CMPSC 101
Data Abstraction
Fall 2020

Exam Two Study Guide
Exam to be Released: Friday, December 4, 2019
Exam Due: Monday, December 7, 2019 at 11:59 pm

Introduction

The exam will be "open notes" and "open book" and it will cover the following materials. Please review the "Course Schedule" on the Web site for the course to see the content and slides that we have covered this semester. Students may bring questions about this material during our review session on Monday, November 30th.

- Chapter One in DSAAJ, all sections (i.e., "Java Primer")
- Chapter Two in DSAAJ, all sections (i.e., "Object-Oriented Design")
- Chapter Three in DSAAJ, all sections (i.e., "Fundamental Data Structures")
- Chapter Four in DSAAJ, skipping Section 4.4 (i.e., "Algorithm Analysis")
- Chapter Five in DSAAJ, skipping Sections 5.5 and 5.6 (i.e., "Recursion")
- Chapter Seven in DSAAJ, skipping Sections 7.3, 7.5, 7.6, and 7.7 (i.e., "Lists and Iterators")
- Chapter Ten in DSAAJ, skipping Sections 10.3, 10.4 and 10.5 (i.e., "HashTables")
- Using the commands in the terminal window (e.g., `cd` and `ls`); building and running Java programs with Gradle; knowledge of the basic commands for using Docker and GitHub
- The class discussion slides available from the course web site
- Source code and writing for all of the assigned laboratory and practical assignments

The exam will be a mix of questions that have a form such as fill in the blank, short answer, true/false, and completion. The emphasis will be on the following technical topics:

- Fundamental concepts in computing and the Java language (e.g., definitions and background)
- Practical laboratory techniques (e.g., editing, building, and running programs; effectively using files and directories; correctly using GitHub through the command-line `git` program)
- Understanding Java programs (e.g., given a short, perhaps even one line, source code segment written in Java, understand what it does and be able to precisely describe its output).
- Knowledge of worst-case time complexities, expressed in the "big-Oh" notation, for all of the algorithms provided by the studied data structures (e.g., `removeLast` for a node-based list).
- Composing Java statements and programs, given a description of what should be done. Students should be completely comfortable writing short source code statements that are in nearly-correct form as Java code. While your program may contain small syntactic errors, it is not acceptable to "make up" features of the Java programming language that do not exist in the language itself—so, please do not call a "`solveQuestionThree()`" method!

No partial credit will be given for questions that are true/false, completion, or fill in the blank. Some partial credit may be awarded for the questions that require a student to give a short answer or write source code. You are invited to furnish short, precise, and correct responses to all of the

questions. When you are taking the exam, you should do so as a "point maximizer" who first responds to the questions that you are most likely to answer correctly for full points. Finally, students may only reschedule the exam for a different date or time if they are facing documented extenuating circumstances that prevent them from attending the scheduled time slot. Please see the instructor if you have questions about any of these policies.

Reminder Concerning the Honor Code

Students are required to fully adhere to the Honor Code during the completion of this exam. More details about the Allegheny College Honor Code are provided on the syllabus. Students are strongly encouraged to carefully review the full statement of the Honor Code before taking this exam. If you do not understand Allegheny College's Honor Code, please schedule a meeting with the course instructor. The following is a review of Honor Code statement from the course syllabus:

> The Academic Honor Program that governs the entire academic program at Allegheny College is described in the Allegheny Academic Bulletin. The Honor Program applies to all work that is submitted for academic credit or to meet non-credit requirements for graduation at Allegheny College. This includes all work assigned for this class (e.g., examinations, laboratory assignments, and the final project). All students who have enrolled in the College will work under the Honor Program. Each student who has matriculated at the College has acknowledged the following pledge:
>
> I hereby recognize and pledge to fulfill my responsibilities, as defined in the Honor Code, and to maintain the integrity of both myself and the College community as a whole.

Detailed Review of Content

The listing of topics in the following subsections is not exhaustive; rather, it serves to illustrate the types of concepts that students should study as they prepare for the exam. Please see the instructor during office hours if you have questions about any of the content listed in this section.

Chapter One

- Basic syntax and semantics of the Java programming language
- Input(s) and output(s) of the Java compiler and virtual machine
- How to use "`gradle`" to build, run, and test a Java program
- The base types available for primitive variables in the Java language
- How to create and use classes and objects in the Java language
- How to define and call methods and constructors of a Java class
- The declaration and use of `String`, `StringBuffer`, and arrays
- Type conversion operators and control flow and iteration constructs in Java
- Software engineering principles as expressed in Java (e.g., packages and JUnit tests)
- How to write effective test cases in JUnit for various data structures and algorithms
- How to use the JavaDoc standard to write informative comments in Java programs
- An understanding of the Java source code supporting a program that performs input and output with the console (e.g., `System.out.println`) and files (e.g., `java.util.Scanner`)

Chapter Two

- The goals, principles, and patterns of object-oriented design in the Java language
- An understanding of the principles known as abstraction, encapsulation, and modularity
- How to use inheritance hierarchies to create an "is a" relationship between Java classes
- The meaning and purpose of abstract classes and interfaces in the Java programming language
- How to create, catch, and handle exceptions thrown in a Java program
- How to performing casting to convert a variable from one data type to another
- The ways in which generics promote the implementation of reusable Java programs
- Knowledge of the Java syntax needed to declare data structures that are generically typed

Chapter Three

- How to use arrays to store primitive and reference variables
- The algorithms for sorting arrays into ascending and descending order
- How to use psuedo random number generators in Java programs
- The similarities and differences between one- and two-dimensional arrays
- The meaning and purpose of techniques for cloning data structures
- The similarities and differences between "deep" and "shallow" copies of arrays
- How to use the Java methods to construct "deep" and "shallow" copies of data structures
- An understanding of the types of nodes in a `SinglyLinkedList` and `DoublyLinkedList`
- The benefits and trade-offs associated with the `SinglyLinkedList` and `DoublyLinkedList`
- Knowledge of the worst-case time complexity for all methods of node-based structures
- Why the `removeLast` implementation is inefficient when implemented in a `SinglyLinkedList`
- The meaning of the term "equivalence testing" and how it connects to data structures
- The "equivalence relationship" that must be upheld by, for instance, a `DoublyLinkedList`
- The trade-offs associated with using either arrays or linked lists to implement data structures
- The benefits that come from using Java's generics to implement node-based list structures

Chapter Four

- A strategy for timing the implementation of an algorithm in the Java programming language
- The challenges associated with experimentally studying an algorithm's performance
- A comprehensive understanding of why to use a doubling experiment to study efficiency
- An understanding of all the steps and source code needed to conduct a doubling experiment

- The challenges associated with conducting an experimental evaluation of an algorithm
- The meaning and purpose of the terms "basic operation" and "psuedo code"
- An intuitive understanding of best-, worst-, and average-case analytical evaluations
- The relationship between an order-of-growth ratio and the worst-case time complexity
- The meaning of the "Big-Oh" notation used as part of the analytical evaluation of algorithms
- Knowledge of the patterns in psuedo code that suggest certain worst-case time complexities
- How to intuitively find the worst-case time complexity of an algorithm using psuedo code

Chapter Five

- How to use a recursive approach to solving a problem (e.g., the base and recursive cases)
- Knowledge of recursive algorithms for arithmetic computation (e.g., factorial and Fibonacci)
- How to determine and justify the worst-case time complexity for the recursive factorial method

Chapter Seven

- A fundamental understanding of the meaning and purposes of a list abstract data type
- An approach to implementing and testing an array-based version of the list data structure
- The benefits that come from using Java's generics to implement array-based list structures
- The four steps that a dynamic array must perform when the internal fixed-sized array is full
- The trade-offs associated with using either arrays or linked lists to implement data structures
- Knowledge of the worst-case time complexities for the methods of the dynamic array
- Understanding of how to use an `Iterator` in a program that creates and manipulates lists

Chapter Ten

- An understanding of how the `HashMap` stores and retrieves key-value pairs
- Knowledge of how to use the `HashMap`'s methods to count the frequency of words in a text file
- The expected and worst-case time complexities of the key methods provided by the `HashMap`