



University of Maryland College Park

Dept of Computer Science

CMSC131

Final Exam

Last Name (PRINT): _____

First Name (PRINT): _____

University Directory ID (e.g., umcpturtle) _____

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

Your signature: _____

Instructions

- This exam is a closed-book and closed-notes exam.
- Total point value is 200 points.
- The exam is a 120 minutes exam.
- Please use a pencil to complete the exam.
- WRITE NEATLY.
- There are **seven** problems on the exam.
- You do not need any import statements for coding questions.
- You must write code that is efficient and that avoids code duplication.

Grader Use Only

Problem #1 (Miscellaneous)	(38)	
Problem #2 (Memory Map)	(20)	
Problem #3 (One-Dimensional Arrays)	(16)	
Problem #4 (Recursion)	(20)	
Problem #5 (Class Implementation)	(64)	
Problem #6 (Two-Dimensional Arrays)	(16)	
Problem #7 (Additional Class Methods)	(26)	
Total	(200)	

Problem 1 (Miscellaneous)

1. (2 pts) What is the default value for a numeric instance variable?
 - a. false
 - b. 0
 - c. null
 - d. None of the above.
2. (2 pts) Which of the following is a method of the Object class that we usually override while implementing a class:
 - a. toString()
 - b. System.out.println()
 - c. Integer.parseInt()
 - d. None of the above.
3. (2 pts) In Java we create objects using new. Which Java component is in charge of collecting/reclaiming the memory associated with an object once it is no longer used?
 - a. The constructor
 - b. Garbage collector
 - c. Stack
 - d. None of the above.
4. (2 pts) Can the following method be defined as static?

```
public void printWarning(String m) {  
    System.out.println(m);  
}
```

- a. Yes, as it makes no reference to instance variables.
 - b. No, it needs to be non-static method otherwise we will not be able to print the value of m.
 - c. None of the above.
5. (2 pts) The following program compiles. Do you see any problems?

```
public class ComparingExample {  
    public static void main(String[] args) {  
        float v1 = .1f, v2 = .08f + .02f;  
        boolean result = (v1 == v2);  
        System.out.println(result);  
    }  
}
```

- a. There are no problems.
 - b. We should not use == to compare floating point values.
 - c. We should not initialize boolean variables at declaration time.
 - d. None of the above.
6. (2 pts) How many objects are present in the following code fragment?

```
String str = "Hello";  
int y;  
String p = str;  
StringBuffer m;
```

How many? _____

7. (2 pts) The static main function of a class has a String array as parameter. We can use that array to:
- To initialize the stack.
 - To provide command line arguments.
 - To check whether the stack collector is working.
 - None of the above.

8. (4 pts) Which values for **FIRST_TYPE** and **SECOND_TYPE** make the following two methods to be in an **OVERLOADING** configuration:

```
public int toCelsius(double t) { // body }  
  
public FIRST_TYPE toCelsius(SECOND_TYPE t) { // body }
```

- FIRST_TYPE** → int **SECOND_TYPE** → int
 - FIRST_TYPE** → int **SECOND_TYPE** → double
 - FIRST_TYPE** → void **SECOND_TYPE** → double
 - None of the above.
9. (4 pts) Which values for **FIRST_TYPE** and **SECOND_TYPE** make the following two methods to be in an **OVERRIDING** configuration:

```
public int toCelsius(double t) { // body }  
  
public FIRST_TYPE toCelsius(SECOND_TYPE t) { // body }
```

- FIRST_TYPE** → int **SECOND_TYPE** → int
 - FIRST_TYPE** → int **SECOND_TYPE** → double
 - FIRST_TYPE** → void **SECOND_TYPE** → double
 - None of the above.
10. (4 pts) Rewrite the following if statement using a single statement that uses the ternary operator (? :).

```
int x = 10;  
String ans;  
  
if (x > 10) {  
    ans = "Yes";  
} else {  
    ans = "No";  
}
```

11. (4 pts) For the following code:

```
public static void process(int x, int y) {  
    if (x >= 10 || ++y == 5) {  
        x += 100;  
    } else {  
        y += 200;  
    }  
    System.out.println(x + " " + y);  
}
```

What is the output for each of the following method calls?

- process(11, 5); **Output:**
- process(2, 8); **Output:**

12. (8 pts) Rewrite the if statement using a switch statement. You must have the minimum number of statements possible.

```
if (x == 10) {  
    y = 1;  
} else if (x == 20 || x == 40) {  
    y = 7;  
} else {  
    y = 14;  
}
```

Problem 2 (Memory Map)

Draw a memory diagram showing both the stack and the heap at the moment this program reaches the point marked **/* HERE */**.

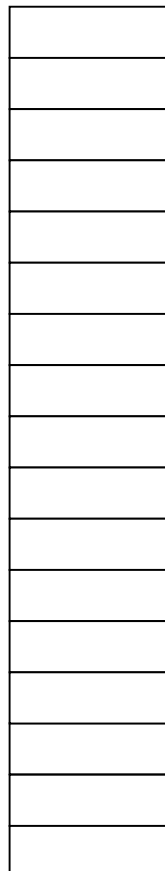
```
public class MemoryMap {
    public static void reduce(int m) {
        if (m > 0) {
            reduce(m - 1);
        }
    }

    public static void process(String[] persons, int unit) {
        int delta = 4;

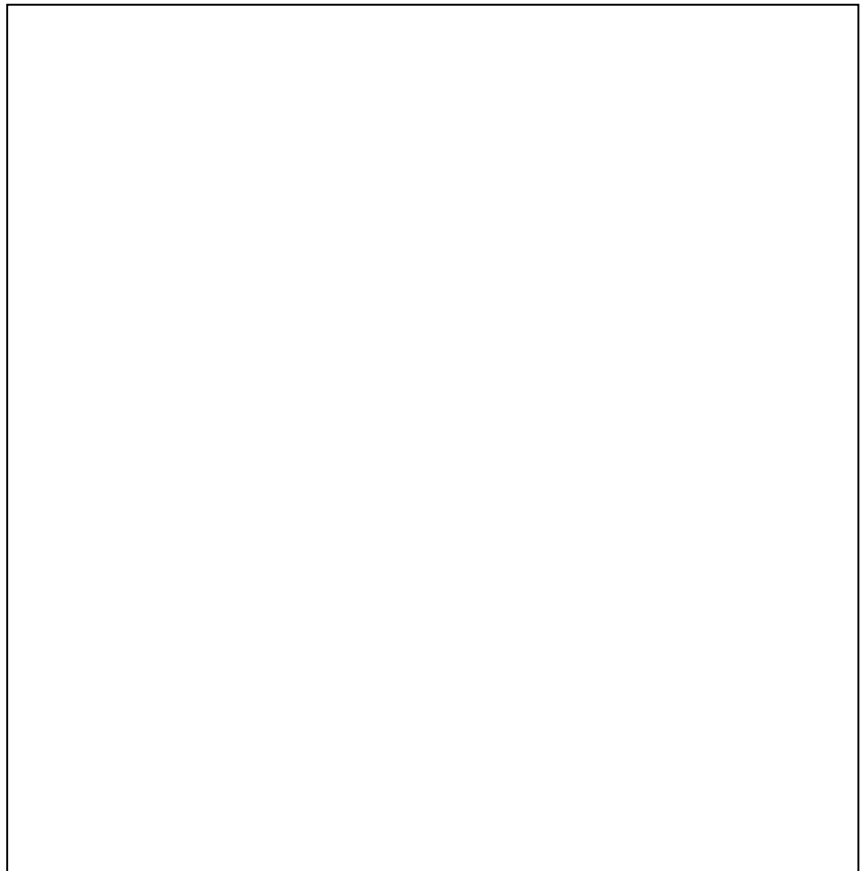
        unit += delta;
        persons[2] = "Linda";
        persons = null;
        int k = 2;
        reduce(k);
        /* HERE */
    }

    public static void main(String[] args) {
        String[] members = {"John", "Mike", null};
        int max = 100;
        process(members, max);
    }
}
```

Stack



Heap



Stack Bottom

Problem 3 (One-Dimensional Arrays)

Write a method called **fourOfAKind** that returns true if the array parameter has at least four values that are the same.

```
public static boolean fourOfAKind(int[] data) {
```

Problem 4 (Recursion)

Implement a recursive method called **replace** that returns a string where all instances of the character **target** in the string **str** has been replaced with the character **replacement**. If you use any iteration statement (e.g., for, while, do while) you will not receive any credit for this problem. Feel free to use the String class **substring** method. We included some information about this method in case you need it.

From Java API:

public [String](#) substring(int beginIndex)- Returns a string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

```
public static String replace(String str, char target, char replacement)
```

Problem 5 (Class Implementation)

Read all of the problem carefully before you start.

```
public interface MutableInt {  
    // returns the current value  
    public int getValue();  
  
    //change the stored value, returns the value it held prior to this change  
    public int setValue(int newValue);  
}
```

You will implement a class called **BoundedMutableInt** that implements the interface **MutableInt**. A specific **BoundedMutableInt** object has two integers: an integer value (current value) and the maximum value this integer value can assume. The current value can be changed by using the method **setValue()**.

1. **Constructor** - Takes two integer parameters: an initial value for the object and the maximum value the object is ever allowed to store. The maximum value associated with the object will be initialized with the maximum parameter. The current value associated with the object will be initialized with the initial value parameter. If the initial value parameter is greater than the maximum value, the constructor will throw an `IllegalArgumentException` with the message "Invalid maximum", and the object will not be considered in the total count returned by the method **getNumCreated()**.
2. **Default Constructor** - Initializes a **BoundedMutableInt** object with the values 0 and a maximum value of 100. Do not use "this" to call the previous constructor.
3. **Copy Constructor** - Makes an exact copy of the object being passed in. Do not use "this" to call another constructor.
4. **getBound** - A getter that will return the maximum value this object is ever allowed to have.
5. **getNumCreated** - A static method that will return the total number of **BoundedMutableInt** objects that have been created.
6. **setValue** - Takes an integer as a parameter. This method will update the current value with the parameter value. If the parameter is greater than the maximum value, the constructor will throw an `IllegalArgumentException` with the message "Invalid maximum" and perform no further processing.
7. **Possibly other methods(s)** - Implement (in a simple and reasonable way) any method(s) that are required because of the fact that this class implements **MutableInt**. Note: You will lose points if you implement methods that are not necessary.

PAGE FOR YOUR ANSWERS

PAGE FOR YOUR ANSWERS

Problem 6 (Two-Dimensional Arrays)

Write a method called **getIndexOfRowWithProduct** that returns the **index** value of the first **row** in a two-dimensional array of integers where the product of elements in the row is the same as the **product** parameter. For example, calling **getIndexOfRowWithProduct(data, 48)** with a **data** array that has the values

{{2, 4, 5}, {100}, {10, 30, 40, 50}, {12, 2, 2, 1}, {10, 20}}

will return **3** (index of the row with elements **{12, 2, 2, 1}**). For this problem you can assume **data** will never be null. If there are two or more rows with the same product return the row with the lowest index. Notice that rows can have different lengths. The method will return -1 if there is no row with the specified **product** value.

```
public static int getIndexOfRowWithProduct(int[][] data, int product) {
```

Problem 7 (Additional Class Methods)

This problem expands the functionality associated with the **BoundedMutableInt** class defined in Problem #5.

1. Provide an **equals** method for the **BoundedMutableInt** class. Two objects are considered equal if they have the same current value and the same maximum value.
2. Assume we would like to sort an ArrayList of **BoundedMutableInt** objects using **Collections.sort()**. The result of the sorting process will place objects with low current values before objects with higher current values. Which changes would you need to do to the **BoundedMutableInt** class and which method(s) would you need to implement? Provide those changes and method(s) here (not on Problem #5). You don't need to rewrite the solution you already provided in Problem #5.
3. Implement the method **getObjsMaximum** that returns an ArrayList of objects that are of type **BoundedMutableInt** and that have a maximum (bound value) equal to **max**. Notice that the parameter is an ArrayList of **MutableInt**.

```
public static ArrayList<BoundedMutableInt> getObjsMaximum(ArrayList<MutableInt> lst, int max)
```

PAGE FOR YOUR ANSWERS

PAGE FOR YOUR ANSWERS

PAGE FOR YOUR ANSWERS