THE BCS PROFESSIONAL EXAMINATIONS
BCS Level 5 Diploma in IT

April 2008

EXAMINERS' REPORT

Object Oriented Programming

Question 1

a) The company for which you work uses object oriented programming techniques to develop all of the software it produces. Recently, the company has merged with another software house who develop all their software using structured programming techniques. Your manager has asked you to prepare a training course on object oriented programming for programmers with a background in structured programming. Give an outline of the topics the course will cover.

(15 marks)

b) *Coupling* and *cohesion* are two measures which are often applied to establish the quality of structured programs. Explain what is meant by the terms *coupling* and *cohesion*. How does the use of an object oriented programming language help programmers to develop code with appropriate levels of *coupling* and *cohesion*?

(10 marks)

Answer Pointers

a) Whilst object-oriented programming makes use of standard programming techniques such as sequence, selection and iteration, these would be familiar to programmers used to structured programming. The course needs to cover issues which are specific to object oriented programming. Candidates were expected to identify some of the following as important:

Objects, data and operations. Classes. Object creation. Constructors. Inheritance, superclasses and subclasses. Method overloading, method overriding. A brief explanation of each concept was expected.

b) Coupling is the extent to which a module relies on a knowledge of the internal functioning of another module in order to achieve a task. If a module needs to access variables declared within another module then coupling is said to be high and this is generally seen as being unsatisfactory because it may be difficult to modify one module without affecting others.

Cohesion measures how strongly related the different functions of a module are. Cohesion is low if a module implements a number of unrelated tasks. High cohesion leads to good levels of reuse.

Object oriented programming supports low coupling because it hides the internal operation of an object from other external objects. Object oriented programming encourages high cohesion by grouping related sets of operations with the data they operate on.

Examiner's Comments

This question examined Syllabus 8a Foundations

Part a) required the candidate to identify and describe those aspects of object-oriented programming languages which are characteristic of the object oriented paradigm. In other words, the aspects of programming which distinguish an object oriented approach from other approaches. In general, candidates gave good answers to this part of the question which listed all the aspects expected and gave accurate explanations of them.

Part b) looked at one aspect of software engineering and asked the candidates to describe the way that object oriented languages address it. This attracted a number of good answers but not as many as part a). Most candidates were able to define coupling and cohesion (although a few confused the two) but many had difficulty in identifying the object oriented concepts which may be used to achieve high cohesion and low coupling.

The question was answered by the majority of candidates.

Question 2

a)

i) Explain the term *object*;
ii) Explain the term *class*;
iii) What is the relationship between a *class* and an *object*? Illustrate your answer with code.

(8 marks)

b)

i) Describe what is meant by the term *instance method*;
ii) Describe what is meant by the term *class method*;
iii) Explain where it is appropriate to use an *instance method* and where it is appropriate to use a *class method*.

(8 marks)

c) The class Coin represents a monetary coin. Each coin has a monetary value (recorded as a whole number) associated with it.

The class has two instance methods setValue and getValue which respectively set and return the value associated with the coin.

The class has a class method that returns the current number of Coin objects.

Write code that implements Coin.

(9 marks)

Answer Pointers

a)

i) An object is a collection of data. Associated with the data is a set of methods to manipulate the data.

ii) A class is a template for an object it describes the data and the methods which every object based on it will possess.

iii) Classes are factories for creating objects.

```
class MyClass{

        private int x = 0;

        public inc(){
            x++;
        }
}

MyClass mc = new MyClass();
```

In this example an the class declaration defines the structure of an object of MyClass. The new operator is used to create an object of MyClass.

b)

i) An instance method is applied to an object and normally manipulates data contained in that object. It is necessary to identify an object instance in order to use an instance method. For example:

```
MyClass mc = new MyClass();
mc.inc();
```

Here the instance method inc is invoked for the object referenced by mc.

ii) A class method does not operate on an object or its data. It is a method placed within a class because its operation is related to the purpose of a class. For example in a class which deals with dates there might be a class method which does not operate on any specific date but instead returns the system date. Such a method is invoked using the class name rather than via an object reference, for example MyClass.myMethod() would invoke the class method myMethod.

iii) Class methods are used to manipulate data which describes an aspect of the class but which is independent of an instance of the class. Instance methods manipulate data which is specific to an instance of a class.

c)

```
class Coin{
        private static int oCount; = 0;
        private int value;

        public Coin(){
                oCount++;
        }

        public setValue (int v){
                value = v;
        }

        public int getValue(){
                return value;
        }

        public static int getNoCoins(){
                return oCount;
        }
}
```

Examiner's Comments

This question examined Syllabus 8a and 8b Foundations and Concepts

This was a popular question which attracted a number of good answers.

Part a) was largely book work with the additional requirement that candidates illustrate the final section with code. Most candidates gave good answers for this part of the question.

In part b) candidates often distinguished between class and instance methods by giving examples of their declarations. These answers were not deemed sufficient to receive the full eight marks. For full credit a candidate had to demonstrate an understanding of why the two types of method are necessary.

Disappointingly, part c) was not answered as well as parts a) and b). Although the code required is very simple, many candidates produced incorrect answers. It appears that whilst candidates have book knowledge of object oriented programming, many lack the practical programming experience which would aid them to produce good answers to this type of question. This year there was a greater variety of programming languages in the answers given. The majority of candidates have a knowledge of Java but clearly C++ and VB.Net are also taught. As always, answers in any object oriented language were accepted.

Question 3

a) Most object oriented programming languages provide a class library which contains a variety of collection classes. Explain what a collection class is and the features you would expect a collection class to possess.

(8 marks)

b) Discuss the relationship between the Iterator design pattern and collection classes.

(8 marks)

c) Give a fragment of code which reads in 100 numbers and prints them out in reverse order.

(9 marks)

Answer Pointers

a) Programs frequently manipulate more than one object. For example in data processing programs manipulate sets of customer records. Collection classes enable objects to be grouped together in a manner which is convenient for the programmer. You would expect to be able to add or remove an object from a collection class, also to locate a specific object within a collection. Collection classes will have different properties which make them useful in different applications. A Set is useful in situations where duplicate elements are not allowed, a List allows duplicates, a Queue specifies FIFO and a Map permits indexed retrieval.

b) The Iterator pattern is specifically for navigating through collection classes. It provides a mechanism which allows a programmer to access a collection class without having to understand the way in which the collection class has been implemented. Consequently, the programmer invokes the same method calls regardless of whether the collection class has, for example, been implemented as an array or a linked list. Iterators typically support methods such first(), next(), atEnd().

c)

```
public static void main(String[] args) {
   Scanner sc = new Scanner(System.in);
   Stack s = new Stack();
   int i;
   for(int j=0;j<100;j++){
      i = sc.nextInt();
      s.push(i);
   }
   for(int j=0;j<100;j++){
      System.out.println(s.pop());
   }
}
```

Examiner's Comments

This question examined parts 8b and 8c of the syllabus Concepts and Design

This question examined a part of the syllabus which has not been tested in very many past papers. In addition, it required candidates to make a connection between two aspects of the syllabus collection classes and design patterns. Only a small number of candidates attempted the question. A few candidates scored very highly indicating they knew the topics but the majority of answers given were very poor. This could be interpreted as evidence that candidates are preparing for the examination by studying past papers and failing to explore the full breadth of the syllabus.

Part c) was a simple programming exercise with many possible solutions. It is unlikely that candidates studying a practical programming course would not come across a problem similar or identical to the one set. A solution using an array would have received full credit. There were, however, very few correct solutions presented. This tends to suggest that a number of candidates are attempting to learn programming from a text book instead of learning by writing code.

Question 4

a) Describe the role of the garbage collector.

(5 marks)

b) Explain the role of a constructor; state how its signature will differ from that of other methods.

(6 marks)

c) A class C1 contains private, public and protected data members called x, y, and z, respectively. A sub-class C2 is to be created by publicly inheriting from C1. Which members of C1 will be inherited, and what will their designation be (private, public or protected) in the sub-class C2? Provide a table to illustrate your answer.

(6 marks)

d) Explain the difference between method overloading, and method overriding, using an appropriate example of each.

(8 marks)

Answer Pointers

(a) The garbage collector keeps track of the number of references to each object, detecting when no more references to an object exist. It de-allocates the memory that was used by objects that move out of scope to avoid memory leakage, returning this memory for re-use by the system. This is in contrast to languages without garbage collection, where manual de-allocation of memory is necessary, e.g. using destructors. With manual memory de-allocation, memory leakage is possible where this process is not carefully managed. (5)

(b) Constructors are special methods used to initialize the state of new objects (2). Their signatures have two special characteristics. Firstly, they always have the same name as the class they belong to (2). Secondly, they do not have a return type. (2)

(c)

Data Member	**Visibility in C1**	**Visibility in C2**	
x	private	not inherited	(2)
y	public	public	(2)
z	protected	protected	(2)

(d) Both method overloading and method overriding are examples of polymorphism (i.e., using the same syntax to perform different tasks, depending upon the context). Method overloading describes the ability to have several methods in a class that all share the same name. They are distinguished only by the fact that their arguments are different (either in type, number, or type order). The compiler will match the invocation to the appropriate version of the method when a call occurs (2). A C++ example is provided below (2):

```
  class A
  {
public:
void foo(int x);              // defines version 1
void foo(double x);           // defines version 2
void foo(double x, int y);    // defines version 3
   };
      main()
     {       A myA;
             myA.foo(5.1);              // invokes version 2
             myA.foo(6.1, 5);           // invokes version 3
             myA.foo(2);                // invokes version 1
     }
```

Method overriding describes the ability of a derived class (sub-class) to redefine a method that was previously implemented in a base class (super class). In some languages this is accomplished with the aid of the keyword *virtual*, which, when used as a prefix on a method, indicates that it is to be redefined by the lowest class in the inheritance lineage. Without the use of the virtual keyword, the prior version can still be invoked through derived class objects using special syntax. An assumption is that the signature of the redefined version of the class which match that found earlier in the class hierarchy, otherwise method overloading will occur instead (2). A C++ example is provided below (2):

```
class A
{
        public:
                virtual void foo(int x) { cout << “Old”; }
};

class B: public A
{
        public:
                void foo(int x) { cout << “New”; }
};
```

Examiners Comments

This proved to be the most popular question. Most students were able to answer the bookwork questions, designed to measure understanding of basic principles in object oriented programming, successfully (a, b, d). However, many had difficulty in applying what they knew about inheritance to the example scenario in question c, showing incomplete understanding of the mechanics of inheritance. Most candidates provided a description of the garbage collector for question a. In question b, candidates were mostly able to describe the role of constructors and often (correctly) stated that constructors have the same name as the class they belong to, however many neglected to mention that they do not have a return type. Many of the candidates confused their descriptions of method overloading and method overriding in question d, which had the highest weighting of any question (8 marks). Like question c, an inadequate description of method overriding showed an incomplete understanding of a very important form of polymorphism used in inheritance.

Question 5

a) Which class members constitute the *interface* of a class?

(5 marks)

b) Distinguish between *service methods* and *support methods*.

(4 marks)

c) Describe two possible roles of a *destructor* method.

(4 marks)

d) Explain the meaning of the term *encapsulation* in the context of object oriented programming.

(4 marks)

e) Explain why is it not possible to declare an *instance* of an *abstract class*. Illustrate your answer with an appropriate example.

(8 marks)

Answer Pointers

a) The interface of a class consists of its public data members and methods, i.e., the portion of the class that can be interacted with. By stipulating that these members are public, they are able to be invoked (for methods) or accessed (for data) through an object of the class from outside, whereas private methods and data are only accessible internally (5).

b) Service methods are public (1), and are part of the externally accessible portion of a class, i.e., they are able to be invoked through an object of the class (1). Support methods are private, and exist to support the other methods within the class by performing some internal duties (1). They are not able to be invoked through an object (1).

c) Two possible roles of a destructor include de-allocating dynamically allocated memory (in languages like C++) (2), and closing data files that have been opened or communication channels such as sockets (2).

d) Encapsulation is the technique of bringing together associated data types/structures (2) and the operations used to manipulate those data types/structures (2).

e) An abstract class contains abstract methods (known as pure virtual functions in C++ or simply as abstract methods in Java), that are *incomplete*. Because a portion of the behaviour of an abstract class is yet to be defined, it is not permitted to create objects, since invoking non-existent methods is impossible. Abstract methods are used to specify that particular named methods must be implemented later. This is an example of inheritance for “specification” purposes. For example, all subclasses of hypothetical class Animal must implement a method Reproduce(), so we include this method in abstract form in Animal despite that we don’t yet know how it is to be accomplished. It is not until we extend to create more specific sub-classes (e.g., Dog and Amoeba) that are concrete (i.e., do not contain abstract methods), that we know how these methods need to be implemented (i.e., sexual or asexual reproduction for the Dog and Amoeba classes, respectively). (4)

A C++ example follows:

```
class Animal
{  public:
    virtual void Reproduce()=0;    // a pure virtual function
};
class Dog: public Animal
{  public:
    virtual void Reproduce() {cout << “sexual reproduction”;}
};
class Amoeba: public Animal
{  public:
    virtual void Reproduce() {cout << “a-sexual reproduction”;}
}; (4)
```

Examiners Comments

Approximately half of the candidates attempted this question. However, it was not generally well answered. The first four parts probed students understanding of some fundamental concepts in object oriented programming (in particular, questions a and d), and many of the students proved insufficiently familiar with these concepts to acquire some of the available marks. The final part of the question required the student to describe and supply an example of abstract classes (including showing why abstract classes are not to be instantiated); many candidates attempted this question, but a smaller proportion were able to identify and correctly present an example to illustrate their argument.

Question 6

mp3Now is a digital music company that operates entirely on the web. Customers browse the catalogue for the songs wanted, add them to their basket, and proceed to the secure check out. The checkout process involves, for first time users, registering a username, password, delivery address, and credit card details. For subsequent visits, this information can be retrieved using the original username and password. Once customer's payment has been accepted, the music files purchased become available to download from the user's on-line account for a period of 7 days.

Items are charged to the customer on a per-song basis; a database of current prices is maintained by the company with modifications made at the request of the record companies (suppliers). A large number of record companies provide permission to mp3Now to offer their artist's work. Each record company has an individual contract with mp3Now stating the percentage of the revenue generated for each song sold that they expect to receive.

a) Identify suitable classes in this system, and draw a *class* diagram showing how they are inter-related.

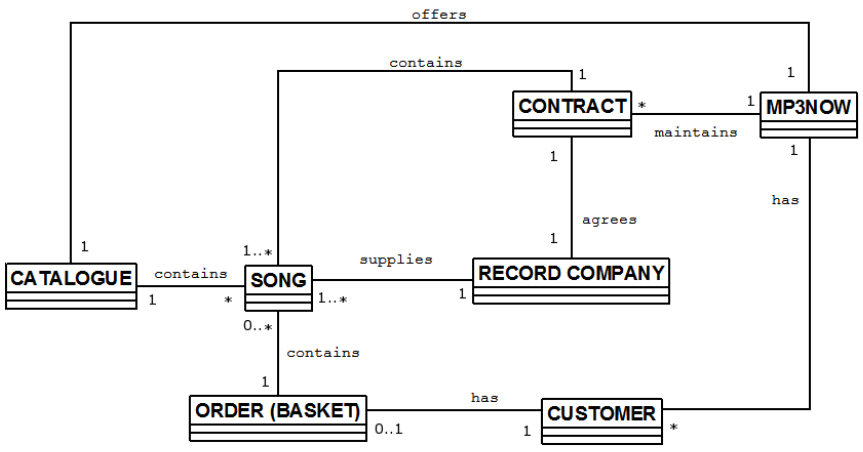
(15 marks)

b) Draw *use-case* diagrams to show the roles of two actors: customer and record company.

(10 marks)

Answer Pointers

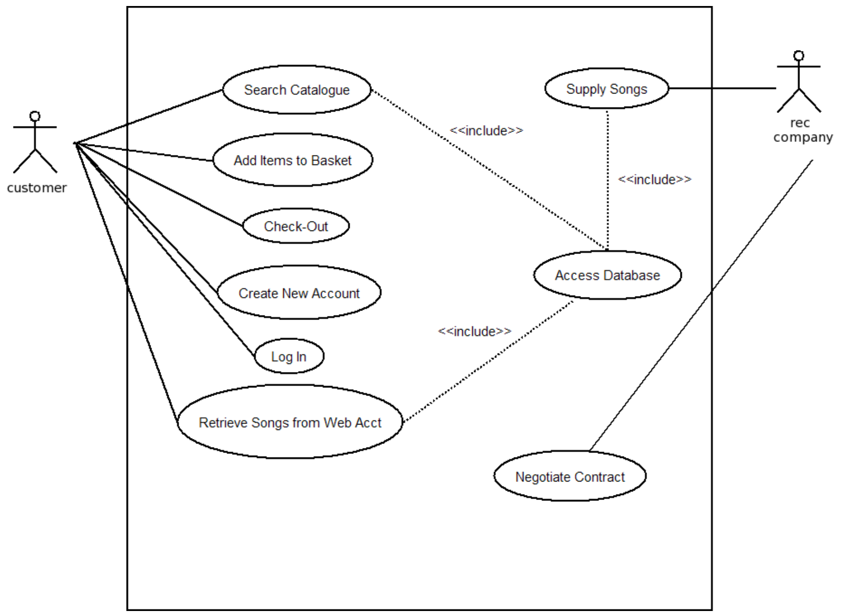
a) An example class diagram is shown below:



Variability in candidate's model of the system is to be expected (including classes identified, and associations), but the primary functions of mp3Now should be feasible with the diagram submitted (in the sense that the customer, record company, and company may interact to enable purchase, supply, and selling of music, respectively), and the diagram should employ correct notation (UML is not specifically mentioned in the question, so other class diagram notations are permissible).

b)

An example use-case diagram is shown below:



Examiners Comments

The question was designed to measure candidate's ability to use object oriented design to create class and use-case diagrams from a clearly written scenario of a company with a straightforward business model. A high proportion of candidates attempted this question, and they were rewarded with the highest pass rate of any of the 6 questions set. Most candidates made a valid attempt at a class diagram. Although many of these were not entirely logical (i.e. would have benefited from greater thought before they were drawn), most of the students gleaned marks for identifying a number of appropriate classes and proposing several valid relationships between those classes. Several candidates incorrectly used inheritance relationships between classes. The second part of the question, requiring candidates to provide a use-case diagram, was, again, reasonably well answered with most candidates providing a valid (though often incomplete) diagram that largely conformed to the scenario provided in the question. Despite that the question specifically requested 2 actors, some candidates provided 3 or more actors in the diagram, accruing no extra marks.