FIRST SEMESTER EXAMINATIONS 2010/11

Advanced Object-Oriented Programming

TIME ALLOWED : Two hours

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 Lists Without Repetitions (which we shall call ‘WRLists’) is given below. WRLists are sequences of integers, in which no integer occurs more than once.

\textbf{mod WRLISTS is}\n\textbf{protecting} INT .
\textbf{sort} WRList .
\textbf{op} empty : \to WRList .
\textbf{op} add : Int WRList \to WRList .
\textbf{op} insert : Int WRList \to WRList .
\textbf{op} remove : Int WRList \to WRList .
\textbf{vars} I J : Int .
\textbf{var} L : WRList .
\textbf{eq} insert(I, empty) = add(I, empty) .
\textbf{eq} insert(I, add(I, L)) = add(I, L) .
\textbf{cq} insert(I, add(J, L)) = add(J, insert(I, L)) \text{ if } I \neq J .
\textbf{eq} remove(I, empty) = empty .
\textbf{eq} remove(I, add(I, L)) = L .
\textbf{cq} insert(I, add(J, L)) = add(J, remove(I, L)) \text{ if } I \neq J .
\textbf{endm}

Operations \texttt{empty} and \texttt{add} are structural operations that provide a list structure; it is possible to use these to construct lists that do not have the intended property that any integer occurs at most once in the list. For example,
\[ \text{add}(1, \text{add}(1, \text{empty})) \]
does not have that property, as 1 occurs twice in the list. However, the operation \texttt{insert} will ensure that this property holds: if it is called from a list without repetitions, it will only insert a value into the list if it does not already occur in the list.

(a) Why should \texttt{add} not be implemented as a public operation? \hfill [2 marks]

(b) Give a Java implementation of class \texttt{WRList} using a linked list structure in a class \texttt{Node}. Marks will be awarded for:
   i. correct implementation of the linked list structure \hfill [3 marks]
   ii. correct implementation of the constant \texttt{empty} as a \texttt{WRList} constructor \hfill [1 marks]
   iii. correct implementation of \texttt{add} as a \texttt{Node} constructor \hfill [2 marks]
   iv. correct implementation of \texttt{remove} \hfill [7 marks]
   v. correct implementation of \texttt{insert} \hfill [5 marks]
   vi. appropriate use of scope modifiers. \hfill [2 marks]

(c) If class \texttt{Node} were declared as a \texttt{protected} inner class within class \texttt{WRList}, where precisely would \texttt{Node} and its public members be visible? \hfill [3 marks]
2. The Abstract Data Type of Count Lists is a list of pairs; each pair in the list has an integer (which we call the ‘value’) and another integer (which we call the ‘count’); the idea is that the ‘count’ integer records how often the value occurs in the list. An example application might be to read in a sequence of integer values, perhaps recording the grades of individuals in a class, e.g.,

\[2, 7, 7, 5, 9, 3, 6, 7, 2, 5\]

and, for each grade value, record how often that grade occurs in the given data. For the example data above, we should have the following Count List (where pairs are grouped in brackets):

\[(2, 2), (3, 1), (5, 2), (6, 1), (7, 3), (9, 1)\].

Note that grades that don’t occur in the data, such as 4, are not listed.

As an Abstract Data Type, Count Lists have one constant, representing the empty list (no data); and the following operations:

- **add** a Pair (consisting of two integer values, a value and its count) to the list;
- **insert** a given integer value into the list (if it already occurs in the list, increment the count value; otherwise, add a new pair with the given value and a count of 1); and
- **getCount** — this takes an integer value as argument and returns the count of how often that value occurs in the list (0 if it doesn’t occur at all in the list).

Count Lists make use of a subsidiary Abstract Data Type of Pairs: this Abstract Data Type has three operations:

- a constructor that takes two integers and returns the Pair built from those values;
- **getValue**, returning the value component of the Pair; and
- **getCount**, returning the count.

(a) Specify the Abstract Data Type of Pairs in Maude. \([6 \text{ marks}]\)

(b) Specify Count Lists in Maude. \([11 \text{ marks}]\)

(c) Write a term in Maude that corresponds to the Count List recording all the data in (1) above. \([3 \text{ marks}]\)

(d) Sketch why the equations in your specification of Count Lists would allow you to conclude that the value 5 occurs twice in that list. \([5 \text{ marks}]\)
3. The following class implements bounded queues of integers: a queue can store at most five integers. The class has a method remove(int) that removes the given number of elements from the queue.

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

    public int get() {
        int first = items[0];
        for (int i = 0; i+1 < top; i++) {
            items[i] = items[i+1];
        }
        top--;
        return first;
    }

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

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

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

(a) Briefly describe the function of the method-call stack in the Java interpreter. [3 marks]

(b) 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. [5 marks]

(c) Briefly describe the differences between ‘checked’ and ‘unchecked’ exceptions. [4 marks]

(d) Write a checked exception class EmptyQueueException, and modify the get() method so that it throws an EmptyQueueException if the queue is empty. What other changes would be necessary to the class? [7 marks]

(e) With these changes, is the property $0 \leq \text{top}$ a class invariant? Justify your answer. [6 marks]
4. Consider the following two classes.

```java
class FriendFinderThread extends Thread {

    int numFriends = 0;

    public void run() {
        FriendBook.join(this);
        try { Thread.sleep(10000); } catch (InterruptedException ie) {}
        FriendBook.leave(this);
    }
}

class FriendBook {

    static Vector<FriendFinderThread> members =
        new Vector<FriendFinderThread>();

    static void join(FriendFinderThread f) {
        int size = members.size();
        for (int i = 0; i < size; i++) {
            members.elementAt(i).numFriends++;
        }
        f.numFriends = size;
        members.add(f);
    }

    static void leave(FriendFinderThread f) {
        members.remove(f);
        int size = members.size();
        for (int i = 0; i < size; i++) {
            members.elementAt(i).numFriends--;
        }
    }

    public static void main() {
        for (int n = 0; n < 100; n++) {
            new FriendFinderThread().start();
        }
    }
}
```
Class FriendBook maintains a list of its ‘members’; a FriendFinderThread instance becomes a member by calling the `join()` method. While it is a member of FriendBook, the `numFriends` field should be equal to the total number of members. When a new FriendFinderThread instance becomes a member, the `join()` method increments the `numFriends` field of all existing members; similarly, the `leave()` method decrements these fields.

(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? [6 marks]

(b) One problem that might arise in the FriendBook program is ‘interference’. Briefly say what interference is, and describe how it might arise in this program. [6 marks]

(c) Describe how synchronization is used in Java to prevent interference. [6 marks]

(d) Briefly describe the difference between a synchronized method and a synchronized block of code. [3 marks]

(e) How would you use synchronization to prevent interference in the FriendBook program? Justify your answer. [4 marks]

5. The Queue 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 Queue class to give a generic implementation of queues: i.e., use a generic type variable `E` so that the values stored in the queue all belong to the class `E`. Add a boolean method `isEmpty()` that tests whether the queue is empty. [6 marks]

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

(c) Write a generic method that takes a queue of queues (i.e., the parameter is of type `Queue<Queue<E>>`) so that every element in the queue is itself a queue), and returns a queue containing the first elements from each queue in the given queue (so that all elements in the returned queue belong to the generic class E). The first element in the result queue should be the first element of the first queue; the second element should be the first element of the second queue; and so on. [6 marks]

(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 Queue class in your answer to part (a); [4 marks]
   ii. your code in your answer to part (b); [2 marks]
   iii. the generic method in your answer to part (c)? [4 marks]