Exercise walk-through

7.3
Inheritance and polymorphism

International Baccalaureate · IB Diploma · Computer Science · HL

Oop · 2.1

Reasoning

A class defines a type with state and behaviour. An object is an instance . Encapsulation controls access through an interface so operations can preserve invariants. Inheritance models an appropriate type relationship; composition models an object containing or using another object. encapsulation: Controlling access to state through an interface. instance: An individual object of a class.

Oop · executable trace

- Trace the state changes and boundary guard.
- Run the boundary case and compare the printed output.

```
class Account:
    def __init__(self, balance):
        self.balance = balance
    def withdraw(self, amount):
        if amount < 0 or amount > self.balance:
            return False
        self.balance -= amount
        return True
a, b = Account(15), Account(20)
print(a.withdraw(30), a.balance, b.balance)
print(a.withdraw(15), a.balance, b.balance)
```

Exercises 2.1·2.2

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]

Exercises 2.1·2.2

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

2.1

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

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]

2.2

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

Exercises 2.3·3.1

2.3 Distinguish overriding from overloading. [2]

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

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

Exercises 2.3·3.1

2.3 Distinguish overriding from overloading. [2]

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

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

3.1 B

- `class Bus extends Vehicle`

Exercises 3.2·3.3

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

Exercises 3.2·3.3

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

3.2 B

- an abstract method must be overridden

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

3.3 B

- the inherited attributes are set in one place

Setup for exercise 3.4

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
```

Exercise 3.4 (a)

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

Exercise 3.4 (a)

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

(a)

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

Exercise 3.4 (b)

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

Exercise 3.4 (b)

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

(b)

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

Exercise 3.4 (c)

(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]

Exercise 3.4 (c)

(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]

(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

Exercise 3.4 (d)

(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]

Exercise 3.4 (d)

(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]

(d)

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