Algorithmique et développement web S2 – 4. Fonctions et Programmation Orientée Objet (POO)–

Christophe BLANC

– IUT MMI –
IUT d'Allier
Université Clermont Auvergne
www.christophe-blanc.fr

2016-2017

- ✔ A l'instar des différents langage de programmation, PHP offre la possibilité de définir ses propres fonctions avec tous les avantages associés (modularité, capitalisation,...)
- ✔ Une fonction est un ensemble d'instructions identifiées par un nom, dont l'exécution retourne une valeur et dont l'appel peut être utilisé comme opérande dans une expression.
- ✔ Une procédure est un ensemble d'instructions identifiées par un nom qui peut être appelé comme un instruction.

Le mot clé *function* permet d'introduire la définition d'une fonction.

Syntaxe

```
function nom_fonction([parametres])
{
        instructions;
}
```

- ✔ nom_fonction : nom de la fonction (doit respecter les règles de nommage)
- ✔ parametres : paramètres éventuels de la fonction exprimés sous forme d'une liste de variables : $param1, $param2,...
- ✔ instructions : ensemble des instructions qui composent la fonction.

- ✔ Si la fonction retourne une valeur, il est possible d'utiliser l'instruction *return* pour définir la valeur de retour de la fonction.
- ✔ Le nom de la fonction ne doit pas être un mot réservé PHP (nom de fonction native, d'instruction) ni être égal au nom d'une autre fonction préalablement définie.
- ✔ Une fonction utilisateur peut être appelée comme une fonction native de PHP : dans une affectation, dans une comparaison,...

Fonctions
Déclaration et appel : exemple

```php
<?php
//fonction sans parametre qui affiche " Bonjour !" sans valeur de retour
function afficher_bonjour(){
        echo "Bonjour␣!</br>";
}
//fonction avec 2 parametres qui retourne le produit des deux parametres
function produit($param1,$param2){
        return $param1 * $param2;
}
//utilisation de la fonction afficher_bonjour
afficher_bonjour();
//utilisation de la fonction produit dans une affectation
$resultat=produit(2,4);
echo "$resultat</br>";
//utilisation de la fonction produit dans une comparaison
if (produit(10,12)>100){
        echo "Le␣resultat␣est␣superieur␣a␣100.</br>";
}
//utilisation de la fonction produit dans une affectation et une comparaison
if (($resultat = produit(10,12))>100){
        echo "$resultat␣est␣superieur␣a␣100.</br>";
}
?>
```

Bonjour !
8
Le resultat est supérieur à 100.
120 est supérieur à 100.

Il est possible d'utiliser une fonction avant de la définir

```
<?php
echo produit(5,5);

function produit($param1,$param2)
{
        return $param1 * $param2;
}
?>
```

Une fonction est utilisable uniquement dans le script où elle est définie. Pour l'employer dans plusieurs scripts, il faut, soit recopier sa définition dans les différents scripts (vous perdez l'intérêt de définir une fonction), soit la définir dans un fichier inclus partout où la fonction est nécessaire.

Exemple

Fichier *fonctions.inc* contenant des définitions de fonctions :

```php
<?php
//definition de la fonction produit
function produit($param1,$param2){
        return $param1 * $param2;
}
?>
```

Script utilisant les fonctions définies dans *fonctions.inc* :

```php
<?php
//inclusion du fichier contenant la definition des fonctions
include ("fonctions.inc");
echo produit(5,5);
?>
```

PHP propose "une fonction variable" qui permet de stocker un nom de fonction dans une variable et d'appeler la variable dans une instruction comme si c'était une fonction, avec la notation *$variable()*. sur une telle écriture, PHP remplace la variable par sa valeur et cherche à exécuter la fonction correspondante (qui doit ,bien entendu, exister).

```
<?php
function produit($param1,$param2){
        return $param1 * $param2;
}
function somme($param1,$param2){
        return $param1 + $param2;
}
function calculer($operation,$param1,$param2){
        return $operation($param1,$param2);
}
echo calculer("produit",3,5)."</br>";
echo calculer("somme",3,5)."</br>";
?>
```

Les paramètres éventuels de la fonction sont définis sous la forme d'une liste de variables. Nous allons étudier les possibilités suivantes :

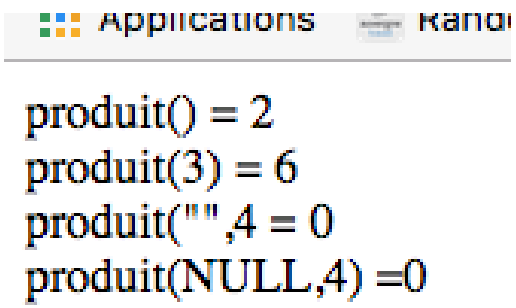
- ✔ définir une valeur par défaut pour un paramètre
- ✔ passer un paramètre par référence
- ✔ utiliser une liste de paramètres

Il est possible d'indiquer qu'un paramètre possède une valeur par défaut grâce à la syntaxe suivante :

```
$param = expression litterale
```

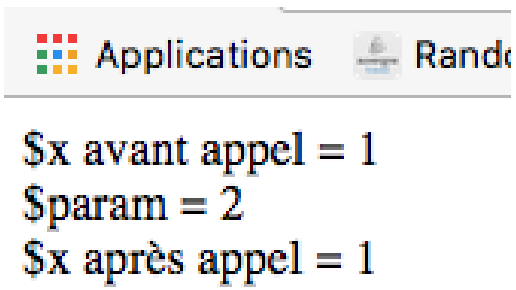
- ✔ La valeur par défaut d'un paramètre doit être une expression littérale et ne peut être ni une variable, ni une fonction, ni une expression composée.
- ✔ La valeur par défaut est utilisée comme valeur d'un paramètre lorsque la fonction est appelée, sans mentionner de valeur pour le paramètre en question.
- ✔ Ne pas donner de valeur à un paramètre ayant une valeur par défaut n'est possible qu'en partant de la droite.
- ✔ Passer un nombre insuffisant de paramètres et ne pas avoir de valeur par défaut génère une erreur. Passer trop de paramètres ne génère pas d'erreur ; les paramètres en trop sont ignorés.

```php
<?php
function produit($param1=1,$param2=2){
        return $param1 * $param2;
}
//appel sans parametre
echo "produit() = ".produit()."</br>";
//appel avec un seul parametre = forcement le premier
echo "produit(3) = ".produit(3)."</br>";
//echo "produit(,4) = ".produit(,4)."</br>";
        //interdit
echo "produit(\"\",4 = ".produit("",4)."</br>";
echo "produit(NULL,4) =".produit(NULL,4)."</br>";
?>
```



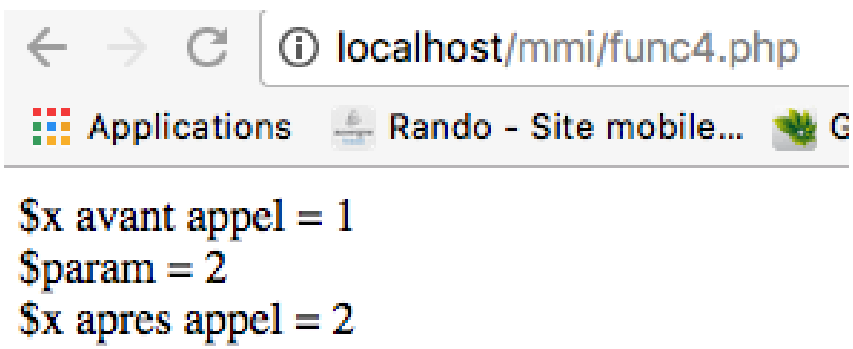
Par défaut, le passage des paramètres s'effectue par valeur : c'est une copie de la valeur qui est passée à la fonction. En conséquence, la modification des paramètres à l'intérieur de la fonction n'a aucun effet sur les valeurs dans le script appelant.

```
<?php
//definition d'une fonction qui prend un parametre
function par_valeur($param){
        $param++;
        echo "\$param = $param </br>";
}
$x=1;
echo "\$x avant appel = $x </br>";
par_valeur($x);
echo "\$x apres appel = $x </br>";
?>
```



En cas de besoin, il est possible d'avoir un passage par référence en utilisant l'opérateur de référence & devant le nom du paramètre dans la définition de la fonction. Avec une telle définition, c'est une référence vers la variable (plus une copie) qui est passée à la fonction : cette dernière travaille directement sur la variable du script appelant.

```php
<?php
//definition d'une fonction qui prend un parametre
function par_reference(&$param){
        $param++;
        echo "\$param = $param </br>";
}
$x=1;
echo "\$x avant appel = $x </br>";
par_reference($x);
echo "\$x apres appel = $x </br>";
?>
```



L'utilisation du mot clé *return* à l'intérieur d'une fonction, permet de définir la valeur de retour de la fonction et de stopper son exécution.

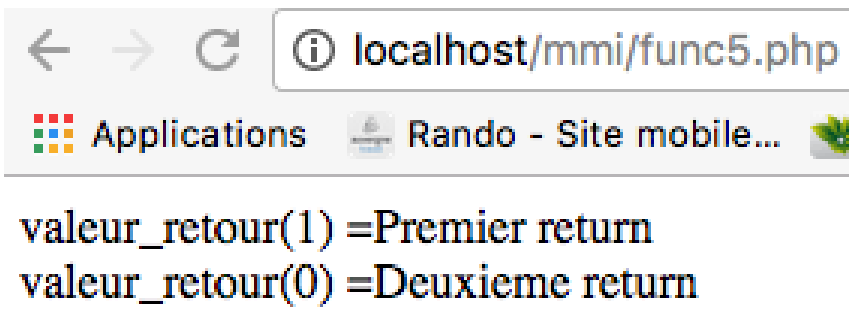
Syntaxe

```
return expression
```

- ✔ *expression* : expression dont le résultat constitue la valeur de retour de la fonction.
- ✔ Le résultat d'une fonction peut être de n'importe quel type (chaîne, nombre, tableau,...)
- ✔ Si plusieurs instructions return sont présentes dans la fonction, c'est la première rencontrée dans le déroulement des instructions qui définit la valeur de retour et provoque l'interruption de la fonction.

Fonctions
Valeur de retour : exemple

```php
<?php
//definition d'une fonction avec deux appels a return
function valeur_retour($param){
        if($param==1){
                return "Premier return";
        }
        return "Deuxieme return";
}
//appel a la fonction
echo "valeur_retour(1) =".valeur_retour(1)."</br>";
echo "valeur_retour(0) =".valeur_retour(0)."</br>";
?>
```

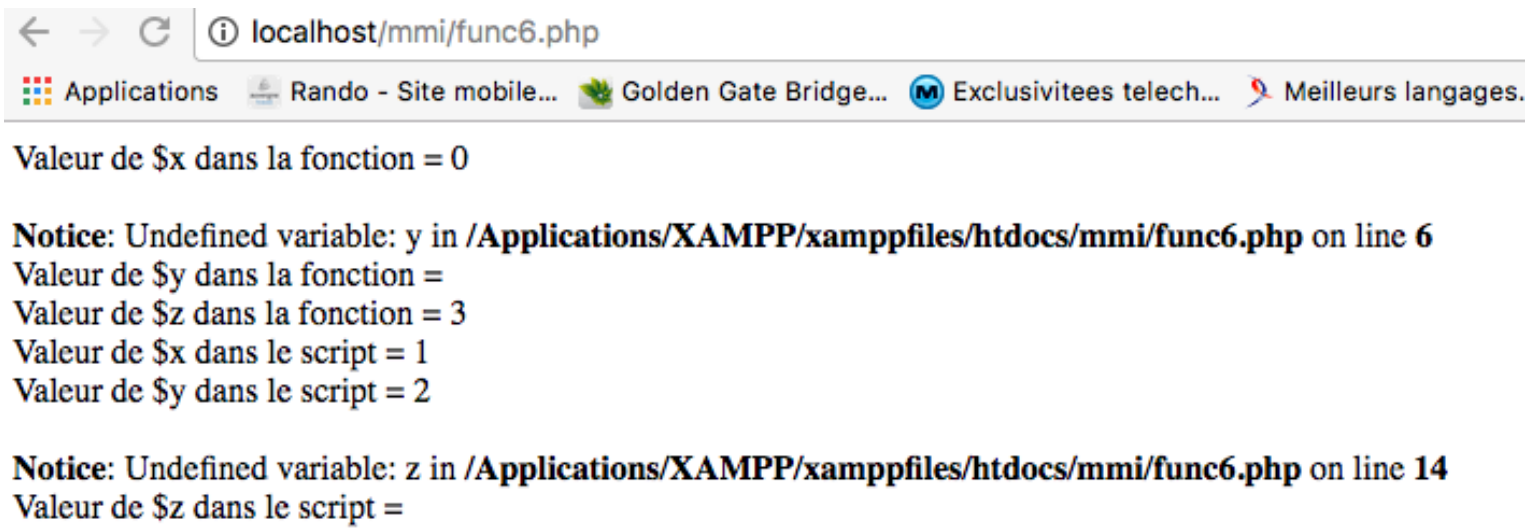


- ✔ Les variables utilisées à l'intérieur d'une fonction sont locales : elles sont non définies en dehors de la fonction et initialisées à chaque appel de la fonction.
- ✔ Réciproquement, une variable définie en dehors de la fonction (dans le script appelant) n'est pas définie à l'intérieur de la fonction.

Fonctions

Variables utilisées dans les fonctions : variables locales - exemple

```php
<?php
function variable_locale(){
        $x = 0;
        $z = 3;
        echo "Valeur de \$x dans la fonction = $x</br>";
        echo "Valeur de \$y dans la fonction = $y</br>";
        echo "Valeur de \$z dans la fonction = $z</br>";
}
$x = 1;
$y = 2;
variable_locale();
echo "Valeur de \$x dans le script = $x</br>";
echo "Valeur de \$y dans le script = $y</br>";
echo "Valeur de \$z dans le script = $z</br>";
?>
```



- ✔ PHP propose une notion de variable globale pour accéder, dans une fonction, aux variables définies dans le contexte du script appelant
- ✔ Pour cela, à l'intérieur de la fonction, il faut déclarer les variables globales que la fonction utilise avec l'instruction *global*.

Syntaxe

```
global $variable,...
```

- ✔ *$variable* : variable du script appelant que la fonction souhaite utiliser. Plusieurs variables peuvent être mentionnées, en les séparant par des virgules.

Fonctions

Variables utilisées dans les fonctions : variables globales - exemple

```php
<?php
function variable_globale(){
        global $x;
        echo "Valeur de \$x au debut la fonction = $x</br>";
        $x = 0;
        $z = 3;
        echo "Valeur de \$x a la fin de la fonction = $x</br>";
        echo "Valeur de \$y dans la fonction = $y</br>";
        echo "Valeur de \$z dans la fonction = $z</br>";
}
$x = 1;
$y = 2;
variable_globale();
echo "Valeur de \$x dans le script = $x</br>";
echo "Valeur de \$y dans le script = $y</br>";
echo "Valeur de \$z dans le script = $z</br>";
?>
```

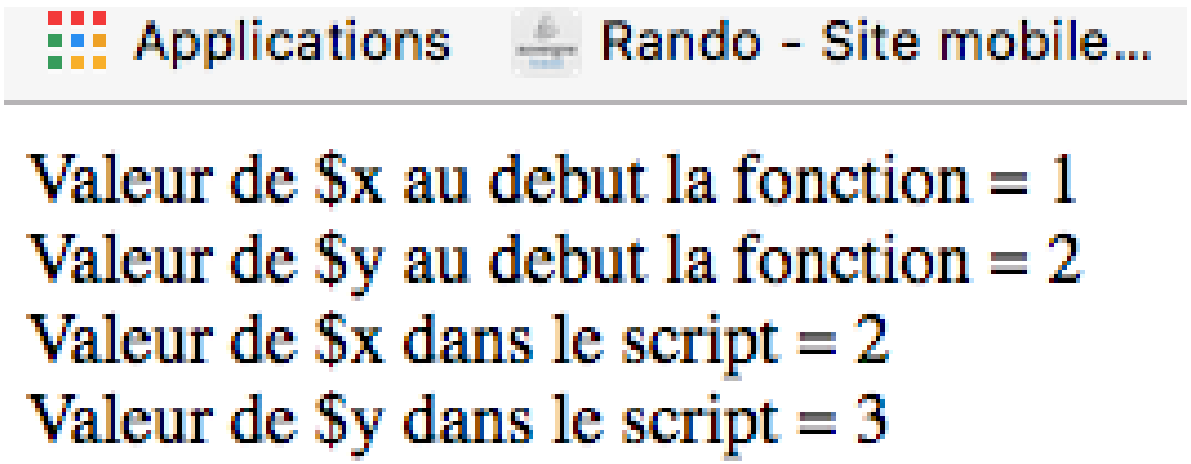
Valeur de $x au debut la fonction = 1
Valeur de $x a la fin de la fonction = 0

Notice: Undefined variable: y in **/Applications/XAMPP/xamppfiles/htdocs/mmi/func7.php** on line **8**
Valeur de $y dans la fonction =
Valeur de $z dans la fonction = 3
Valeur de $x dans le script = 0
Valeur de $y dans le script = 2

Notice: Undefined variable: z in **/Applications/XAMPP/xamppfiles/htdocs/mmi/func7.php** on line **16**
Valeur de $z dans le script =

Il est possible, sans déclaration, d'accéder aux variables globales à l'intérieur d'une fonction, en utilisant un tableau associatif *$GLOBALS* géré par PHP. Dans ce tableau associatif, la clé est égale au nom de la variable globale (sans le $) et la valeur, à la valeur de la variable globale

```php
<?php
function variable_globale(){
        global $x;
        echo "Valeur de \$x au debut la fonction = $GLOBALS[x]</br>";
        echo "Valeur de \$y au debut la fonction = $GLOBALS[y]</br>";
        $GLOBALS["x"]++;
        $GLOBALS["y"]++;
}
$x = 1;
$y = 2;
variable_globale();
echo "Valeur de \$x dans le script = $x</br>";
echo "Valeur de \$y dans le script = $y</br>";
?>
```



Par défaut les variables locales d'une fonction sont réinitialisés à chaque appel de la fonction. L'instruction *static* permet de définir des variables locales statiques qui ont pour propriété de conserver leur valeur d'un appel à l'autre de la fonction, pendant la durée du script.

Syntaxe

```
static $variable = expression_litterale
```

- ✔ *$variable* : variable concernée
- ✔ *expression_litterale* : valeur initiale affectée à la variable lors du premier appel de la fonction.

```php
<?php
function variable_statique(){
        static $variable = 0;
        $autre_variable = 0;
        echo "\$variable = $variable</br>";
        echo "\$autre_variable = $autre_variable</br>";
        $variable++;
        $autre_variable++;
}
echo "<B>Premier appel de la fonction :</B></br>";
variable_statique();
echo "<B>Deuxieme appel de la fonction :</B></br>";
variable_statique();
echo "<B>Troisieme appel de la fonction :</B></br>";
variable_statique();
?>
```

Premier appel de la fonction :
$variable = 0
$autre_variable = 0
Deuxieme appel de la fonction :
$variable = 1
$autre_variable = 0
Troisieme appel de la fonction :
$variable = 2
$autre_variable = 0

- ✔ Nous savons que la portée des constantes est le script dans lequel elles sont définies.
- ✔ A la différence des variables, cette portée s'étend aux fonctions appelées dans le script : une constante peut être utilisée à l'intérieur de la fonction sans qu'elle soit déclarée globale.
- ✔ Réciproquement, une constante définie dans une fonction peut être utilisée dans un script, après appel de la fonction.

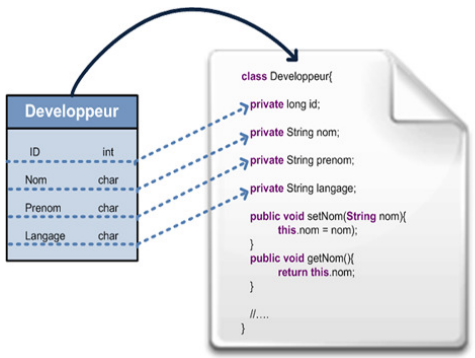
```php
<?php
define("CONSTANTE_SCRIPT","constante␣script");
function constante(){
        define("CONSTANTE_FONCTION","constante␣fonction");
        echo "Dans␣la␣fonction,␣CONSTANTE_SCRIPT␣=␣".CONSTANTE_SCRIPT."</br>";
}
constante();
echo "Dans␣le␣script,␣CONSTANTE_FONCTION␣=␣".CONSTANTE_FONCTION."</br>";
?>
```

Dans la fonction, CONSTANTE_SCRIPT = constante script
Dans le script, CONSTANTE_FONCTION = constante fonction

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

Objet :

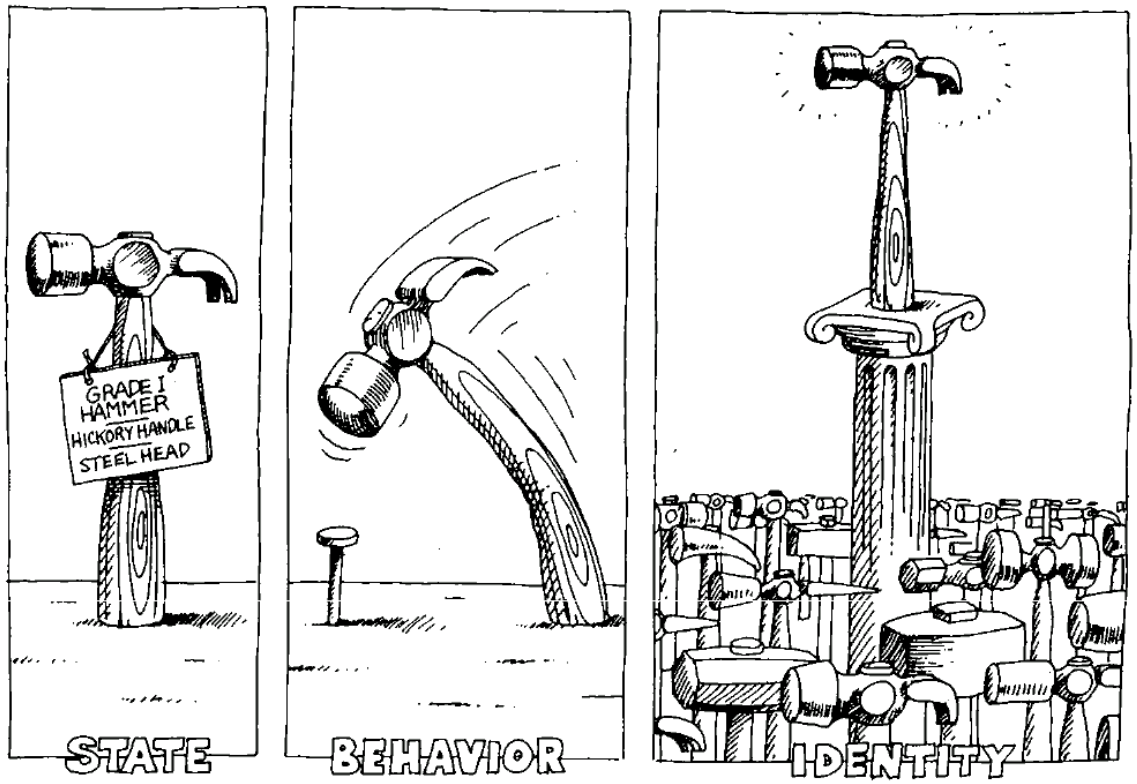
- Un objet est une entité manipulable par un ordinateur. Il correspond à une réalité ou une abstraction.



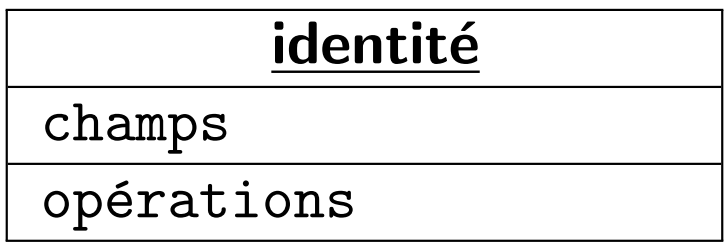
- Un objet comprend trois composantes :
  - une identité ;
  - une composante statique correspondant à une ou plusieurs structures de données utilisées pour stocker ses caractéristiques (état) : les champs ;
  - une composante dynamique correspondant à un ensemble d'opérations qui lui sont particulières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

La notion d'objet

L'anatomie d'un objet : l'objet selon Grady BOOCH



- 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.



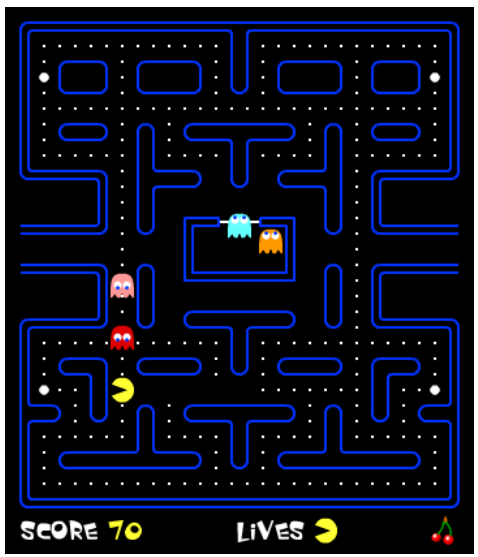
- Dans ce dernier cas, l'identité peut être écrite de trois manières (mais souligné dans les 3 cas de figure) :
  - nom (l'objet a un nom mais on ne précise pas le type)
  - :type (l'objet a un type mais on ne précise pas le nom)
  - nom :type (identité complète)

- 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`



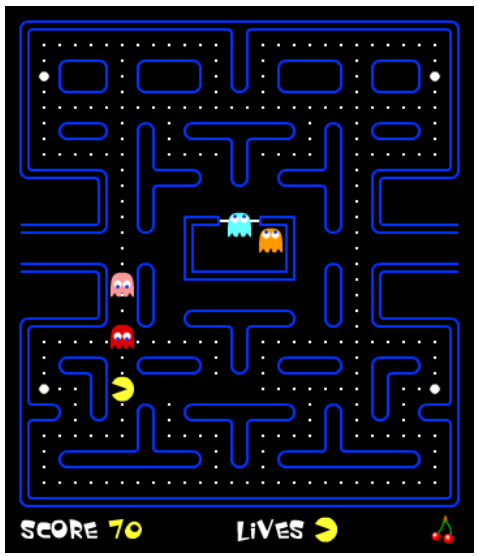
- Les éléments statiques et dynamiques d'un objet peuvent être cachés ou non aux autres objets.
- Cette caractéristique est appelée la visibilité, elle permet au programmeur de contrôler les éléments qu'il peut rendre :
  - publiques ;
  - privés.

- 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.

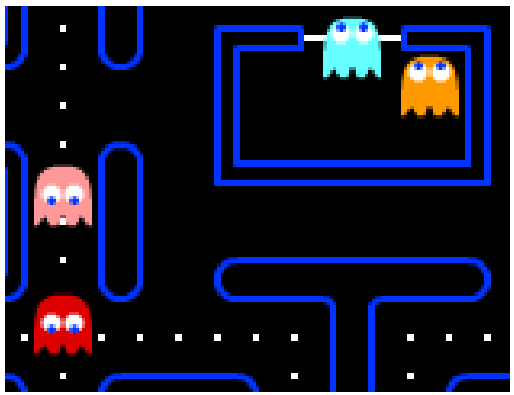
- Si une opération est qualifiée de publique, les objets externes peuvent l'utiliser pour modifier l'état d'un objet.
- Si une opération est qualifiée de privée, elle n'est utilisé que par l'objet lui-même.
- Contrairement aux attributs, les opérations sont considérées par défaut comme publiques, le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

- 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



- 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.



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

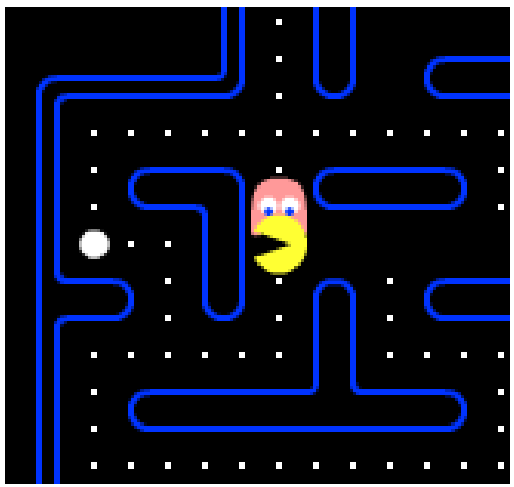
fantôme rose
couleur = rose position état
se déplacer se faire croquer - se détruire - se régénérer

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

- 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.

- 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.

- 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.

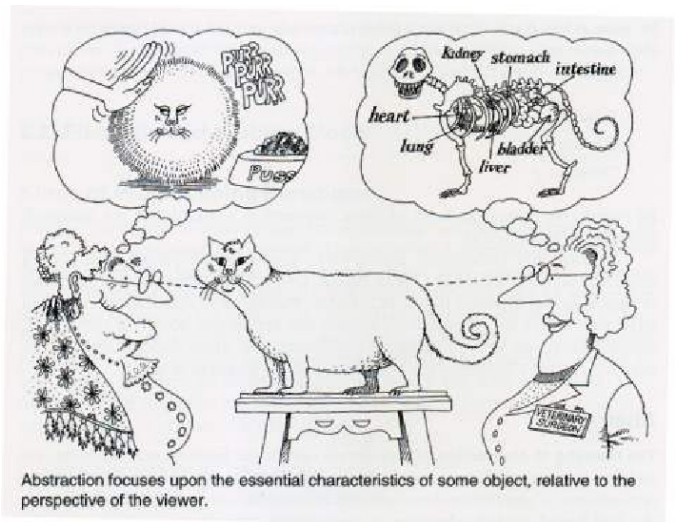


1. La classe vue comme une abstraction des objets.
2. Les attributs et les méthodes de classe.
3. L'objet vu comme une instance d'une classe.

- 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.

- 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.

- 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.



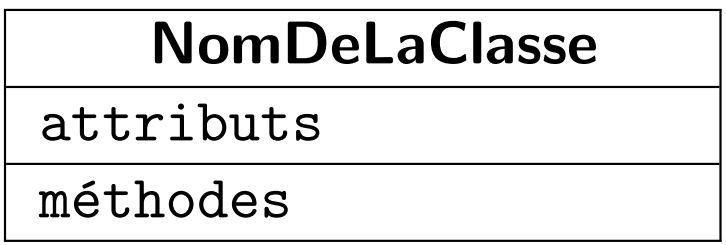
Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

- 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.

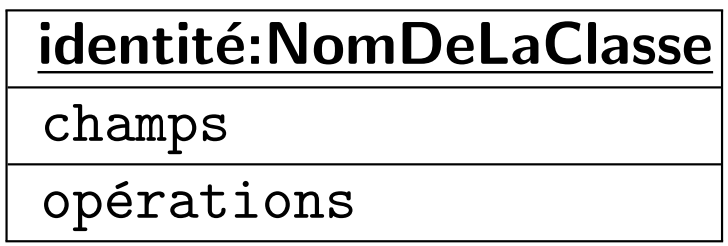
- Il existe deux types de méthodes :
  - les méthodes de classes qui peuvent être invoquées sur la classe (par exemple, les méthodes qui permettent d'instancier les objets) ;
  - les méthodes de représentant (instance méthods) seront utilisées par les objets pour accéder en lecture ou en écriture aux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

- 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.

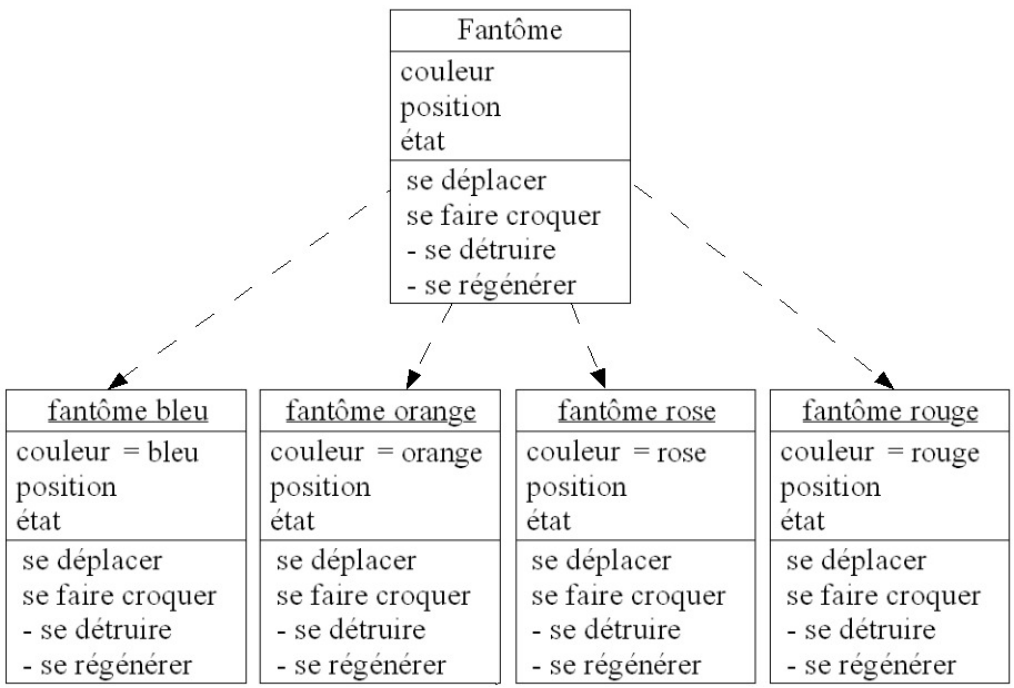
Une Classe



Une instance de la classe est un objet

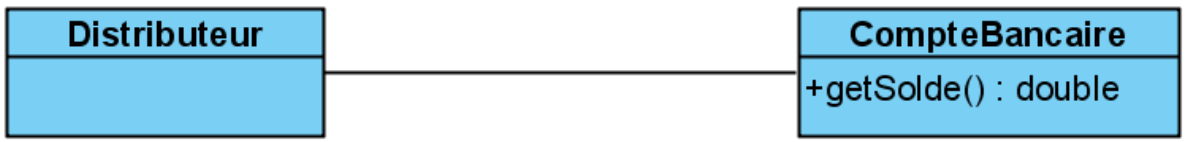


- Si nous considérons l'exemple des fantômes, nous avons alors une classe Fantôme qui permet d'instancier les quatre fantômes du jeux.

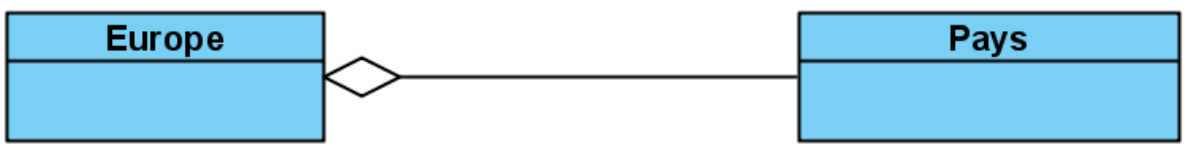


1. L'association.
2. L'agrégation.
3. La composition.
4. L'héritage.
5. Notions de surcharge et de redéfinition.
6. L'héritage multiple.
7. Les classes abstraites et le polymorphisme

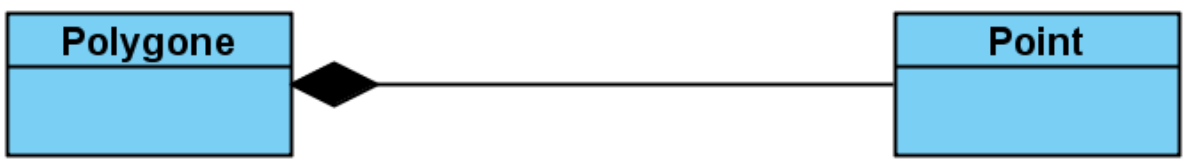
- L'association indique un lien qui peut se créer au cours du fonctionnement lorsqu'un objet d'une classe envoie un message vers un objet d'une autre classe. D'un point de vue graphique, nous modélisons cette relation par un simple trait entre les deux classes.



- 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.



- 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 ≫.



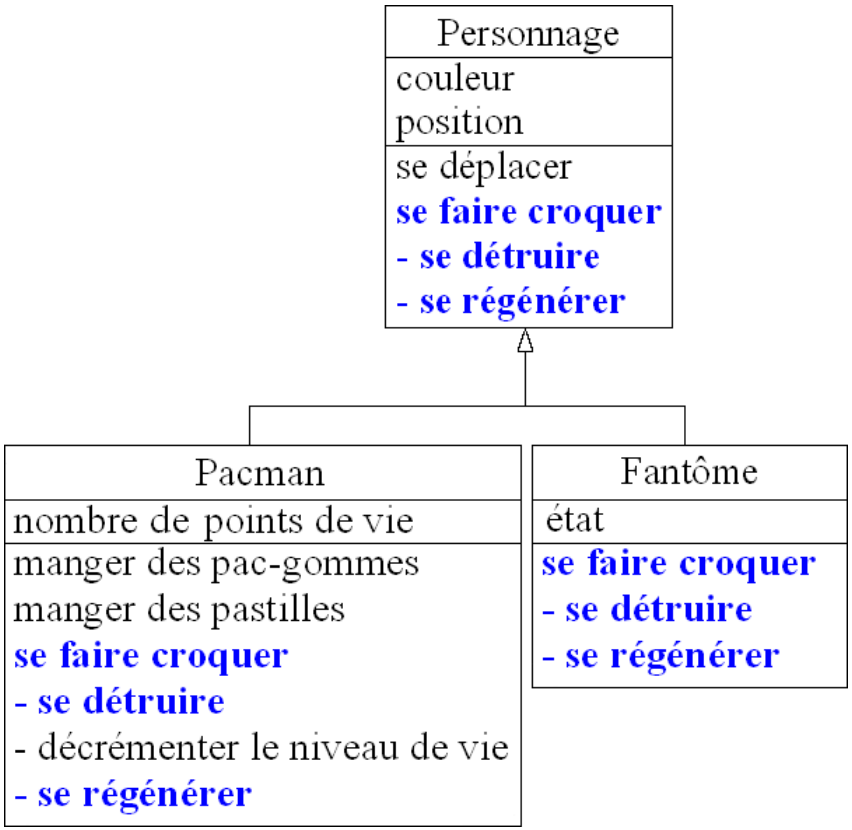
- La relation d'héritage peut être vue de deux manières :
  - Elle permet de factoriser des attributs et des méthodes d'un même ensemble de classes afin de faciliter la déclaration et la maintenance du code.
  - Elle permet de créer des classes spécialisées à partir d'une classe de base.

- 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.

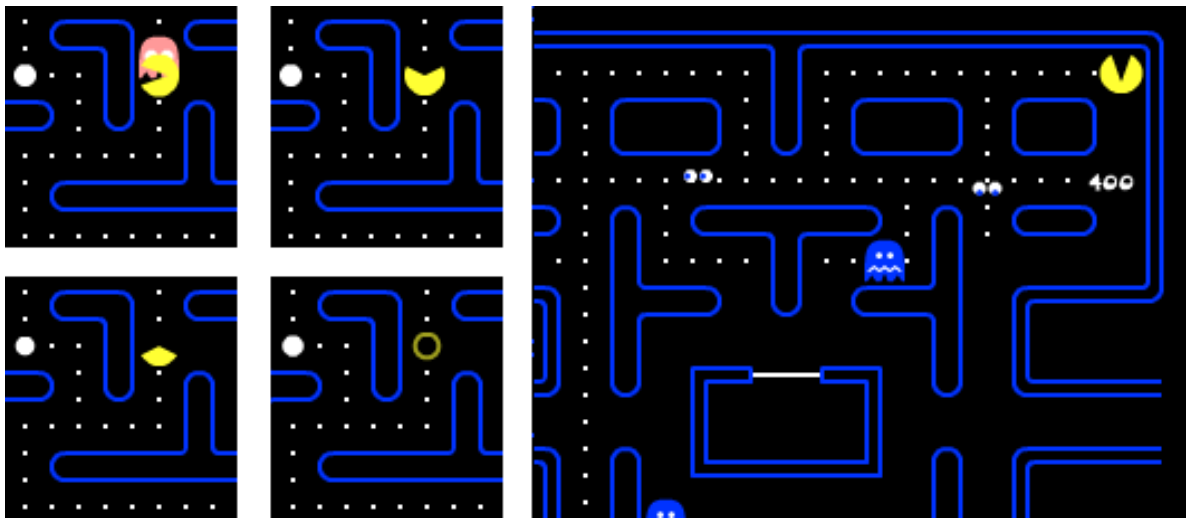
- Si nous reprenons l'exemple du jeu Pacman, nous remarquons les faits suivants :
  - l'objet pacman est issu d'une classe Pacman (généralement, le nom des classes commence par une majuscule alors que le nom des objets commence par une minuscule) ;
  - Les classes Pacman et Fantômes ont des éléments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

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**

- 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.

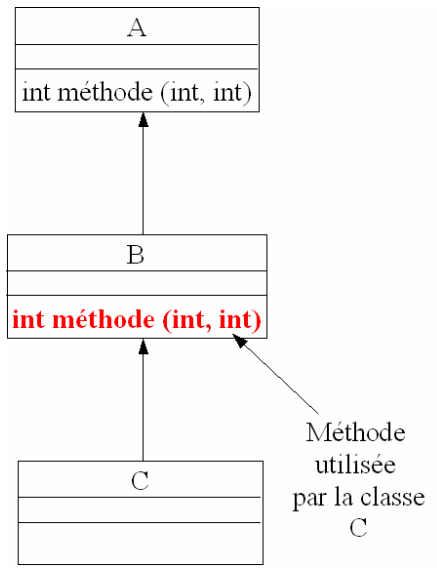

Personnage
couleur
position
se déplacer
se faire croquer
- se détruire
- se régénérer
Pacman
nombre de points de vie
manger des pac-gommes
manger des pastilles
se faire croquer
- se détruire
- décrémenter le niveau de vie
- se régénérer
Fantôme
état
se faire croquer
- se détruire
- se régénérer

- 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).

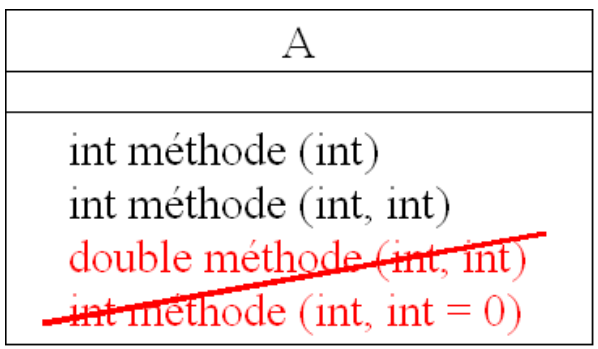

400

- 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.

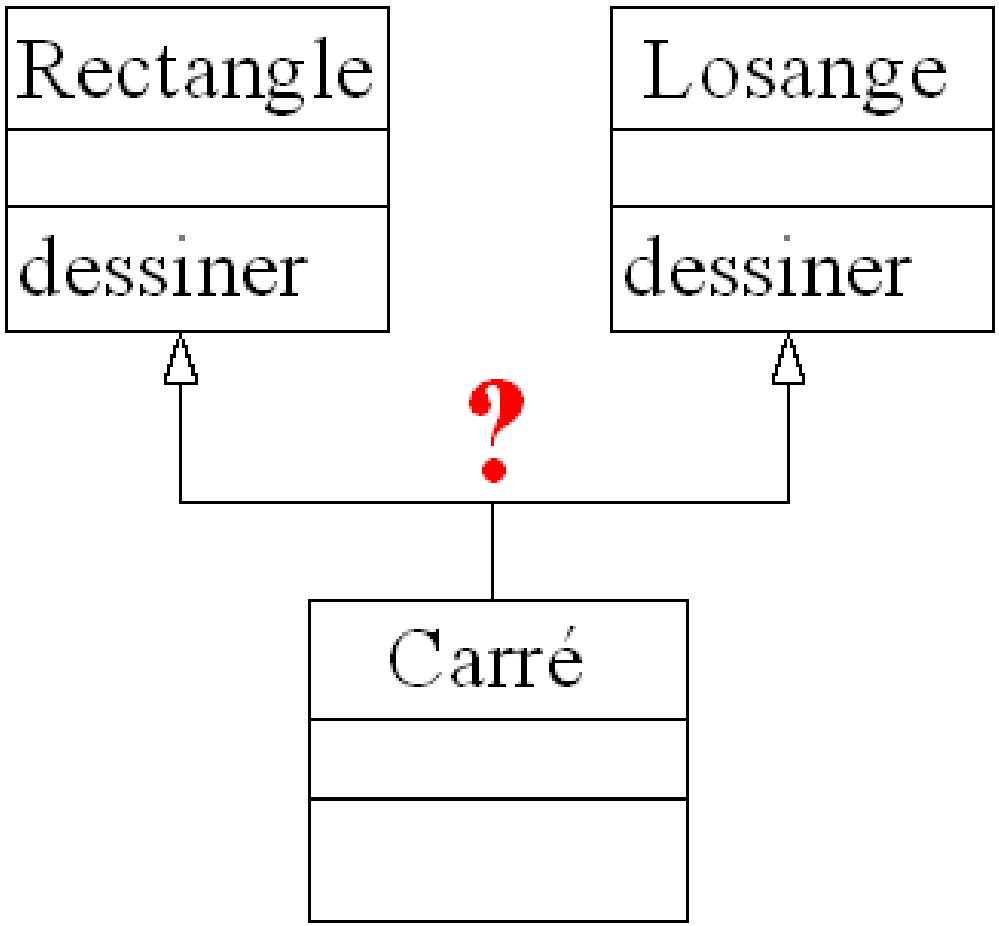
- Lors de l'exécution, l'appel d'une méthode provoque sa recherche depuis la classe correspondant à l'objet où a été invoquée la méthode vers la classe mère, la classe ≪ grand-mère ≫ ...
- La méthode exécutée (si elle est trouvée) correspond à la première trouvée.



- La surcharge (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).

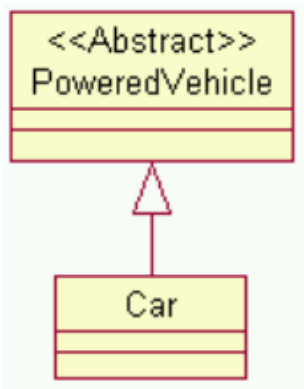


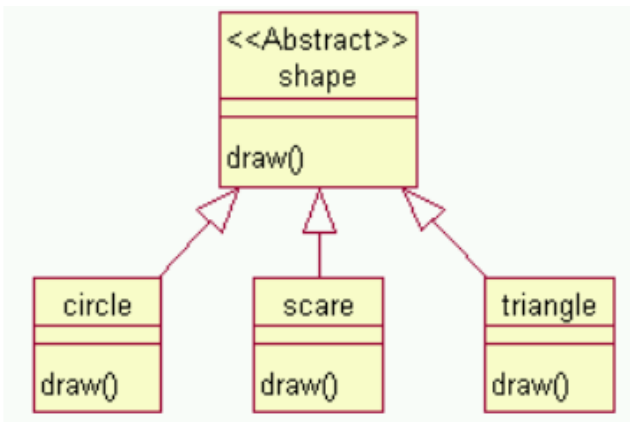
- D'un point de vue conceptuel, il est possible de spécifier qu'une classe hérite de deux ou plusieurs classes mères.
- Si les classe mères possèdent certaines méthodes et attributs en commun, et que ces méthodes et attributs ne sont pas redéfinies au niveau de la classe, nous pouvons avoir une ambiguïté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


Rectangle
dessiner
Losange
dessiner
?
Carré

- 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.

- 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 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 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.

- 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.