BCS HIGHER EDUCATION QUALIFICATIONS
BCS Level 5 Diploma in IT
March 2014
EXAMINERS' REPORT
Object Oriented Programming

Question A1

a) Briefly describe the following in the context of object oriented programming:

i) Class;
ii) Object.

(4 marks)

Answer Pointers

i) A class is a blueprint from which objects are created.
ii) Objects consist of state and related behaviour. An object stores its state in fields (variables in some programming languages) and exposes its behaviour through methods.

b) Describe two types of class members used in object oriented programming languages.

(6 marks)

Answer Pointers

Classes contain field and method members. Fields contain the data that describes the object. Methods describe the operations that may be applied to the object.

c) Explain how three types of class member visibility are used in object-oriented programming.

(6 marks)

Answer Pointers

Class members may be defined as public, private or protected. If a member is declared as public then it can be accessed by code in any class. The private modifier specifies that the member can only be accessed in its own class. The protected modifier specifies that the member can be accessed by a subclass of its class.

d) Using an object oriented language with which you are familiar, give an example of a class definition which illustrates the use of the concepts you described in your answer to parts b) and c).

(9 marks)

Answer Pointers

```
class MyClass
{
   public    int myPublicVar;
   private   int myPrivateVar;
   protected int myProtectedVar;

   private void myPrivateMethod()
   {
   }

   public void myPublicMethod()
   {
   }

   protected void myProtectedMethod()
   {
   }
}

class MySubclass extends MyClass
{
   private void MySubclassMethod()
   {
      myPublicVar++;
      myProtectedVar++;
   }
}

class AnotherClass
{

   public void AnotherClassMethod()
   {
      MyClass myObject = new MyClass();
      myObject.myPublicVar++;
   }
}
```

Examiner's Comments

This question examines the Concepts section of the syllabus

Almost every candidate chose to attempt this question. The topics covered were clearly familiar as the majority of candidates gained a pass mark for the question. The majority of marks were gained on parts a, b and c. In part d candidates lost marks by submitting answers that did not give examples of **all** three modifiers. Answers generally showed the use of private but did not illustrate the use of public and protected.

Question A2

a) Compare and contrast:

i) Object oriented programming;
ii) Procedural programming;
iii) Structured programming.

(15 marks)

Answer Pointers

i) Object oriented programming is a programming style in which the solution to problem is represented as a set of interrelated objects. Object in this sense are collections of related data that represent the state of the object and methods that manipulate the data.

ii) In procedural programming the programmer concentrates on the steps that must be undertaken to solve the problem rather than the data and the operations that apply to that data.

iii) Structured programming is a type of procedural programming. In structured programming a programmer uses a limited set of control structures in the construction of the program. Examples include while, if and for. The advantage of a structured approach over plain procedural programming is that it supports a top-down approach to design.

b) You have been invited to give a talk to trainee programmers outlining the reasons for the widespread use of object oriented programming within the software development industry. Summarise the points you would present in your talk.

(10 marks)

Answer Pointers

The following answer is indicative of the points that might be made in response to the question. A large number of alternatives would be equally acceptable.

There are three main ways in which object oriented programming has been found to 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 that have been known for some time to 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 consequently 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 that 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 examines the Foundations section of the syllabus

Just under 72% of the candidates chose to answer this question. Over 80% gained a pass mark in the question. There were many good answers to part a) as this was largely book-work. Answers to part b) were much more variable. Some candidates simply set out the features of an object oriented programming language. To obtain full marks an answer needed to explain how these features could be used to advantage when producing software.

Question A3

a) Give the meaning of the following terms:

i) Subclass;
ii) Superclass;
iii) Inheritance;
iv) Dynamic binding;
v) Delegation.

(10 marks)

Answer pointers

i) A class that is derived from another class is called a subclass.
ii) The class from which the subclass is derived is called a superclass.
iii) Classes can be derived from other classes, thereby inheriting fields and methods from those classes.
iv) Dynamic Binding refers to linking a procedure call to the code that will be executed only at run time. The code associated with the procedure is not known until the program is executed.
v) Delegation is where an object, instead of performing one of its stated tasks, delegates that task to an associated helper object.

b) Use an object oriented language with which you are familiar to give an example of delegation.

(10 marks)

Answer Pointers

```
public interface ISoundBehaviour
{

   public void makeSound();
}

public class MeowSound implements ISoundBehaviour
{

   public void makeSound()
   {
      System.out.println("Meow");
   }
}

public class RoarSound implements ISoundBehaviour
{

   public void makeSound()
   {
      System.out.println("Roar!");
   }
}
public class Cat
{

   private ISoundBehaviour sound = new MeowSound();

   public void makeSound()
   {
      sound.makeSound();
   }

   public void setSoundBehaviour(ISoundBehaviour newsound)
   {
      sound = newsound;
   }
}

public class Main
{

   public static void main(String[] args)
   {
      Cat c = new Cat();
      // Delegation
      c.makeSound();          // Output: Meow
      // change the sound it makes
      ISoundBehaviour newsound = new RoarSound();
      c.setSoundBehaviour(newsound);
      // Delegation
      c.makeSound();          // Output: Roar!
   }
}
```

c) Compare and contrast delegation with inheritance.

(5 marks)

Answer pointers

See http://www.jguru.com/faq/view.jsp?EID=27916

Inheritance is a relationship between two classes where one class, called a subclass in this context, inherits the attributes and operations of another class, called its superclass

The primary advantages of inheritance are that it is directly supported by object-oriented languages, and provides the context for polymorphism in strongly typed object-oriented languages such as C++ and Java.

But since the inheritance relationship is defined at compile-time, a class can't change its behaviour dynamically during program execution. Moreover, modifications to a superclass automatically propagate to the subclass, making software maintenance and reuse both easier and sometimes more difficult. In summary, inheritance creates a strong, static coupling between a superclass and its subclasses.

Delegation can be viewed as a relationship between objects where one object forwards certain method calls to another object, called its delegate. Delegation can also be a powerful design/reuse technique. The primary advantage of delegation is run-time flexibility – the delegate can easily be changed at run-time. But unlike inheritance, delegation is not directly supported by most popular object-oriented languages, and it doesn't facilitate dynamic polymorphism.

Examiner's Comments

This question examines the Concepts section of the syllabus

This was the least popular question of the paper with less than 30% of the candidates attempting it. Of those who did attempt the question most were only able to answer i), ii) and iii) of part a. There were very few good answers to parts b and c. Clearly candidates were not familiar with the concept of delegation or how it is used in object oriented programming. It is explicitly included in the syllabus for the paper and therefore should be included in courses preparing candidates for the examination.

Question B4

Consider the following class definition that represents a thermostatically controlled water heater.

```
public class thermostat
{
  public int  temperatureLimit;   // from 0 to 30 (celsius)
  public bool currentHeaterState; // true or false (for on/off)
  public int  evaluateState();    // switch on/off as required
  public int  getCurrentTemp();   // obtain temperature from sensor
};
```

a) Provide a redesigned **thermostat** class that uses more appropriate access modifiers.

(5 marks)

b) Provide a getter and a setter method that will enable the **temperatureLimit** instance variable to be retrieved and modified.

(5 marks)

c) Provide a constructor for the **thermostat** class that will initialise the instance variables to suitable (valid) start values.

(5 marks)

d) Write a body for the **evaluateState()** method that enables it to switch on and off the heater (by changing the **currentHeaterState** instance variable) by comparing the current value of **temperatureLimit** with the current temperature, as returned by **getCurrentTemp()** method.

(5 marks)

e) Write a short test harness that instantiates the **thermostat** class and demonstrates that the setter methods you designed above behave correctly by using a set of appropriate boundary tests.

(5 marks)

Answer Pointers

a)

```
class thermostaticHeatingSystem
{
  private:
    int  temperatureLimit;
    bool heaterState;
  public:
    void evaluateState();
    int  getCurrentTemperature();
    bool getCurrentHeaterState();
}
```

(5 marks)

b)

```
bool thermostaticHeatingSystem::setTemperatureLimit(int newTemperatureLimit)
{
  if(newTemperatureLimit >=0 && newTemperatureLimit <=30)
  {
    temperatureLimit = newTemperatureLimit;
    return true;
  }
  return false;
}

int thermostaticHeatingSystem::getTemperatureLimit()
{
  return temperatureLimit;
}
```

(5 marks)

c)

```
public thermstatciHeatherSystem:: thermstatciHeatherSystem()
{
  temperatureLimit = 0;
  heaterState = false;
}
```

(5 marks)

d)

```
public void evaluateState()
{
  if(getCurrentTemperature()<temperatureLimit)
    heaterState = true;
  else
    heaterState = false;
}
```

(5 marks)

e)

```
int main(void)
{
  thermostaticHeaterSystem ths;

  ths.setTemperatureLimit(-1);
  cout << ths.getTemperatureLimit();

  ths.setTemperatureLimit(0);
  cout << ths.getTemperatureLimit();

  ths.setTemperatureLimit(30);
  cout << ths.getTemperatureLimit();

  ths.setTemperatureLimit(31);
  cout << ths.getTemperatureLimit();
}
```

Output should be 0, 0, 30, 30.

(5 marks)

Examiner's Comments

This question examines Syllabus 8D: Practice

Less than half of the candidates that sat this exam attempted this question. Of those that did attempt the question, 62% answered it successfully (scored > 40% of the available marks), with an average of 12 marks from the available 25 being scored. Deficiencies in candidate's answers typically indicated a lack of familiarity with the basic syntax of an object oriented programming language (such as C++ or Java), potentially coupled with a poor understanding of the terminology used in the question. Parts d and e, that required the manipulation and testing of an object of the class, were particularly poorly answered, but most candidates were able to provide set/get methods and a constructor for the class.

Question B5

a) State the UML symbol used to represent the following class member visibility levels:

(i) protected
(ii) derived
(iii) private
(iv) static
(v) public

(5 marks)

b) State what the following inter-class relationships symbols represent in UML:

(i) filled diamond
(ii) empty triangle
(iii) empty diamond
(iv) line
(v) dotted line

(5 marks)

c) State how the following inter-class relationship multiplicity levels would be represented in a UML class diagram:

(i) 1 to many
(ii) many to many
(iii) 0 to 1
(iv) 0 to many
(v) exactly 2

(5 marks)

d) Distinguish between abstract and concrete methods, provide an example of a situation in which each would be used, and show how each would be represented in UML class diagram.

(10 marks)

Answer Pointers

a)

(i) #
(ii) –
(iii) _
(iv) /
(v) +.

(5 marks)

b)

(i) composition
(ii) aggregation
(iii) generalization
(iv) association
(v) dependency

(5 marks)

c)

(i) 1..*
(ii) 1
(iii) 0..1
(iv) 0..*
(v) 2

(5 marks)

d)

Abstract methods are incomplete at the level of the class hierarchy in which they are defined, and are present in a super-class merely to specify that implementation must be completed before the class can be instantiated. Abstract methods feature in abstract classes. An example is a Shape abstract super-class with an abstract method draw that is deferred until concrete subclasses Circle and Square. In a UML class diagram, abstract classes (and the abstract methods therein) are shown in italics.

Shape

string colour

setColor(string colour)
draw()

Circle	Square
int radius;	int width, height

setRadius (int newRadius)
draw()
	setSize(int newWidth, int newHeight)
	draw()

(10 marks)

Examiner's Comments

This question examines Syllabus 8C: Design

88% of candidates attempted this question, making it the second most popular question in this year's paper. Candidates scored well, with 80% achieving 40% of the available marks or more. On average, 14 of the available 25 marks were scored. Questions a-c, which required that candidates were familiar with UML notation, were answered well. Question d, requiring that an example of a class hierarchy that entailed both concrete and abstract classes, was provided was answered rather less well. Most candidates were able to describe how abstract and concrete classes were represented in UML, but fewer were able to provide an example.

Question B6

a) Distinguish between class and instance variables.

(5 marks)

b) Distinguish between aggregation and composition.

(5 marks)

c) Distinguish between static and dynamic data structures.

(5 marks)

d) Distinguish between inherited and compiled languages.

(5 marks)

e) Distinguish between class and instance methods.

(5 marks)

Answer Pointers

a) A set of instance variables exist for each new object that is created from a class definition, such that each object's instance variables may assume unique values. Conversely, class variables, which are typically declared using the static keyword, belong to the class and are not duplicated for each object created. Consequently, one might consider class variables to be shared between all instances of a class, if any instances exist.

(5 marks)

b) Aggregation represents a "has-a" relationship between classes, wherein, unlike composition, both classes may also an independent existence. For example, patient has-a doctor, but both patient and doctor may still sensibly exist without each other. Composition represents an "owns-a" relationship between classes. Unlike aggregation, the subservient class does not have an existence independently of the class that it is owned by. For example, a linked list node may be owned by a linked list, but may not exist without the linked list that it is contained within.

(5 marks)

c) In an interpreted language, the code you write is saved as source code, and executed by conversion to machine code at run-time. The interpreter must be present to enable this. Conversely, in a compiled language, source code is converted into machine code program before it can run, and can thereafter be run without the presence of a development environment/compiler.

(5 marks)

d) Ad-hoc polymorphism refers to the scenario in which code must be provided to explicitly deal with each different data type that the code may encounter (e.g., method overloading). Conversely, in parametric polymorphism, code can be written that can operate upon any data type, dynamically adapting its behaviour without the requirement for specific, detailed instructions that describe how to deal with each individual data type that may be encountered (e.g., template classes). This is sometimes referred to as generic programming.

(5 marks)

e) A virtual method is a concrete super-class method that can be replaced in a sub-class with a new implementation (viz., overriding). A pure virtual method, by contrast, is a method signature in a super-class that has no corresponding implementation within that class (i.e., it is abstract), and therefore must be implemented in any sub-classes created that the programmer wishes to become concrete (i.e., instantiable).

(5 marks)

Examiner's Comments

This question examines Syllabus 8D: Practice

75% of candidates attempted this question, of which only 46% scored 40% or more of the available marks. The average mark was only 9 from 25, indicating a relatively widespread lack of familiarity with object oriented concepts. Furthermore, several of the concepts addressed in this paper have appeared in previous papers indicating a lack of adequate preparation for the exam (i.e., carefully reviewing previous exams).