University of Maryland College Park
Dept of Computer Science
CMSC132
Midterm II

Last Name (PRINT): _____

First Name (PRINT): _____

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

Lab TA (Circle One):

0101/0102 Medhi	**0201Jeremy**	**0203/0204 Stefani**	**0303/0304 Raghav**	**0402 Solomon**	**0404 Victor**
0103/0104 Nisarg	**0202 William**	**0301/0302 Meena**	**0401 Avery**	**0403 Jihoon**	**Honors**

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 50 minutes exam.
- Please use a pencil to complete the exam.
- WRITE NEATLY.
- Your code must be efficient.
- You don't need to use meaningful variable names; however, we expect good indentation.
- **You must stop writing when time is up; make sure you write your name and section now.**

Grader Use Only

#1	Problem #1 (Algorithmic Complexity)	30	
#2	Problem #2 (Miscellaneous)	60	
#3	Problem #3 (Linear Data Structures)	110	
	Honors	15	
Total	Total	200/215	

Problem #1 (Algorithmic Complexity)

1. (18 pts) For the following problems you need to provide the asymptotic complexity using Big O notation. In addition, you need to identify the critical section (circle it) and the time function (Time → below). Here is an example:

```
for (j = 1; j <= n; j++) {
    System.out.println(j);
}
System.out.println("Goodbye");
```

Time → n + 1
Big O → O(n)

a. (6 pts)

```
for (m = 1; m <= n / 2; m++) {
    for (t = 1; t <= n; t++) {
        System.out.println(m * t);
    }
}
System.out.println(m * 2);
```

Time →
Big O →

b. (6 pts)

```
for (i = 1; i <= n * 2; i += n * 2) {
    for (k = 1; k <= n; k++) {
        System.out.println(k);
    }
    System.out.println(i);
}
```

Time →
Big O →

c. (6 pts)

```
int i = 1;
while (i <= n) {
    for (int k = 1; k <= n / 4; k++) {
        System.out.println(k);
    }
    i = 2 * i;
}
```

Time →
Big O →

2. (4 pts) List the following Big O expressions in order of asymptotic complexity (lowest complexity first).

$O(\mathbf{n}\log(\mathbf{n}))$ $O(\mathbf{n^n})$ $O(\log\mathbf{(n)})$ $O(\mathbf{n!})$ $O(1)$ $O(\mathbf{n})$

3. (4 pts) Indicate the complexity (Big O) for an algorithm whose running time increases roughly by a constant when input size doubles.

4. (4 pts) Indicate the complexity (Big O) for an algorithm with the following running times:

Size(n)	Running Time
4	16
8	64
16	256

Problem #2 (Miscellaneous)

1. (3 pts) Which component of the Model View Controller Paradigm did you implement for the Blackjack project?

2. (3 pts) The clone method associated with the Object class returns a shallow copy. **True** / **False**.

3. (3 pts) A dealership can upgrade any cars with the following extra (not free) features: spoiler, sunroof, and security system. Any combination of them are possible. Which of the following design patterns is preferred in order to implement a software that allows the dealership to sell customized cars? Circle only one.

    a. Marker design pattern
    b. State design pattern
    c. Decorator design pattern
    d. Iterator design pattern
    e. Pizza decorator pattern

4. (3 pts) The Marker design pattern can be implemented in Java using:

    a. An interface.
    b. A global variable that assumes the different values of interest.
    c. A static function that updates a value that represents the different states.
    d. None of the above.

5. (3 pts) If we do not override the equals and the hashCode method for a class, the Java HashCode contract will be satisfied. **True / False.**

6. (3 pts) A hashCode function returns -2 (negative 2). Which of the following is true?

    a. It is a valid hashCode function and extremely good one.
    b. It is a valid hashCode function and extremely bad.
    c. It is invalid as the value cannot be negative.
    d. None of the above.

7. (3 pts) A checked exception represents an error that a program cannot ignore (must either catch or declare). **True** / **False**.

8. (3 pts) A finally block is only executed when an exception occurs. **True** / **False.**

9. (3 pts) In Java an exception must explicitly be propagated in order for the caller to receive it. **True** / **False.**

10. (3 pts) All problems that we solve using recursion can be solved without recursion. **True** / **False.**

11. (3 pts) A tail recursive solution is difficult to transform to an iterative solution. **True** / **False.**

12. (3 pts) Outer & inner classes can directly access each other's fields and methods (even if private). **True** / **False.**

13. (4 pts) Complete the following declaration so we can define an array of **E** elements with a number of elements that corresponds to **size**.

```
E[] data =
```

14. (20 pts) The **Operation** interface is defined as follows:

```
public interface Operation {
       public double cost(int hours, double difficulty);
}
```

a. (10 pts) Using an **anonymous** inner class, initialize the variable called **simple** with an object that implements the **Operation** interface and defines the **cost** of an operation as the product of **hours** by **difficulty**. For example, calling **simple.cost(2, 4.5)** will return **9.0**.

```
Operation simple =
```

b. (10 pts) Using a lambda expression, initialize the variable **complex** with an object that implements the **Operation** interface and defines the cost of an **Operation** as twice the product of **hours** by **difficulty**. For example, calling **complex.cost(2, 4.5)** will return **18.0**.

```
Operation complex =
```

Problem #3 (Linear Data Structures)

Use the following classes to implement the methods below. You may not add any instance variables nor static variables to either class, you may not add any methods to the Node class, and you may not use the Java API LinkedList class. The size of the list is represented by the **size** instance variable. Each list is associated with a unique name represented by the **name** instance variable.

```
public class LinkedList<T extends Comparable<T>> {
        private class Node {
                private T data;
                private Node next;

                private Node(T data, Node next) {
                        this.data = data;
                        this.next = next;
                }
        }
        private int size;
        private String name;
        private Node head;

        public LinkedList() { name = "NONAME"; }
        public int hashCode() { /* YOU MUST IMPLEMENT */ }
        public LinkedList(ArrayList<T> arrayList, String name) { /* YOU MUST IMPLEMENT */ }
        public void removeOddPosition() { /* YOU MUST IMPLEMENT */ }

}
```

1. (4 pts) Provide an implementation for the **hashCode** method that tries to avoid collisions as much as possible. Each list has a unique name and two lists are considered equal if they have the same name.

2. (80 pts) Provide a **NON-RECURSIVE** implementation for the **LinkedList** constructor that takes an ArrayList and a string (list's name) as parameters. For this problem:

    a. The constructor will initialize a new linked list using the elements from the ArrayList (**in the same order they appear)** removing any duplicates (e.g., a second "cat" string will not be added to the list if we have already seen one).
    b. If the **arrayList** or the **name** parameter (or both) are null, the exception **IllegalArgumentException** will be thrown and no further computation will take place**.** There is no message associated with this exception.
    c. An empty list will be returned if the **arrayList** is empty.
    d. Feel free to use a set in order to identify duplicates.
    e. You will lose credit if your code is not efficient.

3. (26 pts) Provide a **RECURSIVE** implementation of the **removeOddPosition** method. For this problem:
    a. The method will remove all the nodes from the list whose position is associated with an odd number (first node, third node, fifth node, etc.)
    b. You may not create new nodes; if you do you will lose significant credit.
    c. The code must handle the empty list case.
    d. You may only add ONE auxiliary method.
    e. You may not use the previous constructor.
    f. You will lose most of the credit for this problem if you use any iteration statement (i.e., while(), do while, for).

One the next page we provided a driver that illustrates the functionality associated with the methods you need to implement. You can ignore it if you know what to implement. Notice the driver relies on methods you do not need to implement. You may find the following set methods helpful:

boolean **add**(E e)	boolean **remove**(Object o)
boolean **contains**(Object o)	void **clear**()
boolean **isEmpty()**	int **size**()

Driver	Output

`ArrayList<String> arrayList = new ArrayList<String>();`
`arrayList.add("C");`
`arrayList.add("B");`
`arrayList.add("M");`
`arrayList.add("D");`
`arrayList.add("M");`
 	
`LinkedList<String> list = new LinkedList<String>(arrayList, "First");`
`System.out.println("List1: " + list);`
 	
`list.removeOddPosition();`
`System.out.println("List2: " + list);`
	`List1: ListName: First, Size: 4`
	`C B M D`
	`List2: ListName: First, Size: 2`
	`B D`

PAGE FOR PREVIOUS PROBLEM

PAGE FOR PREVIOUS PROBLEM

HONORS STUDENTS (THERE IS A QUESTION ON THE REVERSE SIDE)

HONORS

Students in the Honor's section must answer this question and only students in the Honor's section will receive credit for it.

Provide a **RECURSIVE** implementation for the **getReversed** method that belongs to the **LinkedList** class defined in Problem #3. The method will return a new list with elements from the original list in reverse order. For example, for a list with strings "C" "B" "M" "D", the method will return a new list with strings "D" "M" "B" "C". For this problem:

a. You can use shallow copy for the data component of each node.
b. You may not modify the nodes of the original list.
c. Your code must be efficient.
d. You may only add one auxiliary method.
e. **You will lose most of the credit for this problem if you use any iteration statement (i.e., while(), do while, for).**

```
public LinkedList<T> getReversed(String name)
```