2.7 Object-Oriented Programming: Test Pack

AQA 7517 · Calculators not allowed · about 35 minutes

Total Marks

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 Object-oriented programming (OOP) uses a specific set of terms to describe how programs are structured. Define each of the following terms, giving a short example in each case.

(a) class **(2)**

(b) object **(2)**

(c) attribute **(2)**

(d) method **(2)**

(Total for Question 1 is 8 marks)

© Revision Library | revisionlibrary.co.uk

2 A programmer writes the following AQA-style pseudocode class definition for a simple Student record.

```
class Student
private studentName
private examScore

public new(name, score)
    studentName <- name
    examScore <- score
endnew

public getScore()
    return examScore
endgetScore

endclass
```

(a) Explain the purpose of the new() subroutine in the class above. **(2)**

(b) Write one line of pseudocode that creates a Student object called student1, representing a student named "Priya" with an exam score of 68. **(2)**

(c) State the value that would be returned by the call student1.getScore(), and identify the visibility keyword that would need to change if examScore was instead to be accessed directly as student1.examScore. **(2)**

(Total for Question 2 is 6 marks)

© Revision Library | revisionlibrary.co.uk

3 A software developer is modelling a Car class. She considers two possible designs.

Design 1: Car has an attribute engine, which stores an Engine object (so Car contains an Engine).

Design 2: A class SportsCar inherits from Car.

(a) State the type of relationship shown in Design 1, and explain what it means for one class to be related to another in this way. **(3)**

(b) State the type of relationship shown in Design 2, and explain what it means for one class to be related to another in this way. **(3)**

(c) A SportsCar is a type of Car, but a Car is not a type of Engine. Explain why this means inheritance is an appropriate relationship between SportsCar and Car, but not between Car and Engine. **(2)**

(Total for Question 3 is 8 marks)

© Revision Library | revisionlibrary.co.uk

4 Explain how encapsulation is achieved in an object-oriented program, and discuss the benefits it brings to the development of a large software system.

(Total for Question 4 is 9 marks)

© Revision Library | revisionlibrary.co.uk

Mark scheme · 2.7 Object-Oriented Programming: Test Pack

Question 1

(a) valid definition: a class is a template/blueprint that defines the attributes and methods that its objects will have (oe) **B1**

(a) valid, clearly named example, e.g. a class called Car defining attributes such as colour and methods such as accelerate() (oe) **B1**

(a) Answer: A class is a template/blueprint that defines the attributes (data) and methods (behaviour) that objects created from it will have. Example: a class called Car defines that every Car object will have attributes such as colour and speed, and methods such as accelerate().

(b) valid definition: an object is a specific instance of a class, created (instantiated) with its own values for the class's attributes (oe) **B1**

(b) valid, clearly named example consistent with a class, e.g. myCar, a Car object with colour set to "red" (oe) **B1**

(b) Answer: An object is a specific instance of a class, created (instantiated) in memory, with its own set of values for the attributes defined by its class. Example: myCar, an object of the Car class, with colour = "red".

(c) valid definition: a piece of data stored by a class/object describing one of its characteristics (oe) **B1**

(c) valid example, e.g. colour or speed for a Car object (oe) **B1**

(c) Answer: An attribute (or property) is a piece of data stored by a class or object that describes one of its characteristics. Example: colour or speed for a Car object.

(d) valid definition: a subroutine defined within a class describing an action/behaviour its objects can perform (oe) **B1**

(d) valid example, e.g. accelerate() or deposit() (oe) **B1**

(d) Answer: A method is a subroutine (procedure or function) defined within a class that describes an action or behaviour that objects of that class can perform. Example: accelerate() for a Car object.

Question 2

(a) new() is the constructor; it runs automatically whenever a Student object is instantiated (oe) **B1**

(a) it initialises the object's attributes (studentName, examScore) using the values passed in as parameters **B1**

(a) Answer: new() is the constructor for the Student class. It is called automatically when a Student object is created, and it initialises the object's attributes (studentName and examScore) using the values passed in as parameters.

(b) uses new Student(...) instantiation syntax assigned to student1 (oe) **M1**

(b) correct arguments in the correct order: "Priya" then 68 cao **A1**

(b) Answer: student1 <- new Student("Priya", 68)

(c) 68 cao **B1**

(c) private would need to change to public **B1**

(c) Answer: getScore() returns 68. The keyword private would need to change to public for examScore to be accessed directly from outside the class.

Question 3

(a) Design 1 shows containership, also called aggregation or composition - a "has-a" relationship **B1**

(a) an object of the Car class stores a reference to (contains) an object of another class, Engine, as one of its attributes **B1**

(a) Car can then use the Engine object's own methods/attributes (e.g. calling engine.start()) rather than reimplementing engine behaviour itself **B1**

(a) Answer: Design 1 shows containership (also called aggregation or composition) - a "has-a" relationship. A Car object stores a reference to an Engine object as one of its attributes. This lets Car make use of the Engine object's own methods (e.g. engine.start()) rather than Car having to implement that behaviour itself.

(b) Design 2 shows inheritance - an "is-a" relationship **B1**

(b) SportsCar (the subclass) automatically has all the attributes and methods of Car (the superclass), without them needing to be rewritten **B1**

(b) SportsCar can also add its own extra attributes/methods and/or override existing ones inherited from Car **B1**

(b) Answer: Design 2 shows inheritance - an "is-a" relationship. SportsCar automatically has all of Car's attributes and methods without rewriting them, and can add its own extra attributes/methods (e.g. a turboBoost() method) or override existing ones from Car.

(c) inheritance should be used for an "is-a" relationship; SportsCar is a Car, so it is appropriate for SportsCar to inherit Car's attributes/methods and be treated as a Car **B1**

(c) a Car is not a kind of Engine, so making Car inherit from Engine would be logically incorrect (it would wrongly give Car all of Engine's own methods); containership better models Car simply having/using an Engine **B1**

(c) Answer: Inheritance models an "is-a" relationship, and a SportsCar genuinely is a type of Car, so it is appropriate for SportsCar to inherit from Car. A Car is not a type of Engine, however, so making Car inherit from Engine would be incorrect - it would give Car all of Engine's own methods and attributes, which does not reflect reality. Containership (Car simply storing an Engine object) models this has-a relationship correctly instead.

© Revision Library | revisionlibrary.co.uk

Question 4

Level 3 A detailed and accurate explanation of how encapsulation is achieved (private attributes accessed only through public methods, e.g. getters/setters, which can include validation) is combined with a well-reasoned discussion of at least two distinct benefits, supported by relevant examples. Technical terminology is used accurately and consistently, and the answer is coherently structured.	**7-9**
Level 2 A reasonable explanation of encapsulation is given, with at least one benefit discussed. Technical terms are mostly used correctly, but the explanation and/or discussion may lack full development, examples, or precision.	**4-6**
Level 1 A basic or partial explanation of encapsulation is given, with little or no discussion of benefits. Use of technical terms is limited, and points made are largely unsupported or generic.	**1-3**
Level 0 No creditable content.	**0**

Indicative content:

- Attributes/data are declared private within a class, so they cannot be accessed directly from outside the class.
- Access is instead provided through public methods (e.g. getters and setters, or task-specific methods such as deposit()/withdraw()).
- These public methods can include validation logic to check that a new value is sensible before the internal state is actually changed.
- This hides (abstracts away) the internal implementation/data representation of the class from the rest of the program.
- Benefit: prevents other parts of the program from directly setting invalid or inconsistent data in an object.
- Benefit: the internal implementation of a class can later be changed without breaking other code that uses it, provided the public interface (method signatures) stays the same.
- Benefit: makes large systems easier to maintain and debug, since all changes to an attribute must go through a small, known set of methods rather than being scattered across the whole program.
- Benefit: reduces coupling between parts of a large system, allowing different programmers or teams to work on separate classes independently.

© Revision Library | revisionlibrary.co.uk