University of Maryland College Park

Dept of Computer Science

CMSC131 Fall 2018

Exam #2 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 each page, 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.
- You don't need to use meaningful variable names; however, we expect good indentation.

Grader Use Only

#1	Problem #1 (Miscellaneous)	30	
#2	Problem #2 (Memory Map)	30	
#3	Problem #3 (Class Implementation)	140	
Total	Total	200	

DirectoryId:

Problem #1 (Miscellaneous)

1. (3 pts) A method should be defined as static if (circle all that apply):

   a. It does not require access to instance variables.
   b. We would like to call the method by using the class name (e.g., JOptionPane.showInputDialog).
   c. The method makes a reference to special value **this**.
   d. It calls a static method.
   e. None of the above.

   Answer: a., b.

2. (3 pts) How many objects exist in the following code fragment?

```
String m;
```

   Answer: 0

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

```
String s = "Taco";
```

   Answer: 1

4. (3 pts) Which of the following actually creates an object?

   a. new
   b. constructor
   c. private
   d. None of the above.

   Answer: a

5. (3 pts) When we call **a.doSomething(b)** which value does the special value **this** (present inside of the doSomething method) have?

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

   Answer: a.

6. (3 pts) What is the output of the following program?

```
public class Values {
   private String distance;
   private double cost;
   private boolean completed;

   public static void main(String[] args) {
      Values values = new Values();
      System.out.println(values.distance);
      System.out.println(values.cost);
      System.out.println(values.completed);
   }
}
```

   Answer:

```
null
0.0
false
```

7. (3 pts) In the previous **Values** class, is there a default constructor associated with the class? Answer: Yes

8. (3 pts) Name one immutable Java class mentioned in lecture:

   One Possible Answer: String

9. (3 pts) Where in memory objects reside in Java? (Circle all that apply)

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

   Answer: a.

10. (3 pts) What happens if **new** is called and there is no space available in the heap?

    a. The program will crash.
    b. The program will continue executing using the stack as a heap.
    c. The main method will be used as a the heap area.
    d. None of the above.

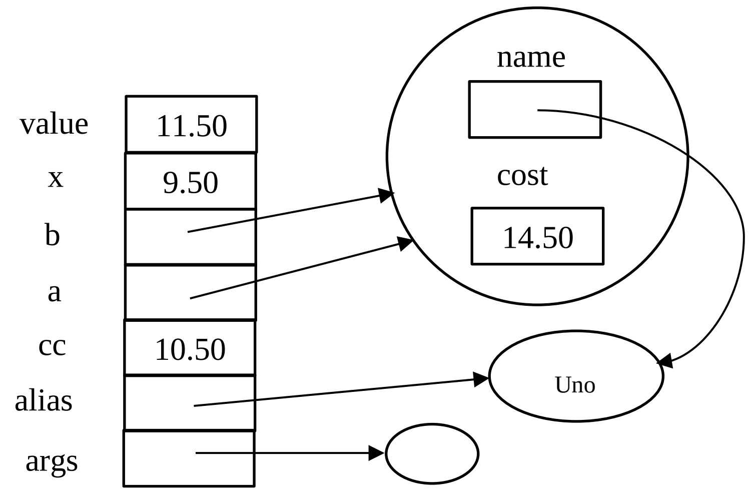
    Answer: a.

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.

Answer:

```
public class Driver {
   public static void process(Road b, double x) {
      double value = x + 1;
      b.increaseCost(x);
      x--;
      /* HERE */
   }

   public static void main(String[] args) {
      String alias = "Uno";
      double cc = 10.50;
      Road a = new Road(alias, 4);
      process(a, cc);
   }
}

public class Road {
   private String name;
   private double cost;

   public Road(String nameIn, double costIn) {
      name = nameIn;
      cost = costIn;
   }

   public void increaseCost(double delta) {
      cost += delta;
   }

   public double getCost() { return cost; }
   public String getName() { return name; }
}
```



Problem #3 (Class Implementation)

Implement a class named **Printer** according to the specifications below. The class allow us to define a printer that can hold a maximum number pages and that keeps track of the content that has been printed. Below we have provided a driver that illustrates the functionality associated with the **Printer** class. Feel free to ignore it if you know what to implement. For this problem keep in mind that the StringBuffer **append()** method allows you to append a String to the StringBuffer object and the **toString()** method returns a string with the StringBuffer content.

1. Provide a class definition that includes the following private instance variables

    a. **brand** → String variable representing a printer's brand.
    b. **maxPages** → Maximum number of pages the printer can print.
    c. **numberOfPrintedPages** → Number of printed pages.
    d. **printedContent** → StringBuffer that stores the content that has been printed.

    and the following private static variable:

    totalPrinters → Keeps track of how many Printer objects have been created.

```
public class Printer {
```

Answer:

```
 public class Printer {
    private String brand;
    private int maxPages, numberPrintedPages;
    private StringBuffer printedContent;
    private static int totalPrinters = 0;
```

2. Define a constructor that takes a string (brand) and maximum number of pages (maxPages) as parameters. The current object is initialized based on these parameters. In addition, a StringBuffer object will be created and the number of printed pages will be set to 0. Add any other initialization you understand is necessary in order to keep track of the number of objects that have been created.

Answer:

```
    public Printer(String brand, int maxPages) {
           this.brand = brand;
           this.maxPages = maxPages;
           this.printedContent = new StringBuffer();
           totalPrinters++;
    }
```

3. Define a constructor that takes a string (brand) as parameter. It initializes the printer with the specified brand and with 100 as the maximum number of pages. You must call the previous constructor in order to implement this constructor.

Answer:

```
    public Printer(String brand) {
           this(brand, 100);
    }
```

4. Define a **static** method **getTotalPrinters** that returns the total number of **Printer** objects created.

Answer:

```
    public static int getTotalPrinters() {
           return totalPrinters;
    }
```

5. Define a method called **printPage** that has a string as a parameter representing the content to print. The method will append the string parameter to the **printedContent** StringBuffer object if the number of printed pages has not exceeded the maximum number of pages and if the parameter is not null; otherwise the message "Printer error" (using System.out.println) will be printed. The method will return a reference to the current object in all cases.

Answer:

```
public Printer printPage(String content) {
        if (numberPrintedPages < maxPages && content != null) {
                numberPrintedPages++;
                printedContent.append(content);
        } else {
                System.out.println("Printer error");
        }
        return this;
}
```

6. Define a **toString()** method that returns a string with the following format:

Brand: <BRAND>
MaxPages: <MAXPAGES>
NumberPrintedPages: <NUMBER_PRINTED_PAGES>
PrintedContent: <PRINTED_CONTENT>

where <BRAND>, <MAXPAGES>, <NUMBER_PRINTED_PAGES>, <PRINTED_CONTENT> correspond to the brand, the maximum number of pages, the number of printed pages and the content of **printedContent**, respectively. See sample driver below for an example.

Answer:

```
public String toString() {
        String answer = "";

        answer += "Brand: " + brand;
        answer += "\nMaxPages: " + maxPages;
        answer += "\nNumberPrintedPages: " + numberPrintedPages;
        answer += "\nPrintedContent: " + printedContent;

        return answer;
}
```

7. Implement an **equals** method for the class. Two printers are considered equal if they have the same brand and the same number of maximum pages.

Answer:

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

        return brand.equals(printer.brand) && maxPages == printer.maxPages;
}
```