Assignment 2: The End of Act One

Pre-Assignment Notes

The **late penalty** is -10% per day late. After two days, late submissions won't be accepted.

This is an **individual assignment**, meaning you're not supposed to share or copy your solution/code from other students or the internet. There's still the class's Piazza if you want to ask questions to the group, or you can ask during lecture, send an email to the class's help account, or attend an office hour on Discord.

Note you can use any code shared by the instructor or anything from class or the tutorial videos linked on the course website. You *can* get outside help from guides and other people, just try to make sure they're teaching you how to solve the assignment, not writing the code for you. You won't learn anything that way, and they can't write your exam or midterm for you. If in doubt about whether something is plagiarism or not, don't be afraid to reach out and ask!

1. Design Decisions

Now that we have covered the basics of data structures, algorithms, and analysis, it's time to put those skills to the test. The next section outlines a number of scenarios you could face as a programmer where you will need to make a code design decision. For each of these scenarios, you'll be asked to write some code that implements your decision, as well as give a written explanation for your choice. You may be asked to perform an asymptotic analysis of your solution as part of that explanation.

For this assignment, you'll be submitting both your code as well as a written answer for each question. The code can be bundled together into a single project, although the code for each question should be stored in a separate Java file. There can be one main file with a main method as the test class for the entire assignment that prints its output to the terminal, but which sections of the main method correspond to which questions should be clearly marked in both the code and the output.

These written answers should be stored in a single pdf document, and should clearly label which written responses correlate to which questions. Each question lays out the different requirements for their written portion, but in general, your written answers should be between 1-3 paragraphs in length.

You may use any classes, functions, techniques, etc, that have been introduced or referenced in class so far. You should avoid using more advanced built-in classes or techniques that we haven't covered yet, though as we have not comprehensively covered Java in class that still leaves a lot of more basic material uncovered. If you're unsure about using anything (particularly if it just seems to short-circuit a question in a single line!) try running it past myself or a TA in an office hour.

2. Assignment Outline

This assignment presents a set of different scenarios. Each one is a separate question, not connected to the others, and should have separate code and written answers.

Each scenario also includes a narrative element, which means that interpreting the question is part of the challenge. Recognizing the right data structure, algorithm, or technique that applies is not the whole of the problem, though, and a well-implemented but sub-optimal solution will still be worth marks.

2.1 She Blinded Me With Library Science

You're working on a data storage system for a library to keep track of the books on its shelves. The library has a fixed amount of shelf space to work with (for convenience here, we're going to assume all books take up the same amount of shelf space), so they don't expect their collection to expand, though old books may be swapped out for new ones from time to time.

The biggest concern is finding books in the system and flagging them as either in the library or out on loan. Every day, the library's system must handle many transactions of looking up a book and flagging it as either in or out. Since people have to wait at the counter for the system to finish processing and the old computers available to the library aren't the fastest, time efficiency during these operations is a priority.

You've been assigned to work on the data structure and functions that will support this library system. Each book is stored with a name (there can be duplicates), an author's name, a boolean that flags the book as either out or in, and a unique ID integer. These ID integers are tied to spaces on the library's shelves and start at 0, so they can also be used to retrieve and return a book when someone checking it out at the counter requests it.

The available functions should include checking out, returning, checking the status of, and getting the name and title of a book based on its ID. It should also be possible to remove an old book from a given shelf spot, put a new book into an empty spot, or swap an old book for a new book, all also based on the ID. Finally, it should be possible to check how many books the library owns, currently has in, and currently has lent out. A reasonable amount of error-handling should be expected throughout, though no specific instructions are given here.

You must submit one Java file that provides this functionality, though it may include more than one class. The test class must demonstrate all of the desired functions, though you're free to define and load as many test books into your library system as you believe the tests need.

Your written answer for this question must explain why your choice of data structure matches the needs of this scenario. You should include an asymptotic run-time analysis for each of your functions, and explain how these match the needs of the library.

2.2 Double-Back To The Future

You've just joined a software team for a company that makes a diary program. The diary program lets the user write a new entry at the end of each day, filing it into the diary program's archive. This archive is organized as a stack, where new entries are pushed on top, so the entries are ordered from most recent to oldest.

You've been asked to help build a new calendar feature, that's essentially just the diary feature in reverse – now, the user can write a new entry for a day to come, which will be added to a queue of days ordered from the oldest-added entry (the nearest day) to the most recently-added entry (the furthest day). Your supervisor likes the symmetry of old calendar entries leaving the queue to become new diary entries. There's just one problem: the original designers of the software didn't include support for queues in the program's database, only stacks.

As a work-around, you've been asked to build your own version of a queue class that is built on top of two stacks. You should use the built-in Java Stack class (set to take Strings) to represent these stacks, but your stack-based queue does not need to officially implement a queue interface or extend any existing class, so long as it correctly implements the methods from a Queue's ADT.

You must submit one Java file with one class, the stack-based queue. The test file must test all of the standard functions of a queue, proving that it works as a queue is expected to. You may assume the diary entries and calendar entries are both simple Strings, and may make up whatever dummy test

values for them you want. You need not show your stack-based queue passing hypothetical calendar entries back to a diary-storing stack, unless of course you feel like it.

In your written answer, walk through how your stack uses the two stacks to implement each of a queue's functions, and include an asymptotic run-time analysis for each of these stack-based queue functions. If any of the run-times have changed, be sure to note this and explain why.

2.3 Reversing Course

You're on a slightly dysfunctional software team, with two teammates who constantly argue about the best way of doing things. This arguing has led to a miscommunication about which way a big pile of input data should be read, the result of which is that now you have a huge, singly-linked list of Strings in reverse order.

It's your job to write a function that can reverse the order of a singly-linked list, but your teammates are still arguing about which approach your function should take. Alex believes you should take a straightforward, iterative approach, while Briar believes your solution should be recursive. To make peace with your teammates, you must present two functions for reversing a singly-linked list, one recursive and one iterative, and determine which one is better.

You're primarily concerned about the run-time efficiency of your solution, although memory-space efficiency could be used as a tie-breaker. Be sure to include error and exception handling to keep your teammates from seizing on that. You must also submit your best-effort for both approaches, as your teammates will be able to tell if you deliberately sabotage one function to make picking the competitor easier.

You must submit one Java file, though it may include multiple classes, particularly to support building the singly-linked list of String-bearing nodes. The test file may make a list of any length with any dummy String data inside the nodes you want, though a longer list or multiple lists may help demonstrate your point about how each function performs. Both functions must be proven to actually reverse the list, and the nodes in the final reversed list should be the same ones as in the original list (not just copies of the same content, but the same instanced objects).

Your written answer to this question should explain which teammate's approach is the better choice for this problem, based on the two functions you wrote. Include an asymptotic run-time analysis for both, and use this as part of your argument. Aside from time and possibly space efficiency, you're allowed to justify your argument however you like – be persuasive!

2.4 Order Up!

A restaurant wants a system where their serving staff can bring orders for dishes back to the kitchen. Each dish is represented by a particular integer, and the system would store a sequence of orders for the cooks to complete from start to finish. Normally, servers would put new dishes at the back of this sequence, while cooks would read from the front, but occasionally a server would need to place a rush order at the front of the sequence so that it will be the next one the cooks complete.

You'll be in charge of writing a class that stores this sequence of integers in a data structure, along with the functions that let the servers add new dishes to the front or back and cooks to take them off of the front. You're not in charge of the user interfaces that the cooks and the servers will use to access your system, unfortunately, and your teammates in charge of each interface made different assumptions about what your structure will look like.

Your teammate who's working on the user interface for the servers wants to interact with a Deque. That way, they can give the servers a default option for adding new orders to the back of the

sequence, and another for adding them to the front if they're a priority order. As such, their interface code is expecting to see the functions of a Deque.

Your other teammate who's working on the user interface for the cooks, however, was expecting a simple Queue. They've already written their code for dequeing from the front of a queue, and are expecting to see that function.

Neither of them want to change their existing design, so it's up to you to build an object that can be called on as both a Deque and a Queue. To simplify the problem a little, you are not required to make your code compatible with a particular Java Queue or Deque class or interface, you can make your own version of either/both in the process of solving this question, just so long as each adheres to the expected methods for their respective ADT.

You must submit one Java file, though it may include more than one class. The test file must demonstrate the same object being treated as both a Queue popping numbers off the front and as a Deque adding numbers to both the front and back. The test should interleave these function calls to prove that they are accessing and modifying the same base object.

Your written answer for this question should include an asymptotic run-time analysis for each function. If the worst-case run time for any of your functions is different than it would be for that data structure's usual methods under its ADT, you should note it and explain why. Otherwise, your answer should walk through how your code addresses the needs of the scenario.

3. Deliverables

All assignment submissions will be done through CourSys at https://courses.cs.sfu.ca/. For this assignment, that should include:

1. A .zip archive of your code, organized into a single Java project. To ensure compatibility, it's recommended to use IntelliJ while developing your code, then using the export function under File -> Export -> Project To .zip File.

2. A pdf of your written answers to the assignment questions. The pdf can be exported from any word processor you like, just be mindful that you organize your answers so they're easy for the TAs to read.

Be sure you tidy up your code before submitting! Check the Java code style guide posted on the course website, and do your best to provide helpful comments.