BCS HIGHER EDUCATION QUALIFICATIONS
BCS Level 5 Diploma in IT

March 2012

EXAMINERS' REPORT

Object Oriented Programming

General Comments

Candidates are once again reminded that this paper is set to a standard equivalent to that found in the second year of a UK Honours degree. It is therefore necessary to undertake an academic study of object oriented programming in order to obtain a pass mark. This might be achieved via a formal course or through a careful study of the texts cited in the syllabus. In addition, questions in this paper may ask for code examples and therefore candidates should have some practical experience of programming in an object oriented language.

SECTION A

QUESTION A1

```
class atomClass
{
  private:
    int protons;
    int neutrons;
    int electrons;
  protected:
    static int electronCharge;
  public:
    atomClass();
    atomClass(int p, int n, int e);
    void setProtons(int p);
    int getNeutrons();
};
```

a)

i. Explain the consequences of having set the neutrons data member to private. Why is it considered good practice for data members to be designated as private?

(3 marks)

ii. Under what circumstances would it be more appropriate for neutrons to be designated as protected?

(3 marks)

iii. Identify and briefly explain **ONE** example of polymorphism in atomClass.

(3 marks)

iv. Identify an accessor function in atomClass and provide an implementation for it.

(3 marks)

v. Identify the class and instance variables in atomClass, and state how many variables there would be in total if we declared 3 objects.

(3 marks)

Answer Pointers

i. Having been set to private, the **neutron** data member may only be accessed via the functions of **atomClass**. Designation of data members of **private** protects them from loss of integrity by requiring that modifications from outside the class are requested via

mutator functions, such that bounds-checking and other tests may be performed to ensure the validity of the request. (3)

ii. If the **neutron** data member were set to protected, it could be manipulated directly by the functions of a derived class created from **atomClass**. This may be done for convenience (to save manipulating **neutron** through inherited accessor and mutator functions), or because the base class mutator function is no longer appropriate (e.g., enforces bounds restrictions that are no longer valid). (3)

iii. The **atomClass** contains only a single example of polymorphism – function overloading (more specifically, constructor overloading). In this case, it allows objects of **atomClass** to either be made empty, typically such that data members are initialised to 0, or to be made and initialised to specific values, **p**, **n** and **e**. (3)

iv. The only accessor function who's signature is explicitly provided in the **atomClass** definition is **int getNeutrons();** Its implementation would simply be **{ return neutrons; }**, since its job is to dispatch the value of that data member to whomever called the function.

v. The instance variables in **atomClass** are protons, neutrons and electrons, and the single class variable is **electronCharge**. If we created 3 objects of **atomClass**, there would be 10 variables in total, 3×3 instance variables and 1 class variable.

b) If we wished to publically inherit from atomClass, explain which members would be visible in the derived class, and state their designation.

(5 marks)

Answer Pointers

If we were to publically inherit from **atomClass**, then the 4 **public** member functions would be visible in the derived class. Their designation in the derived class would also be **public**. The single **protected** data member would be visible in the derived class, and its designation would remain **protected**. The 3 **private** data members would not be inherited.

c) Explain the purpose of a copy constructor, and propose how one might be implemented for atomClass, assuming that we had a full suite of accessor functions.

(5 marks)

Answer Pointers

A copy constructor duplicates the state of an existing object passed as an argument (i.e., copies the values of the data members of the argument object into the present object). A copy constructor for **atomClass** could be implemented like this:

```
atomClass(atomClass a)
{
  protons   = a.getProtons();
  neutrons  = a.getNeutrons();
  electrons = a.getElectrons();
}
```

Note that there is no need to duplicate the class variable since this is already shared among all objects of **atomClass**.

Examiner's Comments

This was the second most popular question in the paper, having been attempted by 84% of candidates; however, only one half of candidates 'passed' the question (i.e., scored 40% of the available 25 marks). The answers submitted were not as successful as one would have hoped, despite being broadly similar to questions that have appeared in past papers. Some students answered part (a) fully and completely, but for others, significant holes in the ability to interpret a class definition, expressed in code, was revealed, possibly alluding to more basic deficits in the understanding of fundamental object oriented concepts. In part (b), most candidates had realised that the 4 public methods would be inherited, but some incorrectly thought that the protected member would not be inherited, or that the private data members would be inherited. In part (c), a good number of candidates answered (or deduced) the purpose of a copy constructor, but relatively few provided a viable code fragment illustrating how one would be implemented.

QUESTION A2

a) Differentiate between the following pairs of terms:

i. method and message

(3 marks)

ii. method overloading and method overriding

(3 marks)

iii. early binding and late binding

(3 marks)

iv. constructing and compositing

(3 marks)

v. instance and instance method

(3 marks)

Answer Pointers

i. A method is a class behaviour – typically, a member function that performs an activity enabling objects of a class to be manipulated. Messages are communications between objects – for instance, where one object may call a public member function of another, and supply data as an argument to that function. (3)

ii. In method overloading, multiple versions of the same member function are supplied, such that argument list is unambiguously different, either in terms of the number of arguments, their data types, or order. An appropriate version of the member function is selected to match the arguments supplied at invocation. Method overriding is a technique whereby the implementation of a base class function may be replaced by a new version in a derived class, typically implemented by designating a base class function as **virtual**. (3)

iii. Early binding occurs when at compile time, it is clear (from examining our code) which version of a function will be executed. Late binding enables a program to decide which version of a function to invoke at run-time, depending upon the object type pointed to. It is commonly implemented using virtual functions, and exploiting

the fact that a pointer to a base class type may also be set to point to a class derived from that type, which may change from moment to moment as the program is running. (3)

iv. Constructing is the name given to the procedure by which objects are created from a class definition. Constructor functions undertake such tasks as initialising data members, making dynamic memory allocation requests, opening files or communication sockets. Composition is an ownership relationship between two classes, such that if an object of the owner class is destroyed, the owned class does not survive. In design terms, it implies the possession of the composited class by the compositing class: for instance, a University may be made up of several Faculties, but those Faculties cannot exist without the University, so are composited within it. (3)

v. An instance is another name for an object created from a class. An instance method is a method that is used to manipulate instance variables in an object (in contrast to class method, which can be executed without the creation of any instances, and may only manipulate class variables, not instance variables). (3)

b) What is the ‘yo-yo problem’ in object-oriented programming?

(5 marks)

Answer Pointers

The ‘yo-yo problem’ describes the situation that arises a deep inheritance hierarchy is used, requiring that the programmer flick back and forth between classes to follow program logic and structure. Excessively deep inheritance hierarchies should therefore be avoided, perhaps relying upon inter-class relationships other than inheritance when possible.

c) Why might some languages discourage or prohibit multiple inheritance?

(5 marks)

Answer Pointers

Some languages prohibit or discourage multiple inheritance because it can lead to ambiguities. For example, in creating a class D we might inherit from two base classes (B and C) that were themselves both inherited from a common base class (A) creating a “diamond”. In class D, we effectively inherit from class A twice, meaning that any reference to a data member or function originating from A is ambiguous, since they are visible through both B and C. The “diamond” inheritance issue is particularly problematic where functions originating from A are overridden both in derived classes B and C. In this case, it is not clear which of the two versions of that function should take precedence in our new class D.

Examiner’s Comments

This question was attempted by 73% of candidates, but had a low success rate, with only 24% achieving a ‘pass’ mark of 40% of the available marks. This is somewhat surprising, as many of the questions that relate to basic object oriented terminology have featured in a number of earlier papers. In part (a), in some cases candidates were able explain the meaning of one of the two terms to be contrasted, attracting partial marks. In part (c), many candidates did not seem to know what multiple inheritance is, and were therefore unable to adequately explain why some languages discourage its use.

QUESTION A3

a) What are member access specifiers, and how are they represented in a UML class diagram?

(5 marks)

Answer Pointers

Member access specifiers stipulate the visibility level of class members (attributes and behaviours) to promote an optimum level of encapsulation and information hiding. Symbols are – (private), + (public) and # (protected), which prefix each member in a class diagram to specify its access level.

b) What are aggregation, composition and inheritance, and how are they represented in a UML class diagram?

(5 marks)

Answer Pointers

Aggregation, composition and inheritance are three forms of inter-class relationship that commonly feature in a UML class diagram. Aggregation is represented by an empty diamond, composition by a filled diamond, and inheritance by an empty triangle. Aggregation implies a relationship between classes wherein one relies upon the other, but does not have lifetime responsibility for it. Composition implies that one class has lifetime responsibility for another (that the composited class lives and dies with its owner), and inheritance implies a hierarchy, wherein derived classes are more specific versions of a base class (i.e., may contain additional members).

c) What are abstract and concrete classes, and how are they represented in a UML class diagram?

(5 marks)

Answer Pointers

Abstract classes are incomplete, containing at least one abstract method. They cannot be used to create objects, but are present in an inheritance hierarchy for structural purposes as a form of ‘specification’, stipulating that sub-classes must implement the missing (abstract) methods before they can be instantiated (be used to generate objects). In a UML class diagram, abstract classes can be identified by the class name being shown in *italics*, or annotating it with ‘abstract’.

d) What does it mean to say that two classes are orthogonal?

(5 marks)

Answer Pointers

Two classes are said to be orthogonal if changes to one class have no impact of the function of the other class. In other words, these classes are not dependent upon one another - one is not a hierarchical ancestor of the other, nor are they associated with one another via other relationships such as composition or aggregation. The implication is that a third class can inherit from two orthogonal base classes without risk of conflict.

e) What is a singleton pattern and when might it be used?

(5 marks)

Answer Pointers

A singleton pattern is used when we know a priori that a particular class should only given rise to a single object, no more. Such an object might be used to coordinate global activities across an application. There are various approaches to implementing a singleton pattern in code.

Examiner's Comments

This question was also attempted by roughly 73% of candidates, and tested knowledge and understanding of issues in object oriented design, focusing particularly on UML. Approximately one half of those who attempted this question scored 40% of the available marks or more. Questions (a), (b) and (c) tested basic knowledge of UML class diagrams. A number of candidates confused use-case diagramming with class diagrams. Others drew complex multi-class diagrams in response to these questions, where all that was required was a short statement, or illustration, of how each of the features stated in the question are represented. Only one or two candidates correctly explained the concept of orthogonality (d), but a relatively large number were able to define and suggest where a singleton pattern might be used (e).

SECTION B

QUESTION B4

a) The majority of programs manipulate data structures. How are data structures defined in object oriented programming languages?

(5 marks)

Answer Pointers

A data structure is a specialized format for organizing and storing data. Data structures may be characterised by the operations that can be performed on them. For example a stack is characterised by the operations *push* and *pop*. In object oriented programming languages data structures are made available to users via classes which implement the methods which characterise the data structure. The classes describe complex objects whose constituent parts may themselves be complex objects.

b) During the execution of a program, any data structures it uses must be assigned to some part of the computer's memory. Explain how this is achieved in an object oriented programming language with which you are familiar.

(5 marks)

Answer Pointers

The example given here uses Java.

The declaration:

```
MyClass myObject;
```

Allocates space for a reference to an object of type MyClass but at this point no memory is allocated to the data structure implemented by MyClass.

Memory is allocated by invoking the `new` operator:

```
myObject = new MyClass();
```

At this point the system reserves enough memory to hold an object of MyClass and places a reference to it in myObject.

c) Some data structures (such as counters) need to be initialised when they are created. Show how this initialisation can be achieved using an object oriented programming language.

(5 marks)

Answer Pointers

Constructors can be used to initialise elements of a data structure.

```
class MyCounter{

  int counter;

  MyCounter(int start){
       counter = start;
  }
}
```

The statement

```
MyCounter myObject = new MyCounter(3);
```

will set the initial value of the counter to 3.

d) Show how the mechanism you described in part c) operates when a data structure is derived from another simpler structure via single inheritance.

(5 marks)

Answer Pointers

```
class SimpleStructure{

  SimpleStructure(){
       .
       .
  }
  .
  .
}

class MoreComplexStructure extends SimpleStrucure{

  MoreComplexStructure(){
       .
       .
  }
  .
  .
}
```

If the following statement is executed:

```
MoreComplexStructure myObject = new MoreComplexStructure();
```

then the constructor of SimpleStructure is executed prior to the execution of the code in the constructor for MoreComplexStructure.

e) When a program has finished using a data structure, the memory it occupies can be returned to the pool of available memory. Describe a mechanism that achieves this without the programmer having to explicitly release memory.

(5 marks)

Answer Pointers

This mechanism is known as garbage collection. In the answer to part a) the precursor to allocating memory to a data structure was the creation of a variable to hold a reference to the object created by the new operator. In garbage collection when the system detects that there are no longer any references to an object or data structure it deallocates the memory used by that object.

Examiner's Comments

This question has been asked in previous papers and tests a knowledge of basic object oriented concepts. Despite this, on this occasion it attracted some very poor answers. Most candidates were able to identify a data structure with a class in an object oriented language. Only a few seemed to understand how objects are assigned memory and the way in which constructors can be used in object initialisation.

Question B5

a) Describe the features that differentiate object oriented programming languages from structured programming languages which do not support objects.

(10 marks)

Answer Pointers

Answers could include (but were not limited to) the following features:)

Object – the combination of data and the operations that apply to that data.

Class – the pattern for all objects which have identical data and behaviour.

Inheritance – the facility to derive new classes from existing classes.

Methods – the way in which behaviour is defined.

Messages – the mechanism for invoking behaviour.

Two marks were awarded for the name of each concept and a short description

b) You have been asked to advise the manager of an IT department on the choice of a programming language. The manager wishes to know whether the use of an object oriented programming language would increase programmer productivity. Write a report that sets out the potential benefits and disadvantages of deploying an object oriented language.

(15 marks)

Answer Pointers

The following is an example of one acceptable answer, however, many other approaches were acceptable:

There are three main ways in which object-oriented programming might improve programmer productivity:

i) By providing an environment which assists programmers to improve the quality of their code;
ii) Making it easier for programmers to reuse their own code;
iii) Providing simple mechanisms to make use of existing code libraries.

Object-oriented programming embodies practices which lead to well-constructed programs. It associates procedures with the data those procedures use. It can be used to form a clear separation between underlying data structures and functionality. The concept of object is a good abstraction mechanism that helps a designer separate out a small part of a problem and concentrate on that simpler part and as a consequence increasing the probability of that part of the design being correct. Object-oriented programming languages encourage and support object thinking.

In general the fewer lines of code a programmer has to write the more productive they will be. This can be facilitated by providing powerful features in a language (such as a matrix multiplication operator). The disadvantage of such features is that almost certainly in a bespoke environment they will not do exactly what a programmer needs. Consequently programmers will write their own code to do this. A classic example is a date checking routine. Often such routines are textually copied from one program to the next. This creates a maintenance nightmare. Object-oriented programming facilitates and encourages the use of re-usable classes which allow programmers to reuse the same code over and over again without physically copying it.

Just as programmers may reuse their own classes, they may also easily reuse classes provided by third parties. This is particularly useful where programmers have to produce complex interfaces to their programs. The inheritance mechanism allows programmers to enhance third party classes to meet their individual requirements without the need to alter the original code and therefore tailored software can be developed. The majority of code in any application will often be incorporated in this way and therefore productivity is greatly enhanced.

The disadvantages of object-oriented programming include the learning curve necessary to become proficient in it and the fact that code produced by an object-oriented language compiler is unlikely to be as efficient as code produced by the best machine code programmers.

Examiner's Comments

This question was very popular and produced a good set of answers. The first part of the question invited candidates to specify key aspects of object oriented programming languages. Most candidates could do this although slightly could correctly explain the features they identified. The second part of the question tested whether the candidates could see the practical advantages of object oriented programming as opposed to the technical details of the approach. Some candidates offered very limited answers to this part but the majority were able to cite a number of ways in which object oriented programming techniques improve programmer productivity.

QUESTION B6

a) Explain what is meant by the term *code refactoring.*

(5 marks)

Answer Pointers

“Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behaviour. Its heart is a series of small behaviour preserving transformations. Each transformation (called a 'refactoring') does little, but a sequence of transformations can produce a significant restructuring. Since each refactoring is small, it's less likely to go wrong. The system is also kept fully working after each small refactoring, reducing the chances that a system can get seriously broken during the restructuring”. Fowler (2011) http://martinfowler.com/refactoring/

b) Describe the way code refactoring might be used as part of an iterative and incremental software development process.

(10 marks)

Answer Pointers

In incremental development a system is split into a number of parts and the different parts of the system are developed at different times. Experience gained in developing the first aspects of the system to be delivered may influence the development of the later deliverables. The alternative is a big bang approach where every part of the system is delivered at the same time.

Iterative development may be used as part of an incremental development process but this is not an automatic assumption. In iterative development time is set aside to revise and improve some parts of the system. These improvements will typically be some form of code refactoring.

c) How would you test object oriented software?

(10 marks)

Answer Pointers

Testing would initially centre on individual classes. Each public method of class would require testing. The testing methodology could consist of both white and black box testing. In white box testing we test the paths through the code and provide data which exercises every possible path through the code. In black box testing, test data is developed by considering the specification of the methods. Using the specification a number of equivalence partitions are developed and data is selected from each partition and from the boundaries of the partitions. Once classes have been tested it is not necessary to retest them if their methods are inherited. It is, however, necessary to test inherited and overridden methods in the context of the new class.

Examiner’s Comments

Refactoring is explicitly included in the “Practice” section of the syllabus alongside Iterative and Incremental Development and Testing. This question was therefore effectively requiring

book work answers. Despite this, less than 50% of the candidates chose to attempt it and many of the answers were very poor. Given the straightforward nature of the question, the only explanation for the poor performance is that candidates are unfamiliar with the material relating to this section of the syllabus.