Name _____ Section Number _____

CS210 Exam #2 *** PLEASE TURN OFF ALL CELL PHONES*** Practice
All Sections Bob Wilson

OPEN BOOK / OPEN NOTES
You will have all 90 minutes until the start of the next class period.
Spend only about one minute per point on each question to complete the exam on time.

1. Behavior of Stacks and Queues (20 Points)
a. Show the lines printed by the code below. Draw pictures of the data structure states to be eligible for partial credit. Note: There is a summary of the API for the java.util.Stack class and the java.util.Queue interface on the next to last page of this exam.

```
import java.util.Stack;
import java.util.Queue;
import java.util.LinkedList;

Stack<String> myStack = new Stack<String>();
Queue<String> myQueue = new LinkedList<String>();

myStack.push(“World”);
myStack.push(“Hello”);
myStack.push("Salve");
myStack.push("Mondo");

myQueue.offer(myStack.peek());
myQueue.offer(myStack.pop());
myQueue.offer(myStack.pop());
myQueue.offer(myStack.peek());

myQueue.offer(myStack.peek());

while (!myStack.isEmpty())
   System.out.println(myStack.pop());

while (!myQueue.isEmpty())
   System.out.println(myQueue.poll());
```

2. Performance with Recursion and Tail Recursion (30 Points)
Study the recursive methods below and indicate the performance in big-O notation. Also, indicate if the method exhibits tail recursion or not. Explain your answers.

a. Based on the number n of elements in the linked list O( )

Tail Recursion: Yes ____ No _____

```
public void mysteryPrint(LinearNode<String> next)
{
  if (next != null) {
    mysteryPrint(next.getNext());
    System.out.println(next.getElement());
  }
}
```

b. Based on the number n of elements in the linked list O( )

Tail Recursion: Yes ____ No _____

```
public void mysteryPrint(LinearNode<String> next)
{
  if (next != null) {
    System.out.println(next.getElement());
    mysteryPrint(next.getNext());
  }
}
```

c. Based on the number n O( )

Tail Recursion: Yes ____ No _____

```
public int calculate(int m, int n)    // Assume n > m > 0
{
  if (n > m)
     n = calculate(m, n / 10);
  return m * n;
}
```

3. Collections and Parameterization (20 Points)

For each of the following, indicate if it is (a) valid Java statement(s) or not and explain. (Assume that we have included any needed import statements for the java.util classes.)

Statement(s) Valid(Y/N) Explanation

a. `Stack<String> myStack = new Vector<String>();`

_____ _____

b. `Stack<Vector> myStack = new Stack<Vector>();`

_____ _____

c. `Queue<boolean> myQueue = new LinkedList<boolean>();`

_____ _____

d. In a parameterized collection's method with @SuppressWarnings("unchecked"):
`T [ ] array = (T [ ]) new Object [ ];`

_____ _____

e. `Deque<Integer> myDeque = new Deque<Integer>();`

_____ _____

4. Linked List Application (30 Points)
Write the "Run" portion of this class to solve the Josephus Problem using a singly linked list. The linked list provided is circular representing the circle of soldiers and is NOT null terminated. Note: Because you can't back up easily in a singly linked list, start with a reference to the first soldier and manipulate each soldier object using the reference from its previous soldier object.

```
import java.util.*;

public class Josephus
{
  public static void main(String [] args)
  {
    // get out of static context
    Josephus game = new Josephus();
    game.play();
  }

  private void play()
  {
    // Get the parameters for this game

    Scanner keyboard = new Scanner(System.in);
    System.out.println("Enter the number of soldiers: ");
    int N = keyboard.nextInt();
    System.out.println("Enter the gap between soldiers: ");
    int gap = keyboard.nextInt();
    keyboard.close();

    // Build the Soldier linked list

    Soldier end = new Soldier(N, null); // Start at the end of the number sequence
    Soldier start = end;

    for (int i = N - 1; i > 0; i--)
      start = new Soldier(i, start);
    end.next = start;          // close the loop (end points at the first soldier now)

    // "Run" the simulation of the game (write your code here)









  }
```

Draw a picture of the circular linked list to help write the code.

```
  private class Soldier   // an inner class for the Josephus class
  {
    protected int number;
    protected Soldier next;

    public Soldier(int number, Soldier next)
    {
      this.number = number;
      this.next = next;
    }
  } // end of Soldier class
} // end of Josephus class
```

Printout from Sample Run:

```
> run Josephus
Enter the number of soldiers:
7
Enter the gap between soldiers:
3

3
6
2
7
5
1
Last soldier standing: 4
>
```

The only methods in the java.util.Stack class you should use:

- push is done using:
  `T push(T element)   // pushes and returns the element T`
- pop is done using:
  `T pop()             // pops and removes the element T`
- peek is done using:
  `T peek()            // returns an alias of the element T`
- isEmpty is done using:
  `boolean isEmpty()   // returns true if stack is empty`

The methods in the java.util.Queue interface:

- Enqueue is done using:
  `boolean offer(T element) // returns false if full`
- Dequeue is done using either:
  `T poll()    // returns null value if empty`
  `T remove()  // throws an exception if empty`
- Peek is done using either:
  `T peek()    // returns null value if empty`
  `T element() // throws an exception if empty`
- isEmpty is done using:
  `boolean isEmpty()   // returns true if queue is empty`

Answer Key:

1. Behavior of Stacks and Queues

```
> java StackQueue
Stack contents are:
Hello
World
Queue contents are:
Mondo
Mondo
Salve
Hello
Hello
>
```

2. Performance with Recursion

a. O(n) The recursive call is made once per element in the list.
Tail Recursion: NO. The printing occurs after the recursion returns.

b. O(n) The recursive call is still made once per element in the list.
This method will merely print the list contents in the reverse order of a.
Tail Recursion: YES. The recursive call is the last executable action in the method.

c. O(log n) As each recursive call is made, the value of n is decreased by division by ten.
Hence, the number of times n is divided by 10 to be lower than the value of m is $\log_{10} n$.
Tail Recursion: NO. The local context variables are used in calculation after recursion.
BTW: This method does not calculate any significant function. It is just a recursion.

3. Collections and Parameterization

Statement(s)	Valid(Y/N)	Explanation
`a. Stack<String> myStack = new Vector<String>();`	No	Incompatible types - Stack is a subclass of Vector
`b. Stack<Vector> myStack = new Stack<Vector>();`	Yes	A stack of vector class objects is valid
`c. Queue<boolean> myQueue = new LinkedList<boolean>();`	No	boolean is a primitive data type not a reference data type
d. In a parameterized collection's method with @SuppressWarnings("unchecked"): `T [ ] array = (T [ ]) new Object [ ];`	Yes	This is the required work around for arrays of parameter type T
`e. Deque<Integer> myDeque = new Deque<Integer>();`	No	Deque is only an interface and cannot be instantiated

4. Linked List Application

```
Soldier next = start;
Soldier prev = null;
while (next != next.next) {     // until only one left (Soldier points to himself)
  for (int i = 1; i < gap; i++) {   // skip one less than the gap
    prev = next;
    next = next.next;
  }
  System.out.println(next.number);  // kill this soldier
  prev.next = next.next;            // garbage collect the object
  next = next.next;
}
                                    // and the winner is ...
System.out.println("Last soldier standing: " + next.number);
```

Drawing of the Data Structure