# Curs 10

## SUPRADEFINIREA (CONT.)

Funcții virtuale pure, clase abstracte, interfețe

# Funcțiile virtuale pure

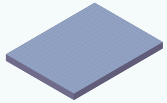
- Pot exista situații când o funcție nu are neapărat sens pentru o clasă de bază, ci doar pentru clasele derivate
- Sau, privind problema din punctul de vedere opus, situații când vreau să oblig clasele derivate să supradefinească anumite metode (astfel încât să nu le moștenească implementarea din clasa de bază)
- Dacă metoda nu conține cod (cazul în care returnează void, de exemplu) sau returnează o valoare aleatoare, problema de mai sus nu va fi rezolvată, deoarece această implementare va fi moștenită de clasele derivate

# Funcțiile virtuale pure

- Din acest motiv au fost introduse **funcțiile virtuale pure**, funcții virtuale ce nu au corp și doar obligă clasele derivate să le supradefinească
- Pentru a nu fi confundate cu signaturi de metode ce vor fi implementate outline, funcțiile virtuale pure au un mod special de a fi declarate, de forma:

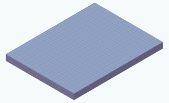
```
virtual tip_returnat nume_functie(parametri) = 0;
```

# Clase abstracte



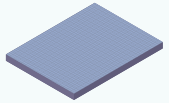
- O clasă ce conține cel puțin o metodă virtuală pură poartă denumirea de **clasă abstractă**
- Clasele abstracte nu se pot instanția (nu se pot crea obiecte de tipul clasei abstracte), ele au rolul doar de bază pentru derivări
- Cu toate că nu se pot instanția, clasele abstracte pot conține atribute, constructori, destructori și alți membri; rolul acestora va fi de a fi moșteniți și utilizați de clasele derivate, nu a de a fi apelați în mod direct (lucru ce nu este posibil - va returna eroare de compilare)

# Exemplu: Clasă abstractă



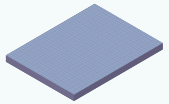
```
class DispozitivIncarcabil
{
protected:
     int nivelBaterie;
public:
     DispozitivIncarcabil()
     {
          nivelBaterie = 0;
     }
     virtual void incarca(int nivel) = 0;
};
```

# Exemplu: Instanțiere clasă abstractă



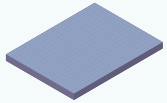
```
int main()
{
    //eroare de compilare, nu se pot instantia clase abstracte
    //DispozitivIncarcabil d;
    //Cu toate acestea se pot declara pointeri la clase abstracte
    DispozitivIncarcabil* pd = nullptr;
    //Linia de cod de mai sus este permisa
}
```

# Exemplu: Derivare clasă abstractă și supradefinire metodă



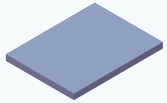
```
class TelefonMobil : public DispozitivIncarcabil
{
public:
	//...
	//supradefinirea functiei incarca() este obligatorie
	//daca dorim ca aceasta clasa sa nu devina la randul ei abstracta
	void incarca(int nivel)
	{
		if(nivelBaterie + nivel <= 100)
			nivelBaterie += nivel;
	}
};
```

# Exemplu: Utilizare pointer la clasă abstractă pentru apel metodă virtuală pură



```
int main()
{
     TelefonMobil t;
     DispozitivIncarcabil* pd = nullptr;
     pd = &t;
     //va duce la apelul functiei din TelefonMobil
     pd->incarca(10);
}
```

# Precizări

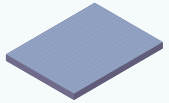


- Așadar singurul rol al claselor abstracte este de a fi derivate (constituie bază pentru derivări)
- Crearea lor, fără a le derivarea este în mare măsură inutilă
- Clasele abstracte, de obicei, modelează un comportament pe care îl impun apoi tuturor claselor derivate (prin intermediul metodelor virtuale pure)
- Un alt mod de a spune asta, pe care îl veți regăsi în literatura de specialitate: clasele abstracte reprezintă contracte - a deriva o clasă abstractă e același lucru cu a semna un contract (te obligi să supradefinești toate metodele virtuale pure)

# Interfețe

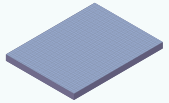
- Cu toate că în limbajul C++ nu există un cuvânt cheie sau o modalitate propriu-zisă de a defini interfețe, acestea sunt privite mai mult ca o convenție
- În C++ numim **interfață** o clasă abstractă ce conține doar metode virtuale pure (și nimic altceva)
- A nu se confunda noțiunea de interfață (tip special de clasă abstractă) cu noțiunea de interfață a clasei (mulțimea membrilor publici ai clasei respective)

# Exemplu: Interfață



```
class Comparabil
{
public:
     virtual int compara(Comparabil* c) = 0;
};
```

# Exemplu: Derivare interfață



```
class TelefonMobil : public Comparabil
{
//...
public:
     int compara(Comparabil* c)
     {
          TelefonMobil* t = (TelefonMobil*)c;
          if(this->nivelBaterie < t->nivelBaterie) return -1;
          else if (this->nivelBaterie > t->nivelBaterie) return 1;
          else return 0;
     }
};
```

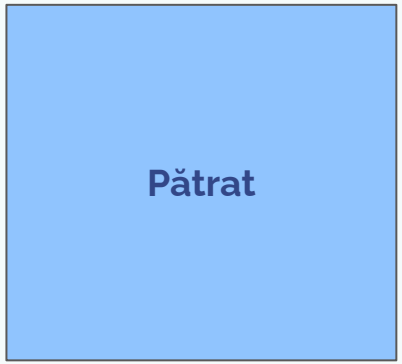
# Interfețele în alte limbaje de programare

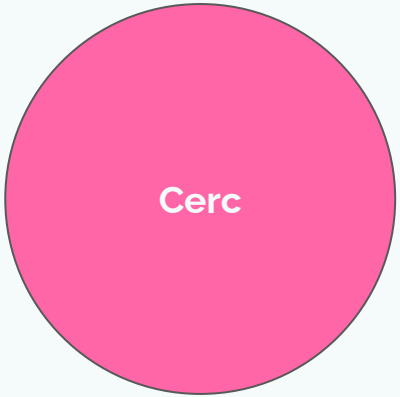
- În alte limbaje de programare orientate obiect (C#, Java, Kotlin, ș.a.) interfețele se termină cu „able” tocmai pentru a sugera un comportament (ex: Comparable, Cloneable, Runnable, etc.)
- În unele limbaje (C#) chiar încep cu „I” pentru a sublinia faptul că sunt interfețe
- De asemenea se utilizează cuvânt cheie separat (interface) pentru a fi definite
- Moștenirea multiplă este permisă în aceste limbaje doar dintr-o clasă și oricâte interfețe

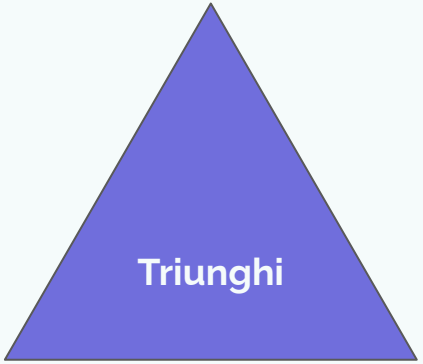
# Discuție

Cum rezolvă acest lucru problemele specifice moștenirii multiple din C++?

# Exercițiul 1




**Cum putem obliga aceste clase să calculeze perimetrul și aria?**

# Exercițiul 2

- Într-o fabrică avem următoarele tipuri de muncitori:
  - muncitor calificat
  - maistru
  - inginer
- Toți muncitorii au următoarele caracteristici:
  - nume
  - prenume
  - salariu de bază
- Cum putem implementa funcția de calcul salarial, știind că un muncitor calificat primește un spor de 10% din salariul de bază, un maistru 20% și un inginer 30% și că orice muncitor poate evolua în carieră în ordinea specificată mai sus?