BCS Higher Education Qualification

Diploma

October 2024

EXAMINERS' REPORT

Object Oriented Programming

Questions Report:

A1	
	The majority of candidates attempted this question, with most being able to say what the three main member access modifiers were in part a). Marks were lost by not explaining the purpose of each one and including an example. For instance, full marks were awarded to responses that explained *why* public member access modifier allows access from anywhere in the program. Some candidates lost marks by mixing up the scope of private and protected access modifiers. For part b), most candidates could describe two different aspects of object-oriented programming, but did not always relate the concept to reuse. Inheritance was the most popular concept often described. However, to gain full marks an appropriate example was needed and an explanation of how it aids reuse. Encapsulation was also popular, with most candidates being able to say what it is, but failed to say how it helps reuse.
A2	
	Few candidates attempted this question, with some candidates being able to provide near-perfect answers for both parts. For part a), a good answer showed a clear class definition with the constructor, setters and getters defined, along with the tick function. Minor marks were lost by not using the setters and getters functions where appropriate. Marks were lost if there was no class definition, with code provided just to carry out the tick function. For part b), some candidates failed to instantiate their class from part a) and call the constructor or appropriate methods to store the values. For full marks, a loop and appropriate call of the tick function was needed.
A3	
	Most candidates answered this question. In part a) candidates generally knew that polymorphism meant that objects can take on different forms but were unable to give a benefit for doing this. Most marks were lost in part b) by not attempting it or describing techniques that were not related to polymorphism. Method overriding, method overloading, and operator overloading were the most common answers. Most described what these were, but marks were lost by not providing an example or not going into enough detail. For part c) candidates produced bullet points and often, for the concrete class, gave just the opposite point of what they had said in the abstract class, with no further elaboration. For example, just stating which can be instantiated is a basic comparison, a fuller explanation could explain what purpose an abstract class has if it cannot be instantiated.

B4	
	In part a), a number of candidates did not provide a use case diagram, but provided a class diagram or other unrelated diagram type. Of those answers that provided a use case diagram, a system boundary was not included, or there was duplication of the same actor multiple times. However, many candidates did identify appropriate actors and the tasks they perform from the narrative provided. In part b), many candidates discussed why use case diagrams are used, but did not relate their points to the specific example given in the question. It was frequently suggested that use case diagrams allow us to identify the tasks that actors will perform, but few commented that this helps in communications with stakeholders, and that they enable us to identify test cases. A significant number incorrectly asserted that use case diagrams allow us to identify classes (a task that is better accomplished with a class diagram).
B5	
	In part a), most candidates provided code, but this was often incomplete or incorrect, indicating a lack of familiarity with practical object-oriented programming, or a lack of understanding of how a case diagram can be implemented. In many cases, a skeleton of some of the class structure was given, but it lacked some elements that were specifically requested, such as constructors. Many did provide an adequate implementation of the class variable. In part b), relatively few responses demonstrated sufficient understanding of object constraint language (OCL). This was evident through a lack of appropriate code fragment to represent the checkLimit() function. In some cases, regular object-oriented code, rather than OCL, was provided. In others, the keywords chosen were not appropriate for the specific constraints needed for this method. In part c), relatively few candidates could identify how OCL code can be used in the design of test cases, or what the potential benefits of this are over developing tests using regular object-oriented code or diagrammatic representations alone.
B6	
	In part a), most candidates were able to describe the role of the class diagram in the development of object-oriented code and provided an appropriate example. Fewer were able to identify the role of object interaction and state transition diagrams, other than just paraphrasing the names of these diagram types. Gaining greater familiarity with diagram types other than the class diagram and use case diagram in UML is recommended. In part b), many candidates had an idea what coupling and cohesion are. Fewer could relate these concepts to object-oriented code specifically, and a significant proportion mixed up the two concepts, making invalid comments about good practice (e.g. suggesting that we aim for high coupling and low cohesion).