BCS THE CHARTERED INSTITUTE FOR IT

BCS HIGHER EDUCATION QUALIFICATIONS
BCS Level 5 Diploma in IT

OBJECT ORIENTED PROGRAMMING

Monday 26th September 2016 - Afternoon
Answer **any** FOUR questions out of SIX. All questions carry equal marks
Time: TWO hours

Answer any Section A questions you attempt in Answer Book A
Answer any Section B questions you attempt in Answer Book B

The marks given in brackets are **indicative** of the weight given to each part of the question.

Calculators are **NOT** allowed in this examination.

SECTION A

Answer Section A questions in Answer Book A

1.

a) Explain how object oriented programming languages can be used to implement abstract data types.

(5 marks)

b) A deque (or double ended queue) is an abstract data type that generalises a queue, for which elements can be added to or removed from either the front (head) or back (tail).

A deque can have any object as its element. It is characterised by four fundamental operations: push, pop, inject and eject. The push operation adds a new item to the front of the queue. If the space allocated to hold the deque is full when the push operation is permitted then an error is raised. The pop operation removes an item from the front of the deque. A pop reveals previously concealed items, or results in an empty deque. If the deque is empty when a pop operation is attempted then an error condition is raised. The inject operation adds a new item to the rear of the deque. If the space allocated to hold the deque is full when the inject operation is attempted, then an error condition is raised. The eject operation removes an item from the rear of the deque. An eject reveals previously concealed items, or results in an empty deque. If the deque is empty when an eject 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 deque. Your code should store the deque elements in an array and should not make use of a deque class from a class library.

(20 marks)

Answer Pointers:

a) Modern object-oriented languages, such as C++ and Java, support a form of abstract data types. When a class is used as a type, it is an abstract type that refers to a hidden representation. In this model an ADT is typically implemented as a class, and each instance of the ADT is usually an object of that class. The module's interface typically declares the constructors as ordinary procedures, and most of the other ADT operations as methods of that class.

(5 marks)

b) Source code below:

```
public class Deque
{
    private final Object[] dequeArray;
    private final int maxSlot;
    private int emptySlot = 0;

    public Deque(int size)
    {
        maxSlot = size;
        dequeArray = new Object[size];
    }

    public void push(Object o) throws OverflowException
    {
        if (dequeFull())
        {
            throw new OverflowException();
        }
        dequeArray[emptySlot] = o;
        emptySlot++;
    }

    public Object pop() throws UnderflowException
    {
        if (dequeEmpty())
        {
            throw new UnderflowException();
        }
        emptySlot--;
        return dequeArray[emptySlot];
    }

    public void inject(Object o) throws OverflowException
    {
        if (dequeFull())
        {
            throw new OverflowException();
        }
        for (int i = emptySlot; i > 0; i--)
        {
            dequeArray[i] = dequeArray[i - 1];
        }
        dequeArray[0] = o;
        emptySlot++;
    }

    public Object eject() throws UnderflowException
    {
        Object result;
        if (dequeEmpty())
        {
            throw new UnderflowException();
        }
        result = dequeArray[0];
        for (int i = 1; i < emptySlot; i++)
        {
            dequeArray[i-1] = dequeArray[i];
        }
        emptySlot--;
        return result;
    }

    private boolean dequeFull()
    {
        return emptySlot == maxSlot;
    }

    private boolean dequeEmpty()
    {
        return emptySlot == 0;
    }
}
```

(20 marks)

Examiners Comments:

This question examines the Foundations section of the syllabus

Very few candidates attempted this question (only 17%), and the question had the lowest mean mark. In part (a), many candidates confused Abstract Data Type (ADT) with abstract class and/or abstract method, and incorrectly wrote a description of the latter. The difficulties candidates faced in answering part (b) stemmed both from having insufficient mastery of an object oriented programming language to organise a class correctly (i.e., to include appropriate fields and methods), along with particular difficulty in using control structures (in particular, iteration) to enable the fundamental operations proposed in the question to be implemented.

2.

a) Describe three modifiers that can be used to define the visibility of class members (fields and methods).

(6 marks)

b) Explain why it is considered good practice to limit the scope of fields and methods in object oriented programming.

(3 marks)

c) Explain the following terms:

(i) Inheritance;
(ii) Single inheritance;
(iii) Multiple inheritance.

(6 marks)

d) Using an object oriented language with which you are familiar, give an example of the use of inheritance.

(10 marks)

Answer Pointers:

a) The public modifier allows full access to code outside the class. The private modifier specifies that the member can only be accessed in its own class. The protected modifier specifies that the member can only be accessed within its own class and, in addition, by a subclass of its class.

(6 marks)

b) Those aspects which programmers who use the class should have access to should be declared public. Any other aspects that are used internally by the class should be defined as private to prevent external code putting the object into an invalid state.

(3 marks)

c) Inheritance is when an object or class is based on another object (prototypal inheritance) or class (class-based inheritance), using the same implementation (inheriting from an object or class) specifying implementation to maintain the same behaviour (realizing an interface; inheriting behaviour). Single inheritance is where subclasses inherit the features of one super class. A class acquires the properties of another class. Multiple inheritance is where one class can have more than one super class and inherit features from all parent classes.

(6 marks)

d) Source code below:

```
class Door
{
  boolean open=false;
  void open()
  {
    open=true;
  }
  void close()
  {
    open=false;
  }
  boolean amIOpen()
  {
    return open;
  }
}

class NumberedDoor extends Door
{
  int number=1;
  void setNumber(int n)
  {
    number=n;
  }
  int getNumber()
  {
    return number;
  }
}
```

(10 marks)

Examiners Comments:

This question examines the Concepts part of the syllabus

This was the most popular question in the paper, with 93% of candidates attempting it. It also scored the greatest mean mark, at 14%, reflecting the relatively high level of familiarity that candidates have with the basic concepts of inheritance. Part (d) required candidates to present a code fragment that demonstrated inheritance. This was answered rather less well. Like question 1, this seems to allude to insufficient mastery of the basics of an object oriented programming language (i.e., an overemphasis on theory, and insufficient emphasis on practice). Many of the inheritance examples submitted either had syntax errors, did not actually implement inheritance, or were very confused.

3.

a) Explain the following terms:

i) Object;
ii) Encapsulation;
iii) Fields;
iv) Method;
v) Message.

(10 marks)

b) Your local BCS branch has decided to create a website that sets out the history of programming languages. You have been asked to provide a page on object oriented programming languages. Give an outline of how you would define an object oriented programming language on this web page.

(15 marks)

Answer Pointers:

a)

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

(2 marks)

ii) Encapsulation prevents our data from unwanted accesses by binding code (methods) and data (attributes) together into a single unit called an object.

(2 marks)

iii) A field is data encapsulated within a class, also known as a data member or member variable.

(2 marks)

iv) A method, also known as a member function, implements the behaviour of a class as a code procedure, and forms the interface by which an object communicates with the outside.

(2 marks)

v) A message is the mechanism by which object communicate. Specifically, a message refers to an object by name, invokes a method, and provides input data in the form of method arguments, if required.

(2 marks)

b) Successful answers placed object oriented programming into context with other programming paradigms, including unstructured, structured/modular, and functional approaches, as well as listing the main features of object oriented programming languages and their conceptual origins and motivation. It is not possible to answer the question and appropriately *define* an object oriented programming language without providing a suitable frame of reference.

(15 marks)

Examiners Comments:

This question examines the Concepts section of the syllabus

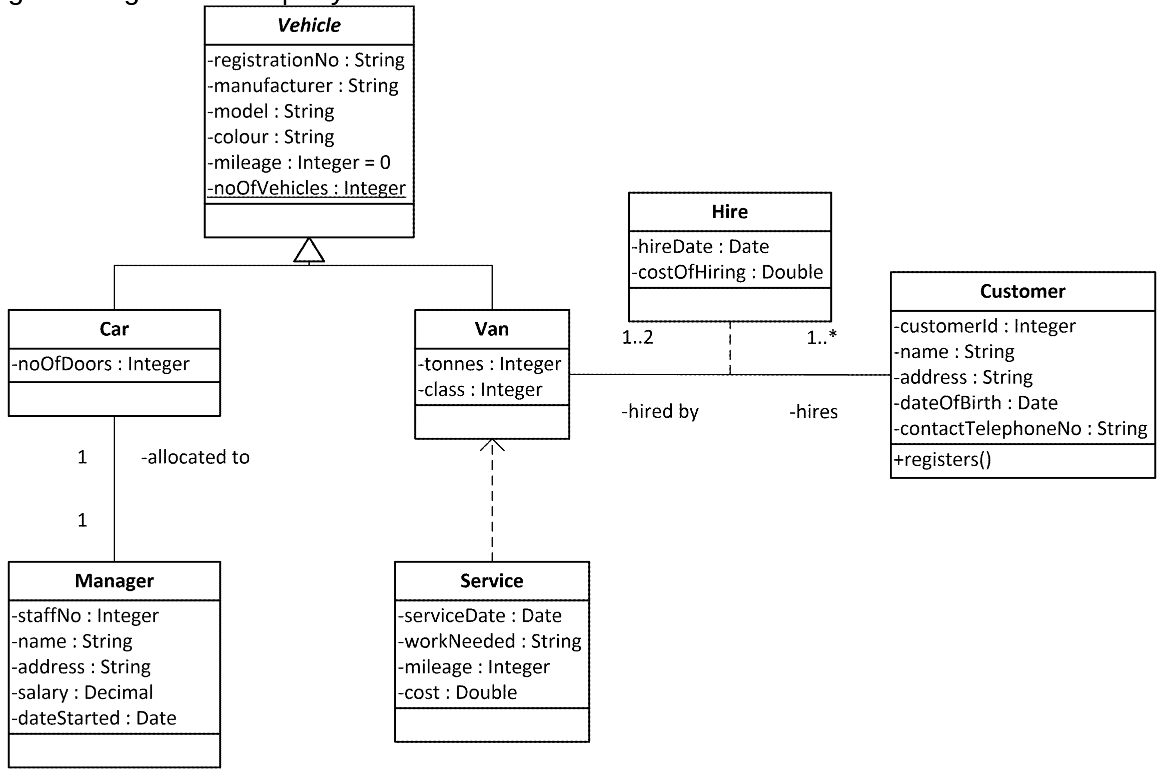
This was the second most popular question in the paper, with 88% of candidates attempting it. However, the mean mark was second from lowest, indicating a degree of overconfidence here. Part (a) was answered generally well, with most candidates able to define the basic terms listed in (i)-(v) relatively clearly. Of these, the terms that was least often correctly defined was Message. Part (b) required candidates to provide text for a website on the history of programming languages. The most common mistake here was not to keep in mind the purpose of the website – the *history* of programming languages, and to neglect to place object oriented programming in an appropriate historical context with its predecessors and relatives (e.g., unstructured, procedural, functional, and so on). Most candidates were able to score some marks by listing some of the features of an object oriented programming language, but often this was merely a regurgitation of the answers provided for other questions in the paper.

SECTION B

Answer Section B questions in Answer Book B

4.

The class diagram below represents a Van Hire system that records the hirings of a van. Managers are given a company car:



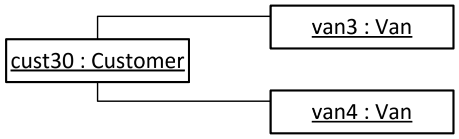
a) Describe what the diagram above represents. Include all structural constraints.

(15 marks)

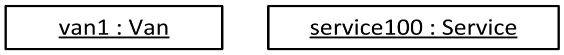
b) Given the object diagrams below (i-vi), state which are valid instances. If an instance is not valid explain why not.

(10 marks)

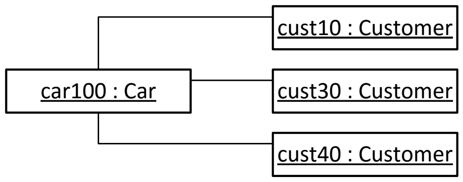
i.



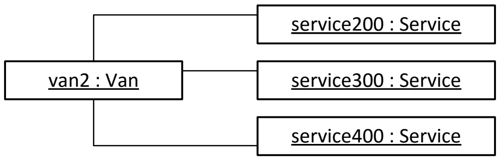
ii.



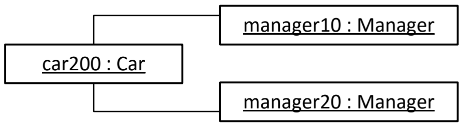
iii.



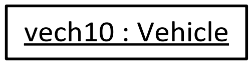
iv.



v.



vi.



Answer Pointers:

(a) The diagram has the following classes:

- Vehicle

  Abstract Class with the instance variables registrationNo, manufacturer, model, colour and mileage. Mileage has the default value of 0 when the object is instantiated. There is one class variable noOfVehicles

  Vehicle has two subclasses:

  - Car
  - Van

  Car has the instance variable noOfDoor.

  Van has the additional instance variables tonnes and class.

There are three other classes:

- Manager
- Customer
- Service

Manager has the instance variables staffNo, name, address, salary and dateStarted.

Customer has the instance variables customerId, name, address, date of birth and contact telephonoNo. There is a registers operation.

Service has the instance variables serviceDate, work needed, mileage and cost.

There are a number of associations in the classes:

- Allocated to between Car and Manager. Each member of staff can be allocated 1 car and 1 car only. A car is allocated to 1 Manager only.
- The relationship between Van and Service is a dependency relationship
- Association class Hires between Van and Customer. A van can be hired by up to 2 customers per hiring. A Customer hires one or more vans.

(b)

i) Ok
ii) No link between van and service
iii) No relationship between car and customer
iv) Ok
v) Car can only have one manager
vi) Vehicle cannot be instantiated

Examiners Comments:

This question examines the Design section of the syllabus

This was a popular question, with over 70% attempting the question and over three-quarters passing it.

For part a, most candidates could describe the main classes and their attributes adequately. Marks were lost by not describing the associations correctly, or not at all. To gain full marks, all aspects needed to be described, for example, the abstract class was often overlooked and the class variable and default value were not commented on. Some candidates just listed the class names, without describing the class too.

Other issues included writing code for the diagram, or describing object-oriented techniques in general, rather than describing the given diagram.

For part b, diagrams ii and vi caused the most problems, with candidates not recognising that Vehicle is an abstract class and there was a dependency relationship between Van and Service.

5.

a) When developing any system, it needs to be tested. Discuss which techniques are appropriate for testing systems developed using object oriented technology.

(10 marks)

b) Given the following classifications for design patterns:

Creational Patterns:

Abstract factory, Builder, Object Pool and Singleton patterns

Structural Patterns:

Adaptor, Decorator, Façade and Proxy patterns

Behavioural Patterns:

Command, Iterator, Observer and State patterns

Pick ONE design pattern from EACH of the above classifications and give a detailed description of each, which should include what the problem they address and an example of their use.

(15 marks)

Answer Pointers

(a) An open-ended question. The Candidate may look at different approaches, for example:

- Fault based testing
- Scenario based testing
- How different UML diagrams can be utilised

Or may discuss black box and white box testing with respect to OO programming.

(b) The candidates needed to choose one design pattern from the three classifications:

Creational Patterns:

Abstract factory
Provides an interface for creating families of related or dependent objects without specifying their concrete classes

Builder
This allows a complex object to be constructed of other parts. It is done in a way that different representations can be produced from the same construction process. The design pattern consists of a Builder class, which is an abstract interface; a ConcreteBuilder that constructs and implements the Builder class; a Director class that constructs the complex object and a class representing the complex object, such as a Product. For example, in a fast food

restaurant there can be different set meals, made up of different items, but the construction process is the same for all.

Object Pool
Manages the reuse of objects for a type of object that is expensive to create or only a limited number of a kind of object can be created.

Singleton
The singleton pattern applies to the many situations in which there needs to be a *single instance* of a class, a single object. Print spoolers and window managers are examples of Singletons. The Singleton is intended to provide a way to ensure that a class provides one instance of itself, and to provide a global point of access.

Structural Patterns:

Adapter
The Adapter is intended to provide a way for a client to use an object whose interface is different from the one expected by the client, without having to modify either. This pattern is suitable for solving issues such as replacing one class with another when the interfaces do not match and creating a class that can interact with other classes without knowing their interfaces at design time.

Decorator
Ordinarily, an object inherits behaviour from its subclasses. The decorator pattern allows the behaviour of an object to be extended and dynamically compose an object's behaviour. Decorator allows this without the need to create new subclasses.

Façade
A single class that represents an entire subsystem and so provides a unified interface to a set of interfaces. It acts like a wrapper to an underlying system. For example, a customer service representative that provides an interface to the company's order, billing and despatch departments.

Proxy
Proxy is one object representing another object. Can be used where you have resource hungry objects and you do not want to instantiate objects until they are requested by the client.

Behavioural Patterns:

Command
The Command pattern encapsulates a command request in an object. Can be used as an object-oriented callback. It is used when you need to issue requests to objects without knowing anything about the operation being requested. The structure is the client who creates a command is not the same client that executes it. Can also be seen as a macro, where the Command design pattern provides the options to queue commands.

Iterator
The Iterator pattern provides a way to access the elements of an aggregate object sequentially without having to know the underlying representation. Iterator also provides a way to define special Iterator classes that perform unique processing and return only specific elements of the data collection. The Iterator is useful because it provides a common interface so programmer does not need to know anything about the underlying data structure.

Observer
The Observer pattern is useful when data is to be presented in several different forms at once. The Observer is intended to provide a means to define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. The object containing the data is separated from the objects that display the data and the display objects observe changes in that data. Often used in MVC.

State
The State design pattern allows you to alter an object's behaviour when its state changes. This allows an object to change its behaviour at run-time. Can be seen as an abstract wrapper to concrete State derived classes. For example, a vending machine will have different states, depending on what items it has to vend and what money has been deposited.

For each category, the candidate should give a short description of one of the diagrams, include a simple example and say when they should be used. The examples above were taken from https://sourcemaking.com/design_patterns and http://www.oodesign.com

Examiners Comments:

This question examines the Practice section of the syllabus

This question was less popular with the candidates, with less than half attempting it and over 60% passing it.

A lot of candidates described white and black box testing in part a. Others just listed a range of techniques, without discussing them. To gain full marks, the techniques used for testing had to be discussed with respect to object-oriented programming.

In part b, the most common design patterns chosen were the Singleton, Adaptor and Observer. A good answer included an example of use and why they should be used. Weaker answers only gave a brief description of the pattern, without any examples, or some described all the patterns very superficially. Some candidates only attempted one or two of the design patterns.

6.

ABC Kids Playgroup is an after school club for children under 12. The club wishes to keep information on the children and the staff who work there. Two types of staff are employed: Secretaries and Play Workers. Personal details, such as name and address are recorded for all staff and if they are a Play Worker they must also pass a police check. The system must record when this has been passed and when it must be renewed, because a Play Worker cannot work with the children until this condition is met.

When a child starts at the club, a Secretary records personal details such as name, address, date of birth and at least one emergency contact number, up to a maximum of three. As part of the registration process, the Secretary also records who is allowed to collect the child and their contact details. The Play Worker will use these contact details, so that they can check that the person collecting the child at the end of a session is authorised to make the collection.
Some children have special circumstances (e.g. suffers from asthma) which are treated with medication, and the Secretary will record what the condition is, what medicine can be used and what to do in an emergency.

There are different groups a child can join, after an initial assessment the Play Worker is responsible for allocating each child to a group appropriate to their age and ability.
Each week the Play Worker will generate a letter for the parents to advise them which activities their children will take part in.

At the end of each month the Secretary will generate an invoice for the parents to pay.

a) Draw a Use Case diagram for this system.

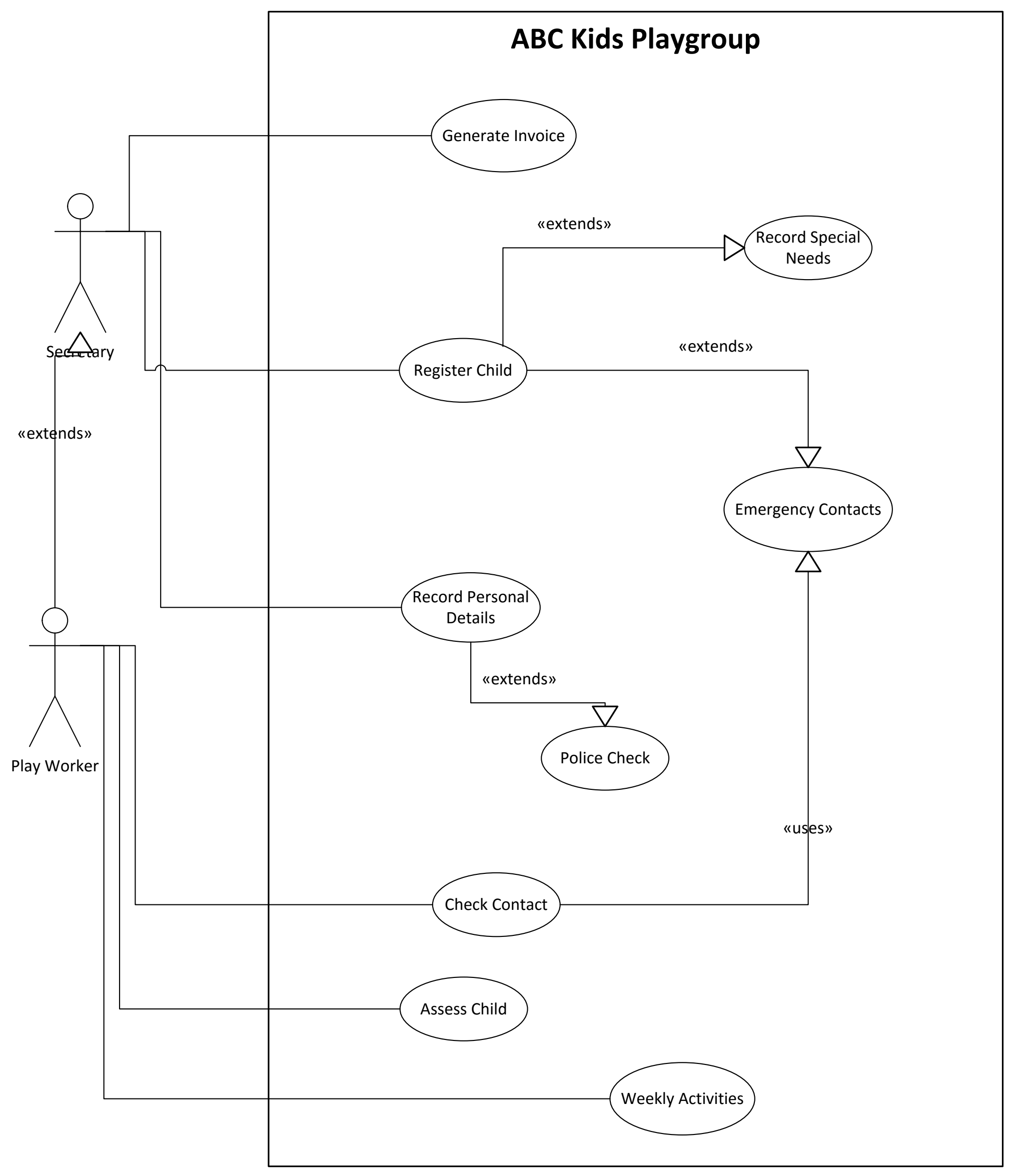
(15 marks)

b) Discuss the role of Use Cases (diagrams and descriptions) in the development of an object-oriented system.

(10 marks)

Answer Pointers:

Sample Use Case diagram for the ABC Kids Playgroup scenario:



(b) the candidate should discuss where Use Cases should be used in developing a system.

For each Use Case on the Use Case Diagram the user should develop a set of scenarios. Scenarios are natural language descriptions of an instantiation of the Use Case.

These can be used in two main areas:

- Initial investigation
  For identifying what functionality is required from the system
  The descriptions give the fuller detail

- Testing purposes
  The Use Case descriptions can be used by the testers to develop test plans, checking that the system meets the requirements of the system.

Examiners Comments:

This question examines the Design section of the syllabus

This was a popular question with the candidates, with over 80% attempting it and over 65% passing it.

For part a, most candidates produced the correct type of UML diagram, though a small minority produced a class diagram, instead of a Use Case diagram. Most candidate identified the correct actors, however, marks were lost by not linking the correct actor to the use case. In some cases, attributes were included as use cases, or very detailed descriptions were given.

For part b, some candidates described the diagram and the actors involved, rather than the role of use case. To gain full marks, the candidate needed to describe the role of both use case diagrams and descriptions, with some answers only describing the former. Candidates who failed this question generally did not attempt part b.