

Module SIL3

PHP5 et architecture MVC  
pour les applications web

2016-2017

# Module SIL3

## Présentation du Module

# Contenu

---

- **Pré-requis** (idéalement)

1. Pages web (X)HTML, feuilles de styles CSS, Formulaires
2. Syntaxe de base de PHP
3. Base de données (MySQL, PostgreSQL,...)
4. Connaissance d'un langage objet (java, C++,...)

- **Plan du cours**

1. Rappels sur les grands principes de la programmation web
2. Présentation de PHP 5
3. Première étape de structuration du code
4. PHP 5 et l'accès aux Bases de Données
5. Principe d'une architecture MVC dans les applications web

- **Objectifs du module**

1. Acquérir des compétences en PHP 5
  2. Comprendre l'architecture MVC
-

# Organisation dans le temps

---

- **Semaine du 10/10**

**séance 1 : cours1** Programmation Web, PHP 5, première structuration du code & Installation d'un serveur web, application blog & image

**séance 2 : tp1** 1er développements pour l'application image

**séance 3 : cours2** Accès aux bases de données **tp2** Remplacement des fichiers par BD dans l'application image

- **Semaine du 31/10**

**séance 1 : cours3** Modèle MVC **tp3** Structuration MVC de l'application image

**séance 2&3 : suivi de projet** Structuration MVC de l'application image, ajout de fonctionnalités

- **Semaine du 23/11**

**séance 1 : suivi de projet**

**séance2 : examen papier**

**séance 3 : démo projet**

---

# Evaluation

---

- **Projet** (en binôme)
  - Développement de l'application image selon le modèle MVC
  - Note fonction de la quantité/qualité des développements et de la maîtrise du projet (questions lors de la démo)
- **Examen** (épreuve individuelle)
  - Questions de connaissances générales sur la programmation web, php5 et le modèle MVC (QCM)
  - Questions pratiques (éventuellement quelques petits bouts de codes à analyser et/ou à écrire)

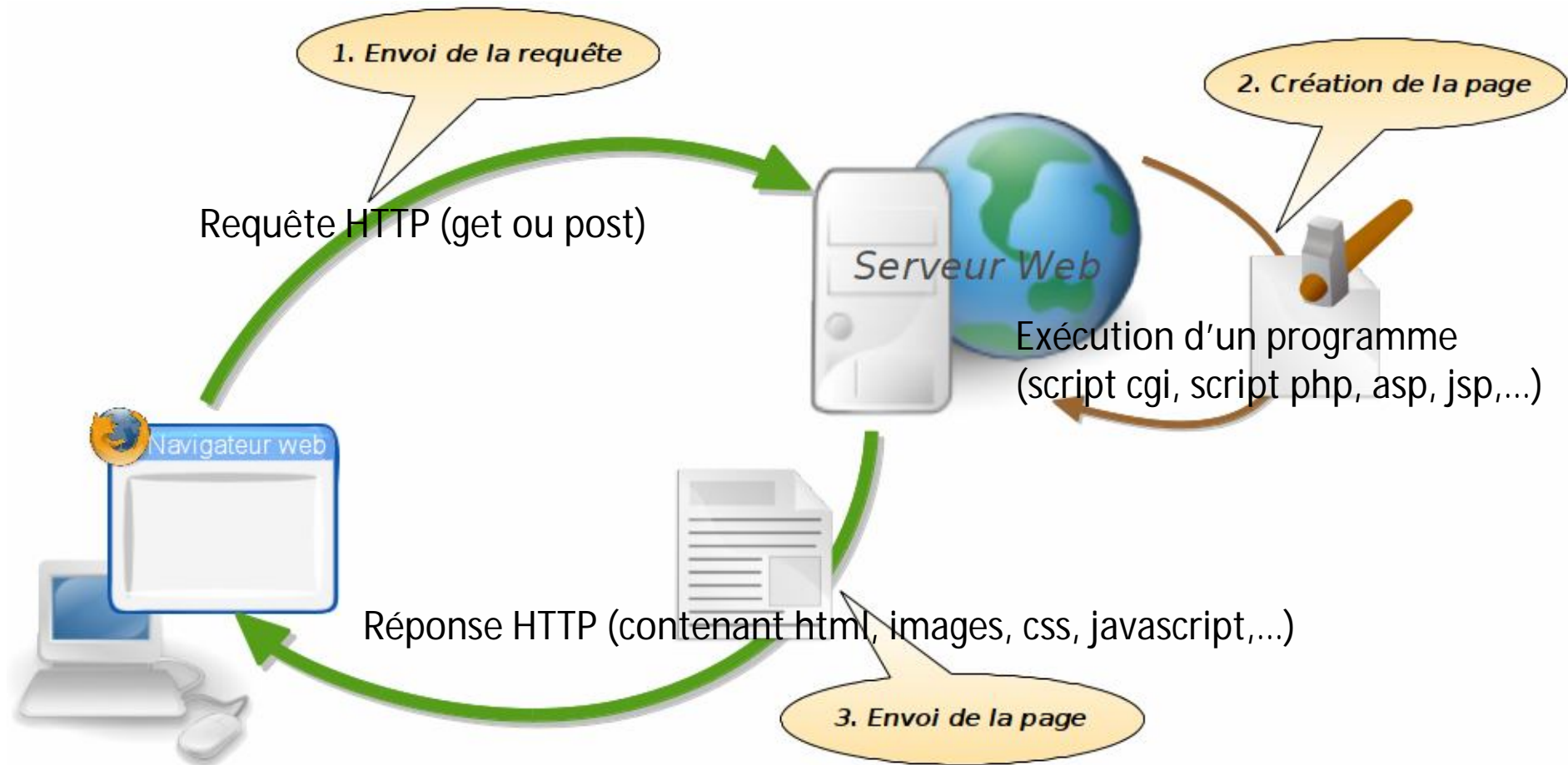
---

# Module SIL3

## Rappels

# Environnement d'exécution d'une application WEB

---



L'application Blog : un exemple d'application Web

---

# HTTP : Protocole sans état

---

- Echange HTTP : connection au serveur http, transfert de la page web, fermeture de la connection.
- A la connection suivante, pas moyen de savoir ce qui s'est passé lors des précédents échanges avec le même client. L'historique de la communication est perdu !
- Problème lors du développement d'une application quand le résultat d'une action dépend de l'historique
  - exemple des boutons « mode » et « suivant » du blog
  - exemple du scénario de type « commerce électronique »



# Solutions

---

1. Chaque page est un état de l'interface (ex : mode)
2. Ajouter des paramètres aux « get » (ex : suivant)
3. Stocker l'état côté client (variables javascript, cookies)
4. Stocker l'état côté serveur (variable, fichier/bd) + identifiant (côté client)  
correspond au principe des sessions dans les applications Web :
  - Un identifiant est associé à un client (stocké sous forme de cookie ou échangé dans l'url)
  - L'état est stocké côté serveur et les informations concernant l'échange avec le client sont stockés côté serveur

---

Module SIL3

Présentation de PHP 5

# PHP (Hypertext Preprocessor)

---

- Langage de script Open Source, interprété côté serveur :
  - ⇒ prévu d'emblée pour être intégré directement dans les pages web  
(code html <?PHP code php ?> code html <?PHP code php ?>...)
  - ⇒ permet la génération dynamique des pages web
- le fonctionnement couplé à un serveur web est l'application la plus répandue (mais peut être utilisé de façon autonome)
- langage extrêmement simple pour les néophytes, mais offre des fonctionnalités avancées pour les experts
- dispose de près de 3000 fonctions utilisables dans des applications très variées ; couvre pratiquement tous les domaines en rapport avec le web
- Créé en 1995 par Rasmus Lerdorf, Interpréteur en Perl, puis en C
- En 2007 : plus de 20 millions de domaines utilisent PHP

# PHP (Hypertext Preprocessor)

---

- Tire son origine de PHP/FI (1995, Rasmus Lerdorf)
  - PHP 3.0 (1998, Zeev Suraski et Andi Gutmans)
  - PHP 4.0 (2000, début des aspects objets)
  - PHP 5.0 (2004, modèle objet complet)
  - Actuellement PHP 7.0 (quelques modifications dans le langage et surtout un gain de rapidité de l'interpréteur)
- Pour connaître son installation : `phpinfo()`

⇒ Sites PHP:

<http://www.phpdebutant.org/>    <http://www.phpfrance.com/> ...

<http://www.php.net/manual/fr/>    <http://www.phpindex.com/> ...

# PHP 5 & 7

---

- **PHP 5** = Langage Orienté Objet (**LOO**) - modèle objet à simple héritage complet
- Par rapport à PHP4:
  - Réel langage objet
  - Gestion des exceptions et les contrôles de type (sûreté de programmation)
  - Un SGBDR intégré: SQLite
  - Existence d'un socle commun aux SGBD : PDO (PHP Data Object)
  - Plus performant (nouvel interpréteur "Zend Engine 2").

# PHP - Rappels

---

- Un Langage de programmation à part entière (utilisable en dehors du contexte Web)
- Une Syntaxe « à la C »
- Faiblement typé (4 types : booléen, entier, flottant, chaîne, la variable prend le type de la valeur affectée)

```
<?php
$a_bool = TRUE;    // un booléen
$a_str  = "foo";   // une chaîne de caractères
$a_str2 = 'foo';   // une chaîne de caractères
$an_int = 12;      // un entier

echo gettype($a_bool); // affiche : boolean
echo gettype($a_str);  // affiche : string

// Si c'est un entier, incrément de 4
if (is_int($an_int)) {
    $an_int += 4;
}
```

# PHP - Rappels

- Les tableaux

## Exemple #1 Un tableau simple

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
);

// depuis PHP 5.4
$array = [
    "foo" => "bar",
    "bar" => "foo",
];
?>
```

```
<?php
$array = array( 'premier' => 'N° 1', 'deuxieme' => 'N° 2', 'troisieme' => 'N° 3' );

foreach( $array as $key => $value )
    echo 'Cet élément a pour clé "' . $key . '" et pour valeur "' . $value . '"<br />';
?>
```

```
<?php
$array = array(
    "foo" => "bar",
    "bar" => "foo",
    100  => -100,
    -100 => 100,
);
var_dump($array);
?>
```

L'exemple ci-dessus va afficher :

```
array(4) {
    ["foo"]=>
    string(3) "bar"
    ["bar"]=>
    string(3) "foo"
    [100]=>
    int(-100)
    [-100]=>
    int(100)
}
```

# PHP - Rappels

---

- Les fonctions/procédures

```
<?php
function faireunyaourt ($flavour, $type = "acidophilus")
{
    return "Préparer un bol de $type $flavour.\n";
}

echo faireunyaourt ("framboise"); // fonctionne comme voulu
?>
```

## Passage par valeur

```
<?php
function foo($var)
{
    $var++;
}
$a=5;
foo ($a);
// $a vaut toujours 5
?>
```

## Passage par référence

```
<?php
function foo(&$var)
{
    $var++;
}
$a=5;
foo ($a);
// $a vaut 6 maintenant
?>
```



# PHP - Rappels

---

- La portée des variables

```
<?php
$a = 1; /* portée globale */

function test()
{
    echo $a; /* portée locale */
}

test();
?>
```

```
<?php
$a = 1;
$b = 2;
function somme() {
    global $a, $b;
    $b = $a + $b;
}
somme();
echo $b;
```

- Variables dynamiques

```
<?php
$a = 'bonjour';
$$a = 'monde';
echo $bonjour;
?>
```

# Concepts objet de PHP5

---

- Les principaux concepts:
  - **Classe**: sert à décrire un type d'objet en termes de **propriétés** (attributs) et **méthodes**
  - **Objet** : instance de classe, encapsulation et protection des données
  - Héritage simple (une classe ne peut hériter que d'une seule classe parente)
  - *Objets passés par référence  $\neq$  PHP4 (passage par copie)*
  - *Pour obtenir une copie, on passe par le clonage*

# Exemple : réifier la « nouvelle »

---

```
// déclaration de la classe Nouvelle
Class Nouvelle{
    public $titre ='Pas de titre';
    public $contenu = '';
    protected $priorité    = 0;

    public function changeTitre($nouveauTitre) {
        $this->titre = $nouveauTitre;
    }
    public function getPriorité() {
        return $this->priorité;
    }
}
// instantiation
$news = new Nouvelle();
echo $news->titre.'<br />';
echo $news->contenu.'<br />';
```

**Attributs**  
public, protected, private

**\$this**  
instance courante

**Méthodes**  
public, protected, private  
(par défaut, public)

# Protection des données (attributs et méthodes)

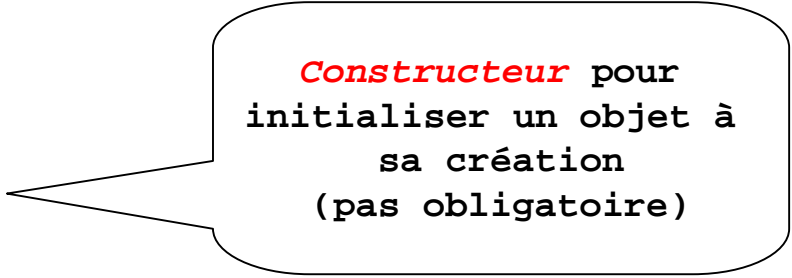
---

Accès	Public (public)	Protégé (protected)	Privé (private)
À partir de la classe elle-même	oui	oui	oui
À partir de classes dérivées	oui	oui	non
De l'extérieur	oui	non	non

## Exemple 2

---

```
Class Utilisateur {  
    public $nom;  
    public $prenom;  
    protected $age;  
  
    function __construct($n, $p, $a) {  
        $this->nom      = $n;  
        $this->prenom    = $p;  
        $this->age       = $a;  
    }  
    public function getAge() {  
        return $this->age;  
    }  
}
```



*Constructeur* pour  
initialiser un objet à  
sa création  
(pas obligatoire)

```
$u2 = new Utilisateur('Truc', 'Chose', 18);  
$u2->pseudo = 'Tortue'; // ajout dynamique d'attribut à l'instance  
echo $u2->nom.'<br />';  
echo $u2->prenom.'<br />';  
echo 'age de 1\'utilisateur: '.$u2->getAge();
```

## Exemple 3

---

```
<php?
Class mere {
    function __destruct() {
        echo 'appel au destructeur de la classe mère <br />';
    }
}

Class fille extends mere {
    function __destruct() {
        echo 'appel au destructeur de la classe fille <br />';
        parent::__destruct();
    }
}

$obj = new fille();
echo 'c\'est fini!<br />';
?>
-----
c'est fini!
appel au destructeur de la classe fille
appel au destructeur de la classe mere
```

### ***Destructeur***

Automatiquement appelée  
à la fin du script  
(ou delete() ou unset())

Rmq:

Pratique pour fermer  
des ressources ouvertes

# Copie et référence

---

- En PHP4 => copie => les objets sont identiques mais ne sont pas les mêmes

```
Class Ordi
{
    public $memoire ;
}
$monMac1 = new Ordi();
$monMac1->memoire = 4;
echo $monMac1->memoire; //affiche 4
$monMac2 = $monMac1;
$monMac2->memoire = 2;
echo $monMac2->memoire; //affiche 2
echo $monMac1->memoire; //affiche 4
```

⇒ Posait des problèmes lorsque qu'un objet passé en paramètre de méthode était modifié (modification non visible en sortie)

⇒ Solution : passer l'objet explicitement par référence : **&\$obj**

# Copie et référence

---

- **En PHP5** => référence => les objets sont les mêmes

```
Class Ordi
```

```
{  
    public $memoire ;  
}
```

```
$monMac1 = new Ordi();  
$monMac1->memoire = 4;  
echo $monMac1->memoire; //affiche 4  
$monMac2 = $monMac1;  
$monMac2->memoire = 2;  
echo $monMac2->memoire; //affiche 2  
echo $monMac1->memoire; //affiche 2
```

- Pour faire une copie => clonage

```
$monMac2 = clone $monMac1;
```

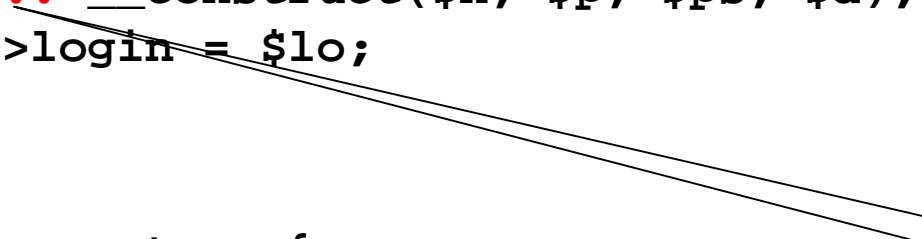


# Héritage simple

---

- Héritage simple: une classe fille ne peut hériter que d'une seule classe mère (pas d'héritage multiple)

```
Class UtilisateurAbonné extends Utilisateur{  
    private $login;  
    function __construct($n, $p, $ps, $a, $lo) {  
        parent:: __construct($n, $p, $ps, $a);  
        $this->login = $lo;  
    }  
  
    function getLogin() {  
        return $this->login;  
    }  
}
```



Accès à la classe  
parent

# Sûreté de programmation

---

- Accès public, protected, private des attributs et méthodes
- Typage des paramètres des fonctions

```
function exemple(ordi $autremac) {  
    ...  
}
```

- Classe abstraite et interface : une étape de plus dans la sûreté de programmation
  - Pour s'assurer que certains types d'objets implémentent certaines méthodes
- Classe finale : permet le concept inverse
  - Pour s'assurer qu'une méthode ou un attribut ne sera pas redéfini

# Classes abstraites et interfaces

---

- **Classe abstraite:**
  - Un début d'implémentation une classe qui ne peut être instanciée
  - On y définit certains attributs et méthodes en obligeant les classes dérivées à les implémenter
  - Peut être vue comme des "implémentations incomplètes à finir"
- **Interface:**
  - Notion proche de celle de classe abstraite mais un peu plus strict : ce n'est pas une classe mais un modèle à suivre !
  - Définit une API (Application Programming Interface)
  - Regroupe toutes les méthodes qu'une classe doit implémenter.
  - Une classe peut implémenter plusieurs interfaces
  - Aucun attribut ne peut être défini mais des constantes peuvent être définies (const)
  - Peut être vue comme un "contrôle qualité" => vérification que des objets correspondent bien aux spécifications

# Classe abstraite

---

```
abstract class ListeUtilisateurs {
    protected $users = array();

    public function ajouteUtilisateur (Utilisateur $u){
        $this->users[] = $u;
    }
    abstract public function ecrireListe();
}

class TextListeUtilisateurs extends ListesUtilisateurs {
    public function ecrire(){
        $str    = "";
        foreach ($this->users as $u){
            $str .= "$u->getPersonne." \n";
        }
        print $str;
    }
}
```

---

## Classe abstraite (suite)

---

```
class XmlListeUtilisateurs extends listeUtilisateurs {
    public function ecrire(){
        $str    "<utilisateur>\n";
        foreach ($this->users as $u){
            ... /...
        }
        $str .= "</utilisateur>\n";
        print $str;
    }
}
```

```
$u1 = new Utilisateur("Truc", "Chose", 18);
$u2 = new Utilisateur("Bidule", "Machin", 20);
$xu = new TextListeUtilisateurs();
$xu->ajouteUtilisateur($u1);
$xu->ajouteUtilisateur($u2);
$xu->ecrireListe();
```

# Interface

---

```
interface EssaiInterface1
{
    function truc();
    function chose();
}
```

```
interface EssaiInterface2
{
    function machin();
    function chouette();
}
```

```
class EssaiImplementation implements EssaiInterface1,
                                     EssaiInterface2
{
    ... /...
}
```

# Classes et méthodes finales

---

- Méthode finale
  - Pour déclarer qu'aucune classe dérivée n'a le droit de modifier l'implémentation

```
final function getPersonne() {  
    return "{$this->nom}".${$this->prenom}";  
}
```

- Classe finale
  - Pour déclarer qu'une classe ne pourra plus être dérivée
  - Toute l'implémentation est considérée comme bloquée

```
final class UtilisateurAbonné extends Utilisateur{...
```

# Erreurs et Exceptions

---

- Classe **Exception** pour la gestion des erreurs
- Similaire à la gestion des exceptions en Java ou C++ :
  - La condition est évaluée dans le bloc **"try"**. Si TRUE, une exception est levée
  - Appel au bloc **"catch"** correspondant

**try**

```
{ instruction1;  
  instruction2;  
  ...  
}
```

**catch** (Exception \$e)

```
{ ...  
  echo $e->getMessage(); echo $e->getLine();  
  echo $e->getFile();  
  exit();  
}
```

- Il est possible de créer ses propres exceptions (en créant une classe fille de Exception)
-



# Parcours d'objet

---

L'itération **foreach** affiche toutes les variables visibles disponibles :

```
class MyClass
{
    public $var1 = 'valeur 1';
    public $var2 = 'valeur 2';
    protected $protected = 'variable protégée';
    private $private = 'variable privée';

    function iterateVisible() {
        foreach($this as $key => $value) {
            print "$key => $value\n";
        }
    }
}

$c = new MyClass();
$c->iterateVisible();
foreach($c as $key => $value)
{print "$key => $value\n";}
```

# Les constantes magiques

---

Quelques constantes PHP magiques:

<code>__CLASS__</code>	Le nom de la classe courante
<code>__METHOD__</code>	Le nom de la méthode courante
<code>__LINE__</code>	La ligne courante dans le fichier
<code>__FILE__</code>	Le chemin complet et le nom du fichier courant.
<code>__FUNCTION__</code>	Le nom de la fonction

# Les méthodes magiques

---

- Les méthodes magiques => automatiser certaines tâches
- Méthodes magiques dans les classes PHP : `__construct`, `__destruct`, `__call`, `__get`, `__set`, `__isset`, `__unset`, `__sleep`, `__wakeup`, `__toString`, `__set_state`, `__clone` et `__autoload`
- On ne peut utiliser ces noms de fonction dans aucune classe sauf si on veut modifier le comportement associé à ces fonctions magiques.
- => cf. documentation !

# Nouveautés PHP7

---

- Nouveaux opérateurs : `<=>`, `??`
- Typage dans les déclarations de fonctions (paramètres et retours)  

```
function foo(int $foo, int $bar) : int  
{ return ($foo + $bar;) }
```

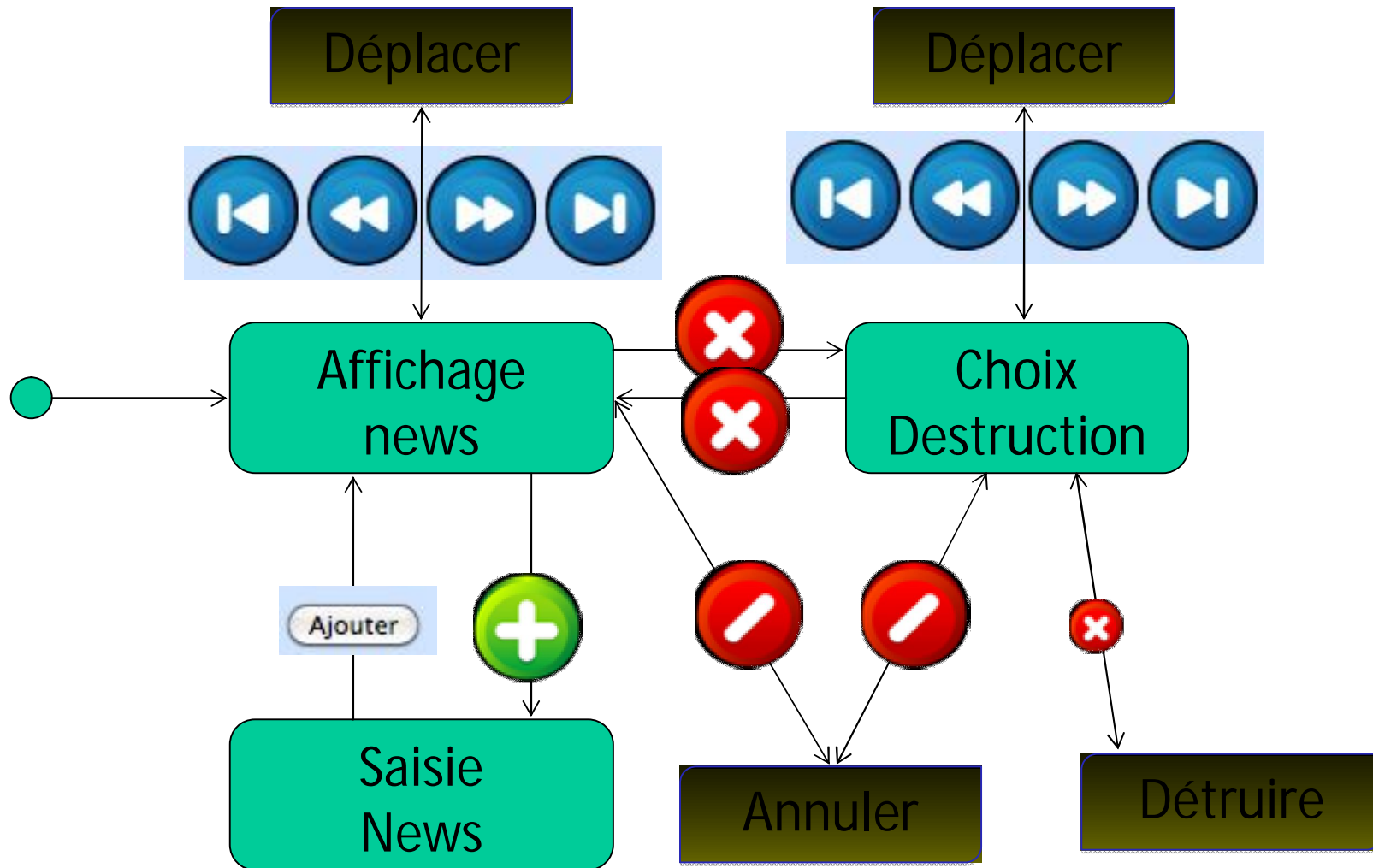
Modification de la Hiérarchie des Exceptions, Des erreurs fatals deviennent des exceptions, Classes anonymes,...

---

## Module SIL3

Vers une plus grande structuration du code

# Analyse de l'application Blog



# Codage

---

Affichage news

blog.php

Saisie  
News

add\_news\_form.php

Besoin d'un formulaire et d'une réaction au formulaire :  
add\_news.php et retour à blog.php (header)

Choix  
Destruction

del\_news\_select.php

Détruire

Réponse au choix : del\_news.php et retour à  
del\_news\_select.php

Annuler

Cancel.php et retour à blog.php

Déplacer

Toutes les possibilités sont **pré-calculées** à partir  
d'une variable d'état (\$from)

# Contenu du répertoire

---

blog.php

Point d'entrée

add\_news\_form.php

Formulaire

del\_news\_select.php

Similaire à blog

del\_news.php

add\_news.php

cancel.php

Calculs sans affichage

blog.txt

Données

icons/

blog.css

Design Interface



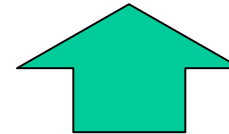
# Codage : le pré-calcul

---

- Technique
  - Nouvelle valeur d'une variable d'état est pré-calculée et associée à un lien (script.php?var=val)
- Caractéristique :
  - Le calcul est fait même si le lien n'est pas utilisé
  - Intéressant quand le calcul à faire est très simple et que le temps passé à ce calcul ne nécessite pas de déclencher un code PHP après une action.

# Pré-calcul : exemple de Google

<http://www.google.fr/#q=test&hl=fr&prmd=n&start=10&sa=N>



## Recherches associées à test

[test gratuits](#)   [test ados](#)  
[test amour](#)   [test love](#)  
[test fille](#)   [test d'amour](#)  
[test drôle](#)   [test adsl](#)



test

Rechercher

## Codage, pré-calcul : page précédente

---

```
<?php
// Revient de 5 en arrière
$fromRewind = $from-5;
// Test du début de news
if($fromRewind <=0) {
    $fromRewind=1;
}
// Construction du lien avec la nouvelle valeur
// précalculée
print "<a href=\"blog.php?from=$fromRewind\">";
// Le bouton lui même en HTML
?>
</a>
```

# Pré-calculs des déplacements

---

```
<a href="blog.php">
</a>
<?php $fromRewind = $from-5;
    if($fromRewind <=0) {$fromRewind=1;}
    print "<a href=\"blog.php?from=$fromRewind\">" ; ?>
</a>
<?php print "<a href=\"blog.php?from=$num\">" ; ?>
</a>
<?php $nb = $max -4;
if ($nb <= 0) {$nb=1;};
print "<a href=\"blog.php?from=$nb\">" ; ?>
</a>
```



## Quelques mots sur la persistance des données

---

- Persistance : garder les valeurs après la fermeture de la connexion, du serveur et du client
- Dans des fichiers sur le serveur
  - Système de fichier : simple mais peu efficace
  - Base de donnée : plus complexe mais efficace
- Choix de l'exemple : un seul fichier pour le blog
  - Nouvelles stockées chronologiquement
  - Codage par ligne
    - 2 caractères pour identifier le type de ligne
    - 'T ' : début de nouvelle, le titre
    - 'C ' : le contenu de la nouvelle
    - '#T', '#C', les lignes sont détruites, invisibles

# Persistance : exemple de fichier

---

T Des millions escroquées aux banques françaises

C L'escroquerie était fondée sur de faux achats en crédit-bail

C de matériel de chantier.

C Les gendarmes ont mis au jour une escroquerie internationale,

C révèle Le Parisien.

C Une soixantaine de personnes ont été mises en examen.

T Mise en garde américaine sur des menaces terroristes en Europe

C Les Etats-Unis ont invité dimanche leurs ressortissants

C à redoubler de vigilance face à des risques potentiels

C d'attentats en Europe

#T Attentats au Nigeria: les autorités admettent avoir été alertées

#C ABUJA —

#C Les services de renseignement du Nigeria ont reconnu

#C avoir été prévenus d'une menace du Mend.

T Google "Street View" débarque en Antarctique

C WASHINGTON --

## Codage add\_news.php : persistance

---

```
$file = fopen("blog.txt", "a");  
// recupère le titre  
$titre=$_GET["titre"];  
// L'ajoute au fichier avec le format  
fwrite($file, "T ".$titre."\n");  
// Recupère le contenu et place chaque ligne dans un tableau  
$content=preg_split("/[\n\r]+/", $_GET["content"]);  
// Le sort dans le fichier  
foreach($content as $line)  
{  
    fwrite($file, "C ".$line."\n");  
}  
fclose($file);
```

## Codage add\_news.php : état suivant

---

```
// Recherche le nombre d'éléments
$file = fopen("blog.txt","r");
$line = fgets($file);
$nb = 0;
while (!feof($file)) {
    if ($line[0] == 'T') {
        $nb++;
    }
    $line = fgets($file);
}
$from = $nb - 4; // pour afficher le nouvel élément
if ($from <= 0) { $from=1; };
// Demande de redirection pour afficher ce nouvel élément
header('Location: blog.php?from='.$from);
```



# Codage blog.php : boucle d'affichage

---

```
while (!feof($file) && $num-$from < 5) {  
    $line = substr($line,2); // Supprime le type (2 caractères)  
    $line=rtrim($line); // Supprime car. blancs à la fin de la  
    ligne  
    switch ($type) { // Regarde la ligne est un titre ou un contenu  
        case 'T' : // Titre  
            if ($num >= $from) {  
                if($tagDivOpen) { print "</div>\n";}  
                print "<div class=\"news\">\n";  
                $tagDivOpen=true;  
                print "<h2><span class=\"newsId\">";  
                print $num."</span>".$line."</h2>\n";  
                break;  
            }  
        case 'C' : // Contenu  
            if ($num >= $from) {  
                print $line."<br/>\n";  
                break;  
            }  
    }  
}
```

---

## Codage blog.php : boucle d'affichage

---

```
// Passe à la ligne suivante
$line = fgets($file);
// Recupere le type de la ligne
$type = $line[0];
// Regarde si on passe à la nouvelle suivante
if ($type == 'T') {$num++;}
```

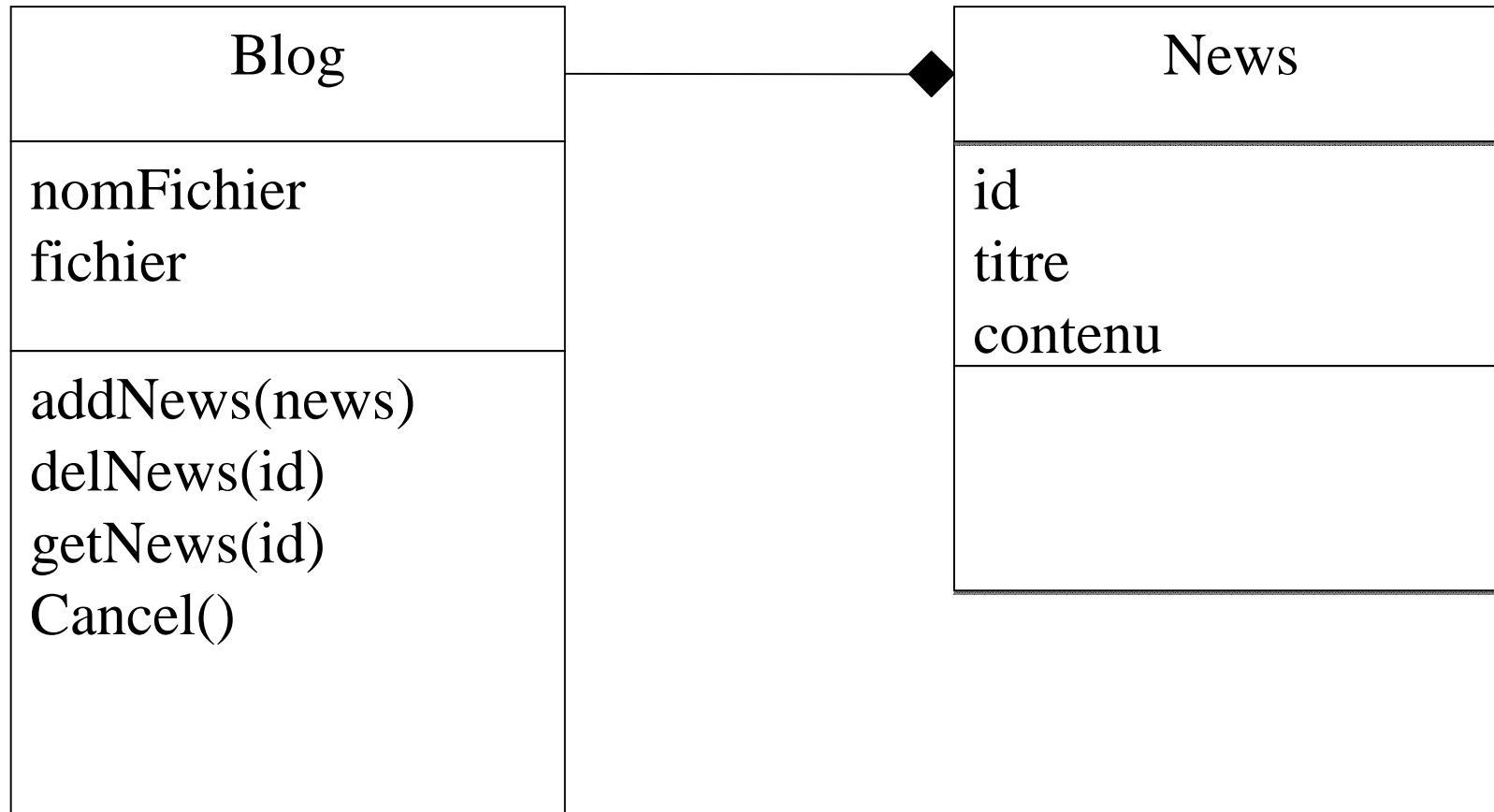
# Gestion des données éclatée

---

- Tous les traitements concernant la gestion du fichier de blog sont éclatés dans toutes les pages => gros risque de duplication de code et donc d'incohérence, maintenance difficile
  - Plusieurs fichiers à modifier si on veut changer notre gestion des données.
- ➔ Vers plus de modularité : identifier le Modèle.

## Séparation du « Modèle »

---



## Codage du modèle

---

- Tous les traitements sur les données se font dans les classes du modèle
- Les classes du modèle sont indépendantes de l'affichage
- Décharge les pages de code PHP

# Nouveau contenu du répertoire

---

blog.php  
add\_news\_form.php  
del\_news\_select.php

Code de l'interface

del\_news.php  
add\_news.php  
cancel.php

Appels au modèle

icons/  
blog.css

Design Interface

Model/  
blog.php  
news.php  
blog.txt


Modèle & Données

Nouveau

## Simplifie le code de l'interface : blog.php

---

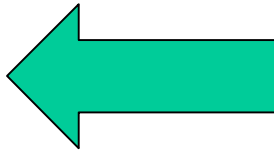
```
require_once("Modele/blog.php");
// Recupère un éventuel no de départ
if (isset($_GET["from"])) {$from = $_GET["from"];}
else {$from = 1;}
// Ouvre le blog
$blog = new Blog();
// Affichage de 5 elements de blogs
$id = $from;
$news = $blog->getNews($id);
while (($news != NULL) && ($id < $from+5)) {
    print $news->toHTML();
    $id++;
    $news = $blog->getNews($id);
}
```



## Simplifie le code de l'interface : add\_news.php

---

```
<?php
require_once("Modele/blog.php");
// recupère le titre et le contenu
$titre=$_GET["titre"]; $contenu=$_GET["content"];
// Cree un objet
$news = new News(0,$titre,$contenu);
// Ajoute la nouvelle au blog
$blog = new Blog; $blog->addNews($news);
// Recherche le nombre d'éléments
$max = $blog->max();
// Positionne sur le dernière page de 5 éléments
$from = $max -4;
if ($from <= 0) {$from=1;};
// Demande de redirection pour afficher cet élément
header('Location: blog.php?from='.$from);?>
```





## Usage d'un modèle : conclusion

---

- Une partie de la complexité de l'application passe dans le modèle
- Plus facile de faire évoluer séparément le modèle et l'interface d'interaction
  - Ex: passer d'une persistance fichier à une BD
- Toujours un peu de mélange entre le design (HTML, affichage) et l'appel au modèle (PHP)
- Structure des pages n'a pas changé du tout !
  - Pas de logique forte pour organiser les pages
    - Calcul ? pré-calcul ?
  - Pas de séparation interaction et affichage

---

Module SIL3

Mise en Pratique

# Installation des outils logiciels

---

- **LAMA** : *Linux Apache MySQL ASP* ;
- **LAMP** : *Linux Apache MySQL PHP* ;
- **MAMP** : *Macintosh Apache MySQL PHP* ;
- **AMPS** : *Apache MySQL PHP Solaris* ;
- **LAPP** : *Linux Apache PostgreSQL PHP* ;
- **FAMP** : *FreeBSD Apache MySQL PHP* ;
- **XAMP** : *Unix Apache MySQL PHP* ;
- **XAMPP** : *X Apache MySQL Perl PHP* ;
- **EWS** : *Windows Apache MySQL Perl PHP WDSCRIPT* ;

## WAMP

---

**WAMP** est un **acronyme informatique** signifiant :

- « **Windows** »
- « **Apache** »
- « **MySQL** »
- « **PHP** » dans la majorité des cas mais aussi parfois, « **Perl** », ou « **Python** ».

Il s'agit d'un **néologisme** basé sur **LAMP**.

---

## Travail à faire

---

- Installation des outils logiciels (au moins apache+php)
- Récupération des codes de blog (versions classique et modèle)
- Test de l'application (localhost)
- Analyse des codes
- Modifier les codes (versions classique et modèle) pour pouvoir contrôler le nombre de nouvelles affichées. Un nouveau bouton permet de doubler le nombre de nouvelles affichées. Un autre permet de revenir à l'affichage de 5 nouvelles.
- tp1...