The BCS Professional Examination
Diploma

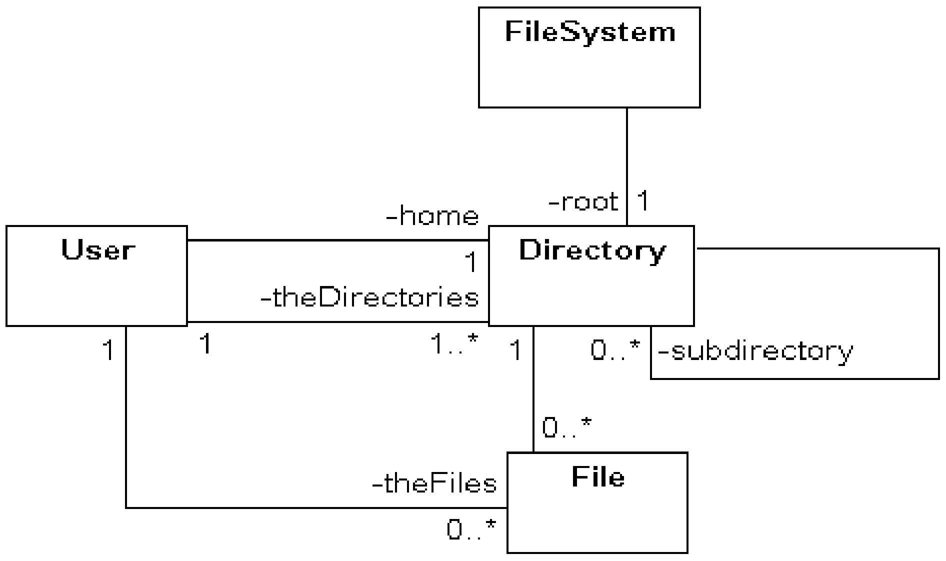
April 2001

EXAMINERS' REPORT

Object Oriented Programming

QUESTION ONE

The following figure models some aspects of a file system using a class diagram, where directories contain subdirectories and files. A file system consists of a set of files below a root directory and users can own directories and files, read files and have a home directory.



a) **Present a diagram showing an example file system, a user object corresponding to your account, your home directory, a subdirectory called "mail", a file called "autoexec.bat" in your home directory, and a file called "message" in the "mail" directory. (5 marks)**

b) **The specification of the file system can be made more contemporary with files and nested directories if a new class "Node" is introduced. This is a superclass to both "File" and "Directory". Redraw the class diagram using this new class to reduce the number of relations in the original model. (9 marks)**

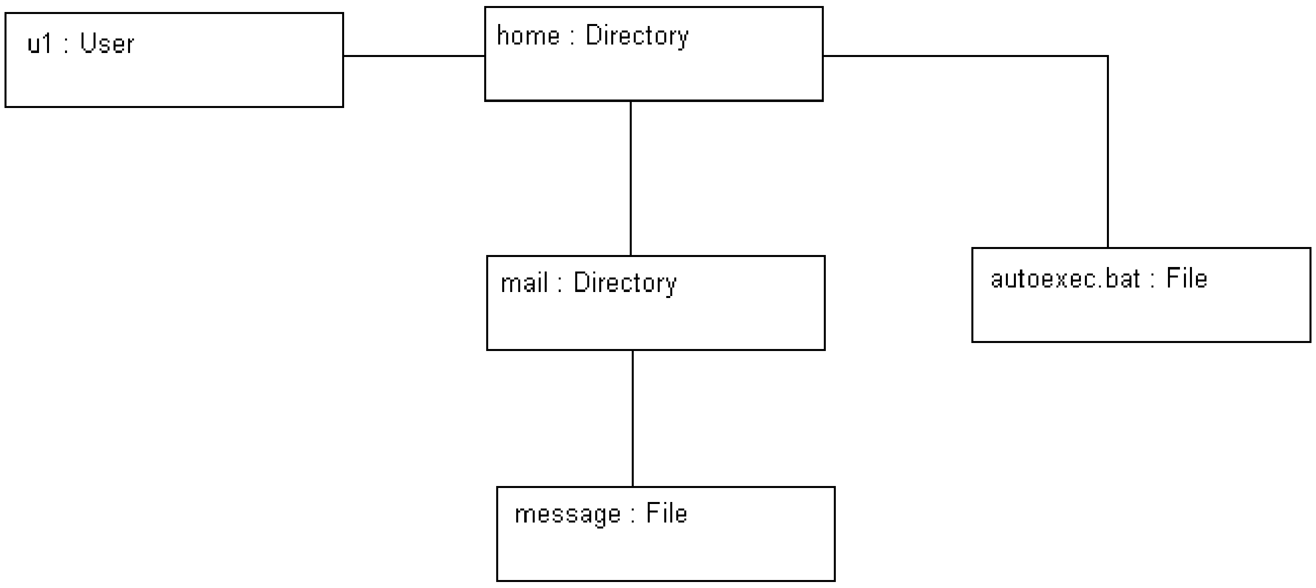
c) **Does the introduction of this new "Node" class have any effect on the diagram you drew for Part a)? If so, then elaborate. (4 marks)**

d) **The introduction of the "Node" class in part c) recasts the diagram into a standard design pattern. What is this pattern and what are its characteristics? (7 marks)**

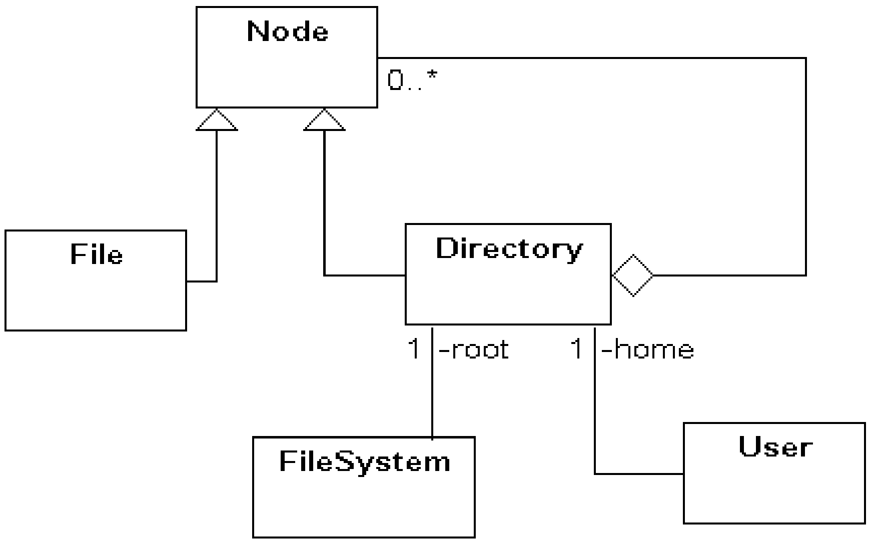
This question was generally well answered, although a sizeable number were unable to recast the model and show the proper use of specialisation. Students need to be able to interpret the information content of these diagrams, and to identify opportunities to simplify a design. Frequently this involves introducing a measure of specialisation. Again (see question 2), the students did not make the immediate connection to the design pattern in part d).

Answer Pointers

a)



b)



c) No. The reason for this is that "File" and "Directory" are concrete classes representing the actual objects in the problem space. Class "Node" is an abstraction for each.

d) Here we have an example of the composite design pattern. This has the advantage of treating all kinds of node (here, file and directory) uniformly. It is for this reason that we greatly reduce the number of associations in the model, hence simplifying it.

QUESTION TWO

a) **Explain what is meant by the term "pattern" in the context of object oriented development. (5 marks)**

b) **Describe any FOUR of the following patterns, detailing the nature of the problem they address and the basis of the solution they offer. Present class diagrams for the patterns and give suitable examples for their deployment.**

i) Singleton
ii) Observer
iii) Composite
iv) Decorator
v) Abstract factory
vi) Visitor (20 marks)

This question was shunned by the majority of students. It is a legitimate element of the syllabus and students need to have knowledge of this very important aspect of OO.

Answer Pointers

a) A pattern is a solution to a problem in a context. They are applicable to certain characteristic problems that occur repeatedly in OO models. Practitioners to certain category of problems usually consider patterns solutions as the accumulation of experience.

b) *The answer should make clear the forces at work in the problem and how the pattern presents a clean solution.*

i) The intention of this pattern is to ensure that only one instance of a particular class is ever created and to provide a single global point of access to it. The central feature of this pattern is to use a private class variable with a reference to the unique instance. A public accessor operation provides the global point of access. The class constructor is usually protected or private so that no instances can be created.

ii) The observer pattern is heavily used in GUI applications where one graphical component is affected by changes in another. The latter is sometimes described as the subject and the other(s) is the listener. When a state change in the subject occurs, then all listener objects are advised of this change and update themselves accordingly.

iii) Composite structures such a file directory stores or nested windows in a GUI, exhibit common recursive characteristics. This is customarily captured in the composite design pattern. The elements of a composite are either leaves or a composite object comprising any number of other elements. The key feature of this pattern is that all the elements within the structure are treated uniformly. There is no distinction between a leaf and a composite node.

iv) The decorator is similar to the composite. However, the decorator object provides a wrapper around some other object. Both exhibit the same interface and hence an undecorated and a decorated object may be used in the same context. The decorator is often used to avoid unnecessary specialisation.

v) The abstract factory provides an interface for creating families of related objects without specifying their concrete classes. Because a factory encapsulates the responsibility and the process of creating product objects, its isolate client application code from the implementation classes.

vi) The intent of the visitor pattern is to represent an operation to be applied to the elements of an object structure by an object that carries that operation. This has the effect of separating out the action of traversing the structure from, that which is applied to the elements of the structure. The action is described by the visitor that is then passed among the elements in the structure. Through subclassing we can offer various visitor objects.

QUESTION THREE

Explain what is meant by ANY FIVE of the following used in the context of object oriented development, giving suitable examples.

i) Public, protected and private visibility of class methods and attributes;
ii) The principle of substitution;
iii) Polymorphism and dynamic binding;
iv) Designing to an interface;
v) No concrete superclasses;
vi) Templates and/or generic classes. (25 marks)

A mixed response to this question. Parts b), d), e) and f) were not well answered by respondents. The students were unaware of the principle of substitutability even where they deployed it in their practical coding. In part d) most students discussed HCI and not the OO notion of an interface. In part e) students were unaware of the implication of multiple inheritance, the problems, and the solution of using interfaces. Perhaps part f) was a difficulty if the student had used Java. However, even here we have generic container classes.

Answer Pointers

i) *The answer needs to highlight the distinction between these levels of visibility and how they are deployed by a developer.*

The public features of a class are visible to all other classes. A public method of one class can be called from 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 superclass.

ii) *The answer should highlight the flexibility and adaptability achievable through this technique.*

Liskov first enumerated the substitution principle. The principle permits an instance of a subclass to be used where an instance of a superclass is required. For example, a method expecting an instance of a Person object as a parameter may be invoked with an instance of an Employee object, where class Employee is a subclass of Person.

Substitution makes systems more adaptable to change. For example, an object model in which there is a one-to-many relation between Bank class and Account class, could at run time relate a Bank instance with various instances of subclasses of Account, such as CurrentAccount or DepositAccount. Further, the Account class hierarchy could be extended both horizontally and vertically without impact on the Bank class.

iii) *This* questions *follows from part b). Here, we are looking to explore the benefits of the effect of dynamic binding.*

Polymorphism = many forms. This captures the notion of substitutability. A Bank object may associate with many Account objects. The latter may be a mix of different types of accounts that are specialisations of the Account class.

A Bank object is permitted to send the same message to each of its Account objects. The actual behaviour will be determined by the type of the recipient object. In the Account class hierarchy a method, possible deferred, will be introduced in the Account superclass and redefined in one or more subclasses. The actual method executed will be determined by the type of Account object receiving the message.

iv) *The answer needs to identify that we aim to decouple an abstraction from its implementation.*

The key to this issue in OO is to define interface classes at the root of a class hierarchy. An interface is characterised by having only deferred methods. In a particular problem a concrete subclass will provide a realisation of that interface.

This provides the flexibility whereby we can extend the interface and provide further realisations of that new interface. A new application can operate to that new interface and the first remains unchanged.

v) *The rationale for this principle should be the aim of the answer.*

A principle related to part d) is that no superclasses in a class hierarchy should be concrete. Violation of this principle can lead to significant problems as a design evolves. The problem that frequently arises is that the superclass ends up having more than one role, namely an interface for all its superclasses and a default implementation for one or more of its methods. In particular this happens when specialised functionality must be provided with instances of the concrete superclass.

We can avoid this situation by making all superclasses abstract.

vi) *A template or generic class lets us prepare a family of classes by providing the definition one.*

The most common example of these classes are the container classes used in OO modelling. A container might be required to hold a collection of bank accounts or a collection of employees. Rather than describing each container separately, we define a container in terms of some arbitrary type. The generic container can then be instantiated any number of times to create the particular container required by the application.

QUESTION FOUR

a) **Explain what is meant by the techniques “black-box testing” and “white-box testing” and how they may be used when testing object-oriented software. (10 marks)**

b) **Explain the purpose of "integration testing" and how it is used to assure the quality of object-oriented systems. Why is the integration testing of an object-oriented system likely to be more complex than that for a system developed using a top-down decomposition approach? (5 marks)**

c) **Discuss whether, when testing a subclass, it is safe to assume that all the methods inherited from a previously tested superclass will function correctly. (5 marks)**

d) **In the traditional waterfall approach to systems design, testing takes place in a separate phase of development. Discuss whether this is appropriate when constructing object-oriented software. (5 marks)**

In part a) most candidates could describe both black and white box testing, however, many were unable to relate it to OO programming. Some candidates argued that encapsulation prevents testing!

Part b) did not attract many good answers. Many candidates chose to write a general description of testing but did not relate it to object-orientation, credit was only awarded to material which was relevant to the question.

In part c), many candidates assumed that superclass methods would work and therefore did not require testing.

Part d) attracted good description of the waterfall approach but only a few candidates discussed it in the context of OO development.

Answer Pointers

a) Black-box testing is testing generated from a specification. The specification is used to identify ranges of inputs which will generate certain values. These are called equivalence partitions. The tester identifies boundaries between equivalence partitions and generates test values either side of the boundaries.

White-box testing is undertaken with respect to a knowledge of the internals of the code. It can be shown that black-box testing often fails to exercise all the code in a routine. In a white-box test branches within the code are examined so as to generate test data that will force all paths in the code to be traversed.

In O-O black box and white box testing can be applied to classes to test that the class meets its specification and that all branches within methods are exercised.

b) Assuming pieces of code have been tested in isolation from each other (using stubs and skeleton code), integration testing checks that when a system is assembled from tested fragments that it works correctly. In hierarchically constructed systems control normally flows up and down the hierarchy whereas in an object-oriented system control flow is more complex. This makes O-O integration testing more difficult.

c) When inheritance is exploited the there will be a certain amount of code reuse. There may therefore be no need to re-test this previously tested code. however, it is important to understand that the code will only be correct if it is used in the same context as it was originally designed for. Therefore a careful tester will apply the same tests to a subclass as to a superclass to ensure that the context the code is used in has not altered.

d) The normal mode of object-oriented development is incremental. This may be evident in two types of development. Basic methods for classes may be developed and the additional ones at a later time. Classes may be developed and then subsequently enhanced through inheritance. Leaving testing to the later stages of development is not appropriate given this incremental approach.

Question FIVE

a) **Compare and contrast the following approaches to programming: structured programming, programming using abstract data types and object-oriented programming. (15 marks)**

b) **Describe the claimed advantages of the object-oriented programming approaches. (10 marks**)

In part a) a number of candidates had not come across the concept of an abstract data types and some confused ADTs with abstract classes. A lot of answers stated that structured programming is the opposite of object-oriented programming; this did not receive credit. Many candidates were unable to define the terms structured programming and/or object-oriented programming. Some candidates mistook a question on programming methodologies for one on systems methodologies. In general candidates were poor at comparing and contrasting ideas.

In general, part b) was answered well. Poor answers simply listed OO features rather than claimed advantages.

Answer Pointers

a) Structured programming: A technique for organising and coding computer programs in which a hierarchy of modules is used, each having a single entry and a single exit point, and in which control is passed downward through the structure without unconditional branches to higher levels of the structure. Three types of control flow are used: sequential, test, and iteration.

A collection of data and a set of operations on that data is called an Abstract Data Type. The definition of the operations must be rigorous enough to specify completely the effect that they have on the data yet the definition must not specify how to store the data nor how to carry out the operations. This leads to data hiding and encapsulation.

O-O programming languages takes the concept of an abstract data type and makes it possible to use ADTs as first class types in the language (i.e. variables can have a user-defined type). In addition OO languages allow new types to be developed through an inheritance mechanism.

In structured programming control flow is the most important issue, whereas with ADTs and OO programming data and its associated operations are the most important issue. OO approaches improve on ADT programming by treating ADTs as types in their own right and by supplying a mechanism to re-use existing code.

b) Faster development
Increased Quality
Easier maintenance
Enhanced modifiability
Exploit power of OOPs
Reuse of software and designs, frameworks
Systems more change resilient, evolvable
Reduced development risks for complex systems, integration spread out
Appeals to human cognition, naturalness

Question SIX

A department store employs a number of sales assistants. Sales assistants are normally stationed at tills. The first thing a sales assistant does when they arrive at a till is to log on to the sales system to which all tills are connected. Logging on identifies them to the system. At the end of the day or whenever they are moved to another department the sales assistants log off the till. Only recognised users may log on to a till. The System Manager registers users with the system. The System Manager may also remove users from the system. When a customer wishes to purchase an item they approach a sales assistant who receives a payment from the customer (which may be in the form of cash, credit card or cheque) enters it into the till and records the item that has been purchased. Occasionally, customers are dissatisfied with the purchases they have made and return them. They take them to a sales

assistant who refunds their money and records the return of the item via the till. The advantage of this system to the department store is that it enables managers to get regular sales reports showing how well given items are selling.

a) Draw a use case diagram for the system. (15 marks)

b) Develop a use case description of the way a customer purchases goods. Your answer should include both a normal sequence and an alternate sequence (10 marks)

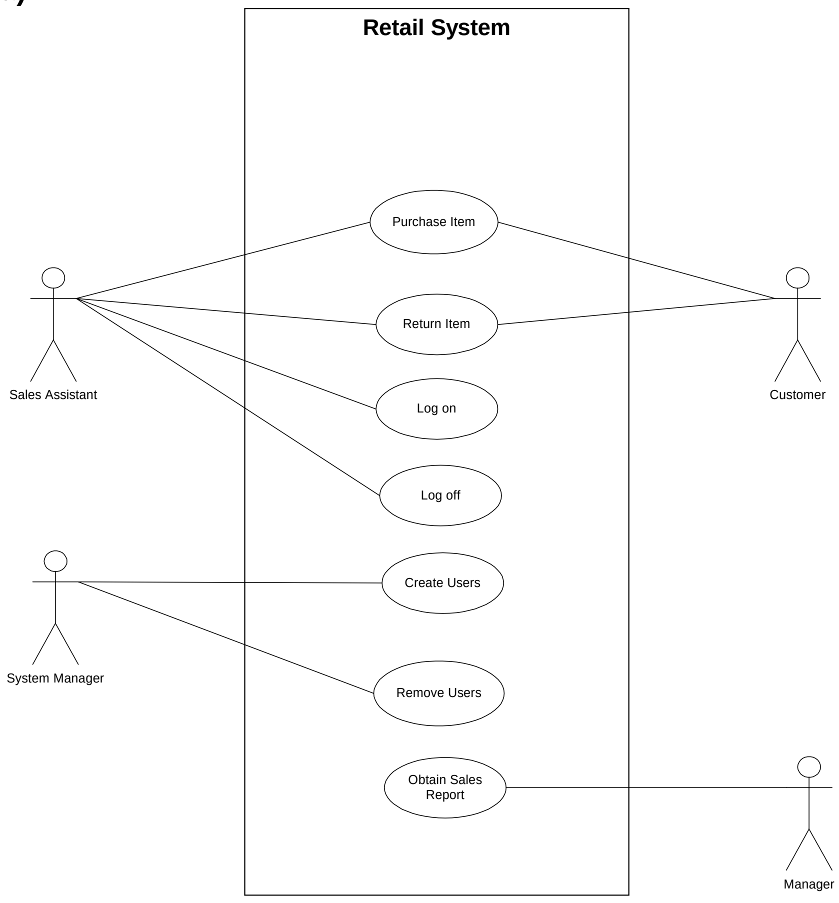
This was a very popular question.

In part a) the most common error was that candidates did not appreciate that there can be more than one actor involved with a use case. One or two candidates submitted answers to last year's use case question indicating that some courses are attempting to teach this topic by rote.

Many answers to part b) attempted to capture all possible outcomes in a single scenario. These answers were full of ifs and elses. What was required was a walkthrough of what happens in the normal operation of the system and in the second sequence a description of one alternative set of events.

Answer Pointers

a)



b) The customer approaches the assistant. Assistant enters goods id. System responds with price. Customer offers credit card. Assistant takes card and swipes in till. Card is accepted. Assistant hands goods to customer.

The customer approaches the assistant. Assistant enters goods id. System responds with price. Customer offers credit card. Assistant takes card and swipes in till. Card is not accepted. Customer offers cash. Assistant checks cash places in till and takes out change. Assistant hands goods to customer.