UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1997

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
BSc Honours Degree in Mathematics and Computer Science Part II
MSci Honours Degree in Mathematics and Computer Science Part II
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 Royal College of Science
Associateship of the City and Guilds of London Institute

PAPER 2.1 / MC2.1

SOFTWARE DESIGN I

Thursday, May 1st 1997, 2.00 - 3.30

Answer THREE questions

For admin. only: paper contains 4 questions

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

1a i) Name and briefly explain three principles of good software design.

ii) What principles does an 'object' in object-oriented development embody? What principle does a 'class' embody?

b You are asked to design the software for controlling access to Huxley Building using swipe cards. Every person has exactly one swipe card. Swipe cards must be valid in order to be used. Staff have 24 hour access to the building, while students can only enter/exit between 6am and 11pm. Only postgraduate students can extend their access hours with permission from their personal tutor who is a member of staff.

Draw a simple *Class Diagram* that models the requirements above. For each class, identify at least one attribute and one operation (either or both may be inherited). Identify an *Abstract Class* in your diagram.

c Students enter the building (into a corridor) to attend lectures or labs. When it is time for a lecture, if a student is sleepy he/she goes to a lecture room and sleeps until the session is over. Similarly, when it is time for a lab, if the student is hungry he/she goes to the laboratory to eat until the session is over. At the end of a lecture or lab, students return to the corridor. While in the corridor, the students spend their time studying.

i) Draw a *state transition diagram* to model the dynamics of the above scenario.

ii) Briefly explain why clustering is used in *Statecharts*, illustrating how it helps in the above example.

The three parts carry, respectively, 30%, 30%, 40% of the marks.

2a Briefly describe the *waterfall model* of development and give one reason why many software engineering methods assume software is developed in this way.

Name and briefly describe an alternative model of development that addresses the weaknesses of the waterfall model.

b i) A transport organisation wants to design a computer-based train ticketing system. The system must allow passengers to make an enquiry about their journey, and pay for their selected ticket if they wish to travel. The system should issue paper-based tickets and alert staff if the ticket supply is running low.

Draw a simple *Context Diagram* to model the high-level functionality of the system.

ii) The transport organisation needs to impose the following set of safety requirements:

"If there is a fire on a train and it is moving, then all the train doors should be closed. If there is a fire on a train and it is not moving, then the doors should open. If the train is not moving and one of the doors is jammed, then all the doors should be open. If a train is moving, then all doors must be closed."

Draw a *Decision Table* which indicates the conditions under which train doors should be open, and the conditions under which they should be closed.

Show how you check that your table is complete and consistent, indicating whether or not this is the case for the table specified. Suggest how you might correct your specification, if necessary.

The two parts carry, respectively, 35% and 65% of the marks.

Turn over ...

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

3a Write classes and their associated methods in Smalltalk which describe the behaviour of foxes and chickens when they are woken up:

Foxes and chickens are animals.

Every animal has a position (an instance of class `Position`), a weight, and an amount of undigested food in its stomach.

If the amount of undigested food multiplied by the hunger factor is smaller than the weight of the animal, then this animal is hungry. The hunger factor is different in foxes and in chickens, and may vary according to conditions.

When a fox is woken up, it digests one unit of its undigested food. Afterwards, if it is not hungry it does nothing; if it is hungry and the next position is not occupied by another animal, it moves to the next position; if it is hungry and the next position is occupied by another animal, it attacks this animal.

When a chicken is woken up, it digests one unit of its undigested food. Afterwards, if it is not hungry it does nothing; if it is hungry and the next position is not occupied by another animal, it moves to the next position; if it is hungry and the next position is occupied by another animal, it retreats.

b Assuming that `P1` and `P2` are global variables pointing at objects of class `Position`, and that `Fiffie` and `Charlie` are global variables, write message expressions to do the following:

i) Initialize the animal world and set the hunger factor for chickens at `3` and for foxes at `1`.

ii) Assign to `Fiffie` a fox at position `P1` with weight `6` and undigested food `3`, and to `Charlie` a chicken at position `P2` with weight `4` and undigested food `4`.

iii) Wake up `Fiffie`; wake up `Charlie`.

iv) Set the hunger factor for chickens at `2` and wake up `Fiffie`.

The two parts carry, respectively, 80% and 20% of the marks.

Assume that methods for attacking and retreating are already defined for animals. Use the classes `Dictionary` and `Position` with the following protocol:

```
class name                Dictionary
instance methods
    at: key
    "key: <Object>, returns the object which is stored under key"
    at: key put: anObject
    "key: <Object>, anObject: <Object>
    "stores anObject under key"
    removeKey: aKey
    "removeKey: <Object>, removes the entry under aKey"
class name                Position
instance methods
    next
    "returns the next Position"
```

4 Consider the following description of the management of bank accounts:

There are several banks holding accounts for people. A person may apply to a bank for an account, and if they do not already possess an account with this bank, a new account is created, with the applicant as the owner.

The owner may query the bank for the balance, pay in money, or withdraw money from his account. Everybody may pay money into some somebody's account, but they may not query the balance or withdraw money, unless given the appropriate rights. The owner of an account may request the bank to give to a certain person query rights (which allows the second person to query the balance of the owner's account), or withdraw rights (which allow the second person to withdraw money from the owner's account).

Each bank has its interest rate. Interest is calculated by augmenting the balance of an account by the product of the balance and the bank's interest rate - we deliberately disregard the time aspect of interest calculation.

A person may hold more than one account, provided that each is held at a different bank. The spending capacity of a person is the sum of the balance of all accounts held by that person.

a Design Smalltalk classes to describe the above, *i.e.* determine the classes, the class hierarchy, the instance and class variables, the instance and class methods, but *without providing the method bodies*; for each instance/class variable and each argument specify the class it belongs to.

b Write message expressions for the following

i) Create two banks: the Listening Bank, with an interest rate of 0.03, and the Talking Bank, with an interest rate of 0.025.

ii) Create two people, John and Kenneth..John applies to the Listening Bank for an account, and pays £100.00 into it.. John applies to the Talking Bank for an account and pays £300.00 into it.. Kenneth applies to the Listening Bank for an account, and pays £50.00 into it

iii) John withdraws £40.00 from his account with the Talking Bank. John calculates his spending capacity. John pays £60.00 into Kenneth's account with the Listening Bank.

vi) Create a new bank, called the Debating Bank, with an interest rate of 0.033. John applies to the Debating Bank for an account, and pays £100.00 into it. John gives to Kenneth the right to query the balance of his account with the Debating Bank.

The two parts carry, respectively, 75% and 25% of the marks.

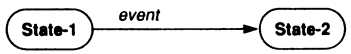
Remember that the classes `Dictionary` and `Set` provide the following protocols:

```
class name                  Dictionary
superclass name             ...
instance methods
    at: key
    "key: <Object>, returns the object stored under key"
    at: key put: anObject
    "key: <Object>, anObject: <Object>
    stores anObject under key"
class name                  Set
superclass name             ...
instance methods
    add: anObject
    "anObject: <Object> adds anObject to the receiver"
    do: aBlock
    "aBlock: <Block>, applies aBlock to each element in the receiver"
```

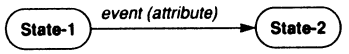
End of paper

Dynamic Model Notation

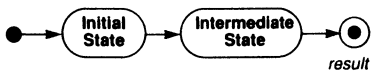
Event causes Transition between States:



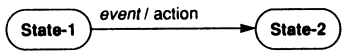
Event with Attribute:



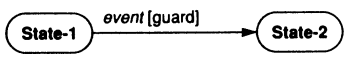
Initial and Final States:



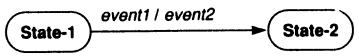
Action on a Transition:



Guarded Transition:

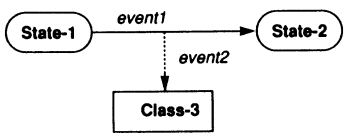


Output Event on a Transition:

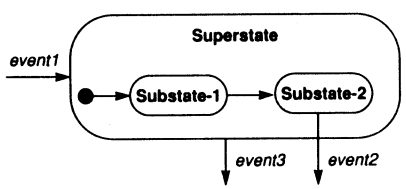


Actions and Activity while in a State:

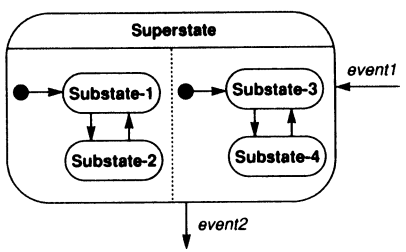
Sending an event to another object:



State Generalization (Nesting):

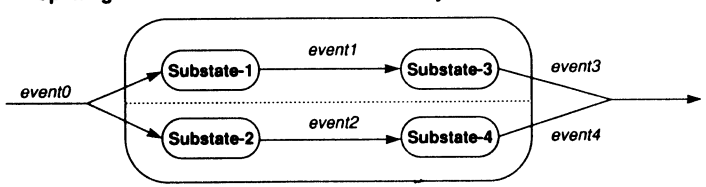


Concurrent Subdiagrams:



Splitting of control:

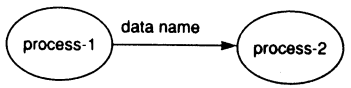
Synchronization of control:



Functional Model Notation

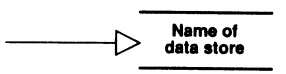
Process:

Data Flow between Processes:



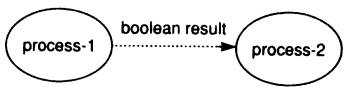
Data Store or File Object:

Data Flow that Results in a Data Store:

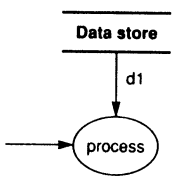


Actor Objects (as Source or Sink of Data):

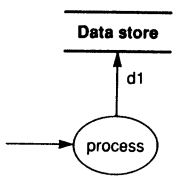
Control Flow:



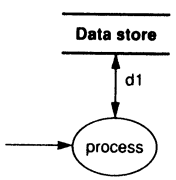
Access of Data Store Value:



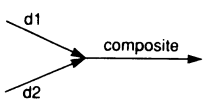
Update of Data Store Value:



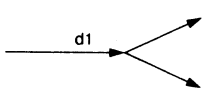
Access and Update of Data Store Value:



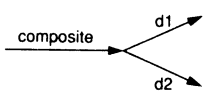
Composition of Data Value:



Duplication of Data Value:

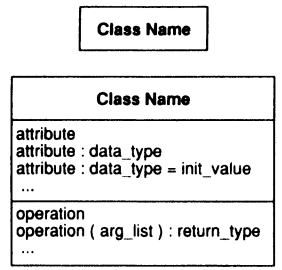


Decomposition of Data Value:

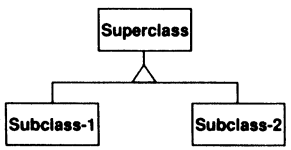


Object Model Notation
Basic Concepts

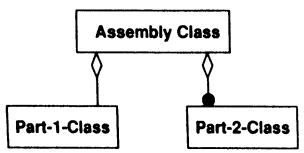
Class:



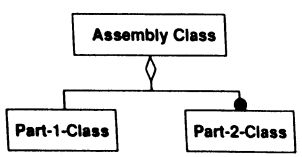
Generalization (Inheritance):



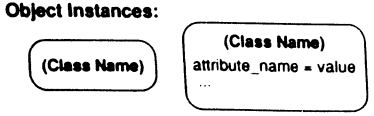
Aggregation:



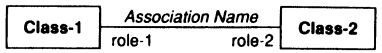
Aggregation (alternate form):



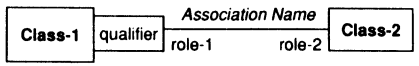
Object Instances:



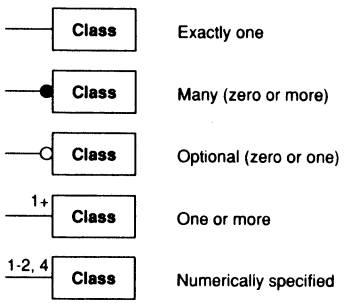
Association:



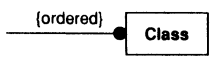
Qualified Association:



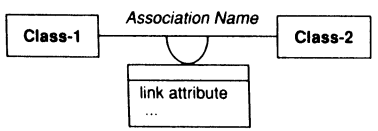
Multiplicity of Associations:



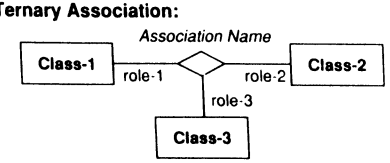
Ordering:



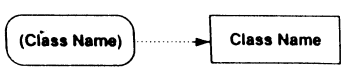
Link Attribute:



Ternary Association:

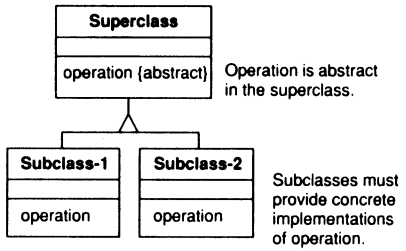


Instantiation Relationship:

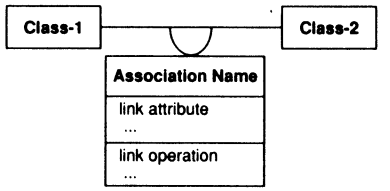


Object Model Notation
Advanced Concepts

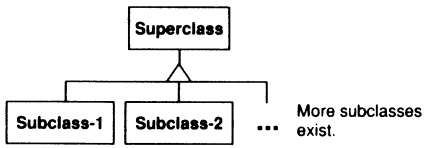
Abstract Operation:

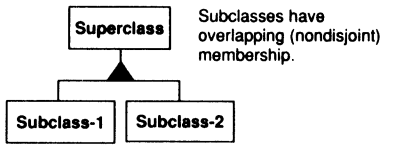


Association as Class:

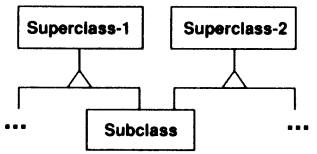


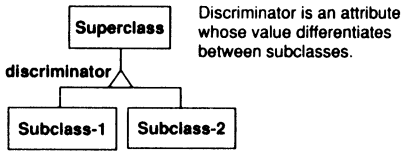
Generalization Properties:



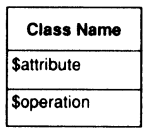


Multiple Inheritance:

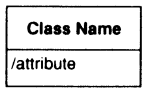




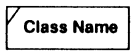
Class Attributes and Class Operations:



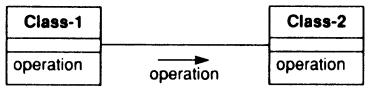
Derived Attribute:



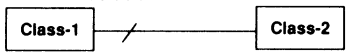
Derived Class:



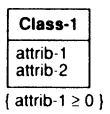
Propagation of Operations:



Derived Association:



Constraints on Objects:



Constraint between Associations: