BCS Higher Education Qualification

Diploma

October 2021

EXAMINERS' REPORT

Object-Oriented Programming

General comments
There were a number of candidates who were unable to demonstrate any significant knowledge of the topics covered by the syllabus. It is important that candidates are aware that experience of programming in an object-oriented programming language needs to be supplemented with an understanding of why object-oriented features aid the programmer. This subject matter can be found in the textbooks cited in the syllabus.

Question number: A1
Syllabus area: Design: (a) 3.1; (b) 3.3
Total marks allocated: 25
Examiners' Guidance Notes
For part (a), many candidates knew what an object diagram was, but many fewer were able to describe a component or state diagram. Self-referential answers like "component diagrams are a diagramming technique in UML that show components" were not acceptable, as they do not demonstrate any knowledge, and many appeared to be the result of guesswork. For part (b), some candidates correctly identified that OCL adds formality/precision to UML through pre and post-conditions and invariants, but again some answers, even relatively long ones, appeared to be the result of guesswork and storytelling, with a few buzzwords thrown in. A general point here, and elsewhere, is that examiners are not looking for very long answers. It is what candidates say that counts – it does seem as if candidates were trying to compensate for not really knowing an answer by writing pages or vague text (seemingly guesswork) in the hope of accruing a mark or two, sometimes in handwriting that was very difficult to decipher. This is not a good strategy - write short, direct answers for which every word is used to address the specific question asked.

Question number: A2
Syllabus area: Foundations: (a) 1.4; (b) 1.1 and (c) 1.3
Total marks allocated: 25
Examiners' Guidance Notes

For part (a), the majority of candidates were able to describe what coupling and cohesion meant, in many cases, in quite a detailed and informative way. However, many candidates neglected to read the question carefully enough, and did not relate what they were writing to *object-oriented systems*, so were only able to win a fraction of the available marks.

For part (b), it was necessary to associate structured programming with sequence, selection and iteration, along with sub-routines, and ideally to point out that they tried to move away from unstructured techniques. For the procedural programming paradigm, it was necessary to mention that these languages call procedures. Many candidates were able to do this, but very few read the question carefully enough and were able to identify how structured and procedural concepts *contributed* to the object-oriented paradigms, or even where these features are to be found in object-oriented languages.

For part (c), the majority of candidates could describe the differences between typed and untyped languages, including provided languages that use each approach, and sometimes associating these terms with static and dynamic, and weak and strong typing. Not all candidates provided the example that was requested in the question to demonstrate the differences, so were not awarded all available marks.

Question number: A3

Syllabus area: Design and Practice: (a) 3.1; (b) 3.2 and (c) 4.4

Total marks allocated: 25

Examiners' Guidance Notes

In part (a), candidates were asked to draw a class diagram relating to a 3-class code fragment provided. There were very few completely correct answers. Common mistakes were: to neglect to include the types of attributes; to mangle attribute names when writing them down; to fail to connect the three classes appropriately; to include only attributes but not methods; to mix code with the class diagram (e.g., including curly brackets, or if statements); to fail to spot that studentCount was a static variable; to fail to spot that noOfCredits had an initial value. It should be noted that there is a composition-like relationship between registration, student and module, since instances of student and module are statistically declared inside registration – student and module are therefore destroyed if registration is destroyed. This was frequently overlooked.

In part (b), candidates were asked to draw two object interaction diagrams (one valid, one invalid) using at least two of the three classes from the previous question, and explain why the invalid diagram was incorrect. Many of the diagrams provided were not recognisable as object interaction diagrams. Some diagrams claimed to be invalid were valid, and vice versa. Some failed to explain why the diagram claimed to be invalid was invalid.

In part (c), candidates were asked to state how state transition diagrams can aid testing. There were some quite vague and convoluted responses that skirted around the issue, and did not make clear unambiguous statements. For instance, many didn't use words like *state* and *event*, and did not associate the diagram with the formulation of black box test cases to determine whether correct states were entered when corresponding events occurred.

Question number: B4
Syllabus area: (a) Foundations 1.1; (b) Concepts 2.1, 2.2, Practice 4.1
Total marks allocated: 25
Examiners' Guidance Notes

For part (a), most candidates could name two features of the object-oriented paradigm and described them in depth, often with accompanying code. This gained some credit if they were relevant, but to achieve a high mark, the candidate needed to explain why each feature had disadvantages compared to procedural programming.

For part (b), the candidate had to produce a code fragment that demonstrated an appropriate use of an abstract class from a realistic scenario. Some candidates presented very outline code, such as an abstract Class A, with subtypes Class B and Class C. Whilst this demonstrates an abstract class, it is not a realistic scenario that would be found in real life. To gain a high mark, the scenario needed to be based on an appropriate real-life scenario. Credit was given to candidates who included some discussion of the scenario to explain that it was suitable for an abstract class.

To gain credit, the code needed to include an abstract class, some candidates presented a superclass with subtypes, but there was no evidence of an abstract class. Any object-oriented programming language was acceptable; the most popular languages were Java or C++/C#.

Some candidates just discussed the process of abstraction, but failed to relate this to the use of abstract classes.

Question number: B5
Syllabus area: (a) Concepts 2.3, Practice 4.3; (b) Practice 4.1
Total marks allocated: 25
Examiners' Guidance Notes

For part (a), most candidates could describe what overloading meant and included some examples. To gain a high mark the examples needed to be based on a realistic scenario, whilst often the examples were very basic and did not reflect real life scenarios.

Part (b) was often not attempted. Those who did attempt it, sometimes confused the open-closed principle (OCP) with encapsulation or access modifiers, assuming that public and private modifiers equated with the open-closed principle.

Some candidates just presented code with no explanation or comments in the code to say why it demonstrated OCP. To gain a high mark some explanation should be included to say how the code supported the principle of components being open for extension, but closed for modification.

Question number: B6
Syllabus area: (a) Practice 4.2; (b) Concepts 2.2, Practice 4.2, 4.3
Total marks allocated: 25
Examiners' Guidance Notes

A majority of candidates attempted this question.

For part (a), most candidates could describe two features of the object-oriented programming paradigm, however, they needed to explain how the features helped produce reusable code. Just describing two features was not sufficient to gain a good mark. For example, encapsulation and abstraction were two common features picked, but some only described what the concepts meant and did not relate them to how they aided reuse. A number of answers gave polymorphism as a feature, but the explanation and code examples implied the reuse was in the reuse of the method names, rather than any reuse of code.

Candidates who gained high marks in part (b), included good examples of an inheritance hierarchy, with a clear superclass and two or more subclasses. Each class had clear attributes and methods appropriate to a realistic scenario. Weaker answers only included a very sketchy outline, often with class A/B/C type examples, rather than considering a real-life scenario, or presented a set of classes with no inheritance at all. The better answers included some comments in the code to explain the scenario.