Object Oriented Programming

Examiner's Report: September 2018 session

Diploma Object Oriented Programming

A1

a) Explain what is meant by:

i) iterative software development;

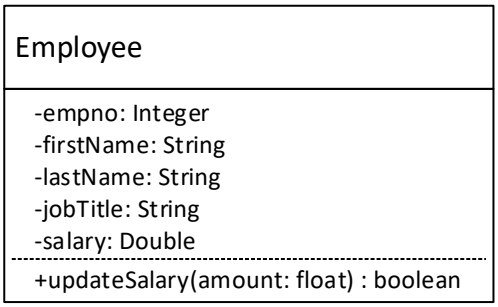
ii) incremental software development.

(9 marks)

b) Using examples from an object oriented programming language you are familiar with, discuss the suitability of object technology for iterative development.

(10 marks)

c) Using the following UML class diagram:



explain what the following OCL statement means:

```
context: Employee::updateSalary(amount: float): boolean
pre:     amount > 0
post:    if amount > (salary@pre * 1.1) then
             result = false
         else
             self.salary = amount
             result = true
         endif
```

(6 marks)

Answer Pointers:

a) Iterative development allows a large software system to be broken down into smaller chunks, the code is designed, developed and tested in repeated cycles. Incremental development allows the system to be designed, implemented and tested incrementally, allowing the developer to learn from the earlier, incremental, deliverable versions of the system. The system is finished when it satisfies all the requirements.

Iterative and incremental development basically means the system is developed in little steps, with the developer compiling and running the program after making each small step.

The advantage with both is that the developers do not have to write the program in one go and can introduce the system gradually to the users.

b) By using classes, object-oriented technology naturally splits an application into manageable units. These are the classes that go together to make the final product. Object technology results in software with clean interfaces that can be tested in isolation. In addition, once an interface has been constructed it is not necessary to produce all the code for a class at once.

   The candidate should include some examples from a programming language they are familiar with (e.g., an outline class definition).

c) The OCL checks that the amount is first of all greater than 0 (that is, no negative numbers), then that the amount is not more than 10% of the current salary. If so, result is set to false, otherwise the salary is updated with the amount and result is set to true. Basically if a salary increase is more than 10% of the original salary, it will be rejected.

Examiners Comments:

Question 1 (a) and (b) examine syllabus section 4 (Practice), part 4.1
Question 1 (c) examines syllabus section 3 (Design), part 3.3

This was not a popular question, with less than 38% of the candidates attempting it and only 25% passing it.

There is evidence that most candidates who attempted this question could answer part (a) well, however marks were lost in part (b) by giving general points rather than giving examples appropriate to object oriented technology. Part (c) was often not attempted; marks were lost by candidates who described the class diagram rather than the OCL statement. Others did not recognise *self* referred to the current object and mistook the setting of the salary value as an additional equality test.

A2 Choose five features of the object oriented paradigm that you consider to be important for good software engineering practice. Describe what they are, explain why you think they are important, and give examples of them with either with code or a diagram.

(25 marks)

Answer Pointers:

This is an open-ended question and the candidate could cover concepts such as the following:

- Inheritance
- Encapsulation
- Class
- Overloading
- Overriding
- Polymorphism
- Specialisation/generalisation

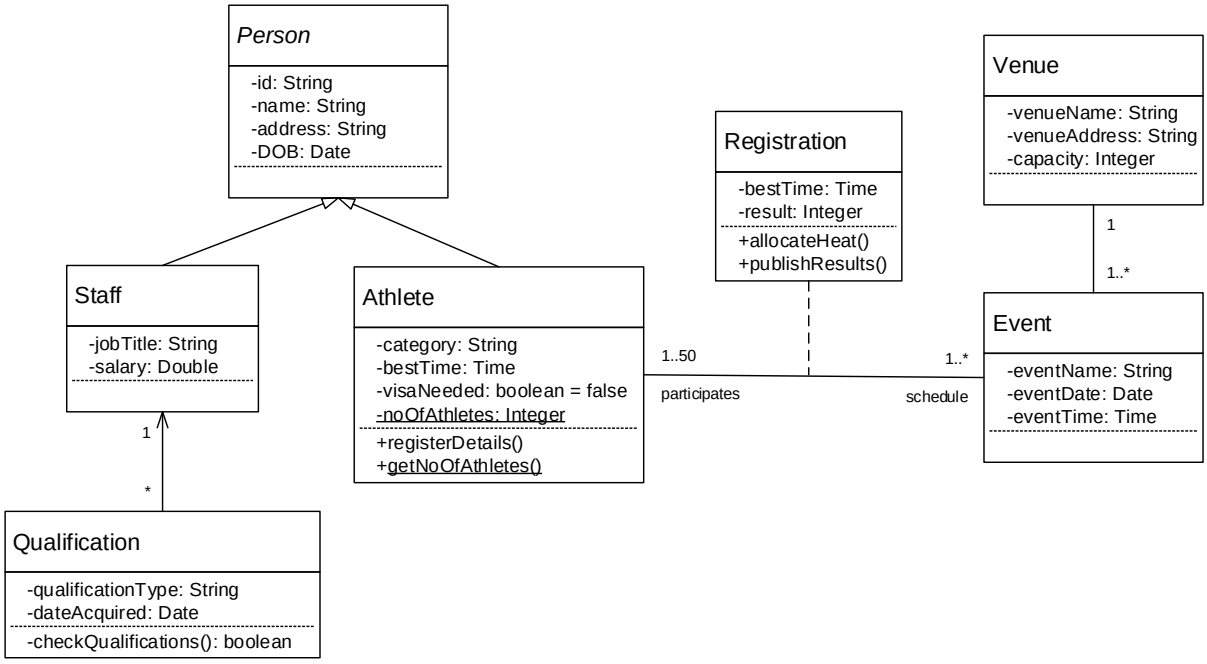
Other topics were given credit provided they were related to the object oriented paradigm. A definition was needed for each of the five concepts, with an explanation of why it is important for good software engineering practice.

Examiners Comments:

This question examines syllabus section 1 (Foundations)

This was attempted by over 50% of the candidates, with 54% passing it.
The question was open ended, but the examples chosen had to relate to the object oriented paradigm. The evidence shows that some candidates discussed concepts such as coupling and cohesion, but described them generally, without any reference to the object oriented paradigm. To gain a high mark the concept had to be object oriented, with some examples of why it was important for software engineering practice.

A3 The following class diagram represents a major sporting event, such as the Olympic Games:



Describe what the diagram represents. The description should include a section for each class shown in the diagram - the section can be referred to by the name of the class. Each section should consist of a full description of the variables and methods of the class. For each class, describe the relationships in which it takes part.

(25 marks)

Answer Pointers:

To gain full marks the candidate needs to describe the constraints for each class and association.

Key points to note:

- Person is an abstract class;
- Staff and Athlete are subtypes of Person;
- noOfAthletes is a class (or static) attribute;
- getNoOfAthletes is a class (or static) method;
- some of the classes have methods as well as attributes, such as registerDetails() in Athlete. checkQualifications() returns a Boolean;
- Qualification is a directed association, so the Staff class is not aware of Qualification, but Qualification is aware of Staff. The association is 1 to many;
- UML uses look-across cardinalities, so the association between Athlete and Events means that 1 Event can have between 1 and 50 participants, whilst an Athlete can take part in 1 or many Events;
- visaNeeded in Athlete has a default value of false;
- Registration is an association class.

Examiners Comments:

Question 3 examines syllabus section 3 (Design), part 3.2

This was a popular question, answered by over 90% of the candidates, with over 80% passing it, some gaining full marks.

The evidence shows that most candidates could give a basic description of each class, however, to gain full marks, there needed to be full details of all variables and methods. For example, it was not enough to say the Person class had 4 variables, the candidate needed to say what data types they were and their visibility. Each association also needed to be fully described, with the constraints given. Marks were often lost for not doing this or mixing up the constraints for the associations. Some answers confused the Staff to Qualification association as representing inheritance, rather than a directed relationship, thereby describing Qualification as a subtype.

Some candidates included code, which was not needed.

B4

a) Describe how is-a and has-a inter-class relationships may be implemented in object oriented programming, giving code examples to support your answer.

(10 marks)

b) Explain the difference between the object oriented design concepts of generalisation and specialisation, and describe how these relate to the inheritance feature in object oriented programming languages.

(15 marks)

Answer Pointers

a) An *is-a* relationship can be realised using inheritance, whereby one class (the sub-class) inherits members from another class (the super-class). A *has-a* relationship can be realised using composition, whereby one class simply contains objects of another class as members (specifically, this is object composition). Other approaches to realising composition also exist which are equally acceptable.

4 marks

i) is-a

```
class A
{
  // body
};
class B: public A
{
  // body
};
```

3 marks

ii) has-a

```
class A
{
  // body
};
class B
{
  private:
    A myA;
}
```

3 marks

b) The design concept of specialisation involves taking an extant class and extending it, which normally entails adding new members. For instance, one might take a particle class and extend it to create specialised particle classes for protons, neutrons and electrons.

5 marks

The design concept of generalisation involves examining an existing set of classes to look for similarities (e.g., shared members, or members that could reasonably be adapted to be shared). It then entails collecting these shared members to create a super-class that is then specialised to create sub-classes. One of its objectives is to reduce code duplication,

by declaring shared members only once (in the super-class). It also increases the probability that the super-class is reusable.

5 marks

Inheritance is the underpinning object oriented programming feature that allows both specialisation and generalisation to take place. To specialise, we simply extend an existing super-class by creating a sub-class that inherits its public and protected members. Generalisation involves taking existing classes and reorganising their members, creating a new super-class in which shared members may be placed, removing them from the original classes, and then have the original class inherit these features (thereby making them sub-classes) rather than each having their own (redundant) copies.

5 marks

Examiner Comments

Question B4 (a) examines syllabus section 2 (Concepts) part 2.2 and section 4 (Practice) part 4.2. Question B4 (b) examines syllabus section 3 (Design) part 3.2 and syllabus section 4 (Practice) parts 4.2 and 4.3.

This question was attempted by 73% of candidates, with 67% of the answers submitted scoring a pass mark.
The evidence shows that question (a) was generally well answered, although some candidates mixed up ‘is-a’ and ‘has-a’ relationships. Others mistakenly thought that both related to inheritance, but one to specialisation and the other to generalisation. In part (b), there is evidence that the majority of candidates realised that these two terms are closely related and described the direction (in the hierarchy) that we move in. However, given that the question asked specifically about how these concepts relate to inheritance in object oriented programming languages, some answers were a little vague in the sense that they did not convey the fact that the language has specific constructs/keywords available to specialise a class (i.e. perform inheritance), but that generalisation is usually considered in the design phase, and as such, languages typically do not contain a feature to identify common members from which to create a generalised ancestor.

B5

a) Describe TWO features of object oriented programming languages that promote code reuse.

(10 marks)

b) Name THREE different types of polymorphism commonly available in object oriented programming languages, giving code examples to support your answer.

(15 marks)

Answer Pointers

a) Code re-use can be realised in object oriented programming languages in several ways. One is through the creation of a class hierarchy (or set of classes) that can be deployed in different scenarios in which an object of the class type is needed; another is in the creation of libraries. These are related, as the library may contain the class hierarchy or set of classes, but in some languages may also contain standalone methods and other primitives, such as constants and type definitions. Other alternative approaches exist and may be equally acceptable. There are 5 marks available for each:

Class hierarchies promote code reuse by incorporating generalised classes that are more likely to be useful in other programming projects than very specialised classes (i.e., they may be extended from to create a new branch on the class hierarchy). Alternatively, if appropriate, one could simply use the extant classes as required.

5 marks

Libraries collect together logically associated classes (and potentially standalone methods, constants, etc.) that can be incorporated into a new program and used as-is. For instance, one might create a class library for socket programming, and simply include this library and use its classes it provides without modification. Indeed, many such libraries exist for use in programming projects, including those that are provided with the language distribution, and those written by third parties.

5 marks

b) Three forms of polymorphism available in common object programming languages are method overloading, method overriding and operator overloading. Other alternatives exist that may be equally acceptable (such as templates). However, fewer marks are available if the candidate chooses method overloading *and* constructor overloading, since these are conceptually identical. There are 5 marks available for each.

Method overloading enables us to include, as members, several methods that share the same name but have different arguments. When the method is invoked, the programming language will select the particular version that matches the arguments that have been supplied.

```
class A
{
  public:
    void myMethod(int i);
    void myMethod(double d);
};
int main(void)
{
  A myA;
  myA.myMethod(10);   // invoke first version
  myA.myMethod(10.2); // invoke second version
}
```

5 marks

Method overriding enables us to override super-class methods in the sub-class, providing that the argument list is the same. The original implementation is invoked via objects of the super-class and the new method via objects of the sub-class. Can be used in late binding.

```
class A
{
  public:
    myMethod() {cout << "I'm in A!"; }
};

class B: public A
{
  public:
    myMethod() {cout << "I'm in B!"; }
};

int main(void)
{
  A myA;
  B myB;
  myA.myMethod(); // invoke version from class A
  myB.myMethod(); // invoke version from class B
}
```

5 marks

Operator overloading enable us to define different behaviours that are invoked when operators (most often +, -, [ ]) are used in the context of different objects.

```
class A
{
  public:
    int a,b;
    A operator+(const A& obj);
    void operator=(const A& obj);
};

void A::operator=(const A& obj)
{
  (*this).a = obj.a;
  (*this).b = obj.b;
}

A A::operator+(const A& obj2)
{
  A tmpA = *this;
  tmpA.a = tmpA.a + obj2.a;
  tmpA.b = tmpA.b + obj2.b;
  return tmpA;
}

int main(void)
{
  A myA1, myA2, myA3;

  myA1.a = 1;
  myA1.b = 1;
  myA2.a = 2;
  myA2.b = 2;
  myA3.a = 0;
  myA3.b = 0;
  myA3   = myA1 + myA2;

  cout << myA3.a << "  " << myA3.b << "\n";
}
```

5 marks

Examiner Comments

Question B5 (a) examines syllabus section 4 (Practice). Question B5 (b) examines syllabus section 2 (Concepts) part 2.3 and section 4 (Practice).

This question was attempted by 77% of candidates, with 54% of the answers submitted scoring a pass mark.

The evidence shows that part (a) was generally well answered, with many candidates able to identify two possible features of object oriented languages that promote code reuse (which included abstraction in general, the creation of libraries, class hierarchies, polymorphism by templates, and so on). In part (b), which asked specifically about approaches to polymorphism in programming languages, there is evidence that some candidates gave a more theoretical answer and did not supply supporting code, therefore only scoring a fraction of the available marks. Several others outlined two approaches to polymorphism but did not give a third, yielding a maximum of 10 marks for the question.

B6

a) Define the terms encapsulation and data hiding and describe the relationship between them.

(10 marks)

b) Briefly explain the purpose of default, copy and conversion constructors and give a code example of each.

(15 marks)

Answer Pointers

a) Encapsulation and data hiding are closely related concepts, but they are not identical.

Encapsulation relates to the packaging (binding) together of data and operations (e.g., to form classes in the object oriented paradigm). A common objective of encapsulation, aside form being an organisational unit, is to prevent data from being visible outside the encapsulated unit in which it is defined, except through methods that can therefore control access and monitor the validity of that data they arbitrate. In that sense, it also accomplishes data hiding where class members that we wish to be inaccessible outside the class are designated private.

5 marks

Data hiding is a more general term that entails secreting certain details of an implementation (data or functions) away from the user to prevent them being accidentally or intentionally damaged, and also to simplify the programmer's view of the system. This may be via through the use of private members in a reused class (i.e., encapsulation), but can also be implemented using libraries, the invocation of pre-compiled program components, and other means. In practice, the program components that are unlikely to need to be modified are hidden, and an interface to that potentially complexity body of code is provided that enables that programmer to exploit it for their project.

5 marks

b) A default constructor is created by the programming language to initialise an object of a class that does not have an explicitly defined constructor. A copy constructor initiates an object by duplicating an existing object of the same type supplied as an argument (sometimes to override undesired shallow/deep copy behaviours). A conversion constructor initiates an object by converting an existing object supplied as an argument, typically duplicating and converting some attributes of that object. A single class can contain all three of these constructor types simultaneously, as demonstrated below:

```
class A
{
  private:
    int a;
  public:
    A(A myA) {a = myA.geta();} // copy constructor;
    A(B myB) {a = myB.getb();} // conversion constructor
    void seta(int newa) {a = newa;}
    int geta() {return a;}
};
```

```
class B
{
  private:
    int b;
  public:
    void setb(int newb) {b = newb;}
    int getb() {return b;}
};

int main(void)
{
  B myB;
  A myA1;       // will invoke default constructor
  A myA2(myA1);// will invoke copy constructor
  A myA3(myB); // will invoke conversion constructor
}
```

Examiner Comments

Question B6 (a) examines syllabus section 1 (Foundations) part 1.2. Question B6 (b) examines syllabus section 2 (Concepts) part 2.3 and section 4 (Practice) part 4.3.

This question was attempted by 68% of candidates, with 39% of the answers submitted scoring a pass mark. This was the second lowest scoring question in this year's paper. There is evidence that part (a) was generally well answered, although many candidates failed to note that encapsulation describes the bundling together of data and methods used to operate upon that data, instead assuming that it described (only) the restriction of access to data using setters and getters.