1.2 SOFTWARE & SOFTWARE DEVELOPMENT · 1.2.4 · FULL A-LEVEL

Object-oriented programming concepts — Mark scheme

38 marks · spec 1.2.4(e)

AO key: AO1 = knowledge & understanding · AO2 = application · AO3 = design/program & reasoned judgements. Accept any valid alternative; do not award the same point twice.

Q	ANSWER	AO	MARKS
1(a)	A blueprint/template for a type of object (1) defining its attributes and methods (1). (2)	AO1	2
1(b)	Object = an instance of a class (1); method = a subroutine defined in a class (1); attribute = a variable storing an object's data (1). (3)	AO1	3

Q	ANSWER	AO	MARKS
2(a)	price or ticketType. (1)	AO2	1
2(b)	new(...) or getPrice(). (1)	AO2	1
2(c)	`adult = new Ticket(12.50, "Adult")` — correct use of new with the class name (1); correct arguments assigned to adult (1). (2)	AO3	2
2(d)	private. (1)	AO1	1

Q	ANSWER	AO	MARKS
3(a)	Keeping attributes private (1) so they can only be accessed/changed through the class's public (get/set) methods (1). (2)	AO1	2
3(b)	The class controls its own data / attributes cannot be set to invalid values from outside (1); the implementation can change without breaking other code (1). (2)	AO1	2
3(c)	Make price private and provide a public setPrice method (1) that only assigns the new value if it is ≥ 0 (1). (2)	AO2	2

Q	ANSWER (INDICATIVE)	AO	MARKS

4	```
	class Dog
	    private name
	    private age
	
	    public procedure new(givenName, givenAge)
	        name = givenName
	        age = givenAge
	    endprocedure
	
	    public function getName()
	        return name
	    endfunction
	endclass
	``
	1 mark each, max 5: class declaration with endclass (1); both attributes declared private (1); constructor (new) taking both parameters (1); constructor assigns parameters to attributes (1); public get method returning name (1).
		AO3	5

Q	ANSWER	AO	MARKS
5(a)	1 mark per point, max 3: • a (sub)class is based on / derived from an existing (super)class (1) • it automatically receives the superclass's attributes and methods (1) • it can add its own, or override inherited ones (1)	AO1	3
5(b)	`class Puppy inherits Dog` — class declaration (1); correct inherits syntax with Dog (1). (2)	AO3	2

<table>
<tr><th>Q</th><th>LEVELS-OF-RESPONSE MARK SCHEME</th><th>AO</th><th>MARKS</th></tr>
<tr><td>6</td><td>Mark using the levels descriptors below. AO1 (knowledge of OOP concepts), AO2 (application to the game's many similar characters), AO3 (justified recommendation).

<table><tr><th>LEVEL</th><th>DESCRIPTOR</th></tr><tr><td>Level 3 (9–12)</td><td>Thorough discussion using several OOP concepts (inheritance, encapsulation, polymorphism, reuse) accurately, clearly applied to the many-similar-characters context, with a balanced view of procedural and a clear justified recommendation.</td></tr><tr><td>Level 2 (5–8)</td><td>Discusses some OOP advantages with some application to the game; recommendation made with partial justification.</td></tr><tr><td>Level 1 (1–4)</td><td>Basic points about OOP and/or procedural with little application or justification.</td></tr><tr><td>0</td><td>Nothing creditworthy.</td></tr></table>
Indicative content (credit any valid point):
• Inheritance: a base Character class holds shared behaviour; Player/Enemy/NPC subclasses add or override only their differences — less duplicated code.
• Encapsulation: each character protects its own state (e.g. health cannot go negative).
• Polymorphism: the game loop can call update()/draw() on every character without knowing its type.
• Reuse and team development: classes can be written and tested independently.
• Balance: procedural is simpler for small linear tasks, but with many similar entities OOP is more maintainable — justified conclusion: use OOP.</td><td>AO1 ×4
AO2 ×4
AO3 ×4</td><td>12</td></tr>
</table>

Total for paper: 38 marks