Héritage

Comme en Java, il n'y a **pas d'héritage multiple** en php.

```
class chien extends Animal{
    function affiche(){
        parent::affiche(); //référence vers la classe mère (statique ou non)
    }
}
```

Constructeur

Contrairement au java, ou il faut faire un appel explicite au constructeur parent, le php hérite de la méthode constructeur comme n'importe quelle autre :

```
function __construct($nom,$prenom,$age,$login){
    parent::__construct($nom, $prenom, $age);
    this->login = $login;
}
```

final

Une **fonction** déclarée dans le parent comme final ne pourra **pas être redéfinie** par les classes filles. Dans le même ordre d'idée une **classe** définie comme finale ne pourra **pas** avoir **de filles**.

final static

(ou constantes propres)

```
const MAX_AGE = 99;
```

Accès à une propriété statique

```
class Etudiant{
    static $nom;
    function hello(){
        echo "hello" . self::$nom;
    }
}
```

On utilisera à l'extérieur/intérieure de la classe :

```
Etudiant::$nom;
```

mais on peut aussi utiliser à intérieur de la classe :

```
self::$nom;
```

> this->nom ne fonctionne pas et les méthodes statiques vont générer un warning

Opérateur ::

il est utilisé dans les exemples ci dessus mais il peut aussi être utilisé comme suit :

```
$nomDeClasse = 'Etudiant';
$nomDeClasse::$AgeMaximum;
```

Traits

Les traits sont des moyens de mettre en commun des éléments commun à une classe ensemble (et aussi de faire du multi-héritage). un trait peut être abstrait.

On peut aussi utiliser un trait pour en définir un autre.

Créer

```
trait message {
  public function msg() {
    echo "Bonjour, je suis un message dans un trait";
  }
}
trait monTrait {
  public function bob() {
    echo "Bob";
  }
}
```

Utilisation

```
class Welcome {
  use message,monTrait;
}
```

Priorité des fonctions

Si une fonction est définie dans un trait, dans la classe mère et redéfinie dans une classe, on suivra l'ordre suivant.

1. Redéfinition dans la classe
2. Définition dans le trait
3. Définition dans la classe mère

Renomer une fonction ou propriété d'un trait

```
class Welcome {
  use monTrait{
      monTrait::bob as NomMoinsStupide;
  }
}
```

Une même fonction ou propriété dans Différents traits

C'est interdit, sauf si on renomme l'une des 2 méthodes/propriétés.

```
trait message {
  public function msg() {
    echo "Bonjour, je suis un message dans un trait";
  }
}
trait message2 {
  public function msg() {
    echo "Bonjour, je suis un message dans un autre trait";
  }
}

class welcome{
    use message,message2{
        message::msg insteadof message2; //cette fonction reste msg
        message2::msg as msg2 //cette fonction devient msg2
    }
}
```