

Etienne LANGLET.

9h00 - 12h30

14h00 - 17h00
30

Administratif.

Emargement: $\frac{1}{2}$ journée

Questionnaires: 1^{er} jour: Mesure des connaissances initiales

2^e jour: Mesurer mes acquis
Satisfaction.

IMPULSE Sign In

Code accès: OKX196

Un environnement de développement pour PHP.

→ Env. d'exécution.

→ Serveur Web : Apache, IIS, Nginx, ...

→ PHP : Support dans le serveur Web

→ Base de données :

↳ Solution 1 : Installer les produits manuellement.

↳ En production

↳ Solution 2 : Packages prêts à l'emploi

↳ En Dev, En test

Un environnement de développement pour PHP. (suite)

Editeurs de texte:

- Notepad++
- Atom
- Sublime Text

Environnement de développement intégré:

- | | | |
|---|---|--|
| <ul style="list-style-type: none">• PHPStorm• Eclipse for PHP Developers• NetBeans• Visual Studio Code | } | <ul style="list-style-type: none">• Coloration syntaxique• Assistance à la saisie du code• Débogage• |
|---|---|--|

LOCALHOST ?

"Espace Web"

http://localhost → C:\xampp\htdocs

URL

Page d'accueil. → index.php

http://localhost/site1 → C:\xampp\htdocs\site1

Constantes et variables

1/ Constantes

⊗ Déclaration.

1)

```
define('MA_CONSTANTE', 'valeur');
```

2)

```
const MA_CONSTANTE = 'valeur';
```

⊗ Utilisation

```
echo MA_CONSTANTE;
```

2/ Variables

⊗ Déclaration

```
$var1 = "Une chaîne";
```

⊗ Utilisation

```
echo $var1;
```

Types de données

2. Types simples (4)

- Entiers *int, integer* → 10
- Réels *float* → 3.14
- Chaînes de caractères *string* → "Chaine"
'Chaine'
- Booléens *bool, boolean* → TRUE, FALSE
true, false

2. Types composés (2)

- Tableaux *array*
- Objets *object*

3. Types spéciaux (2)

- Ressources *resource*
- NULL

Tableaux : Déclarations.

⊗ Numériques

0 →	"un"
1 →	"deux"
2 →	"trois"

`$tab = array("un", "deux", "trois");`
`$tab = ["un", "deux", "trois"];` // PHP 5.3 >

⊗ Associatifs : Paires "Clé → Valeurs"

"un"	1
"deux"	2
"trois"	3

`$tab = array("un" => 1, "deux" => 2, "trois" => 3);`
`$tab = ["un" => 1, "deux" => 2, "trois" => 3];`

Tableaux: Manipulation

→ Accès à un élément. ^{indice}

`$tab[1]`

`$tab['cle']`

→ Modifier un élément

`$tab[1] = 'vn';`

`$tab['cle'] = 'valeur';`

→ Parcours.

<code>foreach(\$tab as \$element) {</code>	<code>foreach(\$tab as \$cle => \$val) {</code>
<code>....</code>	<code>....</code>
<code>}</code>	<code>}</code>

Tableau multi-dimensionnel.

'FRANCE'	0	'Paris'
	1	'Nantes'
	2	'Lyon'
'ITALIE'	0	'Rome'
	1	'Venise'

⊗ Afficher 'Lyon' :

echo \$tab['FRANCE'][2];

⊗ Afficher le tableau

FRANCE
Paris
Nantes
Lyon.
ITALIE
Rome
Venise

```
// Solution 1 : Décomposer
$tabFrance = ['Paris', 'Nantes', 'Rennes'];
$tabItalie = ['Rome', 'Venise'];
```

```
// 1. A
$tab = ['FRANCE' => $tabFrance, 'ITALIE' => $tabItalie];
```

```
// 1. B
$tab['FRANCE'] = $tabFrance;
$tab['ITALIE'] = $tabItalie;
```

```
// Solution 2 : Une seule instruction
$tab = [
    'FRANCE' =>
        ['Paris', 'Nantes', 'Rennes']
    ,
    'ITALIE' =>
        ['Rome', 'Venise']
];
```

Parcourir le tableau :


```
<?php
// On a besoin des clés et des valeurs
foreach($tab as $pays => $tabVilles) {
    echo $pays . "<br />";
    // On a pas besoin des clés
    foreach($tabVilles as $ville) {
        echo $ville . "<br />";
    }
}
?>
```

Opérateurs :Egalité :

$\$a = 1 ;$ $\$b = "1" ;$
 Valeur seulement
 $\$a == \b → VRAI !
 $\$a === \b → FAUX !
 Valeur & type !

Ternaire :

```

if ( $a > 0 ) {
    $reponse = "Ok";
}
else {
    $reponse = "Erreur";
}
  
```

```

$reponse = ($a > 0) ? "Ok" : "Erreur";
  
```

Structures de contrôle

→ Conditionnelles :

if (else, ...) / switch (case, default) / match → PHP 8

→ Itératives :

while / for / do ... while / foreach

Inclusions de fichiers

include ()

include_once ()

require ()

require_once ()

Si le fichier à inclure n'est
pas trouvé :

Renvoie FALSE

Génère une Fatal Error

Fonctions



⊗ Déclaration

```
function ma_fonction ( ) {
```

```
// Corps / Implémentation.
```

```
}
```

Paramètres (variables locales)

⊗ Paramètres / Retour

```
function ma_fonction ( $param1, $param2 ) {
```

```
.....
```

```
return $resultat; }
```

```
}
```

Retour

Paramètres facultatifs.

↳ Ils possèdent une valeur par défaut !

```
function ma_fonction ($a = 0, $b = 0) {  
    ...  
}
```

```
ma_fonction();           // $a = 0 , $b = 0  
ma_fonction(1);          // $a = 1 , $b = 0  
ma_fonction(1, 2);        // $a = 1 , $b = 2
```


"Type Hint" - PHP 7.

↳ Typer les paramètres et les retours de fonctions.

```
function division(int $operande1, int $operande2) : float {  
    ...  
}
```


Programmation Orientée Objet

↳ SmallTalk, Simula (1980)

↳ C++ (1983)

Outils de modélisation: UML (Unified Modeling Language)

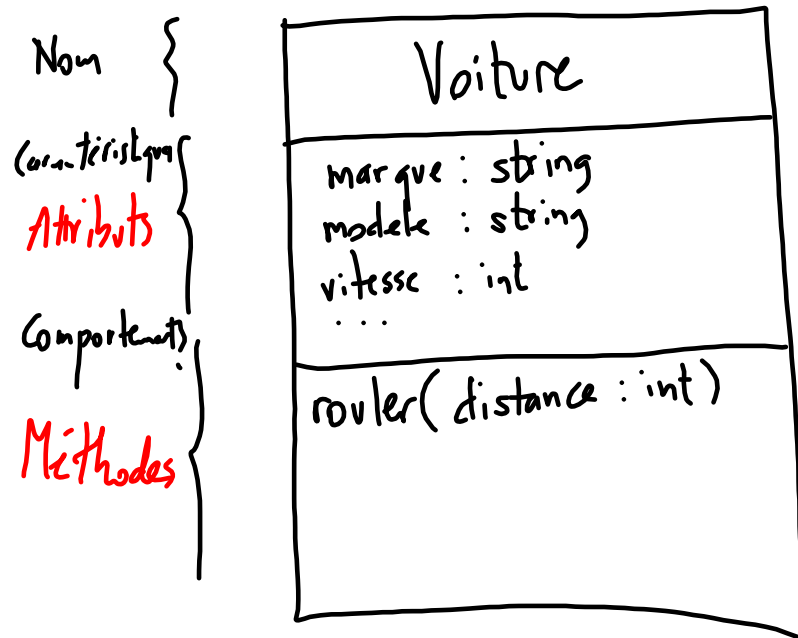
Méthodologies de G.P : Agiles (Scrum)

Théorie de l'objet.

Objet: Entité identifiable du monde réel qui possède des caractéristiques et des comportements.

Classe: Définir la structure d'un objet.

Représentation d'une classe en UML:



Créer un objet, une instance

$\underbrace{\$pub}_{(1)} = \underbrace{new\ Publication() ;}_{(2)}$
(3)

1. Référence
2. L'instance créée
3. L'adresse de la zone mémoire de l'instance est stockée dans \$pub

Accéder aux membres d'un objet

Membres = Attributs + Méthodes

\$ref → attribut

\$ref → méthode()

Le constructeur

⊗ A l'instanciation:

\$pub = new Publication();

Paramètres...

↳ Constructeur

⊗ La déclaration:

function __construct () {

}

Concepts objets.

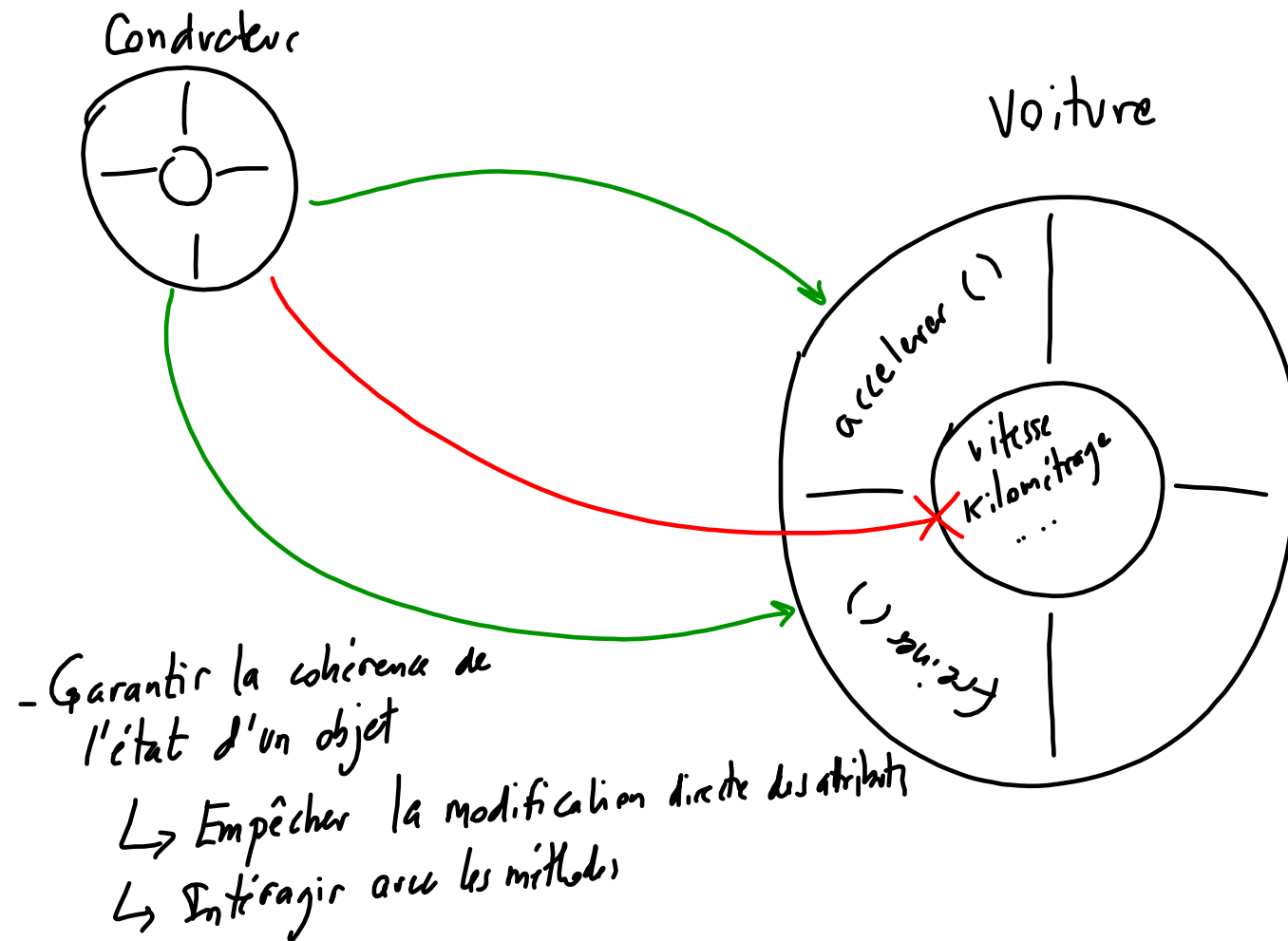
→ Classe / Objet.

→ Encapsulation

→ Héritage

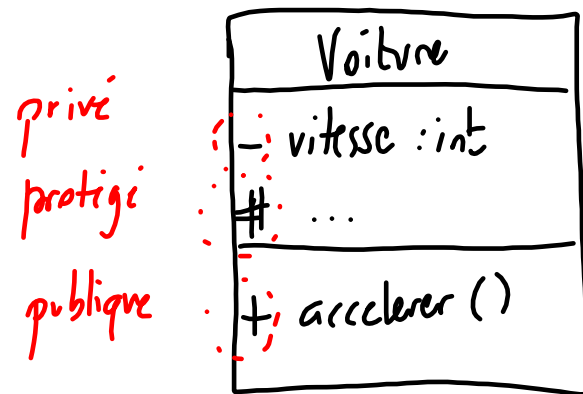
→ Polymorphisme

Encapsulation.



Encapsulation

En UML:



En PHP:

```
private int $vitesse;  
  
public function accelerer () {  
  
}
```


Accesseurs

↳ Méthodes dont le nom est conventionné.

Ex.: `private int $vitesse;`

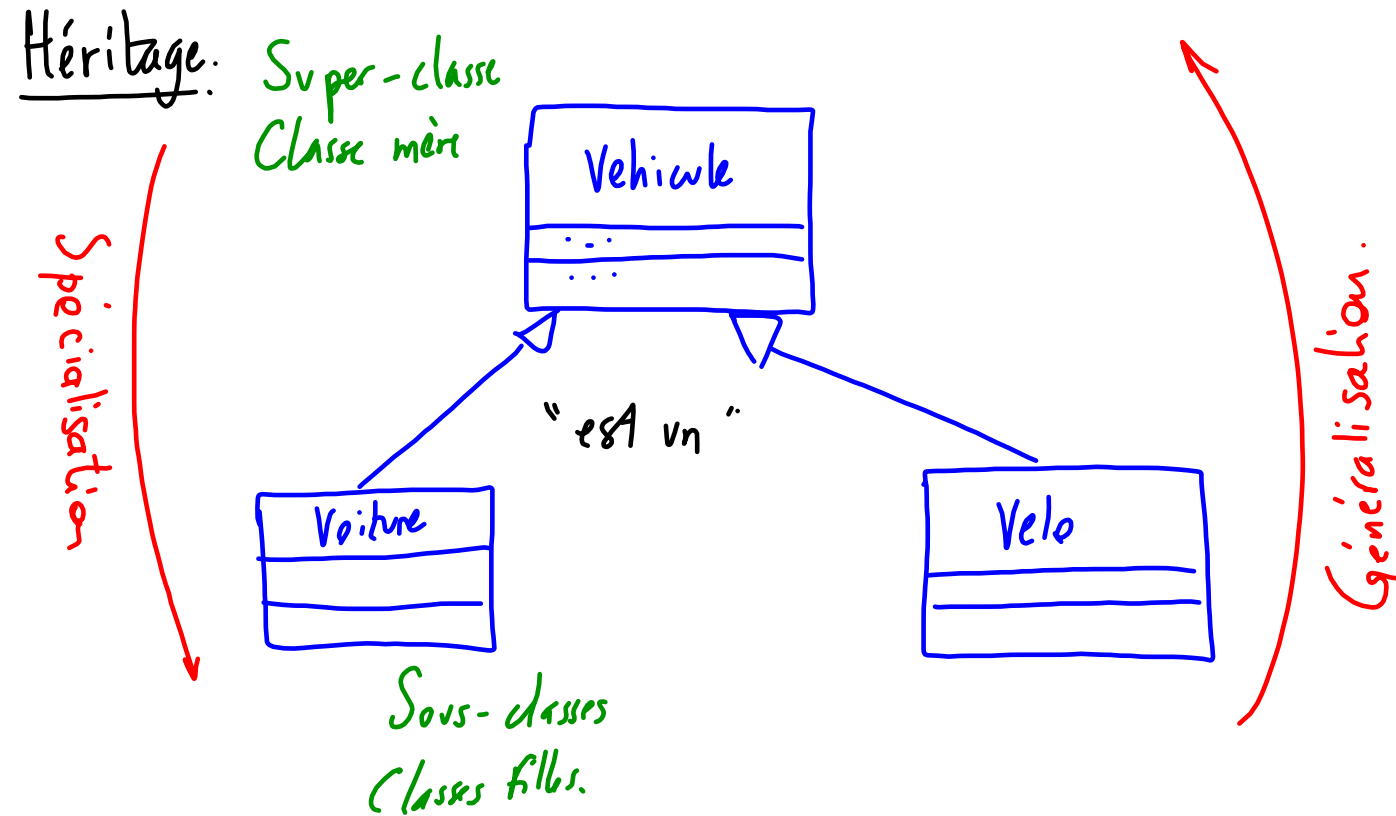
En lecture (getter):

```
public function getVitesse() : int {  
    return $this->vitesse;  
}
```

En écriture (setter):

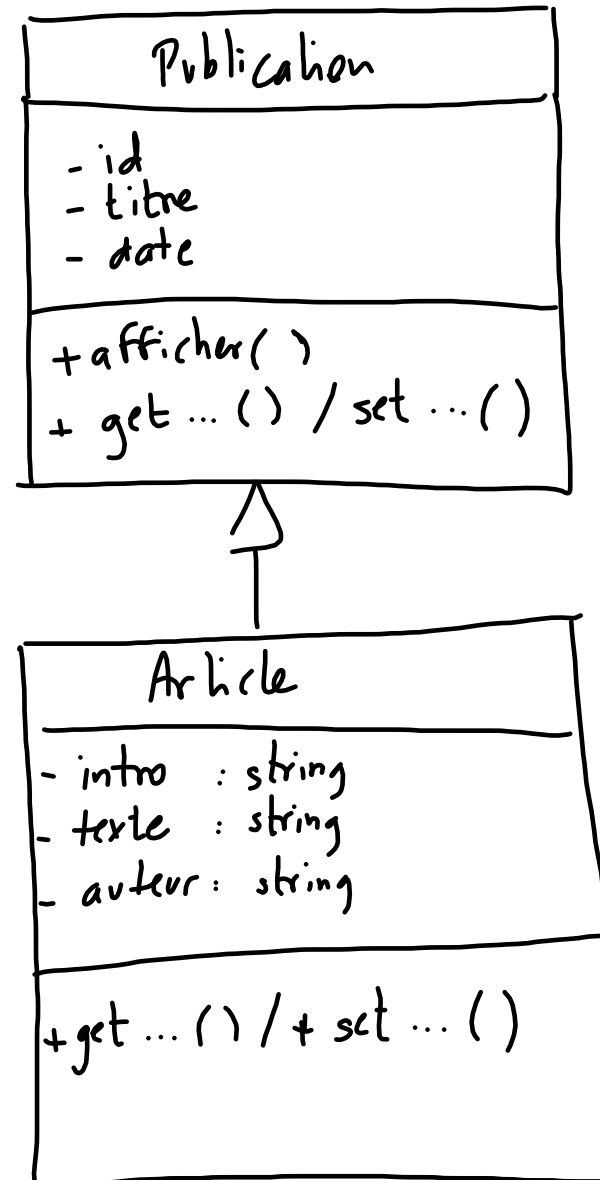
```
public function setVitesse(int $vitesse) : void {  
    if ($vitesse > 0 ... ) {  
        $this->vitesse = $vitesse;  
    }  
}
```

Ne retourne rien.



Héritage : mise en oeuvre

```
class Article extends Publication {  
}
```



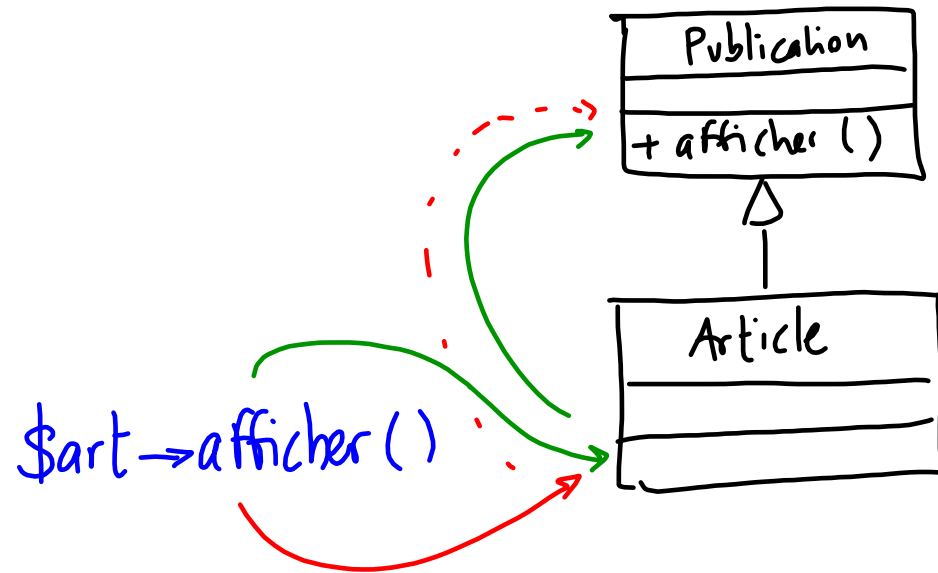
Héritage et construction d'objets

→ On doit prendre en charge la construction complète d'objet !

⊗ Appeler le constructeur de la super-classe dans celui de la sous-classe.

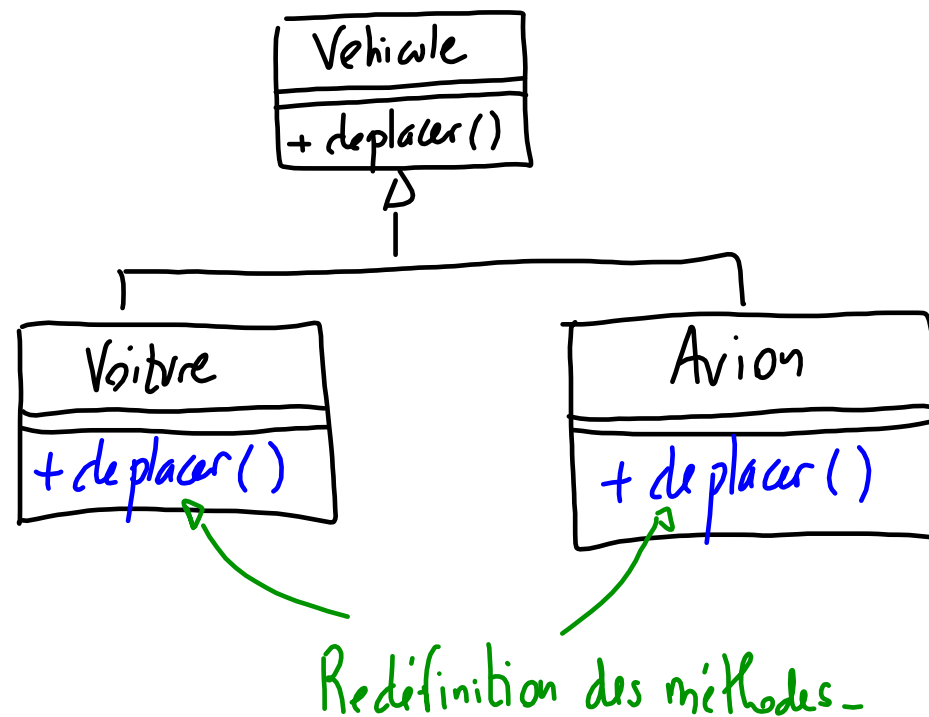
```
public function __construct(int $id, string $titre, string $date, string $intro, string $texte, string $auteur) {  
    // On appelle le constructeur de la super-classe pour initialiser les attributs hérités.  
    parent::__construct($id, $titre, $date);  
    $this->intro = $intro;  
    $this->texte = $texte;  
    $this->auteur = $auteur;  
}
```

⊗ Accéder aux attributs hérités à condition qu'ils soient visibles dans la sous classe
i.e. : Déclarés *protected*

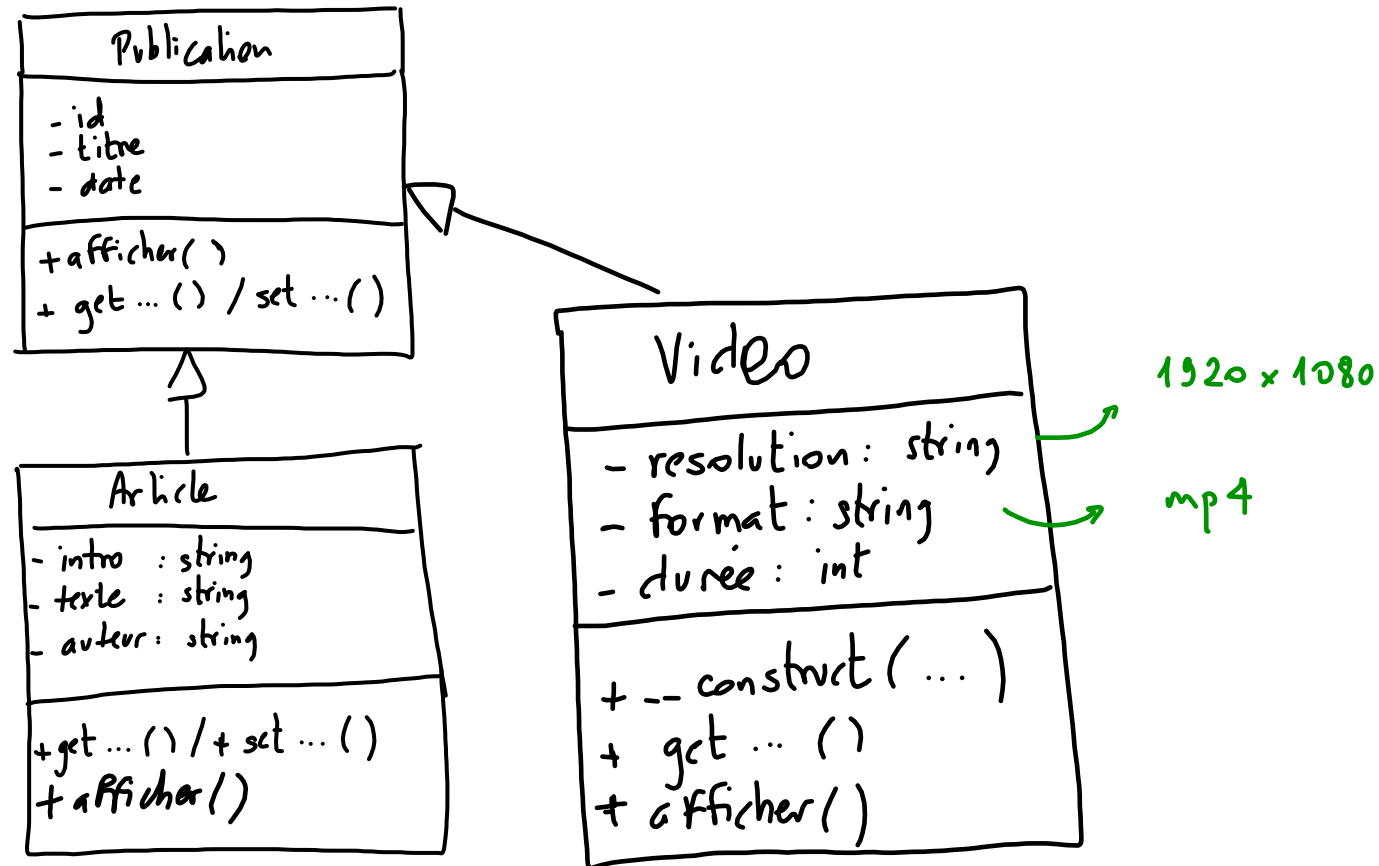


Polymorphisme

↳ 1 même méthode, mais des comportements \neq



Héritage & Polymorphisme : Mise en œuvre.



Classes & Méthodes abstraites

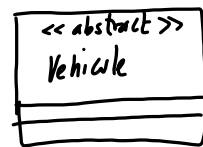
⊗ Classes abstraites

↳ Ne peut être instanciée

↳ Modèle pour l'héritage.

↳ Peuvent posséder des méthodes abstraites.

En UML :



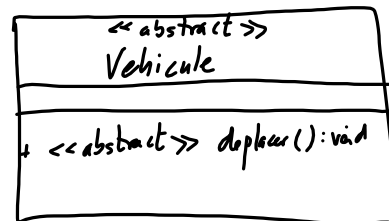
En PHP :

```
abstract class Vehicule {
}
```

⊗ Méthodes abstraites.

↳ Simplement déclarées, mais non-implementées.

En UML :



En PHP :

```
abstract class Vehicule {
    public abstract function deplacer() : void ;
}
```

Les sous-classes récupèrent donc cette méthode abstraite!
Elles doivent donc :

- être déclarées abstraites

ou

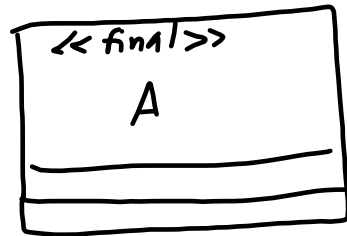
- Redéfinir la méthode et l'implémenter.

Classes et méthodes finales.

⊗ Classes finales.

↳ Ne peuvent avoir de sous-classes -

En UML:



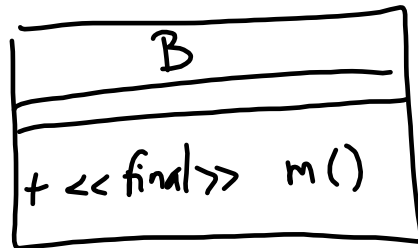
En PHP:

```

final class A {
}
  
```

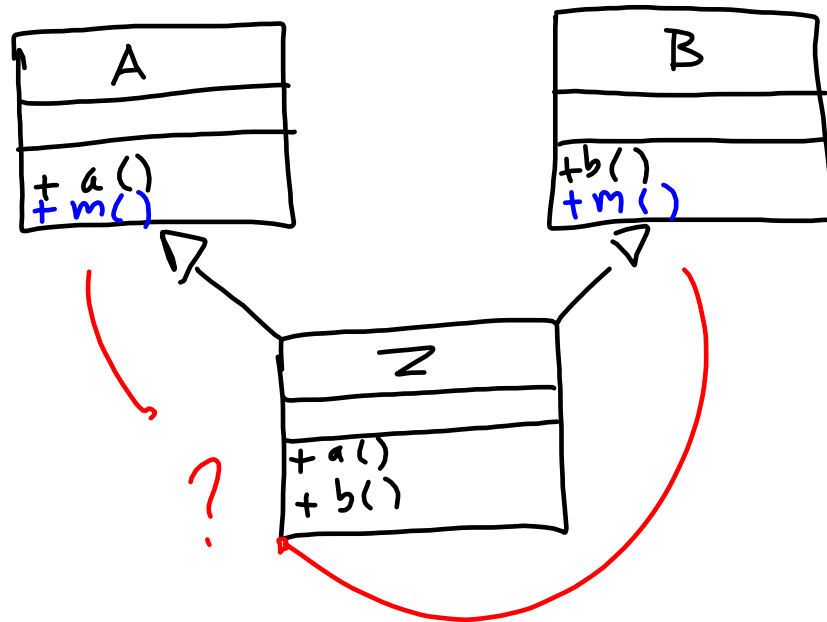
⊗ Méthodes finales

↳ Ne peuvent être redéfinies dans les sous-classes -



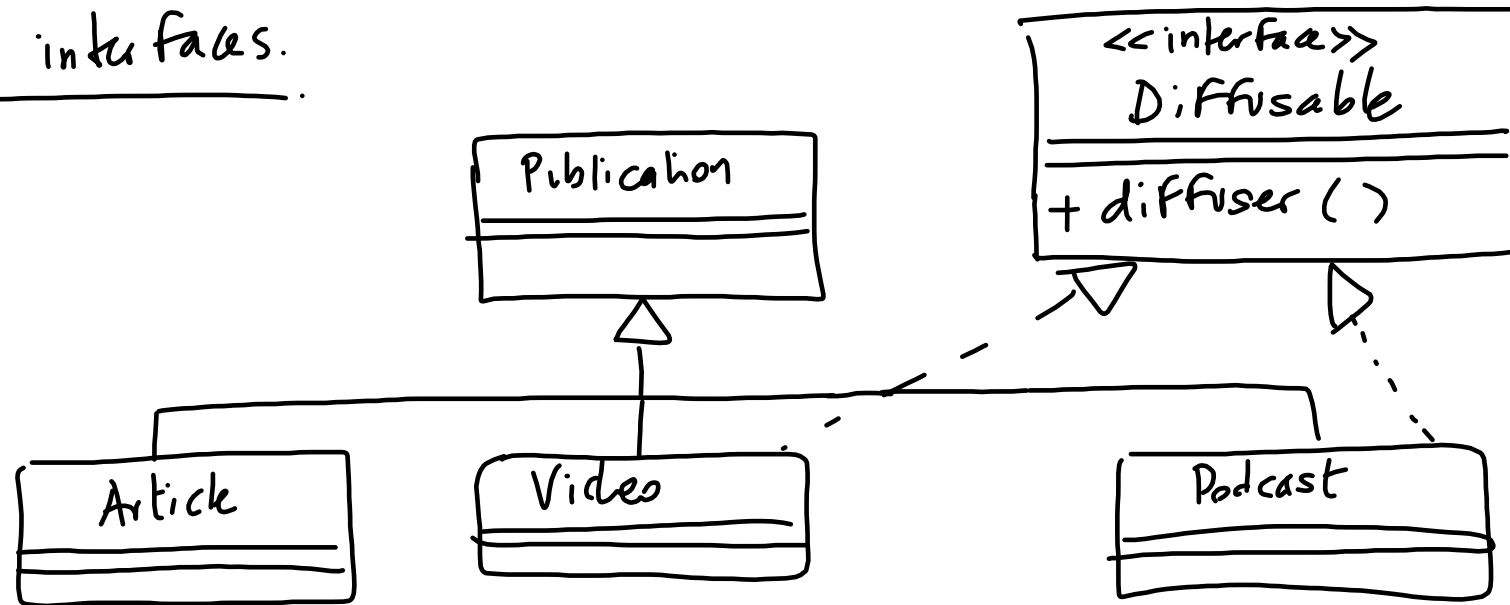
```

class B {
    public final function m() {
        ...
    }
}
  
```

Héritage multiple.

PHP n'autorise pas l'héritage multiple !

Les interfaces.



Une interface ne possède que des méthodes abstraites.

```
interface Diffusable {  
    function diffuser() : void;  
}
```

```
class Video extends Publication  
implements Diffusable {  
}
```

Tester le type d'un objet.

↳ instanceof

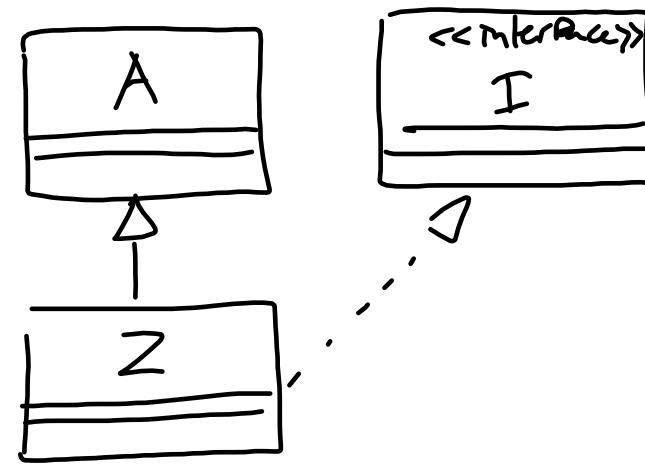
\$var instanceof Classe ↗ true
↘ false

\$z = new Z();

\$z instanceof Z true

\$z instanceof A true

\$z instanceof I true



class Z extends A implements I {
 }

Membres de classes

Membres = Attributs + Méthodes.

Membres de classes \neq Membres d'instances

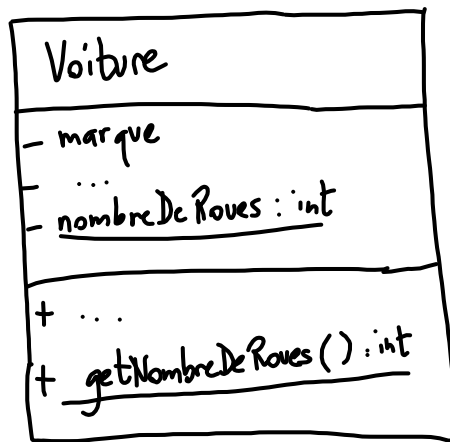
↪ ::

↪ Stockés dans la classe

↪ Propre au modèle : Commun à toutes les instances -

↪ Propres à chaque instance objet

En UML :



En PHP :

```

class Voiture {
    private static int $nombreDeRoues = 4;
    ...
    public static function getNombreDeRoues() : int {
        return self::$nombreDeRoues;
        // ou :
        // return Voiture::$nombreDeRoues;
    }
}
  
```

En dehors de Voiture :

```
Voiture::getNombreDeRoues();
```

Accès aux bases de données

→ Fonctions natives

↳ Spécifiques à chaque bases.

{ mysql - ...
oci - ...
pgsql - ...

→ PDO: PHP Data Object (PHP 5.3)

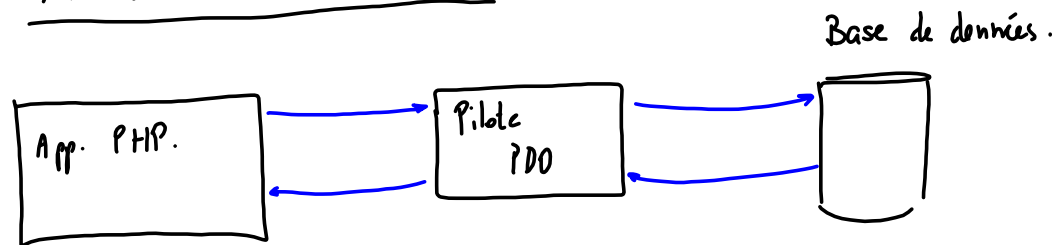
↳ Couche d'accès unifiée aux bases de données

PDO

3 classes:

- ① PDO : Etablir la connexion
- ② PDOStatement : Commande
- ③ PDOException : Erreurs-

API unifiée d'accès aux données :



Activer le pilote PDO nécessaire → php.ini

```
php - Bloc-notes
Fichier Edition Format Affichage Aide
extension=exif ; Must be after mbstring as it depends on it
extension=mysqli
;extension=oci8_12c ; Use with Oracle Database 12c Instant Client
;extension=oci8_19 ; Use with Oracle Database 19 Instant Client
;extension=odbc
;extension=openssl
;extension=pdo_firebird
;extension=pdo_mysql
;extension=pdo_oci
;extension=pdo_odbc
;extension=pdo_pgsql
;extension=pdo_sqlite
;extension=pgsql
;extension=shmop

; The MIBS data available in the PHP distribution must be installed.
; See http://www.php.net/manual/en/snmp.installation.php
;extension=snmp
<
```

Mise en œuvre avec MySQL.

→ Créer la base de données

→ Créer un compte d'accès à la base de données

→ Créer le schéma

} 1 étape avec
phpMyAdmin

Modèle de programmation PDO.

1. Connexion à la base de données (Classe PDO)
↳ Chaîne de connexion : spécifique à la base
2. Expression de la requête SQL
2 bis. Préparation de la requête (Classe PDOStatement)
3. Exécution de la requête
4. Exploitation des résultats.
5. Fermeture des ressources

Structure d'un jeu d'enregistrement PDO

```
$resultat = $cnx->query($sql);
```

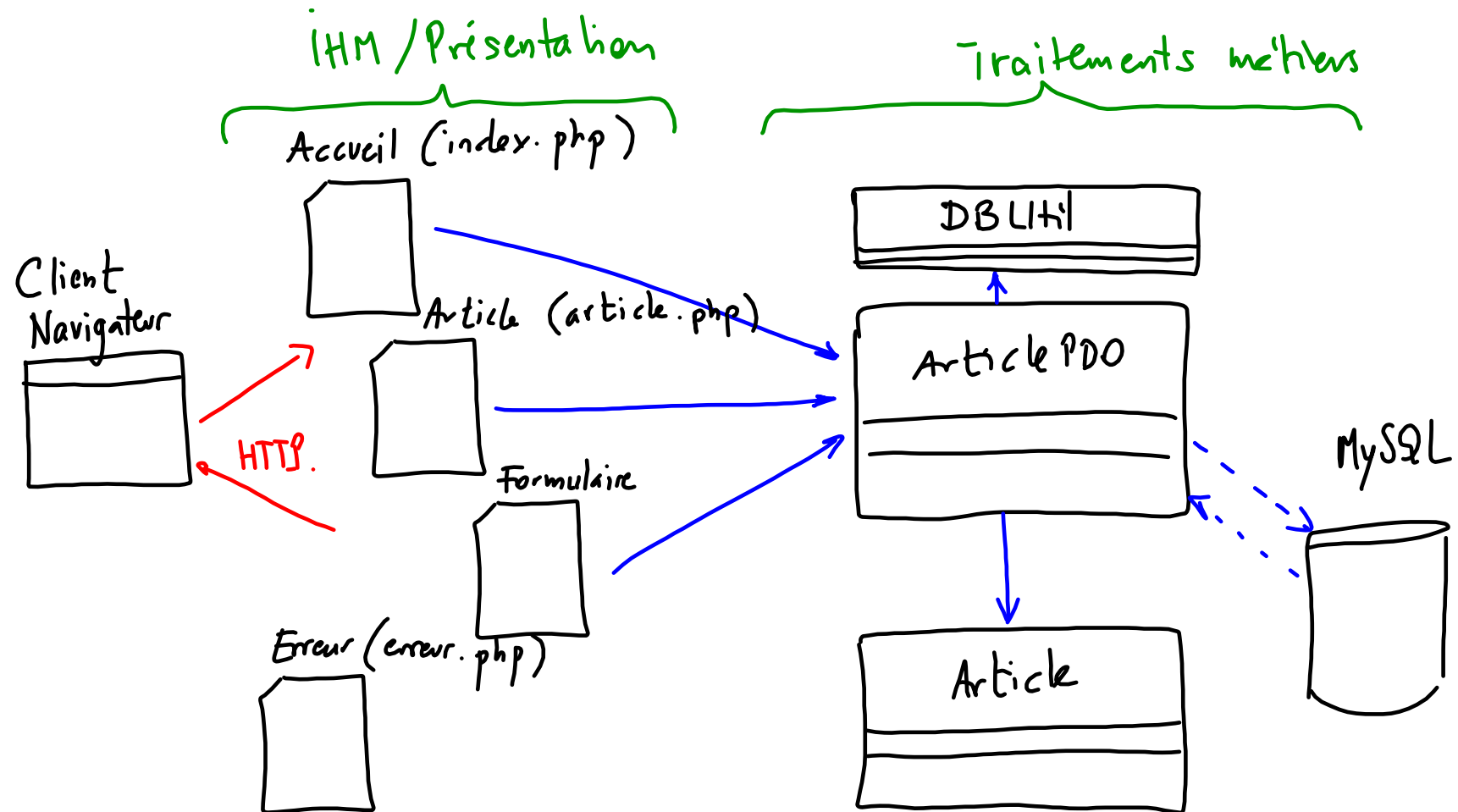
Ø	"id"	1
	"titre"	" "
	⋮	⋮

Notion de fetch

enreg. → fetch → FALSE

Ø	"id"	1
	"titre"	" ... "
1		

Structure de l'application



Le protocole HTTP.

Navigation Web → 2 méthodes HTTP.

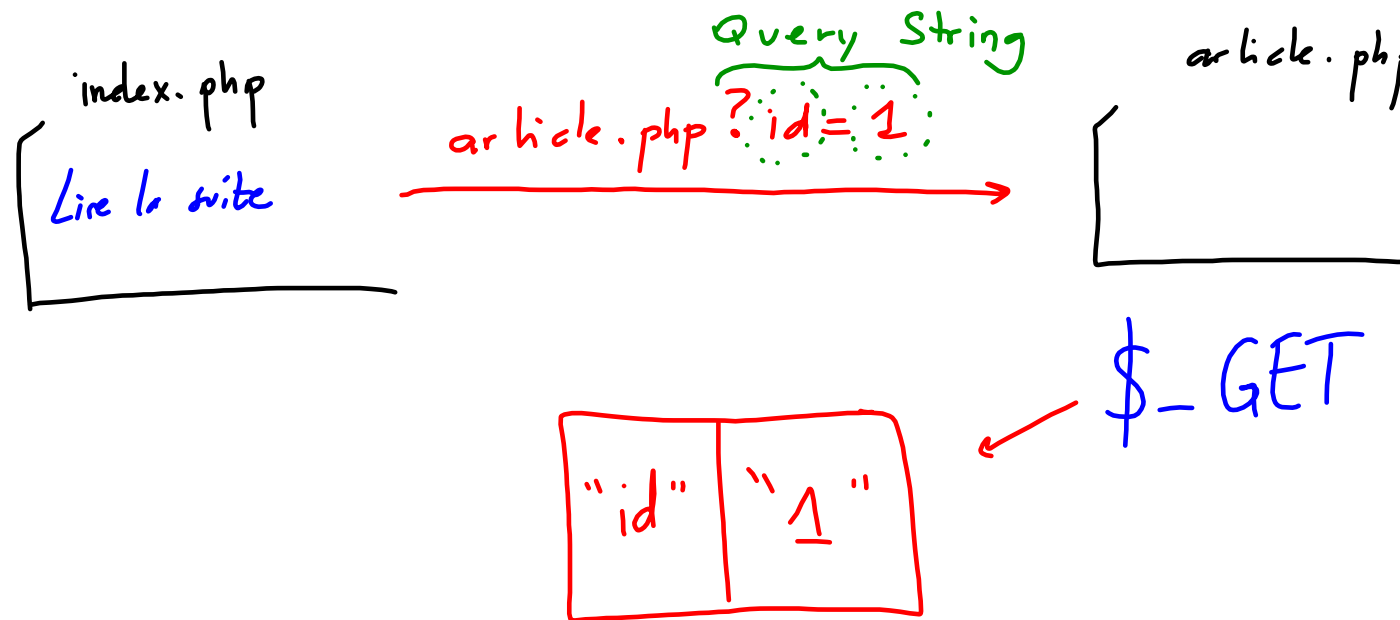
⊗ GET : Demande de ressource.

- URL dans la barre d'adresse
- Clique sur un lien

⊗ POST : Envoi de données à une ressource.

- Via un formulaire.

Transmettre des informations via l'URL



Transmettre des informations via un formulaire.

rediger.php

A hand-drawn form titled 'rediger.php' enclosed in a blue rectangular border. It contains four text input fields stacked vertically, each preceded by a label: 'Titre:', 'Intro:', 'Texte:', and 'Auteur:'. Below the 'Auteur:' field is a button labeled 'Enregistrer'. To the right of the 'Enregistrer' button are three vertical dots. A red curved arrow originates from the 'Enregistrer' button and points towards the 'enregistrer.php' section on the right.

enregistrer.php

1. Récupérer les données transmises.
2. Créer un objet 'Article'
3. Insérer l'article
4. Rediriger vers la page d'affichage de l'article.

`$_POST["auteur"]`

↓
`<input type="text" name="auteur"/>`

Redirection HTTP.

↳ Envoyer l'entête "Location" dans la réponse. Il vaudra l'url vers laquelle rediriger.



Location: page.php

Envoyer un entête en PHP:

```
header('Location: ~~~~');
```

Mise en oeuvre : Authentification des auteurs.

1. Créer la table auteur
2. Créer la classe Auteur
3. Créer la classe Auteur PDO
4. Créer le formulaire d'authentification

Mise en œuvre : Formulaire d'authentification.

→ Affichage & traitement avec 1 seul script: authentication.php

⊗ Affichage : en GET

⊗ Traitement : en POST

On regarde si les clés du tableau \$_POST sont présentes:

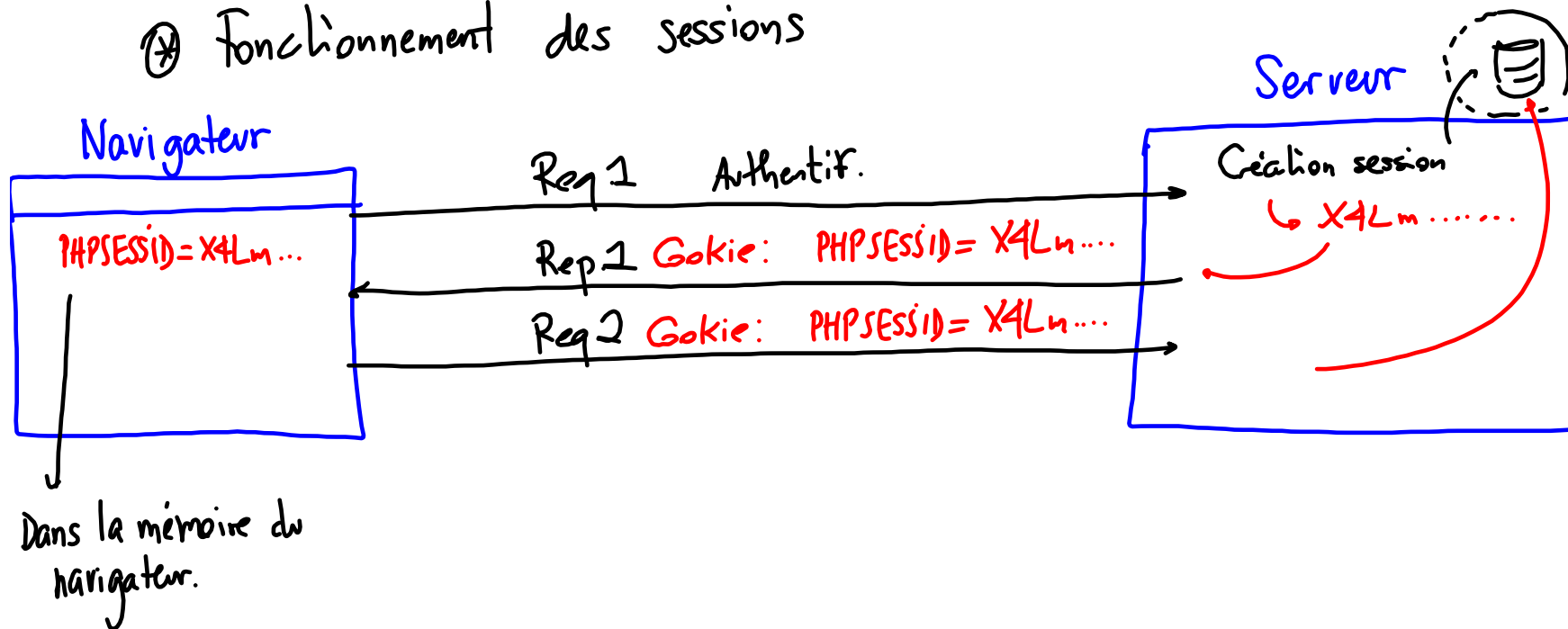
- Oui : POST

- NON : GET


Session HTTP.

→ Pouvoir mémoriser des informations utilisateur sur le serveur.
Ex. : le fait d'être authentifié, un panier d'achat, ...

⑧ Fonctionnement des sessions



Gérer les sessions en PHP

`session_start()` → Créer ou récupérer une session
 L'appel à `session_start()` doit se faire obligatoirement avant l'envoi de données HTML au navigateur.

`session_destroy()` → Détruire une session.

`$_SESSION` → Stocker des données dans la session

 Disponible uniquement après un `session_start()`

⊗ Créer une session

→ Appel à `session_start()`

→ Stocker une données dans `$_SESSION`

* Vérifier qu'une session existe

→ Appel à `session_start()`

→ Vérifier que la donnée est présente dans `$_SESSION`

Transférer des fichiers sur le serveur

→ Le formulaire:

```
<form action="..." method="post" enctype="multipart/form-data">
```

```
<input type="file" name="..." />
```

```
</form>
```

→ Le fichier est transféré sur le serveur dans un rep. temporaire.

→ Dans le script de traitement:

~~\$_POST~~

\$_FILES

<div> <div>photo</div> <div> <input name="photo" type="file"/> </div> </div>	name	
	type	
	size	
	tmp-name	
	error	