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 ‘Key-Value’ Pairs and Lists of Pairs. Strings are used as keys and integers as values.

\[\text{mod \ PAIRS \ is} \]
\[\text{\hspace{1cm} protecting \ STRING . } \]
\[\text{\hspace{1cm} protecting \ INT . } \]
\[\text{\hspace{1cm} sort \ Pair . } \]
\[\text{\hspace{1cm} op \ makePair : \ String \ Int \ -> \ Pair . } \]
\[\text{\hspace{1cm} op \ getString : \ Pair \ -> \ String . } \]
\[\text{\hspace{1cm} op \ getInt : \ Pair \ -> \ Int . } \]
\[\text{\hspace{1cm} sort \ PairList . } \]
\[\text{\hspace{1cm} op \ empty : \ -> \ PairList . } \]
\[\text{\hspace{1cm} op \ add : \ Pair \ PairList \ -> \ PairList . } \]
\[\text{\hspace{1cm} op \ get : \ String \ PairList \ -> \ Int . } \]
\[\text{\hspace{1cm} var \ S : \ String . } \]
\[\text{\hspace{1cm} var \ I : \ Int . } \]
\[\text{\hspace{1cm} var \ P : \ Pair . } \]
\[\text{\hspace{1cm} var \ PL : \ PairList . } \]
\[\text{\hspace{1cm} eq \ getString(makePair(S, I)) = S . } \]
\[\text{\hspace{1cm} eq \ getInt(makePair(S, I)) = I . } \]
\[\text{\hspace{1cm} eq \ get(S, empty) = -1 . } \]
\[\text{\hspace{1cm} cq \ get(S, add(P, PL)) = \begin{cases} \text{getInt(P)} & \text{if \ getString(P) == S .} \\ \text{get(S, PL)} & \text{if \ getString(P) /= S .} \end{cases} } \]
\[\text{endm} \]

The operation ‘makepair’ constructs a Pair from a given String and integer, and these can be accessed using the operations ‘getString’ and ‘getInt’, respectively. Lists of pairs are built up using the constant ‘empty’ and the operation ‘add’. Operation ‘get’ takes a string and a list as arguments: if it finds in the list a pair that has the given string as key, then it returns the integer from that pair; otherwise, if there is no such pair in the list, it returns a default value of -1.

(a) Give a Java implementation of the sort Pair. [6 marks]

(b) Give an implementation of the sort PairList, using linked lists. Marks will be awarded for correctly implementing:
   i. the linked list structure; [4 marks]
   ii. a constructor corresponding to the operation ‘empty’; [2 marks]
   iii. method ‘add’; and [3 marks]
   iv. method ‘get’. [5 marks]

(c) Say why your implementation of the operation ‘get’ behaves in the way specified by the two conditional equations in the Maude specification. [5 marks]
2. (a) What, precisely, is meant by an Abstract Data Type? [5 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. [11 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. (a) The method start() in class Thread will place a new thread in a pool of threads that are ‘ready’ to run under time-slicing. The Java interpreter will select a thread from this pool to ‘run’ (i.e., execute the thread). ‘Ready’ and ‘running’ are two of the possible states that threads may be in, and threads move between these states under the control of the Java interpreter’s time-slicing mechanism. What other states can threads be in, and how do they change from one state to another? [7 marks]

(b) The Java keyword synchronized can be applied to a method or a block of code.

   i. The keyword synchronized was introduced to Java to allow programmers to solve a particular kind of problem that may arise in multi-threaded computation. Briefly describe what this problem is, and how it might arise. [6 marks]

   ii. Describe how synchronized solves this problem. [6 marks]

   iii. During the lectures, we saw that using synchronized may lead to deadlock. Briefly describe what is meant by deadlock, and describe how it may be avoided using the methods wait() and notify(). [6 marks]
4. The following class implements a **bounded** queue of integers: the queue can store at most ten integers.

```java
public class Queue {
    private int[] items = new int[10];
    private int numItems = 0;

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

    public int next() {
        int n = items[0];
        shift();
        numItems--;
        return n;
    }

    private void shift() {
        for (int i=0; i+1 < numItems; i++) {
            items[i] = items[i+1];
        }
    }

    public static void main(String[] args) {
        Queue q = new Queue();
        System.out.println("first in queue is: " + q.next());
        q.add(24);
    }
}
```

(a) Describe the state of the stack of method calls as the main method is executed. What Exception would you expect to be thrown, and what would the stack trace be? 

(b) i. Write a checked Exception class EmptyQueueException, and modify the `next()` method in class Queue so that it throws an EmptyQueueException if the method is called when the queue is empty. What other changes would be required for `Queue.java` to successfully compile?

(b) ii. Now what happens when the main method is run?

(c) With the modification of part (b)(i) above, is 0 <= numItems a class invariant for Queue? Justify your answer.
A ‘function’ is something that takes ‘input’ and produces ‘output’. The input and output can belong to any class. For example, a ‘function’ that computes the length of a string takes a String as input and produces an Integer. As another example, a ‘function’ that concatenates a string with itself has String as both its input and output type. These two functions could be ‘composed’: the output of the second provides the input of the first. This composed function then takes a String as input and produces an Integer as output.

(a) Use Java’s Generic Types to specify an interface of ‘functions’ that have a method `compute`, which takes input (as a parameter) from some class A and produces output (as return-type) belonging to some class B. [5 marks]

(b) Write two classes that implement this interface; one whose `compute()` method takes a String as input and returns the String formed by concatenating the given String with itself; the other with a `compute()` method that takes a String as input and returns the length of the String as an Integer (using the String method `length()`). [6 marks]

(c) Two functions can be composed if their input and output types are compatible, i.e., if the output type of one function is the same as the input type of the other function. Suppose f1 is a function with input type A and output type B, and f2 is a function with input type B and output type C. The result of composing f1 with f2 is a function with input type A and output type B. Given input a, the output will be the result of passing the output of f1.compute(a) as input to f2.compute().

Write a generic method that takes as parameters two functions with compatible input and output types, and returns the composition of the two parameter functions. (Hint: the method should return an instance of an anonymous class that implements the(Function interface.) [9 marks]

(d) Generic types are implemented in Java by erasure. Describe briefly what this means, and illustrate your answer using your answers to parts (a)–(c) above. [5 marks]