BCS HIGHER EDUCATION QUALIFICATIONS
BCS Level 5 Diploma in IT
March 2015

EXAMINERS' REPORT
Object Oriented Programming

Section A

A1.

a) Define the following terms:

i) Abstract data type;
ii) Collection class;
iii) Class library. **(6 marks)**

Answer Pointers

Abstract data types or ADTs are data types described in terms of the operations they support—their interface—rather than how they are implemented

A Collection class is an OOP replacement for the traditional array data structure. Like an array, a collection contains member elements, although these tend to be objects rather than simpler types such as strings and integers.

A class library is a collection of prewritten classes or coded templates, any of which can be specified and used by a programmer when developing an application program.

b) A queue is a first in, first out linear data structure. A queue can have any object as an element. It is characterised by two fundamental operations, called *enqueue* and *dequeue*. The *enqueue* operation adds a new item to the back of the queue. If the space allocated to hold the queue is full when the *enqueue* operation is attempted, then an error condition is raised. The *dequeue* operation removes an item from the front of the queue. A *dequeue* moves previously added items towards the front of the queue, or results in an empty queue. If the queue is empty when a *dequeue* operation is attempted, then an error condition is raised.
Using an object oriented programming language with which you are familiar, write code which implements a queue. Your code should store the queue elements in an array and should not make use of a queue class from a class library. **(19 marks)**

Answer Pointers

```
public class Queue {

    private Object[] queueArray;
    private int maxSlot;
    private int back = 0;

    public Queue(int size){
        maxSlot = size;
        queueArray = new Object[size];
    }

    public void enqueue(Object o) throws OverflowException {
        if (back == maxSlot) {
            throw new OverflowException();
        }
        queueArray[back] = o;
        back++;
    }

    public Object dequeue() throws UnderflowException {
        Object result;
        if (back == 0) {
            throw new UnderflowException();
        }
        result = queueArray[0];
        for(int i=0; i<back-1; i++ ){
            queueArray[i] = queueArray[i+1];
        }
        back--;
        return result;
    }
}
```

Examiner's Comments

This question examines the Foundations aspect of the syllabus

This question was by far the least popular on the paper and was only answered by 30% of the candidates. Coincidently, it was also the least well answered question on the paper. The majority of candidates who attempted the question were able to make a reasonable attempt at part a) although very few obtained full marks. There is evidence that many candidates were not sure what was meant by “Collection Class” even though this term appears in the syllabus. Very few candidates were able to make a realistic attempt at part b). This part of the question was designed to allow candidates to demonstrate that they could implement an abstract data type in an object oriented language.

A2.

a) Explain what is meant by a design pattern. **(5 marks)**

Answer Pointers

A general reusable solution to a commonly occurring problem within a given context in software design. Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

b) Describe five characteristics you could use to document a design pattern. **(10 marks)**

Motivation: A scenario in which the design pattern can be used. Prerequisites: The context in which the design pattern should be used. Structure: A class or interaction diagram which illustrates the pattern. Participants: A list of the classes and objects used in the pattern. Consequences: A description of the results and side effects of using the pattern

c) Give an example of code which uses the iterator pattern. **(10 marks)**

Answer Pointers

```
public class ArrayIterator {

    private Object[] data;
    private int index = 0;

    public ArrayIterator(Object[] d) {
        data = d;
    }

    public boolean hasNext() {
        return (index < data.length);
    }

    public Object next() {
        return data[index++];
    }
}


public class IteratorExample {
    public static void main(String[] args) {
        String[] stringArray = {"Apple", "Pear", "Orange"};
        ArrayIterator arrayIt = new ArrayIterator(stringArray);
        while(arrayIt.hasNext()){
            System.out.println(arrayIt.next());
        }
    }
}
```

Examiner's Comments

This question examines the Design aspect of the syllabus

The majority of candidates were able to explain what a design pattern is and hence gained full marks for the first part of the question. The syllabus sets out a list of headings which may be used for the documentation of design patterns, however, there is evidence that not all the candidates were aware of this topic and therefore part b) was less well answered. The syllabus also sets out a list of design patterns that candidates are expected to know. Iterator is included in the list. Despite this, only a few candidates provided code which implemented the iterator pattern. Many did not attempt part c).

A3.

a) Explain the following terms:

(i) Class;
(ii) Object;
(iii) Class variable;
(iv) Method overloading;
(v) Constructor. **(10 marks)**

Answer Pointers

A class is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behaviour (member functions, methods).

An object is a particular instance of a class where the object can be a combination of variables, functions, and data structures.

A class variable is a variable defined in a class (i.e. a member variable) of which a single copy exists, regardless of how many instances of the class exist.

Overloaded methods are those that have the same name, but that have different formal parameter types (or result value type, if the language supports overloading on result type).

A constructor in a class is a special type of method called to initialise an object. It prepares the new object for use, often accepting arguments that the constructor uses to set the value of member variables

b) Explain why it is sometimes useful to overload a constructor**.** **(5 marks)**

Answer Pointers

The creation of an object may involve the setting of a number of instance and class variables. These may have a number of sensible defaults. A default constructor can be used to set all the defaults. Where the defaults are not suitable an overloaded constructor can set the appropriate values according to the arguments passed to it. There can be a series of constructors each setting a subset of the instance and class variables each of which will allow some defaults to stand.

c) Using an object oriented programming language with which you are familiar write a class which contains a method called getNumberOfInstances. This method should return the number of instances of the class. **(10 marks)**

Answer Pointers

```
public class CountInstances {

    private static int noOfInstances = 0;

    static int getNumberOfInstances() {
        return noOfInstances;
    }

    CountInstances() {
        noOfInstances++;
    }

    protected void finalize() throws Throwable {
        noOfInstances--;
    }
}
```

Examiner's Comments

This question examines the Concepts aspect of the syllabus

This was the most popular question on the paper and was answered by almost all the candidates. Part a) received a number of good answers. Candidates who lost marks tended to be unable to explain what was meant by a class variable. This meant that they struggled in part c) of the question. Part b) attracted less correct answers than part a). Although candidates knew that methods could be overloaded and what this means they were often unable to give a good reason for overloading a constructor. As with all the questions in Section A which asked for code, part c) was poorly answered.

Section B

B4. Consider the following class definition that represents an AM radio:

```
public class radio
{
  public bool isSwitchedOn;          // either true or false
  public int volume;                 // from 0 to 10
  public double frequency;           // from 535 to 1605 Mhz
};
```

a) Provide a redesigned **radio** class that uses more appropriate member access operators. **(5 marks)**

Answer Pointers:

All answers to question B4 are given in C++, but Java and other common object-oriented languages would also have been accepted.

```
public class radio
{
  private int volume;
  private double frequency;
}
```

b) Provide signatures for two mutator (setter) methods that will enable the two instance variables to be changed through an object of the class. **(5 marks)**

Answer Pointers:

Any two of the following:

```
bool setVolume(int newVolume);
bool setFrequency(double newFrequency);
bool setOnOff(bool newState);
```

c) Write bodies for the mutator (setter) methods for which you provided signatures in question (b), above. The implemented methods must prevent the two instance variables from being set to invalid values. Return a boolean flag to indicate the success/failure of the requested operation. **(5 marks)**

Answer Pointers:

Any two of the following:

```
bool setVolume(int newVolume)
{
  if(newVolume>-1 && newVolume<11)
  {
    volume = newVolume;
    return true;
  }
  return false;
}

bool setFrequency(double newFrequency)
{
  if(newFrequency>534 && newFrequency<1606)
  {
    frequency = newFrequency;
    return true;
  }
  return false;
}

bool setOnOff(bool newState)
{
  isSwitchedOn = newState;
  return isSwitchedOn;
}
```

d) Provide a constructor for the **radio** class that will initialise the instance variables to suitable (valid) start values. **(5 marks)**

```
radio::radio()
```

```
{
  volume = 0;
  frequency = 535;
}
```

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

(5 marks)

Answer Pointers:

```
int main(void)
{
  radio r;

  if(r.setVolume(-1) || r.setVolume(11))
    cout << "Volume Failed on Invalid Boundaries";

  if(r.setFrequency(534) || r.setFrequency(1606))
    cout << "Frequency Failed on Invalid Boundaries";

  if(! (r.setVolume(0) && r.setVolume(10))
    cout << "Volume Failed on Valid Boundaries";

  if(! (r.setFrequency(535) && r.setFrequency(1605))
    cout << "Frequency Failed on Valid Boundaries";

  return 0;
}
```

Examiner's Comments:

This question examines Syllabus 8D: Practice

Answers to this question were most commonly submitted using C++ and Java. There was a tendency for candidates to write far more code than was necessary to answer the question, and this additional code was often incorrect. It is more important to answer concisely and ensure that the question is addressed specifically, than write extra code that was not requested. Some candidates did not demonstrate how to formulate a mutator function that tests the boundaries of an input and responses (true/false) to signify the success or failure of the alteration. Some practice at this is recommended. For other candidates, the final part of the question that requested a test harness to be written was weak. The mere creation and modification of an object without testing the state of that object does not really constitute testing, since it does not demonstrate that the member functions created operate correctly.

B5. The follow questions relate to class diagrams represented in the Unified Modelling Language (UML).

a) State the meaning of the following member visibility operators:

(i) #
(ii) –
(iii) _
(iv) /
(v) + **(5 marks)**

Answer Pointers:

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

b) Draw or state how the following inter-class relationships are represented:

(i) composition
(ii) aggregation
(iii) inheritance
(iv) dependency
(v) realization. **(5 marks)**

Answer Pointers:

(i) filled diamond, solid line; (ii) empty diamond, solid line; (iii) empty triangle, solid line; (iv) open arrow head, dashed line; (v) hollow triangle, dashed line.

c) State the meaning of the following inter-class relationship multiplicity indicators:

(i) 1..*
(ii) *
(iii) 0..1
(iv) 2
(v) 0..* **(5 marks)**

Answer Pointers:

(i) one or more; (ii) zero or more; (iii) zero or one; (iv) exactly two; (v) zero or more, same as ii.

d) Show how abstract and concrete classes are represented in a UML class diagram, and briefly describe an example use of these class types. **(10 marks)**

Answer Pointers:

This question examines Syllabus 8C: Design

Abstract classes are incomplete and are present as super-classes a class hierarchy to support inheritance for specification. They typically contain one or more abstract methods, which are signatures with no corresponding body. 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)	setSize(int newWidth, int newHeight)
draw()	draw();

Examiner's Comments:

This was the second most popular question on this year's examination and produced the highest average number of marks. There is evidence that candidates struggled to identify the more obscure class member identifiers (static, derived), some confused the symbols used for aggregation/composition and inheritance, and some struggled to produce a syntactically valid example of a concrete/abstract class relationship, but the question was otherwise well answered.

B6.

a) Distinguish between structural and behavioural UML diagrams, briefly describing one example of each (note: you may <u>not</u> use class diagrams as your example). **(5 marks)**

Answer Pointers:

Structural diagrams show the static aspects of a system, such as those components of a system that must be present for the system to operate. Behavioural diagrams show the dynamic aspects of the system, such as the functional behaviour of the software systems defined in the structure diagrams. An example of a structural diagram is an object diagram. An example of a behavioural diagram is a use case diagram.

b) Distinguish between multiple and multi-level inheritance. **(5 marks)**

Answer Pointers:

In multi-level inheritance, a super-class is inherited into a sub-class that itself serves as a superclass for a further sub-class, thereby forming a linear hierarchy. Conversely, in multiple inheritance, supported in languages such as C++, two super-classes may be inherited to form a sub-class simultaneously. This can produce ambiguities where both super-classes contain methods or variables with shared signatures/names.

c) Distinguish between static and dynamic binding. **(5 marks)**

Answer Pointers:

Dynamic binding enables us to defer the exact behaviour of a code fragment until run-time and exploits the principle of substitutability. It can be implemented using a reference to a super-class type that may also be used to point to sub-classes. When invoking a method originating from the super class that is overridden in the sub-class, the behaviour of that method will correspond to the specific sub-class type referenced at the particular time that the method was invoked. In static binding, on the other hand names are bound to objects at compile time, before the program is run. Typically, this leads to higher performance in method invocations.

d) Distinguish between ad-hoc and parametric polymorphism. **(5 marks)**

Answer Pointers:

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.

e) Distinguish between virtual methods and pure virtual methods. **(5 marks)**

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).

Examiner's Comments:

This question examines the Concepts section of the syllabus

This was the second least popular question in this paper, perhaps reflecting difficulty in grasping some of the less basic concepts in object oriented design and programming. Some candidates used the example of a class diagram for the first question, despite being explicitly instructed not to. There is evidence that most candidates were able to discriminate multiple from multi-level inheritance, but the answer to each of the other questions were more mixed.