UNIVERSITY OF NEW ENGLAND

UNIT NAME: COMP 132

PAPER TITLE: Computer Science II

PAPER NUMBER: First and Only

DATE: **Friday 20 November 2009** **TIME:** **2:00 PM TO 4:00 PM**

TIME ALLOWED: Two (2) hours plus fifteen minutes reading time

NUMBER OF PAGES IN PAPER: EIGHT (8)

NUMBER OF QUESTIONS ON PAPER: FIVE (5)

NUMBER OF QUESTIONS TO BE ANSWERED: FIVE (5)

STATIONERY PER CANDIDATE:

0	6 LEAF A4 BOOKS	**0**	ROUGH WORK BOOK
1	12 LEAF A4 BOOKS	**0**	GRAPH PAPER SHEETS

OTHER AIDS REQUIRED: **NIL**

POCKET CALCULATORS PERMITTED: **YES** **(SILENT TYPE)**

TEXTBOOKS OR NOTES PERMITTED: **NIL**

INSTRUCTIONS FOR CANDIDATES:

- Candidates **MAY** make notes on this examination question paper during the fifteen minutes reading time
- Questions are **NOT** of equal value
- Answer **ALL** questions in workbooks provided. Answers written on this examination paper **will not be marked**
- Candidates may retain this examination question paper

Question 1

Multiple Choice: Choose the ONE alternative that best completes the statement or answers the question.

(a) *[2 marks]*

Like a loop, a recursive method must have

(i) A counter

(ii) Some way to control the number of times it repeats itself

(iii) A return statement

(iv) A predetermined number of times it will execute before terminating

(b) *[2 marks]*

Which of the following problems cannot be programmed recursively?

(i) Towers of Hanoi

(ii) Greatest Common Denominator

(iii) Binary Search

(iv) All of these can be programmed recursively

(c) *[2 marks]*

To compare `String` objects for the purpose of sorting, a programmer should

(i) use the comparison operator $<$

(ii) use the comparison operator $<=$

(iii) use the `compareTo` method of the `Comparable` interface

(iv) use the relational operator $<$ to compare references to the `String` objects

Question 1 (d) is on page 3

(d) *[3 marks]*

The following implementation of QuickSort:

```
static void doQuickSort(int array[], int start, int end)
{
    int pivotPoint;
    pivotPoint = partition(array, start, end);
    doQuickSort(array, pivot+1, end);
    doQuickSort(array, start, pivot-1);
} // doQuickSort
```

(i) will correctly sort the array if the partition method is written correctly

(ii) will give incorrect results because the two recursive calls are called in the wrong order

(iii) will sort the array in descending rather than ascending order

(iv) will be terminated by the system for making too many recursive calls

(e) *[2 marks]*

One of the advantages of using generics is

(i) that more type problems can be uncovered at compile-time rather than at run time

(ii) that programs that use generics are smaller when translated to byte code

(iii) that program that use generics execute faster than programs that do not

(iv) that programs that use generic code require less effort in design and development

(f) *[2 marks]*

When a generic class with an unconstrained type parameter is instantiated without specifying an actual type

(i) the type `Object` is used for the unspecified type

(ii) the compiler generates an error

(iii) the computer throws a `ClassCastException`

(iv) None of these

(g) *[2 marks]*

Which of the following is true?

(i) The `retrieve()` method of an iterator can only be called after `isEmpty()` has returned false

(ii) The `remove()` method of an iterator can only be called after `next()` has been called

(iii) Any iterator can move forward as well as backwards through a collection

(iv) Once an iterator has reached the end of the collection, calling any of its methods will throw a `NoSuchElement` exception

(h) *[2 marks]*

A *collision* occurs when

(i) objects whose class types are not related through inheritance have the same hashcode

(ii) objects instantiated from different generic types have the same hashcode

(iii) objects whose class types are not related through inheritance have hashcodes one of which divides the other

(iv) objects whose values are not equal have the same hash code

(i) *[2 marks]*

A new element is added to an `ArrayList` object at index k. Assuming the list has size s and does not have to be resized,

(i) the elements at current positions $0 \ldots k$ must be moved toward the beginning of the list

(ii) the elements at current positions $k \ldots s-1$ must be moved toward the end of the array

(iii) the elements at current positions $k \ldots s$ must be moved toward the end of the array

(iv) the element at position k is overwritten

(j) *[2 marks]*

The `boolean contains(E element)` method searches an `ArrayList` for a given element. A correct and efficient implementation of this method

(i) throws an exception if the element is not found in the list

(ii) uses binary search to locate the element

(iii) uses sequential search to locate the element

(iv) returns 0 if the element is not found in the list

(k) *[2 marks]*

A `Node` class for a **linked list** that can hold elements of type `Object` can be declared to have fields

(i) `Object element;`

(ii) `Object element; Node next;`

(iii) `Object element; Node *next;`

(iv) `Object element; next element;`

(l) *[2 marks]*

In a typical **circular doubly linked list**, a node has

(i) a field to store the element, and two references to keep track of two successor nodes, and a reference to keep track of the start of the list

(ii) a field to store the element, and two references to keep track of successor and predecessor nodes

(iii) a field to store the element, and two references to keep track of two predecessor nodes and a reference to keep track of the end of the list

(iv) either one of a field to store the element, and two references to keep track of two successor nodes, and a reference to keep track of the start of the list or a field to store the element, and two references to keep track of two predecessor nodes, and a reference to keep track of the end of the list

(m) *[2 marks]*

A **queue** is a container that allows elements to be stored and removed

(i) in a *last-in-first-out* fashion

(ii) in a *first-in-first-out* fashion

(iii) in a *first-in-last-out* fashion

(iv) quickly and efficiently

(n) *[2 marks]*

A **stack** is a container that allows elements to be stored and removed

(i) in a *last-in-first-out* fashion

(ii) in a *first-in-first-out* fashion

(iii) in a *last-in-last-out* fashion

(iv) according to priority

(o) *[2 marks]*

A **binary tree** is a collection of nodes in which

(i) each node has at most one predecessor and at most one successor

(ii) each node has at most one predecessor and exactly two successors

(iii) each node has at most one predecessor and at most two successors

(iv) each node has at least one predecessor and at most two successors

(p) *[2 marks]*

Postorder traversal of a binary tree

(i) first visits the root, then recursively traverses the left and right subtrees

(ii) recursively traverses the left subtree, then visits the root, then traverses the right subtree

(iii) recursively traverses the left subtree, then traverses the right subtree, then visits the root

(iv) visits all the nodes according to their natural order

Question 2 *Exceptions*

(a) *[4 marks]*

Write a statement that throws an `IllegalArgumentException` with the error message "Argument cannot be negative".

(b) *[6 marks]*

Write an exception class that can be thrown when a negative number is passed to a method.

Question 3 *Collections*

(a) *[4 marks]*

How does the Java compiler process an enhanced `for` loop.

(b) *[6 marks]*

How do you get an iterator for a list? Give an example.

(c) *[10 marks]*

Write an algorithm (or pseudo-code) for a method of a `HashSet` that adds a new object.

Question 4 *Lists and Stacks*

(a) *[6 marks]*

What are the three basic operations of an iterator?

(b) *[4 marks]*

The following class will be used to instantiate iterators for a list class. Find the error.

```
public class MyIterator<E> implements Iterable<E>
{
    ... class code here ...
}
```

Question 4 (c) is on page 8

(c) *[4 marks]*

Find an error in the following piece of code, describe it and provide a way to fix it:

```
// print all element in a list myList
Node ref = myList;
while (ref.next != null)
{
    System.out.print(ref.value + " ");
    ref = ref.next;
}
```

(d) *[8 marks]*

A *palindrome* is a word that reads the same backward as forward. For example, the words *madam*, *radar*, *dad* and *kayak* are all palindromes. Write a method that takes a parameter `s` of type `String` and uses a stack to see if s is a palindrome. The method returns `true` if the word is a palindrome and `false` otherwise.

Question 5 *Trees*

(a) *[7 marks]*

Explain why the depth of a complete binary tree with n nodes is at most $\log_2(n+1)$.

(b) *[8 marks]*

Assume that data is stored in a binary tree, but unlike the case of a binary search tree, no attempt is made to maintain any sort of order in the data stored. Give an algorithm (or pseudo-code) for a method `contains()` that searches a binary tree for a particular value `x` and returns `true` or `false` according to whether `x` is is found in the tree.