Tutoriat 4
Upcasting

1. Upcasting

a. Cum se face?

Orice referință/pointer către o clasă **derivată** poate fi convertită într-o referință/pointer către clasa de **bază**.

Adică?

Într-o referință/pointer de tipul clasei de bază rețin adresa către un obiect de tipul clasei derivate:

Clasa_de_bază pointer = new Clasa_derivată();* -> **upcasting dinamic** (cel mai recomandat și frecvent folosit)

```
class Shape {};
class Square : public Shape {};
class Circle : public Shape {};

int main() {
   Shape* s1 = new Square();
   Shape* s2 = new Circle();
}
```

b. Condiția pentru ca upcasting-ul să fie posibil:

Moștenirea să fie de tip **public**.

PS: Apare destul de frecvent un exercițiu cu o astfel de *capcană* la examen.

```
class B{};
class D: private B{};
class C: protected B{};
class A: public B{};
int main(){
   B *p1 = new D(); // error: 'B' is an inaccessible base of 'D'
   B *p2 = new C(); // error: 'B' is an inaccessible base of 'C'
   B *p3 = new A(); //ok
}
```

c. Unde se folosește?
Upcasting-ul este folositor când avem de lucrat cu **vectori de obiecte** care pot fi de mai multe tipuri.

Obs: În exemplu se folosește biblioteca *vector* din STL. Mai multe despre STL puteți afla aici.

```
#include<iostream>
#include<vector>
using namespace std;

class Shape {
   friend ostream& operator <<(ostream& os, Shape& ob);
   friend istream& operator >> (istream& os, Shape& ob);
};
istream& operator >> (istream& os, Shape& ob)
{
   cout<<"citeste datele obiectului"<<endl;
   return os;
}
ostream& operator <<(ostream& os, Shape& ob)
{
   cout<<"afiseaza datele obiectului"<<endl;
   return os;
}
class Square : public Shape {

};
class Circle : public Shape {};

int main() {
   int n;
   cout<<"Dati n= ";
   cin >> n;
   vector<Shape*> formeGeometrice; // vectorul din STL
   for (int i = 0; i < n; ++i) {
      char optiune;
      cout << "Patrat sau cerc (P / C): ";
      cin >> optiune;

      if(optiune == 'P'){
            Square* s = new Square();
            cin >> *s;
```

```
            formeGeometrice.push_back(s);
          }
        else {
            Circle* c = new Circle();
            cin >> *c;
            formeGeometrice.push_back(c);
          }
      }
   cout<<"S-a terminat citirea"<<endl;

   for (int i = 0; i < n; ++i) {
     cout<<*formeGeometrice[i]<<endl;
   }
   return 0;
}
```

d. Suprascriere și upcasting

Printr-o referință/pointer de tipul clasei de bază(nu contează daca conține un obiect de tipul clasei derivate --- upcasting sau de tipul clasei de bază) se accesează doar metoda suprascrisă din clasa de bază.

```
class B{
public:
   void print(){cout<<"Print from B::"<<endl;}
};
class D: public B{
public:
   void print(){cout<<"Print from D::"<<endl;}
};

int main(){

   B *upcast = new D();
   B *b = new B();
   D *d = new D();

   upcast->print();// Print from B::
   (*upcast).print(); // Print from B::
   upcast-> D:: print(); //eroare -> 'D' is not a base of 'B'
   b->print(); // Print from B::
   d->print(); // Print from D::
}
```

e. Upcasting static(nerecomandat în practică, dar există)

Clasa_de_bază ob = (Clasa_de_bază) ob_clasa_derivată;

```
class Shape {};
class Square : public Shape {};
class Circle : public Shape {};
int main() {
    Square s;
    Circle c;
    Shape s1 = (Shape) s;
    Shape s2 = (Shape) c;
    return 0;
}
```