



# University of Maryland College Park

## Department of Computer Science

### CMSC131 Fall 2019

### Exam #3

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 (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.

#### Grader Use Only

#1	Problem #1 (Memory Map)	40	
#2	Problem #2 (Recursion)	30	
#3	Problem #3 (Arrays)	130	
<b>Total</b>	Total	200	



## Problem #2 (Recursion)

1. Implement a **RECURSIVE** method called **printReverse** that prints a string in reverse order. For example, calling **printReverse("Kelly")** will return **y11eK**. If you use any loop construct (e.g., while, do while, for) or add any auxiliary method you will automatically receive 0 credit. You can assume the **str** parameter will never be null.

```
public static void printReverse(String str)
```

2. Implement a **RECURSIVE** method called **dupChar** that returns a new string where the **target** character parameter has been duplicated in the **str** parameter. For example, calling **dupChar("enter", 'e')** will return **eenteer**. If you use any loop construct (e.g., while, do while, for) or add any auxiliary method you will automatically receive 0 credit. You can assume the **str** parameter will never be null.

```
public static String dupChar(String str, char target)
```

DirectoryId:

### Problem #3 (Arrays)

Complete the implementation of the class **Train** that represents a train. A train is associated with an array of **RailCar** objects (**cars** instance variable) and the number of railcars (**numCars** instance variable) it has. A **RailCar** object has a maximum number of passengers it can carry (**maxCapacity** instance variable), the current number of passengers (**numPassengers** instance variable) and a **StringBuffer** (**passengers** instance variable) that will hold the passengers' names. A passenger will be added to a railcar only if there is a space. For this problem you MAY not modify the **RailCar** class. 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 RailCar {
    private int maxCapacity, numPassengers;
    private StringBuffer passengers;

    public RailCar(int maxCapacity) {
        this.maxCapacity = maxCapacity;
        numPassengers = 0;
        passengers = new StringBuffer();
    }

    public boolean addPassenger(String name) {
        if (numPassengers < maxCapacity) {
            numPassengers++;
            passengers.append(name);
            return true;
        }
        return false;
    }

    public int getNumPassengers() { return numPassengers; }
    public StringBuffer getPassengers() { return new StringBuffer(passengers); }
    public String toString() { return numPassengers + ", " + passengers; }
}

/* Train Class */
public class Train {
    private RailCar[] cars;
    private int numCars;
}
```

#### Driver

```
int numCars = 6, maxCapacity = 2;
Train train = new Train(numCars, maxCapacity);

train.addPassenger("Tom", 1);
train.addPassenger("Kelly", 1);
train.addPassenger("Rose", 0);
train.addPassenger("Mary", 3);

System.out.println(train);
train.removeEmptyCars();
System.out.println("After removing empty cars");
System.out.println(train);
```

#### Output

```
Car: #0-Rose
Car: #1-TomKelly
Car: #2-
Car: #3-Mary
Car: #4-
Car: #5-

After removing empty cars
Car: #0-Rose
Car: #1-TomKelly
Car: #2-Mary
```

1. Is there a privacy leak in the following `getCars()` method if it is added to the **Train** class? Yes or No.

```
public RailCar[] getCars() { return cars; }
```

2. Define a **constructor** that takes as parameters two integer parameters called **numCars** and **maxCapacity**. The constructor will initialize the **cars** instance variable with an array of **RailCar** objects that has a size corresponding to the **numCars** parameter. Each railcar will have a capacity that corresponds to the **maxCapacity** parameter. If the **numCars** and/or **maxCapacity** parameters are less than one, the constructor will throw an `IllegalArgumentException` with the message "Error". Make sure you initialize any other instance variable accordingly.

DirectoryId:

3. Define a **default constructor** that initializes a train with 2 railcars where each has a maximum capacity of 10 passengers. You must call the previous constructor in order to implement this constructor, otherwise you will not get any credit.

4. Define a method called **addPassenger** that adds a passenger with a specified name to the train. The method takes two parameters: a string called **name** (passenger's name) and an integer called **carIndex**. If the **carIndex** is different than -1, the passenger will be added to the railcar that has an index corresponding to **carIndex** (e.g., the first car has a **carIndex** value of 0). For this case you can assume there is a car with that index value. If the **carIndex** is -1, the passenger will be added to the first railcar that can fit the passenger. The method will return -1 if the specified railcar (case where **carIndex** is different than -1) is full, or if there is no railcar to which the passenger can be added (case where **carIndex** is -1); otherwise the method will return the index of the railcar where the passenger was added.

5. Define a method called **removeEmptyCars** that updates the **cars** array instance variable with an array where railcars that have no passengers have been removed. You need to create a new array that will only have the railcars that have passengers. You do not need to make copies of the **RailCar** objects. Make sure you initialize any instance variables accordingly. The method will return a reference to the current object.

DirectoryId:

**EXTRA PAGE IN CASE YOU NEED IT (RETURN WITH EXAM)**

**LAST PAGE**