University of Maryland College Park

Dept of Computer Science

CMSC131 Fall 2018

Exam #3 Key

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g. 123456789):

Instructions

- Please print your answers and use a pencil.
- Do not remove the staple from the exam. Removing it will interfere with the Gradescope scanning process.
- To make sure Gradescope can recognize your exam, print your name, write your directory id at the bottom of pages with the text DirectoryId, provide answers in the rectangular areas provided, and do not remove any exam pages. Even if you use the provided extra pages for scratch work, they must be returned with the rest of the exam.
- This exam is a closed-book, closed-notes exam, with a duration of 50 minutes and 200 total points.
- Your code must be efficient.
- You don't need to use meaningful variable names; however, we expect good indentation.

Grader Use Only

#1	Problem #1 (Miscellaneous)	42	
#2	Problem #2 (Memory Map)	42	
#3	Problem #3 (Arrays)	64	
#4	Problem #4 (String Manipulation)	52	
Total	Total	200	

Problem #1 (Miscellaneous)

1. (3 pts) What is the output for the following code fragment?

```
int k = 1, m = 1;
while (k <= 1) {
       while(m <= 1) {
              if (m == 1 ) {
                     break;
              }
              m++;
       }
       System.out.println("Hi");
       k++;
}
System.out.println("Done");
```

Answer:

```
Hi
Done
```

2. (3 pts) Appending (concatenating) strings is:

   a. An expensive operation and an alternative (e.g., StringBuffer) should be used.
   b. A cheap operation.
   c. An operation that have the same performance as using a StringBuffer.
   d. None of the above.

   Answer: a.

3. (3 pts) When we execute the following code:

```
System.out.println(3.9 - 3.8 == .10);
```

   a. We will always get true.
   b. Getting false is possible as some numbers cannot be represented precisely in binary.
   c. An exemption will be thrown.
   d. None of the above.

   Answer: b.

4. (3 pts) When an exception occurs, Java pops back up the call stack to each of the calling methods to see whether the exception is being handled in a catch block of the method. **True** or **False**.

   Answer: True

5. (3 pts) In a JUnit test method if you don't provide an assertion the test is considered to have succeeded. **True** or **False**.

   Answer: True

6. (3 pts) Creating an array of size 0 is not allowed in Java. **True** or **False**.

   Answer: False

7. (3 pts) The code in the finally block is executed:

   a. Only when no exemption takes place.
   b. Always
   c. Only when the exemption takes place.
   d. None of the above.

   Answer: b

8. (5 pts) The following class has a privacy leak. Fix the code to remove it (feel free to edit and cross out the code provided).

```
public class Example {
   private StringBuffer data;

   public Example(String dataIn) {
      data = new StringBuffer();
      data.append(dataIn);
   }

   public StringBuffer getData() {
      return data;
   }
}
```

   Answer: getData method will return a copy of data

9. (3 pts) We have an array of string objects and we would like to make a copy of the array so changes to the new array will not affect the original. To make the copy:

   a. We need to make a deep copy, otherwise changes in the new array will affect the original.
   b. A shallow copy where we only duplicate the array object and not the string objects is enough.
   c. A reference copy is enough.
   d. None of the above.

   Answer: b.

10. (6 pts) Math.random() returns a value between 0 (inclusive) and less than 1. Complete the following assignment so x is assigned a random **integer** value between 1(inclusive) and 100 (inclusive). The only function you can use is Math.random().

```
int x =
```

   One Possible Answer: `(int)(Math.random() * 100) + 1;`

11. (7 pts) Rewrite the following code fragment by replacing the if statement with the ternary (? :) operator.

```
/* Original Code */
String name;
int y = scanner.nextInt();
if (y > 0) {
   name = "Pos";
} else {
   name = "Neg";
}

/* Rewritten Code */
String name;
int y = scanner.nextInt();

name =
```

   Answer:

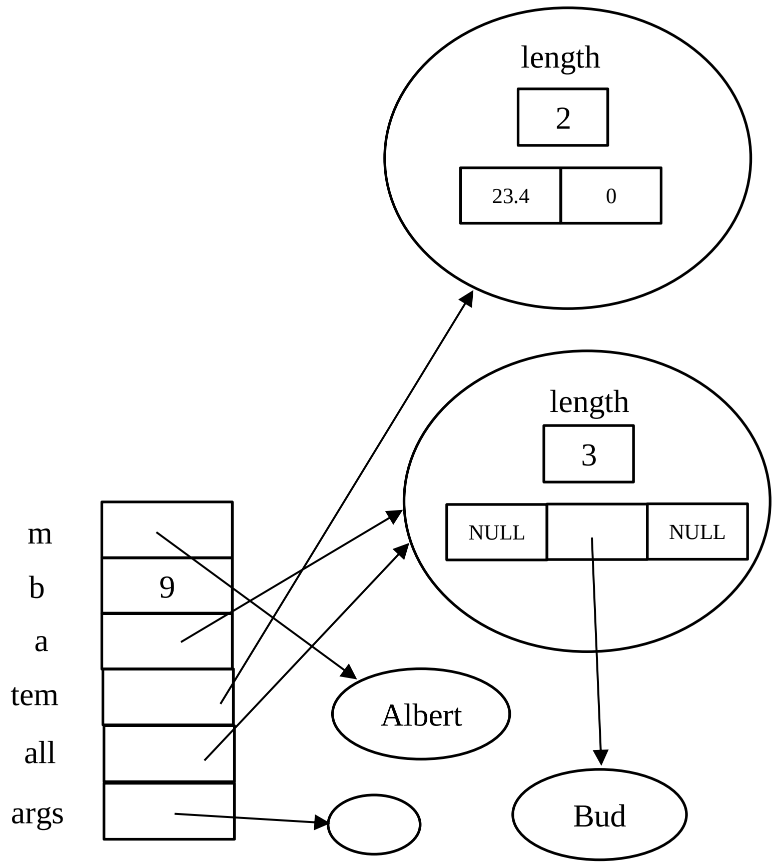
```
name = y > 0 ? "Pos" : "Neg";
```

Problem #2 (Memory Map)

Draw a memory map for the following program at the point in the program execution indicated by the comment **/*HERE */**. Remember to draw the stack and the heap. If an entry has a value of null value write NULL. If an entry has a value of 0 do not leave it blank; write 0.

```
public static void process(StringBuffer[] a,
                           double b) {
   a[0].append("bert");
   StringBuffer m = a[0];
   a[0] = null;
   b = 9;
   /* HERE */
}

public static void main(String[] args) {
   StringBuffer[] all = new StringBuffer[3];
   double[] tem = new double[2];

   all[0] = new StringBuffer();
   all[0].append("Al");

   all[1] = new StringBuffer();
   all[1].append("Bud");

   tem[0] = 23.4;
   process(all, tem[0]);
}
```

Solution



Problem #3 (Arrays)

Implement a method named **getIndicesArray** that has the prototype below. The method returns an array with the INDICES of elements in the **data** array that have a value less than or equal to **limit**. For this problem:

- You may only perform a single pass over the array.
- The array to return must have a length that corresponds to the length of the **data** array plus one.
- The last index added to the result array will be followed by -1 in order to mark the end of data.
- If the **data** parameter is null, the method will throw the exemption **IllegalArgumentException** with the message "Invalid" and no further computation will take place.
- Below we included a driver and the corresponding output that relies on the method **getIndicesArray**. Feel free to ignore if you know what to implement.

  One Possible Answer:

```
public static int[] getIndicesArray(int[] data, int limit) {
       if (data == null) {
              throw new IllegalArgumentException("Invalid");
       }
       int[] answer = new int[data.length + 1];

       int k = 0;
       for (int i = 0; i < data.length; i++) {
              if (data[i] <= limit) {
                     answer[k++] = i;
              }
       }
       answer[k] = -1;

       return answer;
}
```

Problem #4 (String Manipulation)

Implement a method named **getSerialNumber** that has the prototype below. The method will process a string and extract an integer that is part of a valid string. A valid string is one that starts with the uppercase letter A and is followed by a 5 digit number. Any number of spaces can precede A and follow the last digit. For example, “ A24567   “ is a valid string; “A235B7” and “A234” are invalid. The **getSerialNumber** method returns the integer that is part of a valid string parameter and -1 if the string parameter is an invalid string or null. For this problem:

- You can use the trim() method to remove any blank spaces surrounding the parameter.
- You need to use the method Integer.parseInt() to transform to an integer the substring that is part of a valid string. If the NumberFormatException is thrown you can then return -1.
- You can use the String class substring method to retrieve the appropriate substring. The following information is from the Java API:

  public String substring(int beginIndex,int endIndex) - Returns a string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1
- Below we included a driver and the corresponding output that relies on the method **getSerialNumber**. Feel free to ignore if you know what to implement.

`/* Driver */`	`/* Output */`
`System.out.println("SNEx1: " + getSerialNumber("  A83744  "));`	`SNEx1: 83744`
`System.out.println("SNEx1: " + getSerialNumber("  A837   "));`	`SNEx1: -1`
`System.out.println("SNEx1: " + getSerialNumber(null));`	`SNEx1: -1`

```
public static int getSerialNumber(String value) {
```

One Possible Answer:

```
public static int getSerialNumber(String value) {
       if (value != null) {
              value = value.trim();
              if (value.length() == 6 && value.charAt(0) == 'A') {
                     try {
                            return Integer.parseInt(value.substring(1, 6));
                     } catch (NumberFormatException e) {
                            return -1;
                     }
              }
       }
       return -1;
}
```