BCS HIGHER EDUCATION QUALIFICATIONS

Level 5 Diploma in IT

September 2011

EXAMINERS' REPORT

OBJECT ORIENTED PROGRAMMINGSeptember 2011

Examiner's Report Object Oriented Programming September 2011

General Comments

There were a number of candidates attempting this paper who received very low marks. 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. In addition to studying the material explicitly listed in the syllabus, candidates will benefit from practical experience of programming in an object oriented language.

Once again the examiners would like to ask future candidates to clearly mark their choice of questions on the front of their answer books. Similarly, candidates should ensure that they answer Sections A and B in the appropriate answer books.

Question A1

a) Describe FIVE features of the UML and explain how they are used in the design of object oriented software.

(10 marks)

Answer Pointers

Any features accepted but could include:

Use case diagrams

> A graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases.

Class Diagrams

A static structure diagram that describes the structure of a system by showing the system's classes, their attributes, and the relationships between the classes

Object Interaction Diagrams

Diagrams which emphasize the flow of control and data among the things in the system being modelled

Object State transition Diagrams

Depict the various states that an object may be in and the transitions between those states.

Object Constraint Language

A declarative language for describing rules that apply to UML models

b) Describe the feature of the UML that allows a designer to describe rules which apply to the models developed as part of a design exercise.

(5 marks)

Answer Pointers

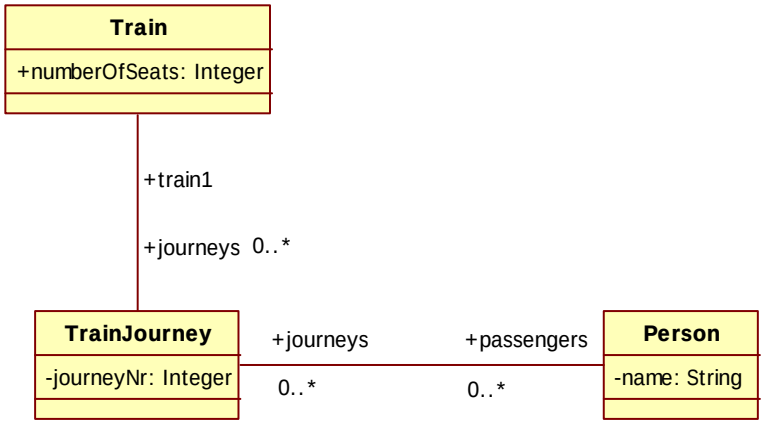
This is Object Constraint Language (OCL)

OCL supplements UML diagrams by providing expressions that have neither the ambiguities of natural language nor the inherent difficulty of using complex mathematics. The Object Constraint Language is the formal language for modelling with the UML. It provides the unambiguous expression that UML diagrams alone cannot provide.

c) Give a simple example of a UML class diagram and a rule which clarifies the semantics of the situation being modelled.

(10 marks)

Answer Pointers



The following OCL

context TrainJourney

inv: passengers->size() <= train.numberOfSeats

states that:

On any TrainJourney the number of passengers will not exceed the number of seats on the Train.

Examiners Comments

This question examines Syllabus 8c: Design

This was the least popular question on Section A of the paper with 58% of the candidates attempting it. The majority of answers were very poor. Part a) invited the candidates to reiterate book work, however, many candidates knowledge of the UML was limited to Use Case and Class Diagrams. Object Constraint Language (OCL) is an aspect of UML specifically set out in the syllabus but one with which the majority of candidates appear to be unfamiliar. Part c) could have been answered (as shown here) with material drawn from previous examination papers, however, only a few candidates attempted this part of the question.

Question A2

a) There are two techniques which are commonly used to test a class, one of which uses prior knowledge of the structure of the code and the other which is based on an understanding of the specification of the tasks the class is expected to accomplish. Explain how each of these testing techniques is carried out.

(10 marks)

Answer Pointers

The two techniques are white box testing and black box testing.

In white box testing the programmer designs test data on the basis of knowledge of the code forming the method. The aim of the test data is to test every possible route through the method. Consequently, the test data is chosen on the basis of the decision branches in the code.

In black box testing the test data is determined by an examination of the specification of the method. No assumption is made about the way in which the code is written. From the specification the programmer derives boundary values and equivalence partitions. The test data consists of values from each of the equivalence partitions and the boundary values. The programmer predicts the expected outcome for each of the values and runs them through the code in order to check that the actual value matches the expected value.

b) After individual classes have been tested using one or both of the techniques referred to in part a) they are often combined and tested as a group. Describe two approaches to this form of testing.

(10 marks)

Answer Pointers

Integration testing is a technique for testing assemblages of classes which will previously been tested individually.

In big bang integration testing, the majority of the components of the system are put together. This is a very quick way of building the system but requires careful design of the test data.

In bottom up testing the components at the lowest levels of the hierarchy are combined and tested first. The software is then put together by including successfully higher level components. In contrast in top down testing the higher levels are integrated first and tested. Initially lower levels will be stubs but as testing continues the stubs are replaced by actual classes.

c) Discuss the limitations of testing.

(5 marks)

Answer Pointers

The assumption of testing is that the programmer will submit a set of test data which is sufficiently inclusive so as to fully test the program but this may not be the case. If the test data submitted by the programmer shows a variance between the predicted and the actual output, this is a bug which will be addressed. There are many programs however which are expected to process an infinite number of acceptable inputs. The programmer can only supply representative samples of these inputs and therefore it is possible that undiscovered problems will remain. In essence, testing can only show the presence of bugs, it cannot prove the absence of bugs.

Examiners Comments

Candidates were generally able to distinguish between black and white box testing, however, details of what constitutes these techniques was less well known. The majority of candidates were aware that part b) referred to integration testing but only a few were able to distinguish between the top down and bottom up approaches. Disappointingly, part c) was very poorly answered even though it is extensively covered in text books which address the issue of testing.

Question A3

a) Explain the following terms:
   i) *Structured programming*;
   ii) *Modular programming*;
   iii) *Abstract data types*;
   iv) *Typed languages*;
   v) *Untyped languages*.

(10 marks)

Answer Pointers

Structured programming is a type of programming methodology where a complex problem is decomposed into tasks which can be modelled by simple structures such as for…next, if….then and while…do

Modular programming is a technique for decomposing a programming problem into simpler task which together can be composed to form the solution.

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

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

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

b) Explain how the following aspects of software design can contribute the overall quality of an object oriented program:
   i) The degree to which each class relies on another class;
   ii) The degree to which each part of a class is associated with each other part;
   iii) The gathering together of related aspects of a design, grouping them and hiding the implementation.

(9 marks)

Answer Pointers

Coupling refers to the degree to which each object relies on each other object. Coupling can be "high" or "low". Low coupling means that one object does not have to be concerned with the internal implementation of another module. Low coupling is a sign of a well structured computer system.

Cohesion refers to the degree to which each part of an object is associated with each other part, in terms of functional relation. Parts of an object are functionally related if each part is essential to the functionality and the interface of a well-defined object (a well-defined object is one that has a single task). Cohesion can be considered "high" or "low". High cohesion means each part

of an object is functionally related, and low cohesion means some aspects of an object is not. High cohesion is considered to indicate better design.

Encapsulation supports the notion of gathering together related aspects of a design, grouping them and hiding the implementation. It therefore encourages high cohesion. The combination of information hiding with a well defined set of interface methods also supports low coupling as it prevents a component of the design gaining access to the internals of an implementation

c) Explain the statement “Objects belong to one class but may conform to more than one type”.

(6 marks)

Answer Pointers

If we have

```
class MyClass {

}

class MySubclass extends MyClass {

}
```

and the following declarations

```
MyClass x = new MySubclass();

MySubclass y = new MySubclass();
```

In each of these declarations an object of class MySubclass is created. Since, however, such an object conforms to type MyClass and MySubclass, both statements are syntactically correct even though x and y .are of different type.

Examiners Comments

This was the most popular question in Section A with 79% of candidates attempting it. Unsurprisingly, as this question has been asked in a number of previous papers, there were a number of good answers. Most candidates could describe structured and modular programming, a smaller number were able to describe ADTs, typed and untyped languages. In part b) most answers produced good descriptions of coupling and cohesion, however, the relationship between encapsulation and these measures of program quality were less well understood. Although part c) has been asked in a number of previous papers, candidates do not

understand this concept which is a key aspect for the provision of polymorphism in a typed language.

Question B4

The class diagram in Appendix A represents *a Student Enrolment System* that records what courses a student takes, who their current tutor is and a record of any fees paid.

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

(15 marks)

Answer pointers

The diagram has the following classes:

- Person

  This is an abstract class with the instance variables personID, name, address and dateOfBirth. Person has two subclasses:

- Lecturer
  Lecturer has the instance variables highestQualification and dateHired. dateHired.
- Student

  Student has the additional instance variable emailAddress and noOf Students. noOfStudents is a class variable, which has a default value of 0 when the object is instantiated.

There are three other classes:

- Course
- School
- Fee History

Course has the instance variable courseCode and courseName.

School has the instance variables schoolCode and schoolName.

FeeHistory has the instance variables invoiceNo, payment and datePaid.

There are a number of associations in the classes:

WorksFor between Lecturer and School. Each Lecturer can work for 1 School and 1 School only. A School must have at least 1 Lecturer working in it and can be many.

Between Student and Course, this has an association class called Enrolment, which has the instance variables dateStarted, dateFinished and classification.

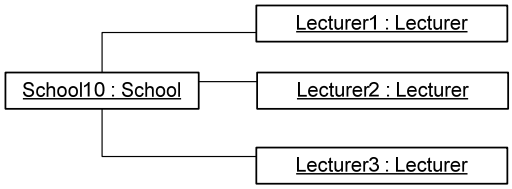
A student studies up to a maximum of 100 courses, whilst a course is taken by 1 or many students.

Pays For between Student and Fee History. The association is one-way and can only be seen from Fee History

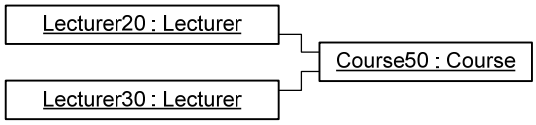
b) Given the object diagrams below (i-vi), state which are legitimate instances. If an object diagram is not legitimate explain why not.

(10 marks)

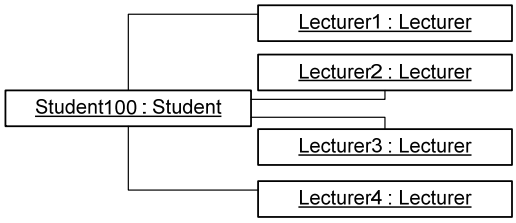
i



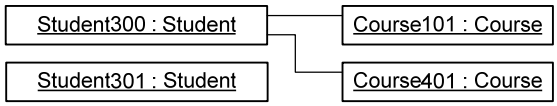
ii



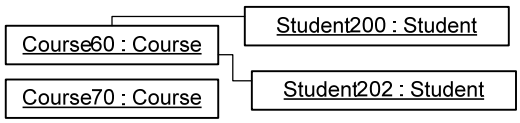
iii



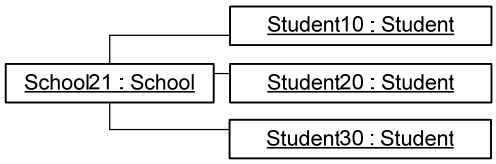
iv



v



vi



Answer pointers

Only i) and iv) are correct.

i) Ok
ii) No link between lecturer and course
iii) Lecturer instances exceeds number of tutors a student can have
iv) Ok
v) Course must be associated with a student
vi) No link between student and school

Examiners Comments

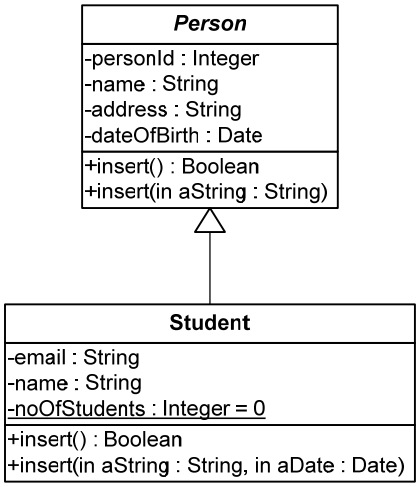
This question examines Syllabus 8c: Design

Over 65% of the candidates attempted this question and over 60% passed. Most candidates identified the correct classes in part a, but lost marks by then not describing the properties in any detail. Better answers identified the associations correctly between the classes, in particular the association class and the one-way association between Student and Fee History. Overall, most candidates made a good attempt at this question.

Candidates lost marks in part b by not saying why the object diagram was incorrect, or wrongly identified which diagrams were correct.

Question B5

The following diagram is a development of part the design in Appendix A:



a) Both classes now include a definition for some methods called **insert** and have an attribute called **name**. Identify the scope of each method and any issues or problems arising from this design.

(10 marks)

Answer pointers

The insert method has been overloaded in both classes: insert with 1 parameter can been seen in both classes, whilst insert with 2 parameters is only available in Student.

The subclass Student has also provided a redefinition (overriding) for insert(). This is in keeping with the designer seeking to deploy dynamic binding of this method and is typical of specialisation architectures.

The re-introduction of the attribute name in the subclass Student might be considered an error. If permitted, this would mean that an instance of Student would have two name attributes as part of its state. Further, methods in the subclass can reference the name variable defined by Student but not that defined in the superclass.

b) Describe in detail **THREE** design patterns with which you are familiar. Your answer should include the circumstances in which they are applicable, when they can be applied and the trade-offs when using them within a larger design.

(15 marks)

Answer pointers

The candidates can choose any design pattern, which can be 3 of the following:

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.

Decorator (Structural)

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.

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.

Observer (Behavioural)

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.

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.

Abstract factory

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

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.

For each category, the student should give a short description of one of the diagrams, include a simple example and say when they should be used.

Examiners Comments

This question examines parts 8c & 8d of the syllabus Design and Practice

This question was less popular, with 35% attempting it and over 36% passing it. In particular part a was poorly attempted, or often not at all. In part b, most candidates could describe at least two design patterns, but often failed to say in what circumstances they were applicable, when they could be applied and what the trade-offs are when using them within a larger design. This was needed in order to gain full marks.

Question B6

Universal Parcels is a company that specialises in the delivery and collection of parcels for business customers. To use their services, a customer must first register for an account. The customer needs to provide their company name, address, telephone number and the name of a main contact for any queries.

As part of the registration process, the customer will have to decide if they wish to pay monthly on receipt of an invoice, or via credit card for each delivery made. If paying by credit card, then the card details are also required. Once these details have been accepted, the customer will be issued with an account number that they must quote when contacting the company.

When a customer requires a parcel to be delivered, they will contact *Universal Parcels* to arrange collection. The customer needs to provide details of where the parcel will be collected from; where it will be delivered to; how many parcels are to be collected and which type of service they want, for example, next day delivery.

Once the collection has been arranged, an *Airway Bill* will be generated. The details on this will be used by a *Dispatcher* to schedule the vehicle needed for the collection. Each parcel will be given a priority number by the *Dispatcher* and those with the highest priority will be collected first.

By 12 noon each day, the *Dispatcher* also needs to generate a delivery schedule to ensure all the parcels are delivered according to the service required.

Each *Driver* has a mobile device with a copy of the *Airway Bill*; a person at the delivery address must sign this to say that the parcel has been delivered. This will flag that the delivery has been completed.

Once the parcel has been delivered, if the customer pays via credit card, their card will be debited by the amount required, or if they pay monthly, then the invoice account will be debited. Once a month, the *Finance Department* will generate the invoices for payment.

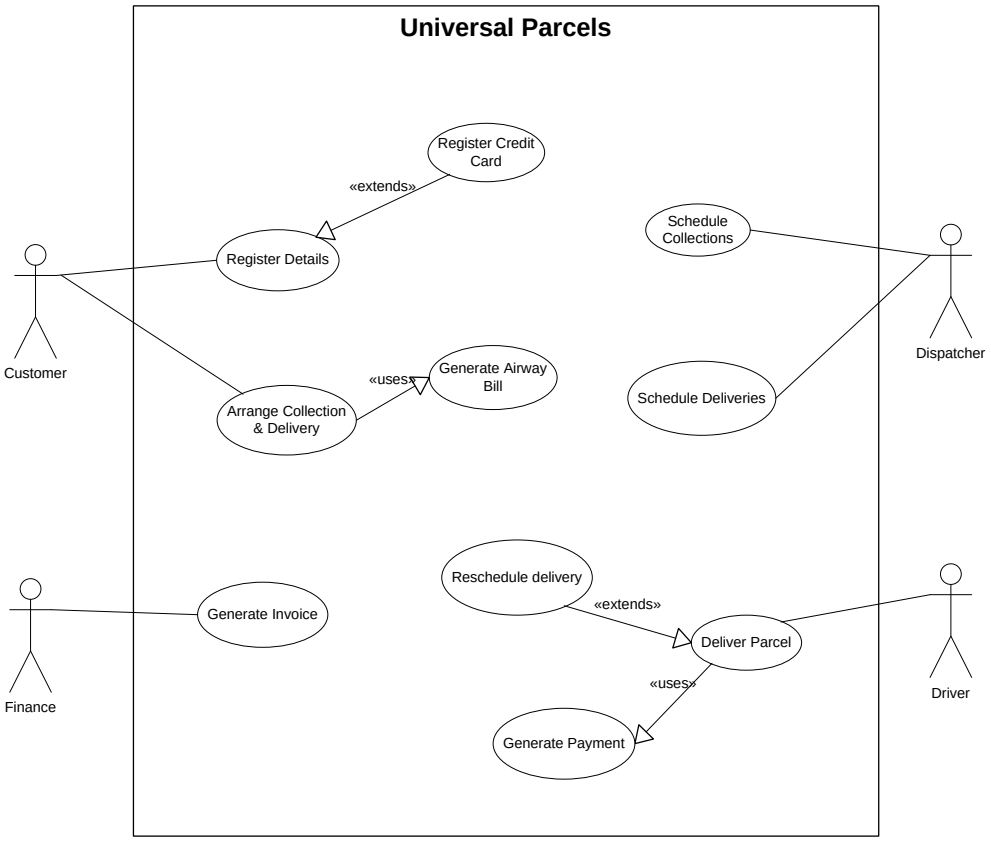
If the parcel cannot be delivered for any reason, it will be returned to the *Depot* and a card will be left with at the delivery address with details of how to arrange re-delivery.

a) Draw a Use Case diagram for this system.

(15 marks)

Answer pointers

Sample Use Case diagram:



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

Answer pointers

The candidate should discuss where Use Cases should be used in developing an object-oriented 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 the following areas:

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

The candidate should include a brief example of a scenario to illustrate their points.

Examiners Comments

This question examines parts 8c of the syllabus Design

This proved to be a popular question, with over 85% of the candidates attempted this question, and over 60% passing. Part a) involved drawing a use case diagram for the Universal Parcels

system. The majority of candidates answered this part well and obtained good marks, since they could identify the main actors and use cases. Some candidates could not distinguish the use cases, instead including all the actions, which led to a much cluttered diagram. Lower marks were given to candidates who missed some of the use cases and actors; did not connect them to the correct actors or connected the Extends/Uses relationships inappropriately.

A very small minority provided the wrong type of UML diagram, such as a class or sequence diagram, which were not appropriate and so lost marks.

Part b was less well answered, with some candidates unable to discuss the role of Use Cases in the development of an object-oriented system.

Appendix A

Class diagram used in Questions 4 and 5