BCS THE CHARTERED INSTITUTE FOR IT

BCS HIGHER EDUCATION QUALIFICATIONS
BCS Level 5 Diploma in IT

Object Oriented Programming

Examiner's Report March 2017

A1.

a) Explain what is meant by the following terms:

(i) Structured programming;
(ii) Procedural programming;
(iii) Abstract data types;
(iv) Typed language;
(v) Untyped languages.

(15 marks)

b) Define the terms *coupling* and *cohesion* in the context of object oriented programming. Within your discussion, explain how *coupling* and *cohesion* can lead to either good or bad software design.

(10 marks)

Answer Pointers:

a)

Structured programming is a programming methodology where a complex problem is decomposed into tasks which 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.

Abstract data types (ADTs) 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.

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.

b)

Cohesion refers to what a class will do. Low cohesion means the class does a lot of actions, whereas high cohesion will mean the class focuses on what it should be doing. For example, given a Student class with two attributes: name and DOB, high cohesion would only include methods appropriate to the intention of that class, such as getName, setName(), getDOB(), setDOB(), whilst one with low cohesion could include other methods appropriate to other classes, such as printResults(), setExamMark().

Coupling refers to how dependent classes are on each other. Low coupling would mean changing something in one class should not affect the other, such as changing the definition of a method. High coupling could mean code is difficult to change, so changes, or maintenance in one area could mean revamping the whole system.

Good design should go for high cohesion and low coupling.

Examiner Comments:

This question examines Syllabus Section 1 (Foundations)

A popular question answered by over 86% of the candidates, though only 40% passed. Most candidates could answer parts a i-iii, but were weaker on parts iv and v, where incorrect answers defined typed languages as being a language that was typed up line by line and

untyped languages were seen as being the opposite, such as those using a graphical user interface (GUI).

Those who attempted part b generally gave good answers. Weaker answers did not explain how coupling or cohesion could lead to good, or bad software design, or gave the wrong definition.

A2.

a) Explain what is meant by a *design pattern* in the context of object oriented programming.

(5 marks)

b) Describe in detail TWO *design patterns* that you are familiar with, stating the motivation for the pattern, including a UML class diagram for the pattern and an explanation of the classes which participate in the pattern.

(20 marks)

Answer Pointers:

a)

Design patterns are a concept originally found in architecture, however, from an object oriented point of view, they offer a general reusable solution to a common occurring problem, such as the singleton pattern.

The pattern is a description or template to how to solve a problem that can be used in different situations. Object oriented patterns also show the relationships between the classes.

b)

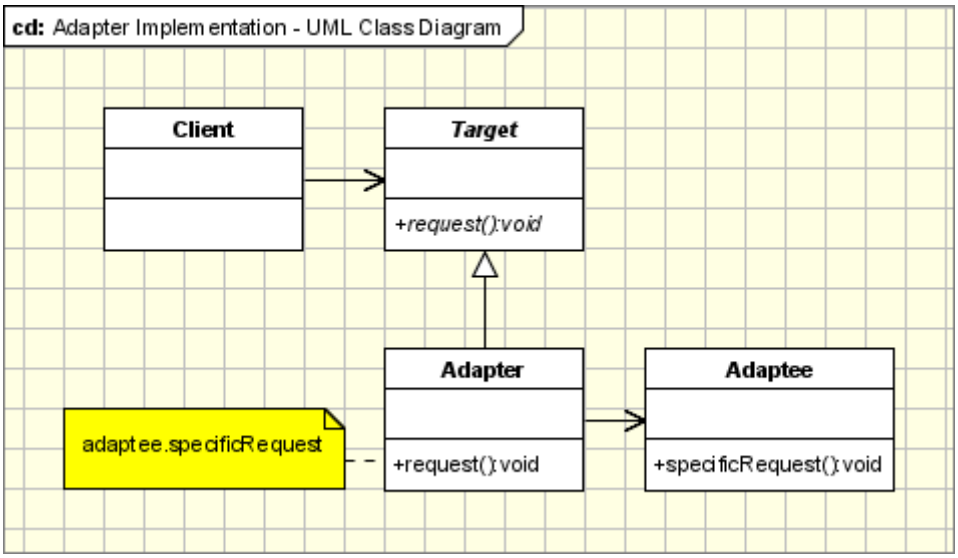
The candidate can pick two different design patterns that they are familiar with.

For example:

Adapter (Structural)

The Adapter design pattern provides 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.

Example UML class diagram (taken from: http://www.oodesign.com):



The classes/objects participating in adapter pattern:

- **Target** - defines the domain-specific interface that Client uses.
- **Adapter** - adapts the interface Adaptee to the Target interface.
- **Adaptee** - defines an existing interface that needs adapting.
- **Client** - collaborates with objects conforming to the Target interface.

Examiner Comments:

This question examines Syllabus Section 3 (Design).

This question was answered by 65% of the candidates and over 51% passed. Most candidates could explain what a design pattern was for part a. For part b, the most popular design patterns were the adaptor, singleton and iterator. Only two design patterns were required, candidates were not given any extra marks for describing several patterns and in such cases, only the best two got credit.

The answer required an example for full credit, which should have been illustrated using an UML diagram. Some candidates decided to explain what UML diagrams were instead, which was not part of the question.

A3.

a) Define what the following Object Constraint Language (OCL) concepts mean:

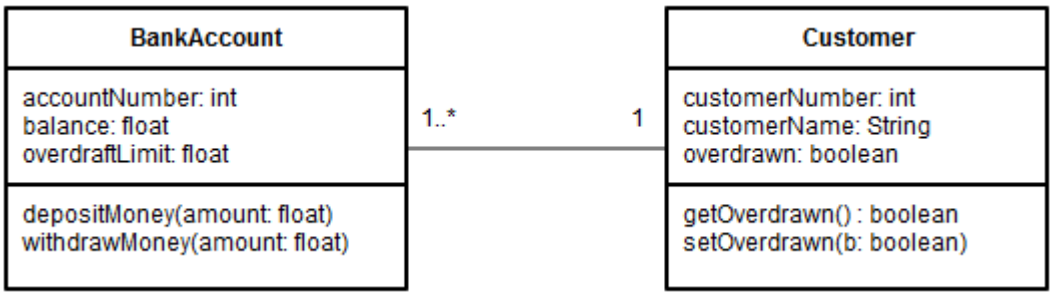
(i) Invariant;
(ii) Pre-condition;
(iii) Post-condition.

(9 marks)

b) Explain the role of the OCL in the development of an object oriented system.

(10 marks)

c) Given the following UML class diagram:



Explain what the following OCL statements mean:

context: BankAccount ::withdrawMoney(amount: float)
pre: amount > 0 and Customer.getOverdrawn() = false
post: balance = balance@pre - amount
if amount > (balance@pre + overdraftLimit)
then Customer.setOverdrawn(true)
endif

(6 marks)

Answer Pointers:

a)

Invariants

An invariant constraint consists of an OCL expression of type Boolean. The expression must be true for each instance of the classifier at any moment in time. Only when an instance is executing an operation, this does not need to evaluate to true.

Preconditions

The purpose of a precondition is to specify the conditions that must hold before the operation executes. A precondition consists of an OCL expression of type Boolean. The expression must evaluate to true whenever the operation starts executing, but only for the instance that will execute the operation.

Post-conditions

The purpose of a post-condition is to specify the conditions that must hold after the operation executes. A post-condition consists of an OCL expression of type Boolean. The expression must evaluate to true at the moment that the operation stops executing.

Taken from OCL Informative Documents (http://www.omg.org/spec/OCL/2.4/)

b)

A UML class diagram can capture constraints such as the multiplicity of an association, which can be fine grained to how many objects may take part in the relationship (some may use the bank example given, where an account is limited to either 1 or 2 people, such as a joint account). Constraints such as this can be captured graphically, but not all constraints can be expressed in this way. There may also be pre and post conditions that should be applied within an operation.

When more complex constraints are needed that cannot be expressed graphically, then something else is needed. A decision table could be used for informal constraints, but where something more formal is needed, OCL can be used. OCL has a precise grammar that allows unambiguous statements to be defined for the properties of the model. These can then be implemented in the language used for the object oriented system, such as Java.

c)

The OCL means, when a customer withdraws money:

- The precondition means the amount is checked to see if it is positive and the customer is not overdrawn.
- The post-condition means the balance is updated to subtract the new amount.

- A check is then made to see if the new amount is greater than the original balance, plus their overdraft limit.
- If so, the customer overdrawn status is set to true.

Examiner Comments:

This question examines Syllabus Section 3 (Design).

This question was answered by 45% of the candidates, with 45% passing.

For part a), most candidates could explain what pre and post conditions were, but a lot could not explain what an invariant was. To gain full marks some explanation was needed of each one, rather than just saying for example, a precondition was just a precondition and little further explanation.

Part b) was weakly answered overall, with little explanation of why OCL might be used in developing an object-oriented system. In many cases it was missed completely, whilst part c was answered well. Most candidates made a good attempt at defining what the OCL meant, gaining full marks. Some weaker answers described what the class diagram meant instead, which was not required.

B4. The atoms of different elements have different numbers of protons, neutrons and electrons. Electrons are negatively charged, protons are positively charged, and neutrons have no charge.

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

(i) fields for storing the numbers of `protons`, `neutrons` and `electrons` with appropriate visibility;

(ii) setter and getter methods for manipulating these fields, ensuring that the minimum value for `electrons` and `protons` is 1, and the minimum value for `neutrons` is 0;

(iii) a constructor that initialises new objects of `atom` to be the smallest element (Hydrogen), for which the number of `protons` is 1, the number of `neutrons` is 0, and the number of `electrons` is 1.

(15 marks)

b) Write a new method for the `atom` class called `isIon` that will return true or false, depending upon whether the atom is an ion. An atom is an ion if it is charged (i.e., if the number of electrons ≠ the number of protons).

(5 marks)

c) Write a new method for the `atom` class called `getAtomicMassNumber` that will calculate and return the atomic mass number of the atom. Atomic mass number of an atom (often denoted A) is defined as the number of protons plus the number of neutrons.

(5 marks)

Answer Pointers:

a)

```
public class atom {

    private int protons, neutrons, electrons;

    public boolean setProtons(int newProtons) {
        if (newProtons > 0) {
            protons = newProtons;
            return true;
        }
        return false;
    }

    public boolean setNeutrons(int newNeutrons) {
        if (newNeutrons > 0) {
            neutrons = newNeutrons;
            return true;
        }
        return false;
    }

    public boolean setElectrons(int newElectrons) {
        if (newElectrons > 0) {
            electrons = newElectrons;
            return true;
        }
        return false;
    }

    public int getProtons() {
        return protons;
    }

    public int getNeutrons() {
        return neutrons;
    }

    public int getElectrons() {
        return electrons;
    }

    public atom() {
        protons = 1;
        neutrons = 0;
        electrons = 1;
    }
};
```

b)

```
public boolean isIon()
{
  return electrons==protons;
}
```

c)

```
public int getAtomicMassNumber()
{
  return protons + neutrons;
}
```

Examiner Comments:

This question examines Syllabus Section 4 (Practice)

This question was attempted by 66% of candidates and was passed by 61%. In general, candidates were able to reproduce the syntax necessary to implement a class. Common mistakes for part a) included failing to specify the data type of the integer members of the class, failing to realise that getter methods need to have a return type, and having inappropriate visibility settings (e.g., public data). In parts b) and c), answers in some cases longer and more complex than necessary, and/or did not specify appropriate return types.

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 extends A
{
  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.

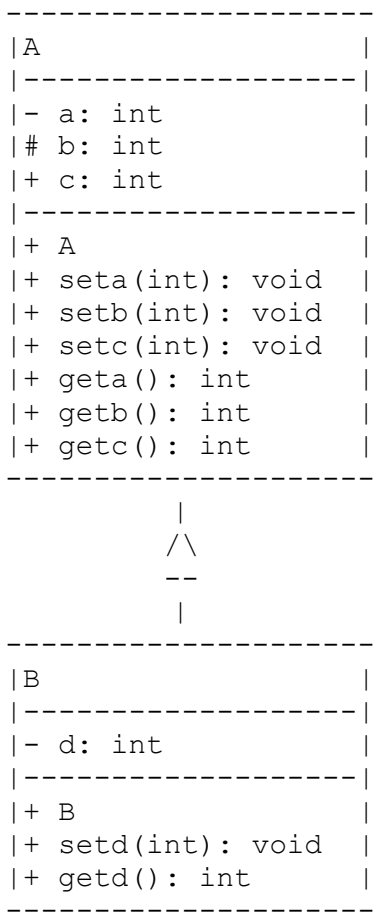
(10 marks)

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:

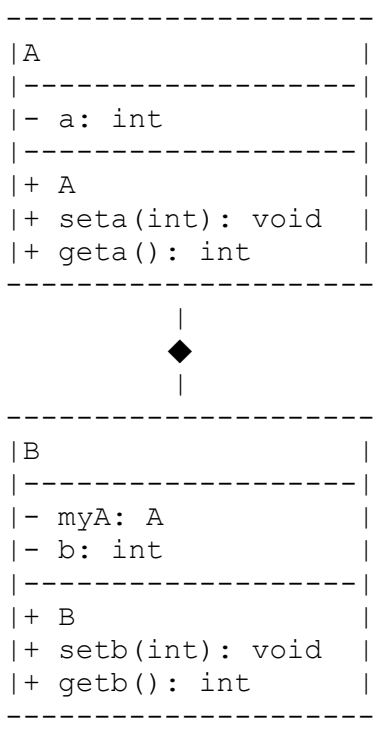
a)



1 mark for identifying correct relationship (inheritance, or *is-a* relationship)
1 mark for correct symbol between classes in UML (empty triangle)
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 & public fields
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
1 mark for correct arguments for setters

b)

Composition (*has-a* relationship). (1 mark)



(7 marks)

class B contains, as one of its members, an object of class A. Code fragment below:

```
public class A
{
  private int a;
  public A();
  public void seta(int newa);
  public int geta()
};
public class B
{
  private int b;
  private A myA;
  public B();
  public void setb(int newb);
  public int getb();
};
```

(7 marks)

Examiner Comments:

This question examines Syllabus Section 3 (Design)

This question was the second most popular in the exam. It was attempted by 85% of candidates but was only passed by 42%. In part a), most candidates correctly identified the relationship between A and B to be based on inheritance, and typically used the correct UML symbol. In some cases, the symbol for composition or aggregation was used, losing those marks. In many cases, the class member visibility was not correctly identified. Part b) was more of a problem, with many students presenting just another inheritance example, rather than the different inter-class relationship requested, meaning that few marks were scored.

B6.

a) Define *compile time polymorphism* and provide a code fragment that implements this concept in an object oriented programming language of your choice.

(10 marks)

b) Define *run-time polymorphism* and provide a code fragment that implements this concept in an object oriented programming language of your choice.

(15 marks)

Answer Pointers

a)

Compile-time polymorphism, also known as static binding, is polymorphism in which the exact code that will be run is determined at the time that the source code is compiled. This is the case where overloading is used.

One example of compile-time polymorphism is *method overloading*, demonstrated below:

```
public class A
{
  private int a;
  public A()
  {
    a=0;
  }
  public void increase()
  {
    a++;
  }
  public void increase(int amount)
  {
    a+=amount;
  }
}

class myTestClass
{
  public static void main(String args[])
  {
    A myA = new A();
    myA.increase();   // invoke first version of method
    myA.increase(10); // invoke second version of method
  }
}
```

b)

Run-time polymorphism, also known as dynamic binding, is polymorphism in which the exact code that will be run cannot typically be determined at the time that the source code is compiled but is decided while the program is executing.

One example of run-time polymorphism is *method overriding*, demonstrated below:

```
import java.util.Random;

public class A
{
  protected int a;
```

```
  public A()
  {
    a=0;
  }
  public void increase()
  {
    a++;
  }
};

public class B extends A
{
  public void increase()
  {
    a+=2;
  }
};

class myTestClass
{
  public static void main(String args[])
  {
    int randomInt = randomGenerator.nextInt(10);
    B myB;

    if(randomInt<5)
      myB = new A();
    else
      myB = new B();

    myB.increase();// Version run depends upon randomInt
  }
}
```

Examiner Comments

This question examines Syllabus Section 4 (Practice)

This question was attempted by 50% of candidates, 68% of which passed.

Many candidates mixed up run-time and compile-time polymorphism, stating (for example) that method overloading was run-time polymorphism. More candidates were able to provide a code fragment for a), using either method or (specifically) constructor overloading, although some also presented operator overloading. Part b) showed in most cases a broad awareness of the principal features of run-time polymorphism (i.e., that binding to a specific function occurs whilst the program is in execution), but the code examples in many cases were variants of method overloading or shadowing of inherited methods that did not truly show run-time polymorphism in action, meaning that fewer marks were scored. In some cases, for this question and question B4, it was obvious that candidates were not familiar or comfortable in

any object oriented programming language, highlighting the need to concentrate on practical as well as book-work in preparation for this exam.