

**THE BCS PROFESSIONAL EXAMINATIONS  
Diploma**

**April 2006**

**EXAMINERS' REPORT**

**Object Oriented Programming**

**General**

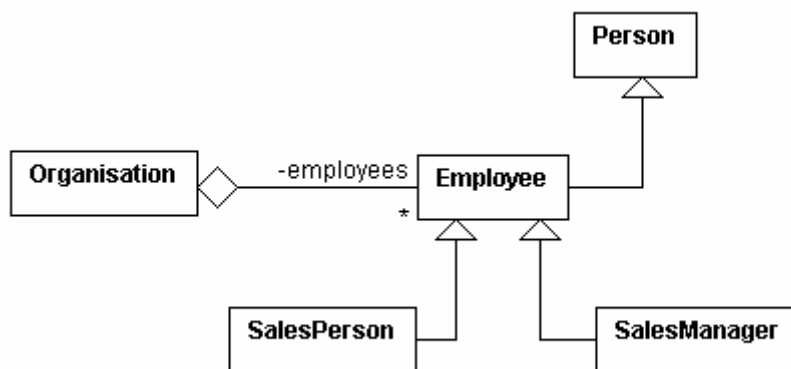
This examination was a major improvement on the previous year. Four of the six questions showed a significant improvement on the average marks obtained.

In the last examiner report we identified that it was our intention that the examination paper would have the same architecture and the same style of questions, and that centres should prepare their candidates accordingly. This uniformity in the setting of the paper would appear to have contributed to the improved performances.

It is our intention that we continue to present this examination paper in this form. However, please note that as most candidates avoided the scenario of question 6, we will attempt to cover that part of the syllabus in a more suitable manner. Centres that prepare students should be advised of this.

**Important Note for Questions 1, 2 and 6.**

If you choose to answer questions 1, 2 or 6 then you should make use of the following Unified Modelling Language (UML) class diagram. It models a company that employs sales staff and sales managers.



**Unified Modelling Language (UML) class diagram.**

**Question 1**

1. a) Using the UML class diagram given at the start of the examination paper, construct a UML object diagram showing one SalesManager and two SalesPersons working for the same Organisation. Give a short explanation of the diagram. **(5 marks)**
- b) If the two SalesPersons and the one SalesManager are employed by the same Organisation, then explain whether there is a need to distinguish between these differing types of employee. **(5 marks)**
- c) Revise the above class diagram to introduce a Secretary class, representing an employee not involved in any sales activities. Explain the principal revisions that have been made to the diagram. **(5 marks)**

- d) Revise the above class diagram so that a **SalesManager** is given managerial responsibilities for a team of sales staff. In your scheme explain how a management hierarchy of **SalesManagers** would be possible. (5 marks)
- e) Revise the object diagram from part 1a) showing both **SalesPersons** being managed by the **SalesManager**. (5 marks)

### Examiners' Comments

Generally, a much stronger performance than in the previous year.

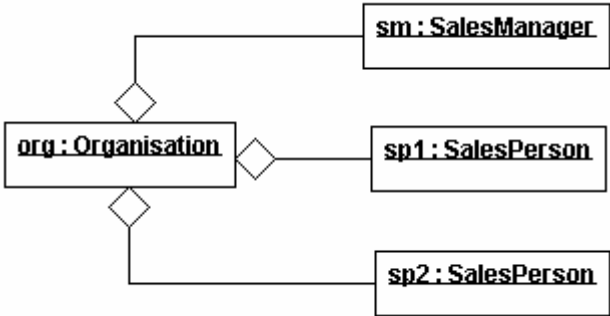
There is strong evidence from the answers given, that the candidates are unable to interpret and infer the knowledge present in the class diagram. For example, when answering part (a) too many candidates included **Person** and **Employee** objects! Further, the object diagram they presented mixed in class diagram elements such as specialisation!

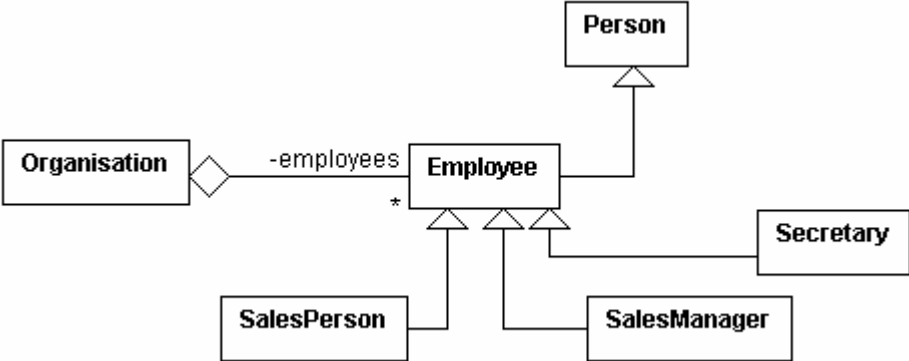
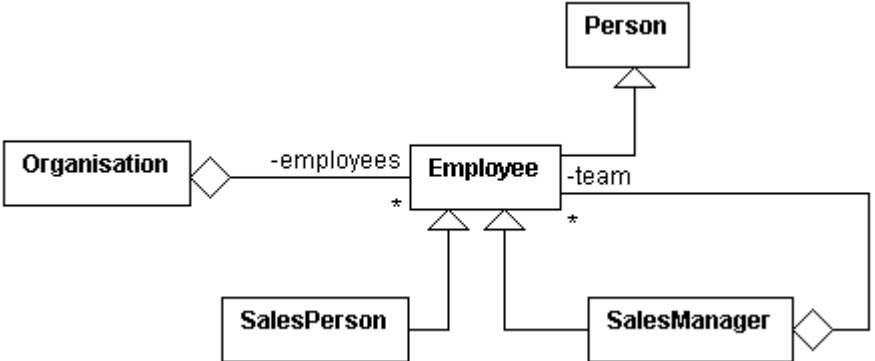
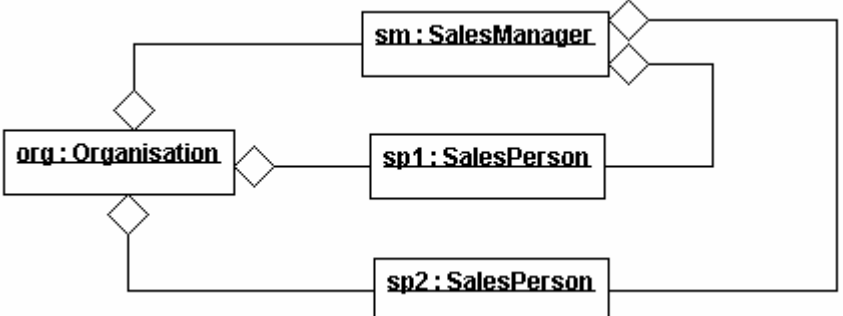
Candidates need practice of working with class, object, collaboration, sequence and use-case diagrams. They also need to demonstrate that they can interpret the information presented by such diagrams.

Some candidates were poor at extending the class diagram as required in parts (c) and (d). For example, the **Secretary** class required in part (c) was incorrectly given as a specialisation of the class **Person** so that a **Secretary** could not be made and employee of the **Organisation**.

Part (b) did not seem to be understood. The question sought to identify that the **Organisation** did not need to distinguish between **SalesPerson** and **SalesManager** since they would all be considered some kind of **Employee** and they would all engage in the same message passing.

### Answer Pointers

Question		Mark
1	This question examines Part 3 (Design) of the syllabus	
(a)	As a consequence of the specialisation hierarchy, both <b>SalesPersons</b> and <b>SalesManagers</b> are managed by the employees (role) set.  	5
(b)	A distinguishing feature of specialisation used in OO systems is that the <b>Organisation</b> would not have to distinguish between either type. The methods describe by the class <b>Employee</b> can be sent to each without concern for what actual type they represent. Method overriding in the class hierarchy might result in differing behaviours, but the <b>Organisation</b> is not concerned with this.	5
(c)	The first revision introduces the class <b>Secretary</b> as a subclass of <b>Employee</b> . Importantly, the <b>Organisation</b> class then forms a relationship with the <b>Employee</b> class so that <b>Secretary</b> , <b>SalesPerson</b> and <b>SalesManager</b> objects can be part of the employees set.	5

	 <pre> classDiagram     class Organisation     class Employee     class Person     class SalesPerson     class SalesManager     class Secretary      Organisation o-- "*" Employee : -employees     Employee &lt; -- SalesPerson     Employee &lt; -- SalesManager     Employee &lt; -- Secretary     Employee &lt; -- Person   </pre>	
(d)	 <pre> classDiagram     class Organisation     class Employee     class Person     class SalesPerson     class SalesManager      Organisation o-- "*" Employee : -employees     Employee &lt; -- SalesPerson     Employee &lt; -- SalesManager     SalesManager o-- "*" Employee : -team     Employee &lt; -- Person   </pre> <p>Here, a group of Employees are made team members of a SalesManager. Since both SalesPerson and SalesManager are subclasses of Employee, then a SalesManager may be a team member of another SalesManager.</p>	5
(e)	 <pre> classDiagram     class org : Organisation     class sm : SalesManager     class sp1 : SalesPerson     class sp2 : SalesPerson      org : Organisation o-- sm : SalesManager     org : Organisation o-- sp1 : SalesPerson     org : Organisation o-- sp2 : SalesPerson     sm : SalesManager o-- sp1 : SalesPerson     sm : SalesManager o-- sp2 : SalesPerson   </pre>	5

## Question 2

2. a) What do you understand by the term *class hierarchy*? (4 marks)
- b) What do you understand by the term *abstract class*? (4 marks)
- c) Draw a revision to the UML class diagram given at the start of the examination paper, clearly distinguishing those classes that have been changed into abstract classes. Explain why they have changed. (6 marks)
- d) Offer a strong argument why object oriented software should be developed in terms of abstract classes. (6 marks)
- e) Using a programming language of your choice and making any reasonable assumptions, give an outline of the implementation of an abstract class from part 2 c). (5 marks)

## Examiners' Comments

Again, a much stronger performance than in the previous year.

Too many candidates answered part (a) by simply describing the relationship between a superclass and a subclass in a class hierarchy. Very few discussed code reuse, method redefinition, polymorphic substitution, etc. That is, few gave concrete illustrations of the consequences of this arrangement.

Usually part (b) was correctly answered with deferred methods and no instantiation. Few candidates identified that an abstract class is used to establish other classes. Certainly, there seemed to be no clear idea of the purpose of abstract classes. Strangely, too many candidates described abstract classes in terms of encapsulation!

## Answer Pointers

Question		Mark
2	This question examines Part 2 (Concepts) of the syllabus	
(a)	A subclass is a specialisation of a superclass and inherits all the features of its superclass. This relationship can be repeated any number of times giving rise to a hierarchy of classes. Class hierarchies support code reuse and support object substitution whereby an object of a subclass can be used where an object of the superclass is required.	4
(b)	An abstract class is used to establish other classes. There is no intention of making object instances of it. It is a way to guarantee that all subclasses share a common set of features.	4
(c)	Since there are no actual examples of <i>Employee</i> or <i>Person</i> then we consider these as abstract classes.	6

(d)	<p>Since an abstract class has no implementation for some of its methods then different subclasses can be defined with differing implementation strategies. Code written to that abstract class is then free to adopt any of the concrete subclasses, even where they differ in their implementation. Therefore our software is more resilient to change.</p>	6
(e)	<pre>public abstract class Employee{     public abstract int getSalary();     // ... }</pre>	5

### Question 3

3. a) A guiding principle for object oriented development processes is that they should be:
- i) Use-case driven
  - ii) Iterative and incremental.

Explain what is meant by these terms.

**(6 marks)**

- b) How does architecture-centric development make the system's architecture the primary focus? **(3 marks)**
- c) How does and iterative and incremental development help minimise risk when developing a system? **(3 marks)**
- d) Discuss how an iterative and incremental development process can be integrated with the testing activity. **(5 marks)**
- e) In the context of object oriented development, explain what is meant by the terms:
- i) Functional testing
  - ii) Unit testing

**(8 marks)**

### Examiners' Comments

It was good to see how many of the candidates were able to answer part (a) with no real difficulty. However many of them strayed into part (d) when answering parts (b) and (c). Perhaps this was a fault in the design of the question that can be remedied in later years. In any event the benefit of the doubt was given in all cases. It is also gratifying to see that the role of testing is (at long last?) understood by students. However, many of them were unclear about the difference between functional and unit testing.

## Answer Pointers

Question		Mark
3	This question examines Part 4 (Practice) of the syllabus	
(a)	<p>(i) A <i>use-case</i> is a typical interaction between a user and the system under development. It is used to capture some functionality to be provided by the software system. For example, in a banking application a use-case may document that a requirement of the system is to support transactions on bank accounts, e.g. make a deposit. Similarly, in a word-processor application changing the font of some text might be presented as a use-case.</p> <p>(ii) An <i>iterative</i> process aims to release a series of versions of the software. Each version augments its predecessor with some additional functionality. We need to ensure that each iteration has a clearly defined aim to avoid undisciplined development. Some iterations may not involve new designs but are concerned with refining or <i>refactoring</i> the model to enhance its quality or its usefulness.</p>	3  3
(b)	Any software system is actually a model of a problem that exists in the real (or imagined) world. Therefore the more closely a software model corresponds to the actual problem then the more effective it will be. OOAD methods recognize this fact and use key abstractions (objects) taken from the problem domain as the fundamental building blocks for the software system.	3
(c)	<p>With this approach there is only one model of the system no matter what stage of development it is at. System development is a process that progressively adds more detail to the model until such time that it can be executed on a computer. Further, there can be different views (perspectives) of the model at any given point in its development. Each view has a specific purpose and uses only part of the information held in the full model. In this way multiple and easily understood views of a complex system can be presented.</p> <p>Within an iteration we conduct an incremental style of development in which we introduce small changes as the software is developed with the aim of minimizing any risk.</p>	3
(d)	<p>More frequent testing than is normal is required. Typically, each increment and iteration gives rise to a testing activity. This puts testing at the forefront of the developer's mind. Given the testing burden placed on the developer, it is likely that the testing process will be automated. Code should be written so that it is relatively easy to test. Ideally it should conform to a standard style and be easy to read and understand. The aim is to avoid introducing errors and to help a developer/tester/maintainer find them.</p> <p>Developing use-cases is a significant activity in the OOAD process. They can be used to communicate with clients, as statements of intent for developers and as specifications for the testing that should be applied to the system when it is under development.</p>	5
(e)	<p>(i) Functional testing is concerned with demonstrating that the system as a whole executes as expected. Typically, in an object-oriented system a functional test would correspond to a use-case. There may be several scenarios for a use-case. Therefore there should be a functional test for each scenario in a use-case.</p> <p>(ii) The term unit testing refers to a test of individual parts of an application. In an object-oriented system a part would typically be a single method, a class or a group of classes. In any event unit testing should take place as soon as possible. There is no need to wait until the complete application has been assembled.</p>	4  4

#### Question 4

4. a) Give definitions of the following:

- i) abstract data type
- ii) encapsulation
- iii) structured programming
- iv) coupling
- v) cohesion

(15 marks)

b) Choose THREE of the above concepts and discuss how each has contributed to the development of object oriented programming.

(10 marks)

#### Examiners' Comments

A much stronger performance than in the previous year.

Part (a) of this question was intended to be relatively straightforward and it proved to be so. It was in fact one of the few questions that seemed to be answered well by most of the candidates. There was a tendency to equate encapsulation with information hiding, and abstract data types was thought to be abstract classes.

Part (b) by contrast was disappointing with many of the students missing the point of the question. Most candidates simply repeated their answer from part (a) and failed to explore the contribution to OO.

#### Answer Pointers

Question		Mark
4	This question examines Part 1 (Foundations) of the syllabus	
(a)	(i) A set of data values and associated operations that are precisely specified and are independent of any implementation.	3
	(ii) Encapsulation involves bringing together several elements to create a new entity with the required behaviour. It is expected that the detailed implementation of the entity will be hidden. It is closely related to abstraction and information hiding.	3
	(iii) A technique for organizing and coding computer programs in which a hierarchy of modules is used. Normally, each has a single entry and exit point. Control is passed downward through the structure without unconditional branches to higher levels of the structure.	3
	(iv) The extent to which an entity is dependent on others. Typically an object communicates with as few objects as possible to achieve its effect.	3
	(v) The extent to which an entity has a clearly defined purpose.	3
(b)	{Each of the following paragraphs should be awarded 3 marks. Students are expected to supply three. Award 1 extra mark to the best.}  The ADT leads to the class (a concrete data type) and objects (instances of the type)	

	represented by the abstractions of the problem domain. Sometimes, as with C++ and Java, there is closer support for the ADT with language primitives <code>int</code> and <code>double</code> .	3
	Encapsulation also leads to the concept of the class with public operations that are implemented privately. Typically, the programming language supports encapsulation with scoping and visibility rules. For example, with Java a class declaration can make use of the key words <code>public</code> and <code>private</code> for the declaration of methods and fields.	3
	Structured programming leads to method bodies whose logic is programmed with structured programming control structures. This avoids entangled code that is the hallmark of code with various “jumps” to labelled statements i.e. the <code>goto</code> . Typically, statements are executed in sequence by default but there is support for selection and repetition of code. Examples are:  Selection: <code>if (condition) statement else statement</code> Repetition: <code>while(condition) statement; do statement while (condition); for( ... ) statement</code>	4
	Good programmers strive to develop classes that are loosely coupled. This minimises the dependencies between them and so aids re-use and maintenance.  Similarly programmers try to develop classes that are highly cohesive. This ensures that they “do one thing and do it well”.	

### Question 5

5. a) Briefly explain what is meant by the term, *Design Pattern*. (3 marks)
- b) Discuss two major advantages of using *Design Patterns*. (6 marks)
- c) Discuss two major disadvantages of using *Design Patterns*. (6 marks)
- d) Describe a *Design Pattern* with which you are familiar. Your answer should include the motivation for the existence of the *Design Pattern*, its structure, participants and consequences of its use. (10 marks)

### Examiners' Comments

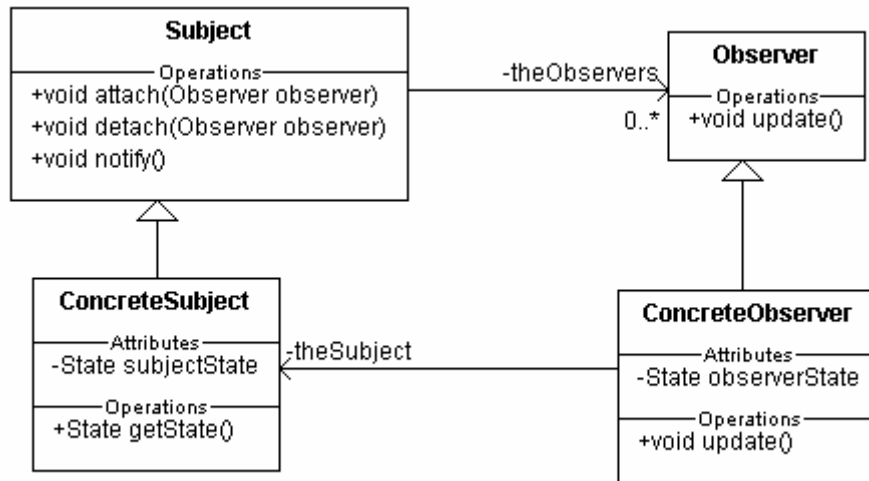
The answers to this popular question were a little disappointing as it was intended to be relatively straightforward. The main problem was that most students did not structure their answer so that it covered the main points asked for. It would help if they are encouraged to prepare a draft of their answer. For example, in part (d) there should be sub-heading for the motivation, structure, participants and consequences.

### Answer Pointers

Question		Mark
5	This question examines Part 3 (Design) of the syllabus	
(a)	A solution to a problem in a context. A design pattern captures in an elegant fashion examples of best-practice as practiced by expert software developers over many years.	3
(b)	{Each of the following paragraphs should be awarded 3 marks. Students are expected to supply two.}  Design patterns illustrate how certain types of problem should be approached. This shortens the learning curve and acts as an excellent source of exemplars. Typically the documentation for a Design Pattern has code samples. Again they should what can/should be done. Discussions of the forces that are taken into account are also very useful from an educational point of view.	3



	<p>Exposure to Design Patterns builds on experience already gained. It may even give confidence that an inexperienced designer was “doing it right” all along. More likely it will widen the developer’s horizons and make it easier to start or take part in a major project. In this respect the discussion of the advantages, disadvantages/implications and related patterns are particularly useful.</p> <p>Design Patterns provide a useful vocabulary for experienced developers. It allows them to communicate at a high level of abstraction without having to worry about unnecessary detail. They have access to a library of properly documented, tried and trusted Design Patterns. It is even possible that they may introduce new Design Patterns and pass on their experience to others.</p> <p>Programmers find it easier to understand what the software does if they understand the Design Patterns on which the software is based. It also helps them make sensible modifications if they appreciate the overall intent and structure of the design.</p> <p>(c) {Each of the following paragraphs should be awarded 3 marks. Students are expected to supply two.}</p> <p>If an inexperienced programmer attempts to use a design pattern it is possible that he will not really understand what he is doing. This may result in a great deal of frustration caused by hard-to-diagnose compiler errors, code that does not execute as expected and a design that is too complex. He may interpret the design pattern as a prescribed algorithm that must be followed slavishly.</p> <p>The use of design patterns may stifle creativity. Some programmers will not really analyse a problem in depth but just decide to use a particular design pattern as it seems to be appropriate. Often, the programmer becomes used to just selecting a documented design pattern rather than thinking about the problem.</p> <p>Design patterns can often add a significant amount of complexity to the design and implementation of a software system. For example, the design pattern may be intended to be deployed in an industrial context. Perhaps the programmer has more modest ambitions. It is also possible that the complexity of the coding has consequences that the programmer does not appreciate. For example, sub-classing a Singleton is “problematic”.</p>	<p>3</p> <p>3</p> <p>3</p>
<p>(d)</p>	<p>A typical Design Pattern selected would be Decorator, Iterator, Observer or Singleton. For example:</p> <p><b>Observer</b> <b>Motivation</b> There is a need to maintain consistency among co-operating objects. However they must not be tightly coupled as they become less re-usable.</p> <p><b>Structure</b></p>	<p>2</p> <p>3</p>



2

3

**Participants**

Subject – knows its observers and provides an interface for attaching/detaching observers.

Observer – defines an updating interface for objects that should be notified of changes.

ConcreteSubject – stores state that is of interest and notifies observers of state changes.

ConcreteObserver – references a ConcreteSubject, stores its own state that should be consistent with the Subject's and implements the Observer updating interface.

**Consequences**

Subjects and observers can be varied independently. Therefore subjects can be re-used with re-using their observers and vice versa. It lets you add observers without modifying the subject or other observers.

## Question 6

6. a) A requirement of object oriented systems is to manage a collection (or container) of objects, e.g. an array or a set.
- i) Identify a collection of objects in the UML class diagram given at the start of the examination paper.
  - ii) Give one example of a collection class (other than the `Vector` class described below) with which you are familiar.
  - iii) Most collection classes are described as *generic*. Explain what is meant by this term. **(12 marks)**

Consider the following scenario:

*A class `Vector` is a collection (container) class that grows dynamically as elements are added to it. Each element has a unique index associated with it. Indices start at 0 and increase by 1. For example, the first element has an index of 0, the second 1 and the third 2.*

*To add an element to a `Vector` we supply an index and the element to be added. For example, `add(2, element)` adds element to the third position in the `Vector`.*

*To return an element from a `Vector` we only need its index. For example, `get(0)` returns the element at the first position in the `Vector`.*

*Similarly, to delete an element we supply the index. For example, `removeAt(1)` deletes the second element from the `Vector`.*

- b) You are required to use the `Vector` class in the construction of a `Stack` class. This new class should mimic a stack. Elements can be added to the top of the stack with a `push` method and retrieved from the stack with a `pop` method. The latter also deletes the element from the stack. Crucially, it should not be possible to add or retrieve elements at intermediate positions in the stack.

Discuss in detail the problems associated with using specialisation (inheritance) to develop the `Stack` class from the `Vector` class. Again, you should use a programming language of your choice to illustrate your answer. **(7 marks)**

- c) Suggest a better alternative to the use of specialisation and explain how it might be implemented. **(6 marks)**

## Examiners' Comments

As in previous years, this question was not attempted by many candidates. It seems that candidates do not like (or are not prepared for) this part of the syllabus.

Given that most OO solutions involve collections of objects, it was expected that the candidates would be comfortable with the basic container classes needed to answer part (a). This proved not to be the case but the assertion that an understanding of collection classes is important still stands.

On reflection, parts (b) and (c) might have been too challenging for the candidates. However, several did manage to give some very good answers. This was gratifying. However, this part of the question should be extensively reviewed next year.

## Answer Pointers

Question		Mark
6	This question examines Part 2 (Concepts) of the syllabus	
(a)	<p>The role denoted as <b>employees</b> denotes a one to many relationship. The many <b>Employee</b> objects would be housed in a collection.</p> <p>The collections classes are used to maintain a reference to a collection of objects. They are frequently required by OO applications such as the employees that work for the software house in Q1. The collection classes include the Java container library and the C++ STL containers. These classes remove the burden of maintaining the object collection.</p> <p>Java supports the <b>ArrayList</b> and the <b>TreeSet</b>. The former maintains the items in the order they are entered into the collection. The <b>TreeSet</b> collects its items in some determined sort order.</p> <p>Containers are described as generic since they are capable of storing objects of any kind. A container, such as an <b>ArrayList</b>, might hold a collection of employees or a collection of bank accounts. Their genericity makes them much more useful than one developed to hold only a particular type of object.</p>	<p>4</p> <p>4</p> <p>4</p>
(b)	<p>With specialisation, a child (subclass) inherits all of the features (methods and attributes) of its parent (superclass). Only one object exists. It is not possible to disinherit a feature. Therefore it may be possible to use a method in a child that that is not appropriate. For example, if we specialize a <b>Vector</b> into a <b>Stack</b> we will be able to access <b>Stack</b> elements at intermediate positions in the <b>Stack</b>.</p> <pre>public class Stack extends Vector {     public void push(Object obj) { add(0, obj); }     public Object pop() { Object element = get(0); removeAt(0); return element; } } // in use Stack stk = new Stack(); stk.push( new Integer(42) ); Object obj = stk.pop(); // but ... stk.add(2, new Integer(44) );</pre> <p>As a consequence we might be tempted to redefine an inherited method to do nothing. This is bad practice.</p>	7
(c)	<p>The alternative is to use delegation (composition) so that the <b>Stack</b> class has a <b>Vector</b> as a private field. It can then make use of the <b>Vector</b> in the methods for push and pop. The important point is that unwanted methods in the <b>Vector</b> class are not available to clients of the <b>Stack</b>.</p> <pre>public class Stack {     public void push(Object obj) { v.add(0, obj); }     public Object pop() { Object element = v.get(0); v.removeAt(0); return element; }     private Vector v = new Vector(); }</pre>	6

	<pre>// in use Stack stk = new Stack(); stk.push( new Integer(42) ); Object obj = stk.pop();  // now ...  // stk.add(2, new Integer(44) ); // will not compile.</pre>	
--	---	--