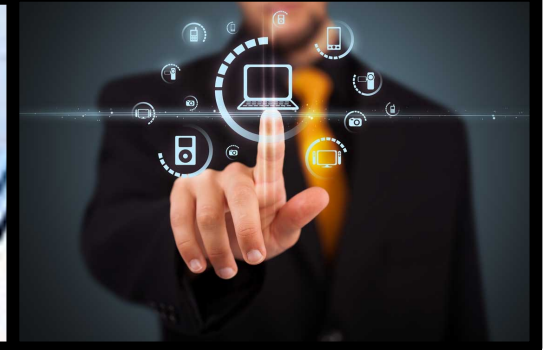
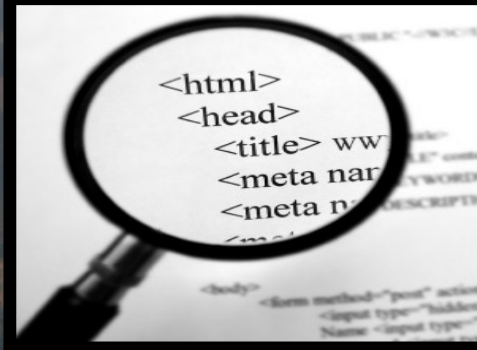


**UNIVERSITE DE CORSE**  
**Licence SPI 2ème année**  
**Option Informatique**

**UE Programmation WEB Back End**  
**CH3 – POO en php**



Evelyne VITTORI  
[vittori@univ-corse.fr](mailto:vittori@univ-corse.fr)

# Plan du cours



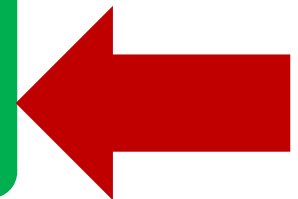
CH1 – Fondamentaux  
Programmation php



CH2 – Structuration et  
organisation du code



CH3 – POO en php



# Bilan CH2 – Structuration du code php

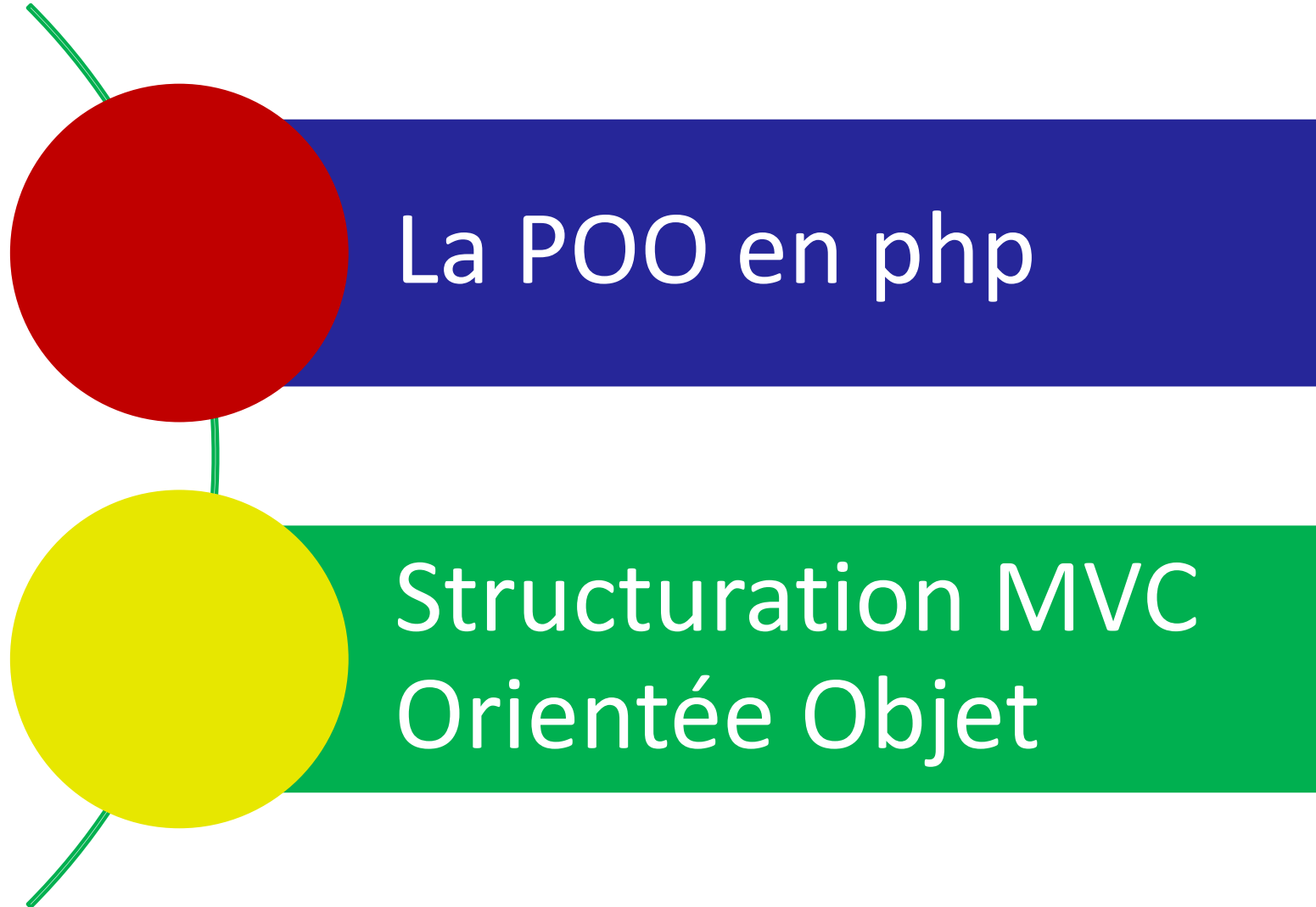
La structuration MVC procédurale du code PHP nous a permis

- De limiter la redondance de code (DRY)
- D'assurer **l'évolutivité** de l'application
  - Structuration fonctionnelle
  - Séparation de la configuration de la logique (connection à la BDD)
  - Séparation HTML/PHP
  - Standardisation du traitement des erreurs
- De **masquer la structure** du site à l'utilisateur
  - Un seul script comme point d'entrée (index.php)
  - Seul le contenu du dossier public est téléchargeable

# Évoluer vers une architecture MVC orientée objet

- Avec une solution procédurale, l'architecture MVC se reflète surtout dans l'organisation des fichiers sources
- L'adoption d'une architecture MVC orientée objet va nous permettre d'aller plus loin :
  - Renforcer la séparation des responsabilités à travers l'encapsulation
  - Factoriser le code à travers l'héritage
  - Fournir un espace de nommage clair grâce aux classes

# CH3 – POO en Php





# 1 – La POO en php





# La POO en PHP

PHP dispose d'un modèle objet uniquement depuis la version 5

- Historiquement un langage procédural
- Les concepts objets supportés :
  - **Classes** (class, new)
  - Héritage (**extends**)
  - **Visibilité** (public, protected, private) depuis PHP 7.1
  - Classes/méthodes abstraites/finales (**abstract**, **final**)
  - Interfaces (interface)

# Exemple classe PHP

```
<?php
class Point {
    const ORIGIN = 0;
    private $x;
    private $y;

    public function __construct($x=0, $y=0) {
        $this->x = $x;
        $this->y = $y;
    }

    public function getX() { return $this->x; }
    public function getY() { return $this->y; }

    protected function deplacer($deltaX, $deltaY) {
        $this->x += $deltaX;
        $this->y += $deltaY;
    }

    public function __toString(){
        return "(".$this->x.", ".$this->y.")";
    }
}
?>
```

attributs

constructeur

méthodes

\$this-> obligatoire

Mot clé function



# Particularités

- Le mot-clé **function** est utilisé pour définir les méthodes
- Le **constructeur** s'appelle toujours **\_\_construct**
- **parent** fait référence au parent d'un objet
  - Exemple appel constructeur parent : `parent::__construct();`
- On accède aux méthodes et attributs publics avec la syntaxe suivante : `objet->attribut`; `objet->methode()`;
- **\$this** est **obligatoire** pour accéder aux attributs de l'instance depuis une méthode : `$this->mon_attribut`;
- On accède aux éléments statiques avec la syntaxe suivante : `MaClasse::MA_CONSTANTE`; `MaClasse::methodeStatique()`;
- Le nom de la classe ou **self** est **obligatoire** pour accéder aux attributs statiques depuis une méthode statique : `self::var_statique`; `self::methodeStatique()`

# Visibilité

- Méthodes : public par défaut
- Attributs
  - La visibilité doit être explicite
  - Possibilité d'utiliser le mot-clé var (équivalent à public)
- Constantes :
  - restriction de la visibilité possible depuis PHP 7.1

```
<?php
class MaClasse {
    public $a = "publique";
    var $b = "publique";
    protected $c = "protégée";
    private $d = "privée";

    function methode_publique1
        { /* ... */ }
    public function methode_publique2
        { /* ... */ }
    protected function methode_protegee
        { /* ... */ }
    private function methode_privée
        { /* ... */ }
}
?>
```

# Méthodes/attributs statiques

```
<?php
```

```
class MaClasseStatique {
```

```
    public static $mon_attribut_statique = "statique";
```

```
    public function valeurStatique() {
```

```
        return self::$mon_attribut_statique;
```

```
    }
```

```
}
```

```
echo MaClasseStatique::$mon_attribut_statique;
```

```
// => "statique"
```

```
echo MaClasseStatique->valeurStatique(); // => "statique"
```

```
?>
```

Comme en java

# Méthodes/attributs statiques


```
<?php
class Point {
    const ORIGIN = 0;
    public static $origin;
    private $x;
    private $y;

    public static function initOrigin(){
        self::$origin=new
            Point(self::ORIGIN,self::ORIGIN);
    }

    public static function getOrigin(){
        return self::$origin;
    }
}
//....
?>
```

```
//TEST
$p=new Point(2,1);
Point::initOrigin();
$orig=Point::getOrigin();

echo "<p>".$p."</p>";
echo "<p>".$orig."</p>";
```



(2,1)  
(0,0)

# Classes abstraites

- Rappel : une classe abstraite peut contenir des attributs et des définitions de méthodes en plus des méthodes abstraites

```
<?php
abstract class Forme {
    abstract public function bornes();
    abstract public function contient($point);
}
?>
```

# Héritage

```
<?php
class Rectangle extends Forme {
    private $origine;
    private $largeur;
    private $hauteur;

    public function __construct($origine, $largeur, $hauteur) {
        $this->origine = $origine;
        $this->largeur = $largeur;
        $this->hauteur = $hauteur;
    }

    public function bornes() { return $this; }

    public function contient($p) {
        //teste si le point $p est contenu dans le rectangle
        $orig = $this->origine;
        $maxX = $orig->getX() + $this->$largeur;
        $maxY = $orig->getY() + $this->$hauteur;
        return $p->getX() >= $orig->getX() &&
            $p->getY() >= $orig->getY() &&
            $p->getX() <= $maxX && $p->getY() <= $maxY;
    }
}
?>
```

# Interfaces

- Une interface ne contient que des méthodes abstraites.
- Une classe qui **implémente** une interface doit implémenter toutes les méthodes abstraites définies dans l'interface.

```
<?php

interface Forme {
    public function bornes();
    public function contient($point);
}

class Rectangle implements Forme {
    /* ... */
    public function bornes() { /* ... */ }
    public function contient($point) { /* ... */ }
}

?>
```



# Classes et méthodes finales

- Une méthode déclarée *final* ne peut plus être redéfinie dans une sous classe
- Une classe déclarée *final* ne peut plus être dérivée (ne peut plus avoir de sous-classe)

```
<?php
class MaClasse {
    final public function methodeFinale()
    { /* ... */ }
}

final class Forme { /* ... */ }
?>
```

# Nouvelles fonctionnalités

## Déclarations de types

- Comme en java, php offre depuis la version 7.4, la possibilité de déclarer et contrôler les types des:
  - variables
  - paramètres de méthodes
  - valeurs de retour des méthodes

Nous ne l'utiliserons pas dans ce cours

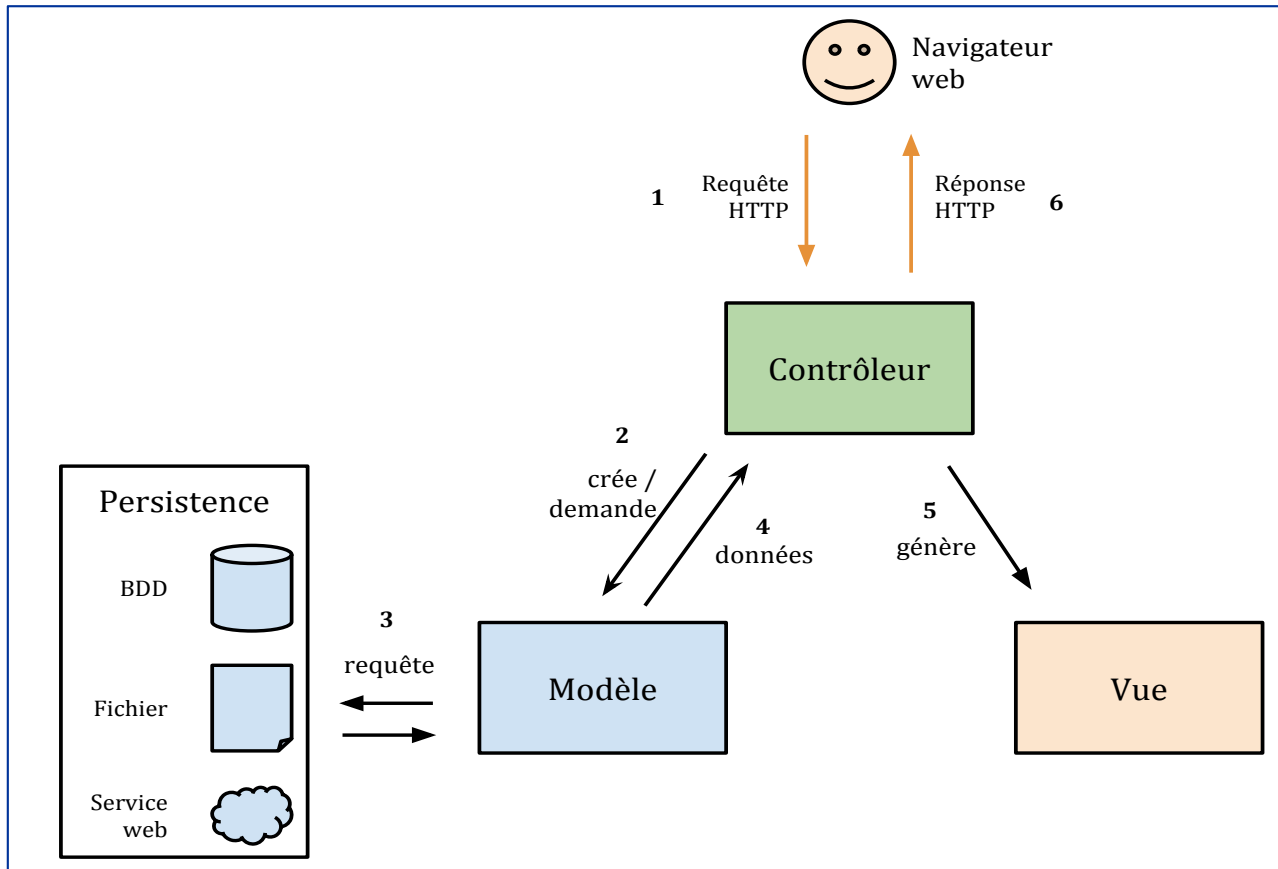
Pour en savoir plus :

<https://www.php.net/manual/fr/language.types.declarations.php>



## 2 – MVC orienté Objet

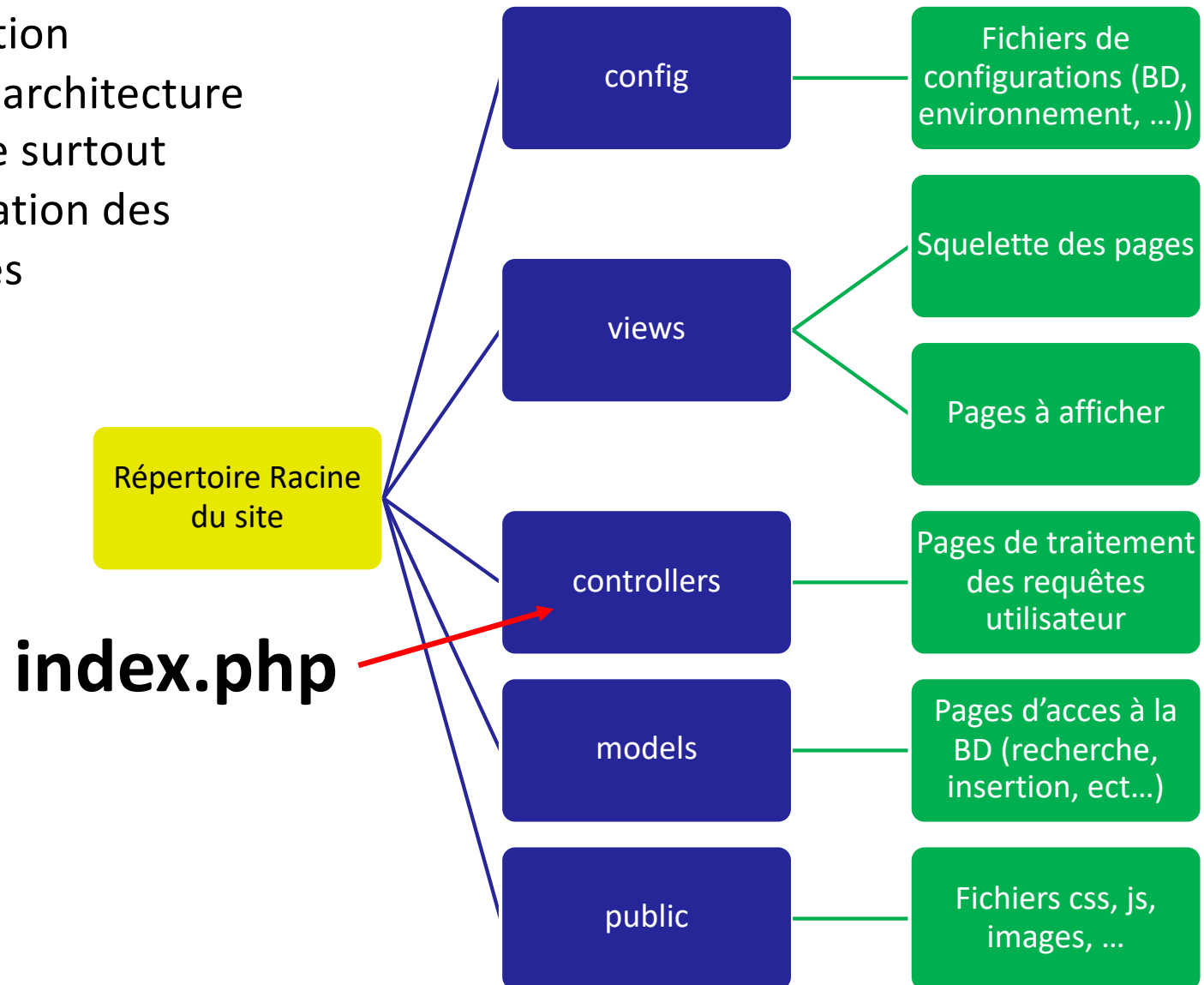
# Architecture MVC : rappel



1. Le contrôleur reçoit une requête de l'utilisateur et vérifie ses paramètres
2. Le contrôleur demande aux modèles les données nécessaires .
3. Le modèle récupère les **données** demandées.
4. Le modèle renvoie les données **encapsulées** au contrôleur.
5. Le contrôleur génère la vue d'affichage des données.
6. Le contrôleur renvoie à l'utilisateur la vue générée.

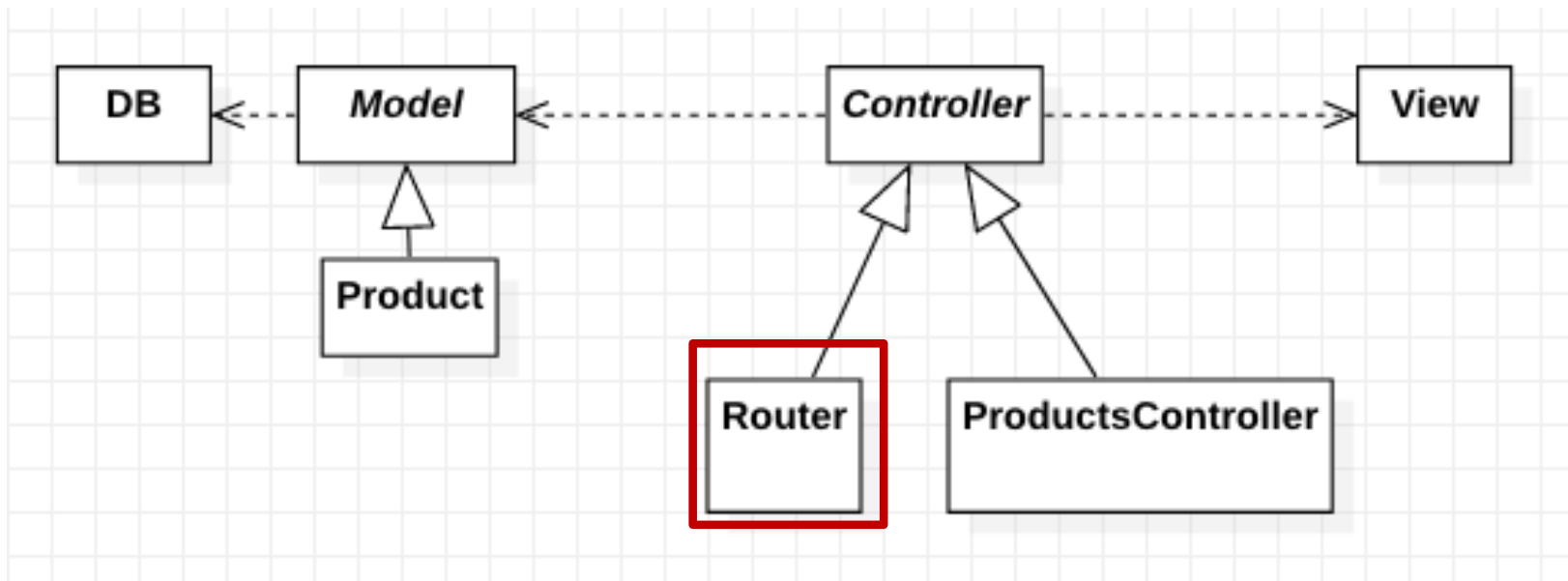
# Exemple: MVC « procédural »

Avec une solution procédurale, l'architecture MVC se reflète surtout dans l'organisation des fichiers sources



# Structuration MVC orientée objet

- Objectifs : modularité, séparation des composants, évolutivité, réutilisation
- Exemple d'illustration :
  - évolution de l'application de gestion de produits
  - Code à récupérer sur l'ENT : `ex_mvc_poo.zip`

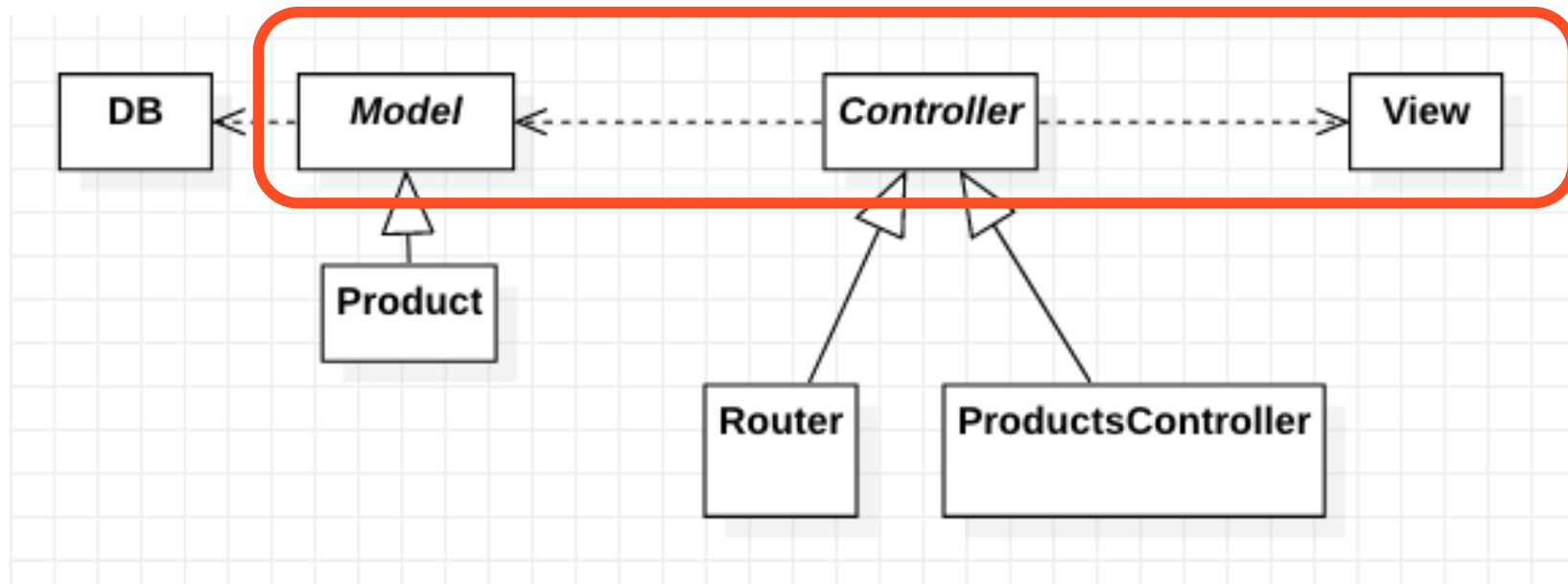


# Classes de base

Définition de classes de base Modèle - Vue - Contrôleur

- Permet l'encapsulation
- Mutualisation des services communs à tous les modèles/contrôleurs/vues
- Séparation claire des responsabilités

*Model et Controller:*  
classes abstraites





# Classe Model

```
<?php require_once 'config/db.php';
abstract class Model {
    private static $db = null;
    private static function getConnection() {
        if (self::$db == null) {
            /* ... */
        }
        return self::$db;
    }

    protected static function executeQuery($sql) {
        return self::getConnection()->query($sql);
    }

    protected static function prepareQuery($sql) {
        return self::getConnection()->prepare($sql);
    }
} ?>
```

Fichier models/model.php

# Configuration DB

```
<?php
abstract class DB {
    const HOST = "localhost";
    const PORT = "8889";
    const NAME = "ecommerce";
    const USER = "root";
    const PWD = "root";
}
```

```
DB::HOST
```

```
?>
```

Fichier config/db.php

# Classe Vue

- Architecture procédurale
  - Vue = fichier PHP correspondant à l'interface utilisateur d'une page
  - Inclut toujours un fichier squelette (layout.php)
  - Contient majoritairement des balises HTML

```
<?php $title = 'Mon E-Commerce - ' . "Produits"; ?>

<?php ob_start() ?>
<?php foreach ($products as $product): ?>
    <article>
        <a href="index.php?action=product&id=
            <?= $product['id_produit']; ?>">
            <h2>
                <?php echo $product['titre'] . ' (' .
                $product['prix']; ?> €)
            </h2>
        </a>
        <p>par <?php echo $product['fabricant'] ?></p>
    </article>
    <hr />
<?php endforeach ?>
<?php $content = ob_get_clean(); ?>
<?php require 'layout.php'; ?>
```

products.php

- Désavantage
  - La logique liée au tampon de sortie est complexe et répétée dans chaque vue
- Solution
  - Mutualisation de la logique dans **une seule classe Vue**
  - **1 instance** pour chaque fichier HTML

# Classe View

```
<?php
class View {
    const VIEW_PATH = "views/";
    const LAYOUT_VIEW = "views/layout.php";
    const DEFAULT_TITLE = "Mon site";

    private $file_to_render;
    private $title;

    public function __construct($view) {
        $this->file_to_render = self::VIEW_PATH . $view . ".php";
        $this->title = self::DEFAULT_TITLE;
    }

    private function generateView($view, $variables) {
        /* ... */
    }

    public function generate($variables) {
        /* ... */
        return $html;
    }
}
?>
```

Fichier views/view.php

# Classe View

```
<?php
class View {

    private function generateView($view, $variables) {
        if (file_exists($view)) {
            // clés/valeurs dans $variables définies comme variables locales
            extract($variables);
            ob_start();          // Mise en tampon de la sortie
            require $view;       // Génération de la vue
            return ob_get_clean(); // Renvoi du tampon
        } else {
            throw new Exception("La vue '$view' n'existe pas");
        }
    }

    public function generate($variables) {
        /* ... */
    }
}
?>
```

Génération d'une vue  
HTML paramétrée

# Classe View

```
<?php
class View {

    private function generateView($view, $variables) {
        /* ... */
    }

    public function generate($variables) {
        // Génération de la vue
        $content = $this->generateView($this->file_to_render, $variables);
        // Génération du layout (vue partagée)
        $html = $this->generateView(self::LAYOUT_VIEW,
            array('title' => $this->title,
                'content' => $content));

        return $html;
    }
}
?>
```

Réutilisation de la mise en tampon

Génération du layout

# Classe View

- Simplification des pages dédiées aux vues

```
<?php $this->title = 'Mon E-Commerce - Produits'; ?>
```

```
<?php foreach ($products as $product): ?>
```

```
<article>
```

```
<a href="index.php?action=product&id=<?= $product->id; ?>">
```

```
<h2>
```

```
<?php echo $product->titre . ' (' . $product->prix; ?> €)
```

```
</h2>
```

```
</a>
```

```
<p>par <?php echo $product->fabricant ?></p>
```

```
</article>
```

```
<hr />
```

```
<?php endforeach ?>
```

**fichier views/products.php**



# Classe Contrôleur

```
<?php
```

```
require_once("views/view.php");
```

Fichier controllers/controller.php

```
abstract class Controller {
```

```
    protected function render($view_name, $variables, $status = 200) {  
        http_response_code($status);  
        $view = new View($view_name);  
        $html = $view->generate($variables);  
        echo $html; // Envoi du HTML généré au navigateur  
    }
```

```
    protected function error($code, $msg) {  
        $this->render($code, array('msg' => $msg));  
        die();  
    }  
}
```

```
?>
```

# Accueil du site : routeur

Fichier index.php

- Rappel architecture procédurale

```
try {  
    if (isset($_GET['action'])) {  
        echo $_GET['action'];  
        if ($_GET['action'] == 'products') {  
            productsIndex();  
        } else if ($_GET['action'] == 'product') {  
            if (isset($_GET['id'])) {  
                productShow();  
            } else {  
                error(404, "/index.php?action=product");  
            } else {  
                throw new Exception("Action non valide");  
            }  
        } else {  
            home();  
        }  
    }  
} catch (Exception $e) {  
    error(500, $e->getMessage());  
}
```

**Fonctions du controleur** (pointing to `productsIndex()` and `productShow()`)

**Cas d'appel de la page à partir des vues** (pointing to `productsIndex()` and `productShow()`)

**Appel initial** (pointing to `home()`)

# Accueil du site : routeur

- Fichier index.php
  - Instantiation d'un routeur (un type de contrôleur réutilisable)
- Configuration des routes
- Routage des requêtes

```
<?php
ini_set('error_reporting', E_ALL);
ini_set('display_errors', 1);

require_once('controllers/router.php');
require_once("controllers/products.php");

$productsController = new ProductsController();
$routeur = new Router("products", array(
    "products" => array($productsController, "index"),
    "product" => array($productsController, "show"),
))
);
$routeur->routeRequest();

?>
```

# Classe Routeur

```
<?php

require_once("controllers/controller.php");

class Router extends Controller {
    private $default_action;
    private $routes;

    public function __construct($default_action, $routes) {
        $this->routes = $routes;
        $this->default_action = $default_action;
    }

    public function routeRequest() {
        /* ... */
    }
}

?>
```

Fichier controllers/router.php  
Routes configurables

# Classe Routeur

```
<?php
require_once("controllers/controller.php");
class Router extends Controller {
    /* ... */
    public function routeRequest() {
        try {
            if (!key_exists("action", $_GET)) {
                $action = $this->default_action;
            } else {
                $action = $_GET['action'];
            }

            if (key_exists($action, $this->routes)) {
                call_user_func($this->routes[$action]);
            } else {
                $this->error(404, "/index.php?action=" . $action);
            }
        } catch (Exception $e) {
            $this->error(500, $e->getMessage());
        }
    }
}
?>
```

Fichier controllers/router.php

Méthode de routage générique

# Contrôleur liés aux produits

```
<?php
require_once "controllers/controller.php";
require_once 'models/product.php';

class ProductsController extends Controller {
    public function index() {
        $products = Product::getAll();
        $this->render("products", array('products' => $products));
    }

    public function show() {
        $id = intval($_GET['id']);
        if ($id != 0) {
            $product = Product::getById($id);
            if ($product != null) {
                $this->render("product", array('product' => $product));
                return;
            }
        }
        $this->error(404, "Produit " . $id);
    }
}
?>
```

Fichier controllers/products.php

# Modèle produit (1/2)

```
<?php
require_once 'models/model.php';

class Product extends Model {
    public $id;
    public $titre;
    public $descriptif;
    public $stock;
    public $prix;
    public $fabricant;

    public function __construct($row) {
        $this->id = $row["id_produit"];
        $this->titre = $row["titre"];
        $this->descriptif = $row["descriptif"];
        $this->stock = $row["stock"];
        $this->prix = $row["prix"];
        $this->fabricant = $row["fabricant"];
    }
}
```

Fichier models/product.php



# Modèle produit 2/2

```
public static function getAll() {  
    $rows = self::executeQuery("select * from produits;");  
    $products = array();  
    foreach ($rows as $row) {  
        array_push($products, new self($row));  
    }  
    return $products;  
}
```

Fichier models/product.php

```
public static function getById($id) {  
    $query = self::prepareQuery("select * from produits where id_produit=?");  
    $query->execute(array($id));  
    if ($query->rowCount() == 1) {  
        //$row = $query->fetch();  
        $row = $query->fetch();  
        return new Product($row);  
    } else {  
        return null;  
    }  
}  
}  
?  
>
```