

THEME B: COMPUTATIONAL THINKING · B3.2 · HL ONLY

Fundamentals of OOP for multiple classes — Mark scheme

34 marks · covers B3.2.1–B3.2.5

How to mark. Each creditworthy point is on its own line and ends with a semicolon (;). A line beginning with ... is an *expansion* of the line above it, so the pair is worth the marks shown. A slash (/) separates acceptable alternative wordings. Words in (brackets) are not needed for the mark. Mark positively: credit what the student has got right rather than penalising what is missing, and accept any valid alternative not listed here. Do not award more than the maximum for a part. Where a later part uses a wrong answer from an earlier part correctly, award follow-through marks (FT). HL marks parts that assess higher level only content.

Q	ANSWER	MARKS
HL 1(a) B3.2.1	Award [3] max Award [1] per point up to [3] max. Must be applied to this hierarchy. The attributes and methods common to everyone, such as name and dateOfBirth, are written once in Person; Member and StaffMember inherit them rather than declaring their own copies; ... (so) the same code is not repeated in two places, and a change to it is made once and takes effect in both; Each subclass then adds only what is specific to it, such as membershipNumber or staffId; A further class, such as Instructor, could be added later and reuse the same base;	3
HL 1(b) B3.2.1	Award [1] max JAVA <pre>public class StaffMember extends Person</pre> PYTHON <pre>class StaffMember(Person):</pre>	1
HL 1(c) B3.2.1	Award [2] max Award [1] for the effect and [1] for the consequence. A private attribute is not accessible from a subclass, so StaffMember cannot refer to name directly; ... (so) it must use a public or protected accessor method inherited from Person, whereas protected would have allowed direct access within the subclass;	2
HL 2(a) B3.2.2	Award [2] max Award [1] per point up to [2] max. Overriding is where a subclass provides its own implementation of a method that it inherited from its parent; The method has the same name, parameters and return type as the one it replaces; ... (so) calling that method on an object of the subclass runs the subclass version instead of the parent's;	2
	Award [3] max	3

Q	ANSWER	MARKS
HL 2(b) B3.2.2	Award [1] for a correctly named method with the same signature as the parent's; [1] for reusing the parent version rather than repeating its code; [1] for adding the membership number to the output. JAVA <pre>@Override public String describe() { return super.describe() + ", membership " + membershipNumber; }</pre> PYTHON <pre>def describe(self): return super().describe() + ", membership " + str(self.membership_number)</pre> The @Override annotation is good practice but is not required for the mark. Accept an answer that rebuilds the full description without calling the parent, but award [2] max, since the point of overriding here is to extend rather than duplicate.	
HL 2(c) B3.2.2	Award [2] max Award [1] for the correct version and [1] for the reason. The Member version runs; The method to call is decided at run time from the actual type of the object, not from the declared type of the variable; this is dynamic polymorphism;	2
HL 3(a) B3.2.2	Award [3] max Award [1] per point up to [3] max. The contrast must be clear. Overriding replaces an inherited method in a subclass, whereas overloading provides several methods with the same name in the same class; An overriding method has an identical signature to the one it replaces, whereas overloaded methods must differ in their parameters; Overriding requires inheritance, whereas overloading does not; Overriding is resolved at run time from the object's actual type (dynamic), whereas overloading is resolved at compile time from the arguments supplied (static);	3
HL 3(b) B3.2.2	Award [2] max Award [1] for two methods sharing the name and [1] for correctly differing parameter lists. JAVA <pre>public void addMember(String name) { // assign the next available number } public void addMember(String name, int membershipNumber) { // use the number supplied }</pre> PYTHON <pre>def add_member(self, name, membership_number=None): if membership_number is None: pass # assign the next available number</pre>	2

Q	ANSWER	MARKS
	<pre>else: pass # use the number supplied</pre> Python does not support true overloading, since a later definition replaces an earlier one. A default parameter, as shown, is the accepted equivalent and gains full marks. Do not award the second mark for two Java methods whose parameter lists are identical, which would not compile.	
HL 4(a) B3.2.3	Award [4] max Award [1] per point up to [4] max. An abstract class declares methods that every subclass must provide, without supplying the implementation itself; Any subclass is required to implement those abstract methods before it can be used; ...(so) every subclass is guaranteed to offer the same set of operations, which is the common interface; Code elsewhere can therefore treat any subclass through the abstract type and call describe() without knowing which subclass it holds; The abstract class can also supply shared implementation for the behaviour that is the same everywhere, so it is not duplicated; This hides the differences between the subclasses behind a single agreed set of operations;	4
HL 4(b) B3.2.3	Award [2] max Award [1] for the point and [1] for the expansion. It is incomplete: at least one of its methods is declared without an implementation; ...(so) there would be no code to run if that method were called on such an object; It exists to define what subclasses have in common rather than to represent a real thing; ...(so) a Person that is neither a Member nor a StaffMember has no meaning in this system;	2
HL 5(a) B3.2.4	Award [3] max Award [1] per point up to [3] max. Both must be addressed. Both describe a whole made up of smaller component objects (a has-a relationship); In composition the components cannot exist independently of the whole: they are created and destroyed with it; In aggregation the components can exist on their own, and may belong to more than one whole; ...(so) composition is the tighter coupling of the two; In a UML diagram composition is shown with a filled diamond and aggregation with a hollow one;	3
HL 5(b) B3.2.4	Award [2] max Award [1] for the correct relationship and [1] for the justification. No mark for the relationship alone. Composition; The lockers are created when the gym is built and destroyed if it closes, so they have no existence independent of the gym; by contrast members can leave and join another gym, which makes that relationship aggregation;	2
HL 6(a) B3.2.5	Award [3] max Award [1] per point up to [3] max. A problem it solves is required for full marks. The singleton pattern ensures a class has only one instance, and provides a single global point of access to it; The constructor is made private and a static method returns the one instance, creating it on first use; It solves the problem of a resource that must not be duplicated, such as a single connection to the gym's database or one shared configuration object;	3

Q	ANSWER	MARKS
	...(so) two parts of the program cannot each create their own copy and end up with conflicting state;	
HL 6(b) B3.2.5	Award [2] max Award [1] per point up to [2] max. The observer pattern lets one object (the subject) notify a list of dependent objects automatically whenever its state changes; The observers register with the subject, and are updated without the subject needing to know what they are; ...(so) new observers can be added without changing the subject, for example a display and a logger both reacting when a locker is assigned;	2

Total for this paper: 34 marks