7.3 Inheritance and polymorphism

Name: _____ Class: _____ Date: _____ **Total: 20 marks**

KEY WORDS subclass • inheritance • superclass • polymorphism • override • overloading • abstract class • composition • aggregation

Objective

Build the skills to answer HL exam questions on **inheritance and polymorphism** – subclasses, overriding, abstract classes, composition and aggregation.

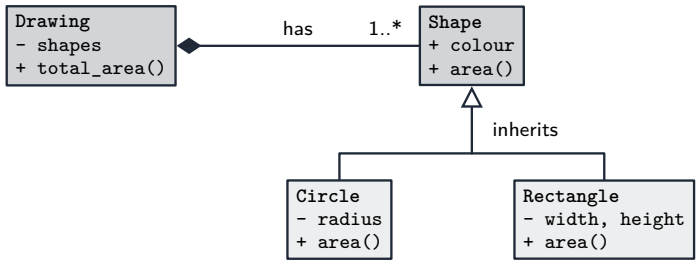
You must be able to:

- write a **subclass** that **inherits** from a **superclass** and calls its constructor with `super()`
- explain **polymorphism** and trace a loop that calls an **overridden** method
- tell overriding from **overloading**
- explain why an **abstract class** cannot be instantiated
- choose between inheritance ("is a"), **composition** and **aggregation** ("has a")

1 Key knowledge and worked examples

Study these first. Each one shows the method for a question type used later.

■ Key knowledge: the relationships



Relationship	Test	Part outlives whole?	UML
inheritance	"is a"		hollow triangle
composition	"has a"	no	filled diamond
aggregation	"has a"	yes	hollow diamond

■ Worked example: a subclass and polymorphism

```
class Ticket:
    def __init__(self, seat):
        self.seat = seat
    def price(self):
        return 10

class StudentTicket(Ticket):
    def price(self):                    # overrides Ticket.price
        return 6

class VipTicket(Ticket):
    def __init__(self, seat, lounge):
        super().__init__(seat)          # superclass sets seat
        self.lounge = lounge
    def price(self):
        return 25 if self.lounge else 18

sold = [Ticket("A1"), StudentTicket("A2"), VipTicket("V1", True)]
total = 0
for t in sold:
    total = total + t.price()
```

1. `StudentTicket` has no constructor of its own, so it uses `Ticket.__init__`.
2. Each `t.price()` runs the version in the object's own class: 10, then 6, then 25.
3. `total` is **41**. The loop never checks the type: that is polymorphism.

■ Key knowledge: overriding, overloading, abstract

- **Overriding**: a subclass method with the same name and parameters replaces the superclass's.
- **Overloading**: one class has several methods with one name but different parameters (Java: two constructors). Python uses default values instead.
- An abstract class (`class Shape(ABC)`, Java `abstract class Shape`) cannot be instantiated; every **abstract method** must be overridden before a subclass can be.

2 Practice

2.1 For the ticket code, state the value of (i) `StudentTicket("B4").seat`; (ii) `VipTicket("V2", False).price()`. [2]

2.2 State whether each is inheritance, composition or aggregation: (i) a `Laptop` and its `Keyboard`, built in; (ii) a `Teacher` and `Employee`; (iii) a `Playlist` and its `Songs`, which can be in many playlists. [3]

2.3 Distinguish overriding from overloading. [2]

3 Exam-style questions

3.1 Which keyword in Java makes `Bus` a subclass of `Vehicle`? [1]

- **A** `implements`
- **B** `extends`
- **C** `super`
- **D** `abstract`

3.2 `Animal` is abstract with the abstract method `speak()`. Which statement is true? [1]

- **A** `new Animal()` creates an object with no speech
- **B** every subclass that can be instantiated defines `speak()`
- **C** `speak()` in `Animal` runs for every animal
- **D** a subclass cannot add methods

3.3 A subclass constructor calls `super().__init__(name)`. Why? [1]

- **A** to make a second object
- **B** to run the superclass's code that sets the inherited attributes
- **C** to override the superclass
- **D** to make the class abstract

3.4 A delivery firm models its vehicles.

```
class Vehicle:
    def __init__(self, reg, load):
        self.reg = reg
        self.load = load              # kg carried now
    def max_load(self):
        return 500
    def can_add(self, kg):
        return self.load + kg <= self.max_load()

class Van(Vehicle):
    def max_load(self):
        return 1200

class Bike(Vehicle):
    def max_load(self):
        return 40
```

(a) State the output of `print(Van("V7", 900).can_add(250), Bike("B2", 30).can_add(15))`. [2]

(b) Explain how `can_add`, written once in `Vehicle`, gives different results for a van and a bike. [2]

(c) Write a subclass `Truck` whose maximum load is 8000 kg, plus 2000 kg if it has a trailer, given as a Boolean parameter of its constructor. [4]

(d) A `Depot` holds a list of vehicles; when the depot closes, the vehicles are moved to another depot. State the relationship between `Depot` and `Vehicle`, and justify it. [2]

Solutions

Each answer shows the steps, then the **result**.

2.1

- (i) `"B4"`: the constructor is inherited
- (ii) no lounge: **18**

2.2

- (i) **composition**
- (ii) **inheritance**: a teacher is an employee
- (iii) **aggregation**

2.3

- overriding: a subclass redefines a superclass method with the same name and parameters
- overloading: methods in one class share a name but have different parameters

3.1 B

- `class Bus extends Vehicle`

3.2 B

- an abstract method must be overridden

3.3 B

- the inherited attributes are set in one place

3.4

(a)

- van: $900 + 250 = 1150 \leq 1200$: **True**
- bike: $30 + 15 = 45 > 40$: **False**

(b)

- `can_add` calls `self.max_load()`
- each subclass overrides `max_load`, so the object's own version runs: polymorphism

(c) One answer:

```
class Truck(Vehicle):
    def __init__(self, reg, load, trailer):
        super().__init__(reg, load)
        self.trailer = trailer
    def max_load(self):
        if self.trailer:
            return 10000
        return 8000
```

- `class Truck(Vehicle)`
- constructor calls `super().__init__(reg, load)`
- stores `trailer`
- overrides `max_load` with 8000 or 10000

(d)

- **aggregation**
- the vehicles exist on their own and move to another depot when this one closes