UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2002

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
BEng Honours Degree in Information Systems Engineering Part II
MEng Honours Degree in Information Systems Engineering 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 City and Guilds of London Institute
This paper is also taken for the relevant examinations for the Associateship of the Royal College of Science

PAPER C220=MC220=I2.1

SOFTWARE ENGINEERING - DESIGN I

Monday 22 April 2002, 14:00
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)*

1 In this question you can assume that the following constructs exist:

```
class Collection {
    boolean add(Object o) ;
    boolean remove(Object o);
    boolean contains(Object o);
    boolean isEmpty();
    Iterator iterator();
    int size();
}
```

```
interface Iterator {
    boolean   hasNext();
    Object    next();
    void      remove();
}
```

a An application displays graphs of share prices. It has a *TopLevel* window containing several *Graph* windows, which are tiled horizontally one below the other. The *Graph* windows have equal height and the sum of their heights is equal to the height of the *TopLevel* window. The width of the *Graph* windows is equal to the width of the *TopLevel* window in which they are contained. All windows are characterised by their height, width and the coordinates of the top left corner. Coordinates are positive and measured from the top left corner of the screen. All windows can be resized using a method `void resize(double height, double width)`. When a *TopLevel* window is resized all the *Graph* windows it contains are resized and re-tiled horizontally one below the other.

Assume that the following methods are provided in each of the window classes:

```
void paint();                    /* displays the window on the screen */
void move(double x, double y);   /* moves the window so that its top left
                                    corner is at coordinates x and y */
```

Give a Java implementation for the *TopLevel* and *Graph* window classes, and any super class you may have introduced, showing their attributes, constructors and methods (including the `resize` method).

b *Graph* windows can display several graphs overlaid on top of each other. Each *Graph* window has a collection of data points (the *DataPoint* class is given below) and a collection of graph algorithms. Assume the *Graph* window has a method `void plot(DataPoint d)`, which plots a single point on the graph taking care of all scaling and display aspects.

```
class DataPoint {
  public int day;          // day expressed as an integer
  public double value;     // the closing share price on that day
}
```

All graph algorithms will need to implement the following methods:

```
void setIterator(Iterator it);  /* sets an iterator over
                                   a collection of DataPoints */
DataPoint getNextPoint();       /* returns the next Datapoint */
boolean hasNextPoint();         /* returns true if there is another
                                   datapoint to be drawn      */
```

i) Give the Java implementation of a method `void draw()` of the *Graph* window class which draws the graphs corresponding to all the algorithms.

ii) Give the Java implementation of the classes implementing the following two algorithms and any super class you may have introduced. The first algorithm returns the data points unchanged (i.e. it draws a history of the share price). The second algorithm returns the average price of the share between the current day and the previous day. It returns an error if there are less than two data points.

The two parts carry equal marks.

2 In this question you may define any additional classes and methods you feel are useful.

a Explain the concepts of high-cohesion and low-coupling and show using examples how they are used in software design.

b An art gallery exhibits paintings and sculptures. Both are linked to displacement sensors, which are triggered if the objects are moved. All exhibits (paintings and sculptures) are shown in rooms and every room has one or several motion sensors, which can also be triggered. Each room has an alarm, which is notified when either motion or displacement sensors are triggered. The alarm is connected to a central security system, to which all alarms are notified. When notified of an alarm, the security system creates an alarm report containing the room number in which the alarm occurred and a description of the exhibits that are affected. When a displacement sensor is triggered, only the exhibit linked to that sensor is considered to be affected. When a room motion sensor is triggered, all exhibits in that room are considered to be affected. The security system then appends the alarm report to a log, identifies the closest security guard to the scene of the incident and notifies her on her pager. Assume that objects representing security guards have an attribute indicating the guard's current location.

i) Draw a class diagram of the system's architecture. Indicate the nature and cardinality of all relationships, any abstract classes and indicate which attributes and methods you would expect each object to provide.

ii) In no more than 15 lines comment on your design and discuss any alternatives.

c i) Give a collaboration diagram indicating the invocations occurring when a sensor connected to a painting is triggered. Indicate sequence numbers and transient links if any.

ii) Discuss your design in no more than 15 lines showing what decisions you have made to reduce coupling and increase cohesion.

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

3a For statechart models of a system explain what is meant by *composite states*, what different kinds of composite states there are, why they are useful and when a composite state is said to be *active*.

b The following is a description of the functioning of a portable radio. Draw a state chart modelling its behaviour. State explicitly any additional assumptions that you make.

The radio has five buttons: an *on/off* button, an *up* button, a *mute* button, a *save* button, and a *mode* button. The radio has two modes of operation: it can play the sound received on an arbitrary frequency *(normal* mode*)* or play the sound received on a pre-memorised frequency *(preset* mode*)*. When in *normal* mode, the frequency is displayed and the *up* button will increment the current frequency by a set amount. When in *preset* mode both the current frequency and the memory number are displayed and the *up* button will tune the radio to the next memorised frequency. The *mode* button toggles the radio between the two modes of operation. Each time the mode of operation is changed, the current frequency (and memory number if applicable) is saved and retrieved when returning to that mode.

In both modes, pressing the *mute* button will temporarily suspend the sound. Pressing it again will return the radio into whatever mode it was in before.

Pressing the *save* button while in *normal* mode will first check if the current frequency has already been saved. If it has already been saved, the radio will retrieve the memory number and play the same frequency in preset mode. If the frequency has not been already saved, it will attempt to save the frequency in the first memory. The memory number then flashes on the screen and the *up* button goes to the next memory number. Pressing *save* again will save the frequency into the current memory and continue playing that frequency in *preset* mode.

Pressing the *on/off* button while the radio is on will switch the radio off. Pressing it while the radio is off will switch the radio on and return it into the *normal* or *preset* mode in which it was last playing. However, if the radio is switched off while muted or while memorising a station this information will be lost. When switched back on again the radio will revert to its last playing mode.

c For each composite state in your answer to b) explain why you have decided to use a composite state and how you have determined which sub-states should be part of it.

If you didn't use any composite state in your answer to b) identify where you could have used one and how you would determine which sub-states should be part of it.

The three parts carry, respectively, 20%, 70%, 10% of the marks.

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

4a i) Describe the difference between *state schemas* and *operation schemas*, and explain the standard use of the four symbols ?, !, Δ, and Ξ in specifying operation schemas.

ii) Consider the following schemas, and write the schema disjunction *Div = DivOK ∨ DivError* in full, without using any schema inclusion.

Number
r:**R**

DivOK
ΔNumber
m?: **N**
report! = {Ok, DivbyZero}
m? > 0 ∧ r' = r/m?
report! = OK

DivError
ΞNumber
m?: **N**
report! = {Ok, DivbyZero}
m? = 0
report! = DivbyZero

b A car rental shop uses a system for handling its own car renting process. The system keeps track of the cars that are rented and the cars that are available. It allows a customer to rent one car or to return a rented car. To rent a car a customer identifies the car that he/she wants to rent and the number of days, and receives in return a confirmation message and the outstanding cost to pay. A car can be rented only when it is available. To return a car, the customer again provides the car to be returned and the number of rented days. When a car is returned, the system automatically updates the account of the shop. The system also allows the addition to the shop of new cars for rental. The following schema *CarRentalShop* defines the basic parameters of the system.

CarRentalShop	
fullstock: **F**CARS	(the set of all the cars owned by the shop)
CC: CARS → **N**	(it gives the cubic capacity (cc) of each car)
price: CARS → **R**	(it gives the rental price per day of each car)
∀x: CARS. (price(x) = CC(x)/100)	

i) Write a state schema *SystemState* for the overall state of the system, covering the cases when the system is open and when it is closed. The system is closed when there are no more cars available and open otherwise. In both cases, the schema includes *CarRentalShop*, uses the variables *account* (the shop's account), *CarsAvailable* (the set of cars available) and *CarsRented* (the set of cars rented). Include relevant constraints on the variables.

ii) Write operation schemas for the following operations using a defensive approach. All the schemas include the inputs *car?* and *days?*, and the output *report!*.
Marks will be awarded for correct use of Δ and Ξ.

RentCar returns the outstanding cost to pay, if the requested car is available. It returns the error "CarNotAvail", when the car is not available and the system is open, and the error "NoCarsAvail" when the system is closed.

ReturnCar updates *account* if the input *car?* is currently being rented. Otherwise it returns an error.

iii) Write the operation schema *AddNewCar*, which takes the input *newcar?* and updates the basic parameters of the system.

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