Homework 5

CS 1323, Fall 2015

This homework is due on November 20, by 11:59 p.m. You must submit homework in a PDF file online to the dropbox on Janux. Please make sure that the formatting of the file is appropriate (no empty pages, pages with single words or excessive indentation or large spaces between lines).

If you are using Open Office or Star Office, check to be sure that the figures and diagrams in your PDF file are properly formatted, as this is a common problem with these products.

Name (5 points): Hunter Black

Student number: 113229605

1. (10 points) Determine whether each of these statements is true or false.
   a. Some classes have no objects. **T**
   b. Objects are allocated in the heap. **T**
   c. You can have two references pointing to the same object. **T**
   d. Objects can have more than one reference. **T**
   e. StringBuilder objects can be constructed without using new. **F**

2. (12 points; 6 points each) Tracing the loops below in the table at the right. You may add more lines to the table if you need them. Only the variables in the table need to be traced. If the loop is an infinite loop, trace three iterations and write "Infinite."

a)

```
String input = "lmnop";
for (int index = 1; index < input.length();++index )
{
        input = input.substring(index, input.length());
        // read the API
}
```

index	input
1	lmnop
1	mnop
2	nop
3	op
4	p

b) To show lists of Strings, use curly braces to enclose the list, and separate the elements with commas (e.g. {a, b, c}). This is not a memory diagram.

StringBuilder sb = new StringBuilder ("LMNOPQ");

```
for (int index = 0; index < sb.length(); index = index + 2 )
        //  watch increment—it is not 1
{
        sb.insert(index, "a");
}
```

index	**sb**
0	LMNOPQ
0	aLMNOPQ
2	aLaMNOPQ
4	aLaMaNOPQ
6	aLaMaNaOPQ
8	aLaMaNaOaPQ
10	aLaMaNaOaPaQ

3. (10 points) Draw a memory diagram for the code fragment below. You may assume this code fragment is in the main method.

```
ArrayList<String> data = new
ArrayList<String>();
data.add("Go");
data.add("Sooners");
data.add("Beat");
data.add("Texas");
data.remove(3);
data.add("Baylor");
```

Stack frame

Identifier	Address	Contents
data	100	1000
	101	
	102	
	103	

Heap

Identifier	Address	Contents
0	1000	~~Null~~, Go
1	1001	~~Null~~, Sooners
2	1002	~~Null~~, Beat,
3	1003	~~Null~~, ~~Texas~~, Baylor
4	1004	Null
5	1005	Null
6	1006	Null
7	1007	Null
8	1008	Null
9	1009	Null
size	1010	~~0, 1, 2,~~ 4
Capacity	1011	10
	1012	
	1013	

4. (10 points; 2 points for a); 4 points each for b) and c)) Use the Collections class to accomplish each of the following tasks in a few lines of code. You may assume that the ArrayList below has been declared, constructed, and initialized with data.

ArrayList<Integer> list; // Notice Integer, not int

Remember when reading the Collections class API that ArrayList can be used for anything that is a List or a Collection. Don't worry about the crazy <? extends T> or similar syntax—everything that is logical to do is legal here. When you see Object, Integer can be used.

a) Find the int value of the largest Integer in the list.

Collections.max(list);

b) Put the list in reverse sorted order (in other words, in descending order instead of ascending order). Consider using more than one command to accomplish this task.

Collections.sort(list);

Collections.reverse(list);

c) Create a list that contains all of the numbers that are between the largest and smallest numbers in a list, but not in the list. So if the list contained {1, 3, 5} the new list should contain {2, 4}.

```
ArrayList<Integer> newList = new ArrayList<Integer>();
Collections.sort(list);
for (int j = list.get(0) + 1; j < Collections.max(list); ++j)
{
        If (Collections.frequency(list, j) == 0)
        {
                newList.add(j);
        }
}
```

5. (10 points) The AtomicInteger class is used for programs with threads. A thread in a program is like a train of thought for a person. For example, when you use your email program you could have five new message open at a time. Each message has its own thread. The problem with programming with threads is that two threads can try to change a single data element at the same time. The results are unpredictable, called a race condition. Either thread could get there last (and therefore set the value the way it wants it), but they also could get mixed up together and get a variable that is in neither thread. This class is used to avoid this problem. If this doesn't make sense, don't worry about it. You don't have to understand threads to do this problem successfully.

If you think this is no big deal, read about the biggest software disaster that has happened to date here: https://en.wikipedia.org/wiki/Therac-25. This type of error was one of the errors involved.

If you enjoy reading that, you may want to read this over Winter Break: http://ethics.csc.ncsu.edu/risks/safety/killer_robot/killer_intro.html.

a. Are AtomicInteger objects mutable or immutable?
   Mutable

b. Create an AtomicInteger object whose value was read from the console.
   ```
   Scanner input = new Scanner(System.in);
   AtomicInteger num = new AtomicInteger(input.nextInt());
   ```

c. Add 1 to the AtomicInteger with reference size.
   ```
   size.addAndGet(1);
   ```

d. Convert the AtomicInteger object with reference size to a String object.
   ```
   String str = size.toString();
   ```

e. Convert the AtomicInteger object with reference size to a String object in a second way. You may use methods in the String or Integer classes.
   ```
   String str = size.toString(); //using the Integer toString method
   ```

6. (10 points) Suppose there is a class in Java called RGB. This class is used to store the amount of red, blue, and green light used (these are int values) to paint a color on the screen using graphics. Determine whether each of the method signatures below should be a class method or an instance method, based on the signature. Do not try to get this by looking at the Color class in Java—I've changed the method signatures.

   a. void setRed(int red)

   The method changes the amount of red stored in this RGB object to red.

   Class

   b. RGB setGreen(RGB color, int green)

   The method changes the amount of green stored in the RGB object color to green.

   Class

   c. String toString()

   The method converts this RGB object to a String object.

   Instance

   d. String toString(RGB color)

   The method converts the RBG object color to a String object.

   Class

   e. int getRed(RGB color)

   The method returns the amount of red in the RGB object color.

   Class

   f. int getBlue()

   The method returns the amount of blue in this RGB object.

   Instance

   g. void darker(RGB color)

   The method makes the RGB object color darker.

   Class

   h. RGB decode (String color)

   This method takes a String with the following format: "(127, 53, 27)" and changes it into an RGB object. The String above would become an RGB object with red = 127, green = 53, and blue = 27.

   Class

   i. void lighter()

   The method makes this RGB object lighter.

   Instance

   j. RGB getRGBFromHSV(int hue, int saturation, int value)

   The method returns a new RGB object that has red, blue, and green values converted from the hue, saturation, and value. Hue, saturation, and value is another way of describing a color.

   Class