Homework 5 Solution

CS 1323, Fall 2015

1. (10 points) Determine whether each of these statements is true or false.
   1. Some classes have no objects.

True. The Math and Arrays classes are examples.

* 1. Objects are allocated in the heap.

True.

* 1. You can have two references pointing to the same object.

True.

* 1. Objects can have more than one reference.

False. Each object has only one reference.

* 1. StringBuilder objects can be constructed without using new.

False. String objects can be, but StringBuilder can’t.

1. (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”;

|  |  |
| --- | --- |
| **index** | **input** |
| 1 | lmnop |
| 2 | mnop |
| 3 | op |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

for (int index = 1; index < input.length();++index )

{

input = input.substring(index, input.length());

// read the API

}

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”);

|  |  |
| --- | --- |
| **index** | **sb** |
| 0 | LMNOPQ |
| 2 | aLMNOPQ |
| 4 | aLaMNOPQ |
| 6 | aLaMaNOPQ |
| 8 | aLaMaNaOPQ |
| 10 | aLaMaNaOaPQ |
| 12 | aLaMaNaOaPaQ |

for (int index = 0; index < sb.length(); index = index + 2 )

// watch increment—it is not 1

{

sb.insert(index, “a”);

}

1. (10 points) Draw a memory diagram for the code fragment below. You may assume this code fragment is in the main method.

Heap

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

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 |  |

1. (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.

Integer maxValue = Collections.max(list);

int max = maxValue.intValue();

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}.

Integer min = Collections.min(list);

Integer max = Collections.max(list);

ArrayList<Integer> result = new ArrayList<Integer>();

for (int num = min.intValue()+1; num < max.intValue(); ++num)

{

int search = Collections.binarySearch(list, num);

if (search < 0)

result.add(num);

}

1. (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.

* 1. Are AtomicInteger objects mutable or immutable?

Mutable. As an example addAndGet() is a mutator.

* 1. Create an AtomicInteger object whose value was read from the console.

Scanner keyboard = new Scanner(System.in);

System.out.println(“Enter an integer”); // don’t care if this is here or not

int input = keyboard.nextInt();

AtomicInteger value = new AtomicInteger(input);

* 1. Add 1 to the AtomicInteger with reference size.

size.incrementAndGet();

* 1. Convert the AtomicInteger object with reference size to a String object.

String result = size.toString();

* 1. 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.

int value = size.get();

String result = String.valueOf(value); // lots of other possibilities

1. (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.
   1. void setRed(int red)

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

Instance method

* 1. RGB setGreen(RGB color, int green)

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

Class method

* 1. String toString()

The method converts this RGB object to a String object.

Instance method

* 1. String toString(RGB color)

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

Class method

* 1. int getRed(RGB color)

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

Class method

* 1. int getBlue()

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

Instance method

* 1. void darker(RGB color)

The method makes the RGB object color darker.

Class method

* 1. 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 method

* 1. void lighter()

The method makes this RGB object lighter.

Instance method

* 1. 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 method