Computer Science 205
Review Sheet for Comprehensive Final Exam
Monday, December 12th, 2:00 PM - 5:00 PM, CSB 100

Our final exam will be comprehensive. The main topics to be covered are strings, sorting, searching, multidimensional arrays, object-oriented programming concepts (inheritance, containment, polymorphism), graphical user interfaces, linked lists, stacks, queues, trees, and recursion.

Look back over all old quizzes and end of chapter reviews for some good review questions. It will be worth 200 points. You may bring in a one page reference sheet.

Arrays, ArrayLists, & Strings

- Advantages and disadvantages of arrays and arraylists
- Declaring an array variable of any dimension with or without initial assignment
- Filling up, printing out, and summing up a multi-dimensional array
- Writing the code to determine the maximum or minimum value in an array
- Passing multi-dimensional arrays as parameters
- Be able to write and call a method which utilizes an array as a parameter
- How multi-dimensional arrays are actually stored in main memory; How are column-major order and row-major order different?
- Using the `String` ADT; Be comfortable using the `indexOf`, `substring`, `compareTo`, and `equals` operations with string objects
- Using the `ArrayList` ADT; Be able to distinguish `size` and `capacity`; How is this a "self-adjusting" data structure?
- Be able to use the `Object` data type

Sorting and Searching Arrays

- Be familiar with each of the algorithms for sorting and searching Be able to work out by hand how each one sorts a group of numbers. Don't worry about having to write the code for these algorithms.
  - Bubble Sort
  - Selection Sort
  - Linear (Sequential) Search
  - Binary Search
- Be able to show what an unsorted array looks like after each pass through the outer loop of a selection sort.
- Be able to determine the number of checks it takes to find a particular item using a linear or binary search. That is, the number of slots whose contents are accessed in looking for an item.

Run-Time and the *Big-O* Notation

- Why is run-time important to know?
- Be able to give the run-times using *Big-O* Notation for the four sorting and searching algorithms listed above
- Know how the different run-times compare to each other in terms of what happens when the number of items being processed increases; How would the graphs of linear, quadratic, cubic, and logarithmic run-times compare to one another?
- Given a loop be able to express the run-time of that loop using the *Big-O* Notation

Object-Oriented Programming Relationships

- Be able to name and define the three types of relationships that can exist between objects. How do you add inheritance or containment to a class? How do you know whether to use inheritance and whether to use containment?
- Definition of base class and derived class. Be able to draw an inheritance hierarchy diagram. What are the properties of the `Object` superclass, and what methods are defined in it?
- Definition of overriding. Be able to set up the constructor for a derived class and use the `super` keyword.
- Differences between `private`, `protected`, `public`, and no access modifiers. Be comfortable with questions on worksheet.
- Definition of polymorphism. When does it occur? When does a `ClassCastException` occur? What types of polymorphism errors are caught by the compiler? Be able to identify a run-time error and a compiler error involving polymorphism.
- Differences between abstract and final classes. How are both set up?
- Role of `instanceof` operator

Graphical User Interfaces

- What are interfaces and where are they used?
- Be able to set up a listener for a single component or multiple components. If we connect two or more buttons to a single listener object, name two ways we can determine which button is involved.
- Role of an inner class? Can an object of an inner class type be instantiated?
- Be able to name the five layout manager classes. What is the default layout manager for a window? Be able to write a call to `setLayout` that specifies a particular layout.
- What is a panel? What is its default layout manager?

Recursion

- What is recursion and what distinguishes it from iteration?
- Advantages and disadvantages of recursion compared to iteration
- Determining the base case and recursive case of a method
- Tracing through a recursive method (You should show all method calls in a recursive trace on the exam.)
- Writing a method both iteratively and recursively to find a factorial or a summation. The base case of a summation will always be a value of n less than the start value which always returns a value of 0.

Tokenizing

- Definition of a token and delimiter.
- Be able to use the `Scanner` class with the keyboard as well as external files to process different token values; Know how to advance to the next token; What happens if a number and text are glued together (e.g. `5th`)?

Linked Lists

- What is a linked list? What items is it always made up of?
- What two different ways can a linked list be implemented? When might one way be better than another? Be able to trace through code and draw the linked list represented both ways.
- Be able to print out and count the number of items in a linked list
- Be able to insert new items into a linked list
- Be able to delete items from a linked list
- Be able to compare and contrast arrays and linked lists
- The role of a copy-constructor; the dangers of not including it within a class that manipulates dynamic data; the differences between a deep copy and a shallow copy
- What is a "memory leak"? How is garbage collection of memory handled in Java?
- What is an iterator? Be able to use the `LinkedList` class and the `ListItr` class from the Java Collections Framework.

Stacks and Queues

- Definition of both and what distinguishes them
- LIFO vs. FIFO
- Be able to trace through a program which uses the methods of a stack or queue class
- Be able to write class methods for a driver program which needs to use the methods of a stack or queue class

- What are the two main ways a stack class can be implemented?
- Be able to write instance methods for a linked list implementation of a stack (e.g, `findMax` method from Lab on Stacks)
- Using the `Stack` class in the Java Collections Framework; What packages do you need to include? What is different from our own stack class?

Binary Search Trees

- What is a binary tree? What is a binary search tree?
- Tree terminology: root node, leaf node, left child, right child, parent, sibling, height, subtree, balanced, complete, full
- Inserting items into a binary search tree; Why do we want the initial data to come in randomly?
- Be able to give the inorder, preorder, and postorder print traversals of a given binary search tree
- Be able to show what a binary search tree looks like after deleting a given node (Know how to handle all three scenarios discussed in class)
- Be able to show the array that could be used to store a binary search tree if the user is using the array-based implementation of a binary search tree (See class handout)
- Be familiar with the declarations used to set up a pointer-based binary search tree and the insert method used to add an item to a tree using pointers
- Using the `TreeSet` and `TreeMap` classes from the Java Collections Framework; Be sure you are comfortable with using iterators with these classes as well as performing deletions
- Binary tree properties.
  1. The maximum number of nodes in a full binary tree of height h is $2^{h+1} - 1$.
  2. In a binary tree of size n, there are $2n$ available links, $n - 1$ total edges, and $n + 1$ total null links.
  3. The number of leaves in a non-empty full binary tree is one more than the number of internal nodes.
- Be comfortable with the `BinarySearchTree` class; Why is `KeyedItem` specified as a separate class?
- Be able to write recursive methods for traversals or finding the maximum, minimum, total number of nodes, total number of leaves, etc.

Threads

- Definition of a thread and how to use them; Role of `start()` and `run` methods
- Different aspects of the thread life cycle