then feel free to post on Piazza!**