Homework 2 - Sequences

Problem Description

After completing the introductory homeworks, you should have a good idea of how to create and manipulate variables, how to perform calculations and output those results, and finally, how to accept and include user input in your calculations and results. This assignment will focusing on *control flow*, specifically conditionals and loops. To brush up on these concepts, check out the links below:

- conditionals
- while loops
- for loops

You will create a program, `NumberSequences.java`, that will allow a user to generate a few sequences of numbers following a given rule (or rules)!

Make sure you read the entire document! There are some things even at the very end that you will not want to miss for this and future assignments!

Defining a Sequence

A *sequence* is a series of numbers that follow a pattern or property. For example, the sequence of even numbers less than 10 is: 2, 4, 6, and 8.

Here, every number is less than 10, and every number is even.

Solution Description

For `NumberSequences.java`, you will have two sequences to implement:

- the Collatz sequence.
- the Fibonacci sequence.

The user will be prompted to first select one of these sequences to view. They will then provide an integer parameter in order to compute the appropriate sequence. The instructions below will first outline the sequences, then explain flow of the program.

The Collatz Sequence

Given some starting integer, which we'll call `startNum`, it is always possible to arrive at 1 by repeating the following operations in sequence:

- if the number is even, divide the number by 2.
- if the number is odd, multiply the number by 3, and add 1.

For example, if my starting number were 5, then my sequence would be:

`Collatz Sequence: 5 16 8 4 2 1`

Note that `startNum` can be any number from 1 to 100, both ends inclusive. We also want you to display the **the number of steps** from `startNum` to the final number, 1. So, in the case of `startNum = 5`, you can see there are 5 steps (count the spaces in between the numbers).

The user should be prompted to enter the starting number, and the results should be displayed as formatted below:

```
Enter the starting number (1 - 100): 3

Collatz Sequence: 3 10 5 16 8 4 2 1
Number of steps: 7
```

In short, given a starting number you should:

- generate the Collatz Sequence beginning with that number.
- give the amount of steps it takes to get to 1 (i.e. terminate the sequence).

The Fibonacci Sequence

Given the initial Fibonacci numbers 0 and 1, we can generate the next number by adding the *two previous* Fibonacci numbers together. For this sequence, you will be asked to take an input, denoting how many Fibonacci numbers you want to generate. Call this input `upperFibLimit`. The longest Fib sequence you should generate is 40 and the shortest you should generate is 1. So, 1 <= `upperFibLimit` <= 40.

The rule is simple - given f(0) = 0, f(1) = 1:

- `f(n) = f(n - 2) + f(n - 1)`

Where n is the index of the fibonnaci number in the sequence. All you have to do is generate the sequence. *Think about the initial numbers, and what you have to do to get the next number.*

Note: The Fibonacci sequence is commonly generated using *recursion*. Do not do this! The sequence must be generated iteratively! That is, **you must generate the sequence using only while, do-while, for, or for-each loops!**

The user should be prompted for the length of the sequence to generate, and then the sequence should be displayed as follows:

```
Enter the length of the desired fib sequence (1 - 40): 8

Fib Sequence: 0 1 1 2 3 5 8 13
```

Control Flow of the Program

Your program should prompt the user to input which sequence they want to view. The user will then be prompted to input a number specific to that sequence, which will then be used to calculate and display the specific sequence. After computing any given sequence, the user will be brought back to the first prompt, where the process starts all over again. The user can input `E` to exit the program at the main prompt. "E" is **only valid** when prompted with the following: `Enter the first character of the sequence to generate...`

Here is a full example of all the major parts of the UI:

```
$ java NumberSequences
Enter the first character of the sequence to generate
(C)ollatz, (F)ib, or (E)xit: C
Enter the starting number (1 - 100): 3

Collatz Sequence: 3 10 5 16 8 4 2 1
Number of steps: 7
--------------------
Enter the first character of the sequence to generate
(C)ollatz, (F)ib, or (E)xit: F
Enter the length of the desired fib sequence (1 - 40): 8

Fib Sequence: 0 1 1 2 3 5 8 13
--------------------
Enter the first character of the sequence to generate
(C)ollatz, (F)ib, or (E)xit: E
--------------------
$
```

MAKE SURE YOU **FOLLOW FORMATTING EXACTLY** (spaces, newlines, dashes, . . .).

Note the -'s separating each iteration of our program, these need to be printed in the indicated formatting (there are 20 per line). Additionally, note that the responses to the prompts are user input and not output from the program.

Scanner Usage

Using multiple `Scanner`s on `System.in` is bad style and leads to bugs. As such, **only use a single Scanner, and don't declare or assign it within a loop or conditional statment**. The autograder will fail if you do not follow this instruction.

Valid (and Invalid) Inputs

You do not need to deal with cases where input is invalid. This means that for example, if prompted for an integer between 1 and 100, you may assume that the user will provide an integer between 1 and 100, as opposed providing say the character 'a'.

Allowed Imports

To prevent trivialization of the assignment, you are only allowed to import the following classes:

- java.util.Scanner

If you would like to import anything else, ask on Piazza.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach. For that reason, do not use any of the following in your final submission:

- var (the reserved keyword)
- System.arraycopy

Checkstyle

You must run checkstyle on your submission. The checkstyle cap for this assignment is **5** points. Review the style guide and download the checkstyle jar. Run checkstyle on your code like so:

```
$ java -jar checkstyle-6.2.2.jar *.java
Audit done. Errors (potential points off):
0
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the checkstyle cap mentioned above). Any Java source files we provide contain no Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the customization tips page for more information.

Rubric

[100] **NumberSequences.java**

- [5] Exit command 'E' successfully terminates program
- [5] Program uses a single Scanner and loops until user enters 'E'
- [40] Collatz
  - [5] Correct input prompt format

    - [10] Correct output format
    - [10] Correct sequence and steps for basic case
    - [15] Correct sequence and steps for all cases
- [40] Fib
    - [5] Correct input prompt format
    - [10] Correct output format
    - [10] Correct sequence for basic case
    - [15] Correct sequence for all cases
- [10] Correctly generates multiple sequences in a row

We reserve the right to adjust the rubric, but this is typically only done for correcting mistakes.

Collaboration

Collaboration Statement

To ensure that you acknowledge a collaboration and give credit where credit is due, **we require that you place a collaboration statement as a comment at the top of at least one java file that you submit**. That collaboration statement should say either:

I worked on the homework assignment alone, using only course materials.

or

In order to help learn course concepts, I worked on the homework with [give the names of the people you worked with], discussed homework topics and issues with [provide names of people], and/or consulted related material that can be found at [cite any other materials not provided as course materials for CS 1331 that assisted your learning].

Allowed Collaboration

When completing homeworks for CS1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution

You may **not** discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others. In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `NumberSequences.java`

Make sure you see the message stating “HW## submitted successfully”. From this point, Gradescope will run a basic autograder on your submission as discussed in the next section.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the homework. We will only grade your last submission: be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. Each test typically corresponds to a rubric item, and the score returned represents the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

1) Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
2) Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine.

Other portions of your assignment are also autograded once the submission deadline has passed. The autograders used are often dependent on specific output formats, so **make sure that your submission passes all test cases marked "FORMAT:"**.

Important Notes (Don't Skip)

- Non-compiling files will receive a 0 for all associated rubric items
- Test your code in addition to the basic checks on Gradescope
- Submit every file each time you resubmit
- Ensure you pass all "FORMAT:" tests
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points
- Check on Piazza for a note containing all official clarifications
- Only use a single Scanner, and don't declare or assign it within a loop