Programmation Orientée Objet

Jean-Louis Dewez
P. Graffion

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

Avril 2006

Programmation procédurale

```
Construire(Maison m){
        creuser(fouilles);
        commander(béton)
        couler(fouilles);
        commander(parpaings);
        while(!fini(sous-sol)){poser(parpaings);}
        commander(hourdis);
        while(….
        commander(béton);
        couler(plancher);
        ...
}
```

Programmation non structurée

```
Construire(Maison m){
        creuser(fouilles);
        commander(béton)
        couler(fouilles);
        commander(parpaings);
        while(!fini(sous-sol)){poser(parpaings);}
        commander(hourdis);
        while(….
        commander(béton);
        couler(plancher);
        ...
}
```

```
commander(Béton b){
…
}
couler(Fouilles f){
…
}
commander(Parpaings p){
…
}
couler(Plancher p) {
…
}
```

Structuration par les fonctions

```
package commandes;
        public commander(béton){……}
        … commander(Parpaings p){……}
        … commander(Hourdis o) ){……}
        … commander(Béton b) ){……}
        ...
```

```
package coulages;
        public couler(Fouilles f){……}
        … couler(Plancher p) ){……}
        ...
```

```
package construction;
import commandes.*;
import coulages.couler;
…
construire(Maison m){
        creuser(fouilles);  commander(béton)
        couler(fouilles);  commander(parpaings);
        while(!fini(sous-sol)){poser(parpaings);}
..}
```

Visibilité

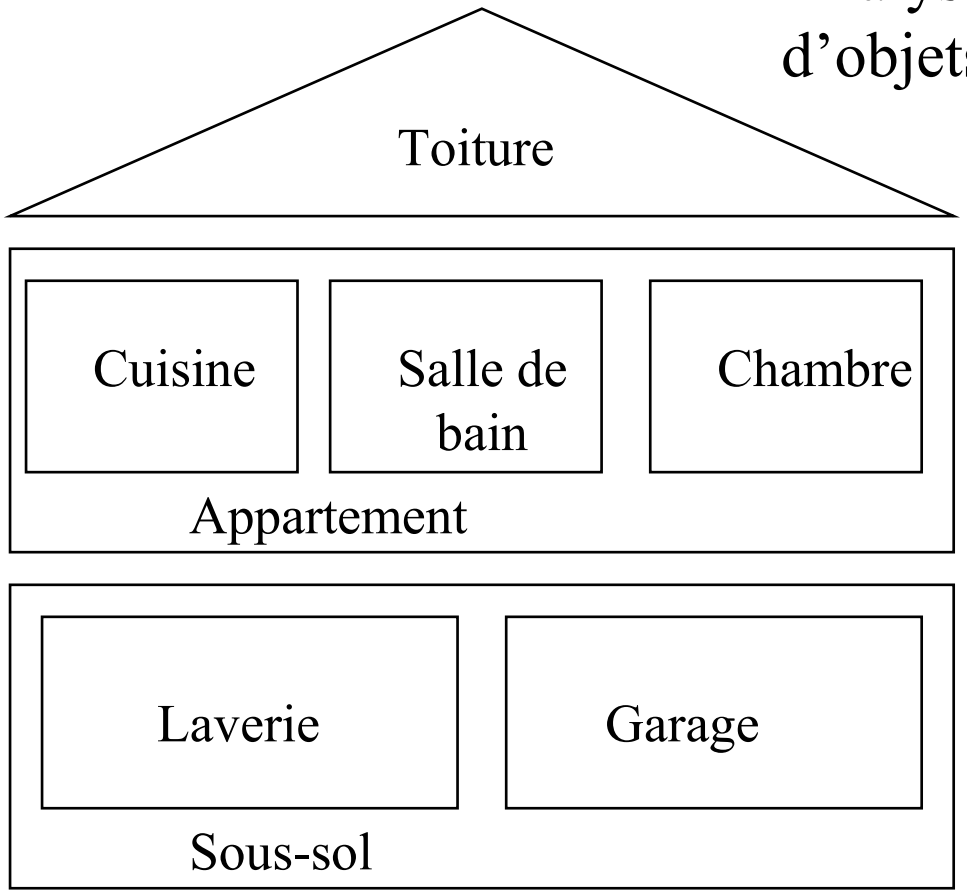
Portée

Protection

Structuration par les objets

Plan de la Maison

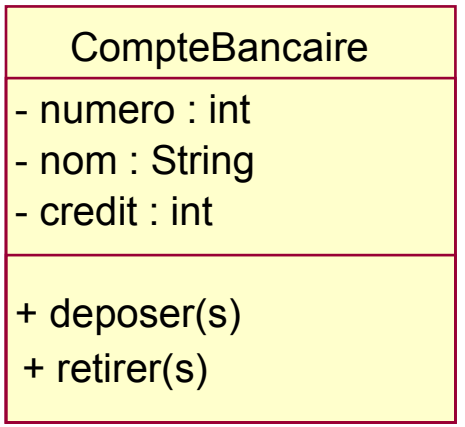
Analyser les classes d'objets du réel



Classe

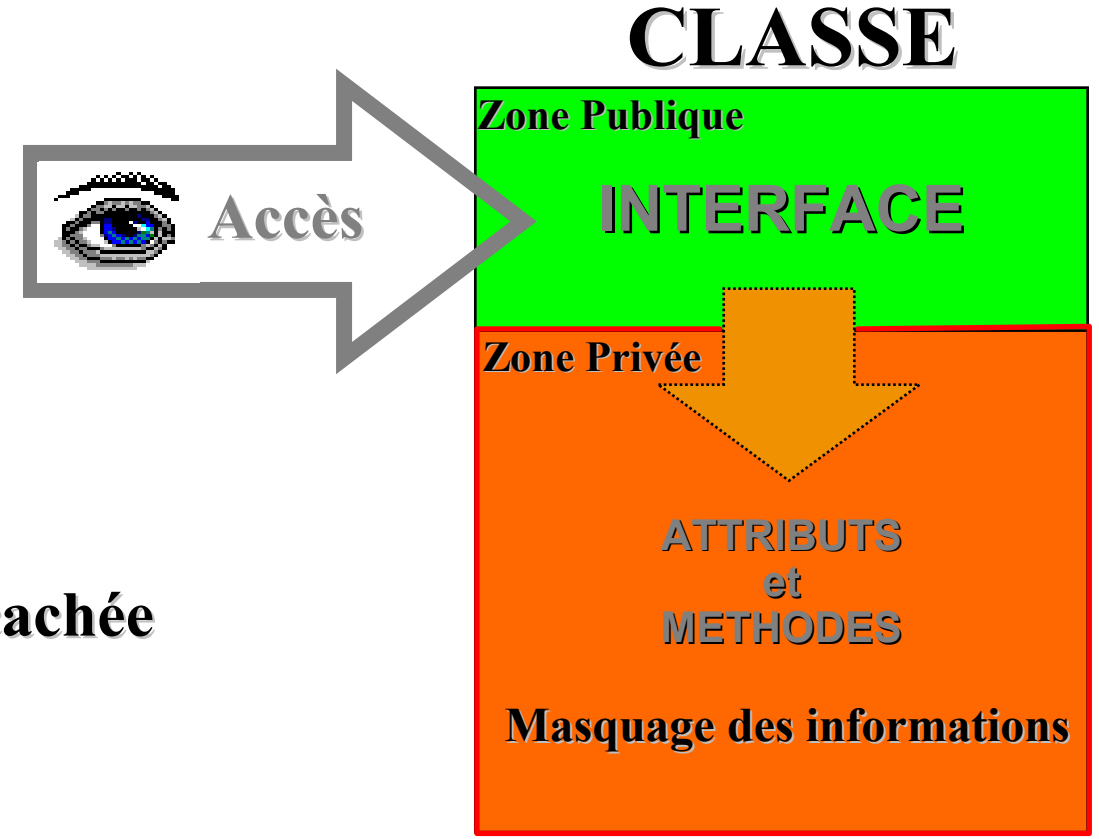
- **Modèle représentant une abstraction**
- **Décrit les propriétés communes d'une collection d'Objets**
  - **Attributs = données**
  - **Méthodes = opérations possibles**

La Classe CompteBancaire



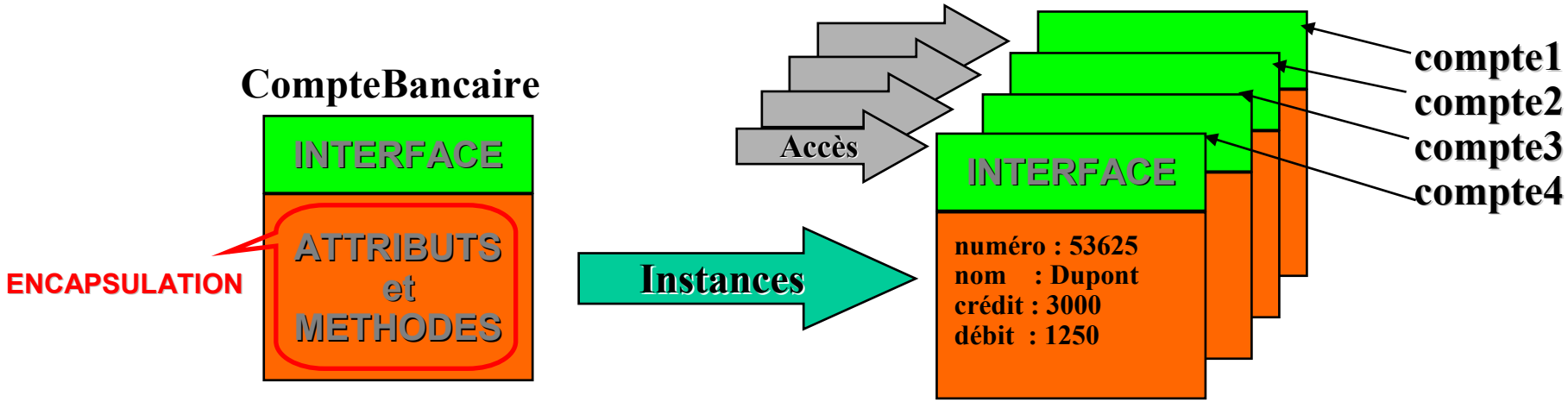
Interface

- Définit les services qu'une classe peut fournir
- Vue restreinte de la classe pour ses clients (ensemble des méthodes publiques)
- Organisation interne cachée



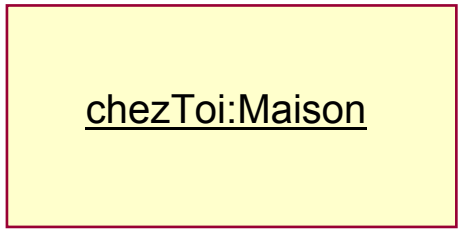
Instance (objet)

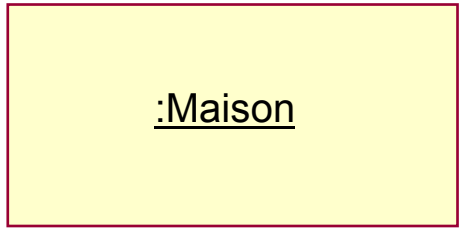
- Objet créé à partir (du Constructeur) d'une Classe
- Comporte les propriétés de la Classe
- Contient des données qui la rend unique




```
compte4 = new CompteBancaire("Dupont", 4250);
```

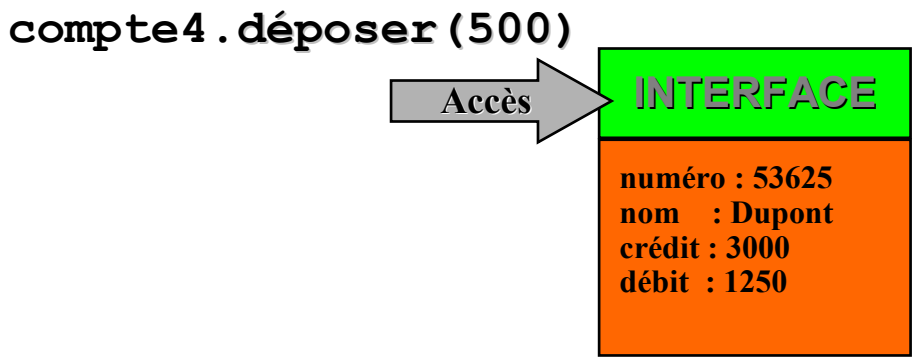
Classe versus Objets (instances)





Message

- Moyen de formuler une requête à un Objet
- Se compose du nom d'une Méthode et d'arguments éventuels
- Exemple: Déposer 500 francs sur le compte de Dupont



Encapsulation

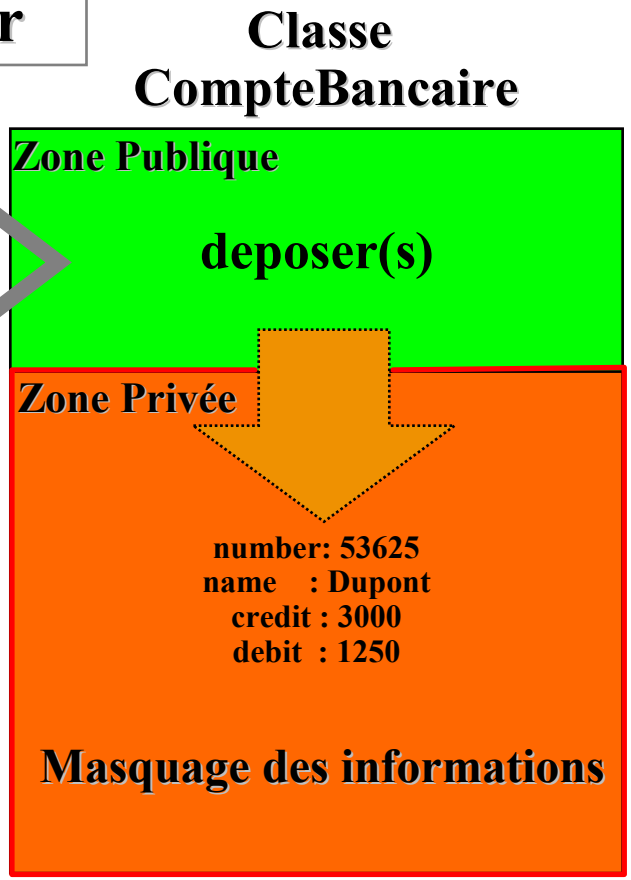
- Masquage des informations - les données sont inaccessibles de l'extérieur

```
compte4.credit = compte4.credit  + 500; // Erreur
```

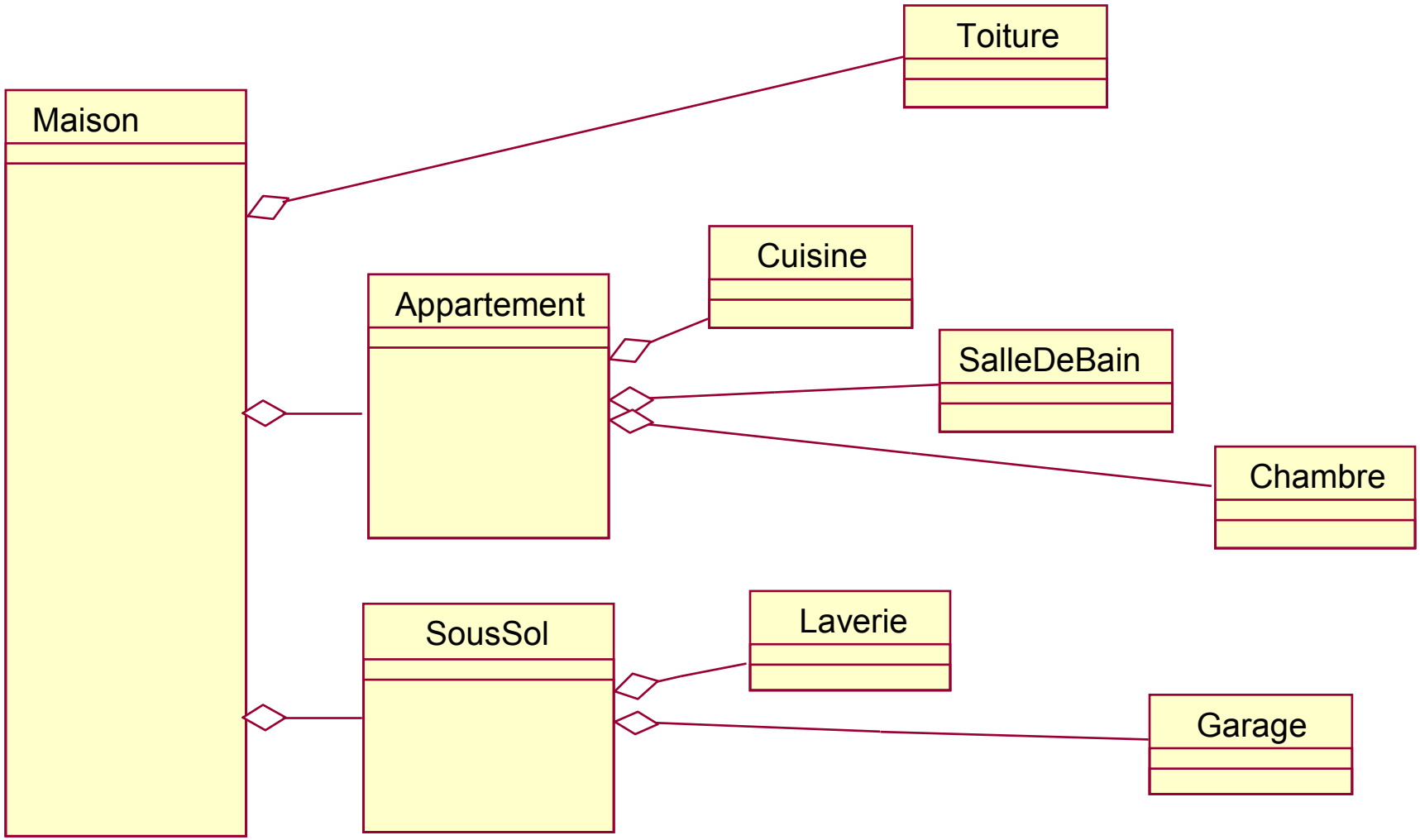
- Accès uniquement via l 'Interface


```
compte4.deposer(500); // OK!
```

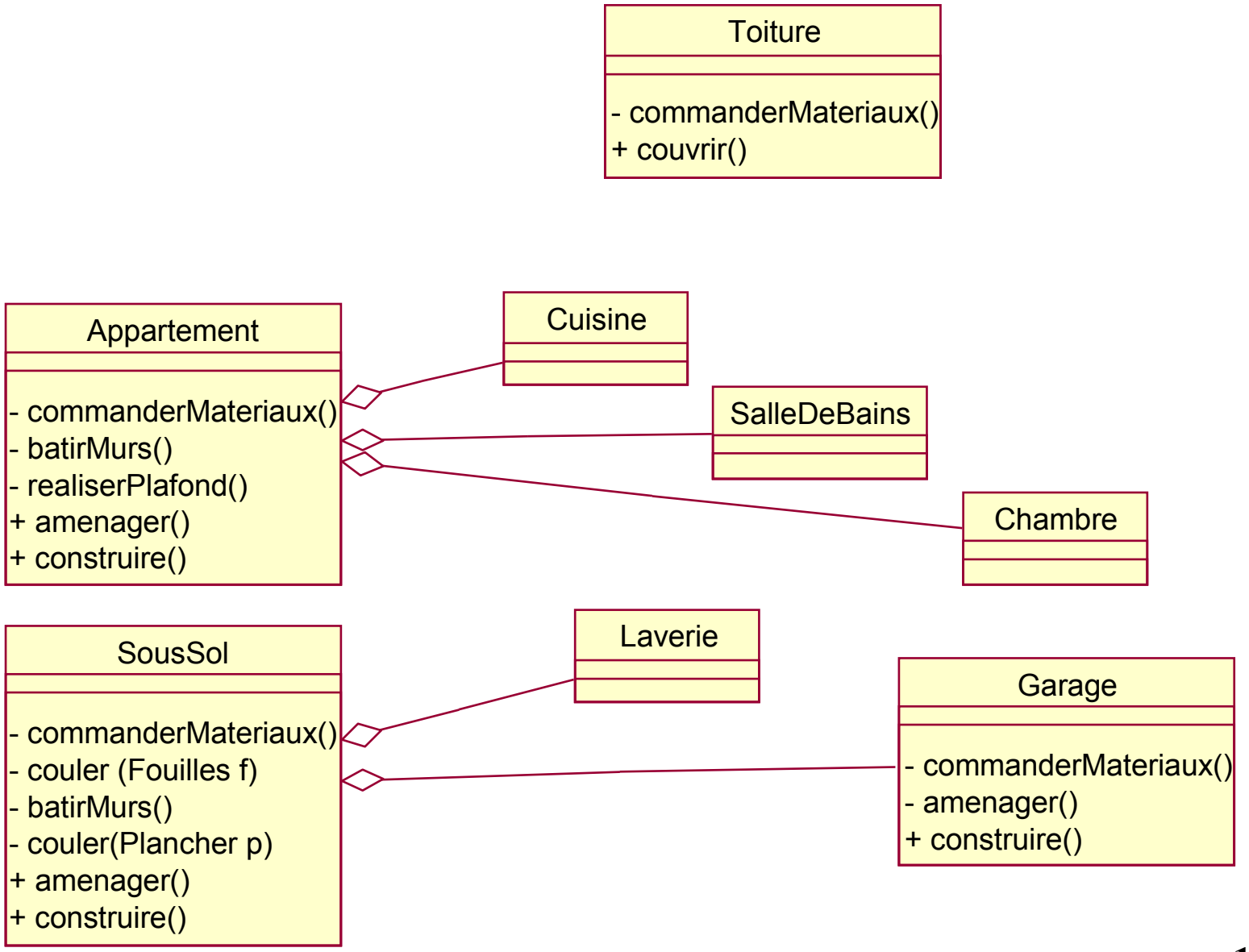
- Organisation interne cachée



Plan du Logiciel (modélisation)

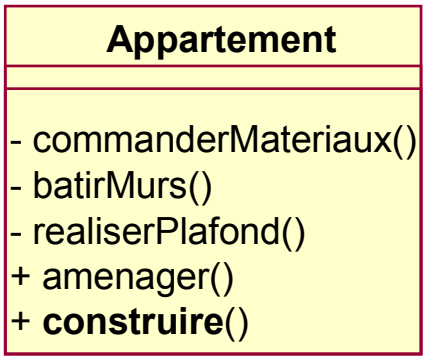


Structurer les fonctions par les classes



Des Objets du réel aux classes PHP5

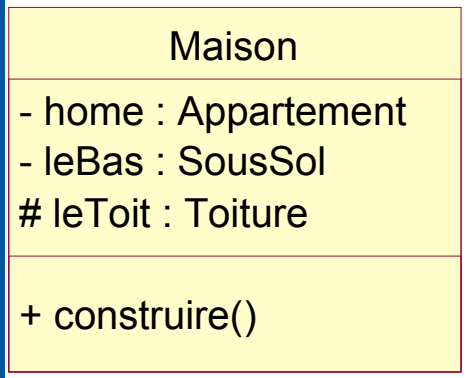
```
public class Appartement {
    ...
    private function commanderMateriaux() {
        ...
    }

    private function batirMurs() {
        ...
    }

    private function realiserPlafond() {
        ...
    }

    private function amenager() {
        ...
    }

    public function construire() {
        $this->commanderMateriaux();
        $this->batirMurs();
        $this->realiserPlafond();
    }

}
```

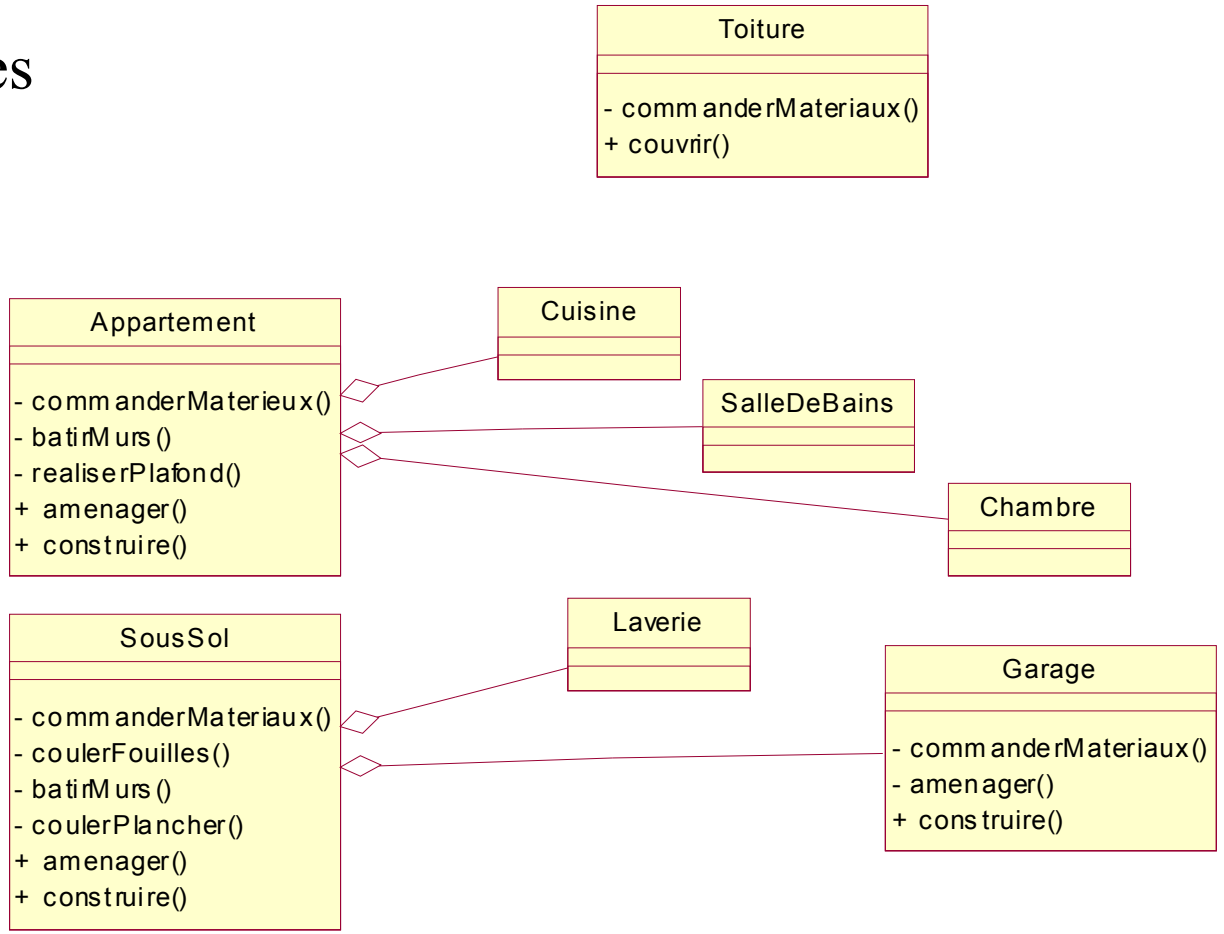


Conventions sur les identificateurs
de classe
de variables

Attributs (propriétés des objets)

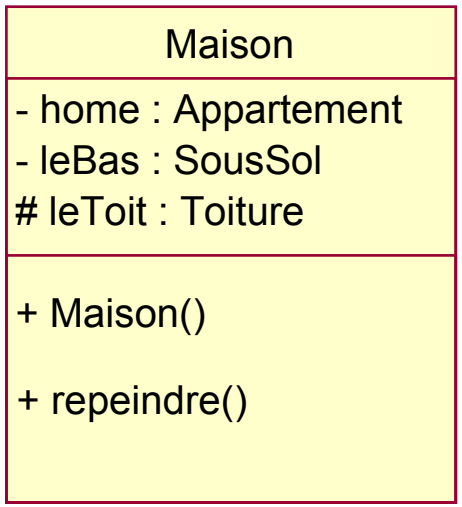
Variables des instances





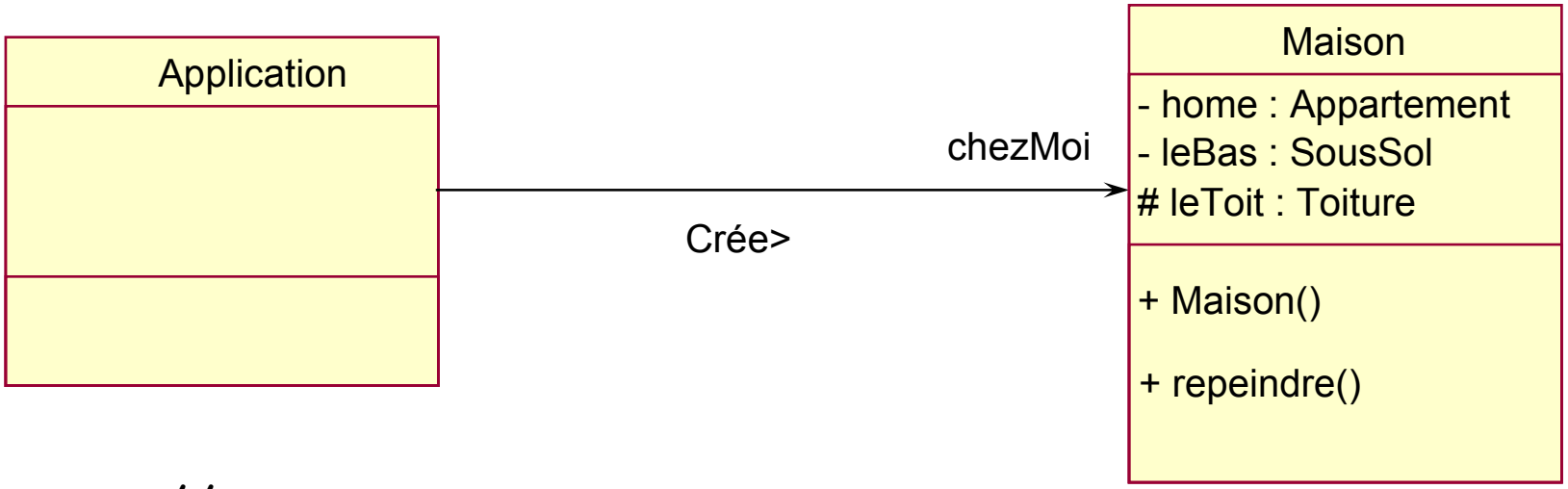
Constructeur ?

Visibilité des attributs




```
// Source file: Maison.php

class Maison {
    private /* Appartement */ $home;
    private /* SousSol */ $leBas;
    protected /* Toiture */ $leToit;

    public function repeindre()
       {
       ...
       }
}
```

Constructeur d'objet

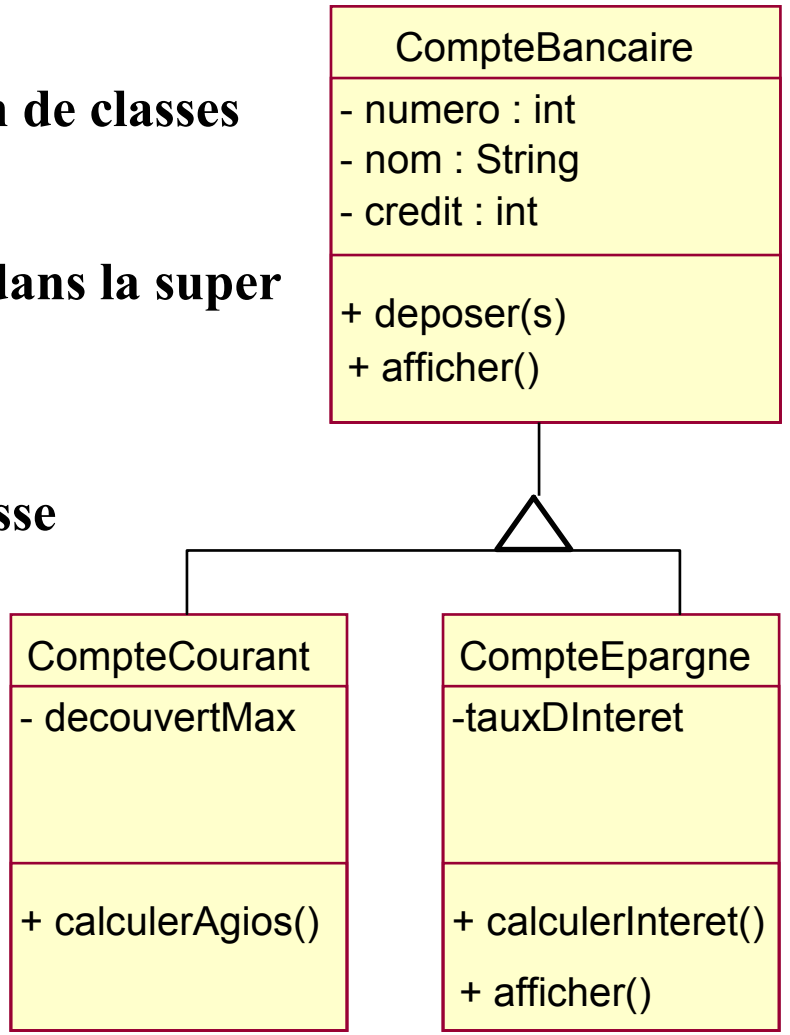



```
// Java
Maison chezMoi;
chezMoi = new Maison();
chezMoi.repeindre();
```

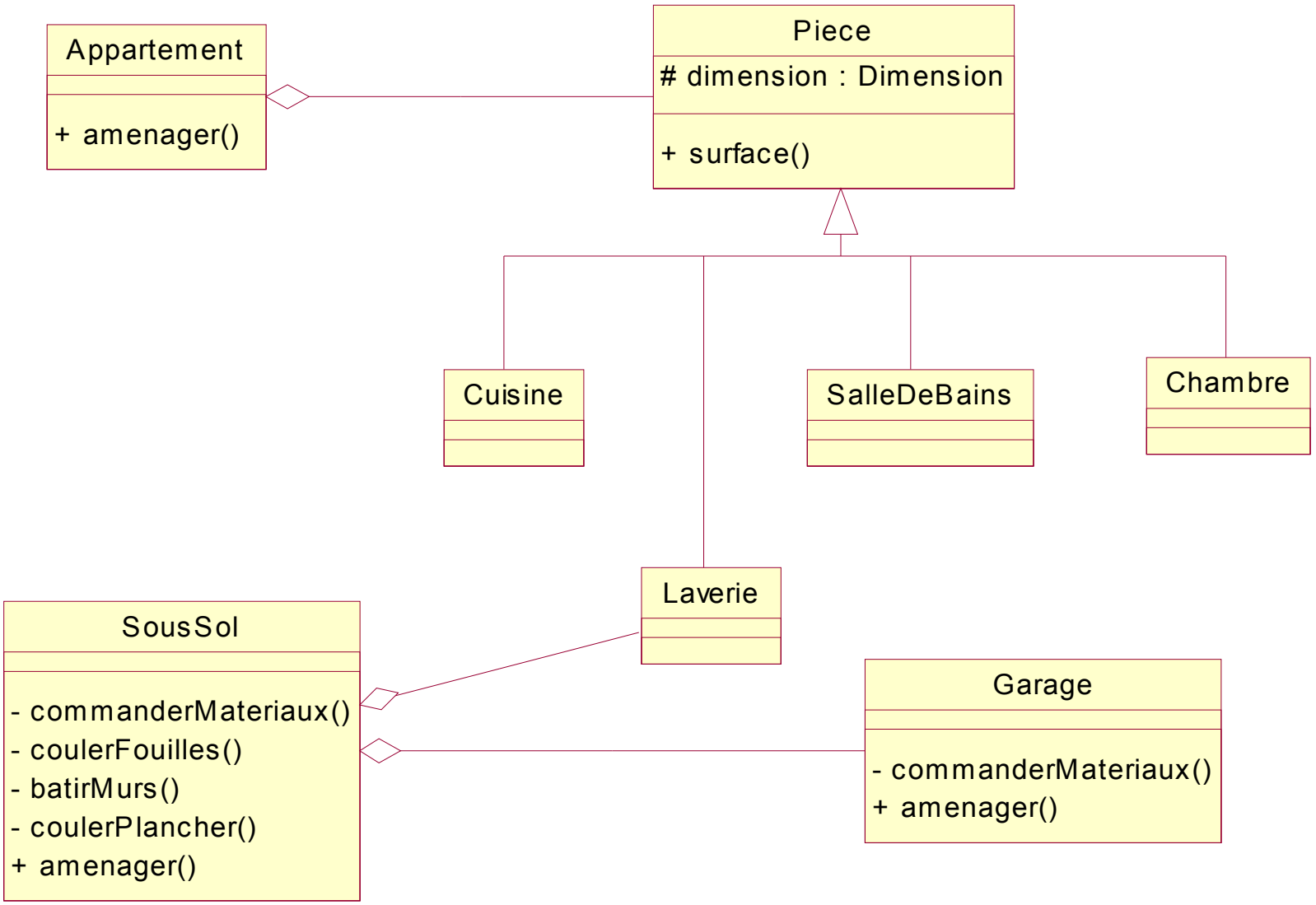
```
// PHP
$chezMoi = new Maison();
$chezMoi->repeindre();
```

Héritage

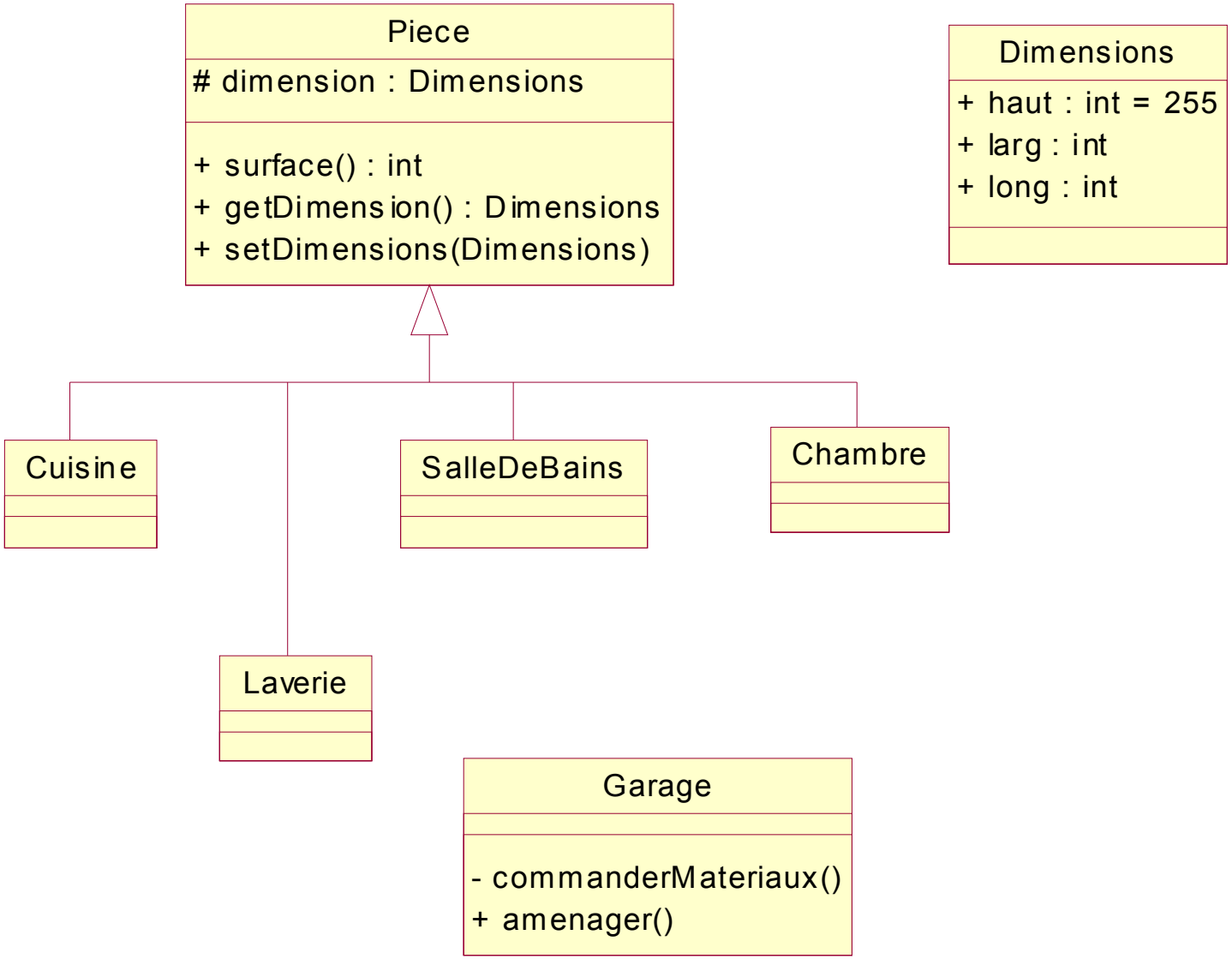
- **Définition de nouvelles classes en fonction de classes existantes**
- **Factorisation des propriétés communes dans la super classe**
- **La sous-classe**
  - **hérite des propriétés de sa super-classe**
  - **peut définir de nouvelles propriétés**
  - **peut redéfinir certaines méthodes**



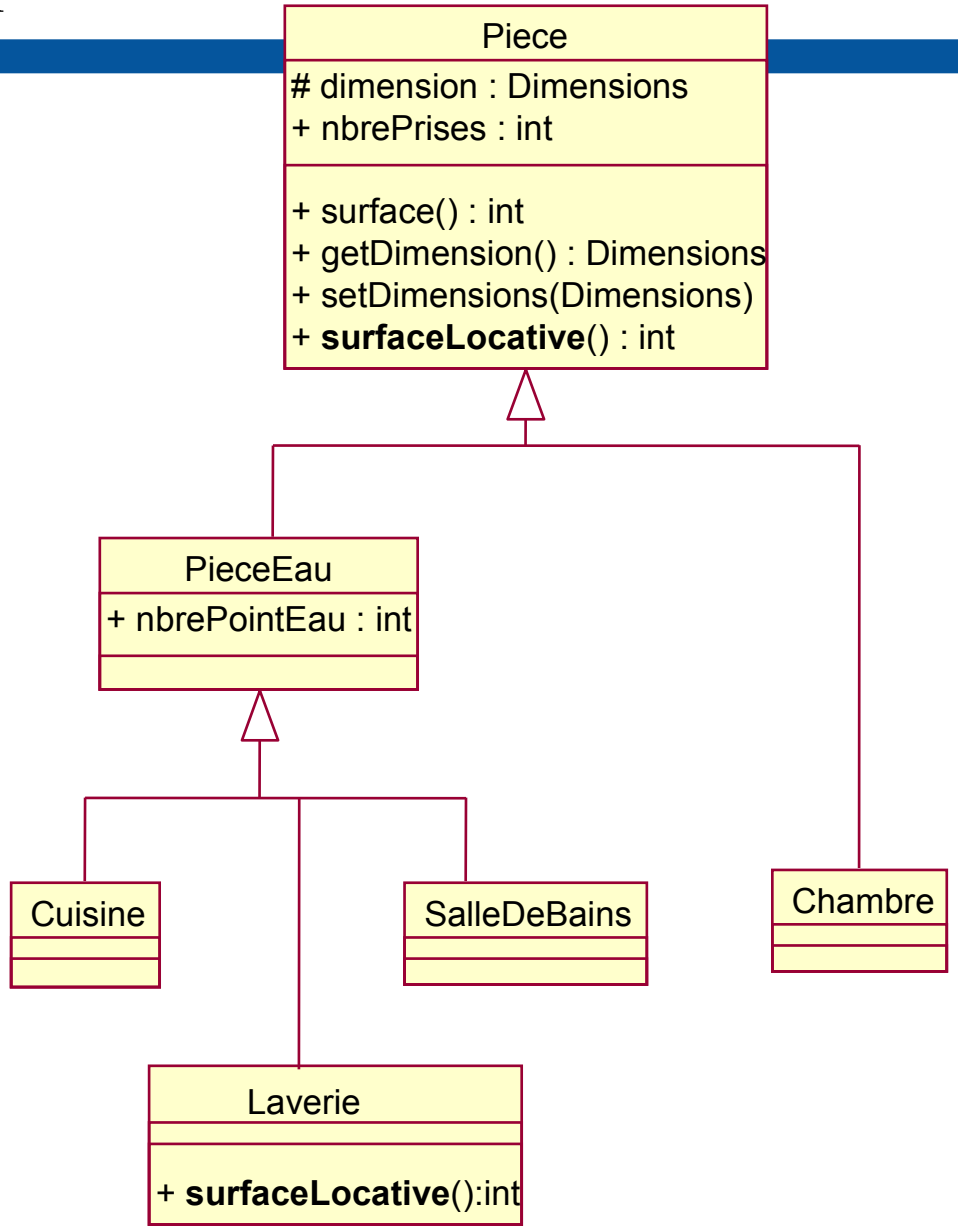
Héritage


Appartement
+ amenager()
Piece
dimension : Dimension
+ surface()
Cuisine
SalleDeBains
Chambre
Laverie
SousSol
- commanderMateriaux()
- coulerFouilles()
- batirMurs()
- coulerPlancher()
+ amenager()
Garage
- commanderMateriaux()
+ amenager()

Accesseurs


Piece
dimension : Dimensions
+ surface() : int
+ getDimension() : Dimensions
+ setDimensions(Dimensions)
Dimensions
+ haut : int = 255
+ larg : int
+ long : int
Cuisine
SalleDeBains
Chambre
Laverie
Garage
- commanderMateriaux()
+ amenager()

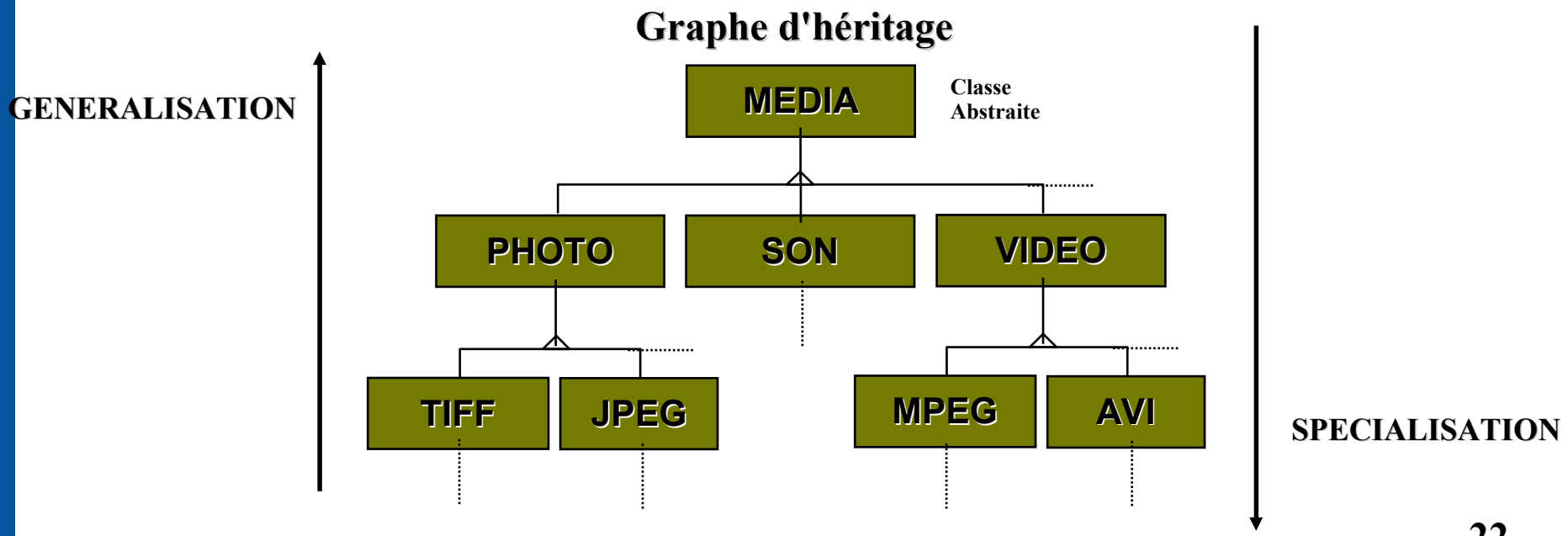
Spécialisation



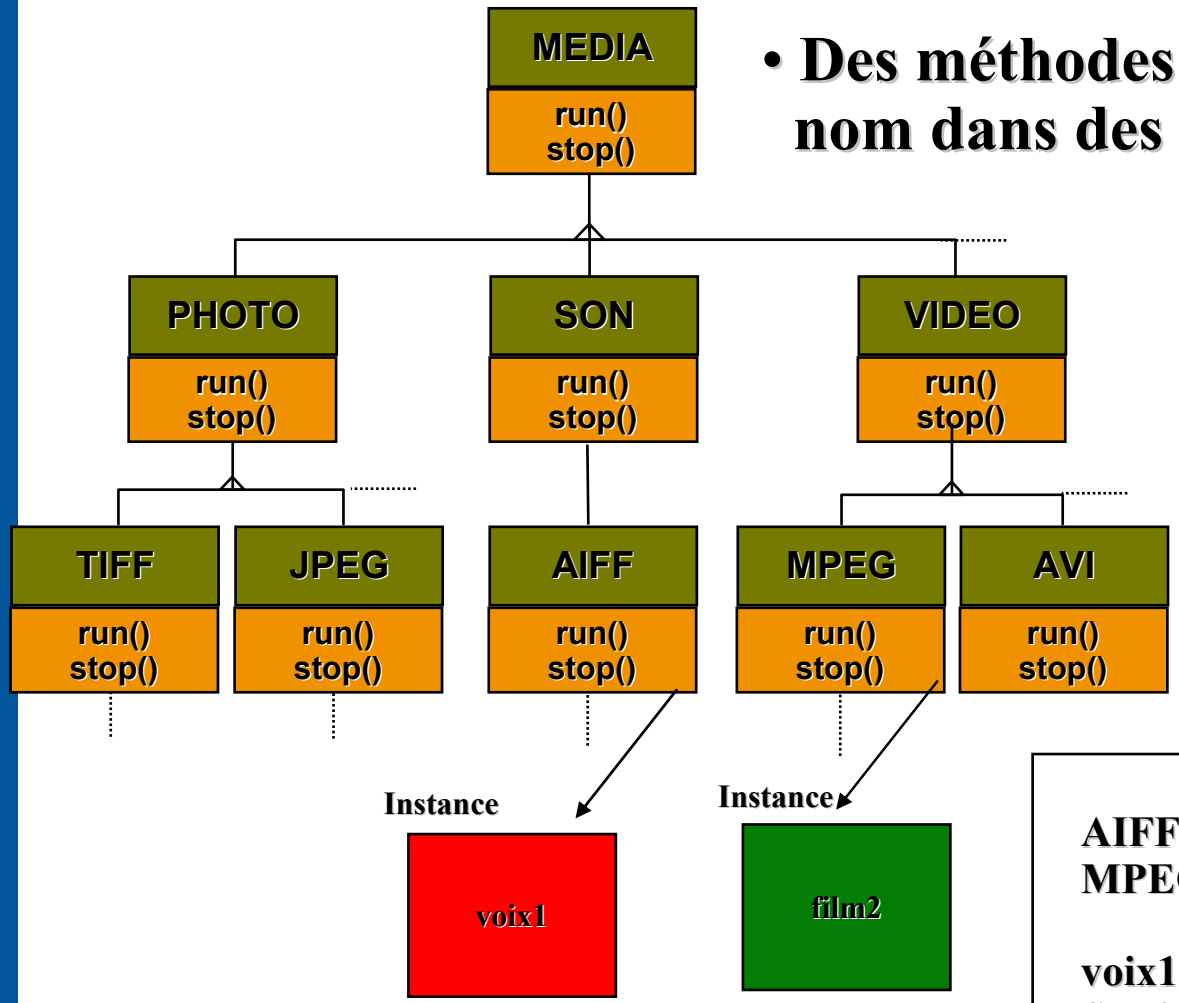
Méthode spécialisée

Héritage – classes abstraites

- Obtention d'une hiérarchie de classes.
- Mécanisme obligatoire pour le polymorphisme.
- Mécanisme important pour la réutilisation, l'évolutivité et la maintenabilité.



Polymorphisme

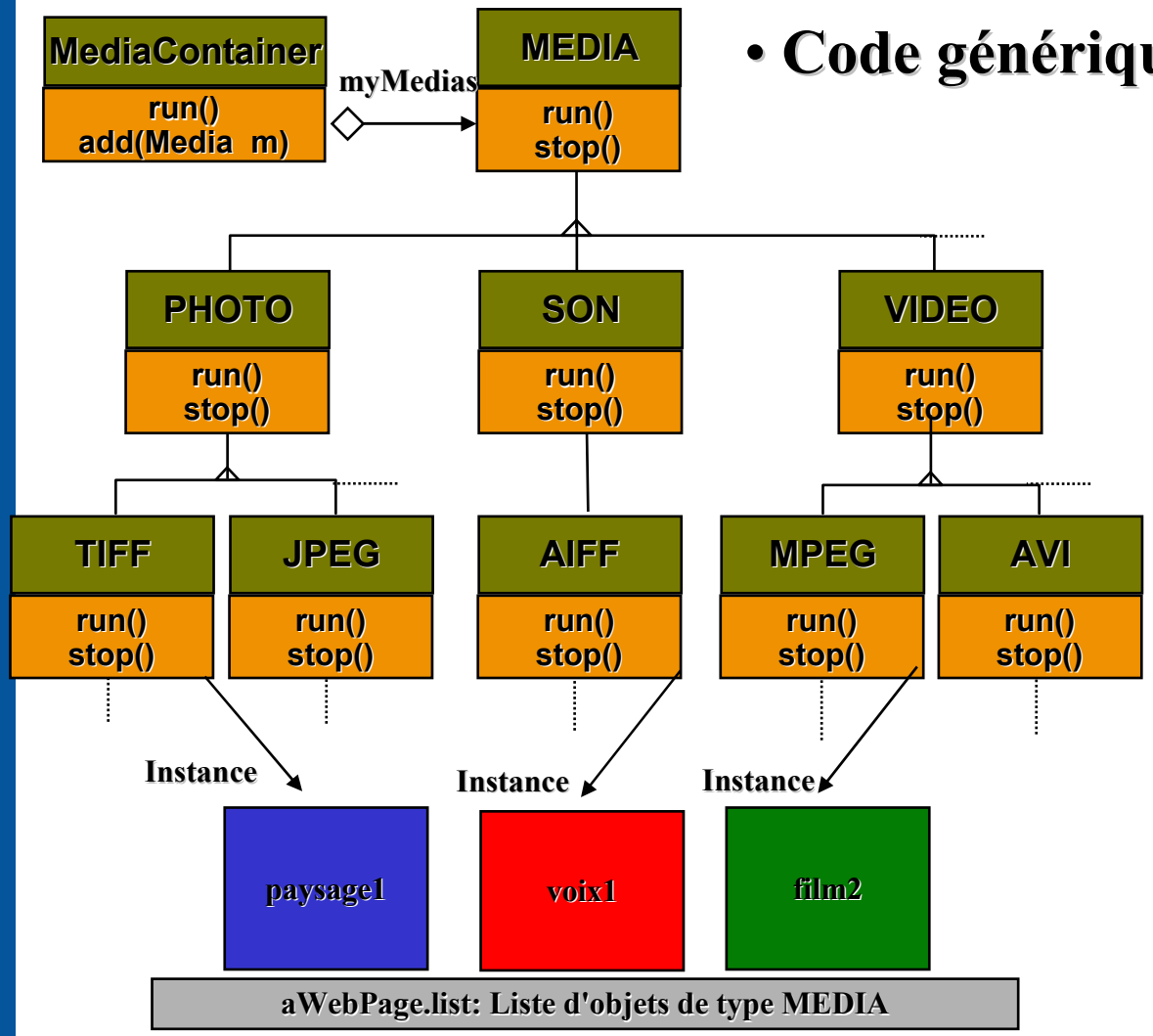


- Des méthodes peuvent porter le même nom dans des classes différentes

- Interface identique

```
AIFF voix1 = new AIFF(...);
MPEG film2 = new MPEG(...);

voix1.run(); // => AIFF.run()
film2.run(); // => MPEG.run()
```

Polymorphisme - Typage Dynamique

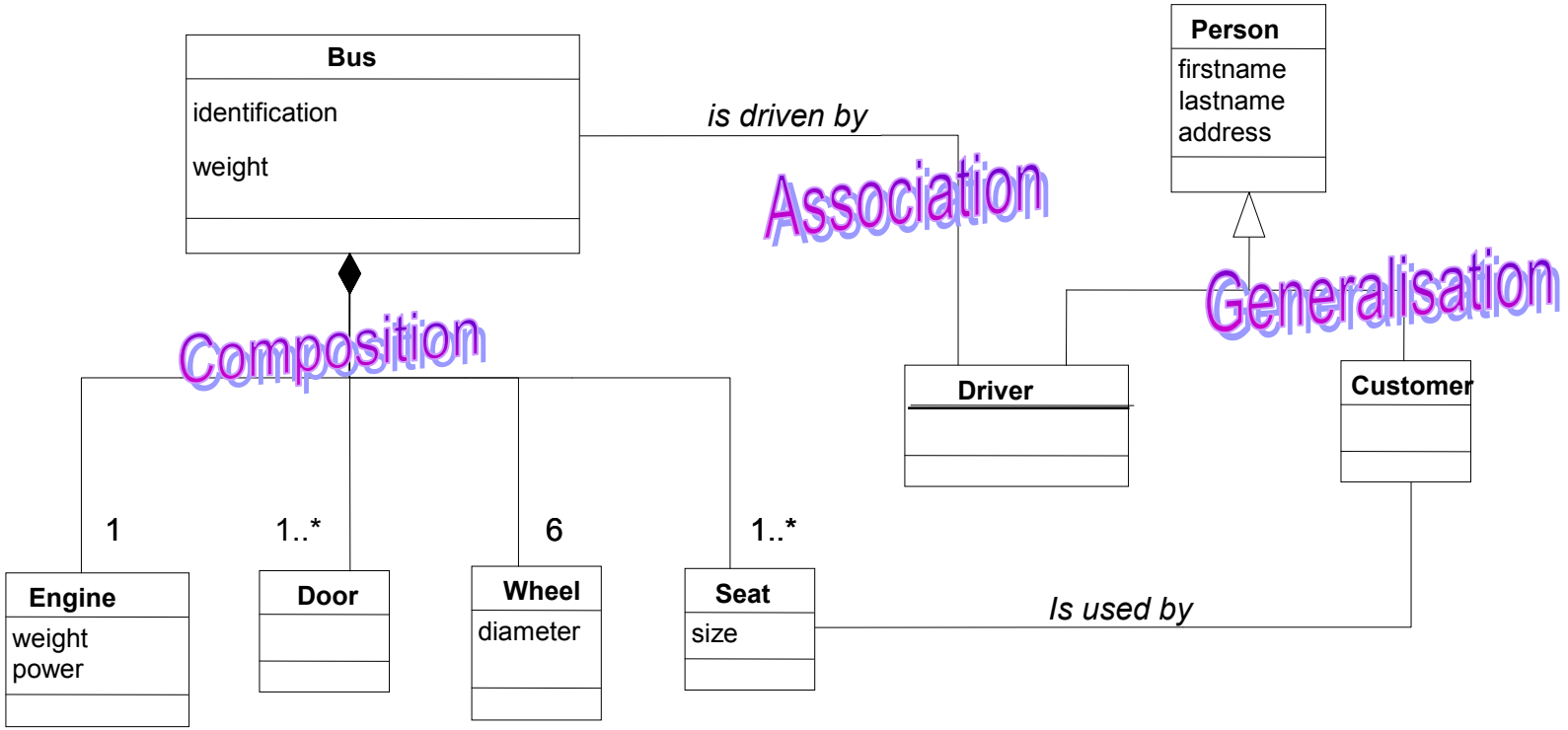


- Code générique et méthodes virtuelles.

```
MediaContainer {
void run()  {
  int i = 0;
  while ( i < nbElem ) {
    Media m = list[i++];
    m.run();
  }
}
```

```
TIFF paysage1 = new TIFF(...);
AIFF voix1 = new AIFF(...);
MPEG film2 = new MPEG(...);
MediaContainer aWebPage = ...;
aWebPage.add(paysage1);
aWebPage.add(voix1);
aWebPage.add(film2);
// affiche la page Web
aWebPage.run();
```

UML – diagramme de classe



UML – diagramme de séquence

Montre les échanges de messages entre les objets