Exercise walk-through

7.2 Encapsulation and UML class diagrams

International Baccalaureate · IB Diploma · Computer Science · HL

Instance and static members

Instance attribute

self.balance: one value per object.

Li and Sam each have their own.

Static (class) attribute

Account.count: one value for the whole class.

Each **constructor** adds 1: after two accounts it is 2.

Advantages and limits of OOP

Advantages

Models real things; easy to discuss.

Classes are reused; **encapsulation** protects data.

Disadvantages

More code and planning for a small program.

More memory; a poor design is hard to change.

Check yourself

UML: three compartments, - **private** or + **public**, with types

↓

Encapsulation: bundled, and hidden behind validating methods

↓

Assigning an object variable copies the reference

↓

HL: name the is-a or has-a test; give a polymorphism example

Exercises 2.1·2.2

2.1 Explain what is meant by encapsulation. [2]

2.2 State **two** advantages of making an attribute private and providing a mutator. [2]

Exercises 2.1·2.2

2.1 Explain what is meant by encapsulation. [2]

2.1

- the data and the methods that act on it are bundled in one class
- the data is hidden and can only be reached through those methods

---

2.2 State **two** advantages of making an attribute private and providing a mutator. [2]

2.2

- the mutator can reject invalid values, so the object stays valid
- the inside of the class can change without breaking code that uses it

Exercises 2.3·2.4

2.3 Draw the UML class diagram for the gym member in the worked example, with a constructor, `add_credits(n)`, `book()` and `get_credits()`. [4]

2.4 State the output of `acc = Account("Bo", -20.0)`, then `print(acc.get_balance())`

Exercises 2.3·2.4

2.3 Draw the UML class diagram for the gym member in the worked example, with a constructor, `add_credits(n)`, `book()` and `get_credits()`. [4]

2.3

- name compartment `Member`
- `- name: String`, `- credits: int`
- `+ Member(name)`, `+ add_credits(n): void`, `+ book(): bool`, `+ get_credits(): int`
- `-` on the attributes and `+` on the methods

2.4 State the output of `acc = Account("Bo", -20.0)`, then `print(acc.get_balance())`

2.4

- **0.0**: the constructor stores `max(0.0, -20.0)`

Exercises 3.1·3.2

3.1 In a UML class diagram, what does **-** before an attribute mean? [1]

- [ ] **A** it is public
- [ ] **B** it is private
- [ ] **C** it is static
- [ ] **D** it can be negative

3.2 Which method is an accessor? [1]

- [ ] **A** `set_age(a)`
- [ ] **B** `get_age()`
- [ ] **C** `birthday()`
- [ ] **D** `__init__(name, age)`

Exercises 3.1·3.2

3.1 In a UML class diagram, what does `-` before an attribute mean? [1]

- [ ] **A** it is public
- [ ] **B** it is private
- [ ] **C** it is static
- [ ] **D** it can be negative

3.1 B

- `-` private, `+` public

3.2 Which method is an accessor? [1]

- [ ] **A** `set_age(a)`
- [ ] **B** `get_age()`
- [ ] **C** `birthday()`
- [ ] **D** `__init__(name, age)`

3.2 B

- it returns a value and changes nothing

Exercise 3.3

3.3 Why should a mutator check a new value **before** changing the attribute? [1]

- **A** checking is faster before the change
- **B** a rejected value then leaves the object unchanged
- **C** accessors cannot read checked values
- **D** the constructor needs the old value

Exercise 3.3

3.3 Why should a mutator check a new value **before** changing the attribute? [1]

- ☐ **A** checking is faster before the change
- **B** a rejected value then leaves the object unchanged
- **C** accessors cannot read checked values
- **D** the constructor needs the old value

3.3 B

- check, then change

Exercise 3.4 (a)

3.4 A school locker system stores a locker's number, whether it is locked and a 4-digit code.

(a) Draw a UML class diagram for `Locker` with a constructor and the public methods `lock()`, `unlock(code)` and `is_locked()`. [4]

Exercise 3.4 (a)

3.4 A school locker system stores a locker's number, whether it is locked and a 4-digit code.

(a) Draw a UML class diagram for `Locker` with a constructor and the public methods `lock()`, `unlock(code)` and `is_locked()`. [4]

(a)

- name compartment `Locker`
- private attributes with types: `- number: int`, `- locked: bool`, `- code: String`
- constructor `+ Locker(number, code)`
- `+ lock(): void`, `+ unlock(code): bool`, `+ is_locked(): bool`

Exercise 3.4 (b)

3.4 A school locker system stores a locker's number, whether it is locked and a 4-digit code.

(b) Write the method `unlock(code)`. It unlocks the locker and returns `True` only if `code` matches the stored code. [3]

Exercise 3.4 (b)

3.4 A school locker system stores a locker's number, whether it is locked and a 4-digit code.

(b) Write the method `unlock(code)`. It unlocks the locker and returns `True` only if `code` matches the stored code. [3]

(b) One answer:

```
def unlock(self, code):
    if code == self._code:
        self._locked = False
        return True
    return False
```

- compares `code` with the stored code
- sets `locked` to `False` only on a match
- returns `True` or `False`

Exercise 3.4 (c)

3.4 A school locker system stores a locker's number, whether it is locked and a 4-digit code.

(c) Explain why the code attribute should be private and have **no** accessor. [2]

Exercise 3.4 (c)

3.4 A school locker system stores a locker's number, whether it is locked and a 4-digit code.

(c) Explain why the code attribute should be private and have **no** accessor. [2]

(c)

- other code cannot read the code and open any locker
- the only way in is through `unlock`, which checks the code