Si une propriété ressemble à une variable, elle n’en est pas une dans la mesure où l’affectation d’une valeur à une propriété peut déclencher l’exécution d’une méthode. Cette dernière, en plus d’accéder au champ visé, pourra effectuer des traitements particuliers nécessaires au programme.

### Modifier une méthode après héritage

Vous pouvez aussi souhaiter modifier le comportement d’une méthode reçue en héritage. Pour remplacer celle héritée, il suffit qu’elle soit accessible à la classe enfant : soit elle est privée mais présente dans la même unité, soit elle est d’une visibilité supérieure et accessible y compris dans une autre unité.

Par exemple, en ce qui concerne la méthode *Manger* définie dans l’ancêtre *TAnimal*, vous estimerez à juste titre qu’elle a besoin d’être adaptée au régime d’un carnivore. Afin de la redéfinir, il suffirait de l’inclure à nouveau dans l’interface puis de coder son nouveau comportement :

TChien = class(TAnimal)

strict private

fBatard : Boolean ;

procedure SetBatard;

public

procedure Manger; // la méthode est redéfinie

procedure Aboyer;

procedure RemuerDeLaQueue;

property Batard: Boolean read fBatard write SetBatard;

end;

La nouvelle méthode remplacera entièrement[[1]](#footnote-1) celle que contenait *TAnimal*.

### Polymorphisme

Avant de passer à la réalisation d’un projet mettant en œuvre ces notions, il vous faut en aborder une dernière : le *polymorphisme[[2]](#footnote-2)*. Il s’agit d’une conséquence importante de l’héritage qui fait que la méthode réelle à appeler

Les objets qui appartiennent à la même arborescence sont interchangeables : ils partagent au moins un ancêtre commun et peuvent par conséquent être affectés à une variable qui serait du type de cet ancêtre.

Ainsi, une affectation telle que celle qui suit est autorisée :

MonAnimal := MonChien ;

Cette affectation signifie que l’objet *MonAnimal* possède à présent toutes les propriétés de *MonChien*.

 La réciproque n’est pas autorisée, car *TChien* étant plus riche que *TAnimal*, la valeur de certaines propriétés resterait indéterminée. Dans le cas présent, ce serait *Batard* qui poserait problème.

Le polymorphisme est une propriété très importante des objets : il va permettre un ensemble d’échanges de propriétés et de partages de méthodes.

## La basse-cour des Contes du Chat perché.

## Structure d’une classe

### Sections d’une classe

### Les champs

### Les méthodes

#### Méthodes abstraites

#### Méthodes virtuelles

#### Méthodes statiques

#### Méthodes de classes

### La variable Self

## La POO en action : des chiens animés

Lorsque vous utilisez des composants dont le propriétaire[[3]](#footnote-3) est défini (ce qui est le cas dès que vous vous servez des fiches et que vous y déposez les composants nécessaires à votre application), c’est ce propriétaire qui est chargé d’allouer et de libérer la mémoire de manière transparente. Vous n’avez par conséquent pas à vous en occuper.

var

MonAnimal : TAnimal ;

begin

MonAnimal  := TAnimal.Create ; // on crée la liste de chaînes

try // on protège le code de manipulation pour être sûr de préserver les ressources

// ici le traitement voulu…

MonAnimal.Dormir ;

finally

// en interne, la méthode *Free* appelle le destructeur *Destroy*

MonAnimal .Free ; // les ressources seront toujours libérées

end ;

end ;

1. On verra plus loin qu’il est possible d’hériter de l’ancienne méthode et d’en modifier simplement le comportement. [↑](#footnote-ref-1)
2. L’étymologie en est grecque : *poly* signifie *plusieurs* et *morphie* signifie *forme*. [↑](#footnote-ref-2)
3. Cette notion est expliquée p. XXX. [↑](#footnote-ref-3)