POO
– Concepts de base de la modélisation objet –

Christophe BLANC

– BUT MMI1 –
IUT Clermont Auvergne
Université Clermont Auvergne

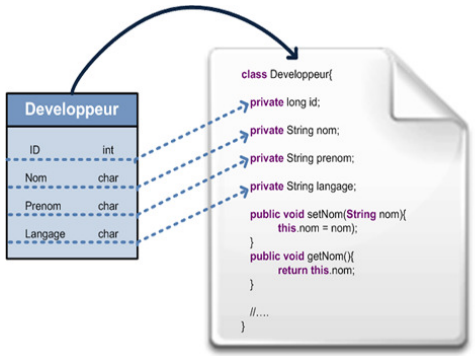
Février 2024

Plan

1. L'anatomie d'un objet.
2. La visibilité et l'encapsulation.
3. L'envoi de messages entre les objets.

Objet :

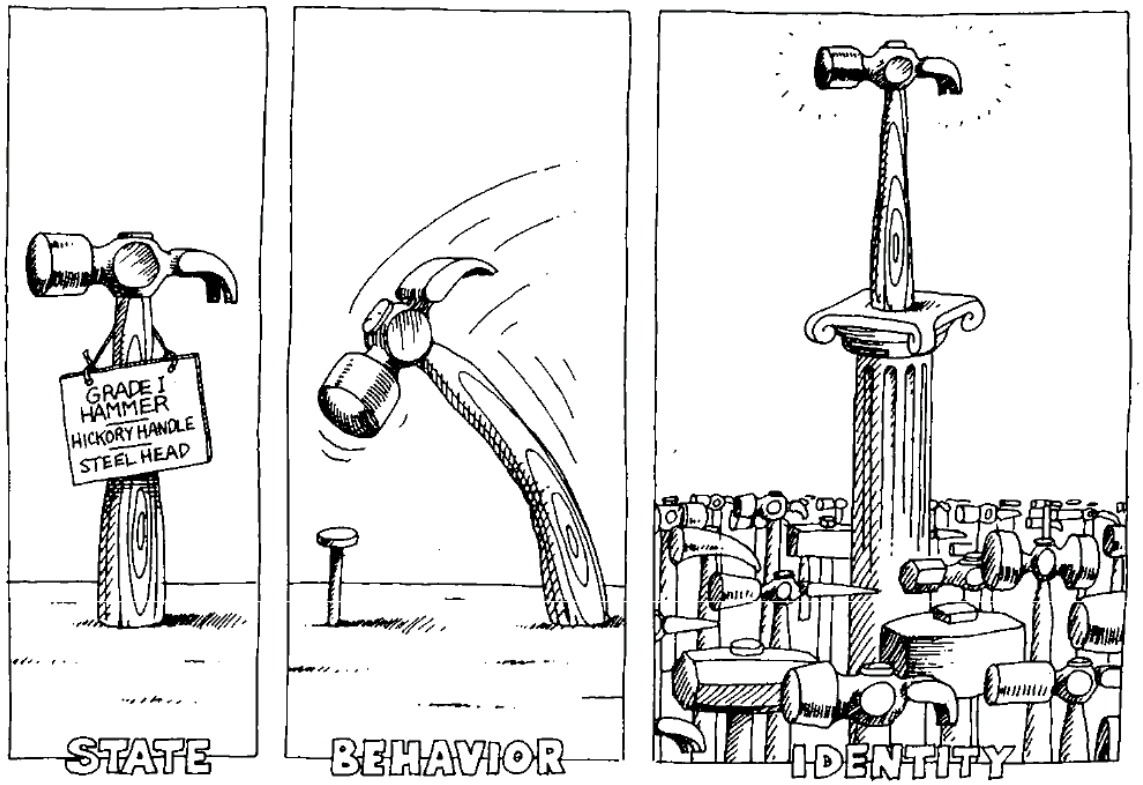
- Un objet est une entité manipulable par un ordinateur. Il correspond à une réalité ou une abstraction.



Les 3 composantes d'un objet

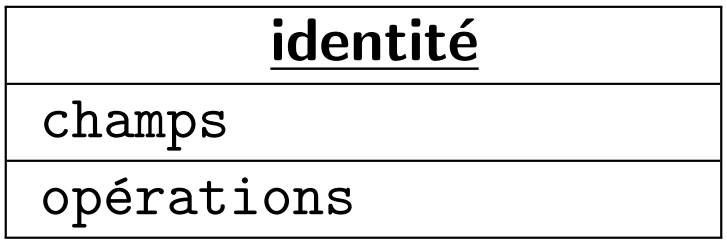
- Un objet comprend trois composantes :
  - une identité ;
  - une composante statique correspondant à une ou plusieurs structures de données utilisées pour stocker ses caractéristiques (état) : les champs ;
  - une composante dynamique correspondant à un ensemble d'opérations qui lui sont particulières et qui, portant sur ces données, mettent l'objet dans un certain état : les opérations.

L'objet selon Grady BOOCH



La représentation graphique d'un objet

- Dans des documents de conception, cet objet peut être représenté graphiquement par un rectangle à coins droits (UML).
- Ce rectangle est scindé en 3 parties pour inscrire l'identité, les champs et les opérations.



L'identité d'un objet en UML

- Dans ce dernier cas, l'identité peut être écrite de trois manières (mais souligné dans les 3 cas de figure) :
  - nom (l'objet a un nom mais on ne précise pas le type)
  - :type (l'objet a un type mais on ne précise pas le nom)
  - nom :type (identité complète)

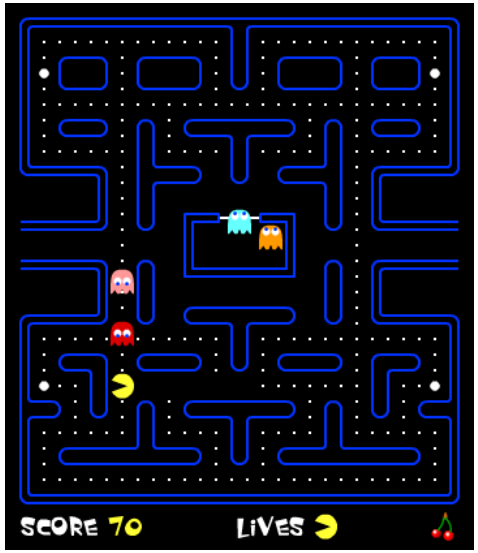
L'exemple de Pacman

- Par exemple, Pacman est un acteur (un objet) qui possède comme champs une couleur, un nombre de points de vie et une position. Cet acteur est capable de se déplacer, de manger des pac-gommes, manger des pastilles et de se faire croquer par un fantôme.

pacman

`couleur`
`nombre de points de vie`
`position`

`se déplacer`
`manger des pac-gommes`
`manger des pastilles`
`se faire croquer`



La notion de visibilité

- Les éléments statiques et dynamiques d'un objet peuvent être cachés ou non aux autres objets.
- Cette caractéristique est appelée la visibilité, elle permet au programmeur de contrôler les éléments qu'il peut rendre :
  - publiques ;
  - privés.

Le cas des champs

- Si un champ est qualifié de publique (nous ajoutons un + devant le nom) : les objets externes peuvent manipuler directement cette donnée.
- Si un champ est qualifié de privé (nous ajoutons un – devant le nom), seul l'objet peut manipuler la donnée.
- Par défaut, un champ est considéré comme privé : le concept d'encapsulation consistant à cacher les données à l'intérieur des objets, il n'est pas conseillé d'utiliser des champs publiques.

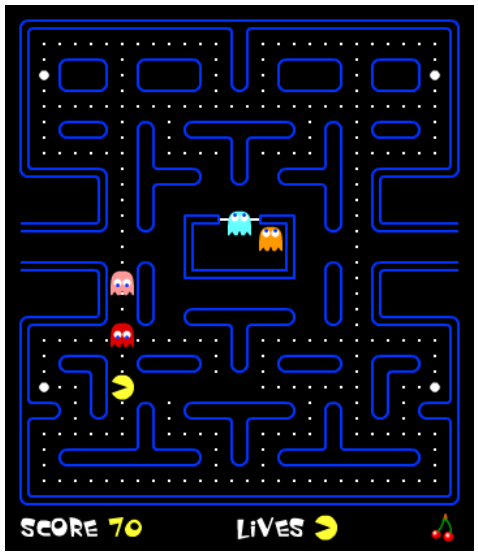
Le cas des méthodes

- Si une opération est qualifiée de publique, les objets externes peuvent l'utiliser pour modifier l'état d'un objet.
- Si une opération est qualifiée de privée, elle n'est utilisé que par l'objet lui-même.
- Contrairement aux attributs, les opérations sont considérées par défaut comme publiques, le programmeur peut cependant ajouter des opérations privées qui serviront pour le traitement interne de l'objet.

L'exemple de Pacman

- Si nous reprenons l'exemple de Pacman, nous pouvons ajouter des opérations privées pour découper certains traitements complexes. Ainsi l'opération « se faire croquer » peut faire appel en interne à trois opérations privées « se détruire », « décrémenter le niveau de vie » et « se régénérer ».

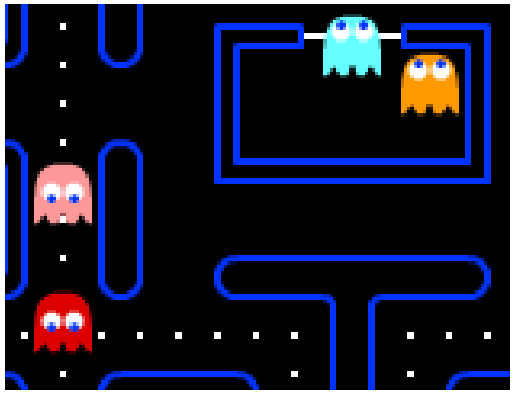
pacman
couleur nombre de points de vie position
se déplacer manger des pac-gommes manger des pastilles se faire croquer - se détruire - décrémenter le niveau de vie - se régénérer



L'exemple des fantômes

- Un fantôme est un autre acteur qui peut modélisé par un objet ayant comme champs une couleur, une position et un état (peureux ou non) et des opérations publiques (se déplacer, se faire croquer) et privées (se détruire, se régénérer).
- Dans le jeu, nous avons 4 fantômes de couleurs différentes que nous modélisations par 4 objets.

L'exemple des fantômes



fantôme bleu
couleur = bleu position état
se déplacer se faire croquer - se détruire - se régénérer

fantôme orange
couleur = orange position état
se déplacer se faire croquer - se détruire - se régénérer

fantôme rose
couleur = rose position état
se déplacer se faire croquer - se détruire - se régénérer

fantôme rouge
couleur = rouge position état
se déplacer se faire croquer - se détruire - se régénérer

La notion d'interface et l'envoi de messages

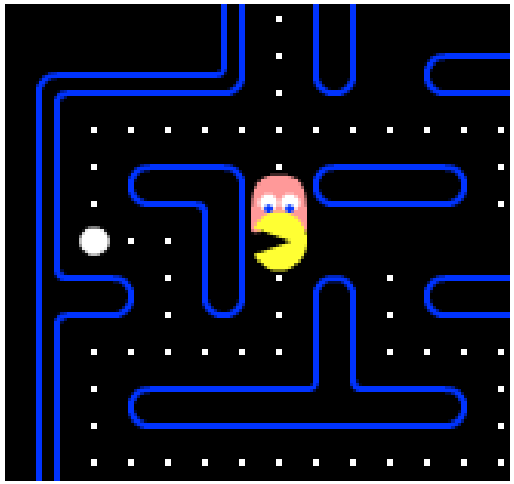
- Le seul accès à un objet se fait au travers de son interface, qui est défini par l'ensemble des opérations publiques (les opérations qui sont visibles de l'extérieur).
- Un objet A modifie l'état d'un objet B en invoquant une des opérations de B. D'un point de vue conceptuel, cette invocation se traduit par un envoi de message de A vers B.

L'anatomie d'un message

- Le message comprend les éléments suivants :
  - l'identité de son destinataire ;
  - l'identification de l'opération qu'il demande à ce destinataire d'activer ;
  - les informations nécessaires à cette activation.

Cas d'une communication entre Pacman et un fantôme

- Par exemple, lorsque le fantôme rose croque Pacman, il lui envoi un message « Se faire croquer » afin de lui demander de déclencher l'animation de sa destruction et de décrémenter son niveau de vie.



Plan

1. La classe vue comme une abstraction des objets.
2. Les attributs et les méthodes de classe.
3. L'objet vu comme une instance d'une classe.

La notion de classes

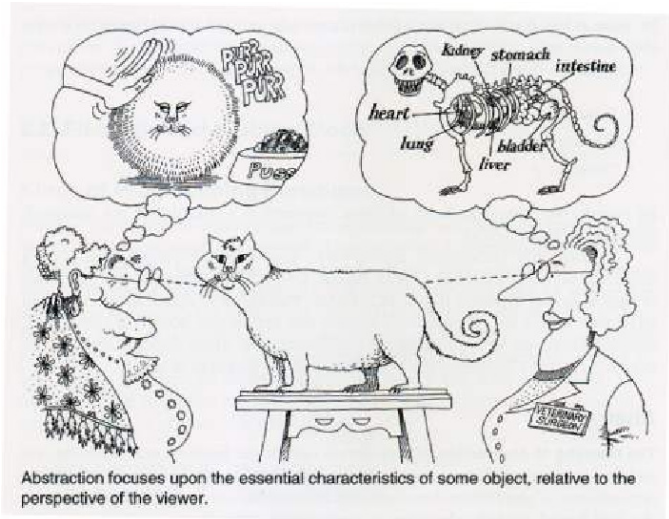
- Jusqu'à présent pour construire des objets « semblables », nous devons dupliquer le code, ce que peut présenter divers inconvénients comme le gaspillage de la place mémoire ou la difficulté à maintenir le code. Simula a introduit la notion de classes qui permet de pallier à ce problème.
- Une classe est une abstraction correspondant à un ensemble d'objets ayant les mêmes attributs et les même méthodes dans le monde réel : cela permet de factoriser le codage des opérations.

L'instanciation

- Une classe peut être vue comme un gabarit permettant de mouler des objets ayant ces caractéristiques. Cette opération est appelée l'instanciation et l'objet est alors considéré comme une instance (un représentant) de la classe.
- Les classes sont définies à la compilation alors que leurs représentants sont créés à l'exécution.

La perception d'une classe

- Le chat peut être perçu de deux manières différentes selon le point de vue :
  - Mamie verra le chat (l'abstraction) comme un animal de compagnie (l'objet Minou) ;
  - la vétérinaire verra le chat comme un animal doté d'un certain nombre d'organes internes.



Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

Attribut et champ, méthode et opération

- En tant que gabarit, une classe doit indiquer :
  - Les attributs auxquels correspondent les champs dans les objets qu'elle permet de mouler. Un attribut n'a pas de valeur (ces valeurs sont attribuées aux variables de champs dans les objets) mais il possède des propriétés comme son nom, le type de variables de champs qui lui sont associés ...
  - Les méthodes auxquelles correspondent les opérations dans les objets qu'elle permet d'instancier.

Les deux types de méthodes

- Il existe deux types de méthodes :
  - les méthodes de classes qui peuvent être invoquées sur la classe (par exemple, les méthodes qui permettent d'instancier les objets) ;
  - les méthodes de représentant (instance méthods) seront utilisées par les objets pour accéder en lecture ou en écriture aux champs des objets.

Le principe de l'instanciation

- D'un point de vue graphique, une classe se représente (comme pour l'objet) par un rectangle au coins droits, scindé en 3 parties pour inscrire le nom de la classe, la liste des attributs et la liste des méthodes.
- La différence est que le nom n'est pas souligné.

Une Classe

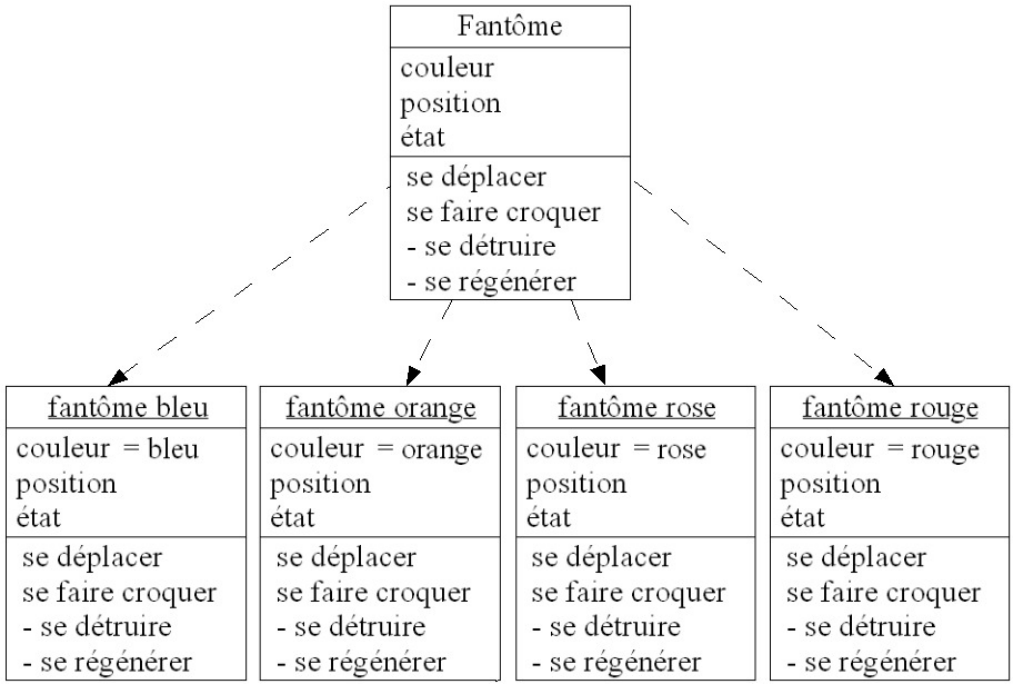
NomDeLaClasse
attributs
méthodes

Une instance de la classe est un objet

identité:NomDeLaClasse
champs
opérations

Le cas des fantômes

- Si nous considérons l'exemple des fantômes, nous avons alors une classe Fantôme qui permet d'instancier les quatre fantômes du jeux.

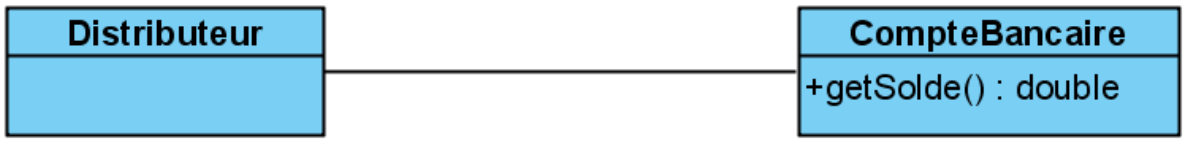


Plan

1. L'association.
2. L'agrégation.
3. La composition.
4. L'héritage.
5. Notions de surcharge et de redéfinition.
6. L'héritage multiple.
7. Les classes abstraites et le polymorphisme

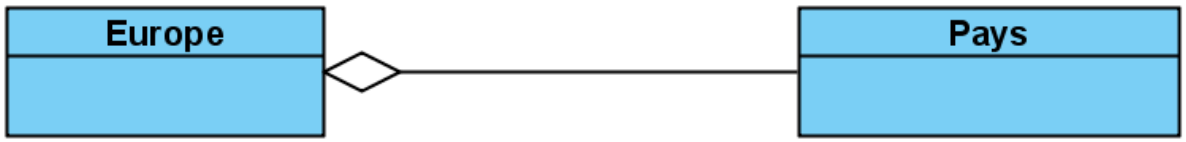
Le principe

- L'association indique un lien qui peut se créer au cours du fonctionnement lorsqu'un objet d'une classe envoie un message vers un objet d'une autre classe. D'un point de vue graphique, nous modélisons cette relation par un simple trait entre les deux classes.



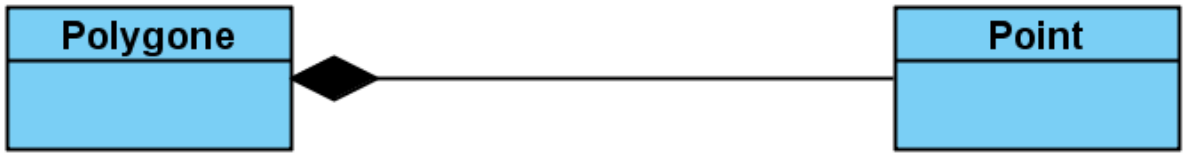
Le principe

- L'agrégation est une relation plus forte que l'association. Elle permet de modéliser une relation de possession de type « fait partie de ». Les classes sont donc agrégées dans une classe qui joue un rôle de conteneur.
- D'un point de vue graphique, cette relation est modélisée par un trait terminé d'un losange blanc du côté du conteneur.



Le principe

- La composition est une relation encore plus forte que l'agrégation. Elle permet d'indiquer que les classes qui compose la classe ≪ ensemble ≫ ne peuvent pas exister sans la classe ≪ ensemble ≫ : elles sont créées par cette classe et détruites par elle.
- D'un point de vue graphique, le losange de la composition est noir et est placé du côté de la classe ≪ ensemble ≫.



Le principe

- La relation d'héritage peut être vue de deux manières :
  - Elle permet de factoriser des attributs et des méthodes d'un même ensemble de classes afin de faciliter la déclaration et la maintenance du code.
  - Elle permet de créer des classes spécialisées à partir d'une classe de base.

Le principe et la représentation graphique

- Si nous considérons une classe A avec un attribut a_a et une méthode m_a et si nous créons une classe B (avec une méthode a_b et une méthode m_b) qui hérite de A, la classe pourra instancier des objets qui auront des champs a_a et a_b et des méthodes m_a et m_b (les attributs et les méthodes de la classe A sont connus de la classe B, mais la classe A ne connaît pas les méthodes et les attributs de la classe B.
- D'un point de vue graphique, cette relation est notée par une flèche qui pointe vers la classe mère.

L'héritage dans Pacman

- Si nous reprenons l'exemple du jeu Pacman, nous remarquons les faits suivants :
  - l'objet pacman est issu d'une classe Pacman (généralement, le nom des classes commence par une majuscule alors que le nom des objets commence par une minuscule) ;
  - Les classes Pacman et Fantômes ont des éléments en communs qu'il conviendrait de factoriser dans une classe Personnage.

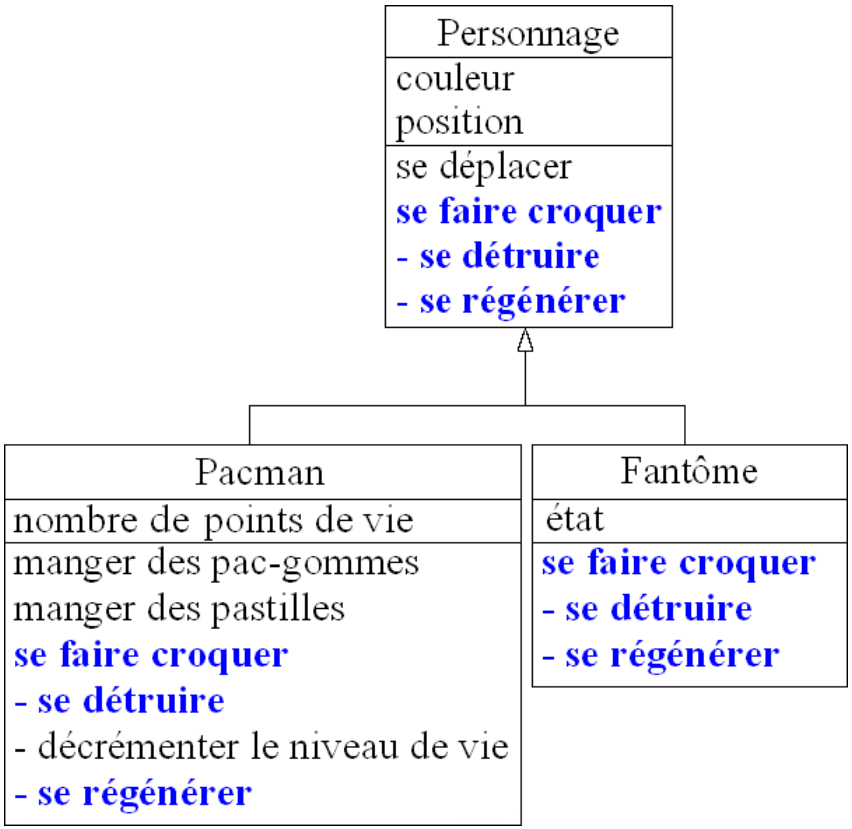
Mise en évidence des éléments communs

Pacman	Fantôme
couleur nombre de points de vie **position**	**couleur** **position** état
se déplacer manger des pac-gommes manger des pastilles **se faire croquer** **- se détruire** - décrémenter le niveau de vie **- se régénérer**	**se déplacer** **se faire croquer** **- se détruire** **- se régénérer**

La construction de la classe Personnage

- Tout n'est pas factorisable :
  - Seuls les éléments rouges sont placés au niveau de la classe Personnage car ils seront repris sans modification par les classes filles.
  - Les éléments en bleu sont placés dans les 3 classes.

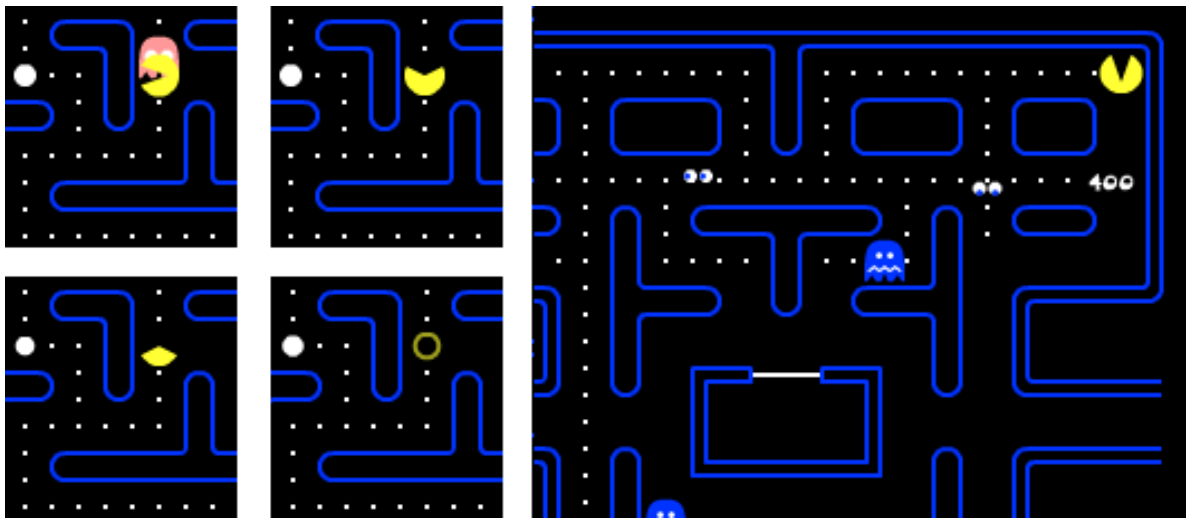
La construction de la classe personnage



Les membres à redéclarer

- Les éléments en bleus sont déclarés au niveau de Personnage. S'ils ne sont pas déclarés au niveau de Fantôme et de Pacman, cela signifie que Pacman et Fantôme utiliseront les méthodes de Personnage.
- Intuitivement, nous sentons que pacman et fantôme bleu, rouge, rose ou orange ne doivent pas se comporter de la même manière (par exemple, l'animation correspondant à la destruction d'un fantôme et celle correspondant à la destruction de pacman sont différentes).

Les différentes animations

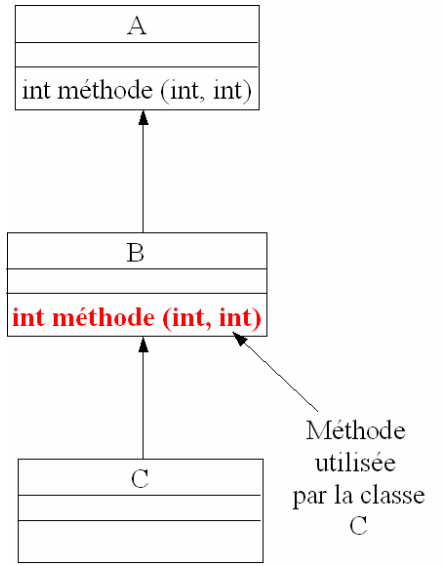


La redéfinition

- Il est donc nécessaire de redéfinir, au niveau des classes Pacman et Fantôme, les méthodes bleues déclarées au niveau Personnage afin d'adapter le traitement de la méthode à la classe.
- La redéfinition (overriding) consiste à réimplanter une version spécialisée d'une méthode héritée d'une classe mère (les signatures des méthodes au niveau de la classe mère et de la classe fille doit être identique).
- Si une méthode possède des arguments par défaut, nous devons retrouver cette caractéristique dans les méthodes des classes filles.

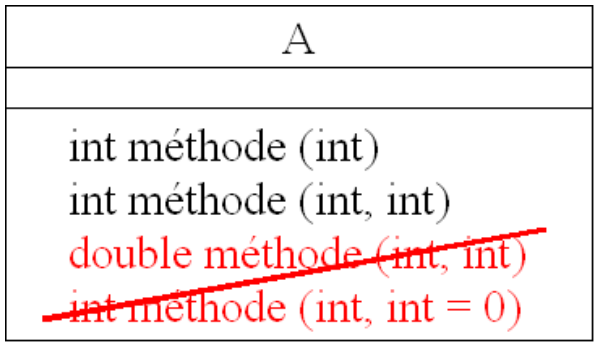
La redéfinition

- Lors de l'exécution, l'appel d'une méthode provoque sa recherche depuis la classe correspondant à l'objet où a été invoquée la méthode vers la classe mère, la classe « grand-mère » ...
- La méthode exécutée (si elle est trouvée) correspond à la première trouvée.



La surcharge

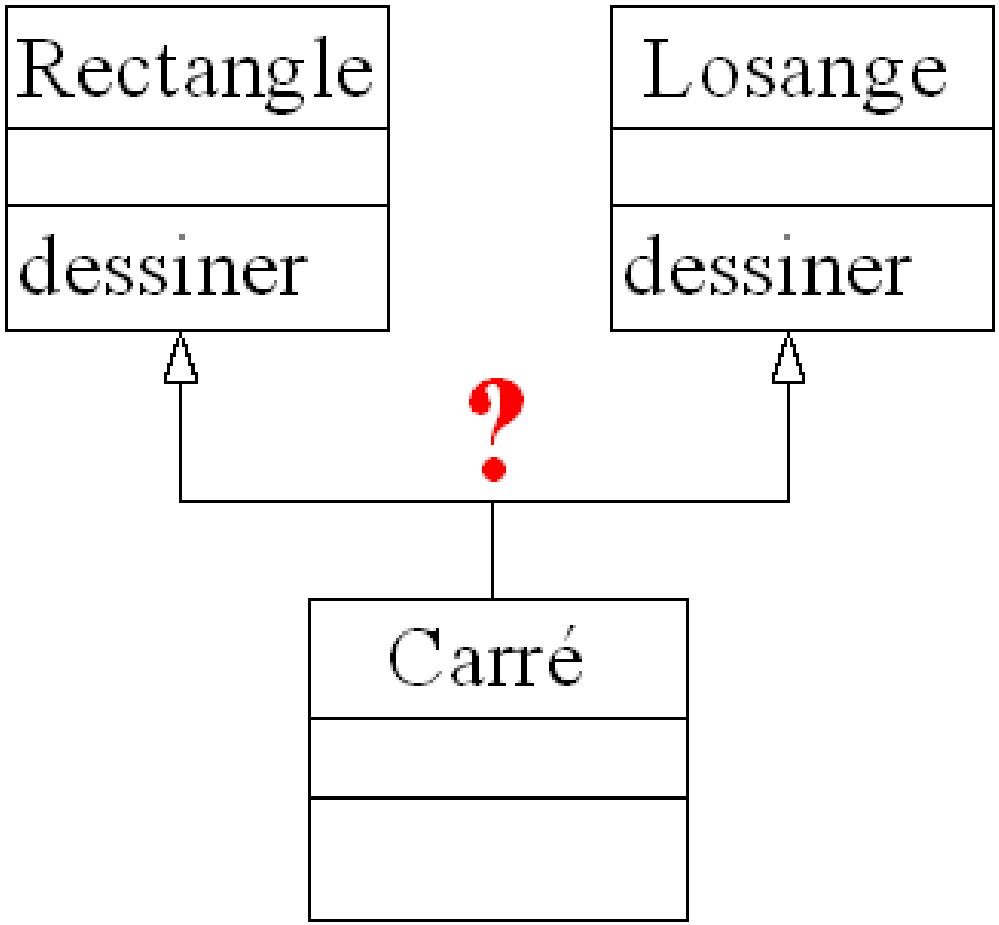
- La surcharge (overloading) consiste à proposer, au sein d'une même classe, plusieurs « versions » d'une même méthode qui diffèrent simplement par le nombre et le type de variables (le nom et le type de retour doit être identique).
- Il est possible d'indiquer une valeur par défaut à un ou plusieurs argument en partant de la fin, la distinction des signatures s'effectue alors sur les variables qui n'ont pas d'affectation par défaut).



Description du problème

- D'un point de vue conceptuel, il est possible de spécifier qu'une classe hérite de deux ou plusieurs classes mères.
- Si les classe mères possèdent certaines méthodes et attributs en commun, et que ces méthodes et attributs ne sont pas redéfinies au niveau de la classe, nous pouvons avoir une ambiguïté : lors de la recherche, quel chemin choisir ? ?

Mise en évidence graphique du problème

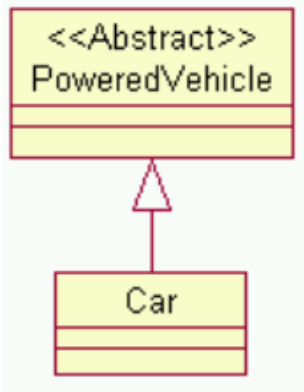


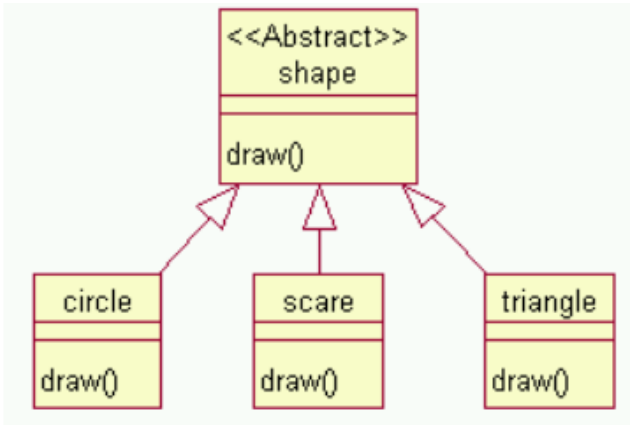
La gestion du problème en Java et en C++

- Certains langages comme Java ont résolu le problème en interdisant l'héritage multiple.
- D'autres comme le C++ autorise l'héritage multiple, le problème est alors résolu en écrivant explicitement quelle méthode utiliser (utilisation de l'opérateur « scope » : :) ou en changeant le nom d'une des méthodes des classes mères.

La classe abstraite

- Une classe abstraite est une classe qui ne peut pas être instanciée. Ce type de classe est généralement utilisée pour mettre en place un mécanisme de polymorphisme. D'un point de vue graphique, la classe abstraite se distingue de la classe « concrète » par l'ajout du mot clé « abstract » au dessus du nom de la classe.





Le polymorphisme

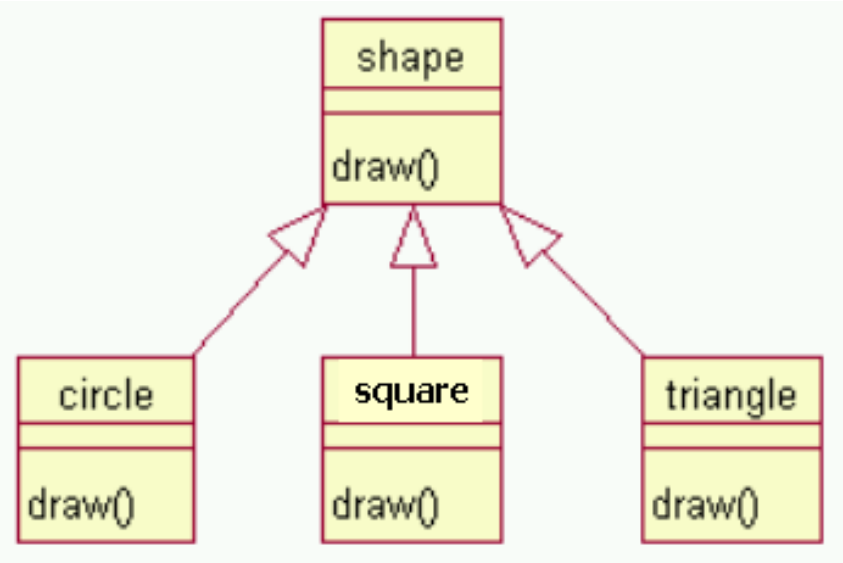
- Le polymorphisme est la propriété d'une entité de pouvoir se présenter sous diverses formes. D'un point de vue POO, ce mécanisme permet de faire collaborer des objets entre eux sans que ces derniers aient à donner leur type.

Les méthodes virtuelles

- Les objets doivent être instanciés à partir de classes qui héritent d'une même classe abstraite. Cette dernière est utilisée comme interface pour garantir l'existence d'attributs et surtout méthodes particulières dans les objets.
- Ces méthodes doivent être déclarées comme virtuelles ce qui a pour effet d'obliger le programmeur à redéfinir les méthodes au niveau des classes qui héritent de la classe abstraite.

La mise en œuvre du polymorphisme

- Grâce à ce mécanisme, il est possible, par exemple, de créer une liste de formes géométriques (shape), en ajoutant des objets de type circle, square et triangle puis d'appeler dans une boucle la méthode draw de chaque shape. Le programme se branche alors automatiquement sur la bonne méthode draw.



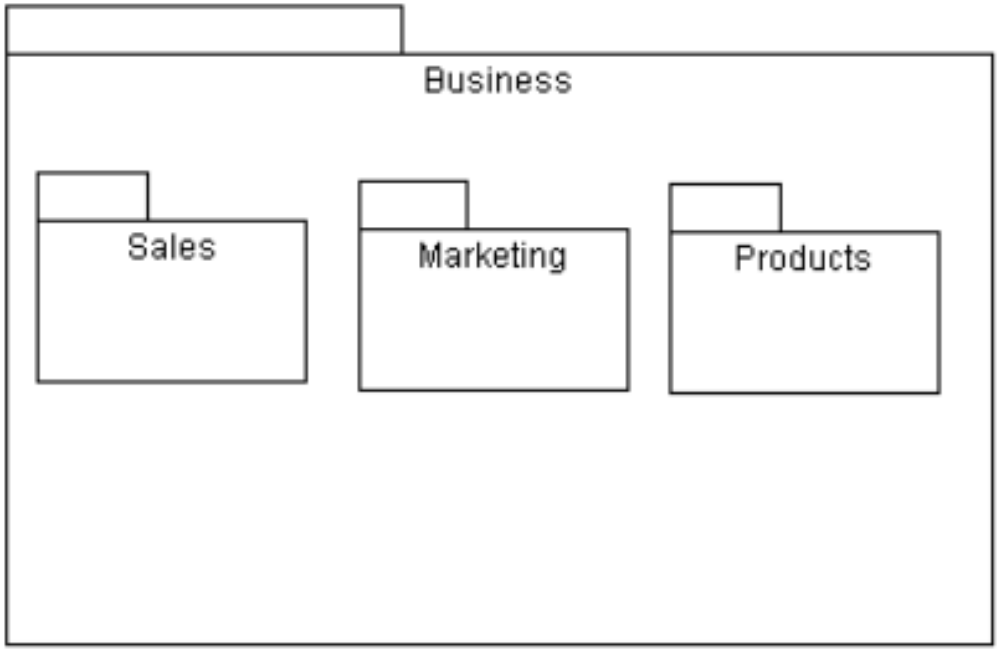
Plan

1. La notion de paquet.
2. La notion d'amitié.
3. Synthèse concernant les types d'accès

Le principe

- Les paquets (package) sont utilisés pour grouper les classes qui traitent d'un même problème pour former des ≪ bibliothèques de classes ≫.
- D'un point de vue graphique, un paquet est représenté par une ≪ chemise ≫ qui contient des classes ou d'autres paquets.
- D'un point de vue programmation, l'utilisation de paquet oblige à ≪ importer ≫ le paquet (spécifier le contexte de travail) ou à indiquer le nom complet de la classe (sous la forme paquet1.paquet2.classe) afin que le compilateur puisse trouver la définition des classes.

La représentation graphique



Les classes anonymes ou amies

- Les indicateurs de visibilité des classes sont les suivants :
  - public : la classe est visible partout où le paquet l'est (le nom de la classe est alors précédé de $+$) ;
  - anonyme (ami, non-public, paquet) : la classe n'est visible qu'aux classes de son propre paquet (le nom de la classe est précédé de).

Quatre types de visibilité

- En ce qui concerne les méthodes et les attributs, nous avons quatre types de visibilité :
  - public symbolisé par $+$;
  - privé symbolisé par $-$;
  - protégé symbolisé par $\#$;
  - paquet, ami symbolisé par $\sim$.

Tableau récapitulatif

- La visibilité effective varie selon le type de classe qui doit accéder aux données ou aux attributs.

Accessible à	+	#	~	-
La même classe	Oui	Oui	Oui	Oui
Une classe dans le même package	Oui	Oui	Oui	Non
Une sous-classe dans un autre package mais publique	Oui	Oui	Non	Non
Une classe publique quelconque dans un autre package	Oui	Non	Non	Non