Programmation Objet

- Introduction
- Les fondements de la programmation Objet
- La communication entre les Objets
- Les différents liens entre les classes
- Conclusion

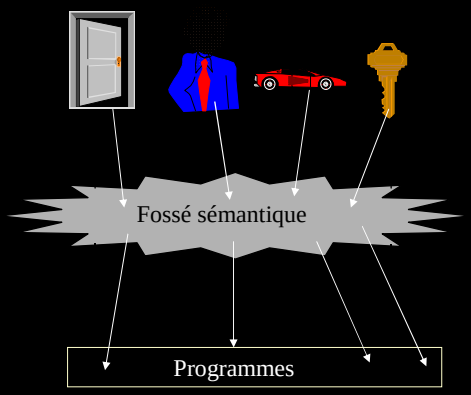
Introduction

- Pourquoi la Programmation Objet
- Historique des langages Objets

Pourquoi la POO

En programmation procédurale le fossé est grand entre le monde réel, et sa représentation informatique.

- Difficultés de compréhension
- Difficultés de maintenance
- Difficultés de réutilisation



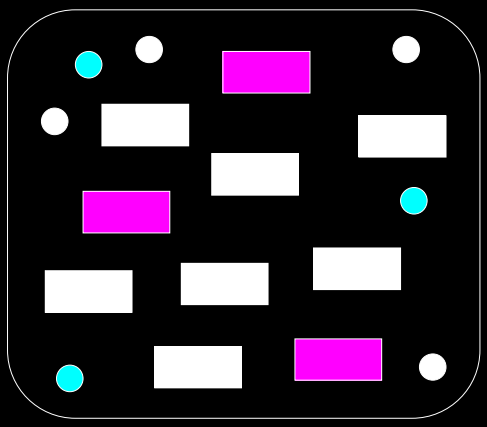
Pourquoi la POO

Dans un programme les données représentant une entité peuvent être disséminées dans le programme, et les traitements aussi.

Compréhension difficile

Réutilisation difficile

Maintenance difficile



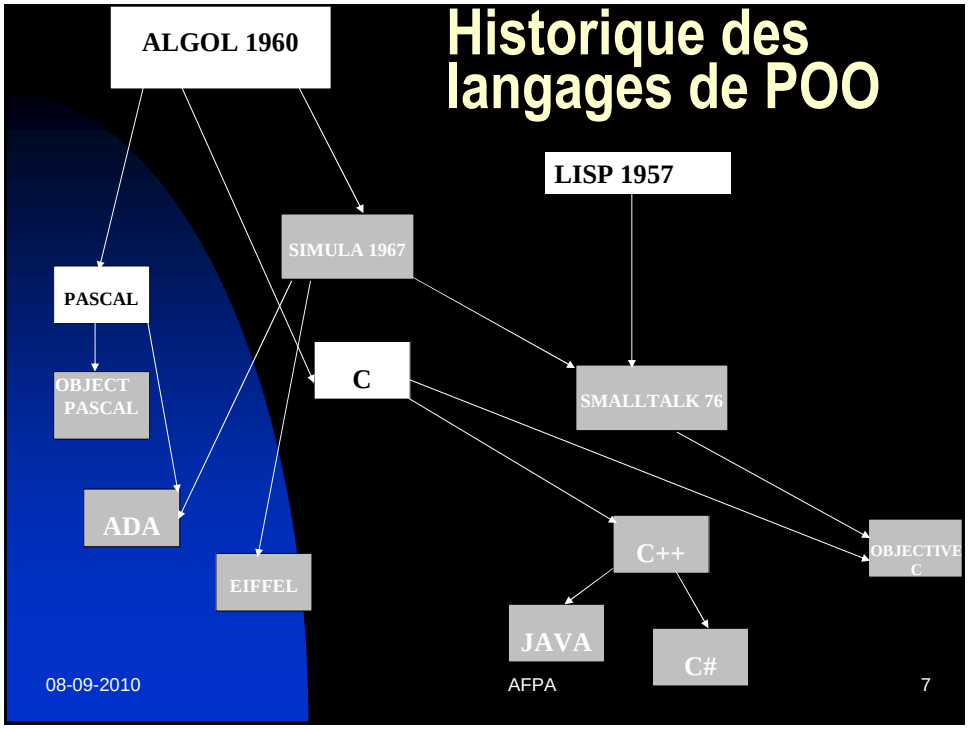
Pourquoi la POO

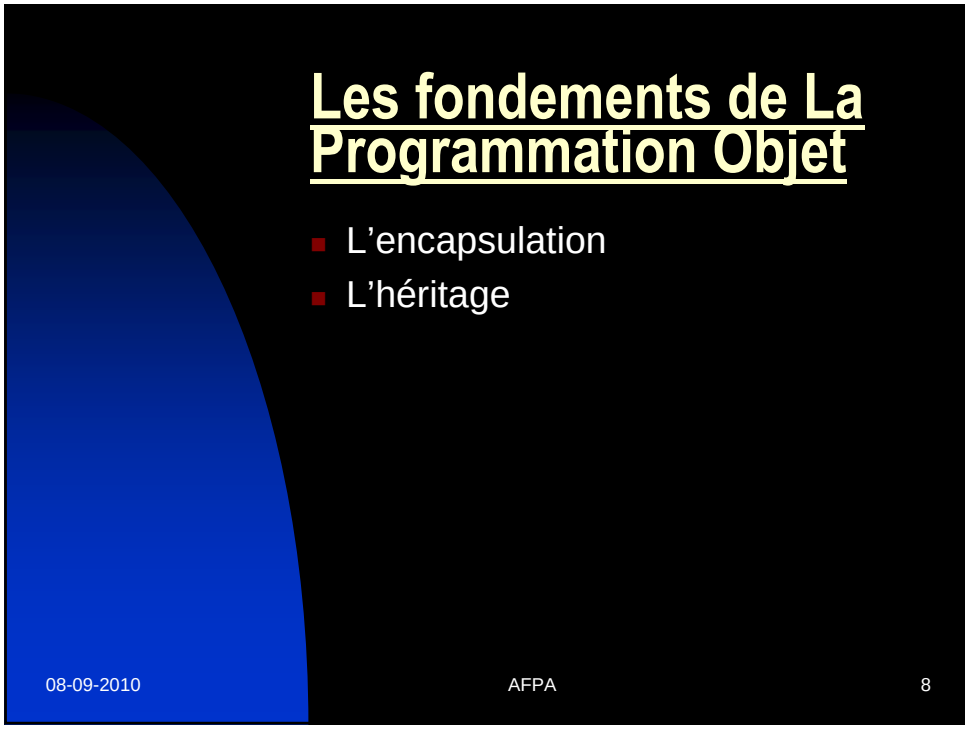
- Les logiciels deviennent de plus en plus gros. Il est difficile de maîtriser la complexité
- Couplage fort entre les modules qui entraîne une fragilité des logiciel

➔LA REUTILISATION DU CODE

Paradigme objet

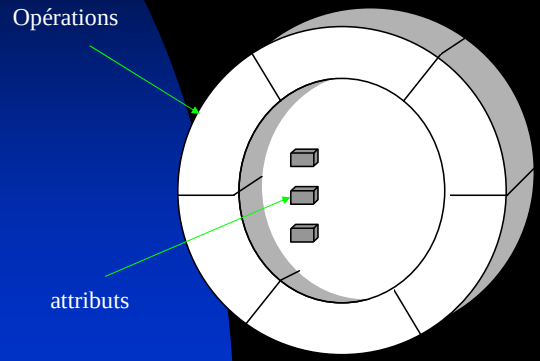
- Un ensemble indissociable données traitements
- Il représente souvent une entité du monde réel (objet métier)
- Un objet est un module
- Un objet est responsable de lui même
- Un objet offre des services


Historique des langages de POO
ALGOL 1960
LISP 1957
SIMULA 1967
PASCAL
C
OBJECT PASCAL
SMALLTALK 76
ADA
C++
OBJECTIVE C
EIFFEL
JAVA
C#
08-09-2010
AFPA
7


Les fondements de La Programmation Objet
L'encapsulation
L'héritage
08-09-2010
AFPA
8

Encapsulation: l'objet

- Un objet est défini par une identité, un état et un comportement



Encapsulation:exemples

Ma voiture

Attributs:

Couleur = bleue

Poids = 979 kg

Puissance = 12 CV

Capacité réservoir = 50 l

Conducteur = Dupont

Vitesse instantanée = 50 Km/h

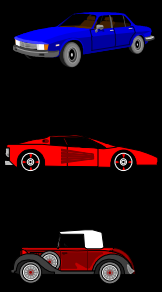
Opérations:

Démarrer()

Se déplacer()

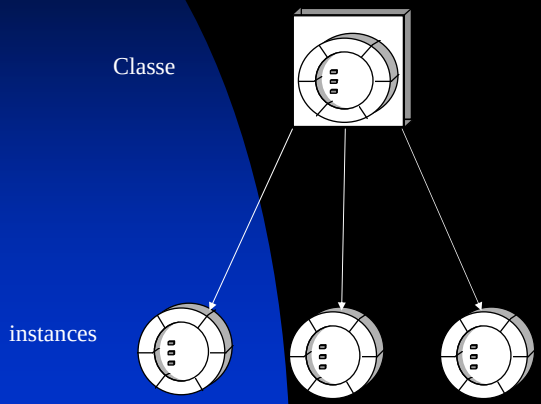
Mettre de l'essence()

Encapsulation: classe



- Une classe est la factorisation des caractères communs à une famille d'objets.
- Une classe sert de moule à objet. A partir d'une classe nous créons des objets

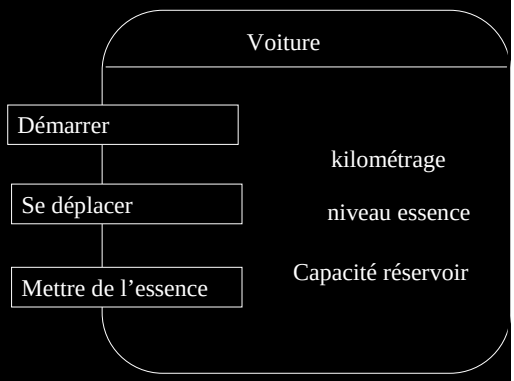
Encapsulation: instanciation



- L'instanciation est le processus par lequel nous créons de nouveaux objets à partir du modèle fourni par une classe

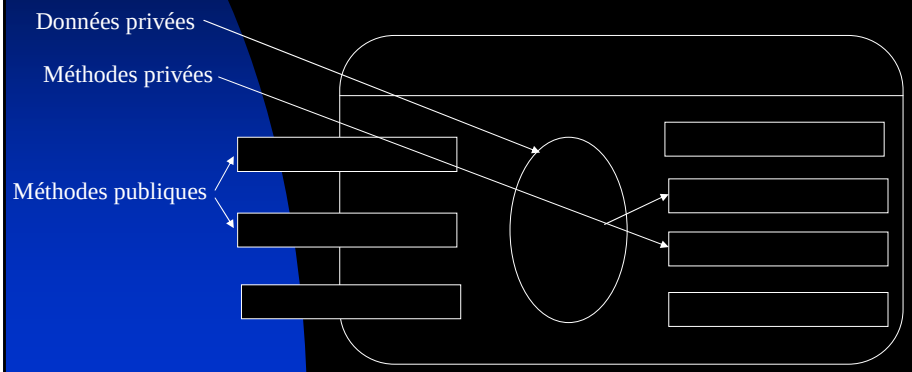
Encapsulation

- Un objet n'est connu que pour les services qu'il peut rendre (contrat, interface, protocole)
- Un objet est responsable, il assure sa propre sécurité.



Encapsulation

- Les attributs sont privés
- Les méthodes de l'interface sont publiques
- Les autres méthodes sont privées



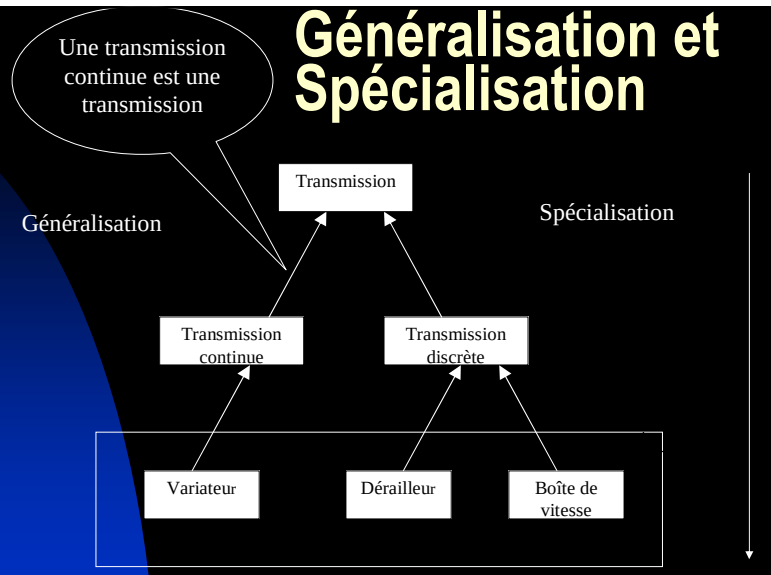
Héritage

- La classification
- Généralisation et spécialisation
- Classes abstraites
- Héritage comment ça marche
- Un exemple
- Polymorphisme
- Surcharge

La classification

- Un soucis de l'homme depuis la nuit des temps (taxonomie)
- Comparer, et, des différences et des ressemblances, en déduire l'évolution
- Ici classifier pour conserver la mémoire de l'évolution.
- Les objets restent, mais leur comportement évolue.

Généralisation et Spécialisation



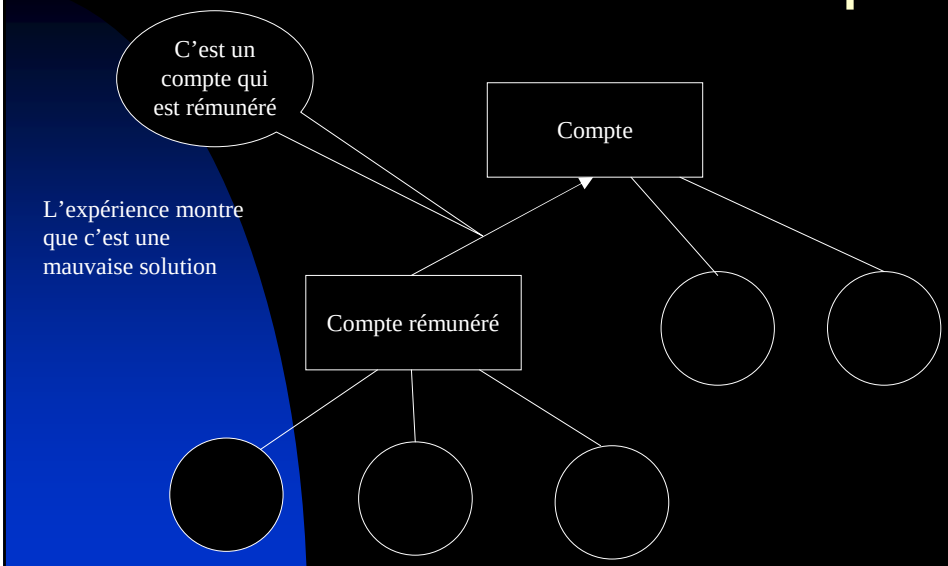
Classe abstraite

- C'est une classe qui a été créée pour construire une classification
- C'est une classe qui représente un concept général
- Cette classe n'est pas un moule à objet
- Elle ne peut pas décrire tous les comportements de ses sous classes

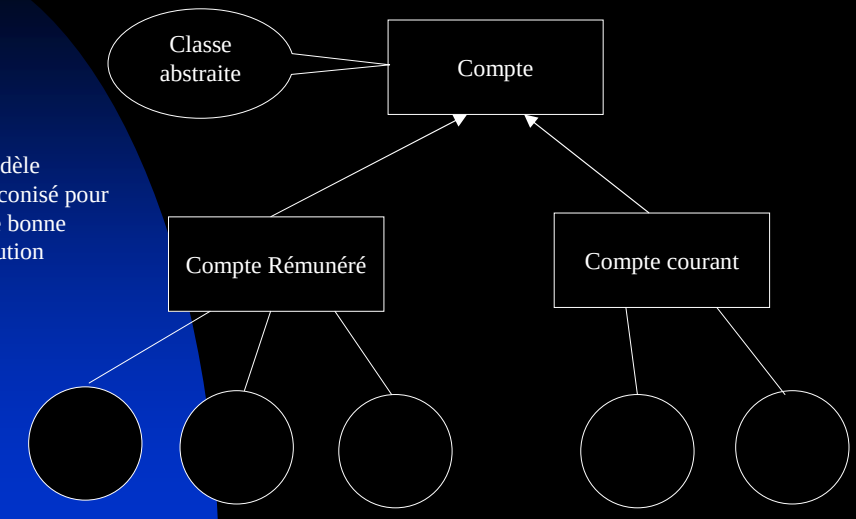
Classe abstraite

- Par exemple une transmission discrète est une classe abstraite
  (cf l'arborescence précédente)
- Cela représente bien un concept général
- Je ne peux pas demander à un commerçant une transmission discrète
- Je ne peux pas décrire de manière précise comment se passe un changement de vitesse. (méthode abstraite).

Classe abstraite: exemple



Classe abstraite: exemple

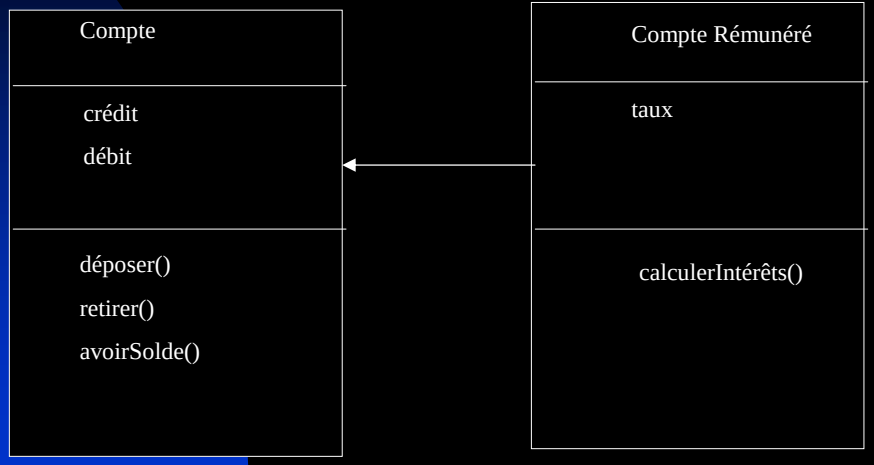


Modèle préconisé pour une bonne solution

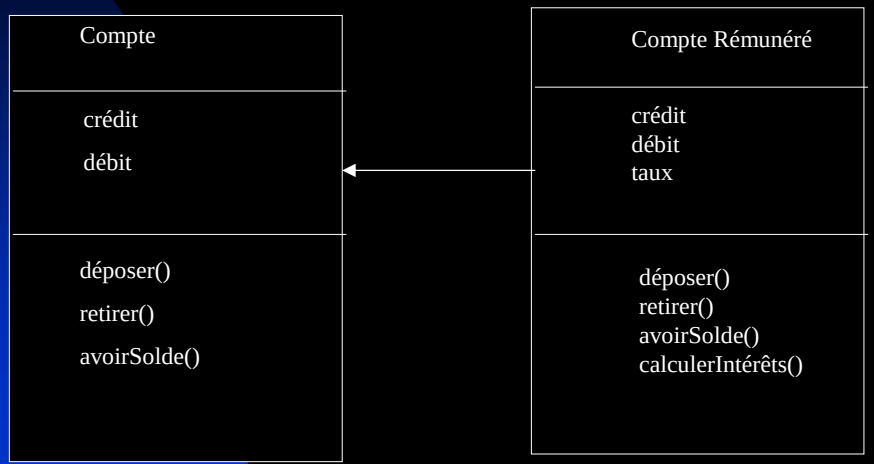
Classe abstraite: règle

- En règle générale:
- Une classe terminale sera concrète (c'est une classe à objet)
- Une classe non terminale sera une classe abstraite.

Héritage: ce qui est écrit

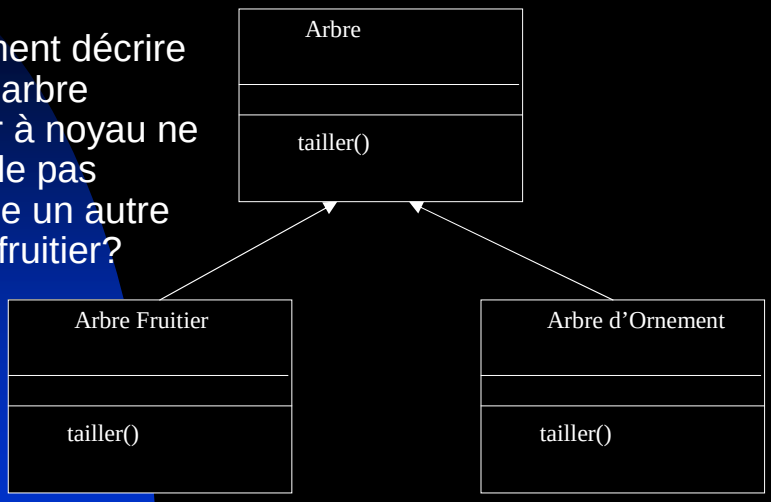


Héritage: ce qu'il faut lire



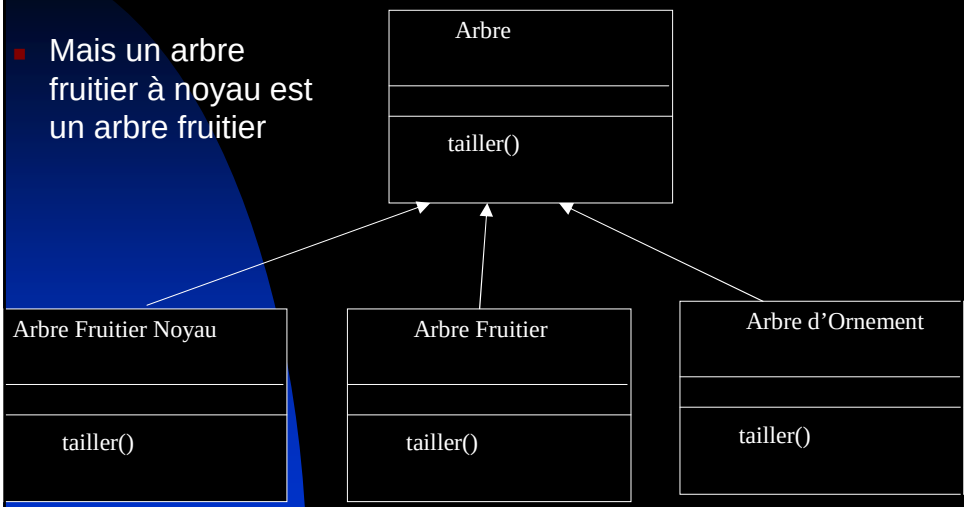
Structuration d'héritage

- Comment décrire qu'un arbre fruitier à noyau ne se taille pas comme un autre arbre fruitier?



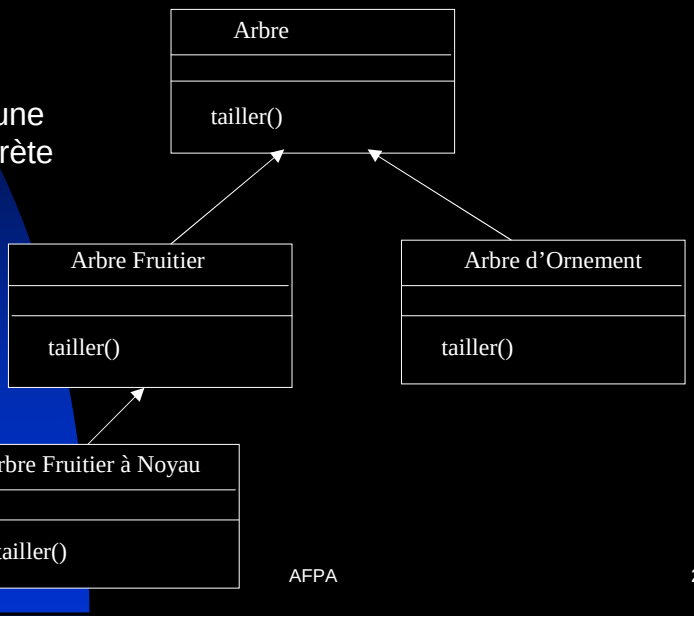
Structuration d'héritage

- Mais un arbre fruitier à noyau est un arbre fruitier

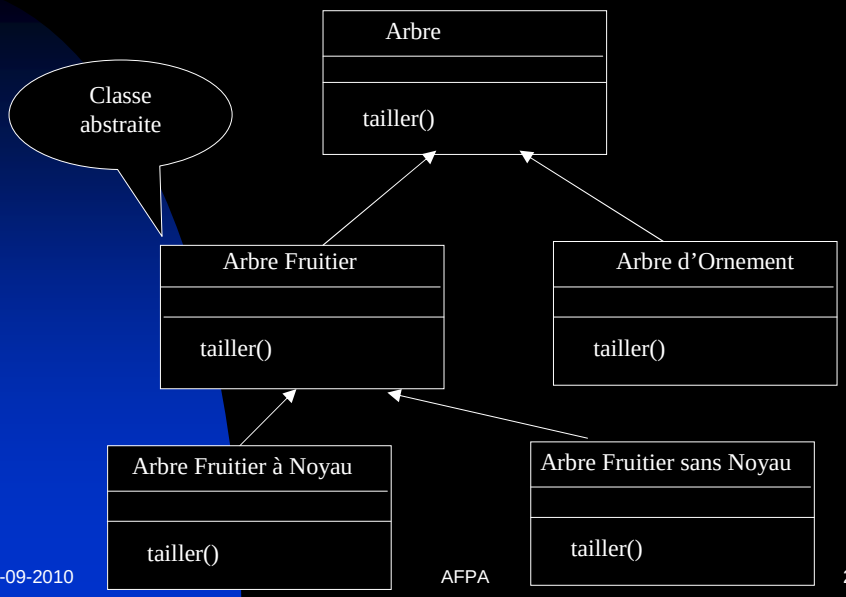


Structuration d'héritage

- Mais Arbre Fruitier est une classe concrète



Structuration d'héritage



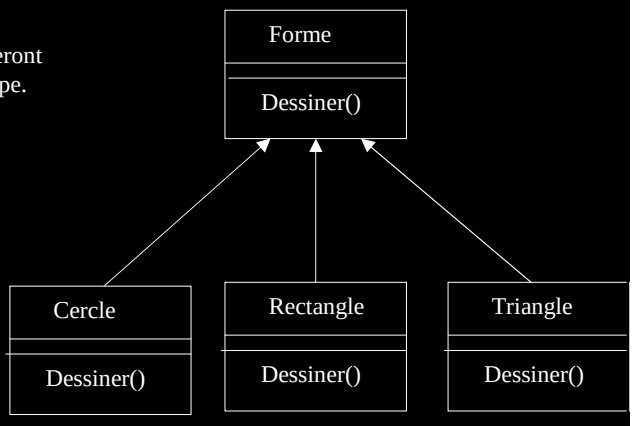
Polymorphisme

- Le polymorphisme caractérise un comportement qui peut prendre plusieurs formes.

Polymorphisme: exemple

- Création d'un logiciel de dessin

Toutes les formes savent se dessiner, mais elle se dessineront différemment suivant leur type.



Polymorphisme: exemple

- Programmation classique

```
dessin: tableau[10] Forme

for (i = 0;i< max; i++)
{
        si dessin[i] est un cercle alors
                dessinerCercle(tab[i])
        si dessin[i] est un rectangle alors
                dessinerRectangle(tab[i])
        si dessin[i] est un triangle alors
                dessinerTriangle(tab[i])
}
```


Polymorphisme: exemple

- Programmation Objet

```
dessin: tableau[10] Forme

for (i = 0;i< max; i++)
{
        tab[i].dessiner()
}
```

Demande à la forme tab[i] de se dessiner

Lire: tab[i] dessine toi

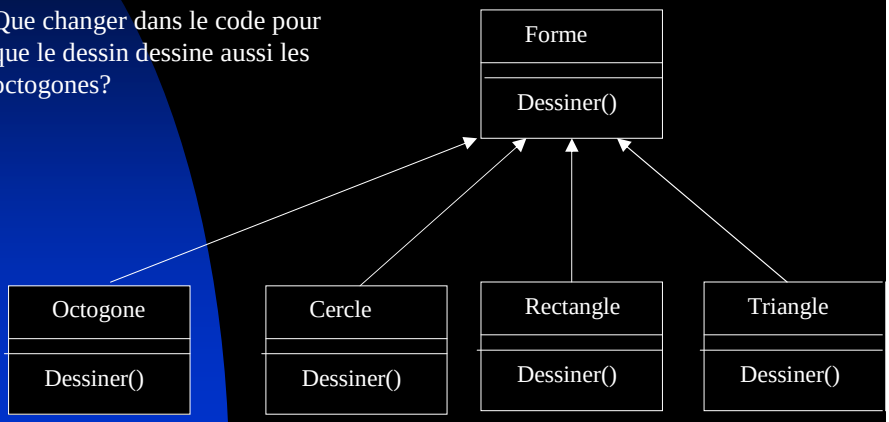
Nous ne savons pas de quelle méthode dessiner il s'agit ici, au moment ou nous écrivons le programme

Nous ne le saurons qu'à l'exécution, grâce au lien dynamique

Polymorphisme: exemple

- Rajout d'une nouvelle Forme: Octogone

Que changer dans le code pour que le dessin dessine aussi les octogones?



Surcharge de méthode

- C'est la ré-écriture d'une méthode avec des paramètres différents.
- Il arrive que l'on puisse faire le même traitement avec des informations différentes. Nous surchargerons alors la méthode : même nom de méthode, mais paramètres différents (signature)

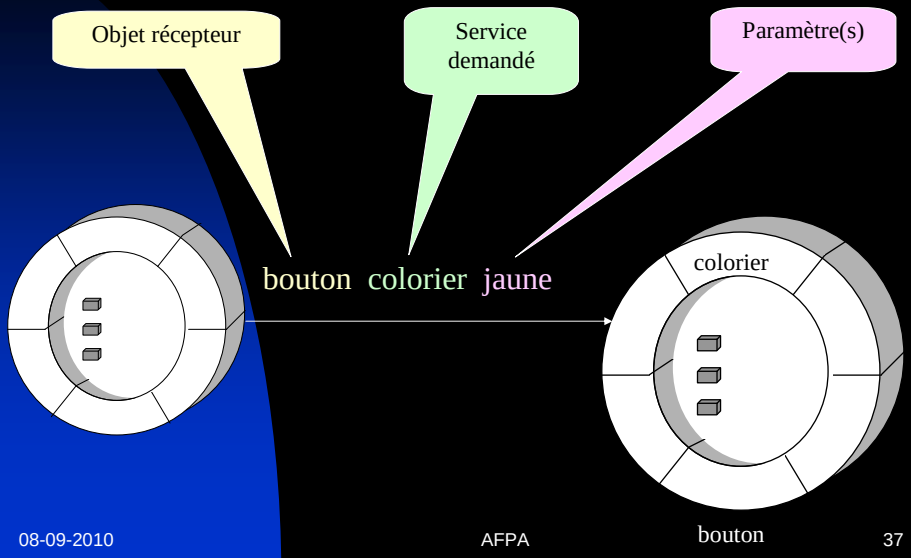
Exemple surcharge

- CréerPermis(CarteIdentité ci; FactureEDF fe)
- CéerPermis(Passeport p; AttestationDomicile ad)
- Voilà une surcharge de la méthode CréerPermis.
- Les surcharges doivent travailler pour le même résultat.

La communication entre les Objets

- L'envoi de message
- Les méthodes
- Les types de messages
- Recherche dynamique des méthodes

L'envoi de message



Les méthodes

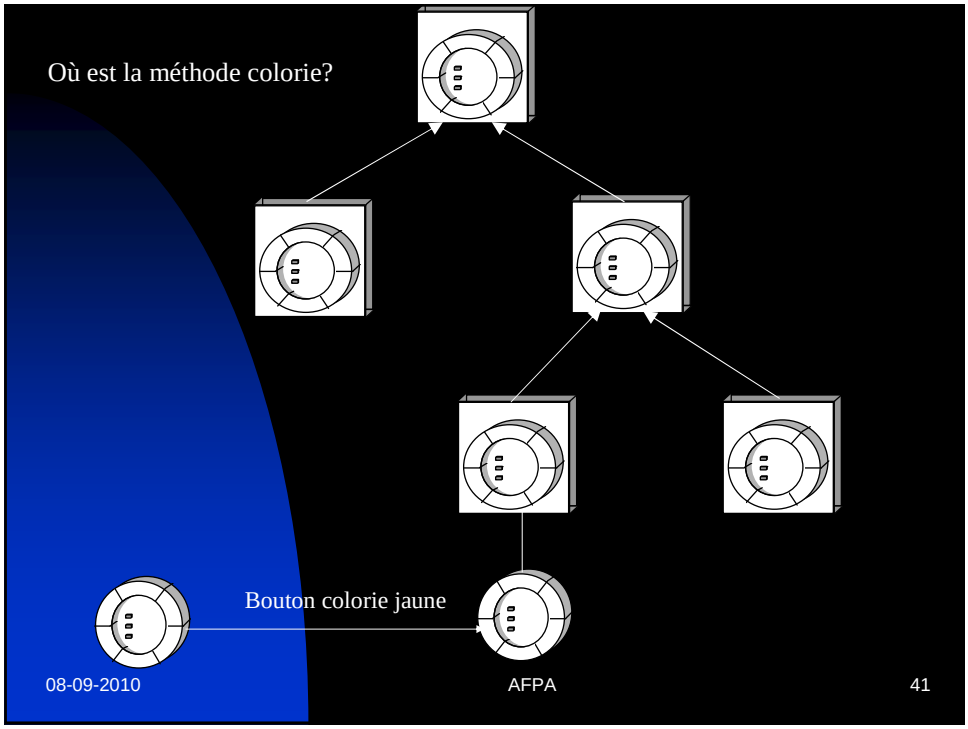
- Les méthodes sont des fonctions internes à un objet.
- Elles sont décrites au niveau des classes
- Elles sont appelées quand l'objet reçoit un message de demande de service.
- Elles sont différenciées par leur nom, sauf pour les surcharges, alors elles doivent êtres discriminées par les types des paramètres.

Les types de messages

- Les constructeurs
- Les destructeurs
- Les accesseurs
- Les mutateurs
- ,…

Recherche dynamique des méthodes

- Quand un objet reçoit un message, comment savoir à l'exécution quelle est la méthode appelée?
- La méthode peut être définie:
  - Dans la classe de l'objet
  - Dans une classe mère ou aïeule
  - De plus elle peut avoir muté (polymorphisme)
- Mais l'objet sait toujours quel est sa classe

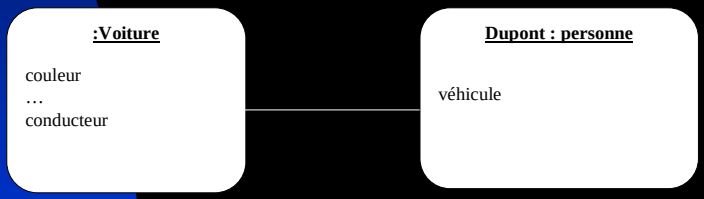


Les différentes relations entre les classes

- L'héritage
- L'association
- L'agrégation
- La composition

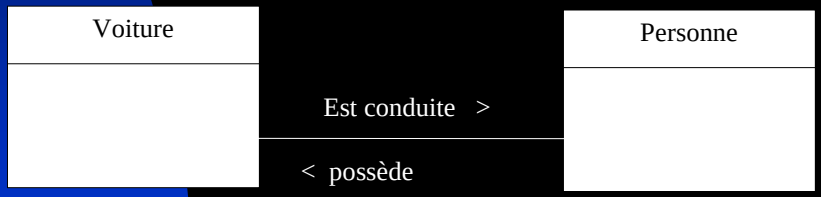
L'association

- Au niveau des objets, certains attributs représentent des liens entre des objets



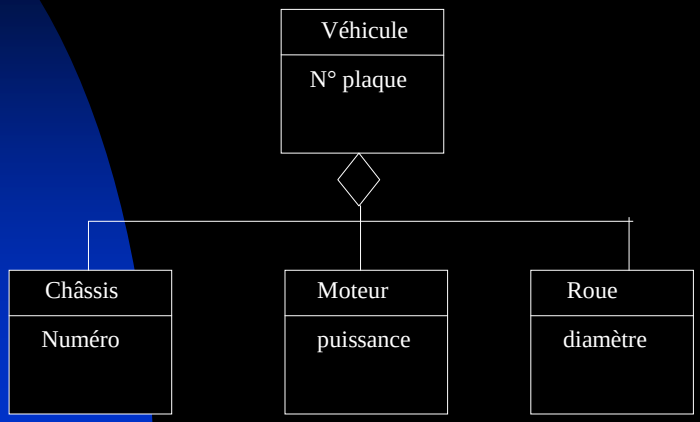
L'association

- Au niveau des classes, l'association exprime une connexion sémantique entre les classes



L'agrégation

- C'est une relation composant composé

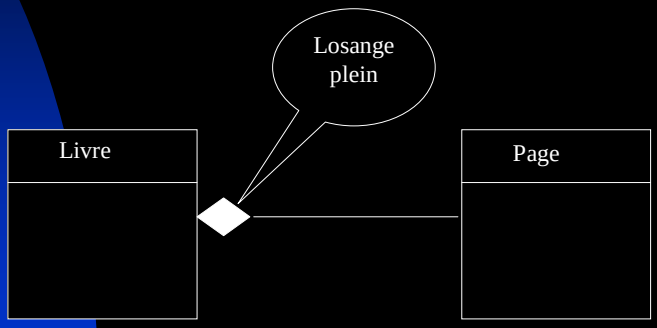


L'agrégation

- Un objet de la classe composée ne peut vivre sans la totalité de ses composants (donc l'association ici n'est pas ponctuelle).
- Un composant peut survivre à l'objet composé: La mort du composé n'entraîne pas forcément la mort de tout ses composants.

La composition forte

- C’est une relation composant composé



La composition forte

- C’est une relation de composition forte, quand le livre est détruit, ses pages aussi sont détruites.
- La mort de l’objet entraîne la mort de ses composants.

Conclusions

- Un grand pas en avant
- Les mots clef

Les mots clef

- Objet, classe, instance
- L'encapsulation
- L'héritage, et le polymorphisme
- La surcharge
- Association, agrégation, composition.


FIN
08-09-2010
AFPA
51