2.4 Programming Paradigms and OOP: Test Pack

OCR H446 · Calculators not allowed · about 25 minutes

Total Marks

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 A programming paradigm is a general style or approach used to structure a solution to a problem. Different paradigms lead to programs being organised in very different ways, even when solving the same problem.

(a) Define what is meant by a 'programming paradigm'. **(1)**

(b) State one feature of procedural programming. **(1)**

(c) State one feature of object-oriented programming. **(1)**

(d) State one feature of functional programming. **(1)**

(Total for Question 1 is 4 marks)

© Revision Library | revisionlibrary.co.uk

2 A school in Preston is building a simple pseudocode class to represent a student's exam mark. The starting point is shown below, using OCR's Exam Reference Language class conventions.

```
CLASS ExamResult
  PRIVATE mark : INTEGER
```

PUBLIC PROCEDURE NEW(startingMark)

```
    mark <- startingMark
  ENDPROCEDURE
```

PUBLIC FUNCTION GetMark() RETURNS INTEGER

```
    RETURN mark
  ENDFUNCTION
ENDCLASS
```

(a) Identify which attribute of ExamResult is encapsulated, and state the keyword in the code that enforces this. **(2)**

(b) Explain why mark is declared PRIVATE rather than PUBLIC in this class. **(2)**

(c) Complete the pseudocode for a public procedure SetMark(newMark) that only updates mark if newMark is between 0 and 100 inclusive; otherwise it should output an error message and leave mark unchanged. **(2)**

(Total for Question 2 is 6 marks)

© Revision Library | revisionlibrary.co.uk

3 A car-leasing company in Leicester wants to model different types of vehicle and calculate their estimated annual running cost.

(a) Write pseudocode for a class Vehicle with one protected attribute annualMileage (REAL), so that subclasses can use it, a constructor PROCEDURE NEW(miles) that sets annualMileage, and a public function CalculateRunningCost() RETURNS REAL that simply returns 0 (this will be overridden by every subclass). **(4)**

(b) Write pseudocode for a subclass ElectricCar that inherits from Vehicle, and overrides CalculateRunningCost() so that it returns annualMileage multiplied by a fixed cost of 0.04 (pounds per mile for electricity). **(3)**

(c) Write a procedure PrintTotalRunningCost(vehicles) that takes a list of Vehicle (and/or subclass) objects and outputs the total combined running cost of every vehicle in the list, using polymorphism so that it works correctly whatever mixture of vehicle subclasses the list contains. **(3)**

(Total for Question 3 is 10 marks)

© Revision Library | revisionlibrary.co.uk

Mark scheme · 2.4 Programming Paradigms and OOP: Test Pack

Question 1

(a) a fundamental style/approach/model of programming that determines how a solution is structured and how it is written (oe) **B1**

(a) Answer: A programming paradigm is a fundamental style or approach to structuring and writing a program, such as procedural, object-oriented or functional.

(b) code is organised as a sequence of instructions grouped into procedures/functions that are called in turn (oe); accept: uses global/shared data that procedures act on **B1**

(b) Answer: Procedural programs are built from a sequence of procedures/functions that are called to carry out steps in order.

(c) data and the methods that act on it are bundled together into objects (encapsulation); accept inheritance or polymorphism as an alternative feature (oe) **B1**

(c) Answer: Data and the methods that operate on it are bundled together into objects created from classes.

(d) programs are built by applying/composing (pure) functions, avoiding changing state or mutable/global data (oe); accept reference to higher-order functions **B1**

(d) Answer: Functional programs are built by applying and combining functions, generally avoiding changing state or mutable data.

Question 2

(a) mark cao **B1**

(a) PRIVATE cao **B1**

(a) Answer: The mark attribute; encapsulation is enforced by declaring it PRIVATE.

(b) it prevents other parts of the program from directly accessing/changing mark from outside the class (oe) **B1**

(b) access must instead go through the class's own methods (e.g. GetMark), which can validate/control how the attribute is read or changed, protecting the integrity of the data (oe) **B1**

(b) Answer: Declaring mark PRIVATE stops other code from reading or changing it directly; instead it can only be accessed through the class's own methods, which can validate any change and protect the data's integrity.

(c) uses an IF statement to test newMark ≥ 0 AND newMark ≤ 100 (oe) **M1**

(c) correctly assigns mark <- newMark only inside the TRUE branch, with an ELSE branch that outputs an error message and does not change mark **A1**

(c) Answer:PUBLIC PROCEDURE SetMark(newMark)

```
  IF newMark >= 0 AND newMark <= 100 THEN
    mark <- newMark
  ELSE
    OUTPUT "Error: mark must be between 0 and 100"
  ENDIF
ENDPROCEDURE
```

Question 3

(a) declares annualMileage as PROTECTED : REAL **M1**

(a) constructor PROCEDURE NEW(miles) correctly assigns annualMileage <- miles **M1**

(a) declares PUBLIC FUNCTION CalculateRunningCost() RETURNS REAL **M1**

(a) function body correctly returns 0; overall class syntax logically complete (ignore minor syntax errors) **A1**

(a) Answer:

```
CLASS Vehicle
  PROTECTED annualMileage : REAL

  PUBLIC PROCEDURE NEW(miles)
    annualMileage <- miles
  ENDPROCEDURE

  PUBLIC FUNCTION CalculateRunningCost() RETURNS REAL
    RETURN 0
  ENDFUNCTION
ENDCLASS
```

(b) class header CLASS ElectricCar INHERITS Vehicle **M1**

(b) overrides PUBLIC FUNCTION CalculateRunningCost() RETURNS REAL **M1**

(b) correctly returns annualMileage * 0.04 cao (ignore minor syntax errors) **A1**

(b) Answer:

```
CLASS ElectricCar INHERITS Vehicle
  PUBLIC FUNCTION CalculateRunningCost() RETURNS REAL
    RETURN annualMileage * 0.04
  ENDFUNCTION
ENDCLASS
```

(c) declares a running total variable initialised to 0 before the loop **M1**

(c) uses a FOR EACH loop over vehicles, calling v.CalculateRunningCost() on each element and adding it to the running total **M1**

(c) outputs the final total after the loop; relies on each vehicle's own overridden CalculateRunningCost() being called automatically (polymorphism), without checking each object's exact subclass (ignore minor syntax errors) **A1**

(c) Answer:

```
PROCEDURE PrintTotalRunningCost(vehicles)
  total <- 0
  FOR EACH v IN vehicles
    total <- total + v.CalculateRunningCost()
  NEXT v
  OUTPUT "Total running cost: GBP ", total
ENDPROCEDURE
```


© Revision Library | revisionlibrary.co.uk