UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2004

BEng Honours Degree in Computing Part I
MEng Honours Degrees in Computing Part I
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute

PAPER C123

PROGRAMMING II

Tuesday 27 April 2004, 10:00
Duration: 90 minutes
(Reading time 5 minutes)

Answer THREE questions

Paper contains 4 questions
Calculators not required

Section A *(Use a separate answer book for this Section)*

1a i) Define the Abstract Data Type (ADT) *Stack* and give the Stack's access procedure headers with post-conditions, specifying also the exception cases.

ii) Give the main axioms that any implementation of the ADT *Stack* must satisfy.

iii) Define what an ADT *Binary Tree* is, and give the headers with post-conditions of the *Binary Tree*'s selector access procedures. Specify exception cases where needed.

iv) Explain the difference between a perfectly balanced tree and a complete tree.

b Assume the availability of an ADT *Stack* with its own access procedures. Write the Java code for the following high-level access procedure.

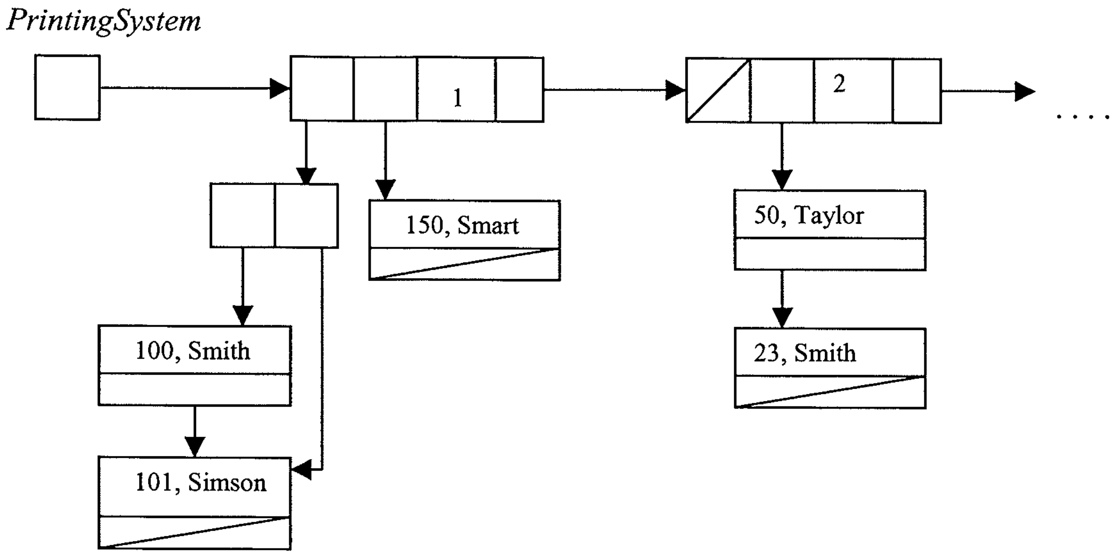
> *Stack* ToweringStacks(*Stack S1*, *Stack S2*)
> //post: Returns a *Stack* given by the elements of *S2* on top of the elements of *S1*, with
> //the elements in each substack in the same order as in the original stacks.

c Assume the availability of ADTs *Binary Tree*, *Stack* and *List* with their respective access procedures. Assume the order of access in a list to be from the first to the last item in the list. Write the Java code for the following high-level access procedure, using an iterative in-order algorithm for traversing a binary tree.

> *List* BTreetoList(*BinaryTree T*)
> //pre: *T* is a Binary Tree
> //post: Returns a *List* containing the items of *T*, with the access order of *L*
> //identical to the in-order access traversal of *T*; *T* is left unchanged.

The three parts carry, respectively, 30%, 20% and 50% of the marks.

2 A printing system in a building has to manage the printing jobs over 5 different printers. The printers are uniquely numbered from 1 to 5. Each printer has its own queue of pending printing jobs and a "log" list of the jobs already completed. The log is organized so that the access order to the list (i.e. from the first to the last element) reflects the chronological order in which the jobs have been completed. Each job includes an *id* number (of type Integer) and the *owner*'s name (of type String). The diagram below gives a partial example of such organization of the system. Printer 1 includes two pending jobs, and one completed job. Printer 2 has only two completed jobs and no pending jobs.



a. Assume *PrintingSystem* to be the definition of an ADT. Give the Java type declarations for this ADT *PrintingSystem* in full, including the type declarations of all the data structures needed to implement it. Do not encapsulate the declaration of a printing job into a separate type.

b. Assume now the existence of a Queue ADT. Give the Java type declaration for the ADT *PrintingSystem* using this standard ADT.

c. Write the Java code for the *PrintingSystem*'s access procedure *ProcessingNextJob* specified below. To do so, use the Java type declaration you have given to part (b).

public void ProcessingNextJob(int printerNumber)
//pre: *printerNumber* is the number of a printer in *PrintingSystem*
//post: The next pending job is removed from the queue of pending jobs of the printer
//numbered *printerNumber* and added to the "log" of completed jobs of this same printer

Give also the implementation of all auxiliary method or other access procedures of the ADT *PrintingSystem* that are needed for *ProcessingNextJob*.

The three parts carry, respectively, 30%, 20%, 50% of the marks.

Section B *(Use a separate answer book for this Section)*

3a Give three differences between abstract classes and interfaces.

b i) Explain what *type casting* is.

ii) Give one reason why up-casting may be required in a Java program and provide a small portion of code to support your explanation.

iii) Give one reason why down-casting may be required in a Java program and provide a small portion of code to support your explanation.

c Given the following method declaration:

```
void sendTxtMessage(String s, MobileNumber n)
                throws InvalidNumber, TxtTooLong;
```

and the following piece of code:

```
void sendLongTxtMessage(String s, MobileNumber n){
    sendTxtMessage(s, n);
}
```

Complete the method `sendLongTxtMessage` so that:

i) it propagates to the method caller InvalidNumber

ii) it catches the exception TxtTooLong and resends the message by splitting it in half. You may need to use the following methods of class String.

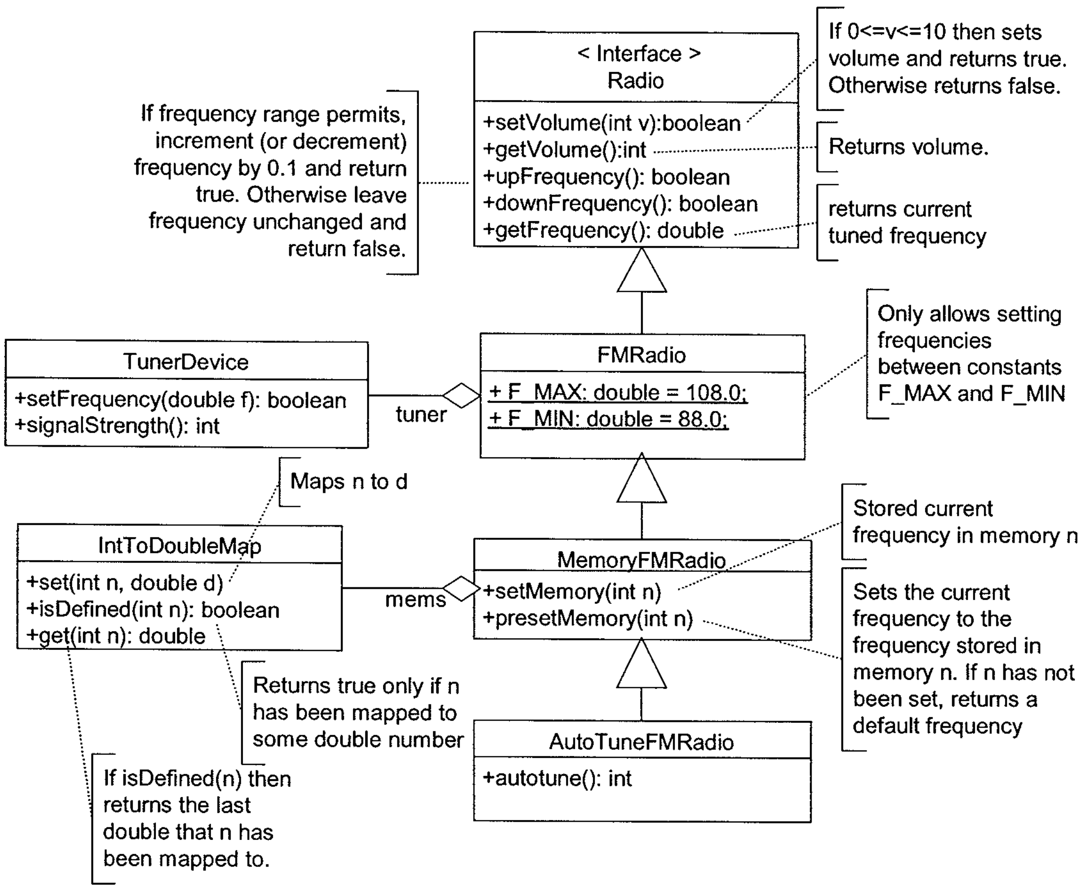
- `String substring(int from, int to)`. Example of use: `s.substring(0,3)` returns `''Good''` if `s = ''Goodbye''`.
- `int length()`. Returns the length of the string.

d Explain why it is incorrect to access a non-static field from a static method as exemplified below.

```
public class PlayChess {
    Chess myChess;
    public static void main(String [] args) {
        myChess = new Chess();
        myChess.play();
    }
}
```

The four parts carry, respectively, 20%, 40%, 20%, 20% of the marks.

4 In this question you are asked to write some of the Java code for the classes that implement a product family of radios. When writing the code, you must take into account all the information provided in the diagram below.


< Interface >
Radio
+setVolume(int v):boolean
+getVolume():int
+upFrequency(): boolean
+downFrequency(): boolean
+getFrequency(): double
If 0<=v<=10 then sets volume and returns true. Otherwise returns false.
Returns volume.
returns current tuned frequency
If frequency range permits, increment (or decrement) frequency by 0.1 and return true. Otherwise leave frequency unchanged and return false.
TunerDevice
+setFrequency(double f): boolean
+signalStrength(): int
tuner
FMRadio
+ F_MAX: double = 108.0;
+ F_MIN: double = 88.0;
Only allows setting frequencies between constants F_MAX and F_MIN
Maps n to d
IntToDoubleMap
+set(int n, double d)
+isDefined(int n): boolean
+get(int n): double
mems
MemoryFMRadio
+setMemory(int n)
+presetMemory(int n)
Stored current frequency in memory n
Sets the current frequency to the frequency stored in memory n. If n has not been set, returns a default frequency
Returns true only if n has been mapped to some double number
If isDefined(n) then returns the last double that n has been mapped to.
AutoTuneFMRadio
+autotune(): int

a Write the Java code for interface `Radio`.

b Write the Java code for class `FMRadio`. An `FMRadio` contains a `TunerDevice` that is used to tune into the frequency desired by the radio user. The radio user controls the tuning frequency through methods `upFrequency()` and `downFrequency()`.

c Write the Java code for `MemoryFMRadio`. The objects of this class allow storing frequencies in preset memories. For example, if the `MemoryFMRadio` is tuned on radio XFM 104.9 and if method `setMemory(2)` is called, the object will store in memory 2 the frequency 104.9. If at a later stage method `presetMemory(2)` is called, the current frequency of the object is set to 104.9 (provided that `setMemory(2)` has not been called in between).

d Write the Java code for `AutoTuneFMRadio`. This is a `MemoryFMRadio` that can set its preset memories automatically. Method `autotune()` will scan through the frequency range using `upFrequency()`, and every time it encounters a frequency that has a local maximum signal strength greater than 8 it will store it in the next available memory. The first memory location used by `autotune()` is 1, memory locations are used consecutively. Method `autotune()` returns the number of frequencies that were stored in memory.

A frequency with local maximum signal strength is one in which the previous and next frequencies have a weaker signal strength. As an example consider the signal strengths for the lower frequencies of the following table. The frequencies with local maximum signal strengths are 88.2 and 88.6. However, method `autotune()` would only store 88.6 in a preset memory because signal strength for 88.2 is not greater than 8.

Assume that there can never be two contiguous frequencies with the same signal strength.

frequency	88	88.1	88.2	88.3	88.4	88.5	88.6	88.7	88.8
signal strength	1	3	5	2	6	8	9	5	4

The four parts carry, respectively, 15%, 25%, 30%, 30% of the marks.