

1. Matching Terms [4 pts]

Select the **best** definition for each of the words below by writing the appropriate letter in the blank beside the word.

1. _____ Dynamic Data Structure
2. _____ Static Variable
3. _____ Stack
4. _____ Interface

- A.** I have no clue.
- B.** The process by which method calls are resolved at runtime based upon the type of the actual object present, not the declared type of the reference.
- C.** A FIFO data structure.
- D.** A data structure which does not have a fixed limit on how many items it can hold- only constrained by available system resources.
- E.** A variable whose value cannot be changed once it is initially set.
- F.** Specifies a set of methods that classes must have to implement it.
- G.** A variable which each object has its own “copy” of the variable.
- H.** A data structure which has a fixed limit on how many items it can hold. Once an instance is created, it can never be expanded.
- I.** A LIFO data structure.
- J.** A class which has some methods declared but not defined- cannot be instantiated.
- K.** A variable that is an attribute of the class in general, not of any particular instance. Only one “copy” of this variable is made, no matter how many instances are created.
- L.** A FIMO data structure.

2. Dynamic Binding Tracing [15 pts]

Given the following code:

```
class Parent_2_1 {
    public void say(String msg) {
        System.out.println("Simon says "+ msg);
    }//say
    public void say(int x) {
        System.out.println("The magic number is "+ x);
    }//say
}//class Parent_2_1
public class Tracing_2_1 extends Parent_2_1 {

    public void say(String msg) {
        System.out.println("I have "+msg);
    }//say
    public static void main(String[] args) {
        Tracing_2_1 a= new Tracing_2_1();
        Parent_2_1 b= new Parent_2_1();
        Parent_2_1 c= new Tracing_2_1();
        System.out.println("here we go!");
        a.say("hello");
        a.say(7);
        b.say("world");
        b.say(13);
        c.say("i love cs1322");
        System.out.println("what now?");
        ((Parent_2_1)a).say("huh?");
        Tracing_2_1 d=(Tracing_2_1) a;
        a.say("all done");
    }//main
}//class Tracing_2_1
```

What is the output when the main method is run?

3. Polymorphism [18 pts]

Given the following inheritance structure:

- The abstract class **Food**
- The interface **ToppableType** has the method **public void addTopping();**
- **Pizza** extends **Food** implements **ToppableType**
- **Tomato** extends **Food**
- **StuffedCrustPizze** extends **Pizza**

You may assume that each class has a default constructor. Determine whether or not each of the following code fragments are legal. If the fragment is legal, write **OK**, otherwise write **ERROR**. *Note: for this question, you will be penalized for guessing.* On each part

- For a right answer: you will receive the points for that part
- For a wrong answer: you will lose the points for that part
- For no answer (left blank): you will neither receive nor lose points for that part

Your score for the question will not, however, be below zero.

- 1 (a) ToppableType a;
- 1 (b) Food b;
- 2 (c) Food c = new Food();
- 2 (d) Pizza d = new Pizza();
- 2 (e) ToppableType e = new ToppableType();
- 3 (f) Food f = new Pizza();
- 3 (g) ToppableType g = new Tomato();
- 3 (h) ToppableType h = new StuffedCrustPizze();

4. Static vs Instance [10 pts]

A misguided java programmer wrote the following code:

```
public class Misguided2 {
    private static int favoriteNumber;
    private int serialNo;
    private int countMade=0;
    public Misguided2(int x) {
        favoriteNumber=x;
        countMade++;
        serialNo=countMade;
    } //Misguided1(String)
    public int getFavoriteNumber() {
        return favoriteNumber;
    } //getName()
    public int getSerialNo() {
        return serialNo;
    } //getCountMade()
    public static void main(String[] args) {
        Misguided2 a=new Misguided2(74);
        Misguided2 b=new Misguided2(78);
        Misguided2 c=new Misguided2(93);
        System.out.println("a's favorite number "+ a.getFavoriteNumber());
        System.out.println("b's favorite number "+ b.getFavoriteNumber());
        System.out.println("c's favorite number "+ c.getFavoriteNumber());
        System.out.println(a.getSerialNo());
        System.out.println(b.getSerialNo());
        System.out.println(c.getSerialNo());
    } //main(String[])
} //end class Misguided1
```

He expected the output to be

```
a's favorite number 74
b's favorite number 78
c's favorite number 93
1
2
3
```

But instead, the output was

```
a's favorite number 93
b's favorite number 93
c's favorite number 93
1
1
1
```

5 (a) Explain the error(s) in the above code.

5 (b) Fix the error(s) in the code, by clearly marking your change(s) in the code provided.

5. Queues[8 pts]

Assume that you are given a correctly working `LinkedList` class which has the following methods (which behave as in P2):

- `public void addToFront(Object o)`
- `public void addToBack(Object o)`
- `public Object removeFromFront()`
- `public Object removeFromBack()`

You may not assume that there are any other public methods or variables in the `LinkedList` class. You are given the task of implementing a `Queue`. Below, you are given the skeleton of the `Queue` class, in which you should write the methods **`dequeue`** and **`enqueue`** to appropriately implement this data structure.

```
public class Queue {
    LinkedList myList;
    public Queue() {
        myList = new LinkedList();
    }
    public Object dequeue() {
//CODE THIS METHOD

}

public void enqueue(Object o) {
//CODE THIS METHOD

}
}
```

For Questions 6,7, and 8:

You may assume that you are given an LLNode class as follows:

```
public class LLNode {
    private Object data;
    private LLNode next;
    public LLNode() { data = null; next = null; }
    public LLNode(Object data) { this.data = data; next = null; }
    public LLNode(Object data, LLNode next) {
        this.data = data;
        this.next = next;
    } //LLNode(Object LLNode)
    public void setData(Object d) { data = d; }
    public void setNext(LLNode n) { next = n; }
    public Object getData() { return data; }
    public LLNode getNext() { return next; }
} //end class LLNode
```

You may also assume that the LinkedList class that you are writing methods in has the following skeleton:

```
public class LinkedList {
    LLNode head;
    // YOUR CODE WOULD GO HERE
} //end class LinkedList
```

You may not make assumptions about any other methods existing in the LinkedList class.

6. LinkedList coding - Iterative [15 pts]

Using iteration only, write the method **public void remove(Object o)** which removes the first occurrence of data equivalent to **o** from the list. For this method, you may **ONLY** use iteration. Failure to use iteration, or any use of recursion will result in zero credit.

```
public void remove(Object o) {
```

```
}
```

7. LinkedList coding - Recursive [15 pts]

Using recursion only, write the method `public int countEqual(Object o)` which counts the number of nodes with data equivalent to `o` from the list. The count of matching occurrences should be returned. For this method, you may ONLY use recursion. Failure to use recursion, or any use of iteration will result in zero credit. You may write any helper methods that you desire. Your method may not have side effects upon the list.

```
public int countEqual(Object o) {
```

```
}
```

8. LinkedLists - equals [15 pts]

Write the **equals** method for the `LinkedList` class. Your method should compare the two lists to determine if they contain equivalent data in the same order. If the reference passed in is not in fact a list, your method should return *false*. If the two list contain equivalent data in the same order, your method should return *true*. If the lists contain different data, or the data is in a different order, your method should return *false*. Your method should NOT have side effects on the list. You may accomplish this via either iteration or recursion, as you desire. You may write any helper methods that you wish.