Solutions

Marking: **M1** a correct method, **A1** the correct answer, **B1** an independent mark.

Each answer shows the steps, then the **result**.

2.1

button and page: presentation

loan limit rule: domain

SQL: storage

2.2

low cohesion: three unrelated jobs, so many modules depend on it

split it into dates, fines (domain) and messaging modules

2.3

email can be swapped for SMS without changing the domain

tests can use a fake notifier that records messages, with no real email

2.4

presentation → domain: `borrow(member, copy)`

domain checks the limit, then domain → storage: `save loan`

storage → domain → presentation: loan with due date; page shows "confirmed"

3.1

(a) the loan rule is copied in three handlers, so the copies can disagree; the rules cannot be tested without a web server and a database

(b) pages call a domain `LoanService`; the service holds the rules and calls a `LoanRepository` interface; a SQL class implements the repository

(c) old: 3 handlers (and any trigger), at least 3; layered: 1 constant in the domain

3.2

(a) the result depends on the day the test runs, so testing "3 days late" needs waiting or changing the computer's clock

(b) `LoanService` takes a `Clock` interface in its constructor; tests pass a fake clock fixed at a chosen date

(c) dependency inversion (or dependency injection)

3.3

(a) each service can be deployed and scaled on its own; a fault or a slow part is contained

(b) network calls can fail, so each call needs timeouts and retries; more servers, pipelines and logs to run, for only 4 people

(c) $0.995^3 \approx \mathbf{0.985}$, i.e. 98.5%

exercises.lol exercises.com.cn

(d) a modular monolith: one program with clear layers and modules; split out a service later only if one part needs to scale or change separately