

C++ OOP CHEAT SHEET - VERSIUNEA A4

INCLUDES + USING

cpp

```
#include<iostream>
#include<vector>
#include<string>
#include<algorithm>
#include<limits>
using namespace std;
```

TEMPLATE BAZA

cpp

```
class Base {
protected: string nume; int valoare;
public:
    Base() = default; Base(const Base&) = default;
    Base& operator=(const Base&) = default; virtual ~Base() = default;
    friend istream& operator>>(istream& is, Base& b) { return b.citire(is); }
    friend ostream& operator<<(ostream& os, Base& b) { return b.afisare(os); }
    virtual istream& citire(istream&) = 0;
    virtual ostream& afisare(ostream&) const = 0;
    virtual Base* clone() const = 0;
    string getNume() const { return nume; }
};
```

TEMPLATE DERIVATA

cpp

```
class Derived : public Base {
private: string membru_specific;
public:
    Derived() = default; ~Derived() = default;
    Base* clone() const override { return new Derived(*this); }
    virtual istream& citire(istream& is) override {
        Base::citire(is); cout<<"membru: "; is>>membru_specific; return is; }
    virtual ostream& afisare(ostream& os) const override {
        Base::afisare(os); os<<"membru: "<<membru_specific<<endl; return os; }
    string getMembru() const { return membru_specific; }
};
```

DIAMANT (cu VIRTUAL!)

cpp

```
class Parent1 : virtual public Base { protected: string m1; };
class Parent2 : virtual public Base { protected: int m2; };
class Diamond : public Parent1, public Parent2 {
public:
    Diamond() { m1="default"; m2=1; }
    Base* clone() const override { return new Diamond(*this); }
    virtual istream& citire(istream& is) override {
        Base::citire(is); cout<<"m2: "; is>>m2; return is; }
    virtual ostream& afisare(ostream& os) const override {
        Parent1::afisare(os); os<<"m2: "<<m2<<endl; return os; }
};
```

FACTORY

cpp

```
Base* createObject(const string& type) {
    if(type=="tip1") return new Derived1();
    if(type=="tip2") return new Derived2();
    if(type=="diamond") return new Diamond();
    return nullptr;
}
```

SINGLETON

cpp

```
class Manager {
private:
    Manager() {}; Manager(const Manager&) = delete;
    Manager& operator=(const Manager&) = delete;
    inline static Manager* instance = nullptr;
    vector<Base*> items;
public:
    ~Manager() { for(auto* i : items) delete i; items.clear(); }
    static Manager* getInstance() {
        if(!instance) instance = new Manager(); return instance; }
    void deleteInstance() { if(instance) { delete instance; instance=nullptr; } }

    void adauga() {
        string tip; cout<<"Tip: "; cin>>tip;
        Base* obj = createObject(tip);
        if(obj) { cin>>*obj; items.push_back(obj); }
    }
    void afiseaza() { for(auto* i : items) cout<<*i<<endl; }
    void raport() {
        int c1=0, c2=0, d=0;
        for(auto* i : items) {
            if(dynamic_cast<Diamond*>(i)) d++;
            else if(dynamic_cast<Derived1*>(i)) c1++;
            else if(dynamic_cast<Derived2*>(i)) c2++;
        }
        cout<<"Total: "<<items.size()<<" D1:"<<c1<<" D2:"<<c2<<" Diamond:"<<d<<endl;
    }
};

Manager* Manager::instance = nullptr;
```



COPY CONSTRUCTOR

cpp

```
ClassName(const ClassName& other) : base(other.base) {
    items.clear();
    for(const auto* item : other.items) items.push_back(item->clone());
}
ClassName& operator=(const ClassName& other) {
    if(this != &other) {
        base = other.base;
        for(auto* i : items) delete i; items.clear();
        for(const auto* i : other.items) items.push_back(i->clone());
    } return *this;
}
```

⚠ VALIDARI

cpp

```
void valideaza() {
    int val; while(true) { try {
        cout<<"Val: "; cin>>val;
        if(cin.fail()) { cin.clear(); cin.ignore(1000, '\n'); throw string("Invalid!");
        if(val<10) throw string("Min 10!"); break;
    } catch(string s) { cout<<s<<endl; } }
}
```

🎮 MENU

cpp

```
void meniu() {
    int opt; cout<<"1.Add 2.Show 3.Report 4.Exit\nOpt: ";
    try {
        cin>>opt; cin.get();
        if(opt<1||opt>4) throw string("Invalid!");
        switch(opt) {
            case 1: Manager::getInstance()->adauga(); break;
            case 2: Manager::getInstance()->afiseaza(); break;
            case 3: Manager::getInstance()->raport(); break;
            case 4: return;
        }
    } catch(string s) { cout<<s<<endl; }
    meniu();
}


int main() { meniu(); Manager::getInstance()->deleteInstance(); return 0; }
```


SORTARI + CAUTARI


cpp

```
// Sort cu lambda
sort(items.begin(), items.end(), [](Base* a, Base* b) {
    return a->getNume() < b->getNume(); });

// Cautare cu conditii
bool found = false;
for(auto* i : items) {
    if(Derived1* d = dynamic_cast<Derived1*>(i)) {
        if(d->getVal() > 10) { cout<<*d<<endl; found = true; }
    }
}
if(!found) cout<<"Nu exista!"<<endl;
```

 **TIMP (90min): 0-5(analiza) 5-15(baza+1deriv) 15-35(toate) 35-55(singleton) 55-75(meniu) 75-85(valid) 85-90(test)**

 **CHECKLIST:** ☐virtual~Base ☐clone() ☐<<>> ☐copy ☐singleton
☐deleteInstance() ☐meniu ☐factory ☐dynamic_cast(ordinea!) ☐pentru
diamant:virtual public ☐getline+cin.get()

 **PRIORITATI: OBLIGATORIU(5-6): ierarhia+IO+meniu / 7-8:
dynamic_cast+rapoarte / 9-10: validari+functii complexe**

REGULA: Functionalitate > Eleganta!