Object-Oriented Programming

Curated computer science study track (not a degree, no licensure claim). Original explanations and practice only, following the reviewed course handout.

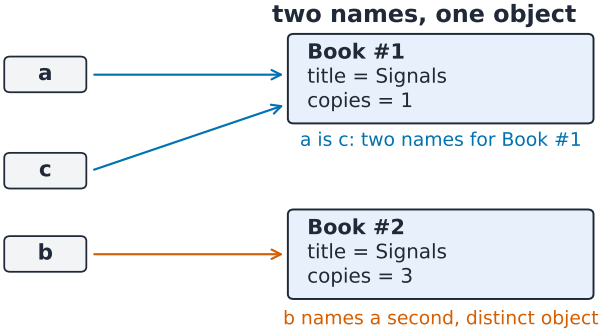
From a script to a system others can change

- 4.1-4.2 build the object (classes, references, walls); 4.3-4.4 connect them; 4.5-4.6 keep the system alive.
- Every class is a contract with its callers; every example here is a program that ran.

Objects and classes: blueprints, names, identity

1. **Blueprint** A class is a blueprint; an object is one thing built from it; the constructor seeds the **instance attributes**, methods take self.
2. **Lookup** Attribute lookup walks outward: **instance** dict, then class dict, then bases - assigning through an instance shadows the class value.
3. **Trace** Two Books made, take twice through alias a: copies reads 1 for a and c (one object, two names), 3 for b; the class counter reads 3.

Object and reference graph



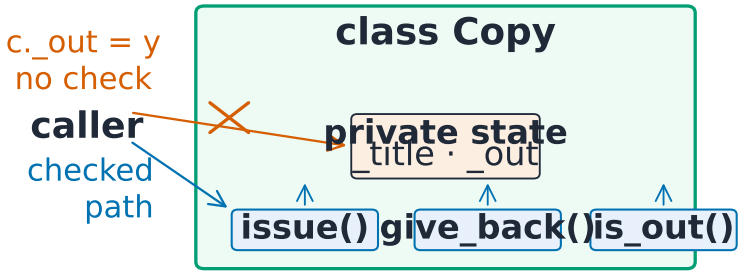
Two names on one object: a and c alias the same Book, b is a second, distinct object.

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

the interface is a wall with checked doors

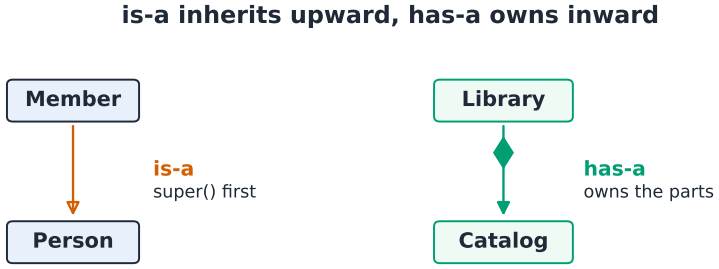


Public methods form a wall: every state change passes the checks; the fields behind the wall are private.

Composition and inheritance: is-a vs has-a

1. **Chain** A Member is a Person: the derived constructor calls the base first, so base state exists before any derived line runs.
2. **Delegate** A Library has a Catalog: five copies added, two issued, available asks the parts and counts three - swap the part, keep the whole.
3. **Fork** An override that copies the base body goes stale the day the base gains a line; delegate to super and subtract - the bonus survives.

Class relationship diagram



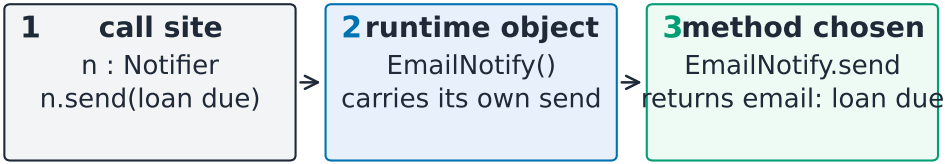
Two connection kinds: **inheritance** says is-a (hollow triangle to the base); **composition** says has-a (filled diamond to the part).

Interfaces and polymorphism: one call, many behaviours

1. **Contract Duck typing** accepts any object with the right methods; a protocol writes the contract down without forcing **inheritance**.
2. **Dispatch** One loop over three senders prints email, sms and queue lines - the call site never learns the concrete classes.
3. **Substitute** A substitute must accept at least what the **interface** promises and break no promise: preconditions only widen, guarantees only grow.

Dynamic dispatch sequence

one call, three steps, one winner



the annotation names the interface; the object decides the method.
Swap in SmsNotify and only steps 2 and 3 change.

One call, three steps: the variable names the **interface**, the object carries the **runtime type**, dispatch picks the send of that type.

Exceptions and error handling: loud, typed, cleaned up

1. **Hierarchy** Loan error at the base, renewal and limit beneath: a parent except catches the child, and the caller picks the grain.
2. **Unwind** A raise at 20 plus 15 over 30 travels out frame by frame, running each finally, until a handler matches - the loan still reads 20 after.
3. **Cleanup** A lock released in finally comes back False on both paths; a returned minus one is ignorable, an exception is not.

Exception flow

the raise travels down the stack until caught



handler chosen by type: a parent except also catches this child — state left as it was before the call

> Raise in renew unwinds issue and main until a matching handler; finally runs on the way out, on every path.

Testing and design quality: pin the behaviour

1. **AAA Arrange** the objects, **act** with one call, **assert** the observable outcome - renew within limit lands 27, exactly 30 is allowed, past 30 raises with state unchanged.
2. **Grid** Behaviours down, tests across: the empty cell is the next test, and behaviour names survive renames that method names do not.
3. **Refactor** Fines move to their own calculator (one reason to change) and the three renew tests still pass: coupling down, cohesion up.

Test coverage map

each behaviour needs a pin

behaviour ↓ · test →	test_renew_ok	test_renew_boundary	test_renew_limit	
renew within limit	pinned			
renew to exactly 30		pinned		
renew past the limit			pinned	
give back twice raises				no test yet

Behaviours down, tests across: each cell is a behaviour a test pins; the empty cell is next to write.

Check yourself

- After c = a, the names a and c…? (One object.)
- A second issue on an out copy…? (Raises; state unchanged.)
- The base constructor runs…? (First, via super.)
- finally runs…? (On every path.)