

Séance 6

Héritage

[Vocabulaire](#)

[Problème à modéliser](#)

[Déclaration de l'héritage](#)

[Stockage des données en mémoire](#)

[Droits d'accès](#)

[Héritage de la partie publique de la classe mère](#)

[Constructeur de la classe fille](#)

[Rédéfinition d'une fonction de la classe mère](#)

[Classes Cadre et Commercial](#)


[Graphe de classes](#)

[Compatibilité au niveau des types](#)

[Fonction virtuelle et classe abstraite](#)

Vocabulaire

Héritage : définir une classe en demandant qu'elle hérite des caractéristiques d'une autre classe déjà écrite et en ajoutant ses caractéristiques propres



classe "fille" ou classe **dérivée**



classe "mère" ou classe de **base**

Problème à modéliser

Un Employé

données : nom + salaire

Catégories particulières d'employé

un Ouvrier est un employé faisant des heures supplémentaires

données : taux horaire + nb heures sup

un Cadre est un employé recevant une prime

données : prime

Catégorie particulière de cadre

un Commercial est un cadre percevant des commissions sur les ventes

données : commission unitaire + nb de ventes

Déclaration de l'héritage

```
class Employe
{
public :
    Employe(const char* no, int sal) {
        strcpy(nom, no);
        salaire = sal;
    }
    void afficheNom() const
        { cout << nom << endl; }

    int calculPaie() const
        { return salaire; }

private :
    char nom[20];
    int salaire;
};
```

un Ouvrier = est un Employé + caractéristiques propres à Ouvrier
est un cas particulier d'Employé
=> classe Ouvrier doit hériter de classe Employe

```
class Ouvrier : public Employe
{
public :
    Ouvrier(...);

    void ajoutHeures(int nbH)
        { nbHeures += nbH; }

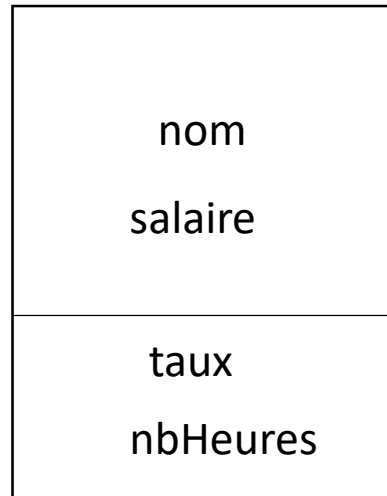
    ...

private :
    int taux;
    int nbHeures;
};
```

*seulement
les fonctions et données
propres à Ouvrier*

Stockage des données en mémoire

un Ouvrier est un Employe => un objet de type Ouvrier contient les données d'un objet de type Employe



données héritées de Employe

données propres à Ouvrier

un objet Ouvrier en mémoire

Droits d'accès

Règle : une classe fille **n'a pas accès à la partie privée** de sa classe mère.

```
// affichage de toutes les infos d'un ouvrier
void Ouvrier::afficher() const {
    ....
    cout << nom << endl;    erreur compilation : nom est privé dans Employe
    ...
}
```

Héritage de la partie publique de la classe mère

La classe fille :

- hérite de la partie publique de sa classe mère
- peut l'utiliser comme ses propres membres

```
// affichage de toutes les infos d'un ouvrier
void Ouvrier::afficher() const {
    ...
    afficheNom();    // fonction de Employe appliquée à l'objet courant Ouvrier
    ...
}
```

Constructeur de la classe fille

Constructeur de Ouvrier doit appeler le constructeur de Employe, dans la liste d'initialisation.

```
Ouvrier::Ouvrier(const char* n, int sa, int tx)
    : Employe(n, sa)
{
    taux = tx;
    nbHeures = 0;
}
```

← initialise les données de Ouvrier héritées de Employe

Quelques cas :

- si pas d'appel du constructeur de Employe, appel automatique du constructeur par défaut de Employe, erreur de compilation si pas de constructeur par défaut
- si héritage et objet membre,
: *Employe(...)*, *un_objet_membre(...)*

Redéfinition d'une fonction de la classe mère

```
class Ouvrier : public Employe
{
public :
    ...
    int calculPaie() const;
    ...
private :
    int taux;
    int nbHeures;
};
```

```
int Ouvrier::calculPaie() const
{
    int salaireFixe = Employe::calculPaie();
    return salaireFixe + taux * nbHeures ;
}
```

*fonction calculPaie de Employe
appliquée à l'objet Ouvrier*

Classes Cadre et Commercial

Un Cadre est un Employe
=> Cadre **hérite de Employe**

Cadre : public Employe

Cadre(n, sa, pri)
: Employe(n, sa)

int calculPaie()
salaire d'employé + prime

int prime

Un Commercial est un Cadre

=> Commercial **hérite de Cadre**

=> Commercial **hérite de Employe**

classe mère directe

classe mère indirecte

Commercial : public Cadre

Commercial(n, sa, pri, com)
: Cadre(n, sa, pri)

int calculPaie()
salaire de cadre + commissions

int commissUnitaire
int nbVentes

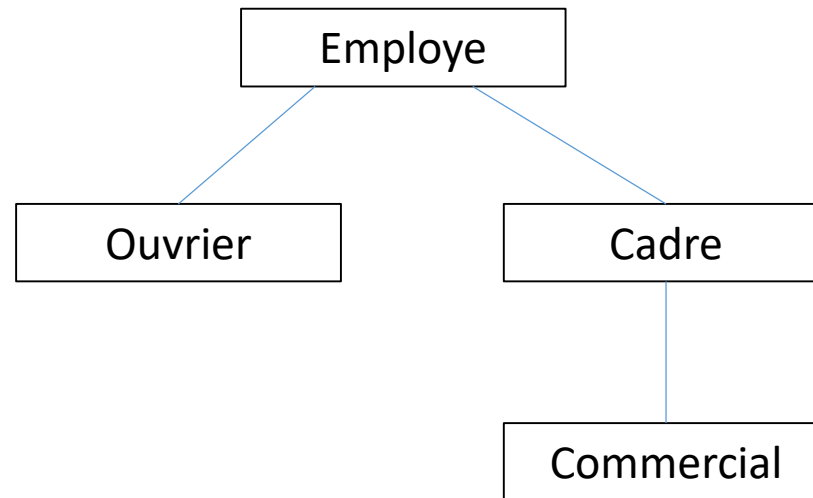
appel du constructeur
de la classe mère directe
seulement

Commercial dispose de
3 fonctions calculPaie :

- calculPaie
- Cadre::calculPaie
- Employe::calculPaie

Graphe de classes

Représentation des relations d'héritage



Compatibilité au niveau des types

Un Ouvrier est un Employe => un objet **Ouvrier** peut **jouer le rôle** d'un objet **Employe**.

```
void f (Employe& emp) {  
    ...  
    emp.afficheNom();  
    ...  
}
```


```
Employe E1(...);  
f (E1);  
  
Ouvrier O1(...);  
f (O1);           // un Ouvrier est un Employe
```

```
void g (Employe* p) {  
    ...  
    p->afficheNom();  
    ...  
}
```

```
Employe E2(...);  
g (&E2);  
  
Ouvrier O2(...);  
g (&O2);           // un Ouvrier est un Employe
```

Fonction virtuelle et classe abstraite

```
void h (Employe* p) {  
    ...  
    cout << p->calculPaie() << endl;  
    ...  
}
```



```
Employe E3(...);  
h (&E3);  
  
Ouvrier O3(...);  
h (&O3);
```

Quelle fonction calculPaie est appelée ?

Comportement par défaut du C++ :

La fonction appelée est celle de Employe (correspondant au **type du pointeur**) \forall le type de l'objet pointé.

Pour changer le comportement (appel de calculPaie correspondant au **type de l'objet pointé**) :
notion de fonction virtuelle (voir Exercice + notion de classe abstraite).

Remarque : la notion s'applique aussi si calculPaie est appelée sur une référence.