BCS Higher Education Qualification

Diploma

April 2022

EXAMINERS' REPORT

Object Oriented Programming

Question number: A1

Examiners' Guidance Notes

In Part (a), most candidates were able to describe what a design pattern is, although there was some confusion concerning whether design patterns are a diagrammatic, programmatic, or both.

In Part (b) was less well answered, with only some students able to both identify and correctly describe five characteristics you would use to document a design pattern. Some answers we seemingly intuitive guesswork. Some candidates named five characteristics in a list but did not *describe* them as the question asked.

In Part (c), most candidates who were able to provide code opted to attempt to implement the singleton pattern. In some answers, code was inappropriate in that there were no apparent restrictions in place to limit object multiplicity. In some other answers, candidates merely described the singleton but this did not accrue any marks as the question specifically asked for an implementation and made no mention of a description.

Question number: A2

Many candidates did not read the question carefully enough, leading to the following common issues:

- Failure to summarise FIVE object-oriented features (some students described fewer, and some described a greater number of concepts but in scant detail);
- Neglecting to include either a diagram or code fragment to illustrate each feature;
- Not selecting features carefully. For instance, choosing topics that significantly overlap, such as specialisation, inheritance and generalisation, not addressing the part of the question that requested the selection of the *most important* features.
- Failing to answer the part of the question that asked them to emphasise why the feature is *important.*
- In some cases, concepts were selected that were not specific to the object-oriented paradigm, such as talking in general terms about coupling and cohesion.

Question number: A3

In Part (a), most candidates produced a valid use case diagram (i.e., featuring the core diagrammatic components). However, the level of completeness of diagrams varied widely, with some candidates carefully scrutinising the description to extract all actors and actions, and some presented a rather oversimplified view. Several candidates didn't include a boundary box, which although not a compulsory feature of use case diagrams, does help to illustrate actors inside and outside the system.

In Part (b), many candidates did not read the question carefully enough and failed to relate their answer to the swimming club scenario used in part (a). Although some answers mentioned requirements capture, testing, and exploiting the relative simplicity of this diagram type to help facilitate discussion with stakeholders, these core purposes with either missing or insufficiently clearly explained by many. Some other answers described *what* a use case diagram is, but didn't really answer the question, which sought an explanation of how this diagram type contributes to the development of a system.

Question number: B4

For part (a), the candidate had to describe a realistic scenario in which overriding may be appropriate or necessary in the development of an object-oriented program. A number of candidates could describe what overriding is, but then could not apply it to a realistic scenario, resorting to a Class A, with methods x and y type examples.

The Shape example was the most common scenario given, with the draw() method overridden in the subclasses. In a few cases some candidates mixed up overloading with overriding so gained no credit.

Part (b) required the candidate to present a code fragment which demonstrated how the Liskov substitution principle (LSP) of object oriented programming may be operationalised in practice. Quite a few candidates did not attempt this part of the question, which would account for the low pass rate. Some students presented code that seemed to reflect part (a) and presented examples that would demonstrate overriding rather than LSP.

To achieve a high mark the candidate needed to show an example of how a superclass object could be replaced with a subclass object without breaking the functionality of the system. A number of students produced code to demonstrate how you could replace a superclass object with a subclass, but did not consider scenarios where this may not always be appropriate and doing so might cause issues, such as producing side effects that would not have happened with a superclass object. Higher marks went to candidates who included some explanation of the code, or include some comments in the code.

Question number: B5

For part (a) the candidate had to describe two features of the object-oriented programming paradigm that lend themselves to the division of labour between multiple programmers working concurrently. A lot of candidates discussed classes and methods as the two features, which could be appropriate, but then went on to discuss inheritance, overloading and overriding, without attempting to discuss these features in the context of helping programmers work concurrently.

Part (b) required examples of code that demonstrated an appropriate use of a composition relationship between classes. This part was often not attempted, which would explain the low pass mark. Those who did attempt it often showed examples of a triangle containing points, with a good explanation of how it represented composition. Weaker answers produced code to demonstrate inheritance, rather than composition.

Question number: B6
Part (a) required a description of two class access specifiers and had to explain the circumstances in which it would be appropriate to use each of them. Most candidates could describe two different class access specifiers, public and private being the most popular. Higher marks went to candidates that could suitably explain when to use them in a system, which was often overlooked in weaker answers. For Part (b), candidates had to present code that demonstrated an appropriate use of both class and instance variables. Again, a lot of candidates did not attempt this part, or produced weak code, in particular by not including any class variables. Some answers appeared to be written to demonstrate access specifiers from Part (a), which could gain credit if they also included examples of both class and instance variables.