Indiana University
CSCI-C212: Introduction to Software Systems (F25)
Joshua Crotts

Problem Set 4
September 17, 2025

Loops

Important Dates:

- Assigned: September 17, 2025
- Deadline: September 24, 2025 at 11:59 PM EST

Objectives:

- Students understand the direct correspondence between iteration and tail recursive methods.
- Students design methods that use iteration to solve complex problems.
- Students make the choice to use iteration versus recursion when the time calls for it.

What To Do:

For each of the following problems, create a class named `ProblemX`, where `X` is the problem number. E.g., the class for problem 1 should be `Problem1.java`. Write (JUnit) tests for each method that you design in corresponding test files named `ProblemXTest`, where `X` is the problem number. Additionally, write Javadoc comments explaining the purpose of the method, its parameters, and return value. **Do not round your solutions!**

What You Cannot Use:

You cannot use any content beyond Chapter 2.3. This includes arrays, regular expressions, data structures, and so forth. Please contact a staff member if you are unsure about something you should or should not use.

Any use of anything in the above-listed forbidden categories will result in a **zero** (0) on the problem set.

Problem 1:

Recall the `isPalindromeTR` method from the last problem set. Design the `boolean isPalindromeLoop(String s)` method that solves the problem using a loop. The same restrictions (i.e., no `StringBuilder`, etc.) hold true for this problem.

Problem 2:

Recall the `hyperfactorial` and `hyperfactorialTR` problems from the last problem set. (Refer back to that problem set for the hyperfactorial definition.) Design the `long hyperfactorialLoop(long n)` method that solves the problem using a loop.

Problem 3:

Recall the `subfactorial` and `subfactorialTR` problems from the last problem set. (Refer back to that problem set for the hyperfactorial definition.) Design the `long subfactorial(long n)` method that solves the problem using a loop.

Problem 4:

Recall the `collatz` and `collatzTR` methods from the last problem set. Design the `String collatzLoop(int n)` method that solves the problem using a loop.

Problem 5:

Recall the `collectParenthesizedStrings` and `collectParenthesizedStringsTR` methods from the last problem set. Design the `String collectParenthesizedStringsLoop(String s)` method, which solves the problem using a loop.

Problem 6:

The C programming language contains the `atoi` "ascii-to-integer" function, which receives a string and, if the string represents some integer, returns the number converted to an integer. Design the `int atoi(String s)` method that, when given a string s, returns its value as an integer if it can be parsed as an integer. An integer may contain a sign as the first character, that being + or −. If the integer does not have a sign, it is assumed to be positive. Ignore all leading zeroes and leading non-digits. Upon reading a sign, the remaining characters must be digits for the number to be a valid integer. Upon finding the first non-zero digit (after the sign), if one exists, begin interpreting the string as a number. At any point thereafter, if a non-digit is encountered, return the number parsed up to that point. You may assume that the bounds of an integer are never exceeded, i.e., all valid integers will be within the bounds of $[-2^{31}, 2^{31} - 1]$. Writing enough tests is *crucial* to correctly solving this exercise! We provide some examples below. **You cannot use methods that trivialize the problem, e.g., `Integer.parseInt`.**

```
atoi("ABCD")           => 0
atoi("42")             => 42
atoi("000042")         => 42
atoi("004200")         => 4200
atoi("ABCD42ABCD")     => 42
atoi("ABCD+42ABCD")    => 42
atoi("ABCD-42ABCD")    => -42
atoi("000-42000")      => -42000
atoi("000-ABCD")       => 0
atoi("-+-+1234")       => 0
atoi("-A1234")         => 0
atoi("000+42ABCD")     => 42
atoi("8080*8080")      => 8080
```

Problem 7:

Wordle is a game created by Josh Wardle, where the objective is to guess a word with a given number of turns, inspired by Mastermind. In this exercise, you will implement a stage of the Wordle game.

Design the `String guessWord(String W, String G)` method that, when given a string W and a "guess" string G, returns a new string with the following properties:

- If $|W| \neq |G|$, return `null`.
- For every index i, if $W_i = G_i$, append W_i to the output string. If $W_i \neq G_i$ but $G_i \in W$, append an asterisk to the output string. Otherwise, output a dash.

Below are some example inputs and outputs.

```
guessWord("PLANS", "TRAP")  => null
guessWord("PLANS", "TRAIN") => "--A-*"
guessWord("PLANS", "PLANE") => "PLAN-"
guessWord("PLANS", "PLANS") => "PLANS"
guessWord("PLANS", "SNLPA") => "*****";
```

Problem 8:

Design the `String substring(String s, int a, int b)` method, which receives a string and two integers a, b, and returns the substring between these indices. If either are out of bounds of the string, return `null`. You **cannot** use the `substring` method(s) provided by the `String` class.

Problem 9:

File names are often compared lexicographically. For example, a file with name `"File12.txt"` is less than `"File2.txt"` because `'1'` is less than `'2'`. Design the `int compareFiles(String f1, String f2)` method that would fix this ordering to return the more sensible ordering. That is, if a file has a prefix and a suffix, where the only differing piece is the number, then make the file with the lower number return a negative number. You may assume that the file names are always of the form `NameNum.Extension`, where `Name` is some alphabetic string (i.e., where the string contains only letters), `Num` is some positive integer, and `Extension` is another alphabetic string. If the prefix or or suffix do not match, return the sign (i.e., -1, 0, 1) of calling `compareTo` on the file names. Take the following examples as motivation.

```
compareFiles("File12.txt", "File1.txt")  => 1
compareFiles("File10.txt", "File11.txt") => -1
compareFiles("File1.txt", "File12.txt")  => -1
compareFiles("File1.txt", "File1.txt")   => 0
compareFiles("File1.txt", "File1.png")   => 1
```

Warning: if you try to Google this problem (or use AI for this problem), it will almost certainly tell you to use regular expressions or some other ridiculous approach. If you do, you will receive a 0 on the problem set. Again, *both of these are violations of the academic integrity policy.* Don't give in. Solve it yourself, like you should for all other problems.

Problem 10:

The *definite integral* of a function f, defined as $\int_a^b f(x)\,\mathrm{d}x$, produces the area under the curve of f on the interval $[a, b]$. The thing is, though, integrals are defined in terms of *Riemann summations*, which provide estimations on the area under a curve. Riemann sums approximate the area by creating rectangles of a fixed width Δ, as shown in 4 for an arbitrary function f. Left-Riemann, right-Riemann, and midpoint-Riemann approximations define the focal point, i.e., the height, of the rectangle. Notice that, in Figure 4, we use a midpoint-Riemann sum with $\Delta = 0.2$, in which the collective sum of all the rectangle areas is the Riemann approximation. Your job is to use this idea to approximate the area of a circle.

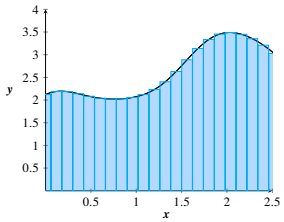

Figure 4: Midpoint-Riemann Approximation of a Function

Design the `double circleArea(double r, double delta)` method, which receives a radius r and a delta Δ. It computes (and returns) a **left**-Riemann approximation of the area of a circle. Hint: if you compute the **left**-Riemann approximation of one quadrant, you can very easily obtain an approximation of the total circle area. We illustrate this hint in Figure 5 where $\Delta = 0.5$ and its radius $r = 2$. Note that the approximated area will vary based on the chosen Riemann approximation, so for this problem, you should approximate the area of quadrant 1, then multiply that value accordingly.[1] Further note that no calculus knowledge is necessary to solve this exercise.

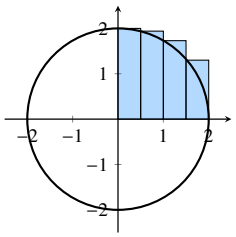

Figure 5: Left-Riemann Approximation of a Function

[1] A left-Riemann sum over-approximates the area in quadrant 1, whereas a right-Riemann sum provides an under-approximation in quadrant 1. A midpoint approximation uses the average between the left and right approximations.