**CSCI 161 Introduction to Computer Science**

# Review Guide 2

## Topics

- Exam I stuff, plus...
- Loops
  - Understand the difference between a counter-controlled condition and a sentinel condition, and which loop is usually appropriate.
  - Be able to trace a given while-loop, for-loop, or do-while loop, and determine the outcome.
  - Be able to write a loop (possibly nesting other loops) to solve a given problem.
  - (Good practice to review the code to determine if a number is prime.)
- Collections
  - Without being prompted, understand when to model problems using 1D arrays, 2D arrays, `ArrayLists`, or `HashMaps`.
  - Be familiar with the methods provided for manipulating `ArrayLists` and `HashMaps`.
  - Be ready to write methods using loops to work with 1D and 2D arrays.
  - Be able to read and write code involving the use of `ArrayLists` and `HashMaps`.
- User Input and Strings
  - Given the API, know how to manipulate Strings for various purposes.
  - Know how to parse (or tokenize) Strings using the `split(..)` method, and understand when you would need to tokenize Strings.
  - Know how to obtain user input from the terminal using the `Scanner` class.
- Defensive Programming
  - Given a piece of code, be able to identify "edge cases" that could present a runtime error (e.g., negative inputs when you don't expect them, a null being input for an object, or an illegal index given for an array).
  - Unless instructed to ignore edge cases, you are expected to program defensively on the exam.
- Miscellanea
  - Wrapper classes and auto-boxing
  - It is important that you're able to read code and be able to quickly summarize and/or apply it where necessary.
  - Be familiar with classes you worked on in your homework and lab assignments, including `GuessingGame`, `ComboGuesser`, etc. For instance, you may be asked to add a new functionality to an old assignment (provided the code to the old assignment).
  - Understand how use other classes to solve problems-at-hand when given only the API (e.g., what you had to do for TweetProcessor)

## Practice Problems

1. While loops have the same expressive power as for loops (i.e., can be used interchangeably).

   **TRUE / FALSE**
   Explain:

2. Private methods can only be called by methods within the same class, which is why they are not very useful in practice.

   **TRUE / FALSE**
   Explain:

3. **Explain why the following code prints 0.

```
ArrayList<Integer> A = new ArrayList<Integer>(100); // create 100 elements?
System.out.println(A.size());
```

4. What is a "wrapper class?" What is its purpose?

5. Define a class called `Point` that represents an $(x, y)$ pair of integers. For full credit, your class must have a two-argument constructor, getters for retrieving $x$ and $y$ values, a `toString()` method, and an `equals()` method for comparing instances of Point as shown below. (Two instances are equal if their $x$ coordinates are equal, as are their $y$ coordinates.)

```
> Point p = new Point(5, -3);
> p.toString()
"(5,-3)"  (String)

> p.getX()
5 (int)

> p.getY()
-3 (int)

> Point z = new Point(5, -3);
> p.equals(z)
true (boolean)
```

6. Next, define a class called `PointCollection`, which stores an `ArrayList` of `Point` objects. Implement the following methods:

   - `public void insertInPlace(Point p)` – inserts the given `Point` into the collection. It must be inserted in such a way that the collection of points is sorted in ascending order of the $x$ value.
   - `public boolean notExists(Point p)` – returns true if the given point is not found in the collection, and false otherwise.
   - `public String toString()` – returns a String representing the collection of points.

```
> PointCollection list = new PointCollection();
> list.insertInPlace(new Point(0,0));
> list.insertInPlace(new Point(-5,2));
> list.insertInPlace(new Point(2,3));
> list.insertInPlace(new Point(-2,-9));

> list.toString()
"(-5,2), (-2,-9), (0,0), (2,3)"  (String)

> list.notExists(new Point(0,0))
false (boolean)
```

7. Consider the method below:

```
1  public void mystery(int x)
2  {
3      Circle c;
4      for(int i=x; i > 0; i -= 1)
5      {
6          c = new Circle();
7          c.changeSize(i*20);
8          if (i % 2 == 0) {
9              c.changeColor("red");
10         }
11         else {
12             c.changeColor("blue");
13         }
14         c.makeVisible();
15     }
16 }
```

(a) What does this method do if passed a zero when invoked?

(b) What does this method do if passed a three when invoked?

(c) Describe what the method does. (To get in the right frame of mind, think about what you'd write as a comment for this method.)

8. ** Write a method that inputs two int values $a$ and $b$. The method should return $a^b = a \times a \times a \times ...$ ($b$ times). You may assume $a \geq 0$ and $b \geq 0$.

9. ** Write a method that accepts an integer, then print out all positive odd numbers less than or equal to that integer. For instance, if 6 was input, then print 1, 3, 5.

10. ** The Fibonacci Sequence starts off with 0, followed by 1. The next number is always the sum of the two previous numbers. Therefore, the first 6 Fibonacci numbers in the sequence is: 0, 1, 1, 2, 3, 5. Write a method called `void fib(int n)` that prints the first $n$ Fibonacci numbers.

11. Consider the method defined below. Explain what it does using a few brief sentences.

```
public int mystery(int[] A, int B)
{
    int x = 0;
    for (int i = 0; i < A.length; i++)
    {
        if (A[i] < B) {
            x++;
        }
    }
    return x;
}
```

12. ** Consider the method defined below. Explain what it does using a few brief sentences.

```
public boolean mystery(int[] A)
{
    for (int i = 0; i < A.length; i++) {
        for (int j = i+1; j < A.length; j++) {
            if (A[i] == A[j]) {
                return true;
            }
        }
    }
    return false;
}
```

13. ** Consider the method defined below. Explain what it does using a few brief sentences.

```
public boolean mystery(int[][] A)
{
    for (int i = 0; i < A.length; i++)
    {
        for (int j = 0; j < A[i].length; j++)
        {
            if (A[i][j] < 0)
            {
                return false;
            }
        }
    }
    return true;
}
```

14. ** Consider the method defined below. What is the output when `mystery("john doe")` is called?

```
public String mystery(String name)
{
    String[] name_tokens = name.split(" ");

    String ret = name_tokens[1].substring(0,1).toUpperCase() + name_tokens[1].↩
        substring(1,name_tokens[1].length()).toLowerCase() + ", " + name_tokens↩
        [0].substring(0,1).toUpperCase();

    return ret;
}
```

15. Write a method `public int findLargest(int[][] A)` that returns the largest number found in the given 2D array. You **may** assume that `A` is not `null` and is of a square shape. Consider the following usage:

```
int[][] table = {
  {32,10,3},
  {56,54,69},
  {32,83,49},
};

System.out.println(findLargest(table));
> 83
```

16. Write a method `public int maxRow(int[][] A)` that returns the row number that has the maximum sum. You **may** assume that `A` is not `null` and is of a square shape. Consider the following usage:

```
int[][] table = {
  {32,10,3},
  {56,54,69},
  {32,83,49},
};

System.out.println(maxRow(table));
> 1

int[][] table2 = {
  {32,10,3},
  {-56,-54,-69},
  {-32,-83,-49},
};

System.out.println(maxRow(table2));
> 0
```

17. ** Consider the Notebook class below. The constructor has not been provided.

    (a) Write the default constructor, which creates an empty collection of notes.

    (b) Write a new Notebook method called `countNotesContaining(..)` that takes a String as argument, and returns the number of notes in the notebook that contain the specified string.

    (c) Write a new Notebook method called `getNotesContaining(..)` that takes a String as argument, and returns an array list of notes in the notebook that contain the specified string.

    (d) Write a new method called `countWords(..)` that counts the total number of words in all the stored notes.

```
1  public class Notebook
2  {
3      // Storage for an arbitrary number of notes.
4      private ArrayList<String> notes;
5
6      public void storeNote(String n)
7      {
8          this.notes.add(n);
9      }
10
11     // other methods not shown
12 }
```

Here is the expected result in codepad:

```
1  Notebook nb = new Notebook();
2  nb.storeNote("Java is lots of fun");
3  nb.storeNote("Exams are fun too");
4  nb.storeNote("I like donuts");
5  nb.countNotesContaining("fun")
6  > 2   (int)
7  nb.countNotesContaining("o")
8  > 3   (int)
9  nb.countWords()
10 > 12  (int)
11 ArrayList<String> results = nb.getNotesContaining("fun");
12 for (int i = 0; i < results.size(); i++) {
13     System.out.println(results.get(i));
14 }
15 > Java is lots of fun
16 > Exams are fun too
```