ESMT
École Supérieure Multinationale des Télécommunications

Programmation Web Avancée

LPTI 3 - DAR

ESMT
École Supérieure Multinationale des Télécommunications

SOMMAIRE

Programmation Web avancée

I. POO avec PHP 7

ESMT
École Supérieure Multinationale des Télécommunications

POO avec PHP7

- Déclaration des classes et objets
- Destructeurs
- Constantes de classe
- Attributs et méthodes statiques
- Héritage et interface
- Les espaces de noms
- Auto-chargement de classes

Déclaration des classes et objets

```
class nom_classe {
//liste des attributs
   private  $attr1=0;
   protected $attr2;
   ...
//constructeur
   public function __construct() {
      //code constructeur
   }
   ...
//liste des méthodes
    public function methode1(){
      //code méthode1
   }
   ...
}
```

Déclaration des classes et objets

```
class Client{
  private $id;
  private $nom;
  public function __construct($id){
    $this -> id = $id;
  }
  public function getId(){
    return $this -> id;
  }
  public function getNom(){
    return $this -> nom;
  }
  public function setNom($nom){
      $this -> nom = $nom;
  }
}
```

Destructeurs

- Méthode appelée à la destruction de l'objet
- Utilisée généralement pour fermer proprement des ressources encore ouvertes : connexion à la base de données, fichiers ouverts..

```
class Foo{
  private $logFile;
  public function __construct(){
    $this -> logFile = fopen("/tmp/log","w+");
  }
  public function __destruct(){
    fclose($this -> logFile);
  }
}
```

Constantes de classe

- Elle se déclare par l'emploi du mot-clé **`const`**.
- On accède à une constante de classe en employant le double deux-point “ ::”.
  - `NomDeClasse::NOM_CONSTANTE`

```
class Foo{
   const DEFAULT_DATE='2015-01-01';
   function getDefaultDate(){
     echo self::DEFAULT_DATE;
   }
}
```

Attributs et méthodes statiques

- La classe Client possède un attribut statique $**count** et une méthode statique **getCount**. L'appel se fait par :
  - `Client::getCount() ;`

```
class Client{
  private static $count;
  public function __construct(){
    $this::$count++;
  }
  public function __destruct(){
    $this::$count--;
  }
  public static function getCount(){
    return self::$count;
  }
}
```

ESMT
École Supérieure Multinationale des Télécommunications

Héritage et Interfaces

- PHP autorise l'héritage simple mais pas multiple.
  - L'héritage est spécifié à l'aide du mot clé **extends**.
- Les interfaces sont déclarées à l'aide du mot clé **interface**.
  - Contrairement aux classes, les interfaces peuvent hériter de plusieurs interfaces mères
  - L'implémentation d'une interface par une classe se fait par le mot clé **implements**.
- Le mot clé **final** permet d'indiquer qu'une méthode ou une classe sont finales, c'est à dire que la méthode ne peut être surchargée et la classe ne peut être étendue.

ESMT
École Supérieure Multinationale des Télécommunications

Les espaces de noms

- Les **namespaces** ont pour but d'éviter les conflits qui peuvent se produire dans l'appellation d'une constante, d'une fonction ou d'une classe.
- L'espace de noms permet de restreindre la visibilité d'un élément à son seul espace.
- Deux éléments portant le même nom mais déclarés chacun dans un espace différent pourront cohabiter sans provoquer d'erreur.

Les espaces de noms

- **Déclaration**
  - En début de script avant toute instruction à l'aide du mot clé namespace.
  - Exemple 1 :

```
<?php

 namespace Libraries ;
```

  - Exemple 2 :

```
<?php

 namespace Libraries\Storage ;
```

Les espaces de noms

- **Déclaration**

```
<?php
 namespace Libraries\Storage ;
 class Personne {...}
 function fopen() {...}
 const MACONSTANTE = 1;
```

Les espaces de noms

- **Appel**

```
<?php
 namespace Libraries;
 $p= new Storage\Personne() ;
 $f= \Libraries\Storage\fopen() ;
```

Les espaces de noms

- **Appel**

```
<?php
 namespace Libraries;
 $p= new Storage\Personne() ;
 $f= \Libraries\Storage\fopen() ;
 $t=\fopen() ;
```

- \ représente le namespace global

Auto-chargement de classes

- **`__autoload()`**
  - Fonction invoquée par PHP pour charger un fichier de classe quand cela est nécessaire.
  - Doit être implémentée par le développeur

```
function __autoload($class) {
   $filename="classes/".$class.".class.php";
   if(file_exists($filename))
     include $filename ;
 }
```

Auto-chargement de classes

- **`spl_autoload_register()`**
  - Permet d'empiler plusieurs autoloaders

```
spl_autoload_register(function ($class) {
    $filename="module1/".$class.".class.php";
    if(file_exists($filename))
      include $filename ;
});

spl_autoload_register(function ($class) {
    $filename="module2/".$class .".class.php";
    if(file_exists($filename))
      include $filename ;
});
```

ESMT
École Supérieure Multinationale des Télécommunications

Programmation Web avancée

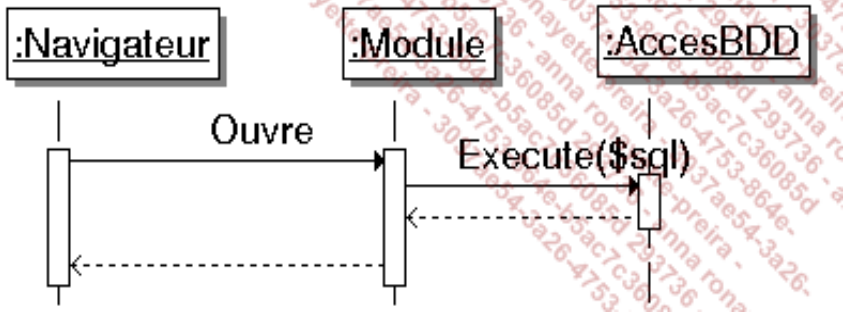
II. Accès aux données via PDO

Accès aux données via PDO

- La classe PDO
- Connexion à une source de données
- Construction et exécution de requêtes
- Requêtes préparées
- Exploitation des données

La classe PDO

- La classe PDO (*PHP Data Object*) implemente une interface unique de connexion à un serveur de bases de données, quel que soit son type : MySQL, Oracle, SQL Server, etc.
- Elle permet de prendre en charge les différentes commandes utilisées pour se connecter et exécuter les requêtes, et permet au développeur de s'affranchir du type de SGBD utilisé.
- Autres classes pour l'accès aux données
  - PEAR::MDB2
  - ADODB



Connexion à une source de données

- La connexion à la source est établie dès l'appel au constructeur de la classe PDO, qui prend en paramètres les informations de connexion (Data Source Name, login et password)
- `PDO::__construct()` émet une exception `PDOException` si la tentative de connexion à la base de données échoue.

```
$login = 'login';
$pwd = 'mdp';
$dsn = 'mysql:host=localhost;dbname=NomDeLaBase';
try {
$pdo = new PDO($dsn, $login, $pwd);
}
catch (PDOException $e) {
die("Erreur de connexion : " . $e->getMessage() );
}
```

Construction et exécution de requêtes

- La méthode `query($sql)` exécute une requête et retourne le résultat sous forme d'un `PDOStatement`.
- La méthode `exec($sql)` exécute une requêtes et retourne le nombre de lignes affectées On l'utilise pour des requêtes en modification de type `UPDATE, DELETE`, etc.

```
$sql = "INSERT INTO clients (id, nom) ".
       " VALUES (null,'PREIRA')";

if($pdo->exec($sql)){
    echo"Le client a été inséré";
}
```

Requêtes préparées

- Les requêtes préparées s'exécutent en deux ou trois temps , elles améliorent la sécurité et le temps d'exécution

  i. Préparation de la requête à l'aide de marqueurs nommés;

```
$sql = "INSERT INTO clients (id, nom) VALUES (:id,:nom)";
$rp=$pdo->prepare($sql) ;
```

  ii. Exécution de la requête par la méthode **execute** qui prend en paramètre un tableau associatif contenant les valeurs à affecter aux marqueurs;

```
$rp->execute(array('id'=>$_GET['id'],'nom'=>$_GET['nom']));
```

  - Une étape intermédiaire peut s'effectuer pour l'association des données aux marqueurs grâce à la méthode bindParam ;

```
$rp->bindParam(':id',$_GET['id'],PDO::PARAM_INT);
$rp->bindParam(':nom',$_GET['nom'],PDO::PARAM_STR);
```

  - Dans ce cas l'exécution par la méthode exécute se fait sans paramètres : `$rp->execute() ;`

Exploitation des données

- Le parcours du résultat d'une requête se fait par le biais de quatre méthodes de l'objet `PDOStatement` :
  - `fetch(fetch_style)` : renvoie la ligne de résultat suivant. Elle prend en paramètre fetch_style qui indique la forme que devront prendre les données retournées :
  - `fetchAll(fetch_style)` : renvoie un tableau de toutes les lignes de résultats.
  - `fetchColumn(column_number)` : renvoie la donnée de la colonne spécifiée en paramètre sur la ligne de résultat suivant.
  - `fetchObject(nom_de_la_classe)` : renvoie la prochaine ligne de résultat sous forme d'un objet de classe fournie en paramètre.

Exploitation des données

- Les valeurs possibles pour **fetch_style** sont :
  - `PDO::FETCH_NUM` renvoie un tableau de paires numéro_de_colonne => valeur.
  - `PDO::FETCH_ASSOC` renvoie un tableau nom_de_colonne => valeur.
  - `PDO::FETCH_BOTH` renvoie un tableau indexé sur les noms de colonnes et sur leur numéro.
  - `PDO::FETCH_OBJ` renvoie un objet ayant pour attributs les noms des colonnes retournées.

```
$sql = "SELECT * FROM clients";
$resultat=$pdo->query($sql) ;
while ($ligne = $resultat->fetch(PDO::FETCH_NUM)){
  echo"<td>$ligne[0]</td>
       <td>$ligne[1]</td>" ;
}
```

Programmation Web avancée

III. Echanges de données

Échanges de données

- Gestion de données JSON
- Gestion de données XML
- Web services : Modèle REST

Gestion de données JSON

- La fonction json_decode
  - La fonction PHP json_decode permet d'analyser une chaîne au format JSON et d'en convertir le contenu, tout en préservant les types d'origine.

```
<?php
$json='{"client":[
    {"id":1, "nom":"DIOP"},
    {"id":2, "nom":"FAYE"},
    {"id":3, "nom":"DIOUF"}
]}';
```

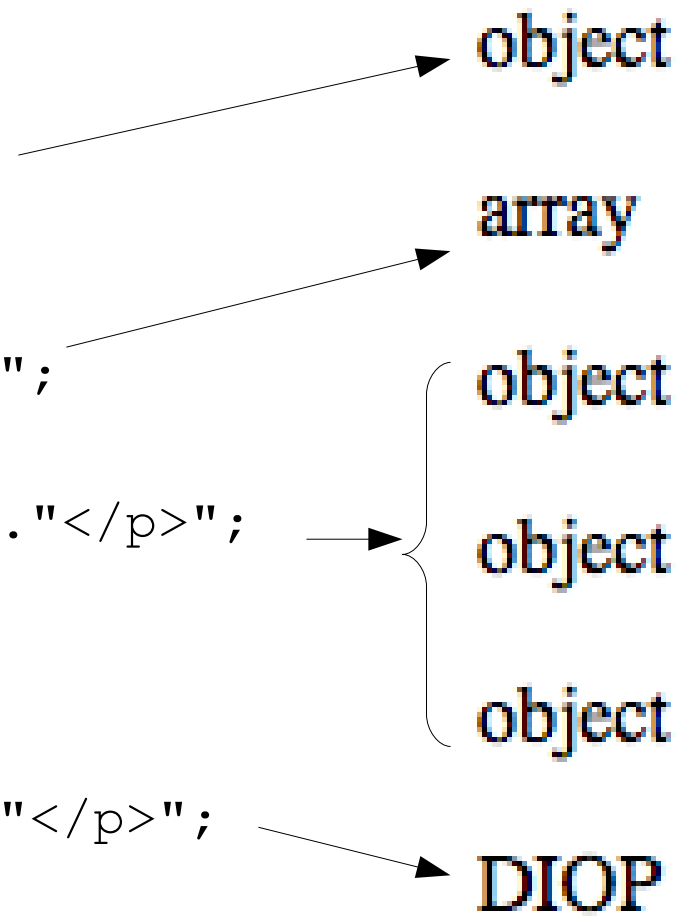
Gestion de données JSON

```
$parse=json_decode($json);

echo "<p>".gettype($parse)."</p>";

foreach($parse as $elt){
    echo "<p>".gettype($elt)."</p>";
    foreach($elt as $value){
        echo "<p>".gettype($value)."</p>";
    }
}

echo "<p>".$parse->client[0]->nom."</p>";
```



Gestion de données JSON

- La fonction json_encode
  - A l'inverse la fonction json_encode permet d'encoder n'importe quelle variable au format JSON

```
<?php
$client1=$parse->client[0];
echo json_encode($client1);
//Affiche  {"id":1,"nom":"DIOP"}
```

Gestion de données XML

- L'extension SimpleXML de PHP
  - Elle est activée par défaut et permet d'analyser le contenu XML structuré, en des structures PHP.
  - Les structures sont converties en objets de type SimpleXMLElement et les données scalaires en propriétés des objets SimpleXMLElement
  - La lecture d'un fichier xml se fait avec la fonction PHP `simple_xml_load_file`
  - La lecture d'une chaîne avec la fonction `simple_xml_load_string`
  - Les deux méthodes retournent un objet de type SimpleXMLElement.

Gestion de données XML

- L'objet SimpleXML
  - Cet objet représente un élément de contenu XML
  - Il possède des méthodes d'ajout, de parcours et de sauvegarde des données.
  - La méthode children(), renvoie/cherche l'enfant d'un élément
    - Fichier clients.xml

```
<?xml version='1.0'>
<clients>
    <client id="1" nom="DIOP"></client>
    <client id="2" nom="FAYE"></client>
    <client id="3" nom="DIOUF"></client>
</clients>
```

Gestion de données XML

```
<?php
$xml=simplexml_load_file("clients.xml");
echo"<p>".$xml->count()." enfants trouvés</p>";

$i=0;
foreach($xml->children() as $child){
    $i++;
    echo"<p>Enfant $i:</p>";
    foreach($child->attributes() as $key=>$value){
        echo"<p>$key=$value</p>";
    }
}
```

3 enfants trouvés

Enfant 1:

id=1

nom=DIOP

Enfant 2:

id=2

nom=FAYE

Enfant 3:

id=3

nom=DIOUF

Gestion de données XML

- Autres méthodes de l'objet SimpleXML
  - `addChild` : permet d'ajouter un élément enfant à un noeud
  - `addAttribute` : ajoute un attribut à un élément
  - `asXML` : retoune l'élément dans un format xml

```
$client4=$xml->addChil('client');
$client4->addAttribute("id",4);
$client4->addAttribute("nom","Diallo");

echo $client4->asXML();
//Affiche dans le code source <client id="1" nom="DIALLO"></client>

$xml->asXML("clients2.xml");//Ecrit dans le fichier spécifié
```

Web services : Modèle REST

- REST (Representational State Transfer)
  - Modèle d'architecture Web basé sur le principe de client-ressource
  - Le client accède à une ressource identifiée par son URI, et qualifie sa demande par un verbe HTTP correspondant aux 4 opérations CRUD
    - POST : créer/déposer une ressource
    - GET : récupérer/lire une ressource
    - PUT : remplacer/mettre à jour une ressource
    - DELETE : supprimer une ressource
  - Le format d'échange est libre : json, xml, ...

Web services : Modèle REST

- Exemples d'identification des ressources
  - Récupération (GET) : Liste des clients
    - http://monsite/clients/
  - Récupération (GET) : Client N° 3
    - http://monsite/clients/3/
  - Création (POST) : Nouveau client
    - http://monsite/client/create/
- Exmples d'API REST
  - https://restcountries.eu

Web services : Modèle REST

- Réécriture des URLs
  - Permet l'écriture d'URL plus “human-friendly”
  - Du type : monsite/page/action/identifiant/
    - site.com/produit/delete/11/
  - Elle peut se faire de deux façons
    - Au niveau du seveur dans un fichier .htaccess
    - En implémentant une classe Router en PHP pour le routage des URLs.

Web services : Modèle REST

- **Réécriture des URLs à partir du serveur Apache**
  - Le module `mod_rewrite` de Apache lorsqu'il est installé permet la réécriture des URLs par le serveur Apache lui même
  - Exemple de fichier `.htaccess` placé dans le dossier monsite

```
RewriteEngine On

RewriteRule ^clients$ liste_clients.php
```

  - L'URL http://monsite/clients/ renverra vers le script http://monsite/liste_clients.php

Web services : Modèle REST

- **Réécriture des URLs en implémentant un routeur**
  - Le fichier `.htaccess` extrait la requête et l'envoie au routeur (`index.php`) via une variable d'URL.

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ index.php?url=$1
```

Web services : Modèle REST

- **Réécriture des URLs en implémentant un routeur**
  - Le fichier `index.php`

```
$url = '';

if(isset($_GET['url'])) {$url = $_GET['url'];}

if($url == '') {require 'home.php';}

elseif(preg_match('#participant/([0-9]+)#', $url, $params))

{
  $id=$params[1];
  require_once('autoload.php');
  $participant=new Participant();
  echo json_encode($participant->details($id));

}else { require '404.php';}
```

Web services : Modèle REST

- **Réécriture des URLs en implémentant un routeur**
  - Les systèmes de réécriture et de routage sont pris en charge par la plus part des frameworks PHP :
    - Silex
    - Slim
    - CakePhp
    - Symfony
    - Laravel

Programmation Web avancé

IV. Le modèle MVC

ESMT
École Supérieure Multinationale des Télécommunications

Le modèle MVC

- La conception MVC
- Le modèle
- Le moteur de template
- Smarty
- Les frameworks PHP MVC

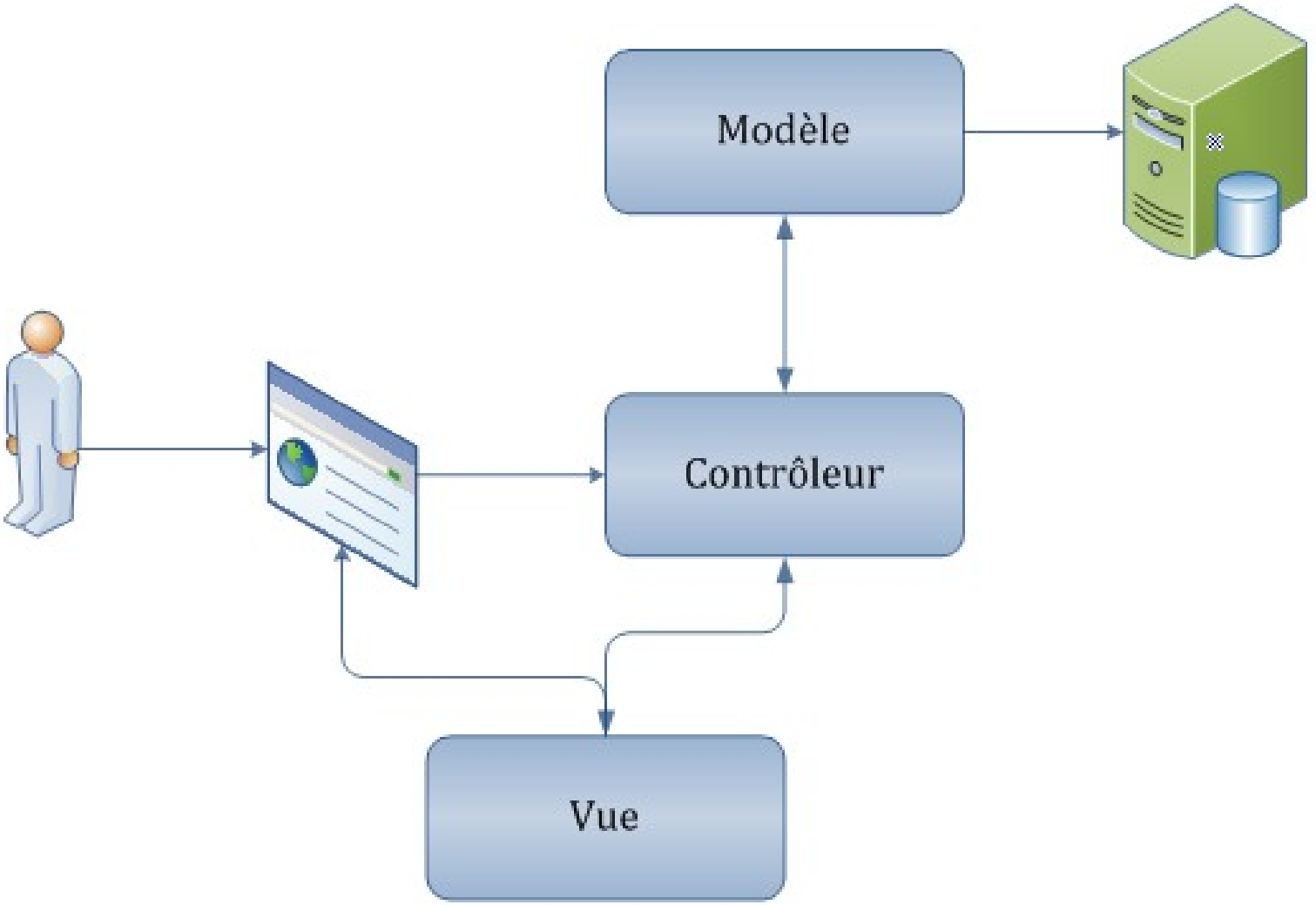
La conception MVC

- Le modèle de conception MVC est un ensemble de méthodes qui décrivent une manière de concevoir une application web selon le principe de la séparation des rôles.
- Il améliore la stabilité et la maintenance de l'application.
- Il repose sur 3 couches essentielles
  - Le modèle (M)
  - La vue (V)
  - Le contrôleur (C)

La conception MVC

- **Le Modèle** représente la partie “données” de l'application.
  - L'implémentation des classes d'accès aux données (bases de données, fichiers) et les interactions objets-données.
- **La vue** représente la couche de résultat et d'interaction avec l'utilisateur. Elle contient tout ce que voit l'utilisateur à l'écran.
- **Le contrôleur** sert à recueillir et traiter les requêtes utilisateur, et à initier les traitements adéquats selon leur nature.

La conception MVC



Le modèle

- Il représente toutes les classes de description des données, de gestion de celles-ci et de leur persistance.
- Il s'abstrait totalement des considérations d'affichage.
- Il est constitué d'objet-métier. A chaque table de la base de données correspond une classe métier.

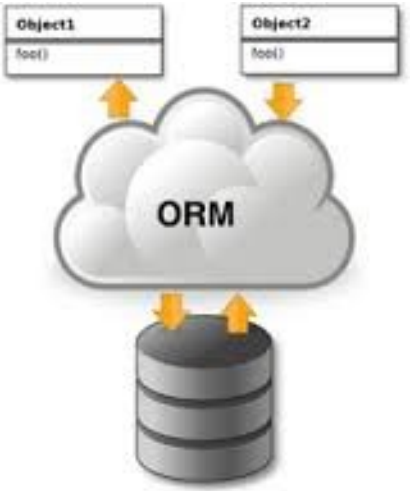
ESMT
École Supérieure Multinationale des Télécommunications

Le modèle

- Conception et gestion du modèle
  - À l'aide d'un ORM (Object Relational Mapping)
  - Ou suivant un Design Pattern
    - Active Record
    - Le couple DTO-DAO

Le modèle

- Object Relational Mapping
  - Est un couplage relationnel – objet, qui consiste à encapsuler dans une classe générique toutes les opérations de manipulation d'une table.
  - Une classe héritée de la classe de base est créée et les accès aux bases de données sont réalisés par l'intermédiaire d'un composant d'abstraction.
  - Nécessite l'apprentissage d'un pseudolangage SQL



Le modèle

- **ORM** : Bibliothèques connues
  - **DOCTRINE** : Elle s'appuie sur PDO pour les accès aux bases de données.
  - **METASTORAGE** : plus qu'une bibliothèque, c'est une application qui va générer automatiquement les classes d'accès aux données à partir d'un schéma XML.
  - **ZENDdbTable** : ce composant est intégré à Zend Framework. Il s'appuie sur PDO pour les accès aux données et dispose de fonctions avancées pour gérer les contraintes au sein de la base de données.

Le modèle

- Le Design Pattern DTO
  - **D**ata **T**ransfer **O**bject,
  - Objet métier,
  - est la représentation objet d'une table.
  - Ne contient que les attributs et les accesseurs correspondants.
  - Ne gère pas l'accès à la BD (pas d'objet PDO)

Le modèle

- Le Design Pattern DTO
  - Exemple

```
class Societe {
  private $id_societe ;
  private $nom_societe ;
  private $email_societe ;

  function __construct($id=null){ $this->id_societe=$id; }
  function getId(){ return $this->id_societe; }
  function getNom(){ return $this->nom_societe; }
  function setNom($nom){ $this->nom_societe = $nom; }
  function getEmail(){ return $this->email_societe; }
  function setEmail($email){ $this->email_societe=$email; }
  function hydrate(array $data){ //Méthode d'hydratation }
}
```

Le modèle

- Hydratation d'objets
  - Technique qui consiste à renseigner l'ensemble des propriétés d'un objet par l'appel d'une méthode unique.

```
public function hydrate (array $data){
  foreach($data as $field=>$value){
    if(property_exists($this,$field)){
      $this->$field=$value ;
    }
  }
}
```

Le modèle

- Hydratation d'objets
  - Technique plus recommandée : pour chaque clé du tableau appeler l'accesseur correspondant

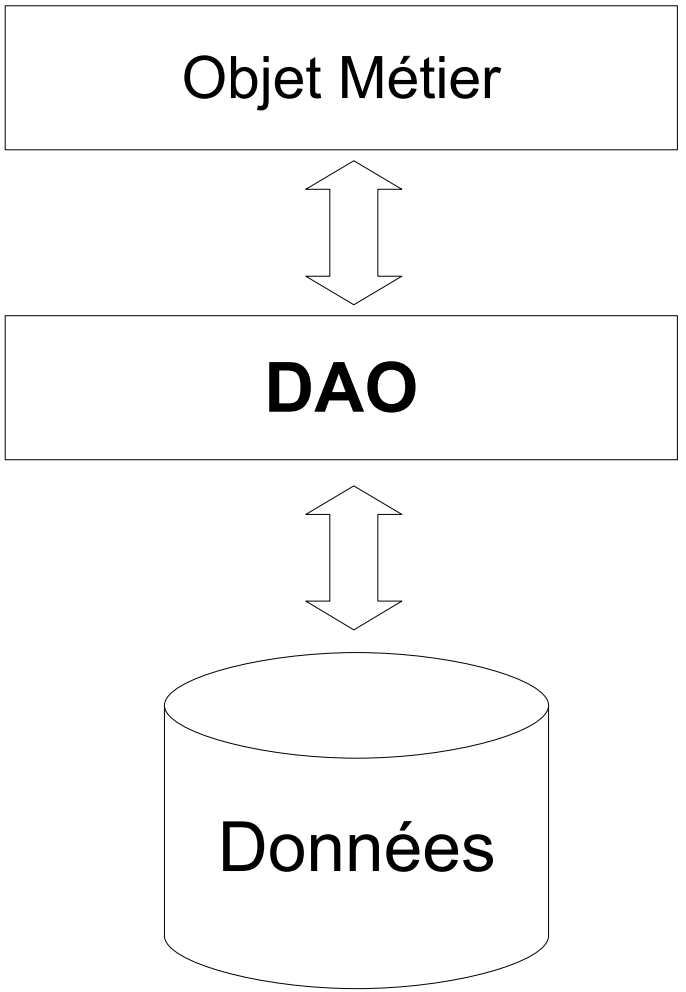
```
public function hydrate (array $data){
  if(isset($data['nom_societe'])){
    $this->setNom($data['nom_societe']) ;
  }
  if(isset($data['email_societe'])){
    $this->setEmail($data['email_societe']) ;
  }
}
```

Le modèle

- Le Design Pattern DAO
  - Data Access Object : sert d'interface entre l'objet métier et la source physique de données via PDO par exemple.
  - Contient les requêtes CRUD
  - Contient d'autres requêtes d'extraction

Le modèle

- L'association DTO-DAO



Le modèle

- L'association DTO-DAO
  - Exemple : classe `SocieteDAO` chargée du lien entre la classe `Societe` et la table societe.

```
class SocieteDAO {
  protected $db;
  function __construct() {
   $this->db = MyPDO::getInstance();
  }
  function insert(Societe $societe){
    $sql="insert into societe values(null,
      '$societe->getNom()', '$societe->getEmail()')";
    return $this->db->exec($sql) ;
  }
...
```

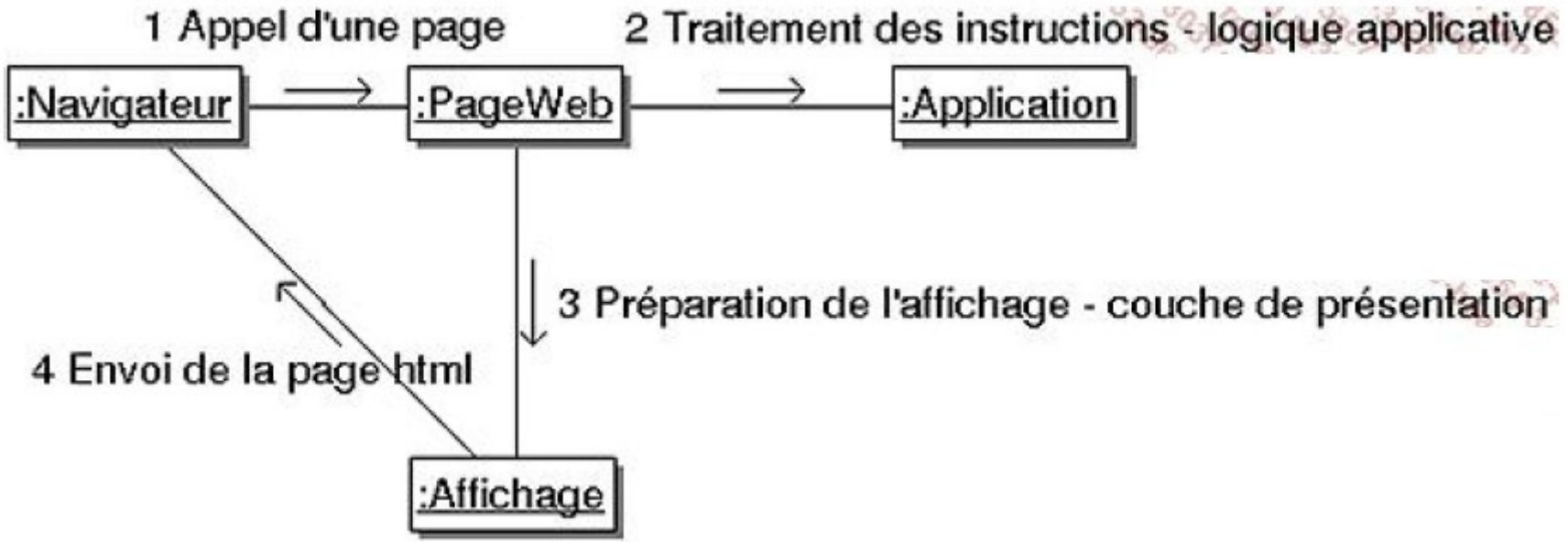
Le modèle

- L'association DTO-DAO
  - Suite : classe `SocieteDAO`

```
...
  public function read(Societe $societe) {
   $sql="select * from societe
         where id_societe='$societe->getId()'";
    $data=$this->db->query($sql) ;
    $societe->hydrate($data) ;
    return $societe
  }
  public function update(Societe $societe){...}
  public function delete(Societe $societe){...}
  public function getList(){...}
}
```

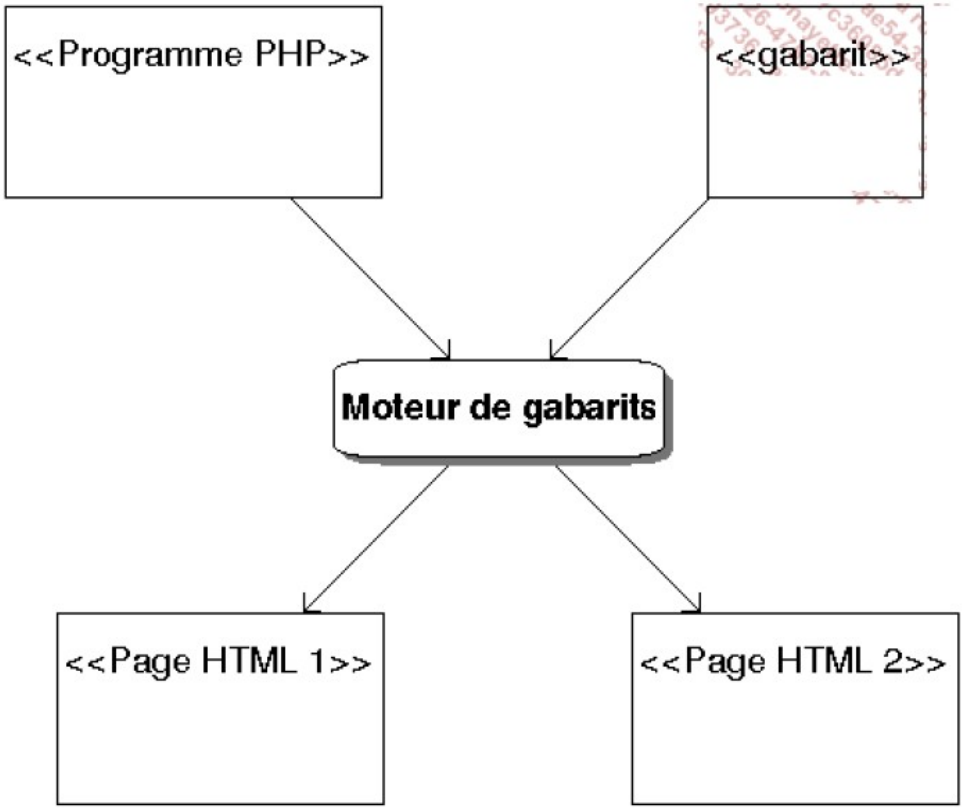
Le moteur de templates

- Système en charge de la génération de la vue
- Le moteur de templates est un composant qui permet d'isoler le code PHP de la couche présentation.



Principe des fonctionnement

- La préparation des données à afficher est réalisée dans les pages PHP.
- Le moteur de templates ne s'occupe que de la mise en forme et de la présentation des données à afficher.



ESMT
École Supérieure Multinationale des Télécommunications

Le moteur de templates

- Les systèmes de templates
  - Smarty (http://www.smarty.net)
  - TinyButStrong (http://www.tinybutstrong.com/fr/)
  - ModeliXe
  - Twig (Symfony)
  - RainTpl

Smarty

- Pour pouvoir utiliser Smarty :
  - Il faut d'abord charger le fichier contenant la classe Smarty :

```
include_once('Smarty-2.6.26/libs/Smarty.class.php');
```

  - Puis le paramétrer :

```
$SMARTY_template ='templates';
$SMARTY_template_c = 'templates_c';
$SMARTY_config = 'param/configs_smarty';
$SMARTY_cache_dir = 'smarty_cache';
$SMARTY_cache = FALSE;
```

  - Il au moins créer les dossiers templates et templates_c dans le dossier racine de l'application.
  - Les templates ou gabarits (.tpl) seront stockés dans le dossier templates.

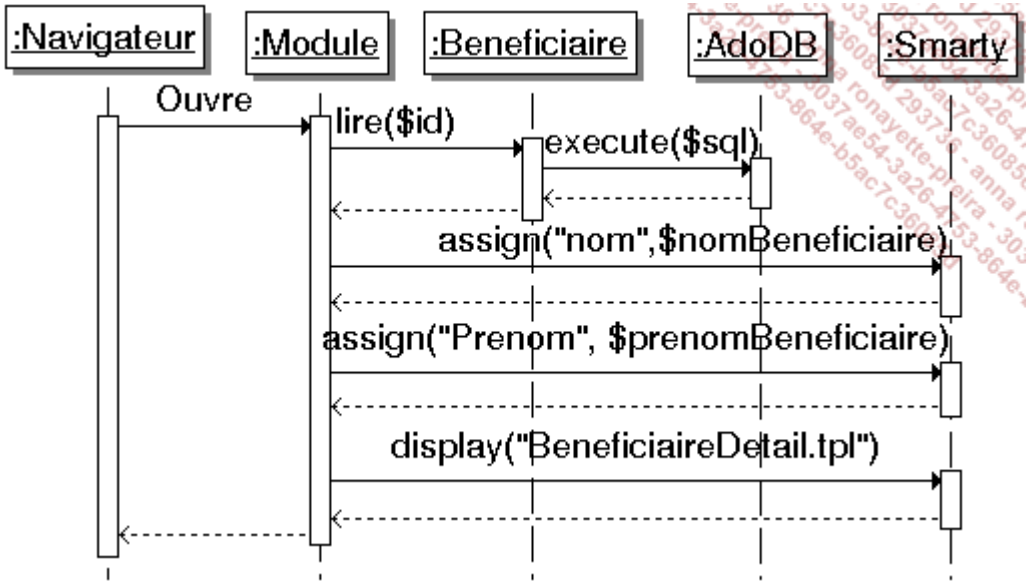
Smarty

- Paramétrage (suite)

```
if (!isset($smarty)){
    $smarty = new Smarty;
//Si on souhaite changer les valeurs par défaut
    $smarty->template_dir = $SMARTY_template;
    $smarty->compile_dir = $SMARTY_template_c;
    $smarty->config_dir = $SMARTY_config;
    $smarty->cache_dir = $SMARTY_cache_dir;
    $smarty->caching = $SMARTY_cache;
}
```

Smarty

- Utilisation de la classe Smarty
  - La classe Smarty dispose de deux méthodes indispensables :
    - **`assign()`**, va permettre de déclarer des variables.
    - **`display()`**, va charger le template à afficher.



Smarty

- Code PHP

```
require_once('autoload.php');
$partDAO=ParticipantDAO::getInstance() ;
$participant=$partDAO->read(new Participant($id)) ;
$smarty=new Smarty() ;
$smarty->assign("nom",$participant->getNom());
$smarty->assign("prenom",$participant->getPrenom());
$smarty->display("ParticipantDetail.tpl");
```

ESMT
École Supérieure Multinationale des Télécommunications

Smarty

- Création du template
  - Le template est, avant tout, une page HTML qui va contenir des balises spéciales.
  - Ces balises seront encadrées par les accolades { }.
  - Ainsi, pour afficher le contenu de la variable nom, il suffira d'indiquer dans le code de la page : `{$nom}`

Smarty

- Code du gabarit

```
<html>
<head>
<title>Test d'affichage des informations sur le participant</title>
</head>
<body>
Nom du participant : {$nom}
<br>
Prénom : {$prenom}
</body>
</html>
```

Smarty

- Gestion des tableaux
  - Code PHP

```
$sql="select * from participant where id='$id'" ;
$query=$pdo->query($sql);
$participant=$query->fetch(PDO::ASSOC);
$smarty=new Smarty();
$smarty->assign("participant",$participant);
$smarty->display("ParticipantDetail.tpl");
```

  - Code du template

```
<body>
Nom du participant : {$participant.nom}
<br>
Prénom : {$participant.prenom}
</body>
```

Smarty

- Gestion des boucles
  - Code PHP

```
$sql="select * from participant" ;
$query=$pdo->query($sql);
$listePart=$query->fetchAll(PDO::ASSOC);
$smarty=new Smarty();
$smarty->assign("listePart",$listePart);
$smarty->display("listeParticipants.tpl");
```

Smarty

- Gestion des boucles
  - Code du template

```
<table id='listePart'>
    <tr>
        <th>Nom du participant</th>
        <th>Prénom</th>
    </tr>
{section name=boucle loop=$listePart}
    <tr>
        <td>{$listePart[boucle].nom}</td>
        <td>{$listePart[boucle].prenom}</td>
    </tr>
{/section}
</table>
```

Les frameworks PHP MVC