Outline of Skills and Concepts

Revision 3, 5.2.14

This is an outline of the specific skills and concepts I will be looking for in your work. These topics will appear both in tests and quizzes, in your homeworks, classworks, and projects. I will check in both contexts for your ability to correctly use each of the topics.

1. Programming Constructs
    A. Declaring and assigning variables
        1. Know the different primitive data types and when to use them.
        2. Know how to declare and assign values to variables with primitive data types
        3. Know the difference between a variable declared as a primitive type and a variable declared as a reference to an object.
        4. Know how to instantiate an object using its constructor and assign the instance to a variable.
        5. Know what it means that Strings are immutable.

    B. Arithmetic
        1. Know how to perform simple mathematical calculations.
        2. Know what the % operator does.
            1. Be able to use it as a test for divisibility
            2. Be able to use it to get an individual digit from an int.
        3. Know how to cast numbers.
        4. Know common bug with integer division.
        5. Know commonly used methods in the Math library (sqrt, min, max, random)

    C. Displaying things
        1. Know the difference between print and println
        2. Know what escape characters are. Specifically \n, \t, \", and \\.
        3. Know how to mix String literals and variables in a print or println statement.

    D. If-statements
        1. Know the primitive comparison operators: <, >, <=, >=, ==, !=
        2. Know how to compare Strings and other objects (how is this different from numerical comparisons?)
        4. Know how to form compound test conditions using && and || and !
        5. Know how to trace more complex nested if, if-else, if-else if-...-else if-else constructions.
        6. Know De Morgan's law as a way to simplify / interpret more complex conditionals.
        7. Know the problem with comparing doubles or floats with == and what you can do instead.

    E. Looping
        1. For loops
            1. Know the four parts of a for-loop.
            2. Know *exactly* what order the four parts execute in.
            3. Use how to use for-loop to loop a fixed number of times.
            4. Use a for loop to loop through an array and ArrayList
            5. Know what the ArrayIndexOutOfBounds exception means
            6. Use a nested for loop to display all ordered pairs (x, y) with x,y integers between 0 and 100.
            7. Use a nested for loop to loop through a 2d array.

        2. While loops
            1. Know how to create a basic while loop with a single test condition
            2. Know how to write a test condition which tests the value of a variable.
            3. Know how to write a test condition which calls a method on an object.
            4. Know how to create a while loop with multiple test conditions.
            5. Know how to use a counter in a while-loop.

        3. For each loop
            1. Know how to write a basic for-each loop to loop through a list.
            2. Know when the possible ConcurrentModificationException bug will occur.

4. Be able to re-write a for loop as an equivalent while loop or for-each loop and vice-versa.
5. Know when to use a for loop, a for-each loop or a while loop.

F. Methods:
1. Know all the parts of a method signature.
2. Know how to create a method with no arguments or return values.
3. Know how to create a method with arguments.
4. Know how to create a method with a return-value.
5. Be able to identify repeated code that you could make into a method.
6. Be able to identify situations when a calculation could use a method with a return value.
7. Know the difference between static and non-static methods.
   Be able to name situations when you would want to use each type.
8. Understand the idea of scope and local variables.

G. Using Objects
1. Know how to create an object by calling its constructor.
2. Know how to call a method on an object.
3. How how to reference an object's public instance variables.
   (How is this different than calling a method?)
4. Know how to call a static method and how this is different than calling a non-static method.

H. Classes
1. Know how to declare a new class.
2. Know the difference between class variables and local variables inside your class's methods.
3. Know what a constructor is, what it does, how it's different than other methods.
4. Know about method overloading.
5. Know about constructor overloading and when/why you might want to do that.
6. Know when/why you would want to use static vs Non-static methods in a class you're making.
7. Know about the toString() method, what it does, when/how to use it.
8. Know about the equals() method, what it does, when/how to use it.

I. Strings
0. Know that they're immutable, what that means, and some practical consequences.
1. Know the following String methods: .indexOf() (both versions), .charAt(), .length(), .split(), .toUpperCase(), .toLowerCase(), .replaceAll(), .substring()
1. Know how to do each of the following with a String:
   1. Loop through each letter and do something with it. (like display it).
   2. Determine its length
   3. Split it on some character (such as space).
   4. Extract a sub-string

J. 1d arrays
1. Have a “mental model” for an array
2. Know how to do the following:
   1. Declare/ instantiate an array.
   2. Put things in it.
   3. Get elements from the array.
   4. Loop through it and do something to each element.
   5. Loop through it and put something into each element.
   6. Find the largest value and smallest value in an array.
   7. Calculate the average of an array of numbers.
   8. Insert a value into an array (sliding other elements down)
   9. Remove a value from an array (sliding other elements up)
3. Know how to interpret the ArrayIndexOutOfBounds exception.

K. ArrayLists
1. Have a mental model for an ArrayList
2. Compare and contrast an ArrayList and an array

3. Know the difference between int and Integer, between double and Double, etc.
4. Know how to do the following with an ArrayList:
    1. Declare/create an ArrayList
    2. Put an item in the ArrayList
    3. Remove an item from the ArrayList
    4. Get the size of the ArrayList
    5. Loop through all elements of the ArrayList and do something to each.
    6. Check if the ArrayList contains a certain value
    7. Know how to set an element, insert an element, and remove and element.

L. HashMaps
1. Have a "mental model" for a HashMap (a dictionary/phonebook/lookup-table)
2. Know when to use a HashMap and when to use a a list.
3. Compare and contrast HashMap and ArrayList.
4. Know how to be able to do the following with HashMaps:
    1. Declare/create a HashMap
    2. Add something to it
    3. Remove something
    4. Look-up a value
    5. Check if the HashMap has a certain key.
    6. Check if the HashMap has a certain value.

M. 2d arrays
1. Know how to declare/create a 2d array.
2. Know how to assign values.
3. Know how to get the dimensions of a 2d array using .length
4. Know how to loop through a 2d array, first by rows then by columns.
5. Know how to loop through a 2d array, first by columns then by rows.
6. Know how to fetch a space above, below, left, and right of a given square.
7. Know how to loop over all array values within k spaces of a given element.
8. Know how find the sum of values in a particular column or row.

N. I/O
1. Use a Scanner object to:
    1. Read in numbers from the user.
    2. Read in strings from the user. (Know what a delimiter is, and how to set it).
    3. Read in lines of text from a text file.
    4. Write lines of text to a text file.
2. Know how to use JOptionPane and the .parse() method to get user input.

O. Recursion
1. Know what recursion is.
2. Be able to write a recursive method.
3. Know what a base case is and why it's important.
4. Be able to trace a recursive method call.
5. Know what tail recursion is and be able to recognize it.
6. Be able to write recursive methods for various problems, including binary search.

P. Algorithm time analysis
1. Know how to determine the Big-O of an algorithm.
2. Understand what, precisely, this does and does not mean.

2. Class Design / Inheritence
A. Know what the following terms mean: parent class, child class, super class, sub class, **extends** keyword, inherits, super constructor
B. Know what overriding a method is. (How is it different from overloading?)

C. Know how to extend a class and when/why you would do this.
D. Know how to decompose a problem into classes, define relationships and responsibilities of those classes.
E. Understand the idea of data abstraction and encapsulation.
F. Abstract classes
    1. Know what an abstract class is and when you should create one.
    2. Know what abstract methods are and when/how to use them.
    3. Know that you cannot instantiate an abstract class.

3. Interfaces
    A. Know what an Interface is, how to declare one, and why you might want to.
    B. Know how to implement an Interface.
    C. Know the parts of the Comparable Interface. Be able to implement it in a class of your own creation.
        1. Know the method signature for the compareTo method and what the return values mean.
        2. Know how to use the compareTo method of an object in a realistic scenario.
        3. Know how to use the compareTo of another object in the implementation of another compareTo method. (For example: to compare Student objects by last name)
    D. Be able to discuss when you should use an Interface and when you should use an Abstract Class.

4. Searching and Sorting
    A. Know what sequential search is and be able to implement it. Know it's time complexity.
    B. Know what binary search is, be able to implement it or trace someone else's implementation, and know it's time complexity.
    C. Know how to recognize selection sort. Be able to implement it or trace someone's implementation of it.
    D. Know how to recognize insertion sort. Be able to implement it or trace someone's implementation of it.
    E. Know how to recognize mergesort. Be able to implement it or trace someone's implementation of it.
    F. Be able to compare/contrast the O() of each of these search algorithms in the best and worst cases.