

Exercise walk-through

4.6

Testing and design quality

University Computer Science

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, at the limit 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_limit	test_renew_owner
renew within limit	pinned		
renew at the limit		pinned	
renew, wrong owner			pinned
return a copy twice			

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.)

Exercises 2.1-2.2

2.1 A behaviour reads: returning an already-returned copy raises. Write the test's name. [3]

2.2 State which survives a method rename: a behaviour-named test or a method-named test. [3]

Exercises 2.1-2.2

2.1 A behaviour reads: returning an already-returned copy raises. Write the test's name. [3]

2.1

- e.g. `test_return_twice_raises` ✓.
- Named for the behaviour, not the method it calls ✓.

2.2 State which survives a method rename: a behaviour-named test or a method-named test. [3]

2.2

- The behaviour-named test survives ✓.
- The method-named test breaks on the rename while pinning nothing that changed ✓.

Exercise 2.3

2.3 Fines leave `Loan` for a `FineCalculator`. Name the principle and what it gives each class. [3]

Exercise 2.3

2.3 Fines leave `Loan` for a `FineCalculator`. Name the principle and what it gives each class. [3]

2.3

- Single responsibility principle ✓.
- One reason to change per class: loans change for loan rules, fines for fine rules ✓.

Exercise 3.1

3.1 A renewal function refuses day counts above 30 and leaves state unchanged when it refuses.

Write three unit tests covering: a within-limit renewal from 20 by 7; an at-limit refusal from 20 by 11; and the state after the refusal. Each test arranges, acts and asserts, and none touches private internals. State what each assert pins and why the tests share no mutable state. [8]

Exercise 3.1

3.1 A renewal function refuses day counts above 30 and leaves state unchanged when it refuses.

Write three unit tests covering: a within-limit renewal from 20 by 7; an at-limit refusal from 20 by 11; and the state after the refusal. Each test arranges, acts and asserts, and none touches private internals. State what each assert pins and why the tests share no mutable state. [8]

3.1

- Test 1: arrange a 20-day loan, act `renew(loan, 7)`, assert the result is 27 ✓.
- Test 2: arrange a 20-day loan, act `renew(loan, 11)`, assert it raises the limit error ✓.
- Test 3: assert the loan still reads 20 after the refusal ✓.
- Each assert pins one promised outcome – the value, the raise, the unchanged state ✓.
- Each test builds its own loan: no shared mutable state, any order runs green ✓.

Exercise 3.2

3.2 The fine calculation moves out of `Loan` into a `FineCalculator` class at 0.50 per day late.

State what the three renewal tests do through the move and what that proves. Name the principle the move follows. State what happened to coupling and to cohesion, and why asserting private internals in tests would have broken this refactor. [7]

Exercise 3.2

3.2 The fine calculation moves out of `Loan` into a `FineCalculator` class at 0.50 per day late.

State what the three renewal tests do through the move and what that proves. Name the principle the move follows. State what happened to coupling and to cohesion, and why asserting private internals in tests would have broken this refactor. [7]

3.2

- They still pass, untouched – behaviour preserved while structure moved ✓.
- The move follows the single responsibility principle ✓.
- Coupling went down (callers depend on less); cohesion went up (one purpose per class) ✓.
- Asserting private internals couples tests to storage – the move would have broken them ✓.