COMPUTER SCIENCE II **MIDTERM REVIEW** LIST

Q1) Consider the following recursive definition for a simple arithmetic progression.

$a_0, a_1, a_2, \ldots a_n$

$a_0 = 0$

$a_n = a_{n-1} \times 2 + 5$, if $n > 0$

Write a recursive function in Java to compute a_n using the definition above.

```
static final int compute(int n) {
    if (n == 0)
        return 0;
    return compute(n-1) * 2 + 5;
}
```

$a_3 = a_2 * 2 + 5 = (a_1 * 2 + 5) * 2 + 5 = ((a_0 * 2 + 5) * 2 + 5) * 2 + 5 = (5 * 2 + 5) * 2 + 5 = 15 * 2 + 5 = 35$

Q2) what is displayed after class A runs?

```
class A {
   private int x;

   A(int x) {
      this.x = x;
   }

   public static void main(String[] args) {
      A a1 = new A(5); // you created an A object
      A a2 = new A(5); // you created another A object
      System.out.println(a1 == a2);
   }
}
```

--> **false**

Q3) ANSWER THE MULTIPLE CHOICE QUESTIONS.

i) operator in Java that is used to ***INSTANTIATE*** an object.

a) **new** **(CORRECT)**

ii) ADT that is characterized by a ***LAST-IN FIRST-OUT*** operating model.

b) **stack** **(CORRECT)**

iii) what's the *output of the snippet* of code below (assume that the `add` method adds a node in the front of the list)?

```
List<String> lst = new LinkedList<>();
lst.add("a");
lst.add("a");
lst.add("b");
lst.add("c");
System.out.println(lst);
```

c) **[c, b, a, a]** **(CORRECT)**

iv) name of the mechanism in Java that allows customizing a class definition using **PLACEHOLDERS** for **types**:

c) **Generics** **(CORRECT)**

Q4) Consider the **Node class** below.

```
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 + ""; }
}
```

> Write the implementation of the `LinkedList` class from scratch.

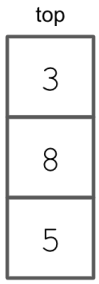
Refer to Activity 06 on GitHub: link here

```
public class LinkedList {

    private Node head;

    public LinkedList() {
        head = null;
    }

    // add in front   -- Add newNode.setNext to head & set head to newNode
    public void add(int data) {
        Node newNode = new Node(data);
        newNode.setNext(head);
        head = newNode;
    }
    // add tail - Check isEmpty & Loop: Find END Node & setNext(newNode)
    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);
        }
    }

    @Override
    public String toString() {
        String out = "";
        Node current = head;
        while (current != null) {
            out += current.toString() + " ";
            current = current.getNext();
        }
        return out;
    }
```

```
public boolean isEmpty() {     //True or False if head is empty
    return head == null;
}

public int size() {  //Keeps count of how many current.getNext's & returns count
    int count = 0;
    Node current = head;
    while (current != null) {
        count++;
        current = current.getNext();
    }
    return count;
}

public int get(int index) {  //Access the LinkedList as an Array, if the Index exists
    if (index < 0 || index >= size())   //Loop until i == Index & return getData();
        return 0;
    int i = 0;
    Node current = head;
    while (i < index) {
        i++;
        current = current.getNext();
    }
    return current.getData();
}

public void set(int index, int data) {  //Check if Index is in bounds
    if (index < 0 || index >= size())
        return;
    int i = 0;
    Node current = head;
    while (i < index) {                 //Loop until i==index & set this.data to data
        i++;
        current = current.getNext();
    }
    current.setData(data);
}

void insert(int index, int data) {     //Insert newNode in bounds of Linked List
    if (index < 0 || index >= size())  //Check if index is in bounds
        return;
    if (index == 0)                    //index = 0? add data
        add(data);
    else {
        Node newNode = new Node(data); //Create newNode with data & loop from head
        int i = 0;                     //until index is reached &
        Node current = head;           //Insert Node within Index
        while (i < index - 1) {
            i++;
            current = current.getNext();
        }
        newNode.setNext(current.getNext());
        current.setNext(newNode);
    }
}
```

```
    void remove(int index) {                //Remove Node @ Index using a 'temp'Node
        if (index < 0 || index >= size())   //Index > size?
            return;
        if (index == 0) {                   // index = head?
            Node temp = head;
            head = head.getNext();
            temp.setNext(null);
        }
        else {
            int i = 0;
            Node current = head;
            while (i < index - 1) {
                i++;
                current = current.getNext();
            }
            Node temp = current.getNext();
            current.setNext(current.getNext().getNext());
            temp.setNext(null);
        }
    }
  }
}
```

Q5) write the code to pull out number 8 out of the stack described by the picture on the left below.




```
Stack st = new Stack();
st.push(5);
st.push(8);
st.push(3);
int val = st.pop(); // pops 3
val = st.pop(); // pops 8!
```