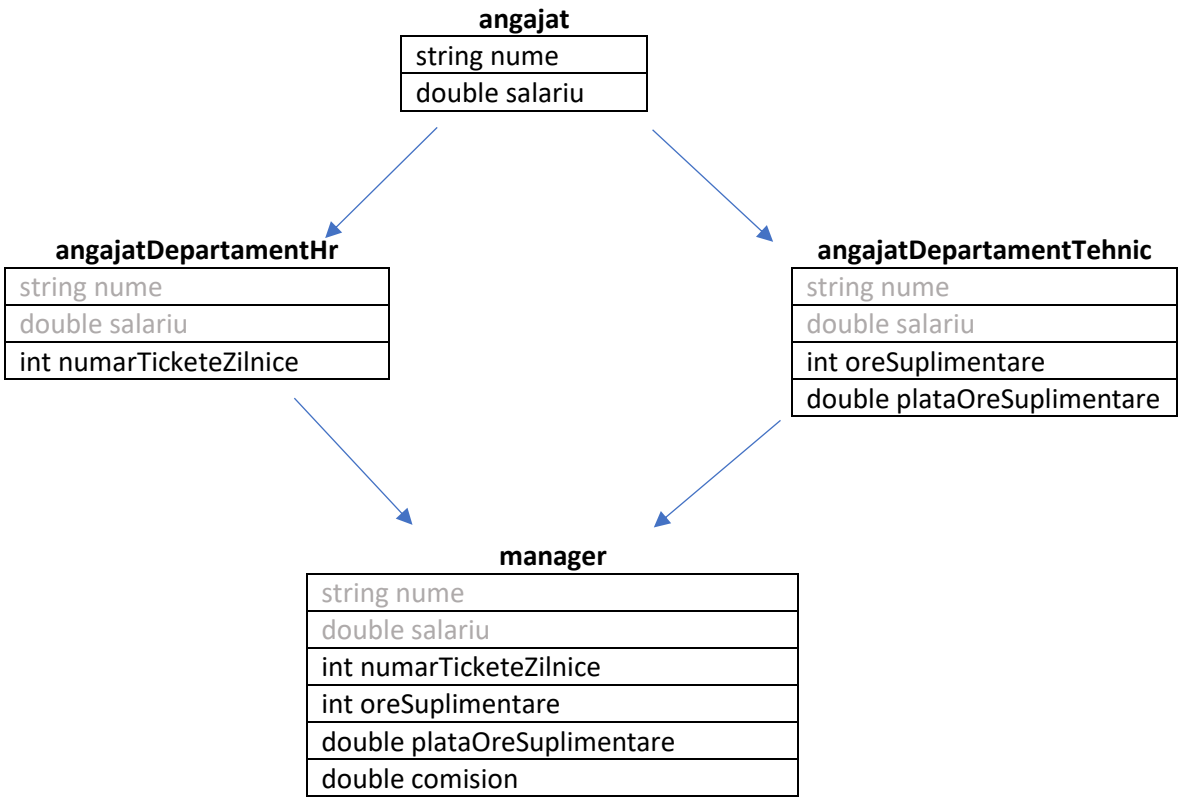
Problema Diamantului

Așa cum am văzut, în C++, conceptul de moștenire este puternic, dar uneori poate conduce la probleme dacă nu este folosit corespunzător.

Considerăm ierahia următoare:



Am vrea o clasa manager care să arate ca mai sus, însă, în mod evident o să apară probleme de ambiguitate, deoarece, atât clasa **angajatDepartamentHr**, cât și **angajatDepartamentTehnic** au datele membre **nume** și **salariu.** Similar si pentru metodele comune din clasa **angajat** (se vor dubla astfel si nu vom sti pe care sa o folosim).

Mai jos avem o ilustrare a **problemei diamantului nerezolvată (ilustrata DOAR pentru a înțelege greselile).**

```cpp
#include <iostream>
using namespace std;
class angajat{
protected:
    string nume;
    double salariu;
public:
    angajat(string nume="", double salariu=0){
        this->nume=nume;
        this->salariu=salariu;
    }
    virtual void afiseaza(ostream& out){
        out<<"Nume="<<this->nume<<"\n";
        out<<"Salariu="<<this->salariu<<"\n";
    }
};
class angajatDepartamentHr:public angajat{
protected:
    int numarTicketeZilnice;
public:
    angajatDepartamentHr(string nume="", double salariu=0, int numarTicketeZilnice=0):angajat(nume,salariu){
        this->numarTicketeZilnice=numarTicketeZilnice;
    }
    void afiseaza(ostream& out){
        angajat::afiseaza(out);
        out<<"Numar tickete:"<<this->numarTicketeZilnice<<"\n";
    }
};
class angajatDepartamentTehnic:public angajat{
protected:
    int oreSuplimentare;
    double plataOreSuplimentare;
public:
    angajatDepartamentTehnic(string nume="", double salariu=0, int oreSuplimentare=0, double plataOreSuplimentare=0):angajat(nume,salariu){
        this->oreSuplimentare=oreSuplimentare;
        this->plataOreSuplimentare=plataOreSuplimentare;
    }
    void afiseaza(ostream& out){
        angajat::afiseaza(out);
        out<<"Numar ore suplimentare:"<<this->oreSuplimentare<<"\n";
        out<<"Plata pentru orele suplimentare:"<<this->plataOreSuplimentare<<"\n";
    }
};
class manager: public angajatDepartamentHr, angajatDepartamentTehnic{
private:
    double comision;
public:
    void afiseaza(ostream& out){
        out<<"Nume:"<<this->nume<<"\n";
        out<<"Salariu:"<<this->salariu<<"\n";
        out<<"Numar tickete:"<<this->numarTicketeZilnice<<"\n";
        out<<"Numar ore suplimentare:"<<this->oreSuplimentare<<"\n";
        out<<"Plata pentru orele suplimentare:"<<this->plataOreSuplimentare<<"\n";
        out<<"Comisionul este:"<<this->comision<<"\n";
    }
};
int main(){
    manager x;
    x.afiseaza(cout);
    return 0;
}
```

În urma executării codului de mai sus o să obținem următoarea eroare:

```
File              Line  Message
                        === Build: Debug in www (compiler: GNU GCC Compiler) ===
C:\Users\Asu...         In member function 'virtual void manager::afiseaza(std::ostream&)':
C:\Users\Asu... 49      error: request for member 'nume' is ambiguous
C:\Users\Asu... 5       note: candidates are: std::string angajat::nume
C:\Users\Asu... 5       note:                 std::string angajat::nume
C:\Users\Asu... 50      error: request for member 'salariu' is ambiguous
C:\Users\Asu... 6       note: candidates are: double angajat::salariu
C:\Users\Asu... 6       note:                 double angajat::salariu
                        === Build failed: 2 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

Deoarece data membrul **nume,** respectiv data membrul **salariu**, creează ambiguitate, fiind dublat. Același lucru se întâmplă și cu metodele dublate.

Pentru a înțelege cum funcționează, ne uităm la reprezentare obiectelor în memorie.

Moștenirea pune implementarea a două obiecte, una după alta, dar în cazul de față, clasa **manager**, este atât **angajatDepartamentHr**, cât și **angajatDepartamentTehnic**, astfel, clasa **angajat** oferă duplicate în interiorul obiectului **manager.** Compilatorul recunoaște acest lucru și ne oferă o eroare de ambiguitate.

Nu își dă seama la care **nume** facem referire, la cel moștenit din **angajatDepartamentHr** sau la cel din **angajatDepartamentTehnic**.

Spunem astfel că întâlnim **problema diamantului.**

Din fericire, C++ ne permite să rezolvăm problema aceasta folosind moștenirea **virtuală.** Pentru a preveni o astfel de eroare, folosim cuvântul cheie **virtual** atunci când realizăm moștenirea, pentru toate clasele derivate ce vor fi folosite ulterior într-o derivare de tip diamant.

```cpp
#include <iostream>
using namespace std;
class angajat{
protected:
    string nume;
    double salariu;
public:
    ...`
};
class angajatDepartamentHr:public virtual angajat{
protected:
    int numarTicketeZilnice;
public:
    ...
};
class angajatDepartamentTehnic:public virtual angajat{
protected:
    int oreSuplimentare;
    double plataOreSuplimentare;
public:
    ...
};
class manager: public angajatDepartamentHr, angajatDepartamentTehnic{
private:
    double comision;
public:
    ...
};
```

Atunci când folosim moștenire virtuală, garantăm obținerea unei singure instanțe a bazei comune. Adică vom avea în clasa **manager**, o singură instanță a clasei **angajat**, distribuită de **departamentHr** și **departamentTehnic**. Având o singură instanță a clasei **angajat**, am rezolvat imediat problema compilatorului legată de ambiguitate.

Compilatorul oferă tabelul virtual pentru clasele angajatDepartamentHr, respective pentru angajatDepartamentTehnic. Atunci când un obiect de tip manager este construit, o să creeze o instanță pentru angajat, o instanță pentru angajatDepartamentHr și una pentru angajatDepartamentTehnic. Clasele angajatDepartamentHr și angajatDepartamentTehnic au un pointer virtual in tabelele lor virtuale care reține un offset către clasa angajat. Dacă din clasa angajatDepartamentHr sau angajatDepartamentTehnic se accesează un câmp din clasa angajat, se folosește de pointerul virtual din tabelul virtual pentru a găsi obiectul angajat si pentru a accesa acel câmp.

Constructor:

Deoarece este o singură instanță virtuală a clasei de bază ce este folosită de clase multiple ce o moștenesc, constructorul pentru clasa virtuală de bază nu este apelat de clasele ce îl moștenesc.

În exemplul de mai jos, clasa manager cheamă direct constructorul din clasa angajat, apoi constructorii din angajatDepartamentHr și angajatDepartamentTehnic. Dacă angajatDepartamentHr și angajatDepartamentTehnic încearcă să invoce constroctorul din angajat, acest apel o să fie sărit atunci când este creat obiectul manager.

Exemplu:

```cpp
manager(string nume="", double salariu=0, int numarTicketeZilnice=0, int oreSuplimentare=0, double plataOreSuplimentare=0, double comision=0)
                                    :angajat(nume,salariu),
                                    angajatDepartamentHr(nume,salariu,numarTicketeZilnice),
                                    angajatDepartamentTehnic(nume, salariu, oreSuplimentare, plataOreSuplimentare){
    this->comision=comision;
}
```

Constructorul din clasa virtuală de bază este mereu chemat înaintea claselor de bază non-virtuale. Acest aspect asigură o moștenire sigură de folosit.

Destructorul în ordinea ierarhiei rulează în mod opus față de constructor. Astfel, destructorul pentru clasa virtuală de bază o să fie apelat la final, pentru că este primul obiect care a fost construit.

O implementare corectă se poate observa în sursa atașată.
(16_problemaDiamantului.cpp)