Name: _Christopher Welch_____ Email: cwelch25@msudenver.edu_____

Print legibly!

CS 2050
Computer Science II

Midterm Exam
Spring 2020

Exam instructions (please read them):

- this exam is being applied remotely due to the current Coronavirus crisis;
- at the scheduled time, download the exam from Blackboard;
- you may choose between two formats: PDF or Word;
- plan ahead and decide how are you going to write your answers;
- you can choose to do it directly on your computer (using an appropriate text editor) or to print and write your answers by hand.
- should you choose to write your answers by hand, make sure you reserve time before the exam ends to scan your answers and submit them through Blackboard;
- during the two hours scheduled for the exam you must remain logged in on the Microsoft Teams' platform with your camera (and audio) ON, microphone muted;
- if you can't have a functional camera you will not be allowed to take the exam;
- the instructor must be able to see the student throughout the exam;
- this exam is closed book and notes;
- you are not allowed to use the internet or any external software tools (like IDEs or compilers, for example);
- you are not allowed to use headphones or headsets;
- you are not allowed to use hats, beanies, or hoods;
- you should not seek external help;
- you should not communicate with others during the exam.

Failure to follow these instructions will result in a zero for the exam.

CLOSED BOOK - CLOSED NOTES

YOUR SCORE: _____ POINTS

Q1 [5 points] Consider the following mathematical definition:

$$f(0) = 5$$
$$f(n) = f(n-1) + 2$$

a) implement function `f` recursively using Java.

```
class Q1 {
   static int f(int n) {
        if(n == 0){return 0;}
        else{
             return (f(n-1) + 2);
        }
   }

   static void int main(String args[]) {
      System.out.println("f(3)=" + f(3));
   }
}
```

b) what's the output when program Q1 runs?

9

~~(3?2?1?0 -> 0+2+3+4)~~

Q2 [10 points] Consider the implementation of the `Node` class on the next page.

```
public class Node {

    private int  data;
    private Node next;

    public Node(int data) {
        this.data = data;
        next = null;
    }

    public int getData() { return data; }

    public Node getNext() { return next; }

    public void setData(int data) { this.data = data; }

    public void setNext(Node next) { this.next = next; }

    @Override
    public String toString() { return data + ""; }
}
```

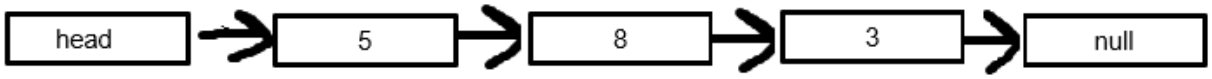
a - 5 points) finish the implementation of the `LinkedList` method `append`.

```
public class LinkedList {

    private Node head;

    public LinkedList() { head = null; }

    public boolean isEmpty() { return head == null; }

    // TODO: add data at the end of the list
        // Check if empty || Find End node & setNext
    public void append(int data) {
         Node newNode = new Node(data);
         if (isEmpty()){head = newNode;}
         else{
              Node current = head;
              while(current.getNext() != null){current = current.getNext();}
              current.setNext(newNode);
         }
    }
 }
```

Consider the `LinkedListDriver` class below.

```
public class LinkedListDriver {

    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        list.append(5);
        list.append(8);
        list.append(3);
    }
}
```

b - 5 points) using boxes to represent nodes and arrows to represent connections between nodes, draw the final state of the linked list after the three additions are executed (and before the program exits). Make sure your drawing shows the `head` reference variable pointing to the appropriate node.



Q3 [20 points] Consider the code for the `Shoes` and `Node` classes described below.

```
public class Shoes {

    private int   size;
    private int   type;
    public static final int TENNIS = 0;
    public static final int DRESS  = 1;

    public Shoes(int size, int type) {
        this.size   = size;
        this.type   = type; // assume that type is always 0 or 1
    }

    public double getSize() { return size; }

    public int getType() { return type; }

    @Override
    public String toString() {
        return "Shoes{" + "size=" + size + ", type=" + type + '}';
    }
}
```

```
public class Node {

    private Shoes shoes;
    private Node  next;

    public Node(Shoes shoes) {
        this.shoes = shoes;
        this.next  = null;
    }

    public Node getNext() { return next; }

    public void setNext(Node next) { this.next = next; }

    public Shoes getShoes() { return shoes; }

    @Override
    public String toString() {
        return shoes.toString();
    }
}
```

a - 5 points) override the `clone` method in the `Shoes` class.

```
    @override
    public Object clone(Shoes shoe) {
         Shoes shoeClone = new Shoes(shoe);
         Node newNodeClone = new Node(shoeClone.getShoe());
    }
```

b - 10 points) complete the `ShoesDisplay` class implementation below by writing the code for the `add` and `total` methods. Note that `ShoesDisplay` maintains two separate lists, one for the tennis shelf, and another for the dress shelf.

```
public class ShoesDisplay {

    private Node tennisShelf;
    private Node dressShelf;

    public ShoesDisplay() {
        tennisShelf = null;
        dressShelf = null;
    }

// TODO: add the given shoes to the correct shelf (head insert is fine)
    public void add(Shoes shoes) {
         if (isEmpty)
         {
              Node newNode = new Node(shoes);       //0 = Tennis
              head = newNode;
              if(shoes.getType == 0) tennisShelf = shoes;
              else{dressShelf = shoes;}
         }
         else{
              Node newNode = new Node(shoes);       //0 = Tennis
              if(shoes.getType == 0) tennisShelf = shoes;
                   else{dressShelf = shoes;}
              newNode.setNext(head);
              head = newNode;
         }
    }

// TODO: return the total number of shoes of the given type
    public int total(int type) {
        int tennisType = 0;
        int dressType = 0;
        Node current = head;
        while(current.getNext() != null){
             if(current.getShoes().getType() == 0){tennisType++;}
             else{dressType++;}
             current = current.getNext();
        }
        if(type == 0){return tennisType;} else{return dressType;}
  }
```

```
    @Override
    public String toString() {
        String out = "Tennis shoes: ";
        Node current = tennisShelf;
        while (current != null) {
            out += current.toString() + " ";
            current = current.getNext();
        }
        out += "\nDress shoes: ";
        current = dressShelf;
        while (current != null) {
            out += current.toString() + " ";
            current = current.getNext();
        }
        return out;
    }
}
```

c - 10) complete the `ShoesDisplayDriver` class implementation below according to the instructions.

```
public class ShoesDisplayDriver {

    public static void main(String[] args) {

        Shoes a = new Shoes(9, Shoes.TENNIS);
        Shoes b = new Shoes(10, Shoes.TENNIS);
        Shoes c = new Shoes(11, Shoes.TENNIS);
        Shoes d = new Shoes(9, Shoes.DRESS);
        Shoes e = new Shoes(11, Shoes.DRESS);

        // TODO: instantiate a "shoes display" list
        ShoesDisplay shoesDisplayList = new ShoesDisplay();

        // TODO: add all shoes into the "shoes display" list
        shoesDisplayList.add(Shoes a);
        shoesDisplayList.add(Shoes b);
        shoesDisplayList.add(Shoes c);
        shoesDisplayList.add(Shoes d);
        shoesDisplayList.add(Shoes e);

        // TODO: print the "shoes display" list
        shoesDisplayList.toString();


        // TODO: print how many shoes are in the tennis shelf displaying
        System.out.println("#tennis shoes = ..."+shoesDisplayList.total(0));

        // TODO: print how many shoes are in the dress shelf displaying
System.out.println("#dress shoes = ..."+shoesDisplayList.total(1));
    }
}
```

Q4 [10 points] Answer the following questions related to stacks and queues.

a - 2 points) if your dynamic queue implementation is based on a linked list data structure, would you rather maintain a reference to the node on the front of the queue, at the rear of the queue, or instead have two references, one to the node on the front and another to the node at the rear (justify your answer)?

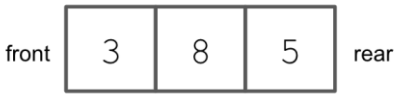
I would rather maintain the reference to both the head and tail. The head reference would be good if you needed some information about the object you are currently dealing with. The tail reference would be good to have so that it is easy to know where to enqueue the objects.

b - 2 points) consider a static stack with a maximum capacity of 3 nodes; what is the stack's configuration after the following code is executed?

```
Stack stack = new Stack(3);
stack.push(1);
stack.push(2);
stack.peek();
stack.push(3);
stack.push(4);
stack.pop();
1,2,3,4 -> 1,2,3
```

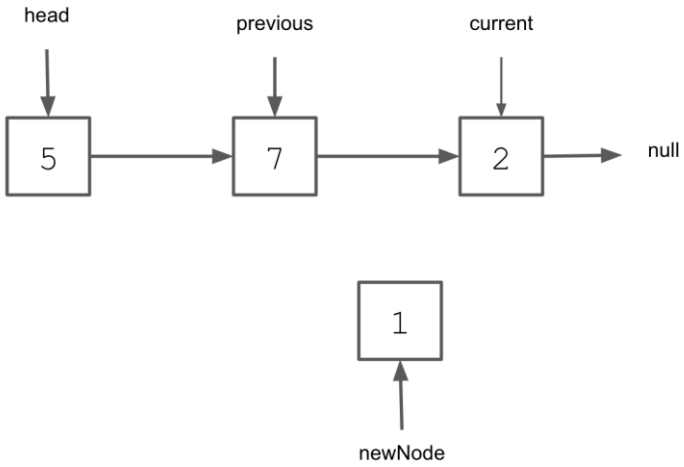
(top)	3	2	1	(bottom)

c - 2 points) write the code to put the queue in the state described by the picture on the left below.




```
Queue q = new Queue();
q.push(3);
q.push(8);
q.push(5);

//If push is not implemented,
//enqueue(num) instead.
```

d - 2 points) consider the linked list on the next page; describe the steps to insert node `1` (pointed by `newNode` reference variable) in between nodes `7` and `2`; assume that `previous` and `current` currently point to nodes `7` and `2`, respectively.



newNode.setNext(current);
previous.setNext(newNode);
current = newNode; //If you need to work with the newNode as current

e - 2 points) give me one good reason to use a static array and another good reason to use a linked list to represent collection of objects.

-Static arrays are one of the fastest because you can access the indexed value in constant time.
-Linked Lists are useful because you don't have to know how many values that you are going to add to your linked lists.