B3 OBJECT-ORIENTED PROGRAMMING · B3.2 · HL ONLY

OOP for multiple classes

How classes relate: **inheritance**, **polymorphism**, **abstraction**, **composition vs aggregation**, and **design patterns**. HL only.

01 Inheritance & polymorphism

Inheritance	Subclass gets superclass members.
"is-a"	A Dog is an Animal.
Benefit	Code reuse.
Polymorphism	Same call, different result.
Overriding	Subclass redefines a method.

02 Abstraction & relationships

Abstraction	Define what, not how.
Abstract class	Method with no body to implement.
Composition	Strong "owns-a" (filled diamond).
Aggregation	Weak "has-a" (hollow diamond).
Patterns	Reusable design templates.

03 The four relationships at a glance

1

Inheritance

"is-a": subclass reuses and extends a superclass.

2

Polymorphism

One call, many forms, mainly via overriding.

3

Abstraction

Hide detail behind a shared interface.

4

Composition / aggregation

"has-a": part dies with, or outlives, the whole.

04 Common design patterns

Singleton	Only one instance of a class.
Factory	A method that creates objects.
Observer	Subscribers notified on change.
Idea	Reusable design templates.
Value	Shared vocabulary, proven structure.

05 Worked example: shapes

Shape	abstract: area() has no body.
Circle	is-a Shape; area = πr^2.
Rectangle	is-a Shape; area = w×h.
Loop	call s.area() on each shape.
Why it works	polymorphism picks the version.

06 Know the difference

is-a vs has-a	Inheritance models "is-a"; composition and aggregation model "has-a".	RELATIONS
Composition vs aggregation	Part dies with the whole (filled diamond) versus part lives on its own (hollow diamond).	LIFETIME
Overriding vs overloading	Redefine an inherited method versus same name with different parameters.	POLYMORPHISM
Abstraction vs encapsulation	Hiding how something works versus hiding the data behind methods.	CONCEPTS

07 Inheritance + overriding = polymorphism

The classes · one superclass, two subclasses that override speak()

```
class Animal:
  __init__(self, name): self.name = name
  speak(self): return self.name + " makes a sound"
class Dog(Animal):
  speak(self): return self.name + " says Woof"     # override
class Cat(Animal):
  speak(self): return self.name + " says Meow"     # override
```

Polymorphism trace · for a in [Dog("Rex"), Cat("Bella"), Animal("Thing")]: print(a.speak())

Object	Its class	speak() that runs	Output
Dog("Rex")	Dog	Dog.speak()	Rex says Woof
Cat("Bella")	Cat	Cat.speak()	Bella says Meow
Animal("Thing")	Animal	Animal.speak()	Thing makes a sound

The point: one call, a.speak(), gives a different result per object (polymorphism). The constructor and name came from Animal once (inheritance = code reuse).

Your turn · add class Cow(Animal) overriding speak() to "... says Moo"

Answer: Cow("Daisy").speak() returns "Daisy says Moo"; the constructor is inherited, so you do not rewrite it.

FINAL PASS BEFORE THE EXAM

Rapid exam tips

Eight things that lose marks if you slip on them. B3.2 is HL only. Skim before you walk in.

01

Inheritance = "is-a" and code reuse; a subclass extends a superclass.

02

Polymorphism = same call, different result, mainly through overriding.

03

Overriding (redefine a method) is not **overloading** (same name, different parameters).

04

Abstraction defines what must happen; subclasses decide how.

05

Don't confuse **"is-a"** (inheritance) with **"has-a"** (composition/aggregation).

06

Composition: part dies with the whole. **Aggregation**: part lives on.

07

Know a few **design patterns** (singleton, factory, observer) and what they do.

08

All of **B3.2 is HL only**; expect it in HL papers.