BCS THE CHARTERED INSTITUTE FOR IT

BCS HIGHER EDUCATION QUALIFICATIONS
BCS Level 5 Diploma in IT

OBJECT ORIENTED PROGRAMMING

Examiners Report

September 2015

Question A1

a) Define the following terms:

i) Data hiding;
ii) Encapsulation;
iii) Typed language;
iv) Coupling;
v) Cohesion.

(10 marks)

Answer Pointers

Data hiding is a software development technique specifically used in object-oriented programming (OOP) to hide internal object details (data members). Data hiding ensures exclusive data access to class members and protects object integrity by preventing unintended or intended changes.

Encapsulation supports the notion of gathering together related aspects of a design, grouping them and hiding the implementation.

In typed languages, there usually are 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.

Coupling refers to the degree to which each program module relies on each other module.

Cohesion refers to the degree to which each part of a module is associated with each other part, in terms of functional relation.

b) A stack is a last in, first out linear data structure. A stack can have any object as an element. It is characterised by two fundamental operations, called *push* and *pop*. The push operation adds a new item to the top of the stack. If the space allocated to hold the stack is full when the push operation is attempted then an error condition is raised. The pop operation removes an item from the top of the stack. A pop reveals previously concealed items, or results in an empty stack. If the stack is empty when a pop 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 stack. Your code should store the stack elements in an array and should not make use of a stack class from a class library.

(15 marks)

Answer Pointers

```
public class Stack
{
    private Object[] stackArray;
    private int maxSlot;
    private int currentSlot = 0;

    public Stack(int size)
    {
        maxSlot = size;
        stackArray = new Object[size];
    }

    public void push(Object o) throws OverflowException
    {
        if (currentSlot == maxSlot)
        {
            throw new OverflowException();
        }
        stackArray[currentSlot] = o;
        currentSlot++;
    }

    public Object pop() throws UnderflowException
    {
        if (currentSlot == 0)
        {
            throw new UnderflowException();
        }
        currentSlot--;
        return stackArray[currentSlot];
    }
}
```

Examiners Comments

This question examines the Foundations section of the syllabus.

Three quarters of the candidates for this examination chose to attempt this question. There were a few good answers however many candidates scored poorly. The evidence shows that there is still confusion of the terms ‘Encapsulation’ and ‘Data Hiding’ with many candidates believing that their meaning is identical. The term ‘typed language’ is not well understood.

Many candidates did not attempt this part of the question at all. Candidates who gained part of the marks for this section of the question were generally able to code push and pop but did not produce code for the constructor.

Question A2

Describe: the following design patterns:

i) Iterator;
ii) Observer;
iii) Singleton.

For each pattern, state the motivation for the pattern, give a UML class diagram for the pattern and an explanation of the classes which participate in the pattern.

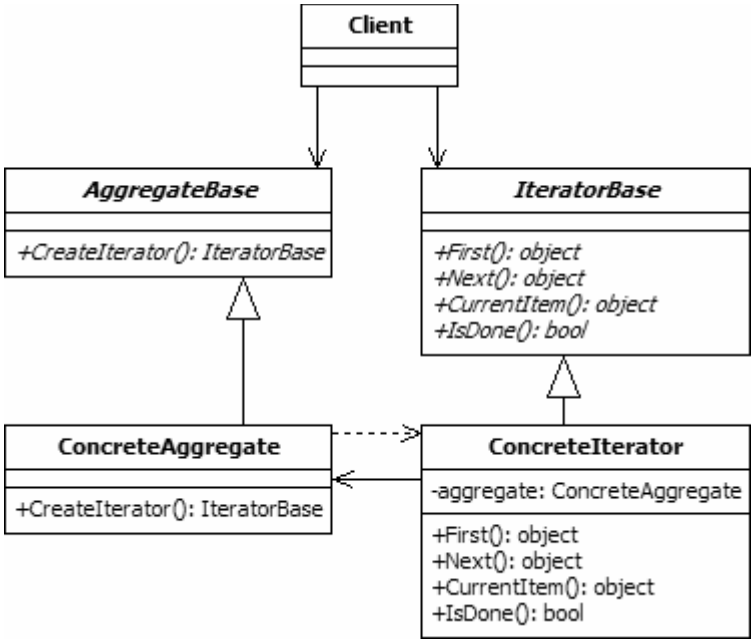
(25 marks)

Answer Pointers

From http://www.blackwasp.co.uk/GofPatterns.aspx

Iterator

The iterator pattern is used to provide a standard interface for traversing a collection of items in an aggregate object without the need to understand its underlying structure.

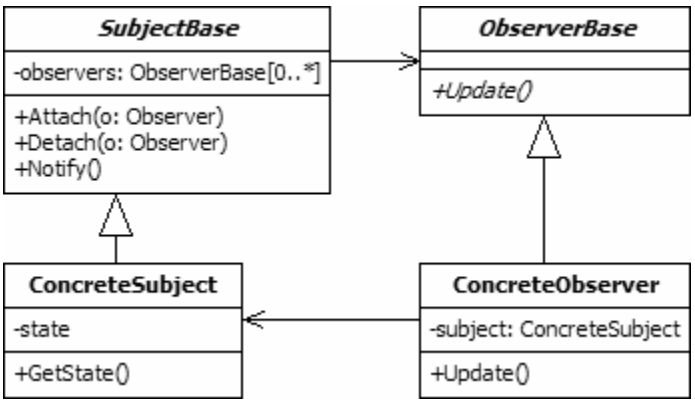


Client. Objects of this type are the consumers of the iterator design pattern. They request an iterator from an aggregate object when they wish to loop through the items that it holds. The methods of the iterator are then used to retrieve items from the aggregate in an appropriate sequence.
AggregateBase. This abstract class is the base class for aggregate objects. It includes a method that generates an iterator, which can be used to obtain references to the objects that subclasses contain. This class is often implemented as an interface.
ConcreteAggregate. The concrete aggregate classes provide the real functionality for aggregate objects that contain collections of items that can be traversed using an iterator.
IteratorBase. This abstract class is the base class for iterators. It defines a standard interface that includes methods to allow the elements of the aggregate that generated it to be looped through in sequence. This class is often implemented as a simple interface.

ConcreteIterator. Concrete iterators implement the interface defined by the IteratorBase class. They provide functionality specific to the ConcreteAggregate class used to generate them, hiding the implementation of the aggregate from the client.

Observer

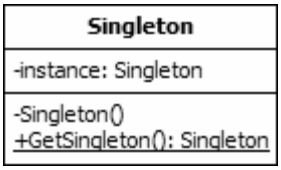
The observer pattern is used to allow an object to publish changes to its state. Other objects subscribe to be immediately notified of any changes.



SubjectBase. This is the abstract base class for concrete subjects. It contains a private collection of the observers that are subscribed to a subject and methods to allow new subscriptions to be added and existing ones to be removed. It also includes a method that can be called by concrete subjects to notify their observers of state changes. This *Notify* method loops through all of the registered observers, calling their *Update* methods.
ConcreteSubject. Each concrete subject maintains its own state. When a change is made to that state, the object calls the base class's Notify method to indicate this to all of its observers. As the functionality of the observers is unknown, the concrete subjects also provide the means for the observers to read the updated state, in this case via a GetState method.
ObserverBase. This is the abstract base class for all observers. It defines a method to be called when the subject's state changes.
ConcreteObserver. The concrete observer objects are the subscribers that react to changes in the subject's state. When the Update method for an observer is called, it examines the subject to determine which information has changed. It can then take appropriate action.

Singleton

The singleton pattern ensures that only one object of a particular class is ever created. All further references to objects of the singleton class refer to the same underlying instance.



In this diagram the only public interface element is the static GetSingleton method. This method returns the single instance held in the private "instance" variable. The constructor for the class is marked as private. This prevents any external classes from creating new instances. The class should not allow inheritance, which could lead to subclassing that breaks the singleton rules.

Examiners Comments

This question examines the Design section of the syllabus

Although this entire question was bookwork as the three design patterns are specifically mentioned in the syllabus, only a small percentage of candidates chose to attempt it. The evidence shows that many answers were not informed by a study of design patterns and therefore failed to attract any credit. In general, candidates were more able to draw the UML diagrams of the design patterns than to describe the functions of the classes shown in the diagrams. There were no solutions which gained full credit for the question.

Question A3

a) Define the following terms:

i) Object;
ii) Class;
iii) Instantiation;
iv) Garbage collection;
v) Destructor.

(10 marks)

Answer Pointers

Object refers to a particular instance of a class where the object can be a combination of variables, functions, and data structures.

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)

Instantiation is the creation of a rea instance for particular realization of an abstraction or template such as a class of objects or a computer process. To instantiate is to create such an instance by, for example, defining one particular variation of object within a class, giving it a name, and locating it in some physical place.

Garbage collection is a form of automatic memory management. The garbage collector attempts to reclaim garbage, or memory occupied by objects that are no longer in use by the program.

A destructor is a method which is automatically invoked when the object is destroyed. Its main purpose is to free the resources (memory allocations, open files or sockets, database connections, resource locks, etc.) which were acquired by the object along its life cycle and/or deregister from other entities which may keep references to it.

b) Explain how object oriented languages implement abstract data types.

(5 marks)

Answer Pointers

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.

c) Explain how object oriented languages attempt to simplify memory management for programmers.

(15 marks)

Answer Pointers

Programs are implemented as a sequence of operations acting upon data structures. In traditional programming the data structures would be superimposed onto an arbitrary piece of memory requested by the programmer. They would only be loosely associated with the code which operated on them. In object oriented programming languages the instantiation of an object results in a reference to the object rather than a set of addresses of where the object is stored. Instantiation ensures that exactly the correct amount of memory is allocated without the programmer having to explicitly state it. The object reference can then be associated with the methods which belong to the object class i.e. code cannot be coerced to operate on inappropriate items. As the programmer modifies the object during maintenance and testing there is no need to adjust pointer offsets etc the compiler can work out how to modify the references to storage in line with any redefinition of the class. When the object is no longer needed the memory management system which supports the OO language ensures that only memory which is no longer necessary is made available for future use. Where languages use garbage collection this is an automatic process. Where for efficiency reasons garbage collection is not used the programmer initiates the reuse of memory but the system manages exactly which locations are returned to the heap.

Examiners Comments

This question examines the Concepts section of the syllabus

This proved to be a very popular question which attracted a number of good answers. Part a) was bookwork and in general attracted good answers from all the candidates who attempted the question. There is evidence that candidates were less sure about the way in which an object oriented language can be used to implement an abstract data type (ADT). As always, there was confusion between an ADT and an abstract class. The majority of candidates were able to gain some credit in part c) although the vast majority of answers to this part only covered the way memory is deallocated and not the mechanisms for allocating it.

Question B4

a) Discuss the role of the following UML diagrams in the development of an object oriented system. Include brief examples of their use:

   i) Use Case diagram;
   ii) Sequence diagram;
   iii) Deployment diagram.

(15 marks)

b) Discuss the techniques that can be used to test systems developed using object oriented technology.

(10 marks)

Answer Pointers

Part a)

The candidate should discuss where the following diagrams are used in developing a system. For each one the candidate should describe what the main aim of the diagram is and give examples of their use.

For example:

i) Use Case diagram:

A graphical overview of the functionality provided by a system in terms of actors, their goals and any dependencies between those use cases

For each Use Case on the Use Case Diagram the user should develop a set of scenarios. These can be used for the following:

- Initial investigation
- For identifying what functionality is required from the system
- Testing purposes

ii) Sequence diagram:

An interaction diagram that shows how processes are linked with each other and the order in which they operate. The sequence diagram shows the objects involved, typically representing a use case; they are shown in a time sequence showing the exchange between the objects needed to carry out the functionality of the use case.

iii) Deployment diagram.

This shows a static view of the run-time configuration of nodes and their components. They are particularly used to show the hardware requirements for the system, or to show the architecture showing how the hardware and software works together. They can be used when an application is to be deployed on different machines.

Can be used for:

- Identifying the scope of the application;
- Showing the technical issues;
- Showing the distribution architecture;
- Identifying the nodes and connections;
- Shows how the nodes and software relate.

Part b)

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 object-oriented programming.

Examiners Comments

This question examined part 4 of the syllabus: Practice

This proved a popular question, with 72% of candidates attempting it, though only 43% passed. A large number of candidates only attempted either part a) or b).

To achieve a good mark in part a, all three types of UML diagrams should be covered. Weaker answers only covered one or two types of diagrams. Examples of use were expected for full marks, some candidates omitted these, or gave inappropriate examples, for example, describing the wrong type of sequence diagram.

For part b, good answers discussed appropriate ways of testing object oriented programs. A lot of candidates just described black and white box testing, without trying to relate how these techniques could be applied to an object oriented program.

Question B5

A University wishes to keep information on its students. The proposed Student class has the following instance variables:

studentNo:	String
studentName:	String
dateOfBirth:	Date
tariffPoints:	Integer

Tariff Points represents the entry qualification achieved by a student, which is a number between 20 and 280.

A class variable is also required, called noOfStudents, which will be incremented each time a Student instance is created.

Using an object oriented programming language that you are familiar with, write code to perform the following, where appropriate include suitable integrity checks:

a) Show the declaration of the Student class, including any *setter* and *getter* methods.

(15 marks)

b) Declare two constructors as follows; both constructors should increment the class variable appropriately:

- The first is a default constructor that has no parameters and sets the instance variables to either "not known" for the strings, 20 for the integer and 1st January 1995 for the date (assume there is a Date constructor that accepts dates in a string format).
- The second takes 4 parameters, one for each of the instance variables.

(8 marks)

c) Show how both constructors could be used to instantiate an object.

(2 marks)

Answer Pointers

Sample code below that covers Part a and b)

```
import java.util.Date;
import java.text.SimpleDateFormat;
import java.text.ParseException;

class Student{
        String studentNo;
        String studentName;
        Date dob;
        int tariffPoints;
        static int noOfStudents = 0;

/* constructors */

        public Student() {
                studentNo = "Not known";
                studentName = "Not known";
                tariffPoints = 20;
                /* will accept simpler (now depreciated):
                   dob = new Date("01-Jan-1995"); */
                try {
                  SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
                  dob = formatter.parse("01-Jan-1995");
                } catch (ParseException e) {
                e.printStackTrace();
                        }
                noOfStudents++;
                }

        public Student(String sno, String sname, String aDOB, int tp) {
                setStudentNo(sno);
                setStudentName(sname);
                setDOB(aDOB);
                setTariff(tp);
                noOfStudents++;
                }

/* getters */

        String getStudentNo() {
                return studentNo;
        }
        String getStudentName() {
                return studentName;
        }
        Date getDOB() {
                return dob;
        }
        int getTariff() {
                return tariffPoints;
        }

/* setters */
```

```
	void setStudentNo(String aName) {
		studentNo = aName;
	}
	void setStudentName(String aName) {
		studentName = aName;
	}
	void setTariff(int tp) { /* or may return an error code */
		if ((tp < 20) || (tp > 280)) {
			System.out.println("Tariff Points not between 20-280");
			System.out.println("Data not added");
			}
		else
			tariffPoints = tp;
	}
	void setDOB(String aDate){
		try {
			SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
			dob = formatter.parse(aDate);
		// dob = new Date(aDate); // only expected this in the exam paper
			} catch (ParseException e) {
		e.printStackTrace();
	}
	}
} // student
```

Part c)

```
Student s1 = new Student();
Student s2 = new Student(“0123”,’”Tom Jones”, “01-Feb-1995”,200);
```

Examiners Comments

This question examines parts 8d of the syllabus Practice

Over 60% of candidates attempted this question, with over 70% passing it. Most candidates who attempted this question produced a good answer, with some achieving full marks. Weaker answers were where candidates failed to produce two appropriate constructors, or did not demonstrate the classes correctly. A higher mark went to those who included the integrity check on the Tariff Points.

Question B6

A private dental practice wishes to computerise its patient records system. A patient must register with the practice and the system needs to store their name, address and mobile telephone number. Each patient is given a unique seven digit patient number. The system will keep a count of how many patients the practice currently has.

Patients can book an appointment with a particular dentist; the system needs to store the date of the appointment and if the patient attended. A text message will be automatically sent out two working days before the appointment.

After the appointment the dentist update the system with the cost of the treatment undertaken.

The practice employs two types of staff: Receptionists and Dentists. The system needs to record their details; which for all staff includes a four digit employee number, their name,

address, gender, contact telephone number and next of kin. Dentists must be qualified; the system will store their highest dental qualification, date awarded and their General Dental Council registration number.

A list of appointment statistics is required at the end of each week. This will be a summary of how many patients turned up and how many were no-shows. If a patient repeatedly misses an appointment they will be charged a fixed amount of money.

All receptionists must go on a first aid course every year. The system must record the date of when they last attended the course and the name of the course provider.

a) Draw a Class diagram for this system.

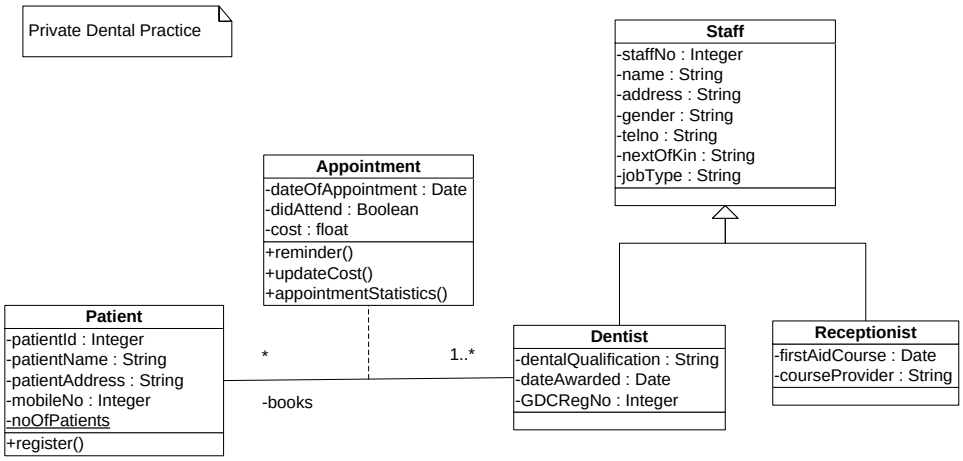
(20 marks)

b) Explain what is meant by the term *refactoring* with respect to class diagrams.

(5 marks)

Answer pointers

Part a) Sample class diagram:



Part b)

Refactoring classes: Refactoring can be used to split a class into partial classes or to implement an abstract base class. This is usually used because methods have become too large.

Examiners Comments

Part a of this question examines part 8c of the syllabus Design
Part a of this question examines part 8d of the syllabus Practice

This question was answered by 87% of candidates with 63% passing. Most candidates who produced a Class Diagram for part a) achieved a pass mark, those who failed were candidates who produced a Use Case diagram instead. Weaker answers did not show the inheritance correctly, either missing it completely, or repeated the common attributes in the subtypes again.

A large number of candidates did not attempt part b). Those who did attempt it generally produced a good answer.