Exercise walk-through

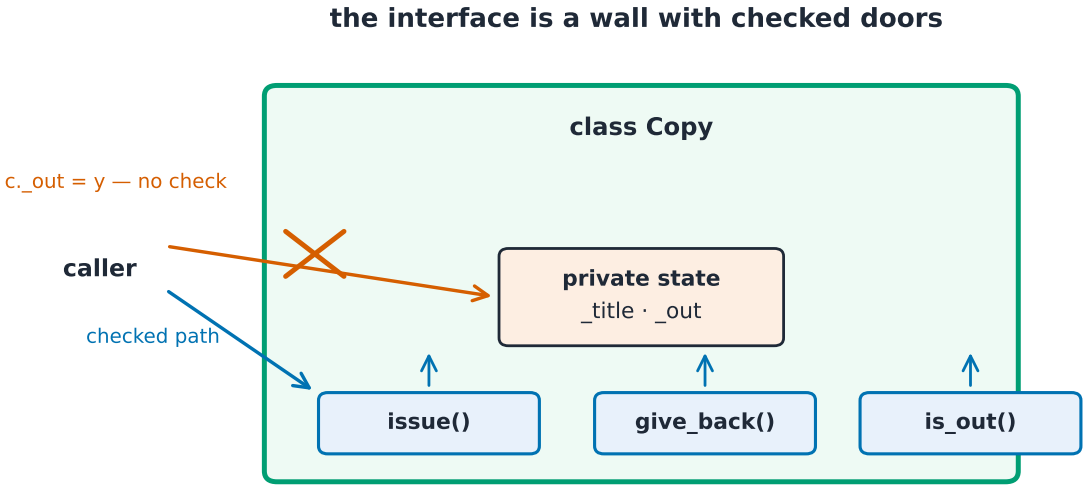
4.2 Encapsulation

University Computer Science

Encapsulation: the invariant behind the wall

1. **Gate** The **invariant** (a copy is out to exactly one borrower, or not out) holds if every change passes issue and give back - two guards, one proof.
2. **Leak** Writing the flag to a truthy-but-not-True value splits callers: if flag: reads out, if flag is True: reads available - a state no method could produce.
3. **Return** Handing back the internal list hands callers your private state; return a copy and expose available instead of get/set pairs.

Interface boundary



Public methods form a wall: every state change passes the checks; the fields behind the wall are private.

Exercises 2.1·2.3

2.1 A caller runs `issue()` twice on the same copy. State what the second call does and what the object reads afterwards. [3]

2.2 A method ends with `return self._loans`. State the risk and the fix. [3]

2.3 Name the invariant a `Loan` class protects when its day count must stay between 0 and 30. [3]

Exercises 2.1·2.3

2.1 A caller runs `issue()` twice on the same copy. State what the second call does and what the object reads afterwards. [3]

2.1

- Raises (already out) ✓.
- Object unchanged – still out to the same borrower ✓.

2.2 A method ends with `return self._loans`. State the risk and the fix. [3]

2.2

- The caller holds the very list the invariant depends on and can mutate it ✓.
- Return a copy ✓.

2.3 Name the invariant a `Loan` class protects when its day count must stay between 0 and 30. [3]

2.3

- Days stays within 0 to 30 between calls ✓.

Exercise 3.1

3.1 Design a `Savings` account class whose invariant is: balance is never negative and never above a fixed cap of 500.

Write the constructor check and the `deposit` method, stating every input each refuses and what the balance reads after each refusal. Explain why check-first means no rollback code is needed, and give the one-line cost argument for the wall. [8]

Exercise 3.1

3.1 Design a `Savings` account class whose invariant is: balance is never negative and never above a fixed cap of 500.

Write the constructor check and the `deposit` method, stating every input each refuses and what the balance reads after each refusal. Explain why check-first means no rollback code is needed, and give the one-line cost argument for the wall. [8]

3.1

- Constructor: refuse an opening balance outside 0 to 500 ✓.
- `deposit`: refuse a non-positive amount; refuse any amount that takes the balance past 500 ✓.
- After each refusal the balance reads exactly what it read before – the guard ran before any change ✓.
- No rollback: nothing was changed on the failing path ✓.
- The wall: repair cost scales with every reader of the field; validation cost was two `if` statements ✓.

Exercise 3.2

3.2 A teammate bypasses the methods and writes a truthy-but-not-True value into the account's private flag.

Explain how two callers can now disagree about the same object, naming the two readings. State what the repair cost scales with and what the validation cost was, and explain why exposing behaviour beats get-and-set pairs for everything. [7]

Exercise 3.2

3.2 A teammate bypasses the methods and writes a truthy-but-not-True value into the account's private flag.

Explain how two callers can now disagree about the same object, naming the two readings. State what the repair cost scales with and what the validation cost was, and explain why exposing behaviour beats get-and-set pairs for everything. [7]

3.2

- Truthy-but-not-True: `if flag:` reads one way, `if flag is True:` the other ✓.
- Two callers disagree about one object – a state no method could produce ✓.
- Repair cost scales with every reader of the flag; the validation cost was the guards at the gate ✓.
- Behaviour (e.g. an `available` query) states a promise; get/set for everything exposes storage and checks nothing ✓.