

**THE BCS PROFESSIONAL EXAMINATION
Diploma**

April 2007

EXAMINERS' REPORT

Object Oriented Programming

General Observations

This examination was another major improvement on the previous year.

In the last-but-one 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.

Question 1

Questions 1, 2 and 6 refer to the following Unified Modelling Language (UML) class diagram (Figure 1). It models a hotel that has a variety of rooms that may be booked by clients.

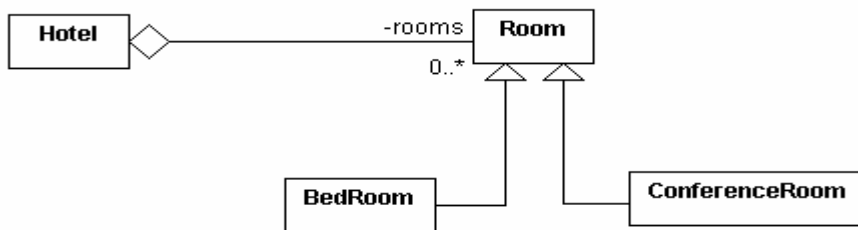


Figure 1

Using Figure 1:

- a) Construct a UML object diagram showing one ConferenceRoom and two BedRooms that are part of the same Hotel. Give a short explanation of the diagram. **(5 marks)**
- b) If the two BedRooms and the one ConferenceRoom are a part of the same Hotel, then explain whether there is a need to distinguish between these differing types of rooms when the Hotel object engages in message passing with them. **(5 marks)**
- c) Revise the class diagram in Figure 1 to introduce a StudyRoom class, representing a room that can be booked as part of a conference. Explain the principal revisions that have been made to the diagram. **(5 marks)**
- d) Revise the class diagram in Figure 1 so that a ConferenceRoom is associated with a number of StudyRooms when booked by a client when organising a conference. **(5 marks)**
- e) Revise the object diagram from part 1(a) showing additionally StudyRooms being associated with a ConferenceRoom. **(5 marks)**

Comments

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

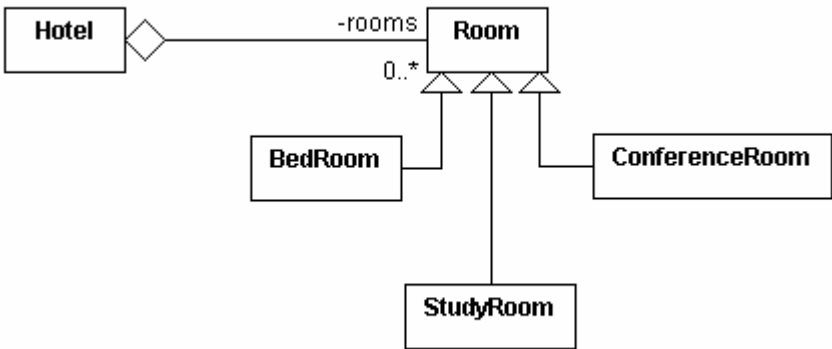
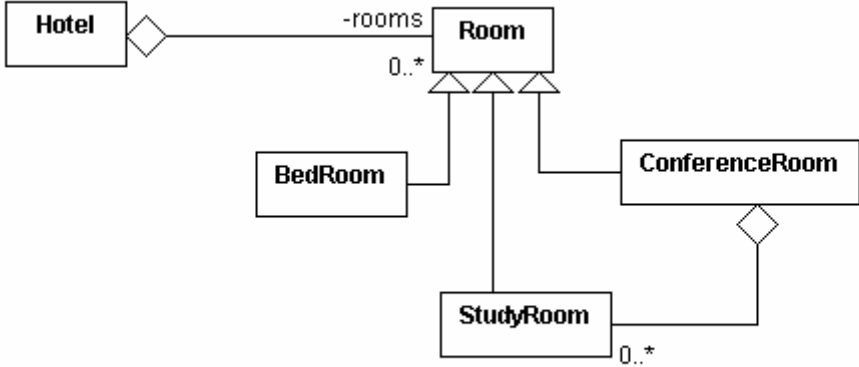
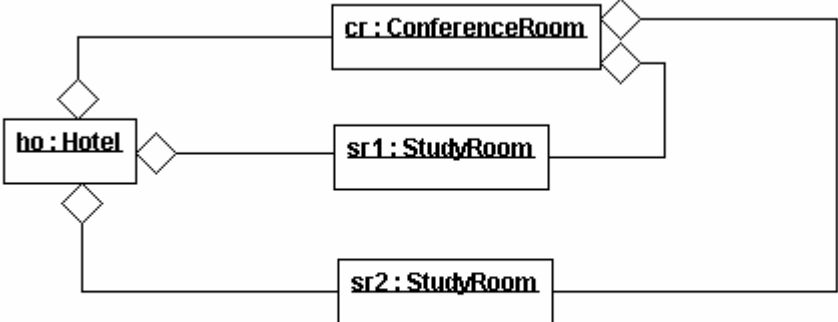
There is still evidence from the answers given, that the candidates are unable to interpret and infer the knowledge present in the class diagram. For example, the object diagrams they presented mixed in class diagram elements such as specialisation!

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 StudyRoom class required in part (c) was incorrectly given as a specialisation of the class ConferenceRoom, presumably because the question said they could be booked along with a ConferenceRoom.

Solution

Question		Mark
1	This question examines Part 3 (Design) of the syllabus	
(a)	As a consequence of the specialisation hierarchy, both BedRooms and ConferenceRooms are managed by the rooms (role) set. <pre> classDiagram class Hotel { +rooms } class ConferenceRoom { } class BedRoom { } Hotel o-- ConferenceRoom Hotel o-- BedRoom Hotel o-- BedRoom </pre>	5
(b)	A distinguishing feature of specialisation used in OO systems is that the Hotel would not have to distinguish between either type. The methods describe by the class Room 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 Hotel is not concerned with this.	5
(c)	The first revision introduces the class StudyRoom as a subclass of Room. Importantly, the Hotel class then forms a relationship with the Room class so that StudyRoom, BedRoom and ConferenceRoom objects can be part of the rooms set.	5

	 <pre> classDiagram class Hotel class Room class BedRoom class StudyRoom class ConferenceRoom Hotel "1" *-- "0..*" Room : -rooms Room < -- BedRoom Room < -- StudyRoom Room < -- ConferenceRoom </pre>	
(d)	 <pre> classDiagram class Hotel class Room class BedRoom class StudyRoom class ConferenceRoom Hotel "1" *-- "0..*" Room : -rooms Room < -- BedRoom Room < -- StudyRoom Room < -- ConferenceRoom ConferenceRoom "1" *-- "0..*" StudyRoom </pre> <p>Here, a group of StudyRooms are made associates of a ConferenceRoom.</p>	5
(e)	 <pre> classDiagram class ho : Hotel class cr : ConferenceRoom class sr1 : StudyRoom class sr2 : StudyRoom ho : Hotel "1" *-- "1" cr : ConferenceRoom ho : Hotel "1" *-- "1" sr1 : StudyRoom ho : Hotel "1" *-- "1" sr2 : StudyRoom cr : ConferenceRoom "1" *-- "1" sr1 : StudyRoom cr : ConferenceRoom "1" *-- "1" sr2 : StudyRoom </pre>	5

Question 2

- a) Define the term class hierarchy. **(4 marks)**
- b) Define the term abstract class. **(4 marks)**
- c) Draw a revision to the UML class diagram in Figure 1 clearly distinguishing those classes that have been changed into abstract classes. Explain why they have changed. **(6 marks)**
- d) Why should object-oriented software 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)**

Comments

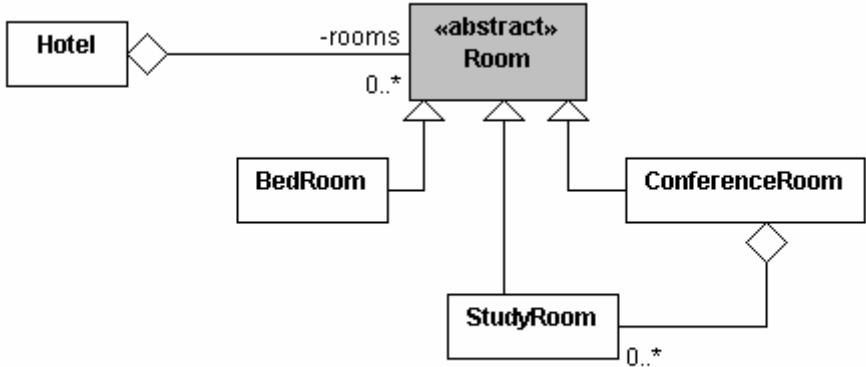
Again, a much stronger performance than in the previous year.

Too many candidates answered part (a) by simply describing the relationship between a superclass and a subclass in a class hierarchy. Most only focussed on 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 (sometimes, but not always) 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.

Solution

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 Employee or Person then we consider these as abstract classes.	6

	 <pre> classDiagram class Hotel class Room { <<abstract>> } class BedRoom class StudyRoom class ConferenceRoom Hotel "1" *-- "0..*" Room : -rooms Room < -- BedRoom Room < -- StudyRoom Room < -- ConferenceRoom ConferenceRoom "1" *-- "0..*" StudyRoom </pre>	
(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

- a) Give an account of the UML use case diagram. Your answer should discuss its overall purpose and explain the meaning of its various symbols. **(10 marks)**

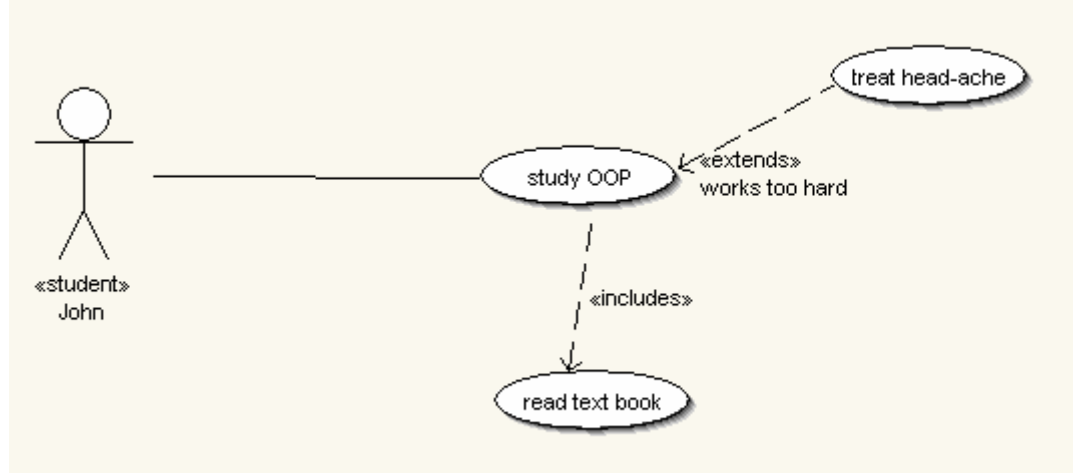
- b) An automated teller machine (ATM) is a computerised device that provides a bank's customers a secure method of performing financial transactions in a public space without the need for a bank clerk. Draw an outline use case diagram for the software that supports a bank's ATM. Having inserted a valid card and password, a user should be able to:
 - i) withdraw money from the ATM
 - ii) get a display the current balance
 - iii) change the password by entering the new details twice**(15 marks)**

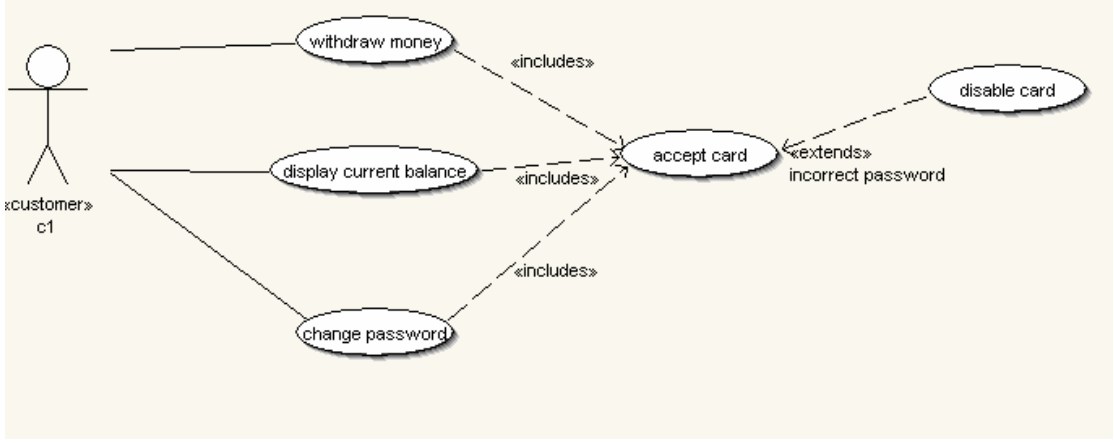
Comments

This was a popular question that was in general answered well by the candidates.

It was pleasing to note that most students found part (a) quite straightforward and clearly appreciated the importance of UML use case diagrams in modern software development. Perhaps part (b) could be improved in future. For example, examiners might preclude the possibility of examples from part (b) being used to answer part (a).

Solution

Question		Mark
3	This question examines Part 3 (Design) of the syllabus	
(a)	<p>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>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> <p>The main symbols used consist of a “stick man” for an actor, an ellipsis for the use case, and a solid line to associate them. A use case may also include another use case. This is shown by the “includes” stereotype. Exceptional behaviour can also be shown by the “extends” stereotype, a generalisation arrow and a condition under which it is taken.</p>  <pre> graph LR John[«student» John] --- study(study OOP) study -.-> «includes» read(read text book) study -.-> «extends» works too hard treat(treat head-ache) </pre>	6

		4
(b)	<p>A typical answer is expected to be based on :</p>  <pre> graph LR actor c1["«customer» c1"] c1 --- withdraw_money["withdraw money"] c1 --- display_balance["display current balance"] c1 --- change_password["change password"] withdraw_money -.-> "«includes»" accept_card["accept card"] display_balance -.-> "«includes»" accept_card change_password -.-> "«includes»" accept_card disable_card["disable card"] -.-> "«extends» incorrect password" accept_card </pre> <p>Many variations are possible. Good students may discuss pre and post conditions as well as paths through a use case. Minor errors should not be penalised. Award approximately 5 marks per use case.</p>	15

Question 4

a) Give definitions of the following:

- i) abstract data type
- ii) encapsulation
- iii) interface class
- iv) polymorphism
- v) the principle of substitution

(15 marks)

b) Choose THREE of the above and discuss how each has contributed to the development of object-oriented programming. **(10 marks)**

Comments

A poorer performance than in the previous year. This was one of two poorly answered questions.

Part (a) of this question was intended to be relatively straightforward and it proved to be otherwise. On this occasion it did not turn out this way. For example, part (i) was often answered by describing abstract methods (word associating, perhaps). Interface class in part (iii) wandered off into the world of user-interface! Many candidates just did not get the idea of substitution (part v).

Part (b) was also 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 contributions to OO.

Solution 4

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 class representing an abstraction/protocol that concrete classes must implement. Interfaces support different implementations for the same abstraction. Application code can then operate to that interface, independent of the actual implementation used.	3
	(iv) The behaviour obtained from a message received by some object will depend on the type of the recipient.	3
	(v) An object instance of a subclass can substitute for the object instance of a superclass.	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

	<p>Programming to an interface reduces the coupling in our code. The interface object can be any instance of a subclass that conforms to this interface. Subclasses might have quite different implementations.</p> <p>Polymorphism increases the adaptability of our code. The same message can be sent to a number of receiving objects and different behaviours can be observed. A new object type able to respond to the same message can be readily accommodated by our code.</p> <p>The substitution principle permits an object of a subclass to be used where an instance of a superclass is expected. Thus, for example, a method may be defined in terms of superclass parameter. When the method is called, any instance of any subclass may be used as the actual parameter.</p>	4
--	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---

Question 5

- a) Explain how an understanding of Design Patterns helps the following people:
- i) students studying computing
 - ii) inexperienced software developers
 - iii) experienced software developers
- (15 marks)**
- b) 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)**

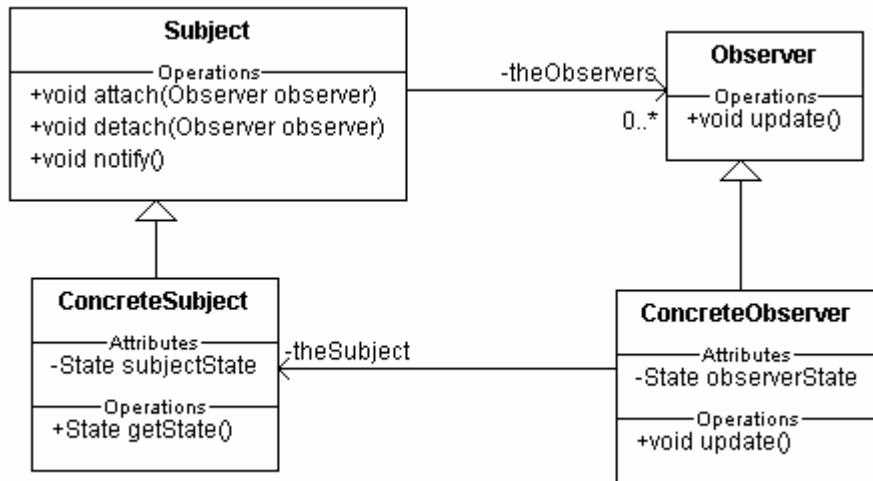
Comments

This was a reasonably popular question with about half of the students attempting it. In general it was answered well.

There were no particular problems with the answers given by students and in general they seemed to be well prepared. It is good to see that students just assume that the use of design patterns are just an accepted part of modern software development.

Solution 5

Question		Mark
5	This question examines Part 3 (Design) of the syllabus	
(a)		
(i)	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.	5
(ii)	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.	5
(iii)	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.	5
(b)	A typical Design Pattern selected would be Decorator, Iterator, Observer or Singleton. For example: Observer Motivation There is a need to maintain consistency among co-operating objects. However they must not be tightly coupled as they become less re-usable. Structure	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.

2

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.

3

Question 6

Often in the development of object-oriented systems there is a requirement to manage a collection (or container) of objects.

- a) Give two reasons why an array may be unsuitable for this purpose. **(4 marks)**
- b) Give one example of a collection class (excluding the array) with which you are familiar. Your answer should explain the main benefits it brings. **(8 marks)**
- c) Identify a collection of objects in the UML hotel class diagram in Figure 1. **(3 marks)**
- d) Using a programming language of your choice, give an outline implementation for the class that maintains the collection identified in 6 c). Your answer should indicate how objects (or object references) are added to the collection and how each element in the collection can be accessed for display purposes. **(10 marks)**

Comments

This was not a popular question but those students that attempted it did much better than last year. This is gratifying but there is room for considerable improvement.

In general, candidates were aware of the benefits of using a collection rather than an array. However, it is clear that very few of them have actually deployed a collection in the implementation of an OO design. Perhaps this is the root of the problem with this particular question. In any event this question does address an important part of OOP. Hopefully, in future, candidates will be better prepared.

Solution 6

Question		Mark
6	This question examines Part 2 (Concepts) of the syllabus	
(a)	<p>Typical responses are:</p> <p>The array is fixed in size which makes it inflexible if the maximum number of elements it must hold is not known.</p> <p>Often there is a need to hold elements in a particular order e.g. sort order. The array is too rudimentary as elements can only be accessed with a simple index. Therefore any processing required must be done by the programmer. This is time consuming and error prone.</p>	<p>2</p> <p>2</p>
(b)	<p>A typical answer using a collection from the Java API is the <code>TreeSet</code>. Some of the benefits it brings are:</p> <ul style="list-style-type: none"> it's size is changed automatically when "one too many" elements is added it holds elements some determined sort order it is generic it has methods to access the elements it has method to add new elements it has methods to help manage the collection it hides its inner working from the programmer. <p>Students are expected to give at least four.</p>	8
(c)	The role name <code>rooms</code> on the zero to many association between the <code>Hotel</code> and the <code>Room</code> class implies a collection of <code>Room</code> objects (or references).	3
(d)	<p>A typical answer (using Java) is as follows:</p> <pre>import java.util.*; public class Hotel{ public Hotel() {</pre>	

```
rooms = new ArrayList();
}

public void addRoom(Room room) {
    rooms.add(room);
}

public displayRooms() {
    iterator iter = rooms.iterator();
    while(iter.hasNext())
        Room rm = (Room) iter.next();
        // display rm perhaps with r.display();
}
private Collection rooms;
}
```

Marks should not be deducted for minor programming errors.

10