CMSC 131 Exam #3 Worksheet

The following exercises cover SOME (NOT ALL) the material that is part of the exam. Solutions to these exercises will not be provided, but you are welcome to discuss your solutions with the TAs or instructor during office hours. You can also post your solutions to Piazza and /or discuss them in Piazza. **It is recommended that you try these exercises on paper first (without using the computer).** Notice that most problems are open-ended without us provide a lot of details; feel free to add any missing information (the more your practice the better).

The full list of exam topics can be found on the class web page.

Exercises

1. What are the default values for an array of integers? How about array of booleans and array of String references? Notice they are NOT instance variables of a class.

2. Define a method that takes an array and two integers (startRange, endRange). The method will return a new array with the INDICES of entries from the parameter array that fall in the specified range. If startRange is greater than endRange the **IllegalArgumentException** will be thrown.

3. Define a class called **Table** that has the following instance variables: tableNumber (integer), inUse (boolean to indicate whether the table is in use), numberOfPersonsAtTheTable (integer) and a StringBuffer object that will be used to store the names of people sitting at the table.

4. Using the above **Table** class, define an array of two Table references and initialize each reference with a table. Draw a memory map for the array.

5. Create an array of two **Table** references and initialize each reference with a table. For this array:

   a. Make a shallow copy
   b. Make a deep copy
   c. Make a reference copy

   Do modifications to the shallow copy affect the original array? Does it make a difference if we use String objects?

6. The **Restaurant** class has the following specifications:

   a. Instance variables
      - Array of Table references
      - Any other values you understand are needed
   b. Methods
      - Constructor
      - Copy Constructor
      - AddTable
      - AddCustomerToTable
      - removeTable (shifts remaining tables to the left)
      - findTableMostPersons

7. A method call findValue may throw the IllegalArgumentException. The method is called from the main() method. In main() define a try / catch block to handle the IllegalArgumentException. The catch block must print the error message associated with the exemption object. Add a finally block that prints "task completed".

8. Define a method called getSize that will return the integer value that is part of a string with the format "Size:<NUMBER>" where number is an integer. For example, "Size:20" and "Size:30" are valid strings. You can assume <NUMBER> will always have two digits. Hint: using the String class substring method and Integer.parseInt can help a lot.

9. The String class subtring method returns a substring that is part of a string. Define your own mySubstring method that returns the substring that starts at startIndex and ends at endIndex.

10. Define a method that transforms a string into an array of characters.

11. Rewrite the following code fragment using the ternary (? :) operator.

```
if (salary > 10.0) {
   type = 1;
} else {
   type = 2;
}
```

12. Rewrite the following code fragment using a switch statement.

```
if (age == 10) {
   type = 1;
} else if (age == 12 || age == 13){
   type = 2;
} else {
   type = 3;
}
```

13. This space available for rent ☺

14. What takes place when no assertion is added to a JUnit test?

15. Implement a static method that computes the area of a triangle. Define JUnit tests that verify the method computation is correct. One of your tests should rely on assertTrue and a second on assertFalse.

16. Can we have more than one assertion in a JUnit test?

17. Provide an example where a privacy leak takes place.

18. Provide two examples of method signatures that overload the method **int average(int x, int y)**.

19. Using Math.random() write a method that simulates throwing a dice.

20. Using Math.random() define an expression that returns:

    a. A floating point value between 200.00 and 350.00.
    b. An **integer** value between 55 (inclusive) and 78 (inclusive).