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. Below is a Maude specification of ‘Sorted Lists’: lists of integers in increasing order.

\textbf{mod} SORTED-LIST \textbf{is}

\textbf{protecting} INT .

\textbf{sort} SortedList .

\textbf{op} empty : \rightarrow SortedList .

\textbf{op} add : Int SortedList \rightarrow SortedList .

\textbf{op} insert : Int SortedList \rightarrow SortedList .

\textbf{vars} I J : Int .

\textbf{var} L : SortedList .

\textbf{eq} insert(I, empty) = add(I, empty) .

\textbf{cq} insert(I, add(J, L)) = add(I, add(J,L)) \textbf{if } I \leq J .

\textbf{cq} insert(I, add(J, L)) = add(J, insert(I, L)) \textbf{if } I > J .

\textbf{endm}

\textbf{(a)} Give a Java implementation of SortedLists, using linked lists. Marks will be awarded for correctly implementing:

i. the linked list structure; \hspace{1cm} [5 marks]

ii. a constructor corresponding to the operation ‘empty’; \hspace{1cm} [2 marks]

iii. method ‘add’; and \hspace{1cm} [3 marks]

iv. method ‘insert’. \hspace{1cm} [6 marks]

\textbf{(b)} Why would it be appropriate \textit{not} to implement ‘add’ as a public method? \hspace{1cm} [5 marks]

\textbf{(c)} Linked lists are one way of implementing lists; it is often possible to implement lists using arrays. Give one advantage and one disadvantage of each of these two ways of implementing lists. \hspace{1cm} [4 marks]
2. For this question, we consider an application where we need to store integer data (for example, exam results) with string ‘keys’ (for example, student names): each string will have an associated integer (for example, the exam mark for a given student). We call a pair consisting of a string and an integer a ‘KeyValuePair’, and we call a list of such pairs a ‘KeyValuePairList’.

A KeyValuePair is constructed by an operation, ‘makePair’, that takes a string and an integer as arguments; there are two ‘accessor’ operations: ‘getKey’, which returns the string key of a KeyValuePair, and ‘getValue’, which returns the integer value of a KeyValuePair.

KeyValuePairLists are constructed by means of a constant operation, ‘empty’, representing an empty list of KeyValues, and an operation, ‘add’, which takes a KeyValuePair and a KeyValuePairList and returns the KeyValuePairList formed by adding the given pair to the given list. In addition, there is an operation ‘get’ that takes a string key and a KeyValuePairList, and returns the integer value associated with the given key. Specifically, the operation searches through the list to find a KeyValuePair with the given key, and returns the integer value for that KeyValuePair; if the list does not contain a KeyValuePair with the given key, the value -1 is returned.

(a) What, precisely, is meant by the term Abstract Data Type? [6 marks]
(b) Give a Maude specification of the abstract data type KeyValuePair. (One mark will be given for correctly specifying the sort KeyValuePair; three marks will be given for correctly specifying the type of each of the three required operations, and four marks for correctly specifying their required behaviour by giving two equations.) [8 marks]
(c) Give a Maude specification of the abstract data type of KeyValuePairLists. (Two marks for correct Maude syntax; three marks for specifying the required operations, and six marks for the equations.) [11 marks]
3. The following class implements bounded stacks of integers: a stack can store at most five integers. The class has a method remove(int) that pops the stack a given number of times.

```java
public class Stack {
    private int[] items = new int[5];
    private int top = 0;

    public boolean isEmpty() {
        return top <= 0;
    }

    public int pop() {
        return items[--top];
    }

    public void push(int i) {
        items[top++] = i;
    }

    public void remove(int i) {
        if (i <= 0) return;
        pop();
        remove(i - 1);
    }

    public static void main(String[] args) {
        Stack s = new Stack();
        s.push(3);
        s.remove(2);
    }
}
```

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

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

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

(d) Write a checked exception class EmptyStackException, and modify the pop() method so that it throws an EmptyStackException if the stack is empty. What other changes would be necessary to the class?

(e) With these changes, is the property $0 \leq \text{top}$ a class invariant? Justify your answer.
4. Consider the following program, which creates two threads sharing a Stack instance (using the class from Question 3).

```java
class ThreadTest implements Runnable {
    private Stack stack;
    ThreadTest(Stack s) {
        stack = s;
    }

    public void run() {
        for (int i = 0; i < 200; i++) {
            if (stack.isEmpty()) {
                stack.push(i);
            } else {
                System.out.println(stack.pop());
            }
        }
    }

    public static void main(String[] args) {
        Stack s = new Stack();
        Thread t1 = new Thread(new ThreadTest(s));
        Thread t2 = new Thread(new ThreadTest(s));
        t1.start();
        t2.start();
    }
}
```

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

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

(c) Describe how *synchronization* can be used to prevent interference. [8 marks]

(d) How would you modify the ThreadTest or Stack class to prevent interference? [6 marks]
5. The Stack class in Question 3 stores integers. Java uses generic classes to allow the implementation of data structures that can store instances of arbitrary classes.

(a) Modify the Stack class to give a generic implementation of stacks: i.e., use a generic type variable E so that the values stored in the stack all belong to the class E.

(b) Using your answer to part (a), give Java code that creates an instance of a stack class that can store instances of class Integer, and add integers 3 and 7 to the stack.

(c) Write a generic method that takes a stack of stacks (i.e., the parameter is of type Stack<Stack<E>>) so that every element in the stack is itself a stack, and returns a stack containing all the elements in each stack in the given stack (so that all elements in the returned stack belong to the generic class E — it does not matter what order the elements occur in).

(d) Java implements generic types by a process called erasure, which removes generic type parameters. What would the results of erasure be for:

i. the Stack class in your answer to part (a);

ii. your code in your answer to part (b);

iii. the generic method in your answer to part (c)?