THE BCS PROFESSIONAL EXAMINATION
BCS Level 5 Diploma in IT
September 2017

EXAMINERS' REPORT

Object Oriented Programming

Question A1

a) Explain the terms *abstract data type* and encapsulation and describe how they implement coupling and cohesion in an object oriented system.

(10 marks)

b) Discuss the role of structured and procedural languages in the genealogy of object oriented languages.

(9 marks)

c) Explain the difference between typed and untyped languages. In your explanation, include ONE example of each.

(6 marks)

Answer Pointers

Part a):

Abstract data types (ADT) are datatypes described in terms of the operations they support, that is, their interface, rather than how they are implemented, which contrasts with a data structure, which is a concrete representation of data.

This can lead to high cohesion, since the interface provided by a good ADT can provide an abstraction of something intuitive, but is complex to implement.
Encapsulation is used as an information hiding mechanism, where the data structure of a class can be hidden, with a public interface provided by functions.

Encapsulation can lead to low coupling, by hiding the internals of an object, it minimises the effect of changing something within the class on other classes

Part b):

Structured programming is a programming methodology where a complex problem is decomposed into tasks that can be modelled by simple structures such as subroutines, block structures, for and while loops.

Procedural programming is a technique for decomposing a programming problem into a set of routines, or functions, which together can be composed to form the solution. Often uses a top-down design model.

Both approaches set the foundation for breaking the development of a system into a set of tasks (classes), that it is found in object oriented languages.

Part c):

Typed languages usually have pre-defined types for individual pieces of data, such as numbers within a certain range, strings of letters, etc. and programmatically named values (variables) can have only one fixed type and allow only certain operations: numbers cannot change into names and vice versa.

Untyped languages treat all data locations interchangeably, so inappropriate operations (like adding names, or sorting numbers alphabetically) will not cause errors until run-time.

For example, in java, a variable has to have a type when defined, whereas PHP does not:

```
String myString = “a string message”  /* java */
myString = “a string message”  /* PHP */
```

Examiner Comments

This question examines Part 1 of the syllabus Foundations

Part a) examines Parts 1.2 and 1.4
Part b) examines Part 1.1
Part c) examines Part 1.3

A popular question answered by over 85% of the candidates, though only 40% passed. Most candidates could answer one of the parts well, but not all three. For part a), weaker candidates only described either abstract data types, or encapsulation, or described both, but then could not relate them to coupling and cohesion. To gain full marks, the candidate needed to state whether ADTs and encapsulation implemented high or low coupling and cohesion.

In part b), weaker candidates only described either structural or procedural languages. Some candidates confused structural languages with SQL.

For Part c) most candidates could describe what a typed language was, but were weaker at defining untyped languages. Marks were lost if no example was given.

Question A2.

a) Give an example of each of the following diagrams and describe the context in which you would use them when developing an object oriented system:

   i) *Object interaction diagram;*
   ii) *Object state transition diagram*.

(10 marks)

b) Design patterns can be categorised into three broad categories:

   i) *Creational*;
   ii) *Structural*;
   iii) *Behavioural*.

   Briefly describe what these categories represent and give ONE example of each category, stating what their motivation is and the basis of the solution they offer.

(15 marks)

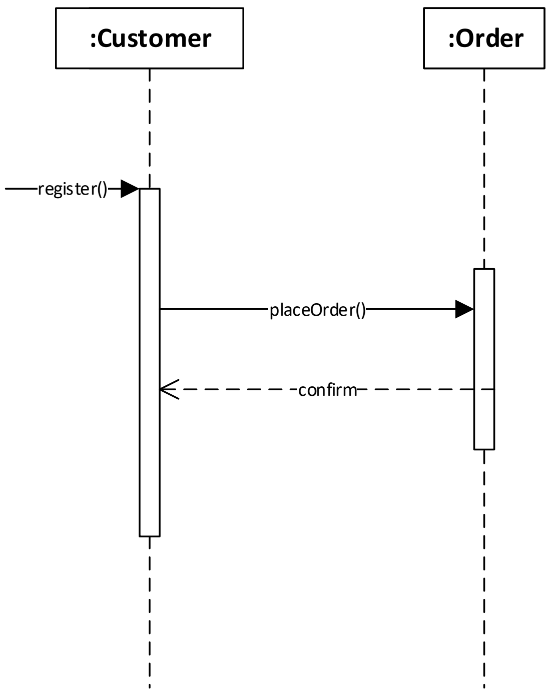
Answer Pointers

Part a):

Object interaction diagram

Interaction diagrams show some type of interaction between the different elements in the model. They can be used to represent the dynamic behaviour of the system and show how objects interact and the message flows between the objects. The candidate may give an example of an object diagram, sequence diagram or collaboration diagram.

For example, a sequence diagram for a sales order system:

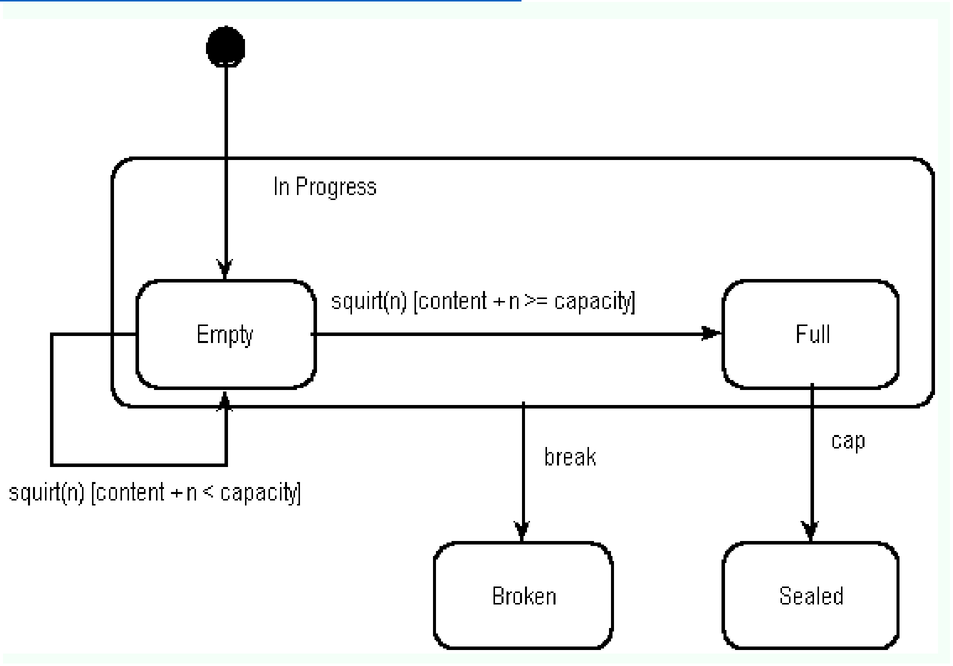


Object state transition diagram

This can be used from the start of the modelling process, the main idea is to define a machine that has a number of states. The machine receives events from the outside world and each event causes the machine to transition from one state to another.

They can be used to describe the behaviour of a single object, they are not so good at showing the behaviour of several objects, where interaction diagrams should be used instead.

For example, the following represents a bottle object in a bottling plant taken from http://www.cs.unc.edu/~stotts/145/CRC/state.html:


In Progress
Empty
squirt(n) [content + n >= capacity]
Full
break
cap
squirt(n) [content + n < capacity]
Broken
Sealed

Part b):

Creational design patterns are concerned with the construction of object instances. Structural patterns deal with structural issues, such as how to structure resources. Behavioural patterns represent dynamic descriptions, such as how things change over time.

Examples of each type include:

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

The candidate needs to describe one of each type, saying what their motivation is and what solution they offer.

For example:

Singleton (Creational)

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.

Adapter (Structural)

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.

Iterator (Behavioural)

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.

Examiner Comments

This question examines Part 3 of the syllabus Design
Part a) examines 3.2
Part b) examines 3.4

This question was answered by 24% of the candidates and over 58% passed. Part a) was often not attempted, whilst most candidates could make a good attempt at describing design patterns.

Object diagrams were the most popular example of an object interaction diagram in part a). Candidates often lost marks by not describing the context in which you would use either type of diagram when developing an object oriented system.

The most popular design patterns for part b) were the singleton (creational), adaptor (structural) and iterator (behavioural). Only one design pattern per type was required, candidates were not given any extra marks for describing several patterns and in such cases, only the best one got credit. Candidates lost marks if the wrong example was given for each category.

To gain full marks, the answer required stating what the motivation was for each pattern, which some candidates overlooked.

Question A3.

Sports World is an Events Management Company that organise the running of major sporting games, such as the Commonwealth Games.

Before being used to host a sporting event, venues are assessed by an Administrator who checks that they are for fit for purpose. If the venue can hold more than 10,000 people, the Administrator conducts additional health and safety checks to ensure that the venue is safe.

A year before the games begin, a Team of Staff are appointed to run the day-to-day operations, including booking successfully assessed/safety-checked venues. Six months before the games begin, the Team of Staff produce a Programme that lists the date, time and location of each sporting event. At this point, Athletes can register for an event by giving their name, address, date of birth and best time for their event. Some overseas athletes need to apply for a visa and the system needs to record whether they were successful.

A week before the games begin, staff produce a Schedule that shows when the registered athletes will participate in their event. At the end of each event, staff produce a Table of Results that records the positions of each athlete.

Once all the events are completed, the administrator checks the Table of Results for accuracy and produces a Medals Table.

a) Produce a Use Case diagram for the above scenario.

(15 marks)

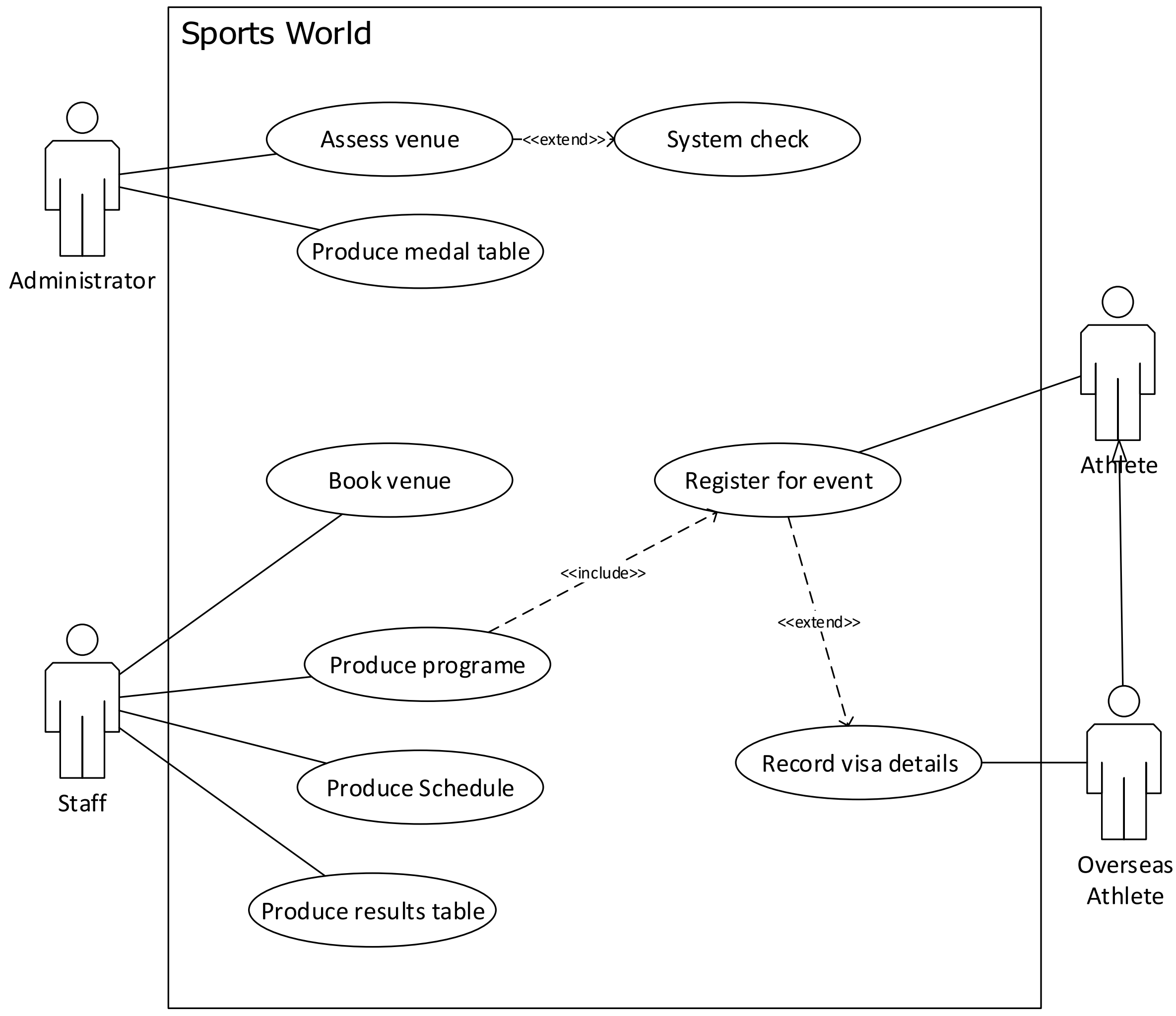
b) Discuss how Use Case diagrams and descriptions provide an overview of the user requirements of a system. Within your answer include examples from the above scenario.

(10 marks)

Answer Pointers

Part a):

Sample answer:



Part 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

The candidate should include brief examples from the Sports World description to illustrate their points.

Examiner Comments

This question examines Part 3 of the syllabus Design (section 3.1)

This was a popular question, answered by 87% of the candidates, with 79% passing.

For part a), most candidates made a good attempt at identifying all the actors and the use cases. To gain full marks, each use case needed to be associated with the correct actor. Some names given for the use cases were very wordy, whereas they should be short, as seen in the diagram above. Some candidates also included attributes on the diagram, which did not gain any credit.

Part b) was sometimes not attempted. An example from the scenario was required, though some candidates only described the scenario again, without trying to discuss how use case diagrams and descriptions provide an overview of the user requirements of a system. To gain full marks the candidate needed to discuss the role of use case descriptions too.

Question B4.

In number theory, a value can be categorised as a natural number (a whole number >0, often denoted ℕ), an integer (zero or a positive or negative whole number, including the natural numbers, often denoted ℤ), or a real number (which includes the natural numbers and integers, along with all other positive and negative numbers that are not integers, often denoted ℝ).

a) In an object oriented programming language of your choice, write a definition for a `number` class that contains:

   (i) a single field suitable for storing either a natural number, or an integer, or a real number;

   (ii) setter and getter methods for manipulating this field;

   (iii) a constructor that initialises new objects of `number` to have the value 1 (unity);

   (iv) a method that determines which kind of number is currently stored (returning 0 if the number is real and an integer and a natural number, 1 if the number is real and integer but not a natural number, and 2 if the number is real but neither an integer nor a natural number).

(15 marks)

b) Write a new method `isPrime()` that returns true or false, depending upon whether the number held is a prime number. A prime number is a natural number >1 that is divisible only by 1 and itself.

(10 marks)

Answer Pointers

Part a):

```
public class number
{
  private double n;
  public void setValue(double newValue)
  {
    n = newValue;
  }
  public double getValue()
  {
    return n;
  }
  public number()
  {
    n = 1;
  }
  public int numberType()
  {
    int numberTypeCode;
    if(Math.floor(n) == Math.ceil(n) && Math.floor(n) >  0)
      numberTypeCode = 0;
    else if(Math.floor(n) == Math.ceil(n))
      numberTypeCode = 1;
    else
      numberTypeCode = 2;
    return numberTypeCode;
  }
}
```

Part b:

```
public bool isPrime()
{
  if(numberType()==0) // n is natural
  {
     int nInt = Math.ceil(n);
     for(int d = 2;d<nInt;d++)
     {
       if(nInt%d==0)
         return false;
     }
     return true;
  }
  return false;
}
```

Examiner Comments

This question examines Part 4 of the syllabus Practice (4.2, 4.3), and also requires an understanding of Part 2, Concepts (2.1, 2.3).

This question was attempted by 44% of candidates, of whom 42% submitted an answer in the pass range. In part a), despite the question explicitly requesting that a single field is used, many candidates including three fields in their class (attempting to correspond these to the three number types discussed). Furthermore, in some cases all three fields were integers, which must be incorrect given that a non-integer real number cannot be held in an integer by definition. The method to determine the type of number was generally poorly implemented, perhaps reflecting the above erroneous choice. A very small number of candidates realised that ceiling and floor operations, or some other analytical process able to determine whether the number held was whole or not, was required. In part b), many answers failed to appreciate that to test the primality of a number requires a loop to iterate through the divisors (using the simplest brute-force approach; however, other approaches would also require loops).

Question B5.

Consider the code fragment written below:

```
public class A
{
  private   int  a;
  protected int  b;
  public    int  c;
  public         A();
  public    void seta(int new_a);
  public    void setb(int new_b);
  public    void setc(int new_c);
  public    int  geta();
  public    int  getb();
  public    int  getc();
}

public class B
{
  private   A    myA;
  private   int  d;
  public         B();
  public    void setd(int new_d);
  public    int getd();
}
```

a) State the relationship between class `A` and class `B`, and show how this code fragment would be represented in a UML class diagram.

(10 marks)

b) State the name of one other kind of inter-class relationship, and show both a code fragment in which this relationship is implemented, and how it would be represented in a UML class diagram.

(15 marks)

Answer Pointers

Part a):

```
---------------------
|A                  |
|-------------------|
|- a: int           |
|# b: int           |
|+ c: int           |
|-------------------|
|+ A                |
|+ seta(int): void  |
|+ setb(int): void  |
|+ setb(int): void  |
|+ geta(): int      |
|+ getb(): int      |
|+ getc(): int      |
---------------------
          |
          ◆
          |
---------------------
|B                  |
|-------------------|
|- myA: A           |
|- d: int           |
|-------------------|
|+ B                |
|+ setd(int): void  |
|+ getd(): int      |
---------------------
```

1 mark for identifying correct relationship (composition, or *has-a* relationship). Could also be interpreted as aggregation, if it is assumed that B uses A but does not exclusively own it.
1 mark for correct symbol between classes in UML (filled diamond for composition, or empty diamond if aggregation assumed)
1 mark for correct UML boxes, with 3 sections
1 mark for placing fields in middle section, and methods in bottom section
1 mark for + symbol used for visibility of constructors and pubic field
1 mark for # symbol used for visibility of protected field
1 mark for – symbol used for visibility of private fields
1 mark for + symbol used for visibility of setters and getters
1 mark for correct return type for getters and arguments for setters
1 mark for showing object of A as a member of B

Part b):

The classes are related by inheritance (i.e., an *is-a* relationship).

```
---------------------
|A                  |
|-------------------|
|- a: int           |
|-------------------|
|+ A                |
|+ seta: void       |
|+ geta(): int      |
---------------------
          |
         /\
          |
---------------------
|B                  |
|-------------------|
|- b: int           |
|-------------------|
|+ B                |
|+ setb(int): void  |
|+ getb(): int      |
---------------------
```

UML Diagram, 9 marks allocated as follows:

1 mark for identifying correct relationship (e.g., inheritance, or *is-a* relationship)
1 mark for correct symbol between classes in UML (e.g., empty triangle)
1 mark for correct UML boxes, with 3 sections
1 mark for placing fields in top section, and methods in bottom section
1 mark for + symbol used for visibility of constructors
1 mark for – symbol used for visibility of private fields
1 mark for + symbol used for visibility of setters and getters
1 mark for correct return type for getters
1 mark for correct arguments for setters

Example Code Fragment Marks, 6 marks, example follows:

```
public class A
{
  private int a;
  public A();
  public void seta(int newa);
  public int geta();
}

public class B extends A
{
  private int b;
  public B();
  public void setb(int newb);
  public int getb();
}
```

Examiner Comments

This question examines Part 3 of the syllabus Design (3.2), and Part 2 Concepts (2.1, 2.2, 2.3), and Part 4, Practice (4.3).

This question was attempted by most (78%) of candidates, and 58% of those scored a mark in the pass range. In part a), a large number of candidates incorrectly interpreted the inter-class relationship as inheritance, despite that no inheritance keywords appear in its definition. It was also clear that many candidates were insufficiently familiar with UML (in parts a) and b) to correctly populate a class with members, and denote their visibility, and (for methods), input and output types. In part b), a relatively large number of candidates replicated the same class structure given in part a). Very few marks were awarded for this. However, many candidates did also present a reasonable UML class diagram, including representing the relationship and members, showing (for most candidates) inheritance.

Question B6.

a) Define *class variable* and *instance variable*, and provide a code fragment that implements these concepts in an object oriented programming language of your choice, and demonstrates how they may be used.

(10 marks)

b) Explain why a developer might declare a member function as private, and show an example code fragment, in an object oriented programming language of your choice, that contains both public and private methods to demonstrate how they may be used.

(15 marks)

Answer Pointers

Part a):

Class variables are held at the class level. They do not require that an object has been created, and are sometimes considered to be shared between all objects (i.e., a single copy exists, regardless of how many instances have been created). An instance variable belongs to an object, such that each object created from a class has its own copy, and the instance variable may hold a different value in each object. An example in C++ is shown below:

```
#include <iostream>

class A
{
  public:
    static int myClassVariable;
    int myInstanceVariable;
}

int main(void)
{
  A::myClassVariable      = 5;
  A::myInstanceVariable   = 10; // compile-time error

  A myA1, myA2;
  myA1.myInstanceVariable = 15;
  myA1.myInstanceVariable = 20; // each object has its
                                // own instance variable
                                // value
}
```

Part b):

Ordinarily, methods are declared public and as such may be invoked through an object of the class (e.g., to manipulate instance variables). Sometimes, methods are declared private, and are only able to be invoked by other methods that are also member of the class. Their purpose is often to assist those public methods by performing some duty that should *not* be exposed to the user of the class through the public interface, since the user of class does not need to be aware of the irrelevant implementation details –

sometimes such private methods are referred to as helper functions or support methods. An example in C++ is shown below:

```
#include <iostream>

class A
{
  private:
    void mySupportMethod() { std::cout << "do support\n"; }
  public:
    void myServiceMethod()
    {
      std::cout << "do service\n";
      mySupportMethod();
    }
}

int main(void)
{
  A myA;
  myA.myServiceMethod(); // will print "do service", and
                         // then "do support"

  myA.mySupportMethod(); // not allowed (compile error)
  return 0;
}
```

Examiner Comments

This question examines Part 4 of the syllabus Practice (4.2, 4.3) and Part 2 Concepts (2.1, 2.2, 2.3, 2.4).

This question was attempted by the majority of candidates (81%), of whom 51% submitted an answer that was in the pass range. In part a), many candidates correctly suggested that class variables belonged to the class they were defined in, rather than to individual objects, and that this behaviour can be implemented in common programming languages using the static keyword. Some other candidates seemed a little confused by this terminology, thinking that a class variable was declared in a class, and an instance variable in a method (i.e., as a local variable with a limited scope). In part b), some candidates provided answers that discussed why data might be designated public, private, or protected, but made no mention of functions (methods), so did not really address the question. However, many good answers were submitted that described private designated functions/methods as providing a support/helper role that did not requite visibility outside the class.