


Cursul 6

# Contracte și invarianți

**Analiza și Modelarea Sistemelor Software**

Traian-Florin Șerbănuță

# Ce promite o operație?

Când o operație refuză o cerere, ce trebuie să rămână adevărat?

# Citatul zilei

> **„Precondiția obligă clientul; postcondiția obligă furnizorul.”**
>
> Original: “The precondition binds the client, the postcondition binds the supplier.”

— **Bertrand Meyer**

Sursa: Object-Oriented Software Construction, ediția a doua, „Using assertions”, p. 873

# Când o cerere este refuzată

Gândiți-vă la o aplicație pe care ați folosit-o sau ați scris-o: o plată, o rezervare, un formular trimis.

**Cererea a fost refuzată. Ce v-ați așteptat să rămână neschimbat? De unde ați știi că a rămas așa?**

Ați întâlnit un refuz după care se schimbase totuși ceva?

# Un refuz la dulapurile muzeului

La intrarea într-un muzeu, vizitatorii primesc dulapuri pentru bagaje. Un **registru**, componenta responsabilă, ține asocierile vizitator–dulap și răspunde la două cereri: atribuirea unui dulap și eliberarea lui.

**Regula convenită cu muzeul (beneficiarul):** un vizitator are cel mult un dulap, iar un dulap are cel mult un vizitator; o cerere imposibilă este refuzată.

Un vizitator cere un dulap deja ocupat. Registrul răspunde `refuz`. **Este suficient rezultatul întors sau trebuie să verificăm și registrul de după cerere?**

*Exemplu didactic; registrul doar ține evidența, nu deschide dulapuri. Lucrăm pe hârtie; toate datele necesare sunt pe slide-uri.*

# Un refuz, două registre posibile

**Regulă:** atribuirea este refuzată dacă vizitatorul are deja un dulap sau dulapul cerut este ocupat. La refuz, **întregul registru rămâne identic**.

**Înainte:** V1–D1, V2–D2. Cerem **atribuie(V1, D2)**.

| Propunere | Rezultat | După |
|---|---|---|
| P1 | `refuz` | V1–D1, V2–D2 |
| P2 | `refuz` | V2–D2 |

Individual, apoi discutăm: care respectă regula? De ce faptul că fiecare dulap are cel mult un vizitator nu este suficient?

# Ce descrie un contract

Un **contract** completează interfața unei operații din cursul 5: pe lângă semnătură și sensul rezultatelor, spune ce promite componenta fiecărui apelant, fără să descrie pașii prin care o realizează:

- **condițiile apelului:** ce presupunem despre cerere și despre registru;
- **rezultatele garantate:** ce întoarce și cum arată registrul după, la succes și la refuz;
- **regulile păstrate de toate operațiile:** ce respectă orice registru, înainte și după fiecare operație.

Pentru registrul muzeului le precizăm pe rând și le verificăm pe scenarii. Nu modelăm autentificarea, concurența sau erorile tehnice.

# Precizăm promisiunea

Ce presupunem și ce garantăm?

# Regulile registrului

| Regulă | Conținut |
|---|---|
| R1 (context) | Identificatorii vizitatorilor și ai dulapurilor sunt cunoscuți, registrul inițial respectă R2 (unicitate), iar apelurile sunt secvențiale: fiecare începe după încheierea celui anterior. Nu discutăm autentificarea, concurența sau erorile tehnice. |
| R2 (unicitate) | Un vizitator are cel mult un dulap, iar un dulap are cel mult un vizitator. |
| R3 (atribuire) | **atribuie(v,d)** reușește exact când v nu are dulap și d este liber; altfel refuză fără modificări. |
| R4 (eliberare) | **eliberează(v)** elimină asocierea lui v; dacă lipsește, întoarce `fara_dulap`, fără modificări. |
| R5 (efect limitat) | La succes, toate celelalte asocieri se păstrează. |

# Trei întrebări diferite

**Precondiție:** ce trebuie să fie adevărat la apel pentru ca operația să garanteze ceva?

**Postcondiție:** ce garantăm la întoarcere, dacă precondiția a fost respectată?

**Invariant:** ce proprietate trebuie să fie adevărată în orice stare, înainte și după operații?

Postcondiția poate depinde de o **condiție de succes**: dintre apelurile admise de precondiție, ea decide care reușesc și care primesc refuzul promis.

În registrul nostru, precondiția cere identificatori cunoscuți, postcondiția dă efectul exact al atribuirii, iar invariantul este unicitatea vizitator–dulap.

# „Dulap liber”: ține de precondiție sau de condiția de succes?

**R1 (context):** apelăm secvențial, cu identificatori cunoscuți, într-un registru care respectă R2 (unicitate).

**R3 (atribuire):** dacă dulapul este ocupat, **atribuie** întoarce `refuz` și păstrează întregul registru.

Prin urmare, într-un singur contract, „dulap liber” este **o parte a condiției de succes**, alături de „vizitator fără dulap”, nu o precondiție a acestei operații publice. Dacă o impunem ca precondiție, nu mai promitem nimic pentru un dulap ocupat.

**Alternativă: câte un contract pentru fiecare caz.** „Dulap liber” devine parte din precondiția contractului de succes, iar „dulap ocupat” intră în cea a contractului de refuz. Atunci trebuie să verificăm că împreună acoperă toate apelurile admise.

# Invariantul privește toate operațiile

**R2 (unicitate):** un vizitator are cel mult un dulap, iar un dulap are cel mult un vizitator.

O stare este **validă** dacă respectă toți invarianții. Aici avem unul singur, R2.

| Registru | Valid? |
|---|---|
| Nicio asociere | ? |
| V1–D1, V2–D2 | Da. |
| V1–D1, V1–D3 | Nu: V1 are două dulapuri. |
| V1–D1, V2–D1 | Nu: D1 are doi vizitatori. |

Proprietatea trebuie păstrată după atribuire, eliberare și refuz. Un registru gol este valid?

# Clasificați afirmațiile

**Reguli:** identificatorii sunt cunoscuți, iar apelurile sunt secvențiale; un vizitator are cel mult un dulap, iar un dulap cel mult un vizitator; atribuim dacă vizitatorul nu are dulap și dulapul este liber, altfel refuzăm fără modificări.

1. Identificatorul dulapului este cunoscut.
2. Dulapul cerut este liber.
3. La `refuz`, registrul de după este identic cu cel de dinainte.
4. Niciun vizitator nu apare în două asocieri.

Individual, apoi discutăm: ce rol are fiecare afirmație în acest contract? La 3 și 4, verificăm o singură stare sau relația dintre două stări?

# Scriem contractul complet

Ce se modifică și ce trebuie să rămână identic?

# Atribuire: cazul de succes

**R1 (context):** identificatorii sunt cunoscuți, registrul este valid, iar apelurile sunt secvențiale.

**R3 (atribuire):** dacă v nu are dulap și d este liber, operația întoarce `atribuit`.

**După:** adăugăm exact asocierea v–d. **R5 (efect limitat):** toate celelalte asocieri se păstrează.

**Scenariul S1 (V3 primește D3):** înainte V1–D1, V2–D2; V3 nu are dulap, D3 este liber. **atribuie(V3,D3)** produce V1–D1, V2–D2, **V3–D3**.

# Atribuire: cazul de refuz

**R3 (atribuire):** dacă v are deja un dulap **sau** d este ocupat, operația întoarce `refuz`, iar întregul registru rămâne identic.

**B:** V1–D1, V2–D2; V3 fără dulap, D3 liber.

| Scenariu independent din B | Motiv de refuz |
|---|---|
| S2: **atribuie(V3,D2)** | Dulap ocupat. |
| S3: **atribuie(V1,D3)** | Vizitator deja asociat. |
| S4: **atribuie(V1,D2)** | Ambele condiții împiedică succesul. |

În toate trei, registrul de după este exact B. Nu ștergem o asociere existentă pentru a face cererea posibilă.

# O notație scurtă pentru efectul exact

**A** = mulțimea asocierilor înainte; **A′** = mulțimea după.

**R3 și R5 (atribuire, efect limitat), pe scurt:**

- dacă v nu are dulap în A și d este liber în A: `atribuit`, **A′ = A ∪ {(v,d)}**;
- altfel: `refuz`, **A′ = A**.

**Formal,** cu $A[v]$ = dulapurile lui v în A și $A^{-1}[d]$ = vizitatorii lui d în A:

- dacă $A[v] = \emptyset \wedge A^{-1}[d] = \emptyset$ atunci *rezultat* = `atribuit` $\wedge\ A' = A \cup \{(v, d)\}$;
- dacă $A[v] \neq \emptyset \vee A^{-1}[d] \neq \emptyset$ atunci *rezultat* = `refuz` $\wedge\ A' = A$.

Reuniunea adaugă perechea; egalitatea păstrează întregul registru. Care parte a formulei exclude dispariția unei asocieri care nu privește cererea?

# Completați rezultatul, fără să scrieți cod

**B:** V1–D1, V2–D2; V3 fără dulap, D3 liber. Fiecare rând repornește din B.

**Reguli:** atribuim numai dacă v nu are dulap și d este liber; altfel refuzăm fără modificări. La succes adăugăm numai perechea cerută.

| Acțiune | Rezultat și întregul registru după |
|---|---|
| **atribuie(V3,D3)** | ? |
| **atribuie(V1,D3)** | ? |
| **atribuie(V1,D1)** | ? |

Individual, apoi discutăm: de ce ultimul rând nu este automat un succes?

# Eliberarea și repetarea

**R4 (eliberare):** dacă v are dulap, eliminăm numai asocierea lui și întoarcem `eliberat`; altfel întoarcem `fara_dulap`, fără modificări. **R5 (efect limitat):** celelalte asocieri se păstrează.

**Scenariul S5 (eliberare repetată), doi pași succesivi:**

| Înainte | Acțiune | Rezultat și după |
|---|---|---|
| V1–D1, V2–D2 | **eliberează(V1)** | `eliberat`; V2–D2. |
| V2–D2 | **eliberează(V1)** | `fara_dulap`; V2–D2. |

Același registru final după repetare nu înseamnă același rezultat întors.

# Ce lipsește din această promisiune?

**Propunere:** „După **eliberează(V1)**, V1 nu mai are dulap.”

**Înainte:** V1–D1, V2–D2. **După propus:** registru gol.

**Regulile complete:** eliminăm numai asocierea lui V1; toate celelalte asocieri se păstrează. Fără asociere inițială: `fara_dulap`, registru identic.

Propunerea scurtă este satisfăcută? Contractul complet este respectat? Reformulați postcondiția în cuvinte.

# Încercăm să infirmăm propunerea

Un invariant respectat poate ascunde o operație greșită?

# Urmăriți operația propusă

**Pseudocod didactic, construit pentru verificare:**

```
atribuie(v, d):
    elimină asocierea existentă a lui v, dacă există
    dacă d este ocupat: întoarce refuz
    adaugă v-d
    întoarce atribuit
```

**B:** V1–D1, V2–D2; V3 fără dulap, D3 liber.

**Reguli:** un vizitator are cel mult un dulap, iar un dulap cel mult un vizitator; atribuirea reușește numai dacă vizitatorul nu are dulap și dulapul este liber; altfel refuz, cu întregul registru identic.

Pe hârtie, apoi discutăm: găsiți două apeluri independente din B: unul pentru care operația întoarce `refuz`, dar modifică registrul; altul pentru care întoarce `atribuit`, deși contractul cere refuz. Scrieți registrul final al fiecăruia și verificați unicitatea.

# Decidem înainte să modificăm

**R3 (atribuire):** atribuirea reușește numai dacă v nu are dulap și d este liber; altfel refuz fără modificări.

Corecția noastră, în contextul **apelurilor secvențiale**:

```
atribuie(v, d):
    dacă v are dulap sau d este ocupat: întoarce refuz
    adaugă v–d
    întoarce atribuit
```

Toate celelalte asocieri se păstrează. De ce primul pas nu are nevoie să „elibereze” ceva?

# De ce se păstrează unicitatea?

**R2 (unicitate):** un vizitator are cel mult un dulap, iar un dulap are cel mult un vizitator.

**Atribuire acceptată:** asociem un vizitator fără dulap cu un dulap liber.

**Eliberare:** eliminăm cel mult o pereche.

**Refuz sau `fara_dulap`:** păstrăm registrul identic.

Pornind dintr-un registru valid, explicați pentru fiecare caz de ce nu poate introduce o a doua asociere pentru același vizitator sau dulap.

# Ce poate și ce nu poate stabili o verificare?

**Contractul nostru:** pentru identificatori cunoscuți, un registru valid și apeluri secvențiale, refuzul păstrează tot registrul.

**Vorbim despre orice implementare,** nu despre cele de pe slide-urile anterioare. Nu presupunem că respectă contractul: tocmai asta vrem să aflăm.

| Verificare, când contractul cere refuz | Ce aflăm dacă trece |
|---|---|
| Rezultatul întors este `refuz` | Numai rezultatul, nu și starea de după. |
| Unicitatea este păstrată | Starea finală respectă acel invariant. |
| Registrul de după este identic cu cel de dinainte | Refuzul păstrează starea în scenariul verificat. |

Niciunul dintre aceste exemple nu verifică încuietori fizice, concurență sau identificatori necunoscuți.

# Contracte scrise lângă cod

| Limbaj | Unde scriem contractul |
|---|---|
| Java | **JML**, în comentarii speciale; instrument: OpenJML |
| C | **ACSL**, în comentarii speciale; instrument: Frama-C |
| C++ | `pre`, `post`, în proiectul C++26, încă disputate |
| Rust | Instrumente externe: Kani, Prusti, Verus |
| Python | Biblioteci, de exemplu icontract |

**În JML:** `requires` este precondiția, `ensures` postcondiția, `\old` starea de dinainte, iar `\result` rezultatul întors. Un instrument poate citi aceste adnotări și le poate verifica.

# Contractul și instrucțiunea `assert`

**Reguli:** atribuim dacă v nu are dulap și d este liber; altfel `refuz`, cu registrul identic. Unicitatea vizitator–dulap se păstrează. *Exemplu didactic, nerulat:*

```
Rezultat atribuie(Vizitator v, Dulap d) {
    if (areDulap(v) || !eLiber(d)) return REFUZ;
    Set<Asociere> inainte = Set.copyOf(asocieri);
    asocieri.add(new Asociere(v, d));
    assert respectaUnicitatea(asocieri);
    assert asocieri.equals(cuAsocierea(inainte, v, d));
    return ATRIBUIT;
}
```

De ce refuzul este `if`, nu `assert`? Ce spune un `assert` trecut despre alte apeluri?

# Verificăm un contract delegat

O formulare convingătoare spune tot ce avem nevoie?

# Ce cerem asistentului AI?

> Propune contractele pentru atribuie și elibereaza: precondiții, condiții de succes, postcondiții pentru fiecare rezultat, invarianți și ce rămâne neschimbat. Verifică S1–S6. Nu scrie cod. Răspunde în română, în maximum 400 de cuvinte.

**Context:** identificatori cunoscuți; apeluri secvențiale; cel mult un dulap pentru fiecare vizitator și invers. Atribuim dacă vizitatorul nu are dulap și dulapul este liber; altfel refuz fără modificări. Eliberăm numai asocierea vizată; dacă lipsește, `fara_dulap`. Restul se păstrează.

**Date:** B = V1–D1, V2–D2; V3 fără dulap, D3 liber. **Scenariile S1–S6:** succesul, fiecare motiv de refuz, eliberarea repetată și repetarea atribuirii.

# Ce promite această propunere?

**Analistul AI — Claude Haiku 4.5, 27 octombrie 2026; fragmente din contract și din verificare:**

> **Condiția de succes:** v nu are dulap asociat. d este liber (nu e ocupat).
>
> **Postcondițiile pentru eșec (`refuz`):** Registrul este identic cu cel inițial [...]
>
> **Invarianți:** Unicitatea: un vizitator ↔ cel mult un dulap; un dulap ↔ cel mult un vizitator. [...] Toate asocierile (v', d') cu v' ≠ v și d' ≠ d se păstrează.
>
> **Verificare S1–S6:** [...] S1 și S5.1 reușesc [...]; S2–S4, S5.2, S6 se refuză [...]

**Reguli:** succes numai dacă v nu are dulap și d este liber; altfel refuz, cu registrul identic; la succes se păstrează celelalte asocieri; eliberarea fără asociere întoarce `fara_dulap`.

Ce rol are fiecare afirmație? Care se verifică într-o singură stare? Ce întoarce al doilea pas din scenariul S5 (eliberare repetată a lui V1)?

# Promptul evaluatorului AI

**Evaluator (agent AI de revizuire), context nou:** enunțul, sarcina analistului AI și răspunsul integral; fără corecțiile noastre.

**Instrucțiunea exactă, la începutul promptului:**

> Ești evaluator (agent AI de revizuire). Verifică propunerea exclusiv față de enunț și sarcină. Pentru fiecare observație indică fragmentul, regula și un scenariu concret. Verifică precondițiile, condițiile de succes, fiecare rezultat, efectul limitat și invarianții. Confirmă și afirmațiile corecte; nu presupune că trebuie să existe greșeli. Nu scrie cod. Răspunde în română, în maximum 350 de cuvinte.

**Predicție, înainte de a vedea răspunsul:** va observa evaluatorul AI că păstrarea celorlalte asocieri este trecută printre invarianți?

# Revizuirea are nevoie de propria verificare

**Evaluatorul AI — Claude Haiku 4.5, captură din 27 octombrie 2026; fragmente:**

> Propunerea este **complet corectă** [...]
>
> Propunerea: “[...] Toate asocierile (v’, d’) cu v’ ≠ v și d’ ≠ d se păstrează.” [...] ✓ Corect și precis.
>
> **Concluzie:** Propunerea nu conține erori.

**Reguli:** un invariant trebuie să fie adevărat în orice stare; păstrarea celorlalte asocieri compară două stări, ca R5 (efect limitat). Eliberarea fără asociere întoarce `fara_dulap`.

Evaluatorul AI are dreptate că regula „toate asocierile … se păstrează” este corectă. Este suficient? Ce ar fi trebuit să mai verifice?

# Decizia noastră

**Păstrăm** din propunere condiția de succes, refuzul cu registrul identic și unicitatea ca invariant.

**Corectăm** două puncte:

- păstrarea celorlalte asocieri iese dintre invarianți: este postcondiție, R5 (efect limitat);
- a doua eliberare din scenariul S5 (eliberare repetată) întoarce `fara_dulap`, nu `refuz`.

**Contraexemplul care justifică prima corecție:** din B, după **atribuie(V3,D3)**, un registru care conține numai V3–D3 respectă unicitatea, dar pierde V1–D1 și V2–D2.

# Transferăm promisiunea

Aceleași întrebări, alte reguli.

# O trusă pentru fiecare tehnician

**Enunț nou:** identificatorii sunt cunoscuți, iar apelurile sunt secvențiale; un tehnician are cel mult o trusă de scule, iar o trusă cel mult un tehnician. Împrumutăm o trusă liberă unui tehnician fără trusă. Repetarea aceluiași împrumut întoarce `deja_imprumutat`; celelalte conflicte întorc `refuz`. Numai un împrumut nou modifică registrul, adăugând perechea cerută.

**Înainte:** tehnicianul T1 are trusa K1; T2 nu are trusă; trusa K2 este liberă.

1. Formulați condiția de succes și starea de după **împrumută(T2,K2)**.
2. Independent, din aceeași stare: **împrumută(T1,K2)**. Ce rămâne identic?
3. Tot din starea inițială: **împrumută(T1,K1)**. Ce diferă față de muzeu?