

THIS PAPER IS NOT TO BE REMOVED FROM THE EXAMINATION HALLS

UNIVERSITY OF LONDON

291 0212 ZB

BSc Examination
for External Students

**COMPUTING AND INFORMATION SYSTEMS AND
CREATIVE COMPUTING**

Programming: Advanced Topics and Techniques

Dateline: Monday 11 May 2009 : 2.30 – 5.30 pm

Duration: 3 hours

Candidates should answer **SIX** questions. Full marks will be awarded for complete answers to **SIX** questions.

Candidates must answer **THREE** questions from **Section A** and **THREE** questions from **Section B**. In Section B you must answer **ONE** question on Prolog (questions 9 and 10).

There are 150 marks on this paper.

Claculators may NOT be used.

© University of London 2009

UL09/893

Page 1 of 12

SECTION A

Question 1

- (a) Which keyword is used when exceptions are declared in Java? [1 mark]
- (b) Explain how exceptions are used for handling errors, such as incorrect input by the user, making reference to the keywords *throw* and *catch*. [6 marks]
- (c) Consider the *AbsVal* class below:

```
public class AbsVal
{
    static int  onlyPositive (int x) throws NewException
    {
        if (x<0) throw new NewException();
        return x;
    }

    public static class NewException extends
    RuntimeException
    {
        public NewException (String msg)
        {
            super(msg);
        }
        public NewException() {}
    }
} // end of class
```

- (i) What will happen if *onlyPositive()* is called with a negative number? [2 marks]
- (ii) Explain how exceptions can give useful information to the programmer when debugging code, using the *AbsVal* class as an example. What two changes could you make to the code in order to give useful information to the programmer if an error is thrown? [8 marks]

(question continues on next page)

- (d) Consider the recursive method *fact()* below, which is intended to return the factorial of a positive number, and to throw an exception if a negative number is entered (assume that the Exception has been defined).

```
public int fact (int a) throws NegativeException
{
    if (a < 0) throw new NegativeException
        ("positive numbers only");
    else return a*fact(a-1);
}
```

- (i) Explain why the method will not do what it is intended to do [6 marks]
- (ii) Explain how the method could be corrected [1 mark]
- (iii) Write the corrected the code so that the method will work as intended. [1 mark]

NB: The factorial of x is written $x!$ and is defined to be $x(x-1)(x-2)\dots 2 \times 1$.
Therefore $4! = 4 \times 3 \times 2 \times 1 = 24$

Question 2

- (a) Explain how design patterns help us to re-use ideas [4 marks]
- (b) What common programming problem is the iterator design pattern intended to solve? [4 marks]
- (c) Explain briefly how Java implements the iterator design pattern. [4 marks]
- (i) Which design pattern does the class *Induction* implement? Justify your answer with reference to the methods of the *Induction* class. [4 marks]

```
abstract class Induction
{
    void inductionPack()
    {
        System.out.println ("please collect your
        induction pack on arrival from reception");
    }

    void history()
    {
        System.out.println ("please read the book on
        company history in your induction pack");
    }

    void healthSafety()
    {
        System.out.println ("please read the health
        and safety information carefully");
    }

    abstract void reportToRoom();
}
```

- (ii) Define a concrete sub-class *Clerk* that has to report to room B110, and another concrete sub-class *Manager* that has to report to the Boardroom. The *Manager* class has an additional method *CompanyCar* that prints a message telling the employee to collect their car keys from reception on arrival. [9 marks]

Question 3

(a) Explain each of the following:

(i) inheritance for extension; [3 marks]

(ii) inheritance for specialization; [3 marks]

(iii) inheritance for specification. [3 marks]

(b) Consider the following class, *Mammal*:

```
class Mammal
{
    boolean spine;
    boolean liveBirth;
    boolean warmBlooded;

    public Mammal()
    {
        spine = true;
        liveBirth = true;
        warmBlooded = true;
    }

    boolean hasFur()
    {
        return true;
    }
}
```

(i) Using the *Mammal* class write a *human* subclass with a method *hasFur* that returns false. [4 marks]
NOTE: You do not need to write a constructor for the subclass

(ii) Does the subclass *human* demonstrate inheritance for extension or for specialisation? Justify your answer. [4 marks]

(iii) Using the *Mammal* class write a subclass *Panda* with a boolean *eatsBamboo* method that takes no parameters and returns true. [4 marks]
NOTE: You do not need to write a constructor for the subclass.

(iv) Does the subclass *Panda* demonstrate inheritance for specialisation or for extension? Justify your answer. [4 marks]

Question 4

- (a) The *swap* method swaps two items in a given array. It is documented by its **REQUIRES**, **EFFECTS** and **MODIFIES** comments.

```
public static void swap(int[]a, int i, int j)
{
    // REQUIRES: 0 <= i,j < a.length
    // EFFECTS: Swaps the contents of a[i] and a[j]
    // MODIFIES: a
}
```

Write the *swap* method.

[8 marks]

- (b) I intend to write a *power* method that will take two positive integers, *a* and *b*. The method will return an integer, *a* raised to the power of *b*, ie *a* times itself *b* times, such that *power*(2, 3) will return 8. If *b* = 0 the method will return 1. The method requires that both *a* and *b* are integers that are greater than or equal to zero, but does not check to see if the input is correct.

[8 marks]

Document the method using **REQUIRES**, **EFFECTS** and **MODIFIES** comments.

- (c) Write the *power* method as a **recursive** method (no credit will be given for iterative methods).

[9 marks]

Question 5

Implement a class to do the following (you may answer all questions within one class if you wish):

- (a) In a 400 x 400 JFrame, display a rectangle with the parameters 100, 100, 40, 60 [10 marks]
- (b) Place a button with no functionality into the JFrame; [5 marks]
- (c) Add the following functionality to the button: when it is pressed the parameters of the rectangle change to 100, 100, 60, 40 [10 marks]

SECTION B

Question 6

a)

i) What is an *infix* function? Give one example of a predefined infix function in SML.

[3]

ii) Describe, giving examples, the concepts of *association* and *precedence* that need to be considered when defining a function as *infix*.

[4]

iii) Describe the syntax required to convert a function of two variables into an *infix* one, explaining clearly how issues of association are dealt with.

[4]

iv) Define an infix function *q* that given two integers, (*x* and *y*, say) returns the difference between their squares so that *1 q 2* results in -3 (that is 1-4).

[2]

v) Explain how two different results might be expected of the evaluation of *1 q 2 q 3* and the circumstances under which they might be obtained.

[2]

b)

i) Explain the concept of currying a function, giving examples and explaining the notation used in SML.

[6]

ii) Consider the SML functions defined by:

*fun h(x, y) = x*x + y*y;*

*fun g x y = x*x + y*y;*

Give, with explanations, the results (values, types and errors returned) of evaluating the following:

h;

g;

h 4;

g 4;

h 2 3;

g 2 3;

h(3, 4);

g(3, 4);

[4]

Question 7

a) SML is said to be a *strongly typed language*. Giving suitable examples, explain what this means and give an example of a feature of SML that breaks the ideal of strong typing.

[4]

b) Consider the SML code:

```
abstype bag = emptybag | add of int * bag
with
  val empty1 = emptybag;
  fun member(x, emptybag) = false |
      member(x, add(element, bag)) = x = element orelse member(x, bag);
  fun empty(emptybag) = true | empty(bag) = false;
  fun next(add(element, bag)) = element;
  fun add1(element, bag) = add(element, bag);
end;
```

i) Explain the meaning and use of each line of the SML code above.

[5]

ii) Give the output produced by the following evaluations:

```
empty1;
val a1 = add1(2, empty1);
val a1 = add1(1, a1);
val a1 = add1(2, a1);
val a1 = add1(1, a1);
if not (empty(a1)) then (a1, next(a1)) else (empty1, 0) ;
```

[5]

iii) Write an SML function *setit(bag)* which, given a *bag* returns a list of the contents of *bag* in increasing order and with duplicates left out. For example *setit(add1(2,add1(1,add1(2,empty1))))* should return *[1, 2]*. You may need to write helper functions to do this.

[11]

Question 8

a) Distinguish between the SML concepts of List, Tuple and Record, giving examples of their definition and use.

[10]

b) Define a record type 'student' that holds the following pieces of information: *student number, name, date of birth, courses taken and results obtained.*

(You will need to decide on the type (and possibly structure) of each of these.)

[6]

c) Define functions that extract each component.

[2]

d) Define a function *average* that, given such a record, produces the average result of examinations taken by the student and give an explanation of how it works.

[7]

Question 9

a)

- i) Give the rules that determine when two Prolog terms match. [4]
- ii) State which, if any, of the following pairs match, giving your reasoning fully for each pair: [6]
- $h(i,j(K))$ with $h(K, L)$.
 - $m([n|p],q)$ with $m([R],S)$.
 - $t(u(V))$ with $t(W)$.
 - $a(b,[c, D],e(F,g))$ with $a(b,D,e(f,C))$

b) Consider the following Prolog predicate:

```
insert(H, [], [H]).
insert(H,L, [H|L]).
insert(H, [F|T], [F|R]):-insert(H,T,R).
```

- i) Describe the relationship between the three parameters of *insert* above. [3]
- ii) Trace the execution of the following query if ';' is typed after each solution is presented: *insert(l,[a,b,c], L)*. [6]
- iii) Modify *insert* so that duplicate solutions are not found for this query. [2]
- iv) Write a Prolog predicate *permute(L, P)* which takes two lists as parameters and returns list *P* as a permutation of list *L* in such a way that if there are no duplicates in *L* there are none in the set of solutions returned if ';' is typed as each solution is presented.

So query *permute([a,b,c], P)* results in the following exchange:

```
?- permute([a,b,c],P).
P = [a, b, c] ;
P = [b, a, c] ;
P = [b, c, a] ;
P = [a, c, b] ;
P = [c, a, b] ;
P = [c, b, a] ;
no
?-
```

[4]

Question 10

Prolog is claimed to be a 'a declarative language based upon predicate logic' (212b guide page 79).

a) Justify this statement by giving those features that are in accord with this description.

[8]

b) On the other hand there are aspects of Prolog that do not fit well with this statement.

i) Give 3 of these aspects together with a description of why you think they do not fit the statement.

[6]

ii) For each of the three aspects that you gave in i) above, describe why that aspect is important to Prolog and how it is used by programmers.

[6]

c) Distinguish between the declarative and the procedural readings of a Prolog program, and explain why a programmer needs to be able to use both readings when writing or debugging Prolog programs.

[5]

END OF EXAMINATION