IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
EXAMINATIONS 2005

EEE/ISE PART II: MEng, BEng and ACGI

Corrected Copy

SOFTWARE ENGINEERING 2

Wednesday, 25 May 2:00 pm

Time allowed: 2:00 hours

There are FOUR questions on this paper.

Q1 is compulsory.
Answer Q1 and any two of questions 2-4.
Q1 carries 40% of the marks. Questions 2 to 4 carry equal marks (30% each).

Any special instructions for invigilators and information for candidates are on page 1.

Examiners responsible First Marker(s) : L.G. Madden, L.G. Madden
Second Marker(s) : J.V. Pitt, J.V. Pitt

This page is intentionally left blank

Q1 (a)

(i) What term is used to describe the combination of Encapsulation and Abstraction?

[2]

(ii) What term describes the property of a type as a set of operations?

[2]

(iii) What is Encapsulation and what does it mean when we say that C++ friend functions "break encapsulation"?

[4]

(b) Name any two UML diagrams, and state the corresponding kind of abstraction(s) shown by each of the diagrams.

[8]

(c) In Figure 1.1 how many members (i.e. data members and member functions) are in class `X` and how many in class `Y`?

[8]

(d) Figure 1.2 shows a class diagram containing four classes. By exploiting inheritance, draw a revised class diagram.

[4]

(e) Earlier this year the Huygens probe successfully landed on Saturn's moon Titan. Owing to a software error, only half of the acquired data was sent back to Earth. A design detail had been overlooked resulting in the omission of one line of code. Explain how Verification during Object Oriented Analysis and Design could be used in future missions to avoid this kind of inconsistency.

[4]

(f) In Figure 1.3, what are the final values of `x` and `y` after execution of the main program? Would the program compile if the first formal argument to the `obfuscate()` function was changed from `&` to `const &`? Explain your answer.

[8]

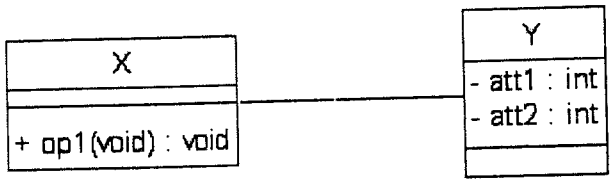


Figure 1.1

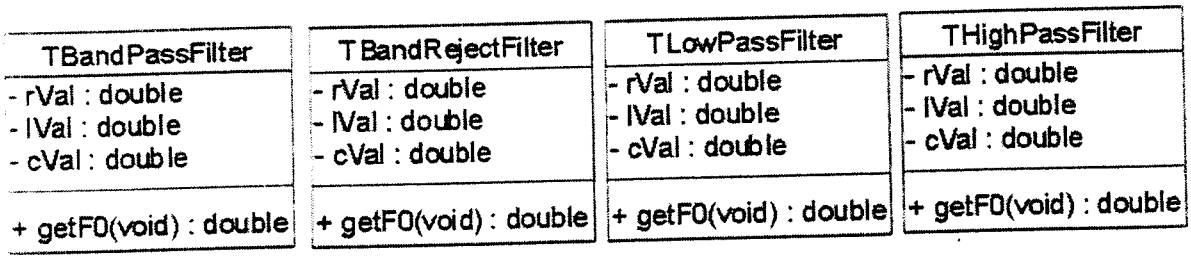


Figure 1.2

```
// int obfuscate (const int &, int);

int obfuscate (int &, int);
void main(void){
  int x, y;
  x = 0;
  y = 1;
  x = obfuscate(y, x);
  // x = ?, y = ?
}

int obfuscate(int &x, int y){
  x = x+1;
  y = y+1;
  return x+y;
}
```

Figure 1.3

2. (a) A C++ implementation of an Abstract Data Type is required to represent the equation of a line. The examples shown below are typical instances of the equation:

```
1y = 2x - 3
-1y = 4x + 3
3y = 2x
```

Write C++ code for the class definition that would appear in the C++ specification file. The class definition should contain only the Orthodox Canonical Class Format members. You do not need to include assessor methods or implementation code.

[9]

(b) Examine the C++ function `xyz()` shown in Figure 2.1

(i) What functionality is provided by the function? Briefly explain how the function works.

(ii) Write a C++ function `zyx()` which provides the inverse functionality i.e. `zyx()` converts the data from the function `xyz()` back into its original form.

[9]

(c) Explain why the C++ class definition for a stack Abstract Data Type in Figure 2.2 demonstrates:

(i) Reuse and

(ii) Adaptation.

[6]

(d) Examine the C++ code extract shown in Figure 2.3. Could this code be described as an example of Generic Programming? Explain your answer.

[6]

```
template <typename T>
  T xyz(const string &s) {
     istringstream iss(s);
     T x;
     iss >> x;
     return x;
}
```

Figure 2.1

```
template <typename T>
  class TStack{
  private:
     vector<T> aStack;
  public:
     void push(T i)   { push_back(i);  }
     T top(void)      { return back(); }
     void pop(void)   { pop_back(); }
}
```

Figure 2.2

```
struct TVolt{
     int      nodeName;
     double   nodeVoltage;
};

typedef struct TVolt TVoltage;

list<TVoltage> voltages;
list<TVoltage>::iterator it;

for(it=voltages.begin(); it != voltages.end(); ++it){
   cout << "Node "
        << (*it).nodeName
        << " has a value of "
        << (*it).nodeVoltage
        << "V";

}
```

Figure 2.3

3. (a) Perform a textual analysis of the text shown below. Identify suitable attributes for a `TrafficLight` class, then draw a class icon (without operations), but with attribute data-types (invented if needed).

> A traffic light changes state from red, to flashing amber, to green, to amber and back again to red. The duration of each state is variable.

[5]

(b) Draw a classification diagram based upon the description of EDIF (the electronic data interchange language) shown below:

> EDIF supports a number of "views" of electronic design data including:
> - NETLIST describes details of the circuit elements and connectivity
> - SCHEMATIC shows circuit symbols and interconnecting wires
> - MASKLAYOUT describes physical layout on the mask
> - PCBLAYOUT describes physical layout on the pcb
> - BEHAVIOUR describes circuit operation
> - LOGICMODEL describes digital logic behaviour
> - DOCUMENT is used for textual documentation
> - GRAPHIC is used for graphical documentation.

[9]

(c) Figure 3.1 shows five class diagrams labelled (a), (b), etc. For each of the following descriptions, state which is its equivalent diagram:

(i) All insects have 6 legs; a wasp is an insect that also happens to have wings.

(ii) A student has studied many subjects; a student also has a favourite subject.

(iii) Imperial College has many students; each is uniquely identified by a college identifier number.

(iv) In a queue, everyone has someone in front of them, except the first person.

(v) Many people visit a cinema; a person might visit many cinemas. For each visit certain details must be recorded e.g. the film being shown, the date and time of performance, seat details.

[7]

(d) Draw a UML statechart for an RS or JK flip-flop circuit. State clearly the type of circuit you have drawn and whether it is a Mealy or a Moore model.

[9]

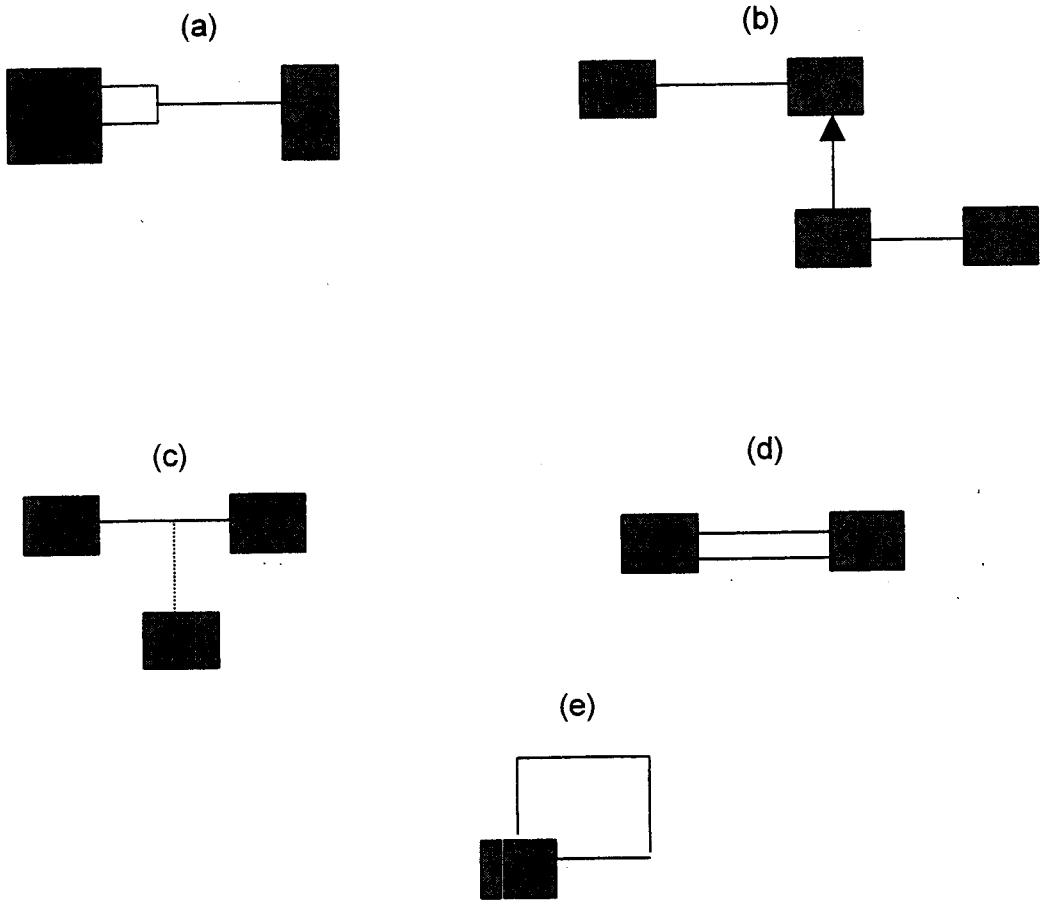


Figure 3.1

Q4. This question is based on the following Requirements Specification:

IP addresses are currently expressed in a dotted decimal notation e.g. 155.198.123.115, and are divided into 5 separate classes:

Class A addresses that range from 1.0.0.0 to 127.255.255.255
Class B addresses that range from 128.0.0.0 to 191.255.255.255
Class C addresses that range from 192.0.0.0 to 223.255.255.255
Class D addresses that range from 224.0.0.0 to 239.255.255.255
Class E addresses that range from 240.0.0.0 to 255.255.255.255

A program is required that will input 4 whole numbers representing a dotted decimal IP address. The purpose of the program is to output a message indicating that the address is class A or B or C or D or E. A reusable component is available to convert between strings and numbers. Additionally, the program should check for typical input errors e.g.

- any of the four numbers is not in range e.g. negative
- one or more values is a non-numeric value
- one or more values is missing at the moment of evaluation.

(a) Produce a Use case for the Requirements Specification above, including all four of the following:

(i) Use case diagram
(ii) formal textual description
(iii) success scenario and
(iv) failure scenario.

[9]

(b) Starting from the Object Model shown in Figure 4.1, a designer has produced the Sequence Interaction diagram that is shown in Figure 4.2,

Make a list of observations and deductions from the details inherent in Figure 4.2 that are not apparent in Figure 4.1.

[9]

(c) Convert the Sequence Interaction diagram in Figure 4.2 into a Collaboration Interaction diagram.

[6]

(d) Describe the meaning of "Design by Contract"? How was it used in this example.

[6]

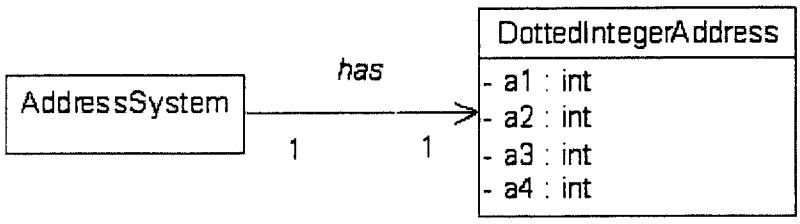


Figure 4.1

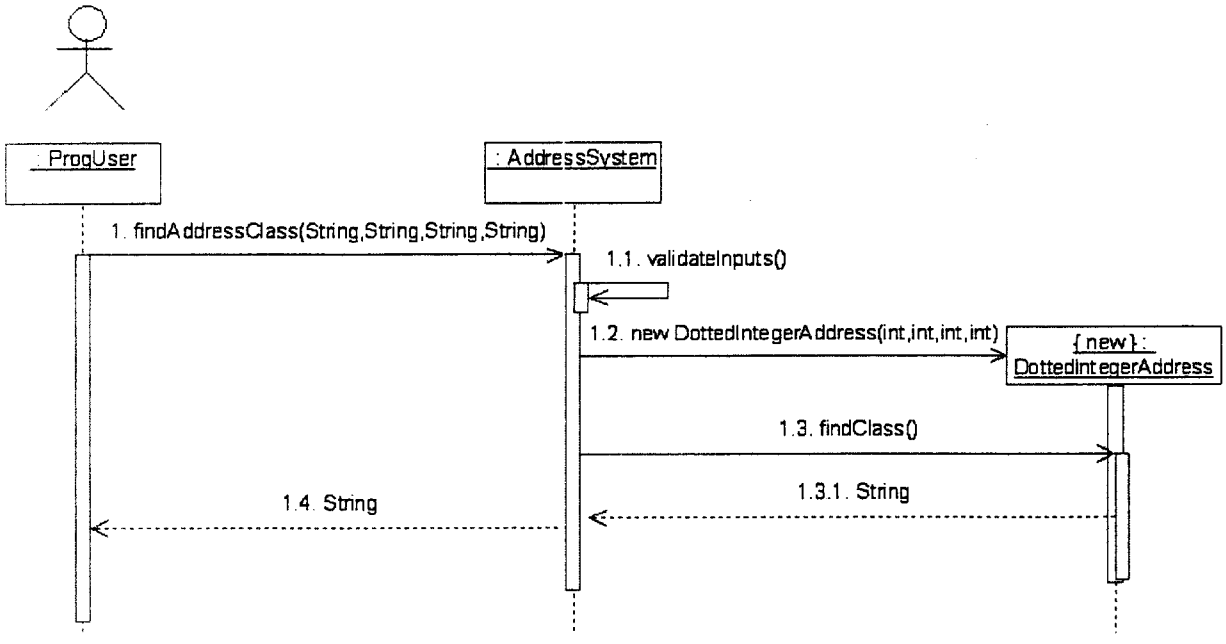


Figure 4.2

A1(a) "bookwork" [8]

i) Information Hiding [2]
ii) Abstract Data Type [2]
iii) Encapsulation - client is *unable to know* any more than what is defined in the interface. OO languages provide visibility control to enforce encapsulation syntactically. However, C++ knowingly allows relaxation of encapsulation for functions declared to be "friends" in the class definition. This is often allowed for I/O operators to access the internal (private) state of the object. [4]

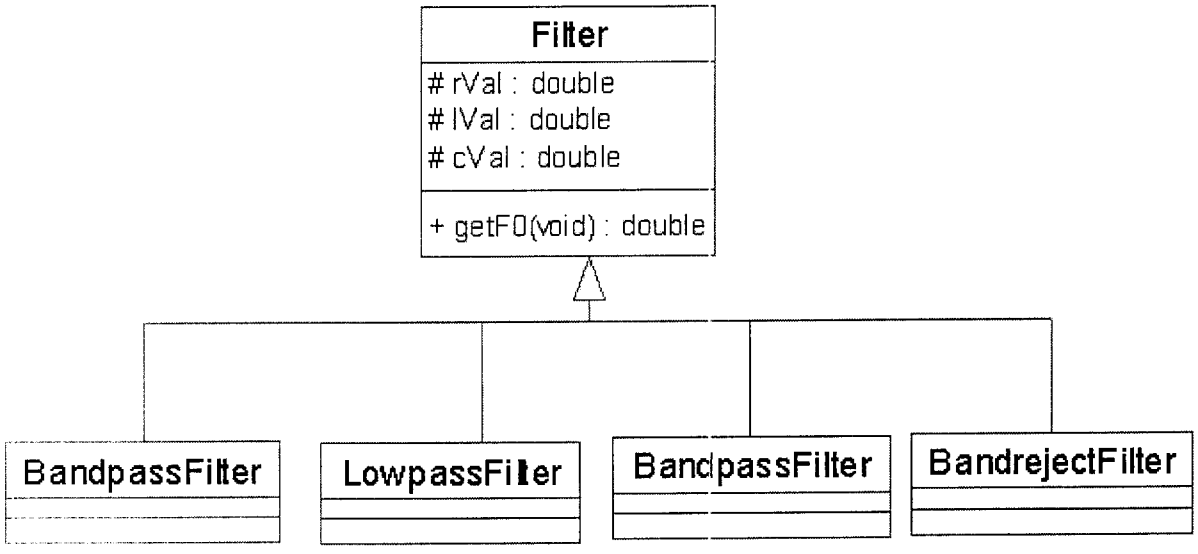
A1(b) "bookwork" [8]

Use case	**Functional decomposition**
Class diagram	**Static/Structure** e.g. Association (Composition), Generalisation/Specialisation (Inheritance)
Component, Deployment	**Static/Structure**
Activity (flow of control from activity to activity, transitions occur on completion of activity), **Statechart** (transitions are event driven), **Sequence/Collaboration Interaction diagrams** (flow of control from object to object)	**Dynamic/Behavour**:
Use-case (customer view), **Class** (developer view)	**Multiple views**
Use-case (functionality), **Class** (structure) **Interaction** (Behaviour)	**Multiple views**
Sequence (emphasises structure)/**Collaboration Interaction diagrams** (emphasises behavior)	**Multiple views**

A1(c) "new computed example" [8]

Class X has 1 member function and 1 association i.e. **2** [4]
Class Y has 2 data members and 1 association i.e. **3** [4]

A1(d) "new computed example" [4]



- `Filter` may be abstract or concrete.
- `Filter` data members must have protected visibility
- if subclasses override `getF0()` then `Filter::getF0()` must be `<<virtual>>`
- if `Filter::getF0()` is `<<virtual>>` then overriding in subclasses is optional

A1(e) "bookwork" [4]

The program development cycle involves well-defined activities which have an input work product and an output work product. Verification involves checking the output against input for consistency. In this example the input was the design document and the output was the source code. Verification should have revealed the omitted element from the design.

A1(f) "new computed example" [8]

```
int obfuscate (int &, int);

void main(void){
   int x, y;
   x = 0;
   y = 1;
   // x = 0, y = 1
   x = obfuscate(y, x);
   // x= ?, y = ?
   // x = 3 (i.e. obfuscate (), y = 2 (i.e. (obfuscate::x)


int obfuscate(int &x, int y){
// x = 1, y = 0
   x = x+1;
// x = 2, y = 0
   y = y+1;
// SBR x = 2, SBV y = 1
   return x+y;
// obfuscate = 3
```

Changing 1st formal argument to a `const` produces a compilation error because the function attempts to change the value of constant i.e. x = x +1

A2(a) "new computed example" [9]

```
class Line{                                          // TLine.h
public:
  Line(double, double, double);                      // OCCF
  Line (const Line &);
  ~Line();
  Line operator= (const Line &);
private:
  double y, gradient, intercept;
};
```

shouldn't have any:

- executable statements (including stubs) and/or initialisers
- SRO preamble e.g. `Line::Line()` since specification is in .h not .cpp file

A2(b) "new computed example" [9]

```
      from string to a primitive (template) data type T

template<typename T> T xyz(const string &s) {
  istringstream iss(s);//convert string to input stream buffer
  T x;       // declare (uninitalised) variable of primitive data type
  iss >> x; // using istringstream built-in conversion
            // initalise variable of primitive data type
  return x; // return primitive data type


// from a primitive (template) data type T to string
template<typename T> string zyx(const T &x) {
  ostringstream oss; // uninitialised output stream buffer
  oss << x;          // insert primitive data type onto buffer
  return oss.str();  // return C++ string after conversion from buffer
}
```

A2(c) "bookwork" **[6]**

Reuse comes from the use of a C++ standard library aggregate class called `vector`, which is a variable sized array. Reuse also comes from Tstack being a "classical" data structure.

Adaptation comes from the use of template type for the data member elements of the `vector` (and `Tstack`). Thus an instance of a `vector` aggregate is capable of containing diverse data member elements, including user created classes. Note: there may be a requirement to overload operators and functions for the user created class.

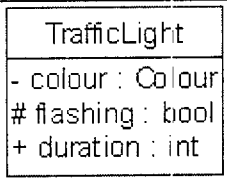
A2(d) " bookwork" **[6]**

Uses the C++ standard library aggregate class called list, which is a linked list. Because the list class has been created with a polymorphic protocol shared by the other aggregate classes (e.g. map, vector) it is possible to send the same message to any of the aggregate classes. The use of an iterator class (a generalisation of a pointer and related to the aggregate class) makes it possible to process an aggregate and its underlying data using polymorphic messages thus not without revealing any structural information. [4]

Thus this is an example of "Generic Programming". [2]

Note: Information hiding could be improved by using a class instead of a structure for the elements of the aggregate

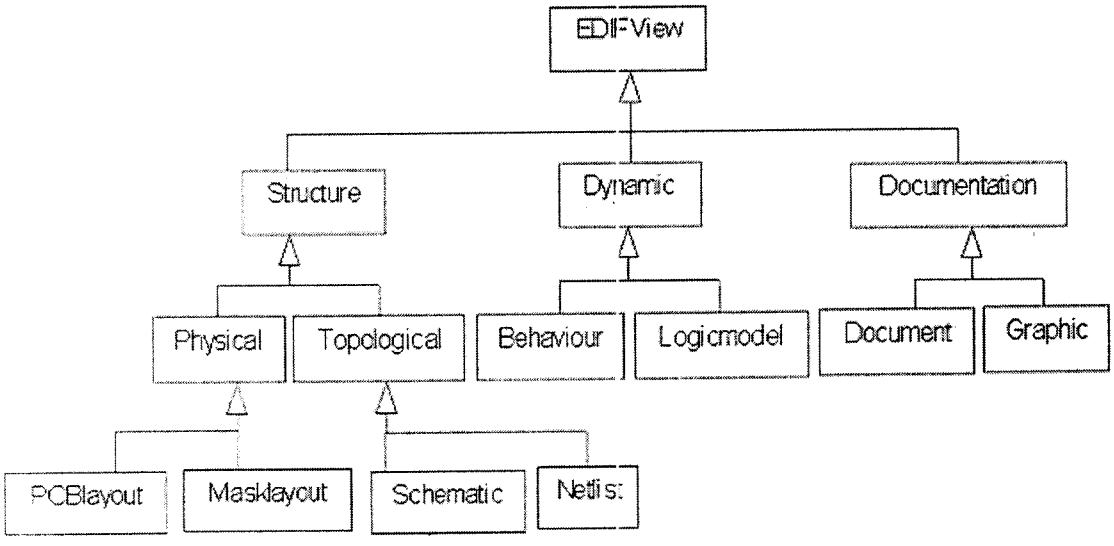
A3(a) "new computed example" [5]



data members [2], type for each data member [2]

Note: any variation acceptable (e.g. attribute/colour, attribute called `state` - with enumerated values) as long as it is apparent how each of 3 data values are provided

A3(b) "new computed example" [9]



must have abstract classes (as in tutorial example)

A3(c) "new computed example" **[7]**

(i)(b) All insects have 6 legs; a wasp is an insect that also happens to have wings. [2]

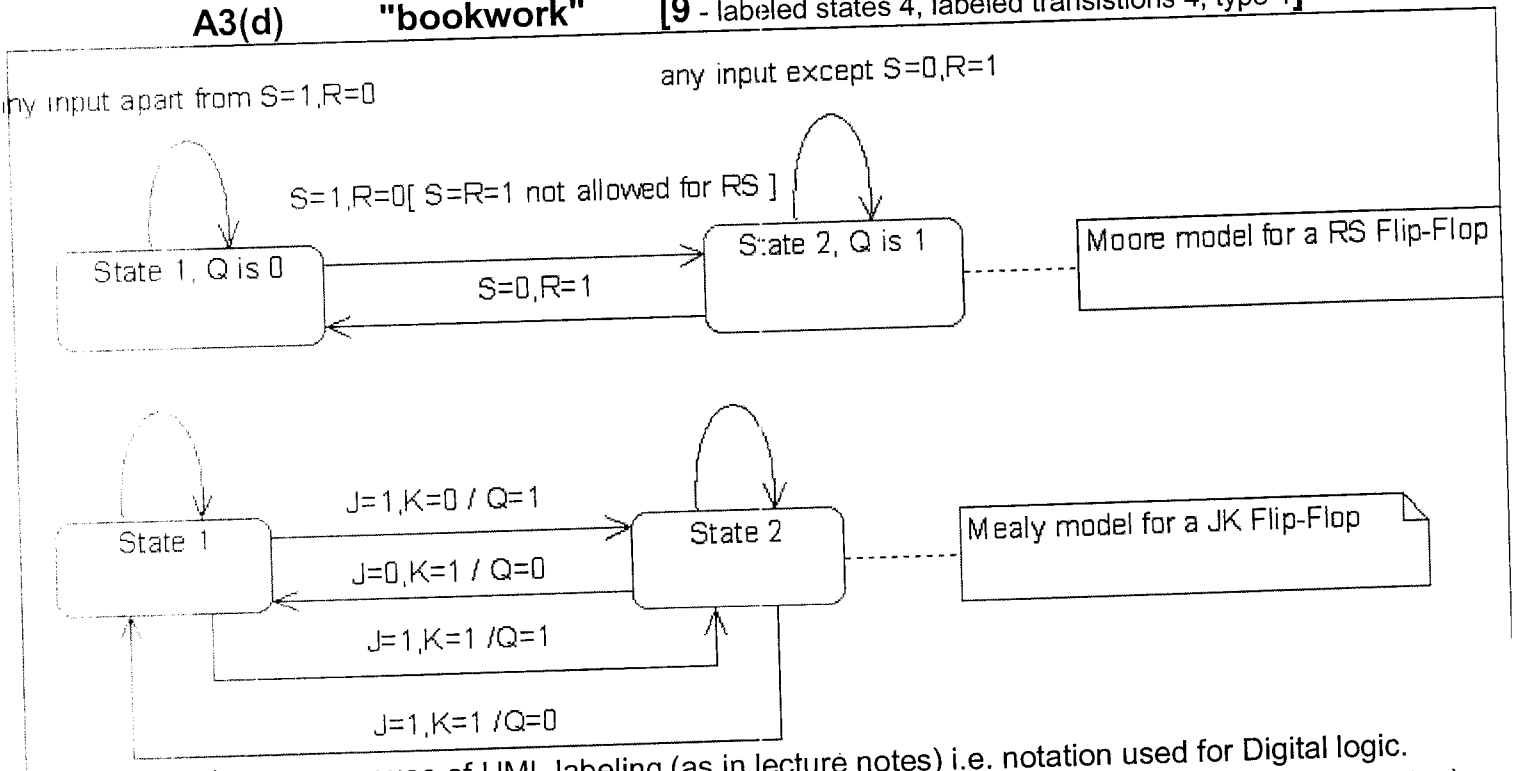
(ii)(d) A student has studied many subjects; A student might also have a favourite subject. [1]

(iii) (a) Imperial has many students who are uniquely identified by their college identifier. [1]

(iv)(e) In a queue, everyone except the first person has someone in front of them. [1]

(v)(c) Many people visit a cinema; a person might visit many cinemas.For each visit certain details must be recorded e.g. the film being shown, the date and time of performance, seat details. [2]

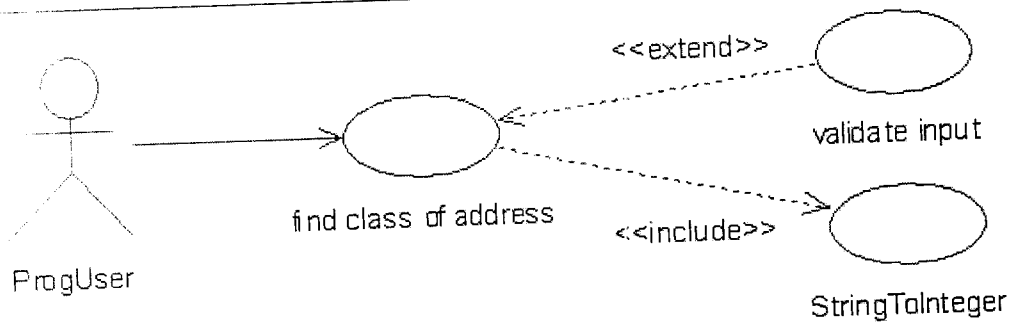
A3(d) "bookwork" **[9** - labeled states 4, labeled transistions 4, type 1**]**



- must have proper use of UML labeling (as in lecture notes) i.e. notation used for Digital logic.
- complete correctness for circuit not required, just correct number of state (i.e. binary == 2 states) and suitable number of transitions (including self)
- initial/final states are optional but if provide enter and exit both states

Moore state machine outputs are completely dependent on the current state of the circuit. This means that the outputs only change with state transition. Meely state machine outputs depend on the current state and the inputs.This means that the outputs can change when there is no state transition.

A4(a) "bookwork/new computed example" [9]



Actor **1**, UC **1**, Reuse **1**, Extension **1**

Identifier and name	UC#1	1
Initiator	ProgUser	
Goal	Determine the class of the I/P address	
Pre-condition	none of the four values is not in range e.g. negative none of the four values is a non-numeric value none of the four values is missing at the moment of evaluation.	1
Post-condition	Returns that class is A or B or C or D or E	1

Actor	**Success scenario #1** 1
1. ProgUser enters 4 numbers	
	2. System validates 4 numbers 3. System returns I/P class type
Actor	**Exception scenario #1** 1
1. ProgUser enters 4 numbers	
	2. System validates 4 numbers and fails 3. System returns error message

- Activity or Interaction diagrams acceptable for scenarios

A4(b) "new computed example" [9]

2 marks/observation, last mark for 5th observation.

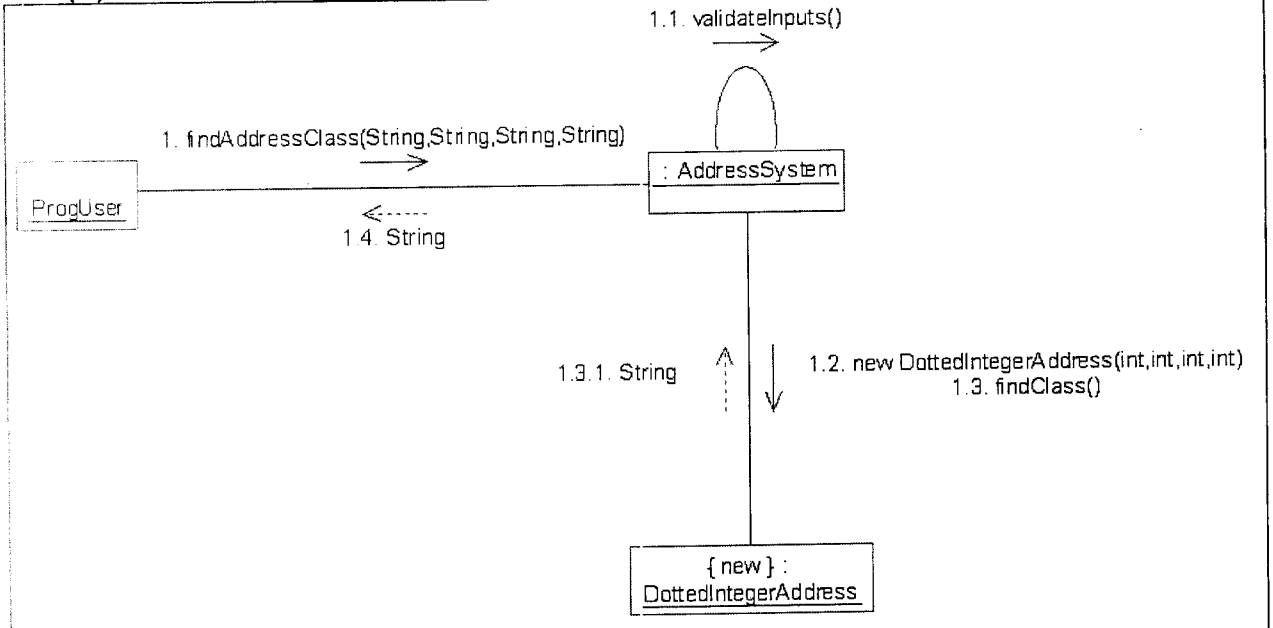
As a result of drawing of the interaction diagram it is possible to make some observations, which may be new, affirm or contradict the current class diagram detail e.g.

- `AddressSystem` must include `findAddressClass(String,String,String,String):String` in its protocol
- `AddressSystem` must include `validateInputs(void):boolean` in its protocol
- `DottedInternetAddress` must include `findClass(void):String` in its protocol
- `DottedInternetAddress` must include a 4-argument constructor `DottedInternetAddress (int,int,int,int)`
- an association between the classes `AddressSystem` and `DottedInternetAddress` exists because the former needs a reference to the later in order to send the message `findClass(void):String` In fact `AddressSystem` is responsible for the creation of the `DottedInternetAddress` instance
- the algorithm for `findAddressClass(String,String,String,String):String` will include:

```
try{
   if (validateinputs(void)){
     anAddress = new DottedInternetAddress (a,b,c);
     cout << anAddress.findClass();
   }
   else
     throw InvalidInputException;
} catch (InvalidInputException err) {
     // do something for invalid pre-condition
}
```

- new class `ProgUser` (orchestrating instance)
- association between `ProgUser` and `AddressSystem`

A4(c) "bookwork/new computed example " [6]



must have:

- sequence numbering
- function calls with formal arguments and return values
- indication of {new} object

A4(d) "bookwork/new computed example " [6]

In OO a contract between 2 objects exists when a client object sends a message to a supplier/server object. The contract is produced by the:

• The Pre-condition which describes what must be true *before* the message-send has been carried out and requires something of the client that benefits the server e.g. the input values are all valid.

• The Post-condition which describes what must be true *after before* the message-send has been carried out and requires something of the server that benefits the client e.g. the I/P class is…

The process of developing software using contracts between objects is known as Design By Contract (DbC) and delivers:

- *correctness* by satisfying contracts and
- *quality* by providing traceability from requirements specification to code

For this example DbC is evident in the:

- requirements specification
- pre/post-conditions of Use case
- sequence/collaboration interaction diagrams i.e. `validateInputs()` is precondition, message answer from `findClass()` is post-condition.
- C++ code e.g. try-catch block