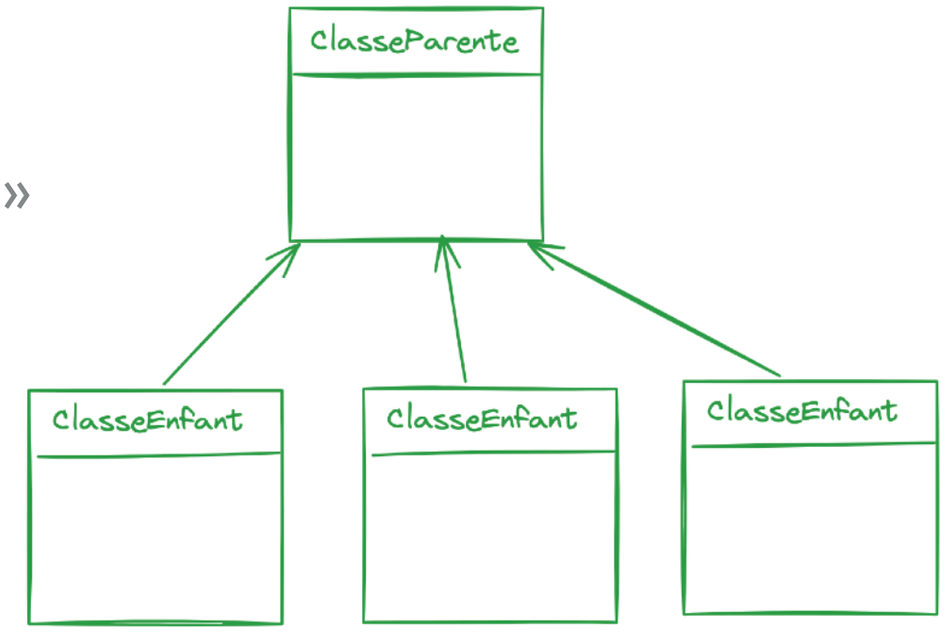
MISE EN PRATIQUE DE LA POO EN PHP 3/6

Auteur : Sébastien Inion

Codes sources support : https://replit.com/@Sebastien11/Point-et-geometrie?v=1

NOTION D'HERITAGE

- En POO l'héritage est un mécanisme qui permet de créer de nouvelles classes basées sur des classes existantes.
- La classe existante est appelée « classe parente »
- La nouvelle classe -> « classe enfant »



- Intérêts :
  - Réutilisation du code
  - Extensibilité : les classes dérivées peuvent ajouter de nouvelles fonctionnalités ou modifier des fonctionnalités existantes.
  - Polymorphisme : grâce à l'héritage, une classe dérivée peut parfois être traitée comme une classe de base, permettant une utilisation flexible des objets.

NOTION D'HERITAGE

- Lorsqu'on dit que la classe Commercial **hérite** de la classe Personnel, c'est que la classe Commercial **hérite** de tous les attributs et méthodes de la classe Personnel.

- Le mot clef en php est extends

- Si l'on déclare des méthodes dans la classe Personnel, et qu'on crée une instance de la classe Commercial, alors on pourra appeler n'importe quelle méthode déclarée dans la classe Personnel du moment qu'elle est publique.

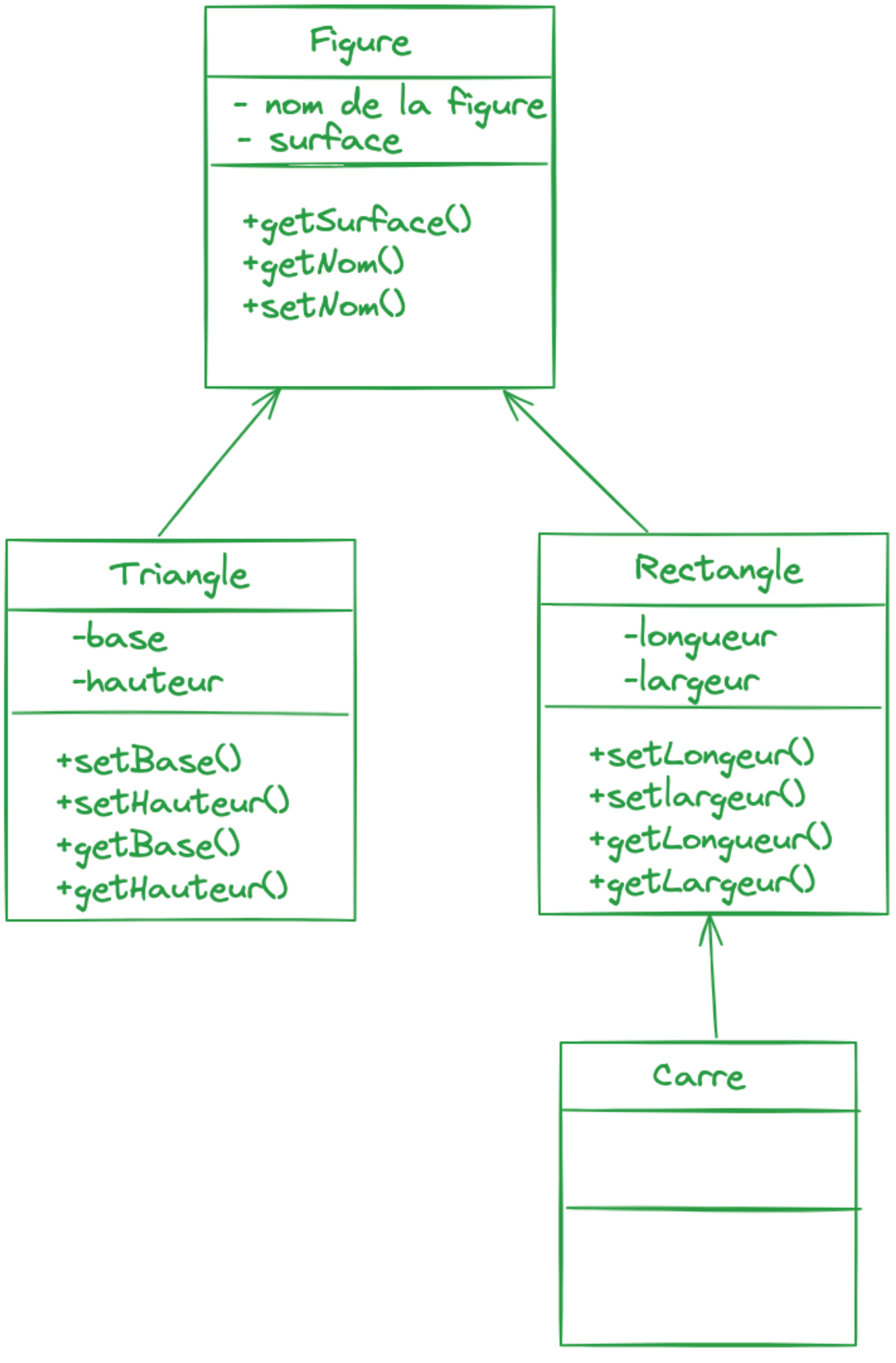
- En PHP une classe ne peut hériter que d'une classe !

- Attention pour faire hériter une classe d'une autre, on doit avoir une filiation et donc pouvoir dire « est un… »

```php
<?php
class Personnel {
    protected static $NbPersonne ;

    // -- Propriétés --
    private $_nom;
    private $_prenom;
    private $_age;

    public function __construct($n, $p, $a) {
        $this->_nom = $n;
        $this->_prenom = $p;
        $this->_age = $a;
        self::$NbPersonne++;
    }

    // Methode statique
    static function NbrPersonne() {
        return self::$NbPersonne;
    }
    // Methodes

    public function AffichePersonne() {
        echo self::$NbPersonne." Personne : ";
        echo $this->_nom."\t".$this->_prenom."\t".$this->_age;
        echo PHP_EOL;
        echo "<br/>";
    }
}

class Commercial extends Personnel {
    private $_CAArealiser;

    public function __construct($n, $p, $a, $ca) {
        parent::__construct($n, $p, $a);
        $this->_CAArealiser = $ca;
    }

    public function AffichePersonne() {
        parent::AffichePersonne();
        echo $this->_CAArealiser;
    }
}
// Programme
$salarie = new Personnel("Combette","Roland",50);
$salarie1 = new Commercial("Fereira","Antoine",32,5000);
echo $salarie1->AffichePersonne();
?>
```

REVENONS À NOS FIGURES GÉOMÉTRIQUES

MISE EN APPLICATION



NOTION D'ABSTRACTION

- On peut définir des classes comme modèle (pour l'héritage) mais on ne veut pas pouvoir créer d'objet de ce type. Dans ce cas on met abstract devant.
  On parle de classe abstraite !

- On peut faire la même chose pour les méthodes. L'intérêt est d'obliger le développeur à implémenter ces méthodes lors de l'héritage. De plus on connait déjà le nom ce qui uniformise et facilite l'utilisation des objets.
  On parle de méthode abstraite !

PROTECTED

- Les attributs ou les méthodes protected sont accessible depuis les classes qui en héritent.
- Si les attributs sont en private vous devrez passer par les méthodes publiques.

FIGURE

```php
<?php
abstract class Figure {
    protected $nom;

    public function __construct($nom) {
        $this->setNom($nom);
    }

    // Méthode abstraite que les sous-classes doivent implémenter
    abstract public function getSurface();

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

    public function setNom($nom) {
        $this->nom = ucfirst($nom);
    }

    public function __toString() {
        return "Je suis une figure nommée " . $this->getNom();
    }
}
?>
```

https://replit.com/@Sebastien11/HeritagePOO

RECTANGLE

```php
<?php
class Rectangle extends Figure {
    protected $longueur;
    protected $largeur;

    public function __construct($nom, $long, $larg) {
        parent::__construct($nom);
        $this->setLongueur($long);
        $this->setLargeur($larg);
    }

    public function getSurface() {
        return $this->longueur * $this->largeur;
    }

    public function setLongueur($long) {
        $this->longueur = $long;
    }
    public function setLargeur($larg) {
        $this->largeur = $larg;
    }
    public function getLongeur() {
        return $this->longueur;
    }
    public function getLargeur() {
        return $this->largeur;
    }

    public function __toString() {
        return parent::__toString() . " et je suis un rectangle avec une surface de " . $th
>getSurface()."<br>";
    }

}
```

MERCI POUR VOTRE ATTENTION