# Algorithmique et Programmation 2 : Agrégation et Composition

# Classes en C++ : Associations

Les classes peuvent être reliées structurellement par des relations. En particulier, les relations:

- ***L' agrégation***
- ***La composition***
- ***L'héritage***

# Agrégation : Définition

L'***agrégation*** permet de définir qu'un objet est l'assemblage d'un ou de plusieurs *sous-objets*.

# Agrégation : Remarques

- En ***POO*** :
  - L'objet ***contient*** les sous-objets
  - L'Objet ***communique*** lui même avec les sous-objets

# Agrégation : Règles à suivre

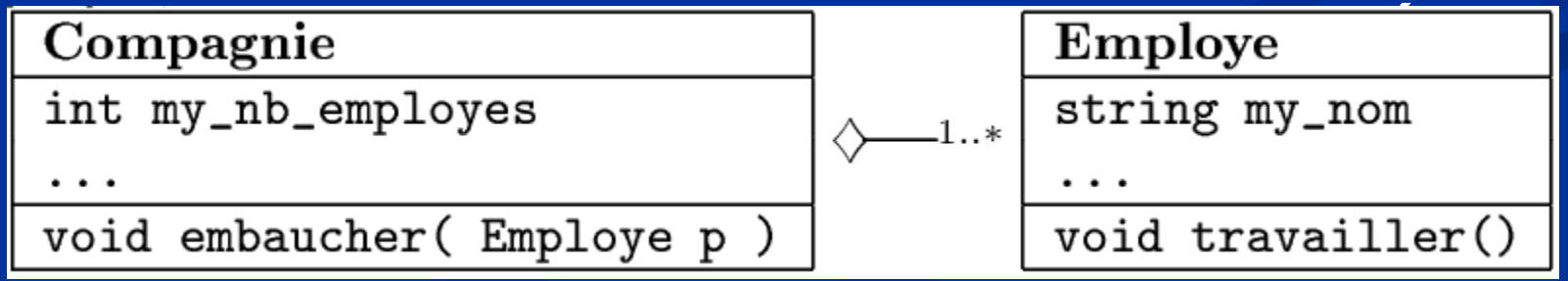
- Les sous-objets doivent avoir une relation structurelle ou fonctionnelle avec l'objet dont ils sont les constituants.
  - Ex : un clavier fait partie d'un ordinateur

# Agrégation : Règles à suivre

- Il faut que la relation d'agrégation respecte les contraintes suivantes :
  - ***Antisymétrie : si*** un objet A fait partie d'un objet B, ***alors*** B ne fait pas partie d'un objet A (même indirectement)
  - ***Transitivité : si*** un objet A fait partie d'un objet B, et que B fait partie d'un objet C, **alors** A doit faire partie d'un objet C. Sinon cela signifie que les champs sémantiques des objets sont mal définis.

# Agrégation : UML

- En UML, on note l'agrégation dans les diagrammes de classes avec une relation non-symétrique terminée sur coté agrégeant par le symbole ◊.
- Ex : si une compagnie possède

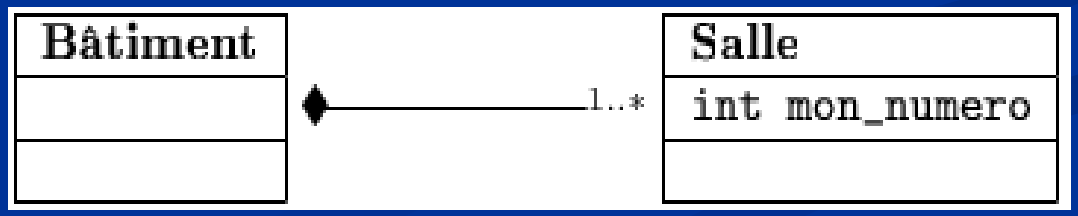


# Agrégation et Composition

- La ***composition*** est un cas particulier d'agrégation qui implique une dépendance plus forte de l'objet agrégé dans l'objet agrégeant.
- Un objet peut être ***agrégé*** (partagé) par plusieurs objets, mais ne peut être ***composé*** que par ***un*** objet unique.
- ***Notion de durée de vie*** : en général les objets qui composent un objet A sont détruits en même temps que A.

# Composition : UML

- En UML, on utilise la même notation pour la ***composition*** que pour l'agrégation, sauf que la pointe ◊ est noircie en ⧫.
- Ex : Un Bâtiment se compose d'un certain nombre de Salles :



# Composition en C++

En C++

Composition : Utiliser un objet comme attribut d'un autre. Par exemple une voiture est composée de 4 roues (une roue appartient à une seule voiture)

```
class Roue {
public:
   ...
private:
...
};
```

```
class Voiture {
public:
   ...
private:
Roue roueAvantGauche_;
Roue roueAvantDroite_;
Roue roueArriereGauche_;
Roue roueArriereDroite_;
...}
```

# Composition

- Si un objet A est un attribut de l'objet B, le constructeur de l'objet A sera appelé avant celui de l'objet B.
- Si vous réfléchissez bien, ceci est logique: pour construire une voiture, il faut d'abord construire ses composantes, comme le moteur et les roues.
- On dit que A et B sont reliés par une relation de composition, c'est-à-dire que B est composé de A
- Il s'agit d'une relation forte: si B est détruit, A disparaît aussi

# Composition en C++ : Exemple (1)

Créer les uniquement les interfaces des classes permettant de modéliser une trottinette (sachant qu'une trottinette est composée de 2 roues inchangeables : si on jette la trottinette on ne peut pas récupérer les roues)

- 2 classes. lesquelles?
- Qui compose qui?
- Les méthodes?
- Les attributs?

# Construction d'un objet composite

- Si une classe X se ***compose*** de sous-objets, alors l'instanciation d'un objet de type X provoque ***d'abord*** l'instanciation des sous-objets de X (et donc des sous-objets des sous-objets de X, etc).
- Il est possible de ***définir quels sont les constructeurs appelés*** pour chacun des sous-objets avec la notation « **:** » dans le corps des constructeurs de X.

# Construction d'un objet composite : Exemple

- Pour une Trottinette, on pourrait écrire :

```
Trottinette::Trottinette()
  : my_roue_avant( 10 ),  // 10 cm
    my_roue_arriere( 10 ) // 10 cm aussi
{ // Ici la trottinette et ses composes sont instancies.
  ...
}
```

# Agrégation/Composition en C++: Remarque

Si l'on ne connait pas à l'avance le nombre d'objets agrégés/composés, on ne peut définir un ***attribut de taille variable*** en C++. On utilisera donc :

- Soit un ***pointeur*** pour faire de l'allocation dynamique
- Soit un ***attribut*** *conteneur* (par ex. un liste, un vecteur, etc,... cf STL)

# Destruction d'un objet composite

- Lors de la ***destruction*** d'un objet de classe X contenant des sous-objets, le destructeur de cet objet est ***d'abord*** appelé, ***puis*** les destructeurs de ses sous-objets sont appelés (en général dans l'ordre de définition)

# Agrégation

- L'agrégation consiste essentiellement en une utilisation d'un objet comme faisant partie d'un autre objet
- Contrairement à la composition, où l'objet inclus disparaît si l'objet englobant est détruit, dans une agrégation cet objet ne disparaît pas
- La manière habituelle d'implémenter l'agrégation en C++ est par l'utilisation de pointeurs

# Agrégation C++

```
class Service
{
  ...
private:
string nom_;
Employe*
receptionniste_;
}
```

Constructeur :

```
Service::Service(string nom)
: nom_(nom),
   receptionniste_(NULL)
{
}
```

Méthode pour associer le réceptionniste:

```
Service::set_receptionniste(Employe* employe)
{
  receptionniste_ = employe;
}
```

# Agrégation C++

- Pour qu'un objet de la classe Service soit réellement intéressant, il nous faut une méthode pour lui associer un employé comme réceptionniste

- Ceci suppose qu'un objet dynamique de la classe Employe a déjà été créé auparavant et que la méthode fera pointer son pointeur sur cet objet

# Agrégation C++

Dans la fonction Principale :

```
int main()
{
...
  Service expedition("Expeditions");
Employe* michel = new Employe("Michel Gagnon", 50000.0);
expedition.set_receptionniste(michel);
...
}
```

# Agrégation C++

On pourrait aussi utiliser un paramètre supplémentaire dans le constructeur:

```
Service::Service(string nom, Employe* employe)
: nom_(nom), receptionniste_(employe)
{
}
int main()
{
...
Service expedition("Expeditions",new Employe("Michel Gagnon", 50000.0));
...
}
```

# Construction/Destruction d'un objet composite : Exercice

Complétez l'interface et le corps des classes Roue et Trottinette. Vérifiez quels constructeurs sont appelés. Vérifiez l'ordre de destruction des objets.

| **Roue** |
| --- |
| - float my_diametre |
| + Roue()<br>+ Roue(float diametre)<br>+ Roue(const Roue &r)<br>+ ~Roue()<br>+ void setDiametre(float diametre)<br>+ float getDiametre() const |

| **Trottinette** |
| --- |
| - Roue my_roue_avant<br>- Roue my_roue_arriere |
| + Trottinette()<br>+ Trottinette(float diametre_avant, float diametre_arriere)<br>+ ~Trottinette()<br>+ void changerRoueAvant(Roue r)<br>+ void changerRoueArriere(Roue r)<br>+ Roue getRoueAvant() const<br>+ Roue getRoueArriere() const |

# Agrégation/Composition en C++ : Exercice (2)

Concevez les classes nécessaires à la modélisation d'une entreprise, un siège social et de ses employés

- Un siège social = une ville
- Un employé = un nom
- Une entreprise = 1 siège social et un nombre >=0 d'employés
- Une compagnie peut recruter un employer
- Si on lui demande, un employé donnera son nom et dira qu'il est content de travailler