2.7 Object-Oriented Programming: 30-minute Pack

AQA 7517 · Calculators not allowed · about 30 minutes

Total Marks

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 A class diagram for a BankAccount class is shown below.

BankAccount

- accountNumber : String
- balance : Real

+ new(accNum : String, openingBalance : Real)
+ deposit(amount : Real) : Void
+ withdraw(amount : Real) : Boolean
+ getBalance() : Real

(a) State what the "-" symbol before accountNumber and balance indicates, and explain why a programmer would choose this visibility for these attributes. **(3)**

© Revision Library | revisionlibrary.co.uk

(b) Using AQA pseudocode conventions, write the class definition for BankAccount, including the constructor new(), the deposit() method, and the getBalance() method. The deposit() method should add amount to balance, provided amount is greater than 0. **(6)**

(c) Using AQA pseudocode conventions, write the withdraw(amount) method. It should return TRUE and reduce balance by amount if there are sufficient funds (balance ≥ amount); otherwise it should return FALSE and leave balance unchanged. **(4)**

(d) A second programmer suggests making balance a public attribute so that other parts of the program can update it directly, instead of using deposit() and withdraw(). Explain two problems this could cause. **(3)**

(Total for Question 1 is 16 marks)

---

2 A start-up company is building a simulation of a hospital, modelling wards, patients, doctors and equipment, each with related data and behaviour, and expects to keep extending the simulation with new features for several years.

Discuss whether an object-oriented programming language or a procedural programming language would be more suitable for this project.

(Total for Question 2 is 9 marks)

© Revision Library | revisionlibrary.co.uk

Mark scheme · 2.7 Object-Oriented Programming: 30-minute Pack

Question 1

(a) "-" indicates the attribute is private **B1**

(a) private attributes cannot be accessed or changed directly from outside the class **B1**

(a) this protects the data (encapsulation): balance can only be changed via methods such as deposit()/withdraw(), which can validate the change, e.g. preventing an invalid negative balance being set directly (oe) **B1**

(a) Answer: The "-" symbol means the attribute is private. This means accountNumber and balance cannot be accessed or altered directly from outside the class. This protects the data (encapsulation): balance, for example, can only be changed through methods such as deposit() and withdraw(), which can check that a change is valid before applying it, rather than any other part of the program being able to set an invalid value directly.

(b) class header class BankAccount with private accountNumber and private balance declared **M1**

(b) constructor new(accNum, openingBalance) with both parameters correctly assigned to the attributes **M1**

(b) deposit(amount) method header defined **M1**

(b) IF amount > 0 THEN condition used to guard the update **M1**

(b) balance <- balance + amount placed inside the IF **M1**

(b) getBalance() method that returns balance cao **M1**

(b) Answer:class BankAccount
private accountNumber
private balance

public new(accNum, openingBalance)

```
    accountNumber <- accNum
    balance <- openingBalance
endnew
```

public deposit(amount)

```
    IF amount > 0 THEN
       balance <- balance + amount
    ENDIF
enddeposit
```

public getBalance()

```
    return balance
endgetBalance
```

endclass

(c) withdraw(amount) method header defined **M1**

(c) IF balance ≥ amount THEN condition used **M1**

(c) balance <- balance - amount and return TRUE placed inside the IF branch **M1**

(c) ELSE branch returns FALSE, leaving balance unchanged cao **M1**

(c) Answer:public withdraw(amount)

```
IF balance >= amount THEN
   balance <- balance - amount
   return TRUE
ELSE
   return FALSE
ENDIF
```

endwithdraw

(d) problem 1: no validation would occur, so balance could be set directly to an invalid value, e.g. a negative number, with nothing checking it **B1**

(d) problem 2: it becomes harder to track down where/how balance changes in a large program, since any part of the code could alter it, not just deposit()/withdraw() **B1**

(d) further valid development, e.g. it breaks encapsulation, tightly coupling other code to BankAccount's internal representation, making future changes to the class harder **B1**

(d) Answer: Problem 1: without deposit()/withdraw() enforcing rules, balance could be set directly to an invalid value (e.g. a negative number) with no validation. Problem 2: it becomes much harder to find and debug where balance is changed, since any part of the program could alter it directly rather than only through the class's own methods; this breaks encapsulation and makes the class harder to maintain in future.

© Revision Library | revisionlibrary.co.uk

Question 2

Level 3 A clear, well-supported discussion applies features of OOP (classes/objects modelling real-world entities; encapsulation protecting data; inheritance/polymorphism supporting shared and specialised behaviour) directly to the hospital scenario, compares this with procedural characteristics, and reaches a justified conclusion while acknowledging at least one valid counterpoint.	**7-9**
Level 2 Some accurate comparison of OOP and procedural approaches is given with partial application to the hospital scenario, but development is limited, the discussion is one-sided (no real counterpoint), or there is some inaccuracy.	**4-6**
Level 1 Generic, undeveloped statements about OOP and/or procedural programming are made with little or no reference to the given scenario.	**1-3**
Level 0 No creditable content.	**0**

Indicative content:

- The hospital simulation involves distinct real-world entities (Patient, Doctor, Ward, Equipment), each with its own data and behaviour, which map naturally onto classes and objects in OOP.
- Encapsulation would let each class protect its own data, e.g. a Patient's medical data could only be changed through the Patient class's own methods, reducing the risk of other parts of the simulation corrupting it.
- Inheritance could model variations, e.g. different types of Ward (GeneralWard, IntensiveCareWard) inheriting shared attributes/methods from a Ward superclass while overriding specific behaviour.
- Polymorphism would let code that processes a collection of, for example, different types of Staff call the same method name and automatically get the correct behaviour for each specific subclass.
- Since the company plans to keep extending the simulation for years, OOP's support for adding new subclasses without editing existing, tested code is well suited to a long-lived, growing project.
- Procedural programming (writing the program as a series of procedures/functions operating on shared data) can be simpler and quicker for small, short-lived programs, but keeping track of which procedures affect which data can become harder to manage as more features and data types are added.
- Counterpoint: OOP can introduce more upfront design effort, and in some cases performance overhead, compared with an equivalent procedural program; this may not be justified for a very small or short-lived project, but the scenario described (a large, long-running simulation) does not fit that case.

© Revision Library | revisionlibrary.co.uk