



# University of Maryland College Park

## Department of Computer Science

### CMSC131 Spring 2019

### Exam #2 Key

FIRSTNAME, LASTNAME (PRINT IN UPPERCASE):

STUDENT ID (e.g., 123456789):

#### Instructions

- 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) at the bottom of pages with the text DirectoryId, provide answers in the rectangular areas, and 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.
- 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.

#### Grader Use Only

#1	Problem #1 (Miscellaneous)	48	
#2	Problem #2 (Memory Map)	50	
#3	Problem #3 (Class Implementation)	102	
<b>Total</b>	Total	200	

## Problem #1 (Miscellaneous)

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

- a. It does not access instance variables.
- b. If it is a public method.
- c. The method makes a reference to special value **this**.
- d. None of the above.

Answer: a.

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

```
StringBuffer m;
```

Answer: 0

3. (3 pts) Instance variables are created when:

- a. An instance of the class is created.
- b. When the constructor is called.
- c. When a set method is called.
- d. None of the above.

Answer: a.

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) We can call a non-static method of a class (circle all that apply):

- a. Only if an object of the class has already been created.
- b. From a static method of the class.
- c. By using the class name (e.g., ClassName.method)
- d. None of the above.

Answer: a. or b. (or both).

6. (3 pts) When we call **a.completeTask(b)** which value does the special value **this** (present inside of the completeTask method) have?

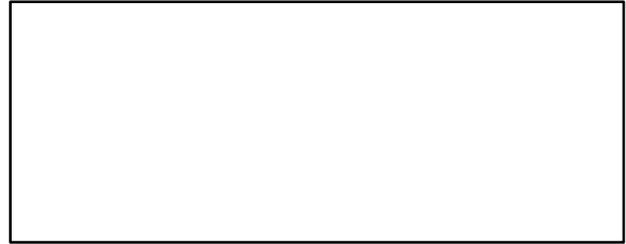
- a. It has the same value **a** has
- b. It has the same value **b** has
- c. It has the value null
- d. None of the above.

Answer: a.

7. (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 or 0  
False

8. (3 pts) In the previous **Values** class, is there a default constructor associated with the class? Yes / No

Answer: Yes.

9. (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. (if they mark c. do not penalize)

10. (3 pts) If you define a JUnit test and don't add any assertions (e.g., assertTrue) then:

- 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.

11. (3 pts) We have an array of StringBuffer 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 StringBuffer objects is enough.
- c. A reference copy is enough.
- d. None of the above.

Answer: a.

12. (3 pts) The code in the finally block is executed:

- a. After a catch clause has been executed.
- b. When two or more exemptions take place.
- c. Before the try block.
- d. None of the above.

Answer: d.

13. (3 pts) When we execute the following code:

```
System.out.println((14.90 - 14.80) == .10 ? "Hello" : "Bye");
```

- a. We will always get “Hello”.
- b. Getting “Bye” is possible as some numbers cannot be represented precisely in binary.
- c. The expression within the System.out.println does not compile.
- d. None of the above.

Answer: b.

14. (3 pts) The class Toy needs to implement the equals method. Which of the following is the correct prototype (first line of the method) for the equals method needed by this class?

- a. public boolean equals(Object obj)
- b. public boolean equals(Toy obj)
- c. public boolean equals(String obj)
- d. None of the above.

Answer: a.

15. (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 2 (inclusive) and 50 (inclusive). The only function you can use is Math.random().

```
int x =
```

Answer: `(int) (49 * Math.random()) + 2`

## 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.

```

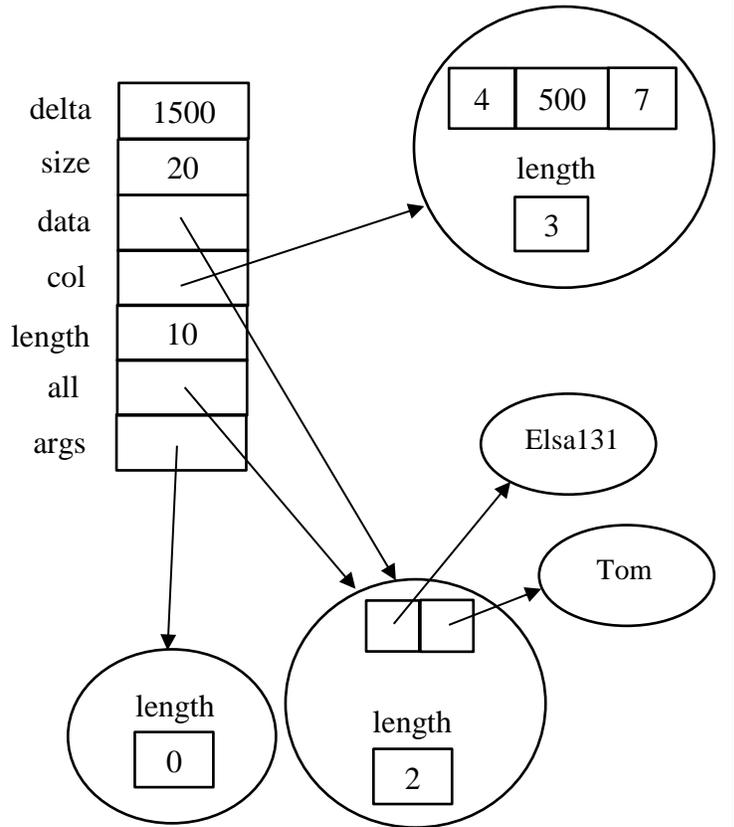
public static void process(StringBuffer[] data,
                          int size, int delta) {
    data[0] = new StringBuffer("Tom");
    data[1].append("131");
    size = 20;
    delta += 1000;
    /* HERE */
}

public static void main(String[] args) {
    StringBuffer[] all = new StringBuffer[2];
    int length = 10;
    int[] col = {4, 500, 7};

    all[0] = new StringBuffer("Dora");
    all[1] = new StringBuffer("Elsa");

    process(all, length, col[1]);
}

```



## Problem #3 (Class Implementation)

Complete the implementation of a class called **Account** that represents a bank account. A bank account has an id, a password, a balance, and an array of integers (deposits) representing deposits done to the account. The maximum size of the array is `MAX_DEPOSITS` and the current number of deposits made is represented by `numDeposits`. Below we have provided a driver that illustrates the functionality associated with the class. Feel free to ignore it if you know what to implement. The driver relies on methods (e.g., `toString()`) you don't need to implement. You **MAY NOT** add any methods beyond the ones specified below (not even private); if you do you will lose credit.

```
public class Account {
    public static final int MAX_DEPOSITS = 100;
    private StringBuffer password;
    private int id, balance, numDeposits;
    private int[] deposits;
```

```
Account a1 = new Account(1, "terpsrock", 20);
```

```
a1.deposit(5).deposit(10).deposit(5);
System.out.println("At Least: " + a1.atLeastTwoDepositsSame());
System.out.println(a1);
```

```
At Least: true
```

```
Id: 1, Passwd: terpsrock, Balance: 40
```

```
Num Dep: 3, Deposits: (5)(10)(5)
```

1. Define a **constructor** that initializes the account with the provided id, password and initial balance. The constructor will create an array of integers with a size corresponding to `MAX_DEPOSITS`. The initial number of deposits is zero. The password parameter is a string, but the class represents it using a `StringBuffer`. The method will not perform any processing and throw an `IllegalArgumentException` with the message "Invalid param" if balance is negative, the password reference is null or the length of the password is less than 8.

Answer:

```
public Account(int id, String password, int balance) {
    if (balance < 0 || password == null || password.length() < 8) {
        throw new IllegalArgumentException("Invalid param");
    }
    this.id = id;
    this.password = new StringBuffer(password);
    this.balance = balance;
    deposits = new int[MAX_DEPOSITS];
    numDeposits = 0;
}
```

2. Define a **constructor** that takes an id as parameter. The constructor initializes the object with the provided id, with "123" as the password value and 10 as the initial balance. You must call the previous constructor in order to implement this constructor, otherwise you will not get any credit.

Answer:

```
public Account(int id) {
    this(id, "123", 10);
}
```

3. Define a **getPassword** method that returns the password value. The method must not generate a privacy leak.

Answer:

```
public StringBuffer getPassword() {
    return new StringBuffer(password);
}
```

4. Define a method called **deposit** that processes a deposit. A deposit will add the specified amount at the next available position in the **deposits** array. For example, if no deposits have been made, the first available entry is the one with index 0. In addition, the balance will be increased by the specified amount. The **numDeposits** variable keeps track of the number of deposits. The method returns a reference to the current object. The method will not perform any processing if the **deposits** array is full of deposits (MAX\_DEPOSITS value) or if the amount is less than or equal to 0.

Answer:

```
public Account deposit(int amount) {
    if (numDeposits < MAX_DEPOSITS && amount > 0) {
        deposits[numDeposits++] = amount;
        balance += amount;
    }
    return this;
}
```

5. Define a method called **atLeastTwoDepositsSame** that returns true if at least two deposits with the same amount has been made and false otherwise.

```
public boolean atLeastTwoDepositsSame() {
```

Answer:

```
public boolean atLeastTwoDepositsSame() {
    for (int i = 0; i < numDeposits - 1; i++) {
        int count = 1;
        for (int k = i + 1; k < numDeposits; k++) {
            if (deposits[i] == deposits[k]) {
                count++;
                if (count == 2) {
                    return true;
                }
            }
        }
    }
    return false;
}
```