THE BCS PROFESSIONAL EXAMINATIONS
Diploma

October 2005

EXAMINERS' REPORT

Object oriented Programming

Question 1

1. *a)* Explain briefly the following terms:
- *i)* Structured programming
- *ii)* Modular programming
- *iii)* Abstract data types
- *iv)* Typed language
- *v)* Untyped languages **(10 marks)**

b) Contrast the way in which classes are implemented in typed object oriented programming languages (e.g. C++, Java or C#) and untyped object oriented programming languages (e.g. Smalltalk). **(5 marks)**

c) Define the terms *coupling* and *cohesion* and explain how these concepts contribute to the quality of a program. Show how the object oriented concept of encapsulation aids a programmer to produce good quality code when measures of coupling and cohesion are used to gauge quality. **(10 marks)**

Answer Pointers

1	**This question examines part 1 of the syllabus: Foundations**	
a)	Structured programming is a type of programming methodology where a complex problem is decomposed into tasks which can be modelled by simple structures such as for…next, if….then and while…do Modular programming is a technique for decomposing a programming problem into simpler task which together can be composed to form the solution. Abstract data types or ADTs are data types described in terms of the operations they support—their interface—rather than how they are implemented In typed languages, there usually are pre-defined types for individual pieces of data (such as numbers within a certain range, strings of letters, etc.), and programmatically named values (variables) can have only one fixed type, and allow only certain operations: numbers cannot change into names and vice versa.. Untyped languages treat all data locations interchangeably, so inappropriate operations (like adding names, or sorting numbers alphabetically) will not cause errors until run-time	2 marks 2 marks 2 marks 2 marks 2 marks
b)	In a typed language a class is usually also a type that is variables can be declared to have a type which is identical to a class name. In an untyped language a class is only an object factory which is used to create object which conform to that class. Typed languages perform compile time checks on the messages sent to objects, in untyped languages the object must report at run time if it receives a message it cannot understand.	5 marks
c)	Coupling refers to the degree to which each program module relies on each other module. Coupling can be "high" or "low". Low coupling means that one module does not have to be concerned with the internal implementation of another module. Low coupling is a sign of a well structured computer system. Cohesion refers to the degree to which each part of a module is associated with each other part, in terms of functional relation. Parts of a module are functionally related if each part is essential to the functionality and the interface of a well-defined module (a well-defined module is one that has a single task, or models a single object). Cohesion can be considered "high" or "low". High cohesion means each part of a module is functionally related, and low cohesion means each part of a module is not. High cohesion of a module is considered better design. Information hiding is leaving out some details of implementation on purpose from the public. Encapsulation is, in addition to Information hiding, the provision of common interfaces. Encapsulation supports the notion of gathering together related aspects of a design, grouping them and hiding the implementation. It therefore encourages high cohesion. The combination of information hiding with a well defined set of interface methods also supports low coupling as it	10 marks

	prevents a component of the design gaining access to the internals of an implementation	

Examiner's Comments

In part a) candidates were generally able to supply satisfactory definitions of structured and modular programming. A number of candidates confused abstract data types with abstract classes. A much smaller number were able to give an adequate explanation of typed and untyped languages. Most were able to say that C++, Java and C# are typed languages and that Smalltalk is untyped but no credit was available for this as it was stated in part b) of the question! For full credit candidates needed to briefly explain the difference in the two approaches.

Very few candidates were able to provide a good answer to part b) of the question. The main point that was expected in the answer was that in strongly typed OO languages classes define types and this was not forthcoming.

In part c) the majority of candidates were able to define coupling and cohesion. Only a few were able to demonstrate the way in which encapsulation can help produce programs with low coupling and high cohesion.

Question 2

2. *a)* Explain what is meant by the term *method signature*. **(3 marks)**

b) What is meant by the term *method overloading* and comment on the importance of the method signature in determining that a method has been overloaded? Illustrate method overloading with code written in an object oriented language with which you are familiar. **(5 marks)**

c) What is meant by the term *method overriding* and comment on the importance of the method signature in determining which method is overridden? Illustrate method overriding with code written in an object oriented language with which you are familiar. **(5 marks)**

d) Describe what is meant by the term *operator overloading*. **(3 marks)**

e) Explain how *overloading* and *overriding* contribute to the implementation of polymorphism in object oriented languages. **(9 marks)**

Answer Pointers

2	**This question examines part 2 of the syllabus Concepts**	
a)	A method is commonly identified by its unique **method signature**. This usually includes the method name, the number and type of its parameters, and its return type.	3 marks

b)	Method overloading a feature found in various programming languages that allows the creation of several functions with the same name which differ from each other in terms of the type of the input and the type of the output of the function. In overloading the name of the function is the same but some other part of the signature is different.
	
	class Myclass{
	
	void myMethod(int x){
	.
	.
	}
	
	void myMethod(float x){
	.
	.
	}
	}
	
	The method myMethod is overloaded,
		5 marks

c)	Method overriding allows a subclass to provide its own implementation of a method already provided by one of its superclasses. A subclass can give its own definition of methods which also happen to have the same signature as the method in its superclass.
	
	class MySuperclass{
	
	void myMethod(int x){
	.
	.
	}
	}
	
	class MySubclass{
	
	void myMethod(int x){
	.
	.
	}
	}
	
	The method myMethod is overridden.
		5 marks

d)	Operator overloading is a specific case of polymorphism in which some or	3 marks

	all of operators like +, = or == are treated as polymorphic functions and as such have different behaviours depending on the types of its arguments.	
e)	Polymorphism is the idea of allowing the same code to be used with different types, resulting in more general and abstract implementations Overloading allows multiple functions taking different types to be defined with the same name; the compiler or interpreter automatically calls the right one. This way, functions appending lists of integers, lists of strings, lists of real numbers, and so on could be written, and all be called *append*—and the right *append* function would be called based on the type of lists being appended. This differs from parametric polymorphism, in which the function would need to be written *generically*, to work with any kind of list. Using overloading, it is possible to have a function perform two completely different things based on the type of input passed to it; Overiding is used to implement parametric polymorphism. Using parametric polymorphism, a function or datatype can be written generically so that it can deal equally well with objects of various types. Object oriented languages employ the idea of *subtypes* to restrict the range of types that can be used in a particular case of parametric polymorphism. In these languages, subtyping polymorphism allows a function to be written to take an object of a certain type *T*, but also work correctly if passed an object that belongs to a type *S* that is a subtype of *T* (according to the Liskov substitution principle).	9 marks

Examiner's Comments

The majority of candidates were able to define a method signature in part a) of the question. In parts b) and c) most answers were correct and were illustrated adequately with correct code fragments. Some candidates, however, confused the two terms overloading and overriding.

Part d) attracted very few correct answers. This is possibly due to the fact that operator overloading is not universally implemented in object oriented programming languages. It is, however, explicitly mentioned in the syllabus and therefore candidates should be able to explain the term.

Part e) was a question that has been asked many times previously in papers examining the Object Oriented Programming syllabus. On each occasion the response from candidates has been similar. The majority are able to give a definition of polymorphism but few are able to show how object oriented features such as overloading and overriding allow programmers to make use of polymorphism.

Question 3

3. *a)* Define the following terms:

i) Object
ii) Class
iii) Constructor
iv) Destructor
v) Memory management **(10 marks)**

b) The following code, written in an idealised object oriented pseudo language, shows a situation where there are two classes myClass and mySuperclass. myClass inherits from mySuperclass. Both myClass and mySuperclass have a single constructor.

```
class mySuperclass{
        private integer x

        constructor(){
                x=0
        }
}

class myClass inherits from mySuperClass {
        private integer y

        constructor(){
                y=0
        }
}
```

Describe what occurs in terms of memory allocation and value assignment when an object of class myClass is created. **(5 marks)**

c) Discuss the advantages and disadvantages of implementing garbage collection in an object oriented programming language. **(10 marks)**

Answer Pointers

3	**This question examines part 2 of the syllabus Concepts**	
a)	An object is a language supported mechanism for binding data tightly with methods that operate on that data. A class specifies the structure of data which each instance contains as well as the methods (functions) which manipulate the data of the object. A constructor is a special method invoked when an object is created which normally assigns default values to instance variables of the object. A destructor is a special method invoked when an object is no longer needed which normally is used to reclaim memory allocated to the object. Memory management is the act of managing computer memory. In its simpler forms this involves providing ways to allocate portions of memory to programs at their request and free it back to the system for reuse when no longer needed.	2 marks 2 marks 2 marks 2 marks 2 marks
b)	When an object of myClass is created space will be allocated to hold two integers. This will hold the variables labelled x and y. The constructor for mySuperclass will be invoked and x will be set to zero, then the constructor for myClass will be invoked and y will be set to zero.	5 marks
c)	Memory management can be an extremely complex task. Every object must be assigned memory before it can be used. Objects no longer in use should be disposed of and their memory reclaimed as failure to do this can result in programs failing because they are unable to allocate any more memory. A number of commercial systems exhibit faults of this kind Programmers have to be careful however not to release storage which is still required. Garbage collection is an automatic way of claiming unused storage which is not reliant on the programmer. This makes programs far easier to construct as the programmer need only concentrate on the task to be done and not the low level implementation details. Accurate garbage collection is however difficult to implement. For example, it may seem logical that one should only reclaim storage if an object has nothing referencing it but if this is the only policy in use, self-referencing objects will never have their storage reclaimed. In addition, garbage collection adds a significant processing overhead since the garbage collector may be active when it is not necessary to reclaim storage.	10 marks

Examiner's Guidance Notes

Part a) could have been answered from book knowledge but despite this not all candidates were able to answer it. Many confused class and object. Only a minority could offer a definition of a destructor.

Part b) was generally answered correctly. Two main things were sought in the answer. Firstly, an understanding of the memory allocation that takes place when an object is created and secondly the order in which memory is allocated.

Part c) was poorly answered. A number of candidates showed no appreciation of the need for memory allocation and deallocation. Of those who were able to define garbage collection, few were able to discuss its advantages and disadvantages.

Question 4

4. Felix Pets Ltd is a veterinary surgery, which caters for domestic animals. A pet owner can either phone the receptionist to book an appointment, or turn up to an open session in the morning.

When an owner takes their pet to the surgery they first have to register their animal to be seen by one of the vets on duty. The receptionist asks for the owner's name and the name of their pet, so their details can be looked up. If the owner is new to the surgery, or an existing owner has a new pet, their details have to be added to the system. On most occasions only one pet is brought to the surgery, but sometimes two or more can be brought in, which means all pets must be registered. Once the pets are registered they are then allocated to one of the vets on duty and the owner waits for their consultation.

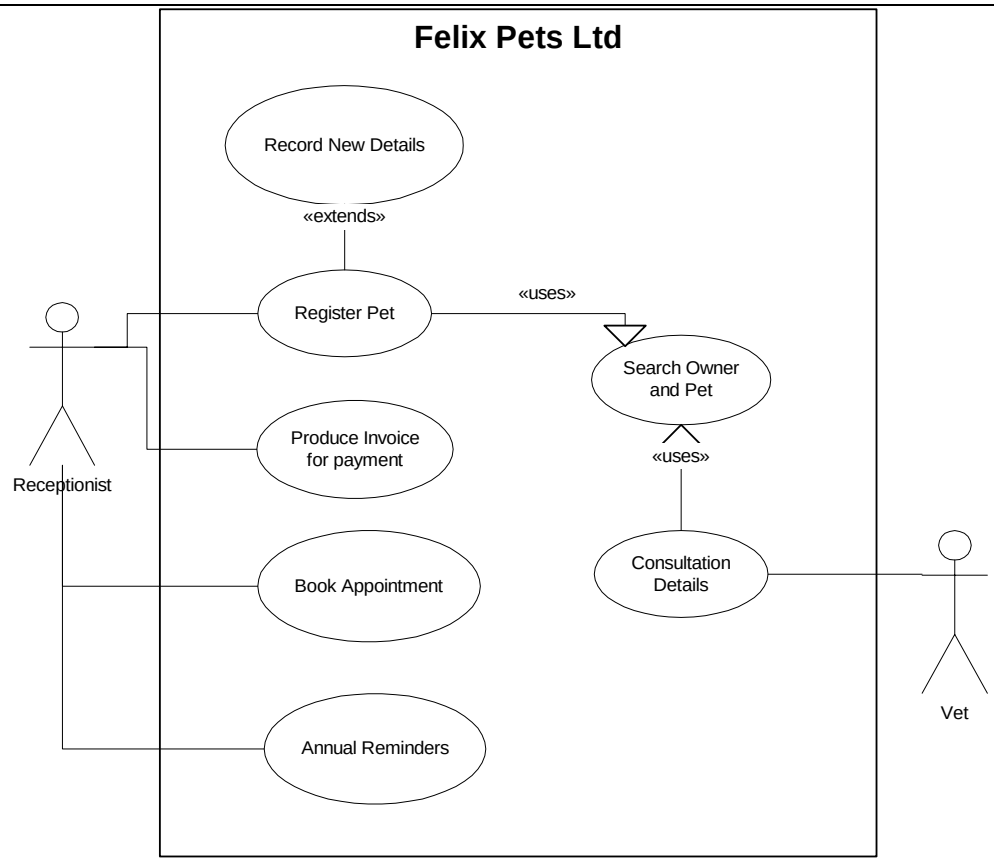
When the vet is ready, the vet first has to look up the details of the owner and pet on the system. After the vet treats the animal they will write up their notes on the system, detailing any problems and the treatment given. If any further treatment is needed the vet will produce a list of medication for the receptionist to dispense. The receptionist will then produce an invoice for the owner to pay.

Some pets will have annual injections, for example, for flu jabs. Once a week the receptionist will produce a list of reminder letters to send out to owners, reminding them their pet's booster is due.

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

b) Write down a use case description of the way an owner registers their pet. Your answer should include a normal sequence and two alternative sequences. **(10 marks)**

Answer Pointers

4.	**This question examines part 3 of the syllabus Design**	
a)		15 marks 1 per use case 1 per actor 2 for extends 2 for each uses
b)	**Actor Action** 1. User registers pet and provides their name 3. User provides pet's name 5. Allocate to vet on duty **Alternative sequences** Step 2. User not on system, add new details Step 4. Pet not on system, add new details	10 marks

Examiners Comments

The UML question provided to be popular and on the whole, a good attempt was made on part A. Most candidates identified the main use cases and actors, but some lost marks by not recognising the potential for the *extends* and *uses* relationships. A few candidates identified the main use cases, but

then neglected to connect them up appropriately. Some inappropriate actors were used, such as an animal.

In part B, most candidates provided a basic description, with marks lost for not providing any alternatives. The use case description should only be for the one use case – registering the pet, a lot of candidates lost marks by merging the descriptions of several use cases together.

Question 5

5. *a)* In the context of object oriented development, what is meant by the term *design pattern*? **(5 marks)**

b) Explain the motivation for using design patterns from a programmer's point of view. **(5 marks)**

c) Choose THREE of the following design patterns and give a detailed description of each, stating the problem they address and the basis of the solution they offer:

i) Adaptor

ii) Decorator

iii) Iterator

iv) Observer

v) Singleton

Answer Pointers

5.	**This question examines part 3 of the syllabus Design**	
a)	In OO terms a pattern is a named problem/solution pair that can be applied in new contexts with advice on how to apply it in novel solutions. Basically general principles and idioms codified in a structured format.	5 marks
b)	For a programmer design patterns are devices that allow programs to share knowledge about their design. Programmers encounter many problems that occur again and again and they often have to ask themselves is "how we are going to solve it *this* time". Documenting patterns can lead to reuse, possibly allowing the information to be shared with other programmers about how it is best to solve a specific program design problem.	5 marks
c)	The five listed design patterns are from the ones listed in the syllabus and from Gamma. **Adapter** The Adapter is intended to provide a way for a client to use an object whose interface is different from the one expected by the client, without having to modify either. This pattern is suitable for solving issues such as replacing one class with another when the interfaces do not match and creating a class that can interact with other classes without knowing their interfaces at design time.	15 marks 5 per design pattern

	Decorator Ordinarily, an object inherits behaviour from its subclasses. The decorator pattern allows the behaviour of an object to be extended and dynamically compose an object's behaviour. Decorator allows this without the need to create new subclasses. **Iterator** The Iterator pattern provides a way to access the elements of an aggregate object sequentially without having to know the underlying representation. Iterator also provides a way to define special Iterator classes that perform unique processing and return only specific elements of the data collection. The Iterator is useful because it provides a common interface so programmer does not need to know anything about the underling data structure. **Observer** The Observer pattern is useful when data is to be presented in several different forms at once. The Observer is intended to provide a means to define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The object containing the data is separated from the objects that display the data and the display objects observe changes in that data. Often used in MVC. **Singleton** The singleton pattern applies to the many situations in which there needs to be a *single instance* of a class, a single object. Print spoolers and window managers are examples of Singletons. The Singleton is intended to provide a way to ensure that a class provides one instance of itself, and to provide a global point of access.	

Examiners Comments

Most candidates were good at defining what a design pattern was, but were weaker at saying why a programmer might use one. The adaptor, singleton and observer design patterns proved the most popular choices. Most candidates could describe the pattern adequately, though marks were lost for not saying how the pattern solved the particular problem it addressed.

Question 6

6. A leading UK Bank has employed you as a systems tester to work on their new on-line banking system. The system will be developed using object oriented techniques and testing is needed throughout the project life cycle. You will have to work with the developers and team leaders to develop test cases. It is envisaged the project overall will take 18 months to complete.

a) Describe briefly what different types of testing you might employ as the project develops. **(5 marks)**

b) The team leaders have asked you whether to use the techniques *black-box testing* or *white-box testing*. Explain what these terms mean and how they could be used when testing object oriented software. Include the advantages and disadvantages of each in your discussion. **(15 marks)**

c) Describe what aspects of the design process you might use to develop the plan. **(5 marks)**

Answer Pointers

6.	**This question examines part 4 of the syllabus Practice**	
a)	Types of testing might include: Unit testing Integration testing Sub-system testing System testing Acceptance testing	5 marks
b)	Black-box is a software testing technique whereby the internal workings of the item being tested are not known by the tester. It is also known as *functional testing*. For example, in a black box test on software design the tester only knows the inputs and what the expected outcomes should be and not how the program arrives at those outputs. The tester does not ever examine the programming code and does not need any further knowledge of the program other than its specifications. Advantages include (www.webopedia.com): • The designer and the tester are independent of each other so should be unbiased • The tester does not need knowledge of any specific programming languages • The test is done from the point of view of the user, not the designer • Test cases can be designed as soon as the specifications are complete Disadvantages include: • The test can be redundant if the software designer has already run a test case. • The test cases are difficult to design. • Testing every possible input stream is unrealistic because it would take an inordinate amount of time; therefore, many program paths will go untested. White-box testing is a software testing technique whereby explicit knowledge of the internal workings of the item being tested are used to select the test data. It is also known as *glass box*, *structural*, *clear box* and *open box testing*. White box testing uses specific knowledge of	15 marks 5 per description of each 5 for use with OO

	programming code to examine outputs. The test is accurate only if the tester knows what the program is supposed to do. The tester can then see if the program diverges from its intended goal. White box testing does not account for errors caused by omission and all visible code must also be readable. For a complete software examination, both white box and black box tests are required, not just one. 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.	
c)	The test scripts should be based on the use cases and scenarios.	5 marks

Examiners Comments

This question proved not to be popular. For those who answered the question, most candidates could describe at least 3 types of testing in part A. For part B, generally good descriptions were given of what white and black box testing was, but many candidates failed to say how they could be applied to object-oriented systems. Few candidates answered part C, but those who did, generally gave correct answers.