2.4 Programming Paradigms and OOP: 30-minute Pack

Total Marks

OCR H446 · Calculators not allowed · about 30 minutes

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 A payroll system for a chain of shops in Glasgow uses a class Employee, shown below, and needs a subclass Teacher for staff at the company's in-house training academy.

```
CLASS Employee
  PRIVATE name : STRING
  PRIVATE salary : REAL
```

PUBLIC PROCEDURE NEW(n, s)

```
    name <- n
    salary <- s
  ENDPROCEDURE
```

PUBLIC FUNCTION GetDetails() RETURNS STRING

```
    RETURN name + ", salary GBP " + STR(salary)
  ENDFUNCTION
ENDCLASS
```

(a) Complete the class header line below so that Teacher inherits from Employee and adds one new private attribute subject (STRING). **(2)**

```
CLASS Teacher _______
  PRIVATE subject : STRING
  ...
ENDCLASS
```

(b) Write pseudocode for Teacher's constructor PROCEDURE NEW(n, s, subj), which should call the Employee constructor to set name and salary, then set subject from its own parameter. **(3)**

(c) Explain what is meant by 'method overriding', using a version of GetDetails written for Teacher as your example. **(2)**

(Total for Question 1 is 7 marks)

© Revision Library | revisionlibrary.co.uk

2 A graphics package for a design studio in Manchester defines a superclass Shape and two subclasses, Circle and Rectangle, each overriding a function CalculateArea.

```
CLASS Shape
  PUBLIC FUNCTION CalculateArea() RETURNS REAL
    RETURN 0
  ENDFUNCTION
ENDCLASS

CLASS Circle INHERITS Shape
  PRIVATE radius : REAL
  PUBLIC PROCEDURE NEW(r)
    radius <- r
  ENDPROCEDURE
  PUBLIC FUNCTION CalculateArea() RETURNS REAL
    RETURN pi * radius ^ 2
  ENDFUNCTION
ENDCLASS

CLASS Rectangle INHERITS Shape
  PRIVATE width : REAL
  PRIVATE height : REAL
  PUBLIC PROCEDURE NEW(w, h)
    width <- w
    height <- h
  ENDPROCEDURE
  PUBLIC FUNCTION CalculateArea() RETURNS REAL
    RETURN width * height
  ENDFUNCTION
ENDCLASS
```

A list shapes contains a mixture of Circle and Rectangle objects. The following loop is used to print each shape's area:

```
FOR EACH s IN shapes
  OUTPUT s.CalculateArea()
NEXT s
```

(a) Explain what is meant by polymorphism in object-oriented programming, referring to the Shape, Circle and Rectangle classes above. **(3)**

© Revision Library | revisionlibrary.co.uk

(b) Using the loop given in the question, explain why the code that processes the shapes list does not need to know whether each item is a Circle or a Rectangle. **(3)**

(Total for Question 2 is 6 marks)

---

3 A weather-logging tool for a network of stations across the UK uses functions to process a list of daily temperature readings (in degrees Celsius).

```
globalTotal <- 0

FUNCTION AddToGlobalTotal(x) RETURNS REAL
  globalTotal <- globalTotal + x
  RETURN globalTotal
ENDFUNCTION

FUNCTION CelsiusToFahrenheit(c) RETURNS REAL
  RETURN (c * 9 / 5) + 32
ENDFUNCTION
```

(a) Define what is meant by a 'pure function' in functional programming. **(2)**

(b) State one benefit of avoiding mutable/global state when writing functions. **(2)**

(c) Identify which of the two functions given in the question, AddToGlobalTotal or CelsiusToFahrenheit, is impure, and give a reason for your answer. **(2)**

(Total for Question 3 is 6 marks)

© Revision Library | revisionlibrary.co.uk

4 A vehicle-hire company in Oxford models its fleet using the classes below.

```
CLASS Vehicle
  PROTECTED make : STRING

  PUBLIC PROCEDURE NEW(m)
    make <- m
  ENDPROCEDURE

  PUBLIC FUNCTION Describe() RETURNS STRING
    RETURN make + ": generic vehicle"
  ENDFUNCTION
ENDCLASS

CLASS Car INHERITS Vehicle
  PUBLIC FUNCTION Describe() RETURNS STRING
    RETURN make + ": car, 4 wheels"
  ENDFUNCTION
ENDCLASS

CLASS Motorbike INHERITS Vehicle
  PUBLIC FUNCTION Describe() RETURNS STRING
    RETURN make + ": motorbike, 2 wheels"
  ENDFUNCTION
ENDCLASS
```

The following driver program is then run:

```
fleet <- []
fleet.APPEND(NEW Vehicle("Generic Co"))
fleet.APPEND(NEW Car("Skoda"))
fleet.APPEND(NEW Motorbike("Triumph"))

FOR EACH v IN fleet
  OUTPUT v.Describe()
NEXT v
```

(a) State the three lines of output produced by the driver program, in the order they are produced. **(4)**

© Revision Library | revisionlibrary.co.uk

(b) Explain why the second and third lines of output use the overridden versions of Describe() in Car and Motorbike, rather than the version defined in Vehicle, even though the loop only ever calls v.Describe(). **(4)**

(Total for Question 4 is 8 marks)

© Revision Library | revisionlibrary.co.uk

Mark scheme · 2.4 Programming Paradigms and OOP: 30-minute Pack

Question 1

(a) uses an inheritance keyword linking Teacher to Employee (e.g. INHERITS) **M1**

(a) CLASS Teacher INHERITS Employee cao **A1**

(a) Answer: CLASS Teacher INHERITS Employee

(b) constructor header PUBLIC PROCEDURE NEW(n, s, subj) with three parameters **M1**

(b) calls the superclass constructor, e.g. SUPER.NEW(n, s), to set name and salary **M1**

(b) assigns subject <- subj after the SUPER call; overall logic complete (ignore minor syntax errors) **A1**

(b) Answer:PUBLIC PROCEDURE NEW(n, s, subj)
SUPER.NEW(n, s)

```
  subject <- subj
ENDPROCEDURE
```

(c) method overriding is when a subclass defines its own version of a method that already exists in its superclass, replacing that inherited behaviour when called on an object of the subclass (oe) **B1**

(c) e.g. Teacher could define its own GetDetails() that also includes subject in the returned string, so calling GetDetails() on a Teacher object uses Teacher's version rather than Employee's (oe) **B1**

(c) Answer: Method overriding is when a subclass provides its own implementation of a method inherited from its superclass, so that version runs instead. For example, Teacher could override GetDetails() to also include subject, so calling GetDetails() on a Teacher object returns extra information not present in Employee's version.

Question 2

(a) polymorphism means objects of different classes can be treated through a common interface/superclass, while each responds to the same method call in its own way (oe) **B1**

(a) here, Circle and Rectangle both inherit from Shape and both override CalculateArea, so both types of object can be treated as Shape objects (oe) **B1**

(a) calling CalculateArea() on a Circle object runs Circle's version, and calling it on a Rectangle object runs Rectangle's version, even though the same method name/call is used (oe) **B1**

(a) Answer: Polymorphism means that objects of different classes can be treated through a shared superclass/interface, but each still responds to a call in its own way. Here, Circle and Rectangle both override CalculateArea from Shape, so calling CalculateArea() on either type of object uses that object's own version of the method, even though the same method name is used throughout.

(b) every object in shapes is treated simply as a Shape, since Circle and Rectangle both inherit from Shape (oe) **B1**

(b) the loop calls s.CalculateArea() the same way for every element, without checking or needing to know its exact class (oe) **B1**

(b) because of polymorphism/dynamic (run-time) method binding, the correct overridden version of CalculateArea for that object's actual class is automatically called (oe) **B1**

(b) Answer: Because Circle and Rectangle both inherit from Shape, every element of shapes can be treated as a Shape and called in exactly the same way with s.CalculateArea(). The program does not need to check each object's exact class, because polymorphism (dynamic method binding) ensures the correct overridden version of CalculateArea runs automatically for whichever type of object it actually is.

© Revision Library | revisionlibrary.co.uk

Question 3

(a) a function whose return value depends only on its input parameters/arguments (oe) **B1**

(a) and which has no side effects, i.e. it does not change any state/data outside itself (e.g. global variables) (oe) **B1**

(a) Answer: A pure function is one whose result depends only on its input arguments, and which causes no side effects, meaning it does not change any variable or state outside the function.

(b) the function's behaviour is easier to predict/test/debug, because calling it does not depend on, or change, any state elsewhere in the program (oe) **B1**

(b) it makes the function safer to run in parallel/concurrently, since there is no shared state that could be altered by another process at the same time (oe) **B1**

(b) Answer: It makes functions easier to test and reason about, since their output only depends on their own inputs; it also makes them safer to run in parallel, since there is no shared state for two calls to interfere with.

(c) AddToGlobalTotal cao **B1**

(c) it reads and permanently changes the global variable globalTotal, which is a side effect outside the function, so the same call can return a different result each time depending on globalTotal's current value (oe) **B1**

(c) Answer: AddToGlobalTotal is impure, because it changes the external variable globalTotal each time it is called, so the value it returns depends on more than just its own parameter x, and it has a side effect outside the function.

Question 4

(a) "Generic Co: generic vehicle" **B1**

(a) "Skoda: car, 4 wheels" **B1**

(a) "Triumph: motorbike, 2 wheels" **B1**

(a) all three lines given in the correct order matching the order objects were appended to fleet **B1**

(a) Answer:Generic Co: generic vehicle
Skoda: car, 4 wheels
Triumph: motorbike, 2 wheels

(b) each element of fleet is declared/stored as a Vehicle type but actually refers to an object of its real (sub)class, Car or Motorbike (oe) **B1**

(b) when v.Describe() is called, the program does not simply use Vehicle's method because the object's actual class defines its own version (oe) **B1**

(b) this is dynamic (run-time) method binding/dispatch, where the version of the method that is run is decided by the object's actual class at run time, not by the declared type used to store the reference (oe) **B1**

(b) this is exactly what allows polymorphism: the same call v.Describe() behaves differently depending on which class of object v actually refers to at that point in the loop (oe) **B1**

(b) Answer: Each element of fleet is stored using the general Vehicle type, but it actually refers to an object of its true class, Car or Motorbike. When v.Describe() is called, dynamic (run-time) method binding means the version that runs is chosen according to the object's real class, not its declared type, so Car's and Motorbike's own overridden Describe() methods are used. This is exactly what allows the same call v.Describe() to behave differently for each object, which is polymorphism in action.

© Revision Library | revisionlibrary.co.uk