BCS HIGHER EDUCATION QUALIFICATIONS
BCS Level 5 Diploma in IT

April 2011

EXAMINERS' REPORT

Object Oriented Programming

General Comments

There were a number of candidates this year who received less than ten marks for the paper. This indicates that they were completely unprepared for the examination. Candidates should recognise that the standard of the paper is equivalent to that expected from students on the second year of a UK honours degree and that it is unlikely that anyone could expect to succeed without either attending an appropriate course or undertaking detailed reading of the recommended texts.
Once again the examiners would like to ask future candidates to clearly mark their choice of questions on the front of their answer books.

SECTION A

QUESTION A1

a) For each of the programming paradigms described below provide a phrase which describes that paradigm and brief notes which explain how programmers develop code in a language that supports the paradigm.

i) A programming style in which solutions take the form of a set of interrelated objects;
ii) A programming style which emphasises the steps taken to reach a solution;
iii) A programming style based on a limited set of control structures.

(15 marks)

Answer Pointers

i) Object oriented programming

Programmers identify objects (data and the associated operations on that data) in the domain their application is trying to model. They then consider the interactions between objects required to fulfil the requirements of the task

ii) Procedural Programming

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

iii) Structured 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) Compare and contrast the three programming approaches outlined above setting out the advantages and disadvantages of each approach.

(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.

Procedural programming concentrates on the steps necessary to produce the required solution and therefore tends to neglect the nature of the data which has to be manipulated. The data structures used in procedural programming tend to be primitive eg. arrays and are not closely aligned with the problem area. It therefore becomes easy to attempt operations on data which do not correspond to those which are legal in the problem domain. For example, if a stack is implemented as an array then there is nothing to prevent the programmer accidently taking something from the middle of a stack instead of limiting access to pop operations. Where a procedural programming language is not structured it is easy for the programmer to leave data in an illegal state by jumping out of a loop.

Structured programming exhibits the same problem as procedural programming with regard to data. Again complex data structures are often implemented on top of simple data structures in a manner which permits operation that would be illegal in the context of the application are. The advantage of structured programming is that the control structures in the language make it easier for the programmer to localise where in the program given tasks take place.

Most object oriented program languages use a structured programming language approach when defining methods. In addition they enable programmers to define the valid operations on a given data structure. Features within object oriented programming languages prevent programmers from accessing data by bypassing these operations. This makes it easier to ensure that programs behave as they were intended. In addition, the ability to create independent objects means that the opportunity for code reuse increases.

Examiner's Comments

This question was attempted by well over half of the candidates sitting the paper. In general, the answers were acceptable with very few good solutions presented. The topics covered have appeared on a number of past papers and therefore it was reasonable to expect candidates to be familiar with the material. Many candidates were able to answer part a) but did less well in part b) indicating that they were able to quote and remember definitions but not able to discuss the consequences of the use of programming techniques.

QUESTION A2

a) Give a short phrase which describes each of the following object oriented concepts. For each phrase provide an example of code which implements the concept.

   i) The prototype of a method, which enables the programmer to identify the purpose of a method (via its name), the data it requires to operate (via its argument list), and any end result that is produced (its return type);
   ii) The use 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;
   iii) A mechanism which allows a subclass to provide its own implementation of a method already provided by one of its superclasses.

(12 marks)

Answer Pointers

i) This is known as a signature
For example:

```
intmethodName (String x){
}
```

defines a method called methodName which accepts a parameter of type String and returns an integer value.

ii) This is known as method overloading

For example

```
voiddoIt (){
}

voiddoIt (int x){
}
```

The second method overloads the first. Both have the same name but one accepts an argument and the other does not.
iii) This is called method overriding

For example:

```
classMyClass {
        protectedint a;
        public void increment() {
                a = a + 1;
        }
}

classMySubclassextends MyClass{

        public void increment() {
                a = a + 2;
        }
}
```

b) Programmers who use object oriented programming languages frequently design programs in which objects that are instantiated from different classes respond to identical messages. Explain why this is an important feature of object-oriented programming and give an example of how it can be achieved in code.

(13 marks)

This is known as polymorphism
In strongly typed languages, polymorphism usually means that type A somehow derives from type B, or type A implements an interface that represents type B
The primary usage of polymorphism is the creation of objects belonging to different types to respond to method, field, or property calls of the same name, each one according to an appropriate type-specific behaviour. The programmer (and the program) does not have to know the exact type of the object in advance, and so the exact behaviour is determined at run time

```
public class Feline {
voidmakeNoise() {
System.out.println("Non-specific cat sound");
    }
}

public class DomesticCat extends Feline {
voidmakeNoise() {
System.out.println("Meow Meow");
    }
}

public class Lion extends Feline {
voidmakeNoise() {
System.out.println("Roar! Roar!");
    }
}

public class Test {
public static void main(String[] args) {
        Feline f1 = new DomesticCat();
        Feline f2 = new Lion();
f1.makeNoise();
f2.makeNoise();
    }
}
```

Here f1 and f2 both reference Felines but the makeNoise method gives different outputs.

Examiner's Comments

This question was attempted by over two thirds of the candidates; however the majority of answers were quite poor. The topic addressed in part a) has appeared on many previous papers, however, rather than requiring a definition from the candidate, a definition was given and the candidate was asked to name the concept and give an example of its use. It was pleasing to see that some candidates did quote an example of polymorphism given in a previous OOP examiner's report as this indicates that they are benefitting from looking at previous papers.

QUESTION A3

a) Give the object oriented terminology for each of the following object oriented features and supply an example of code that illustrates the feature:

i) A blueprint for an object which defines all the data items contained in the object and the operations that are permitted for the data;
ii) A representation of something within the domain that the program models that contains values of data and which implements operations on that data;
iii) An operation which will manipulate the data contained in an object**;**
iv) A variable which holds data that describes an individual object;
v) A variable which holds data that is relevant to all the objects created from the same template.

(15 marks)

Answer Pointers

i) This is known as a class

```
classMyClass {
        privateint a;
        private static int b;
        public void increment() {
                a = a + 1;
        }
}
```

ii) This is an object

```
MyClassanObject = new MyClass();
```

iii) This is known as a method

```
public void increment() {
        a = a + 1;
}
```

iv) This is known as an instance variable. It can only be referenced within the context of an object

```
classMyClass {
                privateint a;
                private static int b;
                public void increment()
                {
                        a = a + 1;
                }
                }
```

a is an instance variable

v) This is known as a class variable

```
classMyClass {
                privateint a;
                private static int b;
                public void increment() {
                        a = a + 1;
                }
        }
```

b is a class variable

b) Programmers who use object oriented languages frequently reuse code which has been developed by themselves or other programmers. Describe the primary technique which makes this possible and give a code example of it in use.

Answer Pointers

Inheritance is a mechanism used to create new classes which are based on existing classes. The class created is called a subclass whilst the class on which the new class is based is called a superclass. In inheritance, the subclass introduces new data above and beyond that held in the superclass and/or new methods in addition to those implemented in the superclass. The major use of inheritance is to permit incremental development. In this type of development existing classes are reused and new functionality is provided by building classes which inherit from them.

```
classMyClass {
        protectedint a;
        public void increment() {
                a = a + 1;
        }
}

classMySubclassextends MyClass{

        public void decrement() {
                a = a - 1;
        }
}
```

The class MyClass has a method which increments the instance variable a. If we wish to create a class which will both increment and decrement the variable we do not need to copy the code which increments the variable. We simply introduce a class which inherits the functionality from the existing class and adds additional code to implement the new function.

Examiners' Comments

As with question two, the topics covered by this question have been addressed by many previous OOP papers. Well over half of the candidates chose to answer this question but the standard of answers was poor. In general part a) attracted better answers than part b).

SECTION B

QUESTION B4

a) Describe three distinct practical examples of polymorphism, providing illustrative code fragments that show how each is used.

(15 marks)

Answer Pointers

The candidate may choose to describe any three practical examples of polymorphism with which they are familiar, which could include one or more of the following: function/method overloading, function/method overriding (including constructor overloading), dynamic binding, compile time polymorphism (e.g. templates), default arguments, and operating overloading. A maximum of 5 marks are available for each example. Examples of the first 3 are provided below:

Function Overloading (5 marks):
Function overloading is the technique of having multiple occurrences of a function with a common name, but with different arguments. A specific version of the function is selected such that it matches the arguments used at invocation. For example:

```
class A
{
voidb(int n)    {cout <<n;}  // version 1
voidb(double n) {cout <<n;}  // version 2
};

int main(void)
{
  A mya;
a.b(10);  // invokes version 1
a.b(6.5); // invokes version 2
}
```

Function Overriding (5 marks):
Function overriding is the technique of creating a new implementation of a base class function in a derived class. The derived class function must share the same name and argument list as the base class function for overriding to occur, otherwise the derived class function willsimply be added to the set of functions available. For example:

```
class A
{
virtual void b()  { cout<< “message 1”;}
};

class B: public A
{
voidb()          { cout << “message 2” };
}

intmain(void)
{
  A mya;
```

```
mya.b();
  B myb;
myb.b();
}
```

Dynamic (Late) Binding (5 marks):

Dynamic binding enables us to defer the exact behaviour of a code fragment until run-time, and exploits the substitutability principle. It is implemented using a pointer to a base class type that may also be used to point to classes derived from that base class. When invoking a member function originating from the base class that is overridden in the derived class, the behaviour of that function will correspond to the specific derived class type pointed to at the particular time that the function was invoked.

```
class Event
{
public:
virtual void handleEvent() {};
};

class specialEvent1: public Event
{
public:
voidhandleEvent() { cout<< “type 1\t”; }
};

class specialEvent2: public Event
{
public:
voidhandleEvent() { cout<< “type 2\t”; }
};

int main(void)
{
specialEvent1 a;
specialEvent2 b;
  Event* event[2];
event[0]=&a;
event[1]=&b;
for(int i=0 ; i<2 ; i++)
event[i] ->handleEvent();
}
```

Examiners Comments

Most candidates who attempted this question provided a reasonable example of function (method) overloading, but were unable to provide examples of any other forms of polymorphism, thereby failing to score two thirds of the available marks.

b) Distinguish between ad-hoc and parametric polymorphism.

(5 marks)

Answer Pointers

Ad-hoc polymorphism refers to the scenario in which polymorphic code must be explicitly provided by the programmer to account for each different data type that the code may encounter (e.g., function 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).

Examiners Comments

This question was poorly answered, with very few candidates being able to correctly distinguish these concepts, despite a similar question having appeared in a previous exam paper. One or two candidates who did seem partly familiar with these concepts mixed up the two definitions, accruing partial marks.

c) Describe any common dynamic data structure you are familiar within object-oriented terms.

(5 marks)

Answer Pointers

Composition in object-oriented design emphasises i) the whole has exclusive ownership of the parts b) the parts have no existence outside the whole, and c) the parts live and die with the whole. The most likely data structure the student would be familiar with is either a linked list or a binary tree. Candidates should discuss whether or not such a data structure truly reflects composition – An unambiguous coding implementation would include an inner class node embedded within an outer class data structure, but there may be different ways of constructing the data structure, including pointers from one node to the next. All answers should however describe the concept of a node which contains reference(s) to another node, which are created and deleted as needed allowing the structure to expand and contract in size.

Examiners Comments

Most candidates attempting this question did not seem to know what a data structure was, so were unable to score any marks. The term data structure seemed to be most commonly confused with “data member”. Of those that were able to identify a common data structure, very few were able to consider it in object oriented terms. Since data structures are a fundamental part of most programming languages, it is suggested that this concept is explicitly studied in the context of object oriented programming.

QUESTION B5

a) Describe three practical techniques that enable classes to be interconnected and provide code fragments that show how each is used.

(15 marks)

Answer Pointers

The candidate may elect to demonstrate inheritance, composition (via ownership) and aggregation (via pointers or references). Example code fragments are provided below:

Inheritance (5 marks):

```
class A {};
class B: public A {};
```

Composition (5 marks):

```
class A {};
class B { A composited_A; };
```

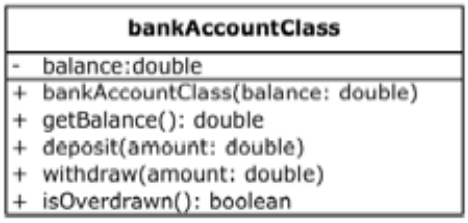
Aggregation via Pointer (5 marks):

```
class A {};
class B
{
  A* pointer_to_A;
B(A anA)        { pointer_to_A = &anA;}
};
```

Examiners Comments

Most candidates were able to describe and show an example of an inheritance relationship, but few were able to identify other forms of inter-class relationship or provide code examples to illustrate their answer.

b) Write code to implement the class diagram shown below. Supply a main() function that instantiates an object of the class and makes appropriate use of each of the functions.



(10 marks)

Answer Pointers

A C++ implementation is provided below. The student scores 2 marks for each of the 5 functions they are to implement.

```
classbankAccountClass
{
```

```
private:
double balance;
public:
bankAccountClass(double bal) { balance = bal;       }
doublegetBalance()          { return balance;  }
void deposit(double amount)  { balance+=amount; }
void withdraw(double amount) { balance-=amount; }
boolisOverdrawn()
    {
if(balance<0) return true;
return false;
    }

int main(void)
{
bankAccountClass b(25.0);
cout<<b.getBalance();
b.deposit(50.0);
b.withdraw(80.0);
cout<<b.getBalance();
if(b.isOverdrawn())
  {
cout<< "overdrawn!";
  }
return 0;
}
```

Examiners Comments

Surprisingly, quite a large number of candidates could not produce valid code to implement this simple class. Several students could not identify methods and data, and many strayed from the specified design and did not accrue available marks. However, several more competent candidates produced excellent answers to this question. It seems that some candidates that scored reasonably well lost marks by not noticing the return types of the functions, writing functions that displayed to screen instead of returning their result.

QUESTION B6

a) Distinguish between accessor and mutator functions, providing an example class definition containing one example of each.

(7 marks)

Answer Pointers

Accessor methods are used to retrieve data member values from within an object, commonly known as getters (1 mark). Mutator methods are used to change the value of data member, and are commonly known as setters (1 mark). Mutator methods therefore accept arguments (reflecting the data member that they are destined to update), whereas accessor methods usually possess a return type reflecting the data member that they are to convey back to the caller (1 mark). Both are typically designated as public to facilitate their invocation through an object (1 mark). An example class containing a data member x, with accessorgetx() and mutatorsetx(), is provided below:

```
class A                                   (1 mark for class definition)
{
private:
int x;
public:
intgetx() { return x; }                   (1 mark for correct getter)
voidsetx(intnewx) { x = newx; }           (1 mark for correct setter)
};
```

Examiners Comments

Most candidates who attempted this question answered it correctly, but some seemed to go rather overboard on their example code. A very simple scenario like the one illustrated above would have sufficed.

b) Briefly explain why a programmer may choose to designate class members as either private, public or protected within a single class.

(6 marks)

Answer Pointers

Within a single class, data members are typically designated as private (1 mark). Private data members must be manipulated via public member functions, and cannot be manipulated directly through an object (1 mark). This enables control to be kept over the validity of the data held, since mutator functions should intercept and reject inappropriate changes (1 mark). Member functions may be designated as public (service methods) or private (support methods), depending upon whether they have an internal 'helper' role, such that they may only be called by other functions in the same class (1 mark), or provide a service that can be invoked through an object by the user o the class (1 mark). Within a single class, private and protected behave exactly the same (1 mark).

Examiners Comments

Many candidates did not answer the question posed, instead discussing issues of inheritance. The question was to specify why, in a single class, a programmer may designated class members as private, public, or protected, so the answer should have discussed issues of information hiding, service/support methods, and so on.

c) Determine class member visibility for each variable in the class hierarchy provided in Appendix A. Present your answer (in your answer booklet) in the form of a table like the one shown below, using the following symbols: private (-), protected (#), public (+), not visible (blank).

(12 marks)

Answer Pointers

1 mark was awarded for each data member with a correct profile of visibility across the 4 classes.

		CLASS			
		C1	C2	C3	C4
DATA MEMBER	a	-			
	b	#	#	#	-
	c	+	#	#	-
	d		-		
	e		#	#	-
	f		+	+	-
	g			-	
	h			#	-
	i			+	-
	j				-
	k				#
	l				+

Examiners Comments

This was the most popular question from Section B, but it was one of the most poorly answered. Most candidates were able to correctly specify the visibilities of class C1, but did not understand the mechanisms for public, private and protected inheritance, leading to a multitude of errors in designating the members and visibilities of classes C2-C4. Presumably this stems from a lack of familiarity with reading code fragments, and it is suggested that candidates must commit to extensive programming practice in common languages (such as Java and C++) in addition to learning object oriented concepts before tackling this exam, since it is object oriented programming, not object oriented design.