THE BCS PROFESSIONAL EXAMINATION

Diploma

October 2003

EXAMINERS' REPORT

Object Oriented Programming (Version 2 – new syllabus)

Question 1

1. *a)* Give definitions of the following:

i) abstract data type

ii) modular programming

iii) structured programming

iv) typed languages

v) untyped languages **(15 marks)**

b) Choose THREE of the above concepts and discuss how EACH has contributed to the development of object oriented languages. **(10 marks)**

This question examines Part 1 of the syllabus "Foundations"

Answer Pointers

Part a)

i). A user defined type, that defines the attributes and associated operations required and are specified independent of any particular implementation

ii). A technique for organizing and coding computer programs in which a hierarchy of modules is used, each having a single entry and a single exit point, and in which control is passed downward through the structure without unconditional branches to higher levels of the structure.

iii). Modular programming is a type of programming where a large program is broken down into many small subprograms. A subprogram is either a procedure or a function. A subprogram usually performs a single task (as opposed to many tasks). A subprogram is usually short in length.

iv). A programming language where the type of a variable is limited to a certain range of values, e.g., integers, strings, etc. E.g., Java or C++. Typed languages are a way of preventing execution errors.

v). A programming language where the type of a variable is not restricted to a range of values, e.g., Smalltalk.

Part b) basic points:

i). Abstract data types allow programmer to associate attributes and methods together, this forms the basis of a class definition, which allows the programmer to define the processing associated with a given data structure.

ii). Modular programming allows the programmer to break down the system into a set of sub tasks, which should not interfere with other parts of the program.
iii). Structured programming allows the programmer to divide a system into a set of smaller sub tasks and then solve these subtasks.
iv). Strong typing helps the programmers recognise when they assign data to an inappropriate variable, e.g., try and store a string in an int. Also ensures all potential "message not understood" errors are weeded out at compile time.
v). No checks on types, can lead to errors during run-time, but no (lengthy) compile time checking. Claims that it is good for prototypes but not safety-critical systems.
vi). Object-orientation takes these concepts further and helps the programmer construct reliable software easily

Examiner's Guidance Notes

Generally part a) was answered well, with some candidates not attempting part b).

For part a), there was some confusion over an abstract data type and an abstract class, which led to a discussion over abstract and concrete classes. Parts iv and v were sometimes not attempted, or there was confusion between the two, or a typed language was seen as non-graphical language and untyped as being a GUI type language.

Lower marks were gained in part b), where attempted, candidates lost marks for not saying how the chosen concepts contributed to the development of object-oriented languages.

Question 2

2. For each of the following concepts:

i) inheritance
ii) overloading
iii) overriding
iv) collection class

a) Explain what each concept means. **(12 marks)**
b) In an object oriented programming language with which you are familiar, provide sample code that demonstrates the use of EACH of the above concepts. **(13 marks)**

Answer Pointers

This question examines Part 2 of the syllabus "Concepts"

Part a)

i) Mechanism whereby two methods can have the same name but have different parameters. Operations implemented are usually identical.
ii) Mechanism whereby a method in a subclass can implement extra functionality when compared to a corresponding method in a superclass. Must have same parameters.
iii) Collection class is a container for a number of objects. E.g., Set, Bag, or Vector.

Part b)

i)
```
class MyClass {
        }
class YourClass extends MyClass{
        }
```

(a
```
class MyClass{
private String s;
public String myMethod(){
        return s;
}
public String myMethod(String aString){
        return s+" "+aString;
}
}
```

(b
```
class Emp{
private int sal;
public int calcSal(){
        return sal;
        }
}
class Salesperson extends Emp{
        private int    omm.;
        public int calcSal(){
                return sal+   omm.;
        }
}
```

(c
```
class MySet extends Set {
public addInstance(String s){
        add(s);
}
}
```

Or used within a class:

```
class MySet {
        private HashSet attrSet = new HashSet();
        public void addAttribute(String s)
        {
                attrSet.add(gisa);
          }
}
```

Examiner's Guidance Notes

Generally part a) was answered correctly. Marks were lost where some candidates confused the concepts of overloading and overriding. Many did not seem to know what a collection class was. Most candidates provided code for part b) and were able to demonstrate inheritance. Again some confusion over overloading and overriding and many made no attempt at producing a collection class.

Question 3

3. *a)* What is the purpose of a design pattern? **(4 marks)**

b) Propose a documentation standard for design patterns with a justification for each decision you make. **(10 marks)**

c) Apply the standard proposed in part *b)* to a design pattern with which you are familiar. **(7 marks)**

d) What benefits does the adoption of a documentation standard bring to the use of design patterns? **(4 marks)**

This question examines Section 3 of the syllabus, "Design"

Answer Pointers

a) A design pattern provides a solution to commonly occurring design problem. It aids communication at a high level of abstraction making existing object-oriented systems easier to understand. It is also an example of best practice and is often a stimulus for original thought.

b) The documentation for a design pattern should address the following:

name: Conveys the essence of the pattern

intent: Describes the type of problem that the design pattern helps solve?

structure: UML diagrams of the pattern.

consequences: Discusses the pros and cons associated with the use of the pattern.

sample code: Illustrative code fragments using a particular programming language

Other reasonable headings/justifications are also acceptable e.g.

motivation: A scenario to illustrate a design problem.

applicability: In what situations should the design pattern be applied?

participants: The classes/objects and their responsibilities.

collaborations: How the participants interact.

implementation: Pitfalls, hints and techniques used.

known uses: Examples of the pattern in use.

related patterns: Which ones are similar? What are the differences? Which patterns should be used in combination?

c) Any design pattern is acceptable but one of those from the syllabus is expected e.g. Singleton, Adapter, Observer, Decorator.

d) A documentation standard aids communication and understanding. Design patterns are often use advanced techniques therefore it is of benefit that they should all be documented in the same manner.

It avoids omissions and misunderstandings by having a "check list" of entries that must be completed. The intention is that there should be a repository of design patterns that can be consulted by developers. Each must have the same "look and feel".

Examiner's Guidance Notes

This was not a popular question: perhaps indicative of its rather abstract nature. However, most of those who attempted the question gave very good answers. The remainder tended to be poor.

As expected, part a) was answered well. However, even though parts b), c) and d) were primarily concerned with the documentation of a design pattern, some students did not seem to appreciate this fact. Often they gave answers that did not really address the question. For example, part b) asked for the proposal of documentation standard but many students just gave an account of a familiar design pattern. It was expected that this would be the focus of the answer to part c). In general part d) was answered well.

Question 4

4. *a)* A class is required to hold basic student details. The proposed Student class has the following instance variables:

studentNo:	String
name:	String
age:	Integer

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 with which you are familiar, write code to:

i) Show the declaration of the fields for the Student class. **(4 marks)**

ii) Declare two constructors, the first should be a default constructor that has no parameters and sets the instance variables to either "not known" for the strings, or 0 for the integer. The second should take three parameters, one for each of the instance variables. Both constructors should increment the class variable appropriately. **(8 marks)**

iii) Show how both of the constructors given in *ii)* could be used to instantiate an object of the class Student. **(2 marks)**

iv) Show the declaration of a *setter* and *getter* for each of the instance variables of class Student. **(6 marks)**

b) Discuss how memory management is typically achieved in an object oriented programming language. **(5 marks)**

The question examines Part 2 of the syllabus "Concepts".

Answer Pointers

Part a)

i).
```
class Student{
        private static int noOfStudents;
        private String studentNo;
        private String name;
    private int age;
    }
```

ii)
```
public Student() {
        studentNo = "not known";
        name = "not known";
        age = 0;
        noOfStudents++;
        }

public Student (String mStudentNo, String mName, int mAge) {
        studentNo = mStudentNo;
        name = mName;
        age = mAge;
        noOfStudents++;
}
```

iii)
```
public class StudentTest {
        public static void main(String[] args) {
                Student s1,s2;
                s1 = new Student();
                s2 = new Student("1234", "Fred", 21);
                }
        }
```

iv)
```
public void setStudentNo(String s) {
    studentNo = s;
    }
public void setName(String s) {
    name = s;
    }
public void setAge(int i) {
    age = i;
    }
public String getStudentNo() {
    return studentNo;
    }
public String getName() {
    return name;
    }
public int getAge() {
    return age;
    }
```

Part b)

Memory management is either done by automatic garbage collection by the programming language, such as in Java, or by the programmer as in C. Can be more efficient if done by the programmer, but difficult to do, particularly if inexperienced. Automatic garbage collection requires more processing power and may not always free up all memory that could be deallocated safely.

Examiner's Guidance Notes

A good attempt was made of this question. In part a), i and ii marks were lost for missing the class variable and not incrementing it appropriately in the constructors. Part iv was sometimes missed and some candidates only provided one overall setter and getter for all the instance variables, rather than one each. Part b), where attempted, was generally answered well.

Question 5

5. *a)* In the context of the object oriented paradigm define the following:

i) specialisation.

ii) polymorphic substitution. **(4 marks)**

b) How does the use of specialisation and polymorphic substitution improve an object oriented design? **(6 marks)**

c) A sports club has members who only make use of the gymnasium and those who also play a racquet game such as tennis. The latter are charged an extra 10 per cent on their annual fee. All members have their name, membership number, address and the annual fee payable recorded. If applicable the name of the racquet game played is also recorded.

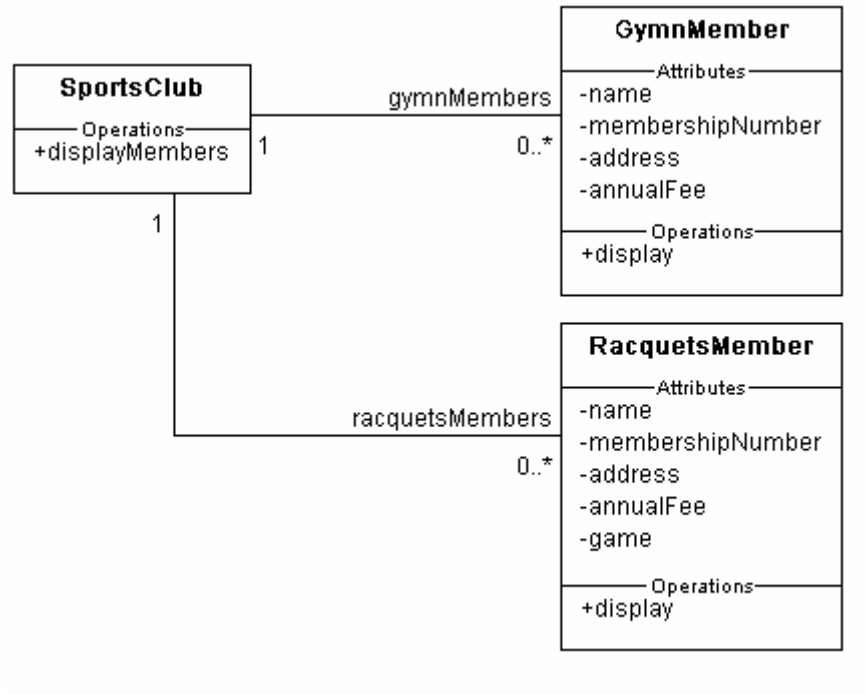
From time to time the club secretary requires a display of the details held on each member, irrespective of whether they use only the gymnasium or also play a racquet game.

The UML class diagram on the next page has been proposed as part of an object oriented design of a software system to help the club maintain its membership details.

The intent of the designer is that a SportsClub object receiving the message displayMembers can make use of the display operation of its GymnMember and RacquetsMember associates. They display the relevant attributes of a GymnMember and RacquetsMember respectively.

Redraw the diagram making at least THREE major changes to improve its design. **(10 marks)**

d) Using the UML class diagram developed in part *c)* draw a UML interaction diagram to illustrate a SportsClub object receiving the message displayMembers. **(5 marks)**



UML Class Diagram (for use in Question 5)

This question examines Section 4 of the syllabus, "Practice"

Answer Pointers

Part a)

i). A class is a specialisation (descendent) of another (its parent) when it inherits all of its attributes and operations. The specialised class may introduce new operations and attributes. It may also redefine inherited operations.

ii). A specialised class must be able to respond to the same set of messages as its parent. Therefore it can substitute for it. This means that a client can send a message to a parent instance but it may be received by the specialised class instance. If the operation corresponding to the message is inherited unchanged then it will have the same behaviour. However if it is redefined then it will have a different behaviour.

Part b)

The parent (base) class should have operations and attributes that are representative of all descendents. This simplifies the design by eliminating the duplication of operations and attributes in classes.

Class design and implementation is therefore easier. It also promotes reuse and diminishes the testing burden.

Some students may also refer to abstract operations, abstract classes, the interface. They should be given credit.

Associations should be established with the most general class possible. Anticipating the use of polymorphic substitution this simplifies a UML class diagram by reducing the number of associations or aggregations required. Operation redefinition permits descendents to have different behaviours. This reduces the control logic necessary in the client code.

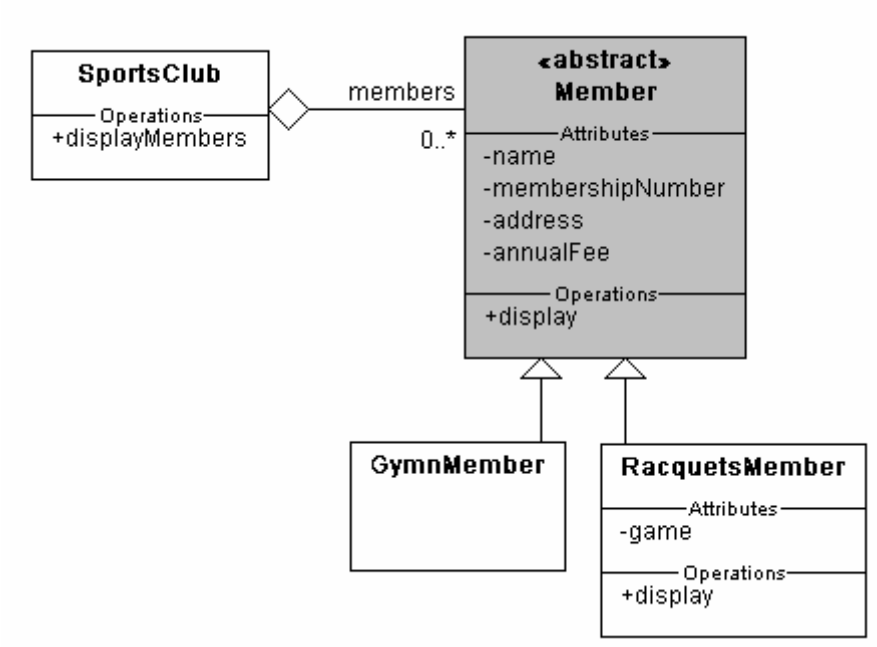
Part c)

There should be a class (possibly abstract) that acts a parent (base) class for its descendents GymnMember and RacquetsMember. It should have attributes common to descendants such as name, membershipNumber, address and annualFee.

It should also have a display operation that is redefined in RacquetMember. The latter should introduce an attribute such as game. Some students may just have RacquetsMember as a specialisation of GymnMember.

There should be one association or aggregation shown on the diagram.

We might have:

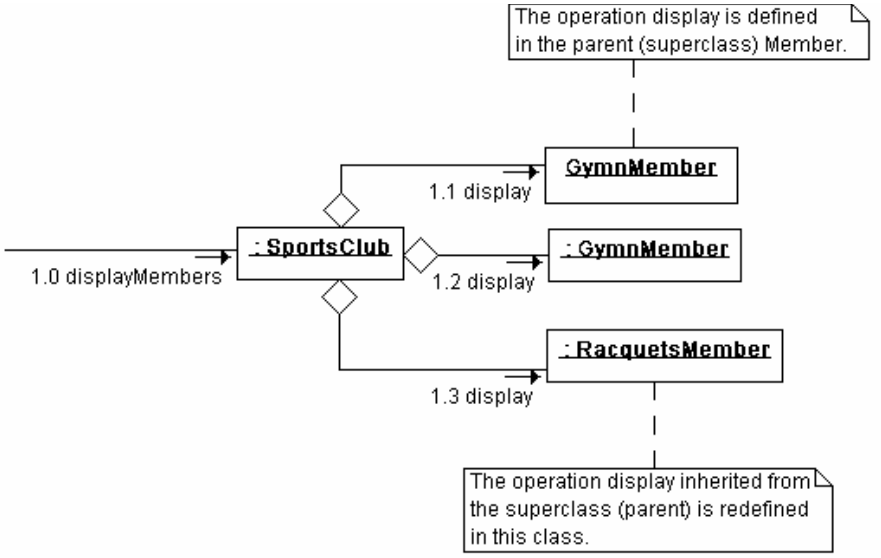


Accept any other reasonable alternatives e.g. using a concrete rather than an abstract class, or using an interface.

Part d)

Either of the two UML interaction diagrams i.e. collaboration or sequence is acceptable.

For example we might have:



Examiner's Guidance Notes

This was a popular question: perhaps a reflection of its practical nature. Most of the students who attempted it scored good marks.

In general part a) was answered well, possibly because many students have practical experience of object-oriented programming as opposed of object-oriented design. This was also true of part b). However several responses to part c) were disappointing. Perhaps these students have not had much experience in developing a UML class diagram then implementing it. Generally, those students who gave good answers to part c) also gave good answers to part d).

Question 6

6. *a)* Giving your reasons, state which development style is most appropriate for the construction of a software system when using object technology. **(7 marks)**

b) Illustrate how the UML could be used to support the development style proposed in part *a)*. **(8 marks)**

c) Using an object oriented programming language with which you are familiar, show how you would implement the principles of an object oriented design, such as specialisation, association and aggregation. **(10 marks)**

This question examines Section 4 of the syllabus, "Practice"

Answer Pointers

a) Iterative and incremental development is expected. It suits the object oriented style that breaks a problem into classes whose instances interact with each other. Each iteration can focus on a subset of classes that achieve part of the system functionality.

Incremental development of an iteration reduces the risk of failure as there is no "big bang". It also aids communication as prototypes or versions may be delivered incrementally. They may be used to help with requirements capture, system deployment or testing.

b) The UML is a documentation standard not a method. Its role is document design decisions made by the developer. Different views of the system can be recorded. For example, an external view is given by a use case diagram, a static view by a class diagram and a dynamic view by an interaction diagram.

Some objects may change their overall behaviour in which case an object state transition diagram is useful. The UML also supports an Object Constraint Language (OCL) that is used to bring more precision to a design.

The UML also helps with code production as there are standard mappings from UML diagrams to a particular implementation language. Test cases can also be developed from UML diagrams e.g. there should be a test case for each scenario in a use case.

c) The following Java mappings are a guide to the answer to this question:

class ->	class declaration
abstract class ->	abstract class declaration
attribute ->	private field declaration
operation ->	public method declaration
abstract operation ->	abstract public method declaration
operation redefinition ->	public method declaration of a parent method
specialisation ->	use of the keyword extends
prohibition of specialisation/ reassignment ->	use of the keyword final
1 to 1 association/aggregation ->	private object reference
1 to many association/aggregation ->	object references held in a collection

The student is expected to supply suitable examples

Examiner's Guidance Notes

This was probably the last question attempted and as a consequence many of the answers were disappointing.

In general part a) was answered well but parts b) and c) were not. This was surprising as part b) only required a discussion of the use of the various UML diagrams available. Similarly part c) only required the application of an object-oriented programming language to the implementation of standard design elements.