DUBLIN CITY UNIVERSITY

SEMESTER ONE REPEAT EXAMINATIONS 2011

MODULE: *(Title and Code)*	CA213 Data Structures and Algorithms
COURSE:	B.Sc. in Computer Applications B.Eng. in Digital Media Engineering B.Eng. in Information & Communications Engineering
YEAR:	2/3
EXAMINERS:	Dr James Power Dr Frank Bannister Prof. J. M. Morris (Ext. 8419)
TIME ALLOWED:	2 Hours
INSTRUCTIONS:	***Answer any 4 questions.*** All questions carry equal marks. There are five questions in total. Programs are to be written in Java and may draw only on those elements of the Java library listed at the end of the paper. As instructed on answer books, candidates must not write using red ink.

Requirements for this paper
Please tick (X) as appropriate

- ☐ ***Log Table***
- ☐ ***Graph Paper***
- ☐ ***Attached Answer Sheet***
- ☐ ***Statistical Tables***
- ☐ ***Floppy Disk***
- ☐ ***Actuarial Tables***

THE USE OF PROGRAMMABLE OR TEXT STORING CALCULATORS IS EXPRESSLY FORBIDDEN

Please note that where a candidate answers more than the required number of questions, the ones to be marked should be clearly indicated; otherwise the first four will be marked.

PLEASE DO NOT TURN OVER THIS PAGE UNTIL ASKED TO DO SO

Question 1

(i) Briefly explain the concept of *interface* in the context of Java. Why are interfaces useful?

[6 marks]

(ii) Write a Java interface `Order` with a single method called `lte` (short for *less than or equal to*) which takes a single parameter of type `Object` and which returns a boolean value.

[6 marks]

(iii) Enhance class `Time` below so that it implements interface `Order`. Method `lte` in `Time` should expect its single argument to be of type `Time`, and should return `true` if the time in the current object is less than or equal to the time represented by the argument.

```
class Time { // the time of day in 24-hour clock form
   private int hours; // 0..23
   private int mins; // 0..59

   Time (int hours0, int mins0) {
      hours = hours0; mins = mins0;
   }

   void put() {
      System.out.printf("%02d:%02d\n", hours, mins);
   }
}
```

[7 marks]

(iv) The following method sorts an array of strings. Make as few amendments to it as possible so that it sorts arrays of objects that implement the `Order` interface. Hint: small changes in three places suffice.

```
static void sort(String[] w) {// Sort w[0..]
    final int n = w.length;
    int j = 0; // w[j..] to be sorted
    while (j<n-1) {
        // find minimum in w[j..]
        int min = j;  int i = j+1;
        // always: w[min] is minimum in w[j..i-1]
        while (i<n) {
            if (w[i].compareTo(w[min])<0) min = i;
            i++;
        }
        // w[min] is minimum in w[j..]; swap w[j] and w[min]
        String temp = w[j];
        w[j] = w[min]; w[min] = temp;
        j++;
    }
}
```

[6 marks]

Question 2

(i) Types such as `LinkedList<T>`, `HashSet<T>`, etc. are said to be *generic collections*. What does this mean?

[6 marks]

(ii) The following ADT describes a generic collection known as a *bag*. Implement it as a Java class using an array of length 100 to store the items in the collection, and an integer variable to keep track of the size of the collection.

> *Class name*: `Bag<T>`
> *Data*: Collection of (at most 100) items of type `T`; items may occur more than once.
> *Constructors*: no-args constructor which creates empty collection
> *Methods*:
> `void add(T x);`
> Add `x` to the collection (guaranteed not to lead to more than 100 items in collection)
> `int freq(T x);`
> How many `x`'s in the collection?
> *...other methods omitted...*

[13 marks]

(iii) Suppose the Java class `Bag<T>` in Part (ii) is replaced with `Bag<T extends Person>` where `Person` is the name of an abstract class. How does this limit the way in which you may use the class? Assuming `Bag<T extends Person>` has been defined, which of the following are legitimate declarations where `Employee` is a class that extends `Person`, and `Manager` is a class that extends `Employee`?

(a) `Bag<Person> b;`
(b) `Bag<Employee> b;`
(c) `Bag<Manager> b;`
(d) `Bag<Integer> b;`

[6 marks]

Question 3

(i) The library accompanying Java and many modern programming languages includes a set ADT. Describe what you consider the four most important operations that you would expect to find in a set ADT. Where relevant, take care to distinguish between the cases when elements are present or absent in the set.

[8 marks]

(ii) Using any implementation of sets in the Java library, write a program which reads a sequence of words, one word per line, and for each word prints a message indicating whether or not the word occurred

previously in the input. An example of input/output follows (first and alternate lines are input):

```
success
First occurrence
breeds
First occurrence
success
Occurred previously
```

[12 marks]

(iii) (a) A program has been written (and works correctly) using `HashSet` from the Java library. Would you expect it to work correctly if `HashSet` is replaced throughout with `TreeSet`? Explain.

(b) Another program has been written (and works correctly) using `TreeSet`. Would you expect it to work correctly if `TreeSet` is replaced throughout with `HashSet`? Explain.

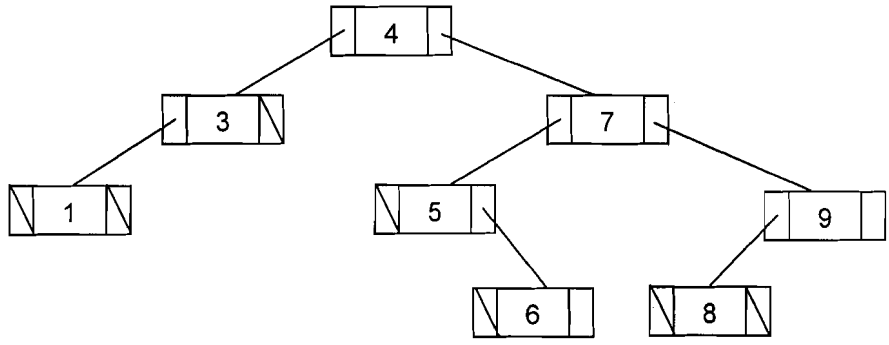
[5 marks]

Question 4

(i) What distinguishes a binary *search* tree from a binary tree?

[4 marks]

(ii) Write down (a) the preorder and (b) the inorder traversal of the binary tree below.



[8 marks]

(iii) Draw the above tree after the node containing 7 has been deleted using the standard deletion algorithm for binary search trees.

[6 marks]

(iv) An outline implementation of Java's `TreeSet` class is provided below, based on binary search trees. Complete the code of `contains` using either iteration or recursion.

```
class TreeSet<T extends Comparable<T>> {

   private static class Node<T> { // node of binary search tree
      private T item;   // data item
      private Node<T> left, right; //left and right subtrees

      Node(T item0, Node<T> left0, Node<T> right0)  {
         item = item0; left = left0; right = right0;
      }
   }

   private Node<T> root = null;      // root of tree

   private int numItems = 0;         // number of  nodes

   boolean contains(T t) { ... }

   // other methods omitted ...

}
```

[7 marks]

Question 5

(i) Outline the idea underlying the following array sorting algorithms: (a) selection sort, (b) insertion sort, and (c) merge sort. A few sentences suffices in each case.

[9 marks]

(ii) Compare the (worst case) time complexities of the above three sorting algorithms. Justify the complexity for one of the algorithms.

[8 marks]

(iii) Compare the additional storage costs of the three sorting algorithms.

[4 marks]

(iv) Given a very large array that you know to be nearly sorted, which of the three algorithms would you use. Explain.

[4 marks]

[End of exam]

Table of Java Classes *(Interfaces in italics)*

ArrayList<T>
ArrayList()
ArrayList(Collection<T>)
methods of List<T>

Boolean
Boolean(boolean)
boolean booleanValue()

Calendar
static int DAY
static int MONTH
static int YEAR

Character
Character(char)
char charValue()
static boolean isDigit(char)
static boolean isLetter(char)
static boolean isLetterOrDigit(char)
static boolean isLowerCase(char)
static boolean isUpperCase(char)
static boolean isWhitespace(char)
static char toLowerCase(char)
static char toUpperCase(char)

Collection<T>
int size()
boolean add(T)
boolean remove(Object)
boolean contains(Object)
void clear()
boolean isEmpty()
boolean addAll(Collection<T>)
boolean retainAll(Collection<?>)
boolean removeAll(Collection<?>)
boolean containsAll(Collection<?>)
Iterator<T> iterator()

Collections
static void sort(List<T>)
static int binarySearch(List<T>, T)
static void shuffle(List<?>)

Comparable<T>
int compareTo(T)

Console
static int readInt()
static boolean readBoolean()
static double readDouble()
static char readChar()
static String readString()
static void skipLine()
static String readToken()
static boolean endOfFile()
static boolean hasMoreTokens()
static void skipWhitespace()

ConsoleReader
ConsoleReader(String)
int readInt()
boolean readBoolean()
double readDouble()
char readChar()
String readString()
void skipLine()
String readToken()
boolean endOfFile()
boolean hasMoreTokens()
void skipWhitespace()

DataInputStream
DataInputStream(FileInputStream)
int readInt() throws IOException
long readLong() throws IOException
boolean readBoolean() throws IOException
char readChar() throws IOException
double readDouble() throws IOException
float readFloat() throws IOException
String readUTF() throws IOException
int read(byte[]) throws IOException
int available() throws IOException
void close() throws IOException

DataOutputStream
DataOutputStream(FileOutputStream)
void writeBoolean(boolean) throws IOException
void writeInt(int) throws IOException
void writeLong(long) throws IOException
void writeDouble(double) throws IOException
void writeFloat(float) throws IOException
void writeChars(String) throws IOException
void writeUTF(String) throws IOException
void writeChar(int) throws IOException
void write(byte[], int, int) throws IOException
void close() throws IOException
void flush() throws IOException
int size()

Double
Double(double)
double doubleValue();
static double parseDouble(String)
static String toString(double)

Exception
void printStackTrace()

File
File(String)
boolean exists()
boolean isFile()
boolean isDirectory()
boolean canRead()

boolean canWrite()
long length()
boolean delete()
String getName()
boolean renameTo(File)
File[] listFiles()

FileInputStream

FileInputStream(String) throws
FileNotFoundException

FileOutputStream

FileOutputStream(String) throws IOException
FileOutputStream(String, boolean) throws
IOException

FileReader

FileReader(String) throws
FileNotFoundException

FileWriter

FileWriter(String) throws IOException
FileWriter(String, Boolean) throws
IOException

Float

Float(float)
float floatValue();
static float parseFloat(String)
static String toString(float)

GregorianCalendar

GregorianCalendar()
int get(int)

HashMap<T,U>

HashMap()
HashMap(Map<T,U>)
methods in Map<T,U>

HashSet<T>

HashSet()
HashSet(Collection<T>)
methods in Set<T>

Integer

Integer(int)
int intValue()
static int parseInt(String)
static String toString(int)

Iterable<T>

Iterator<T> iterator()

Iterator<T>

boolean hasNext()
T next()
void remove()

LinkedList<T>

LinkedList()
LinkedList(Collection<T>)
void addFirst(T)
T getFirst()
T getLast()
T removeFirst()
methods in List<T>

List<T>

int size()
T set(int, T)
T get(int)
boolean add(T)
String toString()
void add(int, T)
T remove(int)
boolean remove(Object)
boolean contains(Object)
int indexOf(Object)
void clear()
boolean isEmpty()
boolean addAll(Collection<T>)
boolean retainAll(Collection<?>)
boolean removeAll(Collection<?>)
boolean containsAll(Collection<?>)
Iterator<T> iterator()

Long

Long(long)
long longValue()
static long parseLong(String)
static String toString(long)

Map<T,U>

void clear()
U put(T key, U value)
U get(Object key)
U remove(Object key)
boolean containsKey(Object key)
int size()
boolean isEmpty()
Set<T> keySet()
String toString()

Math

static double random()
static double abs(double)
static int abs(int) *et cetera*
static double ceil(double)
static double floor(double)
static int max(int, int) etc.
static int min(int, int) etc.
static double rint(double)
static long round(double)
static int round(float)
static double sqrt(double)

Object

boolean equals(Object)
String toString()

PrintWriter

PrintWriter(String)
PrintWriter(FileWriter)

PrintWriter(FileWriter, boolean)
void print(String)
void println(String)
void println()
PrintWriter printf(String, Object...)
void close()
void flush()

RandomAccessFile

RandomAccessFile(String, String) throws IOException
long length() throws IOException
void seek(long) throws IOException
int readInt() throws IOException
long readLong() throws IOException
boolean readBoolean() throws IOException
char readChar() throws IOException
double readDouble() throws IOException
float readFloat() throws IOException
String readUTF() throws IOException
void writeBoolean(boolean) throws IOException
void writeInt(int) throws IOException
void writeLong(long) throws IOException
void writeDouble(double) throws IOException
void writeFloat(float) throws IOException
void writeChars(String) throws IOException
void writeUTF(String) throws IOException
void writeChar(int) throws IOException
void close() throws IOException

Scanner

Scanner(File) throws FileNotFoundException
Scanner(String)
String nextLine()
String next()
int nextInt()
long nextLong()
double nextDouble()
boolean nextBoolean()
boolean hasNextLine()
boolean hasNext()
boolean hasNextInt()
boolean hasNextLong()
boolean hasNextDouble()
boolean hasNextBoolean()
void close()

Set<T>

boolean add(T)
boolean remove(T)
boolean contains(T)
int size()
void clear()
boolean isEmpty()
boolean addAll(Collection<T>)
boolean retainAll(Collection<?>)
boolean removeAll(Collection<?>)
boolean containsAll(Collection<?>)
Iterator<T> iterator()

String

int length()
boolean startsWith(String)
boolean startsWith(String)
boolean endsWith(String)
int indexOf(String)
String substring(int, int)
String substring(int)
char charAt(int)
String toUpperCase()
String toLowerCase()
String trim()
static String valueOf(int)
static String valueOf(long)
static String valueOf(double)
static String valueOf(float)
static String valueOf(boolean)
static String valueOf(char)
static String valueOf(char[], int, int)
static String valueOf(char[])
boolean equalsIgnoreCase(String)
void getChars(int, int, char[], int)

System

static long currentTimeMillis()
static long nanoTime()
static void exit(int)

TreeSet<T>

TreeSet()
TreeSet (Collection<T>)
methods in Set<T>

TreeMap<T,U>

TreeMap()
TreeMap(Map<T,U>
methods in Map<T,U>