FIȘĂ DE LUCRU · CLASA A XI-A · FIȘA 12 · ACOPERĂ LECȚIA XI.21

Programare orientată pe obiecte (introducere)

Durată: 110 min Nivel: intensiv

CE VEI ȘTI DUPĂ LECȚIE

- Prezici ordinea constructorilor și destructorilor, inclusiv la copii, moștenire și membri-obiect.
- Găsești erorile tipice: parametru care ascunde atributul, metodă fără const, set fără verificare, virtual ratat.
- Scrii clase mici, cu date private, constructori, operatori și moștenire, pe probleme reale.

Ce trebuie să știi (o pagină)

Noțiune	Cod	De reținut
clasă, `private` / `public`	`class A { int x; public: int get() const; };`	în `class`, implicit totul e `private`; `;` după `}`
constructor (listă de inițializare, valoare implicită)	`A(int v = 0) : x(v) {}`	fără tip; `A a;` și `A b(5);` îl folosesc pe același
destructor	`~A() { … }`	la ieșirea din bloc, în ordinea **inversă** creării
copie	`void f(A x)` sau `A b = a;`	se creează o **copie** (fără constructorul tău!), distrusă și ea
metodă `const`	`int get() const;`	singura apelabilă pe `const A &`
`this`	`A& inc() { x++; return *this; }`	referință ⇒ apeluri în lanț
`static`	`static int nr;` + `int A::nr = 0;`	un exemplar, comun tuturor obiectelor
operatori	`A operator+(const A &o) const`	`a + b` înseamnă `a.operator+(b)`
moștenire	`class B : public A { … };`	se construiește întâi `A`, apoi membrii lui `B`, apoi `B`
polimorfism	`virtual void f() const;` + `void f() const override;`	prin `A*` / `A&` se apelează metoda obiectului **real**

ATENȚIE — GREȘELI FRECVENTE

Un obiect derivat **copiat** într-o variabilă de tipul bazei (`Animal a = pisica;`) pierde partea derivată: polimorfismul merge doar prin pointer sau referință.

Nivelul 1 · Prezice

★ citește și prezice

P1. Copia invizibilă. Ce afișează programul? De ce apare `-1` de două ori, deși `+1` apare o singură dată?

```
class T { int id; public: T(int i) : id(i) { cout << "+" << id << ' '; } ~T() { cout << "-"
<< id << ' '; } };
void f(T x) { cout << "f "; }
void g(T &x) { cout << "g "; }
int main() { T a(1); f(a); g(a); T b(2); }
```

P2. Bilete numerotate. Ce afișează?

```
class Bilet
{
    static int urmator;
    int nr;
public:
    Bilet() { nr = urmator; urmator += 10; }
    int getNr() const { return nr; }
    static int cate() { return (urmator - 100) / 10; }
};
int Bilet::urmator = 100;
int main()
{
    Bilet a, b;
    for (int i = 0; i < 3; i++) { Bilet c; cout << c.getNr() << ' '; }
    Bilet d; cout << a.getNr() << ' ' << d.getNr() << ' ' << Bilet::cate();
}
```

P3. Care constructor? Clasa `Punct` are trei constructori: `Punct()` afișează `A` (punctul 0, 0), `Punct(int v)` afișează `B` (punctul v, v), `Punct(int a, int b)` afișează `C`. Metoda `suma()` întoarce `x + y`. Ce afișează?

```
Punct p, q(3), r(2, 5);
Punct v[3] = { Punct(1, 1), Punct(4) };
cout << p.suma() + q.suma() + r.suma() + v[0].suma() + v[1].suma() + v[2].suma();
```

P4. Lanțul. Ce afișează? De ce `b` nu ajunge la 6?

```
class Contor
{
    int v;
public:
    Contor() : v(0) {}
    Contor& inc() { v++; return *this; }
    Contor inc2() { v += 2; return *this; }
    int get() const { return v; }
};
// în main:
Contor a, b;
a.inc().inc().inc();
b.inc2().inc2().inc2();
cout << a.get() << ' ' << b.get();
```

P5. Pointer, referință, copie. `Animal` are `void nume() const` (afișează `animal`, **fără** `virtual`) și `virtual void sunet() const` (afișează `...`). `Pisica : public Animal` redefinește ambele (`pisica`, `miau`). Ce afișează?

```
Pisica p;
Animal *a = &p;  Animal &r = p;  Animal copie = p;
a->nume(); a->sunet(); r.sunet(); p.nume(); copie.sunet();
```

P6. Cine se naște primul? Ce afișează?

```
class Motor   { public: Motor()   { cout << "M "; } ~Motor()   { cout << "~M "; } };
class Vehicul { public: Vehicul() { cout << "V "; } ~Vehicul() { cout << "~V "; } };
class Masina : public Vehicul
{
    Motor m;
public:
    Masina()  { cout << "Ma "; }
    ~Masina() { cout << "~Ma "; }
};
int main() { Masina x; cout << "| "; }
```

P7. Ceasul care se normalizează. Ce afișează?

```
class Timp
{
    int h, m;
public:
    Timp(int ore = 0, int min = 0) { int t = ore * 60 + min; h = t / 60 % 24; m = t % 60; }
    Timp operator+(const Timp &o) const { return Timp(h + o.h, m + o.m); }
    bool operator<(const Timp &o) const { return h * 60 + m < o.h * 60 + o.m; }
    void afis() const { cout << h << ':' << m << ' '; }
};
// în main:
Timp a(22, 50), b(1, 25), c(0, 130);
(a + b).afis(); c.afis(); (b + c + c).afis();
cout << (a + b < a) << ' ' << (Timp(25) < Timp(0, 61));
```

P8. Compilează? Clasa este `class Cont { int sold; public: Cont(int s) : sold(s) {} bool maiBogat(const Cont &alt) const { return sold > alt.sold; } };`, iar în `main` avem `Cont a(5), b(9);`. Pentru fiecare rând: compilează? Dacă da, ce afișează?

rând din `main`	compilează?	afișează
`a.sold = 10;`		
`cout << a.maiBogat(b);`		
`Cont c;`		
`const Cont k(3); cout << k.maiBogat(a);`		
`Cont d = 7; cout << d.maiBogat(a);`		

Nivelul 2 · Găsește și repară

★★ găsește și repară

P9. `Cont c(50); cout << c.get();` nu afișează 50, ci o valoare la întâmplare. Unde e greșeala? Repar-o în **două** feluri.

```
class Cont { int sold; public: Cont(int sold) { sold = sold; } int get() const { return
sold; } };
```

P10. Programul nu compilează: „passing 'const Elev' as 'this' argument discards qualifiers". Ce înseamnă mesajul? Repară cu un singur cuvânt.

```
class Elev { string nume; public: Elev(string n) : nume(n) {} string getNume() { return
nume; } };
void afis(const Elev &e) { cout << e.getNume(); }
```

P11. Completează golurile, apoi spune ce afișează ultimul rând.

```
class V2
{
    int x, y;
public:
    V2(int a = 0, int b = 0) : ______ {}
    V2 operator+(const V2 &o) ______ { return V2(______, ______); }
    bool operator==(const V2 &o) const { return ______; }
    int getX() const { return x; }
    int getY() const { return y; }
};
// în main:
V2 a(1, 2), b(3, -2), c; V2 s = a + b + c;
cout << s.getX() << ' ' << s.getY() << ' ' << (a + b == V2(4)) << ' ' << (a + b == V2(4,
0));
```

P12. Promisiunea încălcată. Autorul clasei a promis „mereu `a ≤ b`". Ce afișează codul? Rescrie `setA` ca promisiunea să nu mai poată fi încălcată (alege tu: refuzi valoarea sau o accepți și refaci intervalul) și spune ce afișează după reparație.

```
class Interval
{
    int a, b;                                   // a <= b mereu
public:
    Interval(int x, int y) { a = min(x, y); b = max(x, y); }
    void setA(int x) { a = x; }
    int lungime() const { return b - a; }
    bool contine(int x) const { return a <= x && x <= b; }
};
// în main:
Interval I(8, 2); I.setA(10); cout << I.lungime() << ' ' << I.contine(9);
```

P13. „Câinele nu latră". Ce afișează `Caine c; Animal *a = &c; a->sunet(); c.sunet();` ? De ce prin pointer nu se aude „ham"? Ce cuvânt ar fi transformat greșeala într-o eroare de compilare?

```
class Animal { public: virtual void sunet() const { cout << "... "; } virtual ~Animal() {}
};
class Caine : public Animal { public: void sunet() { cout << "ham "; } };
```

P14. Copia superficială. Ce afișează? Ce s-ar întâmpla dacă ai adăuga destructorul `~Vec() { delete[] v; }`? Ce trebuie scris ca `b` să fie o copie independentă?

```
class Vec { public: int *v; int n; Vec(int k) : n(k) { v = new int[k](); } };
// în main:
Vec x(3); Vec y = x; y.v[0] = 7; y.n = 5;
cout << x.v[0] << ' ' << x.n;
```

Nivelul 3 · Scrie cod

★★ scrie cod

Datele clasei sunt **private**; `main` doar citește, apelează metode și afișează.

P15. Ceasul de pe telefon. Clasa `Ceas` (ore, minute), cu `adauga(int minute)` și `afiseaza()` în formatul `hh:mm`. Se citesc ora, minutul și `k`; afișează ora după `k` minute. Teste: `23 50 20 → 00:10`, `10 0 1440 → 10:00`, `9 5 125 → 11:10`, `0 0 0 → 00:00`.

P16. Stația meteo. Clasa `Termometru` primește citiri prin `bool adauga(double t)` și refuză valorile din afara intervalului [−90, 60]. Are `minim()`, `maxim()`, `media()`, `cate()`. Se citesc `n` valori; afișează minimul, maximul, media (o zecimală) și câte citiri au fost refuzate, sau `nicio citire valida`.

Teste: `5` / `12 -3.5 100 7 20` → `-3.5 20.0 8.9 1`; `2` / `-95 61` → `nicio citire valida`; `1` / `0` → `0.0 0.0 0.0 0`.

P17. Mulțimi. Clasa `Multime` (numere din 0..100) cu `adauga`, `contine` și operatorii `+` (reuniune) și `*` (intersecție). Se citesc două mulțimi (`n`, apoi elementele; pot fi repetate). Afișează reuniunea, apoi intersecția, crescător, sau `vida`. Teste: `4` / `1 5 7 9`, `3` / `5 9 20` → `1 5 7 9 20` / `5 9`; `2` / `3 3`, `2` / `4 100` → `3 4 100` / `vida`.

P18. Statul de plată. Clasa abstractă `Angajat` (nume, `virtual int salariu() const = 0`), cu `Programator` (ore · tarif) și `Manager` (fix + 500 pentru fiecare subordonat). Fiecare rând: `P nume ore tarif` sau `M nume fix subordonati`. Afișează numele și salariul fiecăruia, apoi `total`.

Teste: `3` / `P Ana 160 40` / `M Dan 6000 4` / `P Ion 100 55` → `Ana 6400` / `Dan 8000` / `Ion 5500` / `total 19900`; `1` / `M Eva 7000 0` → `Eva 7000` / `total 7000`.

P19. Numere uriașe. Clasa `NumarMare` ține cifrele într-un `vector<int>` și are constructor din `string`, `operator+` și `afis()`. Se citesc două numere naturale cu până la 1000 de cifre; afișează suma. Teste: `999 1 → 1000`; `0 0 → 0`; `12345678901234567890123456789` + `98765432109876543210987654321` → `11111111101111111111011111111100`.

P20. X și 0. Clasa `Tabla` cu `bool muta(int r, int c)` (alternează X și 0, refuză celulele ocupate sau din afara tablei) și `char castigator()`. Se citesc `n` mutări. Afișează `X castiga la mutarea k` (sau `0 …`), `mutare invalida k`, `remiza` (9 mutări, fără câștigător) sau `joc neterminat`.

Teste (mutările, ca rând coloană): `1 1`, `2 2`, `1 2`, `3 3`, `1 3` → `X castiga la mutarea 5`; `2 2`, `2 2`, … → `mutare invalida 2`; `1 1`, `3 1`, `2 2`, `3 2`, `1 3`, `3 3` → `0 castiga la mutarea 6`; `1 1`, `2 2`, `1 2`, `1 3`, `3 1`, `2 1`, `2 3`, `3 2`, `3 3` → `remiza`.

Nivelul 4 · Pune mintea la contribuție

★★★ pune mintea la contribuție

P21. Proiectează, nu scrie. O parcare are etaje, locuri (normale sau pentru electrice) și mașini care intră și ies; la ieșire se plătește după durată, iar electricele au 50% reducere. Ce clase propui, ce atribute private și ce metode publice? Unde pui moștenirea și unde compunerea („este un" / „are un")? Ce trebuie să **refuze** fiecare metodă?

P22. Problema inversă. Cu clasa `T` din P1 (fără `f` și `g`), ce ieșiri poți obține doar cu obiecte declarate în `main` și blocuri `{ }`? a) `+1 -1 +2 -2` b) `+1 +2 -2 -1` c) `+1 +2 -1 -2`. Scrie `main` pentru cele posibile. Ce îți trebuie pentru cea imposibilă?

P23. Ce se întâmplă dacă... clasa `T` numără obiectele „în viață": `static int vii;`, `vii++` în constructor, `vii--` în destructor? Ce afișează `{ T a(1); f(a); f(a); T b(2); cout << T::vii << ' '; } cout << T::vii;` (cu `f` din P1)? Ce lipsește din clasă ca numărătoarea să fie corectă?

P24. Stiva cu minim. Scrie clasa `StivaMin` cu `push`, `pop`, `top` și `minim`, **toate în O(1)**, chiar și după `pop`. Comenzile sunt `push x`, `pop`, `min` (afișează minimul sau `goala`). Teste: `push 5`, `push 3`, `min`, `push 7`, `min`, `pop`, `pop`, `min` → `3` / `3` / `5`; `min`, `push 4`, `min` → `goala` / `4`. De ce nu ajunge o singură variabilă `mn`?

P25. Mini-proiect: Fibonacci cu matrice. Clasa `M2` (matrice 2 × 2 de `long long`) are `operator*`. Cu ridicare la putere rapidă, `[[1,1],[1,0]]ⁿ` are în colțul dreapta-sus pe F(n). Afișează F(n) pentru n ≤ 90. Teste: `10 → 55`, `1 → 1`, `50 → 12586269025`, `90 → 2880067194370816120`. Câte înmulțiri de matrice faci pentru n = 90?

Probleme de lucru în clasă (pbinfo)

Le rezolvăm împreună în ședință, în ordinea de mai jos.

- [ ] `#3529` **oop-1** — ușoară
  https://www.pbinfo.ro/probleme/3529/oop-1
- [ ] `#3530` **oop-2** — ușoară
  https://www.pbinfo.ro/probleme/3530/oop-2

Temă pentru acasă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] `#3533` **oop-3** — medie
  https://www.pbinfo.ro/probleme/3533/oop-3
- [ ] `#3539` **oop-numere-complexe** — medie
  https://www.pbinfo.ro/probleme/3539/oop-numere-complexe