2.2 PROGRAMMING TECHNIQUES · 2.2.1(F)

Object-oriented programming — Mark scheme

50 marks · spec 2.2.1(f)

AO key: AO1 = knowledge · AO2 = application · AO3 = programming / judgement. Code answers are indicative — award for correct logic in any consistent OOP syntax.

Q	ANSWER	AO	MARKS

1	`class Book`
	`  private title`
	`  private author`
	`  private copiesAvailable`
	`  public procedure new(pTitle, pAuthor)`
	`  title = pTitle`
	`  author = pAuthor`
	`  copiesAvailable = 1`
	`  endprocedure`
	`endclass`
	
	Class declared with correct name (1); three attributes declared **private** (1); constructor (`new`) declared public with two parameters (1); `title` set from parameter (1); `author` set from parameter (1); `copiesAvailable = 1` (1). (6)
		AO3	6

Q	ANSWER	AO	MARKS

2(a)	`public procedure borrowBook()`
	`  copiesAvailable = copiesAvailable - 1`
	`endprocedure`
	
	Correct method header/structure (1); decrease `copiesAvailable` by one (1). (2)
		AO3	2

2(b)	`public function getCopies()`
	`  return copiesAvailable`
	`endfunction`
	
	Returns `copiesAvailable` (1); correct function structure (1). Explanation (any 2 → 2): this is encapsulation; attributes can't be changed directly from outside the class, protecting data from invalid/accidental change; set methods can validate input; the internal implementation can change without breaking other code. (4)
		AO2	4

Q	ANSWER	AO	MARKS
3*	1 mark identify + 1 mark expansion, × 2 benefits: • Encapsulation — data + methods bundled in a class; attributes can be private so data is protected, improving reliability. • Reusability / inheritance — classes can be reused and inherited from, reducing duplicated code and development time. • Maintainability / modularity — self-contained objects mean a change in one class is less likely to affect others. • Polymorphism — the same method name can behave appropriately for different classes, adding flexibility. (4)	AO1	4

Q	ANSWER	AO	MARKS
4(a)	Inheritance. (1)	AO1	1
4(b)	`Book` = superclass / parent / base class (1); `EBook` = subclass / child / derived class (1). (2)	AO1	2

Q	ANSWER	AO	MARKS

5	```
	public procedure new(pName)
	    name = pName
	    fuel = 10
	    shields = 0
	    routePosition = 0
	endprocedure
	``
	Header `public procedure new(pName)` with parameter (1); `name = pName` (1); `fuel = 10` (1); `shields = 0` AND `routePosition = 0` (1). (4)
		AO3	4

Q	ANSWER	AO	MARKS
6	``` public function getFuel() return fuel endfunction ``` Correct function structure (`public function … endfunction`) (1); `return fuel` (1). (2)	AO3	2

Q	ANSWER	AO	MARKS

7	```
	public procedure collectItem(itemKind, itemAmount)
	    if itemKind == "fuel" then
	        fuel = fuel + itemAmount
	    elseif itemKind == "shield" then
	        shields = shields + itemAmount
	    endif
	endprocedure
	``
	Header with two parameters (1); `if itemKind == "fuel"` (1); `fuel = fuel + itemAmount` (1); `elseif itemKind == "shield"` (1); `shields = shields + itemAmount` (no else needed — "neither" is handled by doing nothing) (1). (5)
		AO3	5

Q	ANSWER	AO	MARKS

8	`newItem = new Item("Cell", "fuel", 25)`
	`allItems[3] = newItem`
	
	Create a new `Item` object (1); pass the three correct parameters in order `("Cell", "fuel", 25)` (1); store it in `allItems[3]` (1). (One line also fine: `allItems[3] = new Item("Cell","fuel",25)`.) (3)
		AO3	3

Q	ANSWER	AO	MARKS
9	(1) `"Nova"` (1); (2) `40` (1); (3) `getAmount()` (1); (4) `getFuel()` (1). (4)	AO2	4

Q	ANSWER	AO	MARKS
10	Any four (line + correction): • Line 02: `for x = 0 to 39` (40 sectors are indexed 0–39, not 0–40). (1) • Line 04: `if route[x] == null then` (comparison uses `==`, not `=`). (1) • Line 07: `print(route[x].getName())` (output the name, not the amount). (1) • Line 09: `next x` (the loop counter is `x`, not `y`). (1)	AO2	4

Q	ANSWER	AO	MARKS
11(a)	Bundling an object's data & methods together (1); keeping the attributes private so they are only accessed via the object's get/set methods (1). (2)	AO1	2
11(b)	Different classes respond to the same method call in their own way (1); usually by overriding a parent method (1); valid game example, e.g. a `describe()` or `use()` method behaving differently for different `Item`/ship types (1). (3)	AO2	3
11(c)	Each sector holds an `Item` object that bundles its own data (name/kind/amount) & methods (1); the route array gives ordered, index-based access to the sectors (1); the same `Item` class is reused for every item (no duplicated code) (1); encapsulation means an item's data can only change through its methods, reducing bugs (1). (4)	AO2	4

Total for worksheet: 50 marks