University of Maryland College Park
Department of Computer Science
CMSC131 Fall 2019
Exam #2 Key

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g., 123456789):

<u>Instructions</u>

- Please print your answers and use a pencil.
- This exam is a closed-book, closed-notes exam with a duration of 50 minutes and 200 total points.
- **Do not remove the exam's staple.** Removing it will interfere with the scanning process (even if you staple the exam again).
- Write your directory id (e.g., terps1, not UID (e.g., 111222333)) at the bottom of pages with **DirectoryId**.
- Provide answers in the rectangular areas.
- Do not remove any exam pages. Even if you don't use the extra pages for scratch work, return them with the rest of the exam.
- Your code must be efficient and as short as possible.
- If you continue a problem on the extra page(s) provided, make a note on the particular problem.
- For multiple choice questions you can assume only one answer is expected, unless stated otherwise.
- You don't need to use meaningful variable names; however, we expect good indentation.
- **You must write your name and id at this point (we will not wait for you after time is up).**
- You must stop writing once time is up.

<u>Grader Use Only</u>

#1	Problem #1 (Miscellaneous)	48	
#2	Problem #2 (Memory Map)	22	
#3	Problem #3 (Class Implementation)	130	
Total	Total	200	

Problem #1 (Miscellaneous)

1. (3 pts) How many objects exist in the following code fragment?

```
String s1 = "today";
String s2 = s1;
StringBuffer s3 = new StringBuffer();
```

   Answer: 2

2. (3 pts) Would the following class compile?   Yes / No

```
public class Misc2 {
  public static void main(String[] args) {
     String m;

     System.out.println(m);
  }
}
```

   Answer: No. m has no value.

3. (3 pts) Which of the following actually creates an object?

   a. Destructor
   b. Constructor
   c. new operator
   d. None of the above.

   Answer: c.

4. (3 pts) Circle all the areas where an object can be found.

   a. Stack
   b. Heap
   c. Only in the first method that has been called.
   d. None of the above.

   Answer: b.

5. (3 pts) Object **a** has 20 instance variables and object **b** has one instance variable. The amount of time it will take to pass **a** or **b** to a method is:

   a. Higher for object **a**.
   b. Higher for object **b**.
   c. Same.
   d. None of the above.

   Answer: c.

6. (3 pts) When we call **m.sum(k)** which value does the special value **this** (present inside of the **sum** method) have?

   a. It has the same value **m** has.
   b. It has the same value **k** has.
   c. It has the value null.
   d. None of the above.

   Answer: a.

7. (3 pts) What is the output of the following program?

```
public class Station {
  private boolean rainy;
  private StringBuffer log;
  private int support;

  public static void main(String[] args) {
    Station s = new Station();

    System.out.println(s.rainy);
    System.out.println(s.log);
    System.out.println(s.support);
  }
}
```

Answer:

```
false
null
0
```

8. (3 pts) Where can primitive type variables reside in Java? Circle all that apply.

   a. Heap
   b. Stack
   c. main method
   d. None of the above.

   Answer: a. and b. or a., b, and .c

9. (3 pts) If you define a JUnit test and don't add any assertions:

   a. The test will fail when it is run.
   b. The test will succeed when it is run.
   c. The test will not compile.
   d. None of the above.

   Answer: b

10. (3 pts) The code in the finally block is executed:

    a. Always.
    b. When two or more exemptions take place.
    c. Before the try block.
    d. None of the above.

    Answer: a.

11. (3 pts) When we execute the following code:

```
double x = .10;
while (x != .20) {

   x = x + .01;
}
System.out.println(x);
```

    a. The program will always print 0.20
    b. The loop may become an infinite loop.
    c. None of the above.

    Answer: b. (we should not compare floating point values)

12. (3 pts) What does the **new** operator return?

    a. The object that was created.
    b. The address in the heap where the object was created.
    c. The address in the stack where the object was created.
    d. None of the above.

    Answer: b.

13. (6 pts) The method **Task.completeTask()** is expected to return 20. Write a student test that verifies whether the correct value is returned by the method.

```
public class StudentTests {
```

    Answer:

```
@Test
public void test() {
        assertTrue(20 == Task.completeTask());
}
```

14. (6 pts) Assuming variable **c** is a character type variable and **s** is a string type variable, rewrite the following code using a single statement and the ternary operator so **s** is assigned the correct value.

```
if (c == 'y') {
   s = "Yes";
} else {
   s = "No";
}
```

    Answer:

```
s = c == 'y' ? "Yes" : "No";
```

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 write **null**. If an entry has a value of 0 do not leave it blank; write 0.

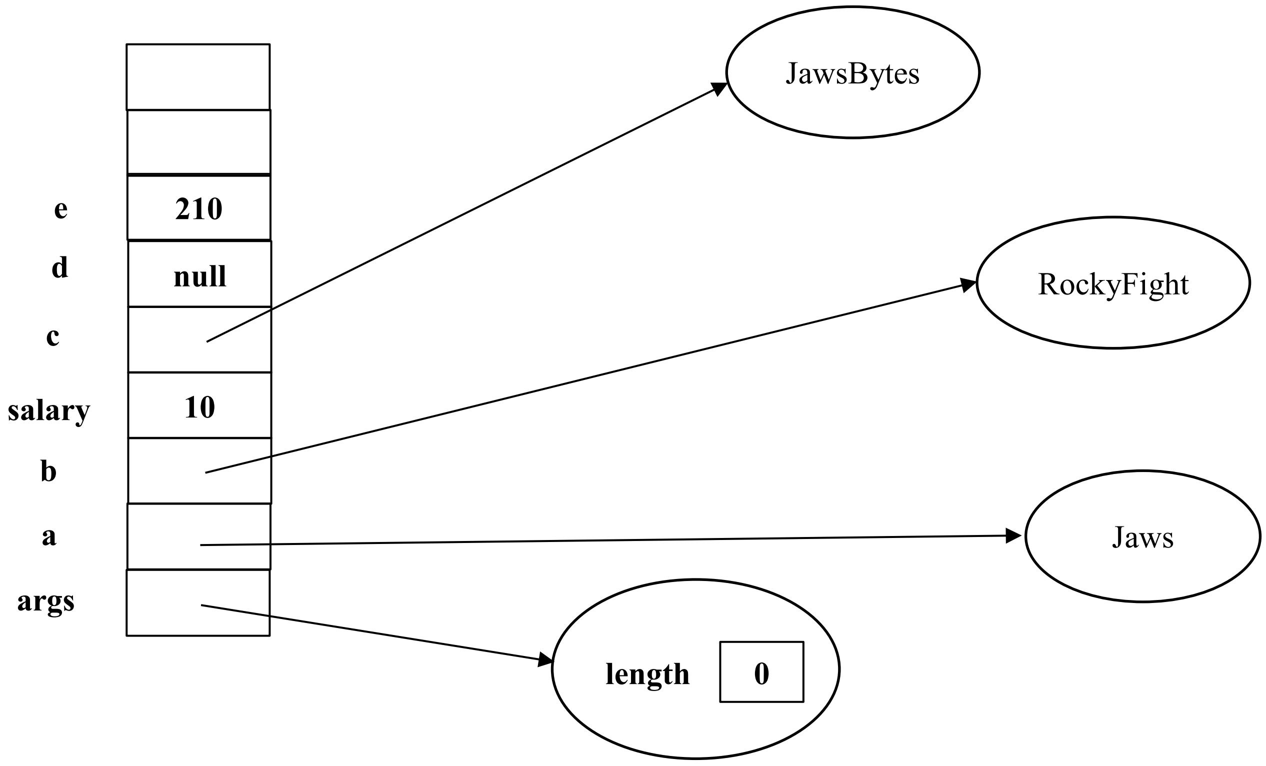
```
public class MemMap {
   public static void filterData(String c, StringBuffer d, int e) {
      c += "Bytes";
      e += 200;
      d.append("Fight");
      d = null;
      /* HERE */
   }

   public static void main(String[] args) {
      String a = new String("Jaws");
      StringBuffer b = new StringBuffer("Rocky");
      int salary = 10;

      filterData(a, b, salary);
   }

}
```

Answer:



Problem #3 (Class Implementation)

Complete the implementation of a class called **Sandwich** that represents a sandwich. A sandwich has a name (**name** instance variable), a number of calories (**cal** instance variable), a number of ingredients (**numOfIng** instance variable), and ingredients (**ings** instance variable). The number of **Sandwich** objects created is represented by the static field **TOTAL**. Below we have provided a driver that illustrates some of the functionality associated with the class. Feel free to ignore it if you know what to implement. You MAY NOT add any methods beyond the ones specified below (not even private). Also you MAY NOT add any additional instance or static variables. All the methods below are public unless specified otherwise.

```
public class Sandwich {
   private String name;
   private int cal, numOfIng;
   private StringBuffer ings;
   private static int TOTAL = 0;
}
```

1. Define a **constructor** that takes a string parameter called **name**. The parameter represents the name of the sandwich. The constructor will initialize the **name** instance variable with the parameter and the **ings** instance variable with a StringBuffer object. If the **name** parameter is null, the IllegalArgumentException with the message "Invalid name" will be thrown. Make sure you adjust the total number of objects created so far.

   Answer:

```
public Sandwich(String name) {
        if (name == null) {
                throw new IllegalArgumentException("Invalid name");
        }
        this.name = name;
        ings = new StringBuffer();
        TOTAL++;
}
```

2. Define a **default constructor** that initializes a sandwich with the name "NONAME". You must call the previous constructor in order to implement this constructor, otherwise you will not get any credit.

   Answer:

```
public Sandwich() {
   this("NONAME");
}
```

3. Define a **copy constructor** method for the class. Changes to the copy should not affect the original object.

   Answer:

```
public Sandwich(Sandwich san) {
        this.name = san.name;
        this.cal = san.cal;
        this.ings = new StringBuffer(san.ings);
        this.numOfIng = san.numOfIng;
        TOTAL++;
}
```

4. Define a method called **addIngredient** that has two parameters: a string called **ing** and an integer called **calories**. The method will add the ingredient represented by the **ing** parameter to the **ingredients** instance variable, and will increase the number of **calories** of the sandwich by the specified parameter value. The method returns a reference to the current object. Make sure you update any other instance variables accordingly.

   Answer:

```
public Sandwich addIngredient(String ing, int calories) {
       ings.append(ing);
       this.cal += calories;
       numOfIng++;

       return this;
}
```

5. Define a method called **getIngredients** that returns the ingredients of the sandwich. You must avoid privacy leaks.

   Answer:

```
public StringBuffer getIngredients() {
       return new StringBuffer(ings);
}
```

6. Define a **static** method called **getTotalSandwiches** that returns the total number of sandwiches created so far.

   Answer:

```
public static int getTotalSandwiches() {
       return TOTAL;
}
```

7. Define an **equals** method for the class. Two sandwiches are considered equal if they have the same name.

   Answer:

```
public boolean equals(Object obj) {
       if (obj == this)
              return true;
       if (obj == null || getClass() != obj.getClass())
              return false;
       Sandwich sand = (Sandwich) obj;

       return name.equals(sand.name);
}
```

8. Define a **compareTo** method that takes a **Sandwich** as a parameter and returns an integer. The method will return a negative value if the number of calories of the current object is less than the parameter, and a positive value if the number of calories is larger than the parameter. If the number of calories is the same, the method will compare the number of ingredients. If the number of ingredients of the current object is larger than the parameter, a negative value will be returned, and a positive value if the number is smaller. A value of 0 will be returned if the number of ingredients in the same. If we were to sort **Sandwich** objects with this method, sandwiches will lower calories will appear first and for sandwiches with the same calories, sandwiches with more ingredients will appear first.

   Answer:

```
public int compareTo(Sandwich san) {
       int res = cal - san.cal;

       if (res == 0) {
              res = san.numOfIng - numOfIng;
       }
       return res;
}
```

9. Define a **toString()** method that returns a string with the name, calories, number of ingredients, and ingredients of a sandwich, each separated by commas. Any number of spaces (including 0) between the values is OK.

Answer:

```
public String toString() {
        String answer = name + ", " + cal + ", ";

        answer += numOfIng + ", " + ings.toString();

        return answer;
}
```

10. Modify the following code in order for the **process** method to print the message associated with the exemption that is thrown when the parameter is null.

```
public static void process(String name) {

   Sandwich sand = new Sandwich(name);
   System.out.println(sand);

}
```

Answer:

```
try {
        Sandwich sand = new Sandwich(name);
        System.out.println(sand);
} catch(IllegalArgumentException e) {
        System.out.println(e.getMessage());
}
```