INSTRUCTIONS TO CANDIDATES

Answer four questions.

If you attempt to answer more questions than the required number of questions (in any section), the marks awarded for the excess questions answered will be discarded (starting with your lowest mark).
1. A Maude specification of an abstract data type of Binary Trees of integers (which we just call ‘Trees’) is given below. The abstract data type includes operations to compute the sum of all the integers in a tree, and to find the leftmost integer in a tree (i.e., the highest integer with no left subtree, and returning -1 if the tree is empty).

\texttt{mod TREES is}
\texttt{
\hspace{1em} protecting INT .
\hspace{1em} sort Tree .
\hspace{1em} op null : -> Tree [ ctor ] .
\hspace{1em} op \_/ \_/ : Tree Int Tree -> Tree [ ctor ] .
\hspace{1em} op sum : Tree -> Int .
\hspace{1em} op leftmost : Tree -> Int .
\hspace{1em} var I : Int .
\hspace{1em} vars T1 T2 : Tree .
\hspace{1em} eq sum(null) = 0 .
\hspace{1em} eq sum(T1 / I \ T2) = sum(T1) + I + sum(T2) .
\hspace{1em} eq leftmost(null) = -1 .
\hspace{1em} eq leftmost(null / I \ T2) = I .
\hspace{1em} cq leftmost(T1 / I \ T2) = leftmost(T1) \text{ if } T1 /= null .
\texttt{endm}

(a) Sketch how Maude would simplify the term \texttt{leftmost((null / 6 \ null) / 3 \ null)}, using the above equations. \hspace{1em} [4 marks]

(b) Give a Java class \texttt{Tree} that implements this specification, using a class \texttt{TreeNode} that provides the branching structure of binary trees. (This is similar to linked lists, but a \texttt{TreeNode} instance will have three fields: one to store an integer, and two fields of type \texttt{TreeNode} to store the left and right subtrees.) Marks will be awarded for:

i. a correct implementation of the branching structure in classes \texttt{TreeNode} and \texttt{Tree} \hspace{1em} [4 marks]

ii. a correct implementation of the constant \texttt{null} as a \texttt{Tree} constructor \hspace{1em} [1 mark]

iii. a correct implementation of \texttt{\_/ \_/ \_/} as a \texttt{Tree} constructor \hspace{1em} [3 marks]

iv. a correct implementation of \texttt{sum} in class \texttt{Tree} \hspace{1em} [2 marks]

v. a correct implementation of \texttt{sum} in class \texttt{TreeNode} (this should be called by method \texttt{sum} in class \texttt{Tree}) \hspace{1em} [2 marks]

vi. a correct implementation of \texttt{leftmost}, using a while-loop rather than a recursive call \hspace{1em} [5 marks]

(c) Write a main method for the class \texttt{Tree} that corresponds to the term given in part (a). \hspace{1em} [4 marks]
2. (a) What, precisely, is meant by an Abstract Data Type? [4 marks]

(b) An abstract data type of Lists Without Repetitions (which we shall call ‘WRLists’) is described below.

WRLists are sequences of integers, in which no integer occurs more than once. The sequences of integers are built by means of:

- the constant ‘empty’, which is the empty sequence, containing no integers; and
- an operation ‘put’ that takes an integer and a WRList, and adds the integer to the start of a WRList, without checking whether the given integer occurs in the WRList.

Further operations are:

- an operation ‘isIn’ that tests whether a given integer occurs in a given WRList; i.e., it takes an integer and a WRList, and returns true if the integer occurs in the list, and false otherwise; and
- an operation ‘add’ that adds a given integer to a WRList, provided that integer does not already occur in the WRList; i.e., given an integer and a WRList, this operation returns the list formed by adding the integer to the list if it does not already occur in the WRList, otherwise it returns the given WRList unchanged.

i. Give a Maude specification of WRLists. [12 marks]

ii. Write a Maude term that corresponds to adding the integer 3 to the empty WRList twice (using the operation ‘add’). Sketch why your equations will result in the integer 3 occurring only once in that WRList (i.e., sketch what happens when the Maude interpreter reduces that term). [5 marks]

iii. Why would it be appropriate to implement the operation ‘put’ as a private method? [4 marks]
3. The following class implements bounded queues of integers: a queue can store at most two integers. The class has a method `add(int)` that adds the given integer to the queue, and a method `get()` that removes (and returns) the integer at the start of the queue.

```java
class Queue2 {
    private int[] items = new int[2];
    private int howMany = 0;

    public void add(int i) {
        items[howMany++] = i;
    }

    public int get() {
        int first = items[0];
        if (howMany > 1) items[0] = items[1];
        howMany--;
        return first;
    }
}
```

(a) Why would it be better to replace the declaration of `items` with

```java
private static final int CAPACITY = 2;
private int[] items = new int[CAPACITY];
```

(b) What is meant by a ‘class invariant’?

(c) Is `0 <= howMany` a class invariant? Give a reason for your answer.

(d) Is `howMany < 2` a class invariant? Give a reason for your answer.

(e) Briefly describe the function of the method-call stack in the Java interpreter.

(f) What happens when the main method in the Queue class is executed? Describe the state of the method-call stack during execution of the main method.

(g) Briefly describe the differences between ‘checked’ and ‘unchecked’ exceptions.

(h) Write a checked exception class `QueueFullException`, and modify the `add()` method so that it throws an `QueueFullException` if the queue already contains 2 elements. What other changes would be necessary to the class?
4. Consider the following program, which creates two threads sharing an instance of the Queue2 class from Question 3. One thread puts the values 0, 2, 4, … 198 into the queue, and the other puts the values 1, 3, … 199 into the queue, provided there is enough room in the queue. If the queue is full, the thread takes the first value off the queue, so it is no longer full, and prints that value to standard output.

```java
class ThreadTest extends Thread {
    static Queue2 queue = new Queue2();
    int startValue

    ThreadTest() {
    }

    public void run() {
        for (int i = startValue; i < 200; i = i + 2) {
            if (queue.howMany < 2) {
                queue.add(i);
            } else {
                System.out.println(queue.get());
            }
        }
    }

    public static void main(String[] args) {
        ThreadTest t1 = new ThreadTest();
        ThreadTest t2 = new ThreadTest();
        t1.startValue = 0;
        t2.startValue = 1;
        t1.start();
        t2.start();
    }
}
```

(a) Why can we not guarantee that when the program is run, the values printed to standard output will be in order 0, 1, 2, … 199? [2 marks]

(b) In the context of multi-threaded programs, what is meant by interference? [3 marks]

(c) Describe how interference might arise when the ThreadTest main method is executed. [8 marks]

(d) Describe how synchronization can be used to prevent interference. [7 marks]

(e) How would you modify the ThreadTest or Queue2 class to prevent interference? [5 marks]
5. Queues are First-In, First-Out lists: elements are removed from the ‘front’ of the queue and are added to the ‘end’ of the queue. A queue can be implemented by using an array, values, to store the elements in the queue, with an integer ‘pointer’, startIndex, indicating the index of the first element in the queue, and another, endIndex, whose value is one greater than the index of the last element in the queue. That is, the elements in the queue are all the values values[i], where startIndex ≤ i < endIndex.

Getting the element at the front of the queue can be done by returning the element stored in values[startIndex] and incrementing startIndex (or throwing an exception if the queue is empty, i.e., startIndex == endIndex).

Adding a value to the end of the queue is done by storing the value in values[endIndex] and incrementing endIndex, if endIndex is less than values.length. If endIndex is equal to values.length, then, if startIndex is greater than 0, room can be made to store the new element by moving all the values in the queue to the start of the array (and updating startIndex and endIndex). Otherwise, room will have to be made by creating a larger array (say, 10 more than values.length), copying all the elements in the queue to the larger array and assigning that array to the variable values, before storing the new element at the end.

(a) Implement a generic class Queue⟨E⟩, of homogeneous queues that store elements of the parameter type E. Marks will be awarded for:

i. correct use of the parameter type E [2 marks]
ii. implementation of the method getFirst() that returns the first element in the queue and removes it from the queue, as described above [5 marks]
iii. implementation of a checked Exception class that is thrown by getFirst() if the queue is empty [3 marks]
iv. implementation of the method addToEnd(E v) that adds the given (parameter) value to the end of the queue, as described above. [9 marks]

(b) Generic types are implemented in Java by ‘erasure’. Briefly describe what erasure is, and illustrate your answer by giving the results of erasure on your addToEnd(E v) method from your answer to part (a) above. [6 marks]