CS171

Quiz 2: Queues (Stacks), Linked Lists, Interface and Iterators

1. (10 points) Which abstract data type, stack or queue, would you choose to implement an "Undo" feature in a word processor? Which would you choose to store mouse click events when implementing a user interface?

2. (10 points) What does the following code fragment print when \( n \) is 10? In general, what does it do when presented with a positive integer \( n \)?

```java
Queue<Integer> q = new Queue<Integer>();
q.enqueue(0);
q.enqueue(1);
for (int i = 0; i < n; i++) {
    int a = q.dequeue();
    System.out.println(a);
    int b = q.peek();
    q.enqueue(a + b);
}
```

3. (10 points) Which of the following is (are) not correct? __________

   A) An interface type specifies all methods required by the interface
   B) An interface type can also have instance variables
   C) Classes can be instantiated (into objects using `new`) while interfaces cannot.
   D) All methods in an interface type cannot have method implementations.
   E) A class can implement more than one interface type
4. (10 points) Consider interface `Measurable` and class `BankAccount` that implements interface `Measurable`. Which of the following statements would generate an error? ______

```java
public interface Measurable {
    double getMeasure();
}

public class BankAccount implements Measurable {
    private double balance;

    public BankAccount() {
        balance = 0;
    }

    public double getMeasure() {
        return balance;
    }

    // additional methods omitted here
}

A) Measurable m = new BankAccount();
B) Measurable m = new Measurable();
C) Measurable m = new BankAccount();
   BankAccount b = (BankAccount) m;
D) BankAccount b = new BankAccount();
```

5. (10 points) If a class implements `Iterable` interface, what can you do with the class objects?
6. (50 points) Consider the program below and answer questions.

class Node
{
    public int iData;  // data item (key)
    public Node next;  // next node in list

    public Node(int id)
    {
        // constructor
        iData = id;
    }
}  // end class Node

class LinkList
{
    private Node first;  // reference to first node on list
    public LinkList()
    {
        // constructor
        first = null;
    }

    public Node find(int key)  // find the node with a given key
    {
        Node current = first;  // start at the first node
        while (current != null && current.iData != key)
        {
            current = current.next;  // go to next node
        }
        return current;
    }

    public void displayList()  // display the list
    {
        for (Node current = first; current != null; current = current.next)
        {
            System.out.println(current.iData);  // print data
        }
    }

    public void insertFirst(int key)  // insert a node at the front of the list
    {
        // fill in your implementation
    }

    public Node delete(int key)  // delete the node with a given key
    {
        // fill in your implementation
    }
}  // end class LinkList

class LinkListApp
{
    public static void main(String[] args)
    {
        LinkList theList = new LinkList();  // create a list
        theList.insertFirst(22);
        theList.insertFirst(44);
        theList.insertFirst(66);
        Node d = theList.delete(44);
        d = theList.delete(88);
        theList.displayList();  // display list
    }  // end main()
}  // end class LinkListApp
(a) (20 points) Implement the `insertFirst` method that inserts a new node with the given data key at the front of the linked list. Assume the list is not empty.

```java
public void insertFirst(int key)
```

(b) (20 points) Implement the `delete` method that deletes a node with the given data key. Assume the list is not empty. If the key does not exist in the list, the method should return null and should not change the list.

```java
public Node delete(int key)
```

(c) (10 points) Predict the printed results of the main method in `LinkListApp`