COMP 2402 Winter 2022
Assignment 4

Due: Thursday March 24 2:00pm
Topic focus: Lec 14 - 17
lates accepted within 24 hours
submit early and often

Academic Integrity

You may:

- Discuss general approaches with course staff,
- Discuss general approaches with your classmates,
- Use code and/or ideas from the textbook,
- Use a search engine / the internet to look up basic Java syntax,
- Send your code / screen share your code with course staff (although it is unlikely the course staff has the bandwidth to look at individuals' code, unfortunately.)

You may **not**:

- Send or otherwise share code or code snippets with classmates,
- Use code not written by you, unless it is code from the textbook (and you should cite it in comments),
- Use a search engine / the internet to look up approaches to the assignment,
- Use code from previous iterations of the course, unless it was solely written by you,
- Use the internet to find source code or videos that give solutions to the assignment.

If you ever have any questions about what is or is not allowable regarding academic integrity, **please do not hesitate to reach out to course staff**. We will be happy to answer. Sometimes it is difficult to determine the exact line, but if you cross it the punishment is severe and out of our hands. **Any student caught violating academic integrity, whether intentionally or not, will be reported to the Associate Dean** and be penalized as follows:

- First offence: F in the course.
- Second offence: One-year suspension from program.
- Third offence: Expulsion from the University.

These are standard penalties. More-severe penalties will be applied in cases of egregious offences. For more information, please see Carleton University's Academic Integrity Policy.

Coding Environment Setup

If you want suggestions on how to set up your coding environment, please watch this video or read the "Setting Up Your Programming Environment" post on piazza.

If you want Alexa's 20-minute "Assignment 1 setup" video, please go here. The setup for assignment 1 is almost identical to that of assignment 2 so there will not be a new video.

How to Get Help

Problem Solving & Assignment Tips Document

- We made this document just for you, full of tips to help you complete your programming assignments faster and better! We will add to it as the course progresses, and you are welcome to add your suggestions as well.

Piazza

- Good for getting a quick response, since it goes to a wider audience than email.
- Good for viewing questions that other students have. Chances are, if you have the question, so does someone else. If tagged correctly (e.g. "a2" for assignment 2), you can easily see all questions relating to a particular subject.
- Good for general questions that don't divulge your approach (e.g. "Am I allowed to use standard JCF data structures in my solution?" "Does anyone else get this strange "file not found" error when submitting?" etc.)
- Good for asking specific-to-you questions of the course staff, which you can ask "privately" on piazza. This gets you the benefits of a wider audience yet keeps your personal details private to course staff. (You can and should also contact Alexa this way.)

Student Hours

- Good for quick questions that may have some back and forth.
- Good for clarifications.
- Good for "tips and tricks".
- Not good for debugging sessions. The person holding the student hour may have many people waiting, and as such cannot spend 20 minutes helping you debug, unfortunately. They may give you a push in the right direction, then ask you to go back in line while you work with that push, while they help someone else.

Discord

- Good for light social interactions and commiseration.

- Please remember that this discord is an official class discord, and as such you should keep your behaviour “professional”. Please be respectful. Jokes are great, just not at the expense of an individual or a specific group. Disrespectful behaviour (intentional or not) will be met with our zero-tolerance policy (removal from server.)
- This is not a good place for asking questions, as the course staff will be spending most of their time monitoring piazza. Piazza has a better system for tracking open questions and answers, and so it is more time efficient for the course staff to spend their time there instead of discord.

Grading

This assignment will be tested and graded by a computer program (and **you can submit as many times as you like, your highest grade is recorded**). For this to work, there are some important rules you must follow:

- Keep the directory structure of the provided zip file. **If you find a file in the subdirectory comp2402w22a4 leave it there.**
- Keep the package structure of the provided zip file. **If you find a package comp2402w22a4; directive at the top of a file, leave it there.**
- Do not rename or change the visibility of any methods already present. If a method or class is public, leave it that way.
- Do not change the `main(String[])` method of any `PartX` class. These are setup to read command line arguments and/or open input and output files. Don't change this behaviour. Instead, learn how to use command-line arguments to do your own testing.
- **Submit early and reasonably often.** The submission server compiles and runs your code, and gives you a mark. You can submit once every 5 minutes and we keep track of your highest overall score. There is no excuse for submitting code that does not compile or does not pass tests. The 5 minute wait limit is to prevent you from using the server to debug, potentially overloading it. (You should debug locally! More on that later.)
- Write efficient code. The submission server places a limit on how much time it will spend executing your code, even on inputs with a million lines. For some questions it also places a limit on how much memory your code can use. If you choose and use your data structures correctly, your code should easily execute within the time limit. Choose the wrong data structure, or use it the wrong way, and your code will be too slow or use too much space for the submission server to work (resulting in a failure of that test).

Submitting and Server Tests

The submission server is here. If this is your first time submitting, you will need to get a secret key which you will then use to upload your assignment to the submission server. If you have issues, please post to piazza to the Instructors (or the class) and we’ll see if we can help.

When you submit your code, the server runs tests on your code. **These are bigger and better tests than the small number of tests provided in the "Local Tests" section further down this document**. They are obfuscated from you, because **you should try to find exhaustive tests of your own code**. This can be frustrating because you are still learning to think critically about your own code, to figure out where its weaknesses are. But have patience with yourself and make sure you read the question carefully to understand its edge cases.

Warning: Do not wait until the last minute to submit your assignment. There is a hard 5 second limit on the time each test has to complete, and 5 minutes between submissions. If the server is heavily loaded, borderline tests may start to fail. **You can submit multiple times and your best score is recorded** so there is no downside to submitting early.

The Assignment

Purpose & Goals

Generally, the assignments in this course are meant to give you the opportunity to practice with the topics of this course in a way that is challenging yet also manageable. At times you may struggle and at others it may seem more straight-forward; just remember to keep trying and practicing, and over time you will improve.

Specifically, assignment 4 aims to improve your understanding of the (Unsorted) Set interface and sorting algorithms by:

- Using the Hashtable, HashMap, sorting algorithms, and any interfaces from previous assignments to solve a variety of problems (Part A),
- Implementing some new methods in the `MyStrictHT`, and `HCPractice` classes (Part B), and
- Reflecting on your choice of data structures and algorithms (Part C).

Setup

(Alexa made a 20-minute Assignment 1 Setup video that may help you do the following steps; it is very similar to the setup for Assignment 1.)

Start by downloading and decompressing the Assignment 4 Zip File (comp2402w22a4.zip on piazza's Resources tab), which contains a skeleton of the code you need to write. You should have the files: Part0.java, Part1.java, Part2.java, Part3.java, Part4.java, StrictHT.java, MyStrictHT.java, HCPractice.java, BinaryHeap.java, DefaultComparator.java, Graph.java, and Algorithms.java.

The skeleton code in the zip file compiles fine. Here's what it looks like when you unzip, compile, and run `Part0` and `Part1` from the command line:

```
a4 % pwd
/Users/asharp/2402/a4
a4 % ls
comp2402w22a4.zip
a4 % unzip comp2402w22a4.zip
Archive:  comp2402w22a4.zip
   creating: comp2402w22a4/
  inflating: comp2402w22a3/DefaultComparator.java
     <...rest of files omitted…>
a4 % ls
comp2402w22a4        comp2402w22a4.zip
a4 % javac comp2402w22a4/*.java
a4 % ls
comp2402w22a4        comp2402w22a4.zip
a4 % java comp2402w22a4.Part0
3
50
4
13 [I hit Ctrl-D to end the input here]
3D [The D here is a remnant of the Ctrl-D]
50
4
13
Execution time: 5.986822196
a4 % java comp2402w22a4.Part1
Execution time: 2.2645E-5
```

You can also run the `StrictHT`, `MyStrictHT`, and `HCPractice` programs:

```
a4 % java comp2402w22a4.StrictHT
Adding 10 integers to StrictHT...
Iterating through 10 elements...
Iterating and removing all 10 elements...
Adding 3000000 integers to StrictHT...
finding the 3000000 elements...
done (13.058795772000002s)
a4 % java comp2402w22a4.MyStrictHT
Adding 10 integers to MyStrictHT...
Iterating through 10 elements...
Iterating and removing all 10 elements...
```

```
Adding 3000000 integers to MyStrictHT...
finding the 3000000 elements...
done (12.788394592000001s)
a4 % java -ea comp2402w22a4.MyStrictHT
Adding 10 integers to MyStrictHT...
Iterating through 10 elements...
Iterating and removing all 10 elements...
Adding 3000000 integers to MyStrictHT...
finding the 3000000 elements...
done (13.253772410000002s)
a4 % java comp2402w22a4.HCPractice
HashSet on Persons...
[(Fname1 Lname3: 2000, lucky number: 0), (Fname2 Lname4: 2000, lucky
number: 7), (Fname1 Lname3: 2000, lucky number: 0), (Fname2 Lname4:
2000, lucky number: 0)]
Done adding
a4 % java -ea comp2402w22a4.HCPractice
HashSet on Persons...
[(Fname1 Lname3: 2000, lucky number: 0), (Fname2 Lname4: 2000, lucky
number: 7), (Fname1 Lname3: 2000, lucky number: 0), (Fname2 Lname4:
2000, lucky number: 0)]
Exception in thread "main" java.lang.AssertionError
      at comp2402w22a4.HCPractice.main(HCPractice.java:53)
```

If you are having trouble running these programs, **figure this out first before attempting to do the assignment**. Check the assignment 4 FAQ on piazza and/or the aforementioned setup video, or ask a question there if you are stuck and course staff or another student will likely help you fairly quickly.

Part A: The Interface

The file `Part0.java` in the zip file actually does something. You can use its `doIt()` method as a starting point for your solutions. Compile it. Run it. Test out the various ways of inputting and outputting data. **You should not need to modify Part0.java, it is a template. Do not modify the main(String[]) methods for any PartX.java file, as they are expected to remain the same for the server tests.**

You can download some sample input and output files for each question as a zip file (a4-io.zip on piazza's Resources tab). If you find that a particular question is unclear, you can probably clarify it by looking at the sample files for that question. Part of problem solving is figuring out the exact problem you are trying to solve.

You may assume that all lines are integers under 32 bits. If you need some help with modular arithmetic, try this khan academy page. As with assignments 2 and 3, let's just use java's floorMod(x,d) to compute x mod d. This always returns a non-negative integer, which will simplify our lives.

1. [8 marks] (Assignment 3.1 Redux) Given a file where each line is a single (32-bit) integer, read in the lines of the file one at a time; take the sum (modulo 2402) of **the distinct values obtained from** the chain of `x` elements obtained by starting at the index-0 element, then using that element as the index of the next element in the chain, proceeding until you have considered `x` elements. When you use an element as an index, you should modulo it by the length of the list. As usual, `x≥0` is a parameter to the `doIt` function. For example, if `x=3` and the input is

   5 ← index 0, first element of the chain. Leads us to index 5%6=5.
   4
   8
   10
   2
   6 ← index 5, second element of the chain. Leads us to index 6%6=0.

   Then the chain of length 3 consists of the elements at indices 0, 5, then 0 again, that is **the distinct elements 5 and 6**, for a total sum of (5+6)%2402=11. If the last line had been a 5 then the output would just be 5.

   Note: Please use `Math.floorMod` instead of `%` to make everyone's lives a bit easier.

2. [10 marks] (Assignment 1.3 Redux) Given a file where each line is a single (32-bit) integer, read in the lines of the file one at a time; output the sum (modulo 2402) of all lines L with the property that the parity of (the sum of) **the distinct values obtained from** the most recent `x` lines (including line L) is even, where `x>0` is a parameter to the `doIt` method. (If there are `<x` lines, you output 0.) For example, if `x=3` and the input is

   4
   2
   2 ← the parity of distinct values from the x=3 most recent lines is even (6)
   3 ← the parity of distinct values from the x=3 most recent lines is odd (5)
   3 ← the parity of distinct values from the x=3 most recent lines is odd (5)
   5 ← the parity of distinct values from the x=3 most recent lines is even (8)

   Then the output is 2+5=7 (mod 2402), which is 7. If there were an additional 1 at the end of the file, the parity of the distinct values would be odd (9) and we would still output 7.

   Note that `parity(i)`=0 if `i` is even, 1 if `i` is odd. Intuitively, it indicates whether an integer `i` is even or odd.

   Note: Please use `Math.floorMod` instead of `%` to make everyone's lives a bit easier.

3. [10 marks] Given a file where each line is a single (32-bit) integer, read in all lines of the file one at a time; output the length of the maximum chain of consecutive multiples of `x` found in the file (not necessarily in order in the file), where `x>0` is a parameter to the `doIt` function. For example, if `x=3` and the input is

   ```
   15
   12
   -3
   4
   1
   0
   6
   3
   ```

   Then the longest chain of consecutive multiples of 3 is -3, 0, 3, 6 (of length 4). Note that if there were another line with a 9 at the end of the file, the longest chain would become -3, 0, 3, 6, 9, 12, 15 (of length 7).

   Note: you shouldn't need to use `Math.floorMod` anywhere in this problem :-)

4. [10 marks] Given a file where each line (other than the first) is a **distinct** single (32-bit) integer, and the first line gives you `n>0`, the number of remaining lines in the file, read in the `n` (last) lines of the file one at a time and compute the sum (modulo 2402) of each element times its position in sorted order. For example, if the input is

   5 ← number of remaining lines in the file, `n`; not part of the sorted elts
   8
   5
   3
   2
   6

   then our output is 2*0 + 3*1 + 5*2 + 6*3 + 8*4 = 63. If the last line were a 10 then the output would instead be 2*0 + 3*1 + 5*2 + 8*3 + 10*4 = 77.

   Note: Please use `Math.floorMod` instead of `%` to make everyone's lives a bit easier. See the note just before Part 1's description.

   Hint: This is the first problem where getting the best asymptotic time/space complexity is not going to be enough; you want to optimize these as much as possible. For this problem, when in doubt, prioritize space over time.

Part B: The Implementation

The (provided) `StrictHT` and `MyStrictHT` classes are implementations of the Set interface via a hashtable that has a strict/constant number of buckets (as opposed to our usual kind that adds more buckets as the load of the hashtable hits a certain threshold.) `StrictHT` is the basic implementation that you do not and should not modify. `MyStrictHT` is a child class of `StrictHT` that you can and should modify in order to better implement the Set interface (with regards to time). More details below in question 5. The `Person` class (within the `HCPractice` class) is an (incomplete) implementation of the `Person` class, in that it will not allow us to correctly create a Set of `Person`s without duplicates. Your task is to complete its implementation so that we can create such a Set. More details below in question 6. Default implementations are provided for both questions 5 and 6, but they will not pass the server tests.

5. [14 marks] In the `MyStrictHT` class, make changes to any of the `StrictHT` non-final methods (**by overriding them in `MyStrictHT`**) as you see fit in order to pass the (time) performance server tests. `MyStrictHT` is a child class of `StrictHT`, so it will compile and run and be correct right out of the gate; the default implementation, however, will not pass the server performance tests because it is too slow. Your task is to determine what makes the methods so slow in this "strict" setting, and make improvements.

   Note that some inherited methods and variables are final so that they cannot be changed; this is on purpose to try to prevent you from taking certain shortcuts I don't want you to take. Anything that is not final can be overridden in `MyStrictHT` and implemented as you see fit. **Do not make changes to `StrictHT`**, as a local copy (not yours) will be used on the server.

   Also note that the main methods in `StrictHT` and `MyStrictHT` are the same, and they start off by taking ~10s each (on Prof Alexa's local machine.) The fastest implementation I have can run these tests in <2s, for reference, and a ~5s solution will pass most, if not all, server tests.

   This is a more open-ended question than we're used to; this is intentional! It's meant to give you flexibility to apply one of many possible solutions, so try to relax and think critically about the existing code and how you can improve upon it.

   If it helps, I will only test this on types T that are Comparable (e.g. Integers, Strings).

   If you want to run `MyStrictHT.main` with the assertions, use the `-ea` parameter, e.g.

   ```
   % java -ea comp2402w22a4.MyStrictHT
   ```

6. [13 marks] In the `Person` class (within the `HCPractice` class), implement the `equals` and `hashCode` methods so that we can correctly store a Set of `Persons` without duplicates, that is, we want to be able to add, remove, and find distinct Persons. Each Person has 4 characteristics:

    `fname` – first name – we assume this is constant over the course of the program
    `lname` – last name – we assume this is constant over the course of the program
    `birthYear` – year of birth – we assume this is constant (!!)
    `luckyNumber` – a person's lucky number, if they even have one. This can change
    over the course of the program

    There are some basic tests in the `HCPractice.main` that you should examine and play around with. If you want to run `HCPractice.main` with the assertions, use the -ea parameter, e.g.

    ```
    % java -ea comp2402w22a4.HCPractice
    ```

Part C: The Debrief

7. [5 marks] Do the assignment 4 debrief (multiple-choice questions) on brightspace after the programming deadline has passed (and the solutions are available). The debrief questions are due 8 days after the assignment deadline (in this case, by Friday April 1st, 2:00pm.) No lates accepted.
    The debrief questions are meant to encourage retrospection on what you were meant to learn from the assignment. If you were stuck on a problem, this is an opportunity to look at the solutions and at the assignment 4 debrief, to figure out what went wrong for you, and to still learn what you were meant to learn.
    It is also an opportunity to provide feedback to Prof Alexa on certain aspects of the course so far.

Local Tests

For Parts 1, 2, 3, and 4, you can download some sample input and output files for each question as a zip file (a4-io.zip, on the Resources tab on piazza). Once you get a sense for how to test your code with these input files, write your own tests that test your program more thoroughly (**the provided tests are not exhaustive!**)

For Parts 5 and 6 (`StrictHT`, `MyStrictHT` and `HCPractice`), the main method of each file provides some tests you can and should add / modify as you go along. You can use the -ea command-line parameter (see the comments in the main methods themselves) to "turn on" the assertions to help you test.

Server Tests

See the “Submitting and Server Tests” section further up this document.

Tips, Tricks, and FAQs

For the most up-to-date Assignment-specific FAQ, please see the Assignment 4 FAQ post on piazza (and/or the a4 filter.)

Regarding tips and tricks, please see the Problem Solving & Assignment Tips document (which may be enhanced as the semester progresses, so check back frequently!)