AVANSAT · POO ÎN C++ · LECȚIA 10

Proiect final: sistem de gestiune pentru o bibliotecă (toate conceptele POO la un loc)

Durată: 180 min Nivel: avansat / pregătire pentru facultate și industrie Înainte: lecțiile oop-01, oop-02, oop-03, oop-04, oop-05, oop-06, oop-07, oop-08, oop-09

CE VEI ȘTI DUPĂ LECȚIE

- Citești și înțelegi un program de 300 de rânduri care combină toate conceptele din pista POO.
- Recunoști în cod: RAII, moștenire și polimorfism, Factory, Strategy, Observer, Command, excepții, smart pointers, șabloane.
- Organizezi un proiect C++ pe fișiere (header-e și surse) și îl compilezi cu CMake.
- Testezi codul cu teste automate și extinzi proiectul pe etape, fără să strici ce funcționa.

1. Ce construim

Un program de gestiune pentru o bibliotecă școlară, controlat prin comenzi text. Este un proiect mic, dar construit „ca în industrie”: fiecare concept din lecțiile oop-01 ... oop-09 are un rol real, nu decorativ.

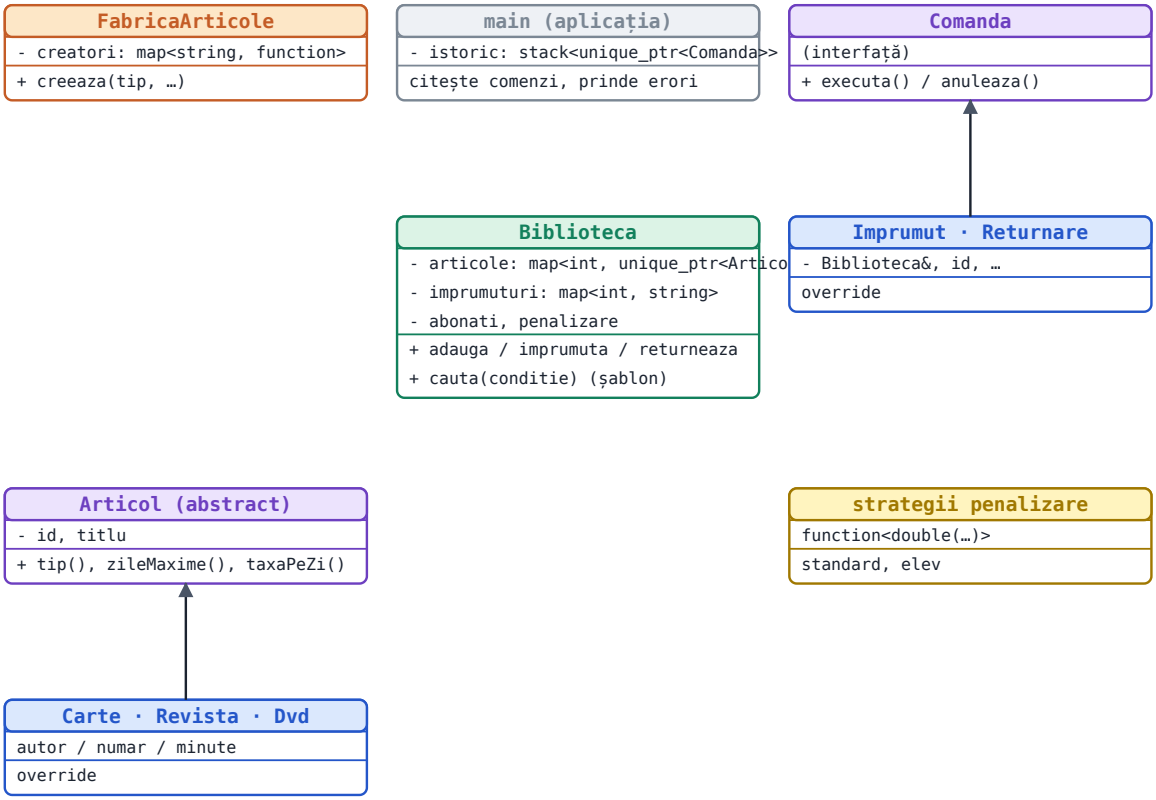
Cerințe funcționale:

1. Biblioteca are **articole** de mai multe tipuri: cărți, reviste, DVD-uri; fiecare tip are altă durată de împrumut și altă taxă de întârziere.
2. Un articol se poate **împrumuta** (o singură persoană o dată) și **returna**; la returnare se calculează **penalizarea**.
3. Regimul de penalizare se poate schimba (**standard** sau **elev**: jumătate, plafonat la 20 de lei).
4. Orice împrumut sau returnare se poate **anula** (`undo`).
5. La fiecare împrumut și returnare se trimit **notificări** (aici, afișare; în realitate, e-mail sau SMS).
6. Comenzile greșite produc mesaje de eroare clare, fără ca programul să se oprească.

Comenzile:

Comanda	Exemplu	Efect
`adauga tip id titlu detalii`	`adauga carte 1 Amintiri Creanga`	adaugă un articol
`imprumuta id membru`	`imprumuta 1 Ana`	împrumută
`returneaza id zile`	`returneaza 1 25`	returnează după un număr de zile, cu penalizare
`undo`	`undo`	anulează ultima operație de împrumut/returnare
`regim standard` / `regim elev`	`regim elev`	schimbă strategia de penalizare
`disponibile`	`disponibile`	listează articolele neîmprumutate

2. Arhitectura



Arhitectura proiectului

Concept (lecția)	Unde apare în proiect
clase, `const`, liste de inițializare (oop-01)	toate clasele; metode de citire `const`
regula lui 0 (oop-02)	nicio clasă nu are destructor scris: `unique_ptr`, `map`, `string` se ocupă de tot
`operator<<` și `friend` (oop-03)	afișarea unui `Articol`
moștenire, `virtual`, clasă abstractă (oop-04)	`Articol` → `Carte`, `Revista`, `Dvd`
SOLID, injectare (oop-05)	`Biblioteca` nu știe ce tipuri de articole există, nici cine primește notificările
șabloane (oop-06)	`Biblioteca::cauta(Pred)` acceptă orice condiție
excepții, `unique_ptr`, lambda (oop-07)	`EroareBiblioteca`; colecții de `unique_ptr`; abonați și creatori ca lambda
Factory (oop-08)	`FabricaArticole` cu registru
Strategy, Observer, Command (oop-09)	penalizări, notificări, undo

3. Codul complet

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <string>
#include <vector>
#include <map>
#include <memory>
#include <functional>
#include <stack>
#include <stdexcept>
#include <algorithm>
using namespace std;

// ------------------------------------------------------------ erori
class EroareBiblioteca : public runtime_error
{
public:
    explicit EroareBiblioteca(const string &m) : runtime_error(m) {}
};

// ------------------------------------------------------------ articole (moștenire +
polimorfism)
class Articol
{
    int id_;
    string titlu_;
public:
    Articol(int id, string titlu) : id_(id), titlu_(std::move(titlu)) {}
    virtual ~Articol() = default;
    int id() const { return id_; }
    const string& titlu() const { return titlu_; }
    virtual string tip() const = 0;
    virtual int zileMaxime() const = 0;            // cât poate fi ținut fără penalizare
    virtual double taxaPeZi() const = 0;           // penalizarea standard pe zi de
întârziere
    virtual string detalii() const = 0;
    friend ostream& operator<<(ostream &out, const Articol &a)
    {
        return out << '#' << a.id() << ' ' << a.tip() << " \"" << a.titlu() << "\" (" <<
a.detalii() << ')';
    }
};

class Carte : public Articol
{
    string autor;
public:
    Carte(int id, string t, string a) : Articol(id, std::move(t)), autor(std::move(a)) {}
    string tip() const override { return "carte"; }
    int zileMaxime() const override { return 14; }
    double taxaPeZi() const override { return 0.5; }
    string detalii() const override { return autor; }
};
```

```
class Revista : public Articol
{
    int numar;
public:
    Revista(int id, string t, int n) : Articol(id, std::move(t)), numar(n) {}
    string tip() const override { return "revista"; }
    int zileMaxime() const override { return 7; }
    double taxaPeZi() const override { return 0.25; }
    string detalii() const override { return "nr. " + to_string(numar); }
};

class Dvd : public Articol
{
    int minute;
public:
    Dvd(int id, string t, int m) : Articol(id, std::move(t)), minute(m) {}
    string tip() const override { return "dvd"; }
    int zileMaxime() const override { return 3; }
    double taxaPeZi() const override { return 2.0; }
    string detalii() const override { return to_string(minute) + " min"; }
};

// ------------------------------------------------------------ Factory cu registru
class FabricaArticole
{
    map<string, function<unique_ptr<Articol>(int, const string&, istream&)>> creatori;
public:
    FabricaArticole()
    {
        creatori["carte"] = [](int id, const string &t, istream &in) { string a; in >> a;
return make_unique<Carte>(id, t, a); };
        creatori["revista"] = [](int id, const string &t, istream &in) { int n; in >> n;
return make_unique<Revista>(id, t, n); };
        creatori["dvd"] = [](int id, const string &t, istream &in) { int m; in >> m; return
make_unique<Dvd>(id, t, m); };
    }
    unique_ptr<Articol> creeaza(const string &tip, int id, const string &titlu, istream
&in) const
    {
        auto it = creatori.find(tip);
        if (it == creatori.end()) throw EroareBiblioteca("tip necunoscut: " + tip);
        return it->second(id, titlu, in);
    }
};

// ------------------------------------------------------------ Strategy pentru penalizări
using StrategiePenalizare = function<double(const Articol&, int zileIntarziere)>;

double penalizareStandard(const Articol &a, int z) { return z * a.taxaPeZi(); }
double penalizareElev(const Articol &a, int z) { return min(z * a.taxaPeZi() / 2, 20.0); }

// ------------------------------------------------------------ biblioteca (subiectul
observat)
class Biblioteca
{
    map<int, unique_ptr<Articol>> articole;
    map<int, string> imprumuturi;                  // id articol -> membru
```

```
    vector<function<void(const string&)>> abonati; // Observer
    StrategiePenalizare penalizare = penalizareStandard;

    void anunta(const string &mesaj) const { for (const auto &f : abonati) f(mesaj); }
public:
    void aboneaza(function<void(const string&)> f) { abonati.push_back(std::move(f)); }
    void setPenalizare(StrategiePenalizare s) { penalizare = std::move(s); }

    void adauga(unique_ptr<Articol> a)
    {
        int id = a->id();
        if (articole.count(id)) throw EroareBiblioteca("id duplicat: " + to_string(id));
        articole[id] = std::move(a);
    }
    const Articol& articol(int id) const
    {
        auto it = articole.find(id);
        if (it == articole.end()) throw EroareBiblioteca("articol inexistent: " +
to_string(id));
        return *it->second;
    }
    void imprumuta(int id, const string &membru)
    {
        const Articol &a = articol(id);
        if (imprumuturi.count(id)) throw EroareBiblioteca("articolul " + to_string(id) + "
este deja imprumutat");
        imprumuturi[id] = membru;
        anunta(membru + " a imprumutat " + a.titlu());
    }
    string returneaza(int id, int zile, double &taxa)
    {
        const Articol &a = articol(id);
        auto it = imprumuturi.find(id);
        if (it == imprumuturi.end()) throw EroareBiblioteca("articolul " + to_string(id) +
" nu este imprumutat");
        string membru = it->second;
        imprumuturi.erase(it);
        taxa = penalizare(a, max(0, zile - a.zileMaxime()));
        anunta(membru + " a returnat " + a.titlu());
        return membru;
    }
    void anuleazaImprumut(int id) { imprumuturi.erase(id); }
    void refaImprumut(int id, const string &membru) { imprumuturi[id] = membru; }

    template <typename Pred>                        // șablon: orice condiție
    vector<const Articol*> cauta(Pred conditie) const
    {
        vector<const Articol*> r;
        for (const auto &[id, a] : articole)
            if (conditie(*a)) r.push_back(a.get());
        return r;
    }
    bool disponibil(int id) const { return !imprumuturi.count(id); }
};

// ---------------------------------------------------------- Command cu undo
class Comanda
```

```
{
public:
    virtual void executa() = 0;
    virtual void anuleaza() = 0;
    virtual ~Comanda() = default;
};

class Imprumut : public Comanda
{
    Biblioteca &b;
    int id;
    string membru;
public:
    Imprumut(Biblioteca &bib, int i, string m) : b(bib), id(i), membru(std::move(m)) {}
    void executa() override { b.imprumuta(id, membru); }
    void anuleaza() override { b.anuleazaImprumut(id); cout << "anulat: imprumutul
articolului " << id << '\n'; }
};

class Returnare : public Comanda
{
    Biblioteca &b;
    int id, zile;
    string membru;
public:
    Returnare(Biblioteca &bib, int i, int z) : b(bib), id(i), zile(z) {}
    void executa() override
    {
        double taxa = 0;
        membru = b.returneaza(id, zile, taxa);
        cout << "penalizare: " << fixed << setprecision(2) << taxa << " lei\n";
    }
    void anuleaza() override { b.refaImprumut(id, membru); cout << "anulat: returnarea
articolului " << id << '\n'; }
};

// ------------------------------------------------------------ aplicația
int main()
{
    Biblioteca bib;
    FabricaArticole fabrica;
    stack<unique_ptr<Comanda>> istoric;
    bib.aboneaza([](const string &m) { cout << "[notificare] " << m << '\n'; });
    int operatii = 0;
    bib.aboneaza([&operatii](const string &) { operatii++; });

    string linie;
    while (getline(cin, linie))
    {
        istringstream in(linie);
        string cmd;
        if (!(in >> cmd)) continue;
        try
        {
            if (cmd == "adauga")
            {
                string tip, titlu;
```

```
                int id;
                in >> tip >> id >> titlu;
                bib.adauga(fabrica.creeaza(tip, id, titlu, in));
                cout << "adaugat: " << bib.articol(id) << '\n';
            }
            else if (cmd == "imprumuta" || cmd == "returneaza")
            {
                int id;
                in >> id;
                unique_ptr<Comanda> c;
                if (cmd == "imprumuta") { string m; in >> m; c = make_unique<Imprumut>(bib,
id, m); }
                else { int z; in >> z; c = make_unique<Returnare>(bib, id, z); }
                c->executa();                       // dacă aruncă, comanda nu intră în
istoric
                istoric.push(std::move(c));
            }
            else if (cmd == "undo")
            {
                if (istoric.empty()) throw EroareBiblioteca("nimic de anulat");
                istoric.top()->anuleaza();
                istoric.pop();
            }
            else if (cmd == "regim")
            {
                string r;
                in >> r;
                if (r == "elev") bib.setPenalizare(penalizareElev);
                else bib.setPenalizare(penalizareStandard);
                cout << "regim: " << r << '\n';
            }
            else if (cmd == "disponibile")
            {
                for (const Articol *a : bib.cauta([&](const Articol &x) { return
bib.disponibil(x.id()); }))
                    cout << "  " << *a << '\n';
            }
            else throw EroareBiblioteca("comanda necunoscuta: " + cmd);
        }
        catch (const EroareBiblioteca &e) { cout << "eroare: " << e.what() << '\n'; }
    }
    cout << "operatii notificate: " << operatii << '\n';
    return 0;
}
```

Rularea pe intrarea de test (comenzile de mai jos) produce ieșirea din tabel:

Comanda	Ieșire	Ce s-a folosit
`adauga carte 1 Amintiri Creanga`	`adaugat: #1 carte "Amintiri" (Creanga)`	Factory, `operator<<`, polimorfism (`tip()`, `detalii()`)
`adauga caiet 4 Notite`	`eroare: tip necunoscut: caiet`	excepție aruncată de fabrică, prinsă în `main`
`imprumuta 1 Ana`	`[notificare] Ana a imprumutat Amintiri`	Command + Observer

Comanda	Ieșire	Ce s-a folosit
`imprumuta 1 Dan`	`eroare: articolul 1 este deja imprumutat`	invariantul bibliotecii, apărat cu excepție; comanda nu intră în istoric
`returneaza 1 25`	`penalizare: 5.50 lei`	Strategy standard: (25 − 14) · 0,5
`undo`	`anulat: returnarea articolului 1`	Command: anularea refăcută din datele reținute
`disponibile`	doar revista #2	șablonul `cauta` cu o lambda
`regim elev`, `returneaza 3 10`	`penalizare: 7.00 lei`	Strategy elev: (10 − 3) · 2 / 2
`returneaza 1 14`	`penalizare: 0.00 lei`	în termen
la final	`operatii notificate: 5`	al doilea abonat (contor)

DE CE AȘA?

De ce comanda intră în istoric doar după `executa()` reușit? Dacă `executa()` aruncă (articol deja împrumutat), operația nu s-a făcut, deci nu are ce anula. **De ce strategia este `function` și nu o ierarhie?** Penalizarea este o singură formulă, fără stare: o funcție ajunge (KISS). Articolele, în schimb, au mai multe comportamente legate între ele (durată, taxă, descriere), deci o ierarhie are sens. **De ce `Biblioteca` ține `unique_ptr<Articol>` ?** Polimorfism fără slicing și eliberare automată (regula lui 0).

DE ȘTIUT

Titlurile și numele sunt citite ca un singur cuvânt (fără spații), ca să păstrăm citirea simplă. O extensie bună este citirea titlurilor între ghilimele cu `std::quoted` .

4. Organizarea pe fișiere

Un proiect real nu stă într-un singur fișier. O împărțire firească:

```
biblioteca/
├── CMakeLists.txt
├── include/
│   ├── articol.h          // Articol, Carte, Revista, Dvd
│   ├── fabrica.h          // FabricaArticole
│   ├── biblioteca.h       // Biblioteca (cu șablonul cauta, care rămâne în header)
│   └── comenzi.h          // Comanda, Imprumut, Returnare
├── src/
│   ├── articol.cpp
│   ├── fabrica.cpp
│   ├── biblioteca.cpp
│   └── main.cpp
└── tests/
    └── test_biblioteca.cpp
```

Regula	De ce
în header: declarații de clase, funcții *inline*, șabloane	sunt necesare oriunde se folosește clasa
în `.cpp`: corpurile funcțiilor obișnuite	se compilează o singură dată; schimbarea lor nu recompilează tot
`#pragma once` (sau gărzi `#ifndef`) la începutul fiecărui header	evită includerea dublă
nu pune `using namespace std;` în header-e	ar „polua" toate fișierele care îl includ
include doar ce folosești	compilare mai rapidă, dependențe clare

Un `CMakeLists.txt` minim:

```
cmake_minimum_required(VERSION 3.16)
project(Biblioteca CXX)
set(CMAKE_CXX_STANDARD 17)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
add_compile_options(-Wall -Wextra)
add_library(biblio src/articol.cpp src/fabrica.cpp src/biblioteca.cpp)
target_include_directories(biblio PUBLIC include)
add_executable(app src/main.cpp)
target_link_libraries(app biblio)
add_executable(teste tests/test_biblioteca.cpp)
target_link_libraries(teste biblio)
```

Compilare: `cmake -S . -B build && cmake --build build`, apoi `./build/app < comenzi.txt` și `./build/teste`.

> **SFAT**
>
> Compilează mereu cu avertismente (`-Wall -Wextra`) și, în timpul dezvoltării, cu **sanitizatoare**: `-fsanitize=address,undefined` găsește accese în afara memoriei, folosiri după eliberare și depășiri de întregi chiar în momentul în care se produc. Programul din lecție rulează curat cu ele.

5. Teste automate

Testele verifică automat că o modificare nu a stricat nimic. Fără nicio bibliotecă, cu `assert`:

```cpp
#include <cassert>
// #include "biblioteca.h", "fabrica.h"

void test_imprumut_dublu_arunca()
{
    Biblioteca b;
    b.adauga(make_unique<Carte>(1, "T", "A"));
    b.imprumuta(1, "Ana");
    bool aruncat = false;
    try { b.imprumuta(1, "Dan"); } catch (const EroareBiblioteca &) { aruncat = true; }
    assert(aruncat);
}

void test_penalizare_elev_plafonata()
{
    Dvd d(1, "Film", 90);
    assert(penalizareElev(d, 100) == 20.0);   // 100 * 2 / 2 = 100, plafonat la 20
}
```

Cu **GoogleTest**, același test devine `EXPECT_THROW(b.imprumuta(1, "Dan"), EroareBiblioteca);`, iar biblioteca afișează frumos ce teste au trecut și ce teste au picat. Injectarea dependențelor (oop-05) face testarea posibilă: în teste, abonatul poate fi o lambda care doar reține mesajele, ca să fie verificate.

6. Etapele proiectului (pentru lucrul tău)

Etapa	Ce adaugi	Concepte exersate
1	citește și rulează programul; scrie 5 teste cu `assert`	înțelegerea codului
2	un tip nou: `CarteAudio` (durată, 10 zile, 1 leu/zi)	moștenire, Factory (o singură linie de înregistrare)
3	comanda `redo`	Command, a doua stivă
4	**rezervări**: un membru rezervă un articol împrumutat; la returnare este anunțat	Observer cu filtru pe articol
5	comanda `cauta autor X` și `cauta tip dvd`	șablonul `cauta`, lambda
6	**persistență**: `salveaza fisier` / `incarca fisier`	fluxuri, `operator<<` / `>>`, excepții la fișiere lipsă
7	împarte proiectul pe fișiere și compilează-l cu CMake	organizare, header-e
8	istoricul membrilor și „cel mai activ cititor”	`map`, algoritmi STL

> **IMPORTANT**
>
> Lucrează **incremental**: după fiecare etapă, programul trebuie să compileze, să treacă toate testele vechi și pe cele noi. Salvează fiecare etapă într-un **commit git** cu un mesaj clar („adaug rezervări"). Așa poți reveni oricând la o versiune care mergea.

7. Cum se evaluează un proiect

Criteriul	Ce înseamnă „foarte bine"
corectitudine	toate comenzile funcționează; erorile sunt tratate fără căderi
proiectare	responsabilități clare (SOLID); tip nou adăugat fără modificarea codului existent
siguranța memoriei	niciun `new` / `delete` manual; rulează curat cu sanitizatoarele
lizibilitate	nume clare, funcții scurte, `const` corect, comentarii doar unde e nevoie de „de ce"
teste	fiecare funcționalitate are cel puțin un test, inclusiv cazurile de eroare
organizare	header-e și surse separate, CMake, istoric git curat

8. Greșeli frecvente în proiecte

Greșeala	Problema	Corect
totul în `main`	imposibil de testat și de extins	clase cu responsabilități separate
`if (tip == "carte")` împrăștiat prin cod	fiecare tip nou cere modificări peste tot	polimorfism + Factory
erori afișate în adâncul claselor	logica amestecată cu interfața	aruncă excepții, afișează în `main`
`using namespace std;` în header	conflicte de nume în tot proiectul	doar în fișierele `.cpp`
copiere accidentală a unui obiect mare	lent sau incorect	`const &`, `= delete` la copiere unde nu are sens
commit-uri uriașe, fără mesaj	imposibil de urmărit ce s-a schimbat	commit-uri mici, mesaje clare
fără teste „ca să meargă mai repede"	fiecare schimbare poate strica ceva neobservat	teste de la început

9. Exersează pe hârtie

1. Ce penalizare se calculează pentru o revistă returnată după 12 zile în regim standard? Dar în regim elev?

2. Ce afișează programul dacă după `imprumuta 2 Ion` urmează `undo` și apoi `undo` din nou, iar istoricul conținea doar acest împrumut?

3. Ce trebuie modificat pentru tipul nou `CarteAudio`? Enumeră fișierele/clasele atinse.

4. De ce `Biblioteca` nu afișează nimic direct, ci trimite mesaje abonaților?

5. Cum ai implementa `redo`? Ce se întâmplă cu stiva de refaceri după o comandă nouă?

6. Ce excepție apare la `returneaza 2 5` dacă revista 2 nu este împrumutată? Unde este prinsă?

7. De ce `cauta` este un șablon și nu primește un `function<bool(const Articol&)>`? Care ar fi avantajul variantei cu `function`?

8. Propune o clasă pentru rezervări. Cum o legi de mecanismul de notificări?

9. Ce s-ar întâmpla dacă `Articol` nu ar avea destructor virtual?

10. De ce comanda `Returnare` reține membrul care a returnat?

10. Mini-grile

1. În proiect, adăugarea unui tip nou de articol cere modificarea: a) clasei `Biblioteca` b) doar a unei clase noi și a unei linii în fabrică c) funcției `main` în mai multe locuri d) tuturor claselor

2. Undo este implementat cu șablonul: a) Strategy b) Observer c) Command d) Singleton

3. Regimul de penalizare elev/standard este un exemplu de: a) Strategy b) Composite c) Adapter d) Builder

4. Șabloanele de clase și funcții se pun: a) în fișierele `.cpp` b) în header-e c) doar în `main.cpp` d) în CMakeLists.txt

11. Mai departe (pentru cine vrea mai mult)

- **Interfață grafică** (Qt) sau **API web** peste aceeași clasă `Biblioteca`, fără a o modifica.
- **Bază de date** (SQLite) în loc de fișiere, printr-o interfață `Depozit` injectată (DIP).
- **Concurență:** mai mulți utilizatori simultan, cu `std::mutex` și `lock_guard` (RAII pentru lacăte).
- **Integrare continuă** (GitHub Actions): testele rulează automat la fiecare commit.
- **Analiză statică:** `clang-tidy` și `cppcheck` pentru probleme detectate înainte de rulare.

Rezumat

- Proiectul combină: clase cu `const` și liste de inițializare, regula lui 0, `operator<<`, ierarhie polimorfică, SOLID, șabloane, excepții, `unique_ptr`, lambda, Factory, Strategy, Observer și Command.
- Fiecare șablon are un motiv concret: tipuri noi fără modificări, penalizări schimbabile, notificări decuplate, undo.

- Proiectele reale se organizează pe header-e și surse, se compilează cu CMake, se testează automat și se versionează cu git.
- Lucrează incremental: fiecare etapă compilează și trece testele.

Citește și exersează în plus

- CMake — tutorial oficial — construirea proiectelor C++ cu mai multe fișiere (engleză) https://cmake.org/cmake/help/latest/guide/tutorial/index.html
- GoogleTest — Primer — teste unitare în C++ (engleză) https://google.github.io/googletest/primer.html
- Pro Git (carte gratuită) — controlul versiunilor pentru orice proiect (engleză) https://git-scm.com/book/en/v2

Răspunsuri

Exersează pe hârtie

1. Revista: 7 zile maxime, 0,25 lei/zi. Întârziere 5 zile: standard 1,25 lei; elev 0,625 lei (programul afișează `0.62` : 0,625 este exact la jumătate, iar afișarea cu două zecimale rotunjește spre cifra pară).

2. `[notificare] Ion a imprumutat Stiinta`, apoi `anulat: imprumutul articolului 2`, apoi `eroare: nimic de anulat`.

3. O clasă nouă `CarteAudio : public Articol` și o linie în constructorul `FabricaArticole`. Nu se modifică `Biblioteca`, comenzile sau `main`.

4. Ca să nu depindă de felul în care sunt trimise notificările (consolă, e-mail, SMS, test). Principiul inversării dependențelor și Observer: abonații se pot schimba fără a modifica biblioteca.

5. O a doua stivă `refaceri` : `undo` mută comanda din `istoric` în `refaceri` după `anuleaza()` ; `redo` o execută din nou și o mută înapoi. O comandă nouă golește `refaceri`.

6. `EroareBiblioteca("articolul 2 nu este imprumutat")`, aruncată în `Biblioteca::returneaza`, trece prin `Returnare::executa` și este prinsă în blocul `catch` din bucla principală din `main`; se afișează mesajul și programul continuă.

7. Șablonul acceptă orice obiect apelabil fără cost la rulare (fără alocare și apel indirect). Varianta cu `function` ar permite ca `cauta` să fie definită într-un `.cpp` (nu în header) și să fie virtuală, deci ar ascunde mai bine implementarea.

8. De exemplu, `map<int, queue<string>> rezervari` în `Biblioteca` (sau o clasă `Rezervari` separată, ca abonat). La returnare, biblioteca verifică dacă articolul are rezervări și anunță abonații cu un mesaj „articolul X este disponibil pentru Y".

9. Ștergerea unui `Carte` printr-un `unique_ptr<Articol>` ar fi comportament nedefinit; destructorii membrilor derivatei (de exemplu, `string autor`) nu ar rula corect.

10. Pentru `anuleaza()` : ca să refacă împrumutul exact pe numele aceluiași membru.

Mini-grile

1. b). **2.** c) Command. **3.** a) Strategy. **4.** b) în header-e.