THE BCS PROFESSIONAL EXAMINATION
Diploma

April 2002

EXAMINERS' REPORT

Object Oriented Programming

Question 1

1. Operations and attributes of a class are given the following *visibility scopes*: public, protected and private.

a) **Explain the meaning of these three scoping terms. (9 marks)**

b) **Describe how each visibility scope would be used for both an attribute and an operation. Identify and explain where good programming practice is being adopted. (12 marks)**

c) **Identify a fourth level of scoping that might prove useful. What are the merits and the deficiencies of this scheme? (4 Marks)**

Answer Pointers

(a/b) The answer needs to highlight the distinction between these levels of visibility and how they are deployed by a software developer. Generally this revolves around the principles of encapsulation and information hiding. Hence we might start from a premise of public methods for the class specification and private attributes for the representation. Equally, we should consider the role for public attributes and private methods. A clear distinction for the protected interface is required.

The public features of a class are visible to all other classes. A public method of one class can be called by a method in another class. Equally, a public attribute can be referenced and modified from elsewhere. This is generally not wise since it exposes the implementation of a class to others. Constant public attributes are generally considered safe.

The private features of a class can only be referenced by that class. For example, a private attribute can only be referred to in the body of any of the class methods. Generally, privacy of attributes is used to secure the implementation of a class. A private method is used in a support role for the other class methods.

The protected features of a class are private to other classes and public to subclasses. This way a subclass can directly reference the features of its immediate super class.

We might consider a scoping whereby private features are shared between two or more classes. Java refers to this as package visibility and C++ as friend classes. Both break the rules of encapsulation and, perhaps, are best avoided for that reason. However, in a controlled environment there may be a case for deploying it.

Examiners' Guidance Notes

Many answers failed to properly explore the software engineering issues associated with visibility. For many candidates the answer to part (b) was to repeat the answer to part (a). Here, there was no evaluation of the role and purpose for the visibility levels. Many answers did not describe why we generally make attributes private. Nor did the candidates consider how we might safely place an attribute in the public interface. Part (c) was not well answered by candidates, not recognising the fourth visibility level offered by C++ friends and Java packages.

Question 2

For each of the following concepts:

i) **constructor**
ii) **method overloading**
iii) **method overriding**
iv) **polymorphism**

a) **Provide a definition of the concept; (12 marks)**

b) **Provide code which demonstrates the use of the concept written in an object oriented language with which you are familiar. (13 marks)**

Answer Pointers:

A constructor is a special method within a class that is executed when an object of that class is instantiated. Its normal use is to initialise the instance variables of the newly created object.

```
Class MyClass{
        int x;

        MyClass(int y){
                x=y;
        }
}

public static void main(String args){
        MyClass myObject = new MyClass(1);
}
```

Code invokes constructor to create and object with instance variable set to 1.

Method overloading is a mechanism whereby within a class definition it is possible to have methods with the same names but with a different number of arguments or arguments of different types. The compiler selects the version of the method that matches the arguments supplied.

```
Class MyClass{

        int value=0;

        int increment(){
                value++;
        }

        int increment(int x){
                value=value+x;
        }
}
```

Here increment() can be used in one form to increment by one or if supplied a value will increment by that value.

In method overriding a subclass implements a method with the same name and signature as one in the superclass. The method effectively replaces the superclass method when invoked on an instance of the subclass.

```
Class MyClass{
        void printClassName(){
                System.out.println(“MyClass”);
        }
}

Class MySubclass extends MyClass{
        void printClassName(){
                System.out.println(“MySubclass”);
        }
}
```

Polymorphism is the ability to take many forms. In programming it means the ability to refers to object that belong to different classes in the same way.

```
Interface MyInterface{
        void display();
}

class MyFirstClass implements MyInterface{
        void display(){
                .
                .
        }
}

class MySecondClass implements MyInterface{
        void display(){
                .
                .
        }
}

public static void main(String args[]){
        MyInterface[] myArray = new MyInterface[2];
        myArray[0]=new MyFirstClass();
        myArray[1]=new MySecondClass();
        myArray[0].display();
        myArray[1].display();
}
```

Polymorphism makes it possible to have an array of objects of different types and apply the same method to each object.

Examiners' Guidance Notes

Candidates frequently confused overloading and overriding. Interestingly a number did not seem to know what a constructor was and left this part blank.

The standard of the code produced was often poor, again mixing up overloading and overriding. In particular, candidates were unable to demonstrate polymorphism in a code fragment. Many appeared to think this only worked in the inheritance hierarchy - very few appreciated that classes that are not part of the same hierarchy can have methods with identical names. Only one candidate gave a Java interface as an example. Most used C++ and Java as the language, one used Visual Basic but with little success.

Question 3

a) **Explain the following terms:**

i) **abstract class**

ii) **single inheritance**

iii) **multiple inheritance** **(9 marks)**

b) **State with an appropriate example what is meant by the term *abstraction*.** **(8 marks)**

c) **Discuss the importance of abstraction in making software reuse possible.** **(8 marks)**

Answer pointers

a) An abstract class is one which represents some but not all of the characteristics of a real-world class. It may not be instantiated. Typically it is used to hold the common properties of its concrete subclasses.

In a language with single inheritance a subclass may only inherit from a single superclass. A subclass inherits the methods and data of its superclass. A superclass may have a number of subclasses.

In multiple inheritance a subclass may inherit from a number of superclasses. As with single inheritance the subclass inherits all the methods and data of all its superclasses. There may be a problem if superclasses have identically names methods and instance variables.

b) Abstraction is the process of identifying the essential issues of a problem from the non-essential. In the context of OO programming it involves identifying the essential characteristics of an object and separates these issues from the way those characteristics are implemented. Consequently objects are identified by their behaviour and not the way in which their data is stored.

One example of abstraction is a Stack class. For this class we can identify a set of essential operations that would include Push and Pop. The issue of how the stack and the elements on the stack are actually stored is not important to the user of the stack.

c) The abstraction process allows programmers to write code that expresses the essential nature of an object. A programmer who is conscious of the storage requirements of the object on a stack often produces code which is only suitable for objects of that type. Consequently the code that is written for a stack of integers is often not easily modified to produce a stack of GraphicElements. A programmer using abstraction produces code that implements the characteristics of a stack and is suitable for objects of any class. This makes it possible to reuse the code written for the stack in many different contexts.

Examiners' Guidance Notes

On the whole candidates were able to provide good answers to part a), a number achieved full marks. A few candidates thought that single and multiple inheritance determined the number of subclasses a class could have, not the allowed number of superclasses.

Candidates were much less able in answering parts b) and c). Many chose not to answer these parts at all. There was confusion between abstraction and abstract classes. Others confused inheritance with reuse. A few candidates explained reuse but did not relate it to abstraction.

Question 4
The drawing in Figure 1 below is representative of the output which might be constructed while using an interactive drawing tool. In the drawing there are a number of rectangular shapes that are connected by lines. The sample diagram shows rectangles, plain lines and lines decorated with arrowheads. It might be envisaged that the user selects the required shape to draw from a palette then places each into the drawing.

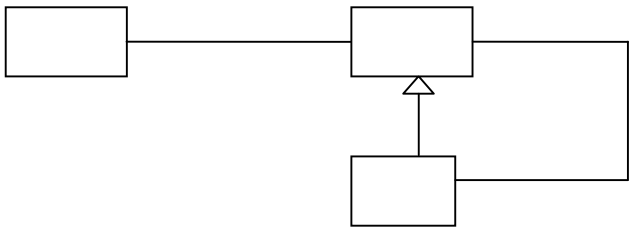


Figure 1

a) **Prepare a class diagram for this drawing tool to compose and edit typical diagrams as shown above. In support of this class diagram you must provide an analysis of the objects in the problem and the relationships that exist between them. (8 marks)**

b) **For each class you should propose typical features (attributes and operations) that you would expect of such classes, explaining their purpose. (7 marks)**

c) **Figure 2 below, presents a possible flowchart that might be constructed with the drawing tool. In it there is a decision box (diamond shape) in addition to the original shapes [previously, there were only rectangles].**

Construct a class diagram for this revised drawing tool. (10 marks)

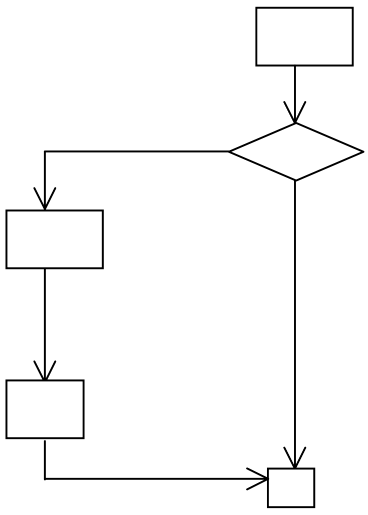
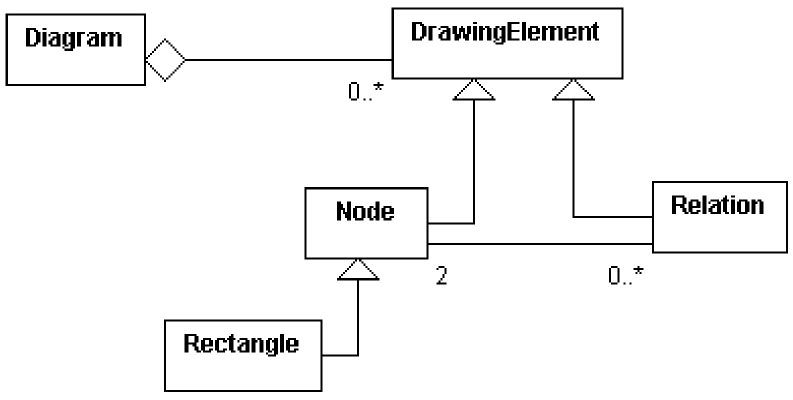


Figure 2

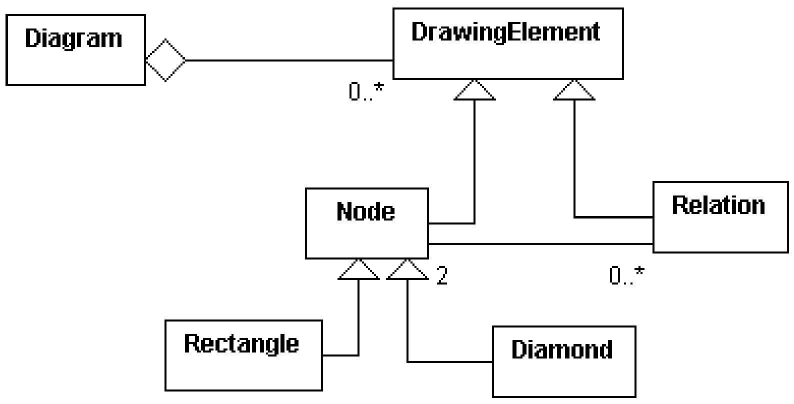
Answer Pointers

(a) The class diagram we should arrive at is given below. A diagram consists of any number of drawing elements. In turn, the latter can be either a node or a relationship. A node represents any vertex within the graph and a relation is any connection between nodes. A specific kind of node is, of course, a rectangle symbol.



(b) Nodes will have geometric properties such as their location on the diagram. Additionally, we might expect nodes to have names. relation might include an indication of the type of decoration it has, eg arrowhead. All drawing elements should be capable of being dragged and edited.

(c) The original class diagram is essentially unchanged. Specifically all we require is to introduce a new class for a diamond shaped decision box. The new figure is:



Examiners' Guidance Notes

In this and in question 5 candidates need to be fully conversant with specialisation and be prepared to assemble solutions to problems using it. Many respondents did not use specialisation in formulating an answer, using only aggregation and association instead. This produced very difficult solutions. For those that tried specialisation then, surprisingly, we had inappropriate isA relationships, for example the subclass Rectangle as a specialisation of the superclass Line. Because of this, part (c) was poorly answered where, in fact, it is a surprisingly easy extension to the architecture.

Question 5
The class diagram in Figure 3 below presents a class hierarchy in which Aaaa is a superclass to both Bbbb and Cccc. The superclass Aaaa has an implementation for some operation mmm.

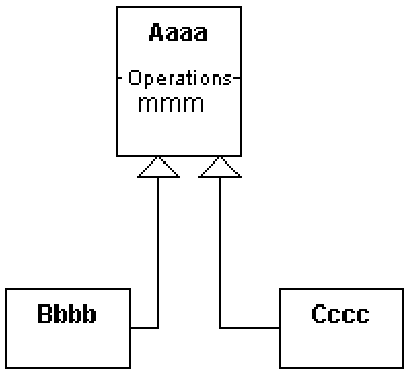


Figure 3

a) **Present a general scheme whereby the implementation of the operation mmm can include some specific behaviours from the subclasses Bbbb and Cccc.** **(9 marks)**

b) **If some object of the class Bbbb receives the message mmm, provide a detailed explanation of the execution flow of this message and any subsequent messages and the recipients of all these messages. What would be the difference, if any, if the receiver of the message mmm were of the class Aaaa?** **(7 marks)**

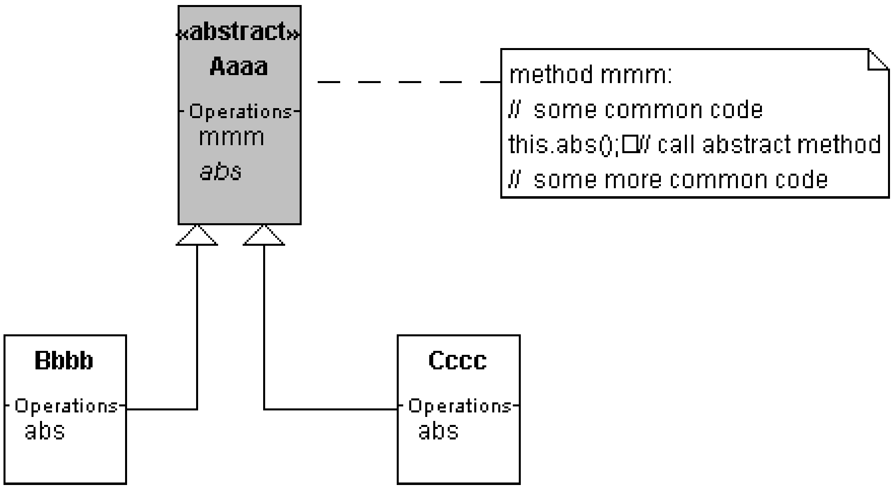
Consider a set of data values, such as that found in a spreadsheet, which are to be presented in a number of views. The spreadsheet data is to be presented: *i)* in the conventional manner of a two-dimensional table; *ii)* as a bar chart; and *iii)* as a pie chart. When the data set is changed, then all of the views must be updated to reflect the revisions.

c) **Suggest a suitable class diagram for this scenario, identifying the motivation for its use, and demonstrating the applicability of its structure for this situation. Specifically, you should seek to ensure that the views are treated uniformly.** **(9 marks)**

Answer Pointers

The question and its answers are motivated by design patterns, but these need not be offered as the solution. What the question seeks is a valid solution to this type of problem that occurs in OO systems.

(a) The solution is achieved through the *template method* design pattern. The superclass Aaaa becomes an abstract class with the introduction of a deferred method abs. The method body for mmm includes a call to this method. In the concrete subclasses Bbbb and Cccc each redefines the abs method with their own specific behaviour.



(b) If some object of the class Bbbb receives the message mmm, then it executes the method defined in the superclass since it has no redefinition for this method. Method mmm then sends the message abs to itself, the recipient, of course, being of the class Bbbb. Since the class Bbbb has a definition for this method, then mmm in class Bbbb is executed.

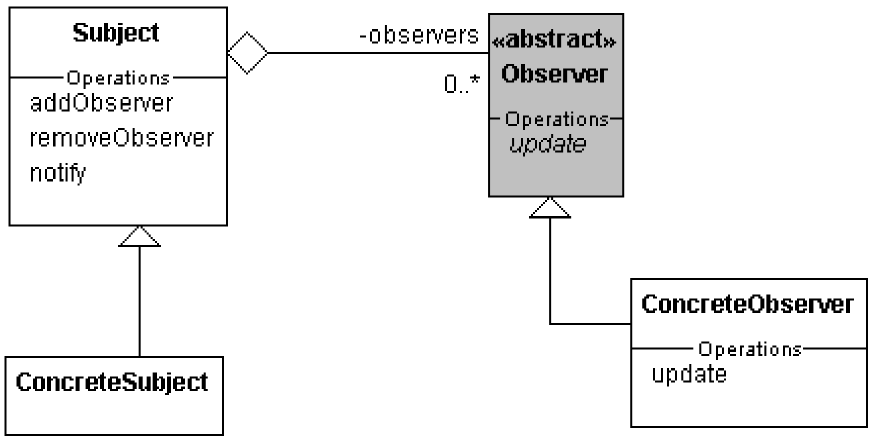
A similar story applies to an object of the class Aaaa receiving the message mmm. This time, the specialised behaviour of abs in the class Aaaa is executed.

(c) The *observer* design pattern is used when a one-to-many dependency exists between objects so that when one object changes state, all its dependents require to be notified. The observer pattern describes how to establish these relationships. The key objects in this problem are the subject and observer. A subject may have any number of dependent observers. All observers are notified when the subject undergoes a state change.

The class diagram for the observer pattern is shown below. A Subject has a one-to-many relationship with objects of the class Observer. Observer dependents can be associated with a Subject using the method addObserver. When a state change occurs in a Subject it invokes its notify method coded as follows:

```
FOREACH obs IN observers DO
   obs.update()
ENDFOREACH
```

The implementation for notify, sends the message update to every Observer object associated with the Subject.



Examiners' Guidance Notes

In this and in question 4 candidates need to be fully conversant with specialisation and be prepared to assemble solutions to problems using it. They need to be aware not only that a subclass inherits all the features of its superclass, but also how method execution 'bounces' around the class hierarchy. Further, we need to appreciate the use of abstract (deferred) methods in a superclass and its redefinition in subclasses.

Part (c) requires a model in which changes in state to one object are notified to other interested objects. The sophisticated solution deploys the observer design pattern but others are possible.

Question 6
A college operates courses in the following way. A college administrator is responsible for introducing courses and removing courses from the college's offering. Students may enrol for courses that are on offer. To enrol, a student must consult with an administrator. The administrator will check that the student is eligible for the course. Students can only enrol if the course exists, if they possess the correct educational background and if they can pay the fees for the course. Students may withdraw from a course by visiting an administrator. Once enrolled, students attend sessions that are delivered by a tutor. The tutor sets examinations. Students attend examinations. After the examinations the student receives feedback on their performance from the tutor.

a) **Draw a use case diagram for this system.**

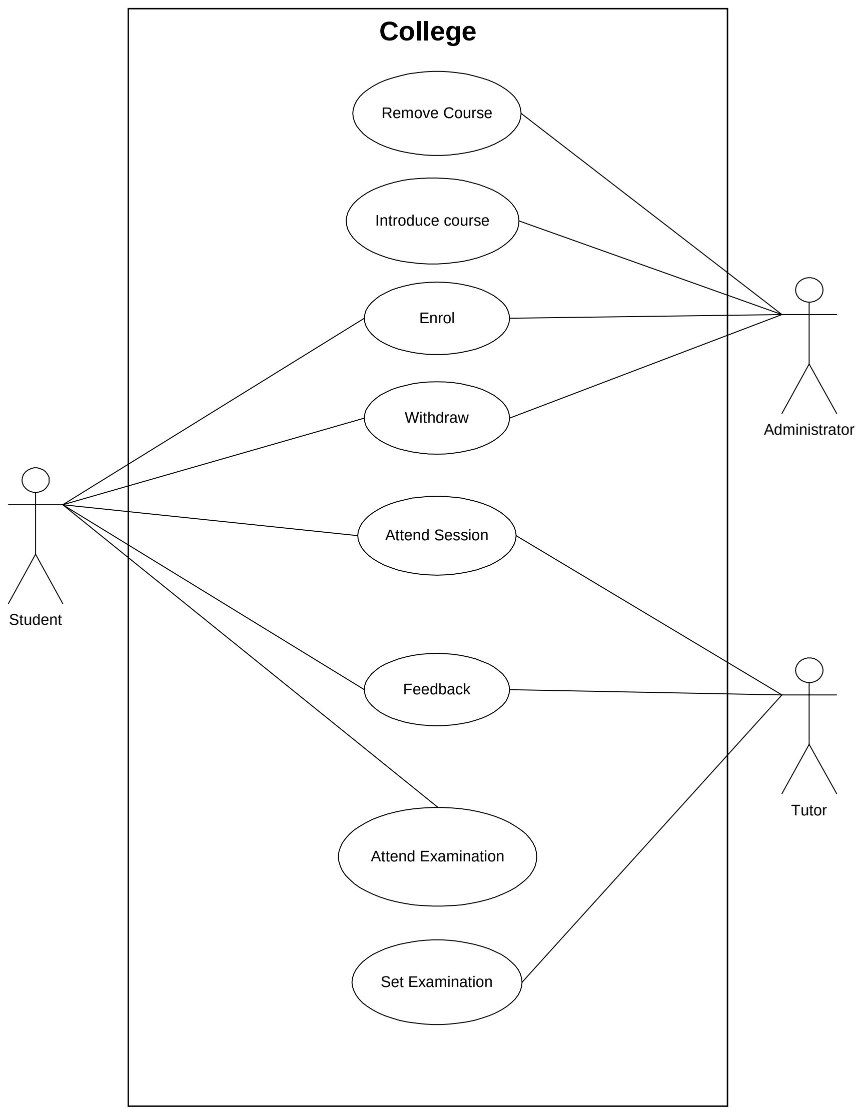
(15 marks)

b) **Develop a use case description of the way a student enrols. Your answer should include a normal sequence and an alternate sequence.**

(10 marks)

Question 6
Answer pointers

a)



b) Normal sequence

The student visits an a administrator;
The administrator checks that the course requested by the student is run by the college;
The administrator checks the student’s educational qualifications;
The adminstrator checks that the student can pay for the course;
The student is enrolled.

Alternate sequence

The student visits an a administrator;
The administrator checks that the course requested by the student is run by the college;
The course is not available so the student is not enrolled.

Examiners’ Guidance Notes

This question was very popular with candidates and was answered well. A question on use case diagrams has appeared on all Object Oriented Programming papers since the introduction of the syllabus. In earlier years answers to the question were quite poor but the quality of answers has improved year on year. Future papers may examine more advanced features of use case diagrams. As in last year’s examination the main failing in answers to part a) was a lack of appreciation that a use case could involve more than one actor.

In part b) candidates often supplied alternatives (via if clauses) in a single sequence. There was a tendency to invent details which did not form part of the supplied case study.