UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1999

MSc Degree in Computing Science
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Diploma of Membership of Imperial College

PAPER COMP I

PROGRAM DESIGN AND LOGIC
Friday, May 14th 1999, 10.00 – 12.00

Answer THREE questions
Answer at least ONE question from Section A
Answer at least ONE question from Section B

For admin. only:
paper contains 4 questions

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

1 Consider the following simplified description of bank accounts:

Banks hold accounts owned by people. A bank has a name and a daily interest rate. Daily interest rates may change.

A person has a name, and may own more than one account. Each account is owned by one person. Every person has an overdraft limit, which sets a limit to the amount this person may be overdrawn on any of his/her accounts. Note that the overdraft limit does not depend on banks or accounts.

Money may be paid into or withdrawn from an account.

When an amount is paid into an account on a given date, then interest is calculated as the product of the number of days from the last transaction until the current transaction, the interest rate of the holding bank, and the balance before the payment. Then, the interest and the amount are added to the balance.

When an amount is withdrawn from an account on a given date, then interest is calculated as the product of the number of days from the last transaction until the current transaction, the interest rate of the holding bank, and the balance before the withdrawal. Then, the interest is added to the balance, and then the balance is decreased by the amount, provided that the balance does not fall below the owner's overdraft limit; otherwise, an error message is printed.

a Develop an OMT object model class diagram to describe the above. A summary of the OMT notation may be found on page 3 of this paper.

b Write C++ classes (i.e. declarations and function bodies) to support the above.

For the representation of dates use a class `Date` defined as follows:

```
class Date{
    Date(int day, int month, int year) { . . . }
    int daysUntil(Date& aDate){ . . . }
    // returns number of days between receiver and aDate
    . . . };
```

c Write a test function that:

i) creates two banks: the Listening Bank with an interest rate of 0.030, and the Talking Bank with an interest rate of 0.025,

ii) creates two people: Kenneth with an overdraft limit of 100.00, and Gordon with an overdraft limit of 45,000.00,

iii) creates A1, an account for Kenneth, with the Listening Bank, on the 2nd March 1995, with an initial balance of 230.00, and creates A2, an account for Gordon, with the Talking Bank, on the 5th May 1998, with an initial balance of 56,000.00,

iv) withdraws 300.00 from A1 on the 10th January 1999, and pays 500.00 into A2 on the 4th March 1999,

v) sets the interest rate of the Talking Bank to 0.035.

Note: you do *not* need to use lists or any other form of collection classes.

The three parts carry, respectively, 20%, 60% and 20% of the marks.

2 Consider the following aspects of airline flight bookings:

i) A flight has a code number, a date and a price.

ii) A passenger has a name and an address. A passenger may be booked on more than one flight, but may not be booked more than once on the same flight.

iii) There are firm bookings, where the whole price of the flight has been paid, and flexible bookings, where only a deposit (smaller than the flight price) has been paid. A flexible booking may be cancelled up to 10 days before the date of the flight; then 80% of the deposit paid will be returned. A firm booking may only be cancelled up to 20 days before the date of the flight, and then the complete flight price will be returned.

iii) When printing information about a flight, then the fight number and date are printed, followed by the name, address and the outstanding amount (ie price of flight - deposit) for each of the passengers booked.

a Draw an OMT object model class diagram describing the above. A summary of the OMT notation may be found on page 3 of this paper.

b Write C++ class declarations *(but no function bodies)* to implement the above. You may use `List<A>` to indicate a class that contains a list of elements of a class or type `A`, and assume the existence of a class `Date` defined as follows:

```
class Date{
    Date(int day, int month, int year) { . . . }
    int daysUntil(Date& aDate){ . . . }
    // returns number of days between receiver and aDate
    . . . };
```

c Write a test function with:

i) F1, a flight with code 555 to take place on the 15th February 2000 and which costs 560 pounds, and a flight F2, with code 666 to take place on the 15th March 2000 and costing 780 pounds,

ii) a passenger Mr Smiley, living on 23 Humour Street, booked firmly on F1,

iii) a passenger Miss Grumpy, living on 44 Complaints Avenue, having paid 400 pounds and booked flexibly on F1, and booked firmly on F2,

iv) information about F1 to be printed,

v) Mr Smiley's booking on F1 to be cancelled on the 15th January 2000.

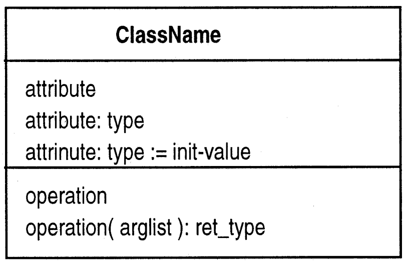
Hint: Since you will not *write the method bodies, your solution* cannot *cover* all *aspects described above.*

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

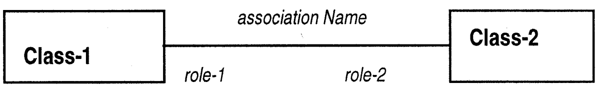
Please turn over ...

OMT: Basic Notation for Object Models

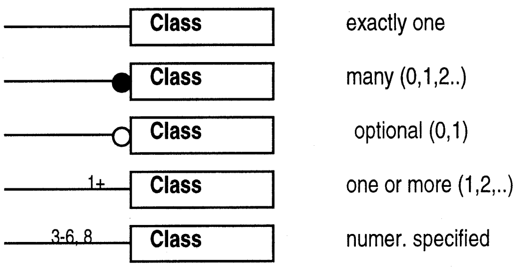
Class:



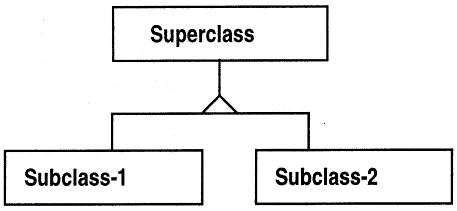
Association



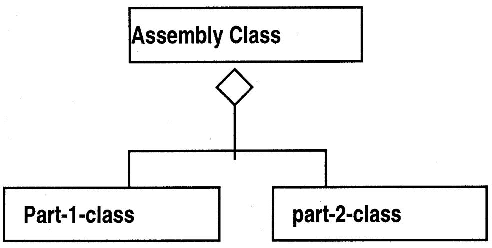
Multiplicity of Association/Aggregation



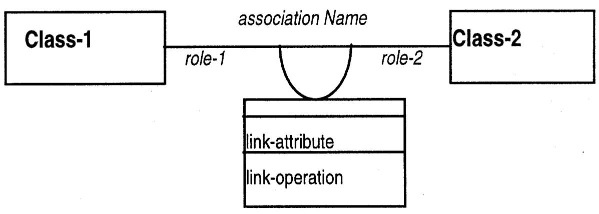
Generalization (Inheritance)



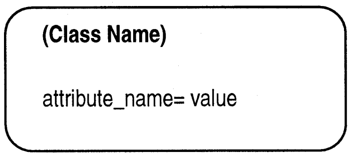
Aggregation



Link Attributes

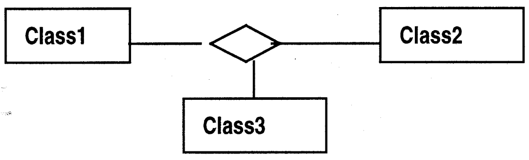


Object Instance



$ for class operations/attributes

Ternary Association



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

3 a Using equivalences only, show that the following equivalence holds

$$((A \wedge \neg B) \vee (C \to B)) \to \neg(A \wedge C \to B) \equiv C \wedge \neg B.$$

b i) Using inference rules only, show

$$P \vdash W \to P$$

where P and W are any wffs of propositional logic. Explain every step of your proof clearly by giving the inference rules and the wffs used.

ii) Show

$$(C \to B) \to \neg(A \wedge C \to B) \vdash \neg B$$

using inference rules, and, if required, b(i), only. Explain every step of your proof clearly by giving the inference rules and the wffs used.

c Using inference rules only, show

$$P1, P2, P3, P4 \vdash \forall X\, (p(X) \to \exists Y\, n(X,Y))$$

where P1-P4 are as follows.

P1	$\forall X\, (p(X) \to ((\exists Y\, \neg q(X,Y)) \vee r(X)))$
P2	$\forall X\, (r(X) \to \forall Y\, n(X,Y))$
P3	$\forall X \forall Y\, (\neg m(X,Y) \to q(X,Y))$
P4	$\forall X \forall Y\, (m(X,Y) \to n(X,Y))$

Explain every step of your proof clearly by giving the inference rules and the wffs used.

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

4a Translate the following sentences (i)-(v) into predicate logic, using the following predicates:

lecturer(X):	X is a lecturer
course(X):	X is a course
student(X):	X is a student
happy(X):	X is happy
likes(X,Y):	X likes Y
teaches(X,Y):	X teaches course Y
la(X):	X is on leave of absence
examines(X,Y):	X examines course Y
doc(X):	X is a member of the computing department

i) Any lecturer is happy if some student likes at least one of the courses they teach.

ii) No student likes all the courses taught by any one lecturer.

iii) Every course is taught by at least one and at most two lecturers.

iv) All lecturers, except those who teach statistics or multimedia, are members of the computing department, and all lecturers will examine all the courses they teach unless they are on leave of absence.

v) Some, but not all, lecturers are on leave of absence. Some of those on leave of absence will not teach any courses, but all of them are happy.

b i) Write a Prolog program for the predicate *insert(I,L,LNEW)*, that, given an item *I* and a list *L* which is already sorted in ascending order, generates a list *LNEW* which includes item *I* and is in ascending order. You can assume that *I* and the elements of *L* can be compared using Prolog's built-in predicates <, >, =, and that *L* has no duplicate elements. *LNEW* should have no duplicate elements. You may use Prolog's built-in predicates <, >, =, *append*, but no others.

ii) Modify your program for *insert* to write a program for *insertNcount(I, L, LNEW, N)*, which is identical to *insert* except that *I* is added to *L* to produce *LNEW* regardless of whether or not *I* is already present in *L* (so *LNEW* can have duplicate elements), and *N* is the number of occurrences of *I* in *LNEW*. You can assume that *I* and the elements of *L* can be compared using Prolog's built-in predicates <, >, =. You may use Prolog's built-in predicates <, >, =, *append*, *is*, but no others.

Parts a and b carry equal marks.

End of paper