Semestre 2

2013-2014

Concepts de la programmation par objets

Christian Percebois

Contenu du cours

- Classes et objets
- Héritage de classes
- Composition d'objets
- Polymorphisme

Chapitre 1 : les classes et les objets

Définition

Objet : structure de données
constituée de valeurs (champs)
et d'opérations (méthodes)

Analogue à un enregistrement étendu par des sous-programmes

L'objet laLacoste :

désignation	"Chemise"
prixHT	80.0

et calculer le prix de vente d'une chemise,
calculer son coût de livraison,
...

L'objet laBadoit :

désignation	"EauGazeuse"
prixHT	1.0

et calculer le prix de vente d'une chemise,
calculer son coût de livraison,
...

Définition

Classe : description d'une famille d'objets ayant même structure et même comportement

2 facettes :

- *Attributs* (Champs de l'objet)
- *Méthodes*

7

```
class Article
{
        private String désignation ;
        private Float prixHT ;

        // calcule le prix de vente d'un article
        // TVA = 20 %
        public Float prixDeVente ()
        {
                return (prixHT * 1.2) ;
        }

        // calcule le coût de livraison d'un article
        // taxe = 5 %
        public Float coûtLivraison ()
        {
                return (prixHT * 0.05) ;
        }
```

8

```
        // fournit la désignation d'un article
        public String getDésignation ()
        {
                return (designation) ;
        }

        // fournit le prix hors taxes d'un article
        public Float getPrixHT ()
        {
                return (prixHT) ;
        }

        // modifie le prix hors taxes d'un article
        public void setPrixHT (Float p)
        {
                prixHT = p ;
        }
}
```

9

Définitions

Constructeur : méthode spécifique de création et d'initialisation des objets d'une classe (instanciation)

Opérateur d'instanciation : opération qui permet de créer un objet (en général, opérateur new)

10

```
class Article
{
        …

        // constructeur : initialise les champs d'un article
        public Article (String d, Float p)
        {
                désignation = d ;
                prixHT = p ;
        }

        // autres méthodes
        …

}
```

11

```
// déclaration de l'instance laLacoste
Article laLacoste ;

// création de l'instance laLacoste
laLacoste = new Article ("Chemise", 80.0) ;


// déclaration de l'instance laBadoit
Article laBadoit ;

// création de l'instance laBadoit
laBadoit = new Article ("EauGazeuse", 1.0) ;
```

12

Principe d'encapsulation

La représentation en mémoire d'un objet et l'implémentation de ses méthodes restent cachées et inaccessibles au monde extérieur

*L'objet est vu comme une **boîte noire***

Chapitre 2 : l'héritage de classes et la composition d'objets

Définition

Héritage : mécanisme permettant le partage et la réutilisation de propriétés entre objets

- *Propriétés = Attributs + Méthodes*
- *Héritage ≠ Partage de code*


```
class Chemise extends Article
{
        private Int taille ;
        private Couleur coloris ;

        // constructeur de la classe Chemise
        public Chemise (String d, Float p, Int t, Couleur c)
        {
                super (d, p) ;
                taille = t ;
                coloris = c ;
        }
}
```



```
class EauGazeuse extends Article
{
        // constructeur de la classe EauGazeuse
        public EauGazeuse (String d, Float p)
        {
                super (d, p) ;
        }

        // redéfinition du calcul du prix de vente
        // de la classe Article
        // TVA = 5,5 %
        public Float prixDeVente ()
        {
                return (getPrixHT () * 1.055) ;
        }
}
```



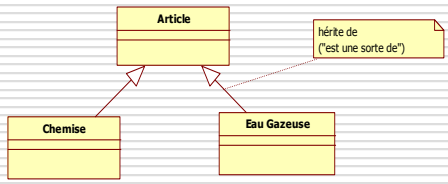
```
// déclaration de l'instance laLacoste
Chemise laLacoste ;

// création de l'instance laLacoste
laLacoste = new Chemise ("Chemise", 80.0, 1, rouge) ;


// déclaration de l'instance laBadoit
EauGazeuse laBadoit ;

// création de l'instance laBadoit
laBadoit = new EauGazeuse ("EauGazeuse", 1.0) ;
```

Etat initial de l'objet laLacoste :

désignation	"Chemise"
prixHT	80.0
taille	1
coloris	rouge

19

En UML (Unified Modeling Langage) :



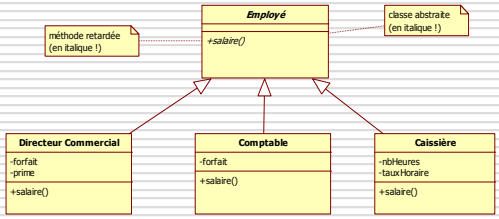
20

Définitions

Méthode retardée : méthode d'une classe dont l'implémentation est définie par les sous-classes

Classe abstraite : classe ayant au moins une méthode retardée

21



22

```
abstract class Employé
{
        …

        // pas de corps
        abstract public Float salaire () ;

        …

}
```

23

```
class DirecteurCommercial extends Employé
{
        private Float forfait ;
        private Float prime ;

        //  constructeur
        public DirecteurCommercial (Float f, Float p)
        {
                forfait = f ;
                prime = p ;
        }

        // détermine le salaire d'un directeur commercial
        public Float salaire ()
        {
                return (forfait + prime) ;
        }
}
```

24

```
class Comptable extends Employé
{
        private Float forfait ;

        // constructeur
        public Comptable (Float f)
        {
                forfait = f ;
        }

        // détermine le salaire d'un comptable
        public Float salaire ()
        {
                return (forfait) ;
        }
}
```

25

```
class Caissière extends Employé
{
        private Int nbHeures ;
        private Float tauxHoraire ;

        // constructeur
        public Caissière (Int n, Float t)
        {
                nbHeures = n ;
                tauxHoraire = t ;
        }

        // détermine le salaire d'une caissière
        public Float salaire ()
        {
                return (nbHeures * tauxHoraire) ;
        }
}
```

26

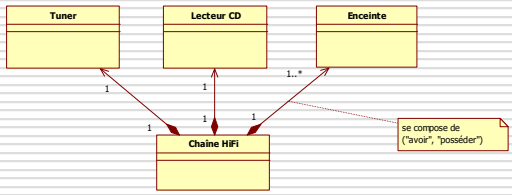
Définition

Composition : mécanisme permettant de définir un objet par assemblage d'autres objets

Relation client entre un objet composite et ses composants (principe d'encapsulation)

27

En UML :



28

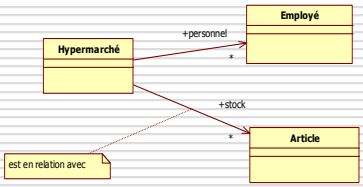
```
class ChaîneHiFi
{
        private Tuner  t ;
        private LecteurCD  l ;
        private Tableau<Enceinte>  e ;

        // constructeur : initialise les champs
        // d'une chaîne haute-fidélité
        public ChaîneHiFi (Int nb)
        {
                t = new Tuner () ;
                l = new LecteurCD () ;
                e = new Tableau<Enceinte> (nb) ;
        }

        // autres méthodes d'une chaîne haute-fidélité
        …
}
```

29

Quelques méthodes de la classe *Tableau* :

```
…

// fournit le nombre d'éléments du tableau
Int taille () ;

// accède à l'élément de rang i du tableau
T getIème (Int i) ;

// modifie l'élément de rang i du tableau par la valeur x
void setIème (Int i, T x) ;

…
```

30




```
class Hypermarché
{
        private Tableau<Article> stock ;
        private Liste<Employé> personnel ;

        // méthodes d'un hypermarché
        ...
}
```


Chapitre 3 : l'envoi de message et le polymorphisme

Définition

Envoi de message : requête adressée à un objet demandant l'exécution d'une méthode

Unique structure de contrôle entre objets

send (objetReceveur, sélecteur (paramètre$_1$, ..., paramètre$_n$))

ou

objetReceveur **.** sélecteur (paramètre$_1$, ..., paramètre$_n$)


```
Chemise laLacoste ;
laLacoste = new Chemise ("Chemise", 80.0, 1, rouge) ;

prix = laLacoste.prixDeVente() ;
```

→ prix = 96 € (TVA = 20 %)

```
EauGazeuse laBadoit ;
laBadoit = new EauGazeuse ("EauGazeuse", 1.0) ;

prix = laBadoit.prixDeVente() ;
```

→ prix = 1,05 € (TVA = 5,5 %)

Définition

Objet receveur : objet ayant en charge l'exécution d'un message

L'objet receveur courant : this

***this**.sélecteur (paramètre$_1$, ..., paramètre$_n$)*


```
class Article
{
        ...

        // calcule le prix de vente avec livraison d'un article
        public Float prixAvecLivraison ()
        {
                return (this.prixDeVente() + this.coûtLivraison()) ;
        }

        // autres méthodes
        ...
}
```



```
class Article
{
        private String désignation ;
        private Float prixHT ;

        // calcule le prix de vente d'un article
        public Float prixDeVente ()
        {
                return (this.prixHT * 1.2) ;
        }

        // calcule le coût de livraison d'un article
        public Float coûtLivraison ()
        {
                return (this.prixHT * 0.05) ;
        }

        ...
}
```


Définition

Superméthode : méthode d'une classe ancêtre masquée par héritage (substitution)

Accès : super

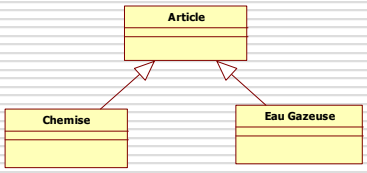
***super**.sélecteur (paramètre$_1$, ..., paramètre$_n$)*


```
class ChaîneHiFi extends Article
{

        private Float coûtGarantie ;

        // calcule le prix de vente d'une chaîne haute-fidélité
        public Float prixDeVente ()
        {
                return (super.prixDeVente() + this.coûtGarantie) ;
        }

        // autres méthodes
        ...
}
```


Définition

Polymorphisme : capacité pour une entité à prendre plusieurs formes à l'exécution

En POO, polymorphisme lié à l'héritage et à la redéfinition de méthode




```
Article a ;
Chemise c ;
c = new Chemise ("Chemise", 80.0, 1, rouge) ;
```

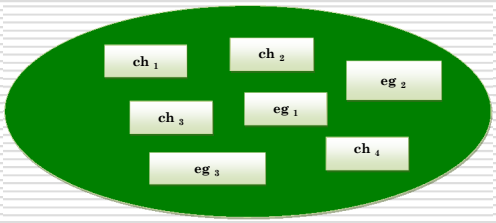
Ecritures valides :

```
a = c ;
a = new EauGazeuse ("EauGazeuse", 1.0) ;
```

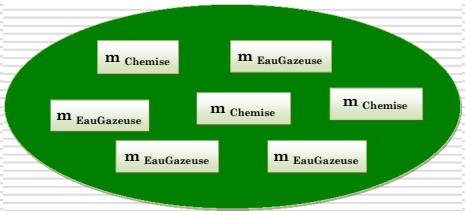
Ecriture en général invalide (affectation inverse) :

```
c = a ;
```


Traiter une famille d'objets en ignorant
pour chacun sa classe d'appartenance
(ici des chemises *ch* et des eaux gazeuses *eg*)



Appliquer une même méthode à des objets
en ignorant pour chacun sa classe d'appartenance
(ici la méthode *m* des chemises et des eaux gazeuses)



Dans *Article* :

```
// calcule le prix de vente d'un article
public Float prixDeVente ()
{
        return (this.prixHT * 1.2) ;                    // TVA = 20 %
}
```

Dans *EauGazeuse* :

```
// calcule le prix de vente d'une eau gazeuse
public Float prixDeVente ()
{
        return (this.getPrixHT () * 1.055) ;            // TVA = 5,5 %
}
```

Dans *Supermarché* :

```
prix = this.stock.getIème(i).prixDeVente() ;
```


Dans *Supermarché* :

```
// détermine la masse salariale mensuelle
public Float masseSalariale ()
{
        Float s = 0.0;
        for (Int i = 0 ; i < this.personnel.taille() ; i++)
                s = s + this.personnel.getIème(i).salaire() ;
        return (s) ;
}
```