CSCI 2341 Sample Midterm

1. **[10]** The following method shows a typical use of an object of type **MyADT**. Write an interface for **MyADT**.

```
public static void doThis(MyADT obj) {
    obj.showValues();
    if (obj.hasZeroValue()) {
        int zero = obj.getZeroLocation();
        double val = obj.solve(zero);
        System.out.println("solution @" + zero + " = " + val);
    }
}
```

public interface MyADT {

public void showValues();

public boolean hasZeroValue();

public int getZeroLocation();

public double solve(int n);

}

2. **[10]** Use the space provided to show how the stack evolves when we translate the following infix expression into a postfix expression. The first space is for the initial stack, the next for the stack after the 1 has been processed, the third for the stack after the + has been processed, and so on. **Also** show the expression that results in the space below the table. You do **not** need to evaluate the expression. **Fill in every cell where the stack is not empty.**

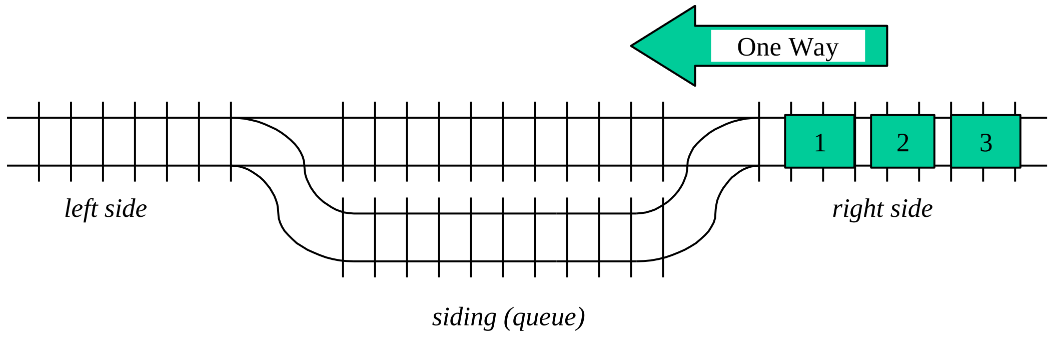
											+	+			
				/	/	*	*		(	(	(	(			
		+	+	+	+	+	+	-	-	-	-	-	-	+	+
	1	+	2	/	3	*	4	–	(	5	+	6	)	+	7

1 2 3 / 4 * + 5 6 + - 7 +

3. **[8]** Write the method getFrequency for the LinkedBag class (see the partial definition on the extra code page). It counts how many times a given entry appears in the **LinkedBag** object. Do not use any undefined methods, nor any methods that need to be imported, and do not modify the contents of the bag in any way. **Note that this version of LinkedBag does not have a numEntries field.**

```
public int getFrequency(T item) {
        int count = 0;
        Node cur = first;
        while (cur != null) {
                if (cur.data.equals(item)) {
                        ++count;
                }
                cur = cur.next;
        }
        return count;
}
```

4. **[6]** Consider the following railway switching yard.



As the diagram suggests, a car can be moved directly across to the left side, or it may be shunted onto the siding. The siding operates the way a queue does: putting a car onto the siding is an enqueue operation, and removing one from the siding is a dequeue operation. *Once a car has been moved off the right-hand track, we do not allow it to return to the right-hand track.*

For each possible permutation of three cars (listed below), show a sequence of Enqueue, Dequeue and Move (going straight across) operations that will achieve that permutation on the left-hand side. If a permutation is not possible, simply write "Cannot be done" in the space provided.

a) 1, 2, 3 **Move Move Move**

b) 1, 3, 2 **Move Enqueue Move Dequeue**

c) 2, 1, 3 **Enqueue Move Dequeue Move**

d) 2, 3, 1 **Enqueue Move Move Dequeue**

e) 3, 1, 2 **Enqueue Enqueue Move Dequeue Dequeue**

f) 3, 2, 1 **Cannot be done**

5. **[8]** Write the definition of the generic method maxElement. The method is given an array and returns the largest element in that array. It should work with any type of array with elements that can be sorted. Do not use any methods that need to be imported, and do not modify the contents of the array in any way. Assume that the array has at least one element.

```
public static <T extends Comparable<? super T>> maxElement(T[] arr) {

    T max = arr[0];

    for (T item : arr) {    // OR replace this one line with the two below

        if (max.compareTo(item) < 0) {

            max = item;

        }

    }

    return max;

}


    for(int i = 0; i < arr.length; ++i) {

        T item = arr[i];
```

6. **[12]** Multiple Choice: select the *best available* answer from the options shown.

- We should use a linked implementation instead of an array implementation if
  - a) the list will generally be processed from back to front.
  - b) the list will generally be processed from front to back.
  - **c) there will be many deletions/insertions in the middle of the list.**
  - d) we need to access random elements of the list quickly.
  - e) (any of these is a good reason to prefer a linked implementation)

- If we have a Measurable variable m, and we want to check to see if it holds a Circle object, the way to do that is
  - a) `if (Circle extends Measurable)`
  - b) `if (Circle implements Measurable)`
  - c) `if (Circle instanceof Measurable)`
  - d) `if (m extends Circle)`
  - **e) `if (m instanceof Circle)`**

- The class SpecialBox has all the methods defined in the class Box. The declaration of SpecialBox will start with the line
  - a) `public class SpecialBox {`
  - **b) `public class SpecialBox extends Box {`**
  - c) `public class SpecialBox implements Box {`
  - d) `public class SpecialBox instanceof Box {`
  - e) `public class SpecialBox isa Box {`

- What is the output of the following code, assuming that setColor and getColor do as their names imply?

```
Car myCar = new Car("Blue");
Car disCar = myCar;
Car stevesCar = new Car("Green");
disCar.setColor("Red");
myCar.setColor("Yellow");
System.out.print(disCar.getColor());
```

  - a) Blue
  - b) Green
  - c) Red
  - **d) Yellow**
  - e) (it's impossible to tell from the given information)

- If the method `method` may result in the exception `MyException` being thrown, then the method header should be
  - a) `public void method() catch MyException`
  - b) `public void method() throw MyException`
  - c) `public void method() throw new MyException()`
  - **d) `public void method() throws MyException`**
  - e) `public void method() throws new MyException()`

- A variable that's declared static is
  - a) available to be changed by methods in other classes.
  - b) copied to each instance of the class it's declared in.
  - c) never going to have its value changed.
  - d) safe from being changed by methods in other classes.
  - **e) shared by all objects of the class it's declared in.**

- The Comparable<T> interface requires the ___ method to be defined:
  - a) boolean compareTo(T one, int result)
  - b) boolean compareTo(T one, T other)
  - c) boolean compareTo(T other)
  - d) int compareTo(T one, T other)
  - **e) int compareTo(T other)**

- Among the reasonable options for dealing with a possible operational failure in an ADT operation is **NOT**:
  - a) have the method return a boolean value: true for success, false for failure.
  - **b) print an error message.**
  - c) return a special value (such as null).
  - d) throw an exception.
  - e) (all of the above are reasonable options for dealing with failure)

```
public class LinkedBag<T> implements BagInterface {

    private Node first;

    private class Node {

        private T data;
        private Node next;

        public Node(T data) {
            this(data, null);
        }

        public Node(T data, Node<T> next) {
            this.data = data;
            this.next = next;
        }
    }

    public LinkedBag() {
        first = null;
    }

    // more methods not shown
}
```

(You may tear this page off your test booklet.)