RECAP POO

➔ La forme d'une classe : class Nom{

Mettre en private les attributs

Private $nom ;

…

Mettre le construct

```
public function __construct(string $nom,string $prenom, string $numerClient){
        $this->nom=$nom;
        $this->prenom=$prenom;
        $this->numeroClient=$numerClient;
    }
```

Mettre les get et les sets

```
    public function getNom(){return $this->nom;}
    public function setNom($nom){$this->nom=$nom;}
```

(facultatif) On peut ajouter des méthodes

```
public function seNourrir(){echo "Le lapin se nourrit\n";}
```

}

➔ On fait un $this->nomAttribut, quand on est dans la classe de l'attribut. (Exemple : Dans la table Animal, il y a comme attribut la couleur du pelage et nombre de patte. Tu peux faire this->couleur) **PAS UN GET**

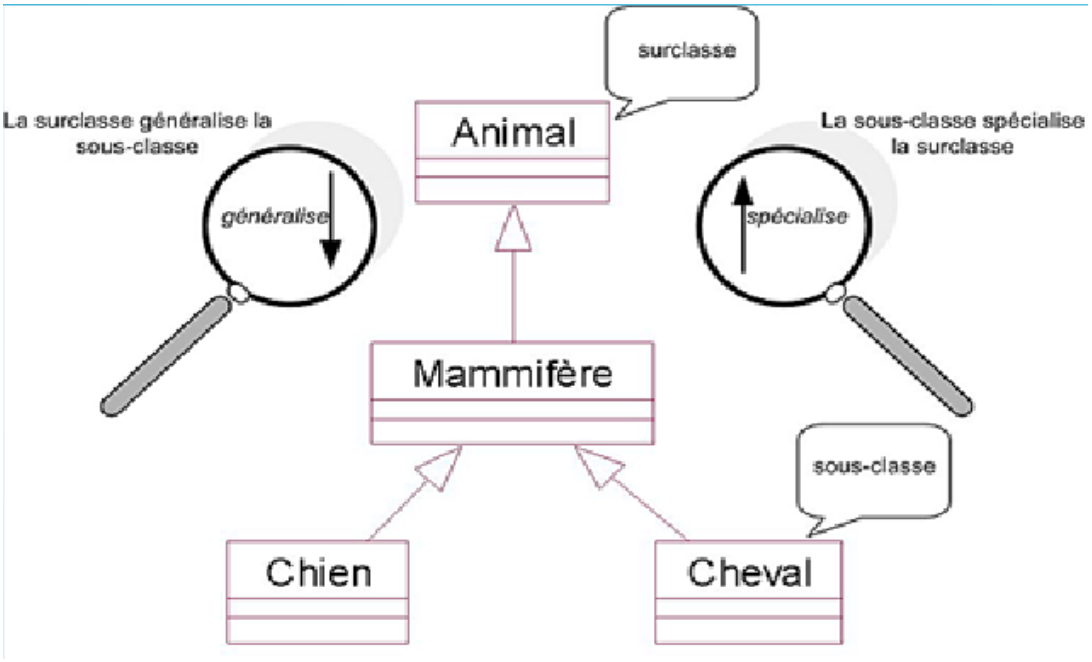
➔ On fait un $variable-> GetNomattribut, quand on est en-dehors de la classe de l'attribut et qu'on veut récupérer la donnée de l'attribut. Donc, on le fait dans l'index. OU dans une autre classe. (Exemple : Dans la table Animal, il y a comme attribut la couleur du pelage et nombre de patte. Dans la table Chat, il y a l'attribut longueur griffe. Tu peux faire un $...->getCouleur() dans la classe Chat pour aller chercher dans la table animal, la couleur du pelage de l'animal.)

➔ On fait un $variable->SetNomattribut, quand on est en-dehors de la classe de l'attribut et qu'on veut modifier la donnée de l'attribut. Donc, on le fait dans l'index. OU dans une autre classe. **Attention, Si on ne met rien dans le construct, on a possibilité de remplir la table soit avec un new soit avec des SET.**

➔ On peut mettre dans les attributs un private static $tab=[]. Pour éviter de faire $Animaux=[$animal1,$animal2,…].

➔ On peut faire un static pour l'id.

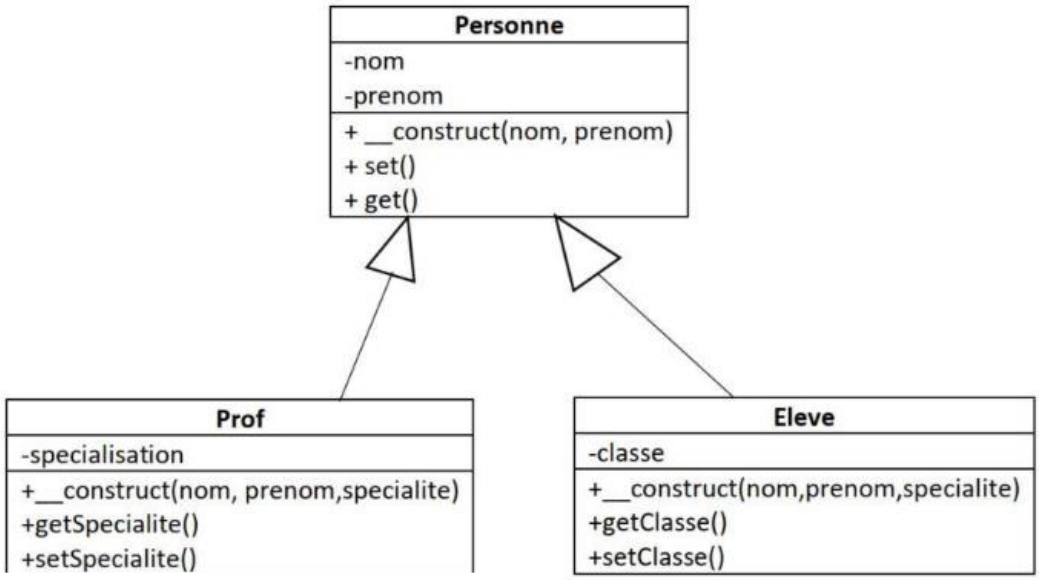
➔ On fait l'héritage, quand 2 tables on les mêmes attributs et que l'une des 2 tables à besoin d'attributs supplémentaires. (Exemple : Ici la table Animal qui est la mère possède les attributs : nom, capaciteàbouger. La table mammifère a besoin des attributs de la table Animal et d'ajouter de nouveaux attributs comme le mode de reproduction)




```
class Lapin extends Animal{ }
```

On prend la classe lapin (fille) qu'on étends à la classe animal (mère)

➔ On fait une classe abstraite, quand 2 tables on les mêmes attributs et que l'une des 2 tables à besoin d'attributs supplémentaires et qu'on l'une des 2 tables n'a pas besoin d'être instancié. (Exemple : Ici la table personne qui est la mère possède les attributs : nom, prénom. La table prof a besoin des attributs de la table personne et d'ajouter de nouveaux attributs comme la spécialisation. Sauf qu'on n'a pas besoin d'instancié la classe personne, on a juste besoin de récupérer ses méthodes par la classe abstraite.)

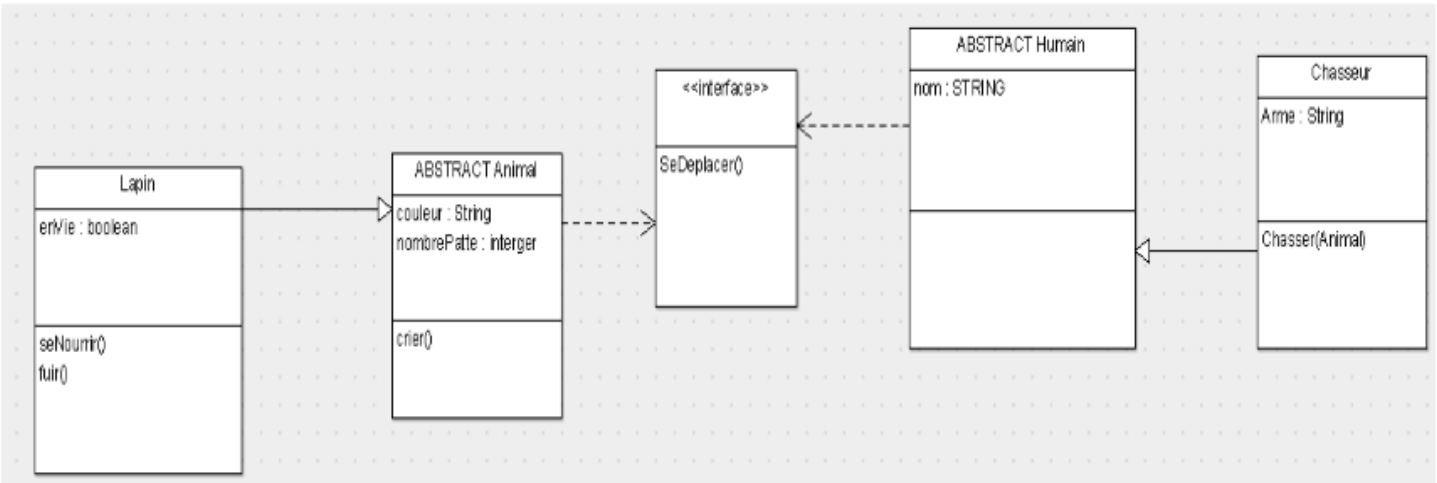


Personne ::__construct($nom,$prenom)

-

```
abstract class Animal{}
```

➔ On fait une interface, quand on a besoin d'une méthode pour plusieurs classes. (On peut le faire sur classe simple ou sur une classe abstrate.)




```
abstract class Animal implements IDeplacement{}
```

On prend la classe abstraite(fille) dont on implémente l'interface(mère).

FONCTIONS IMPORTANTES :

1. Pour éviter de faire plein de require dans l'index. Il faut créer un dossier Service dont lequel on met fonction.php. dans cette fonction on fera :

function chargerClasse($classe){

require « chemin/class/$class/.class.php

}

Ensuite,dans l'index, on fera un require sur le dossier service et la fonction.php. Puis, on mettra spl_autoload_register('chargerClasse') afin de récupérer cette fonction.

2. Conversion d'un string en date :

```
3.      public function formatageDate($date)
4.      {
5.          $conversion = strtotime($date);
6.          return date('d/M/Y',$conversion);
7.      }
```