R3.01

CM 2 : Langage + MVC

Objet

Proche de java mais pas de surcharge compensé par des paramètres optionnels

Constructeur : __construct

Instance courante : $this

Accès à une méthode $objet->methode()

Résolution objet ::

Objet : subtilité

Les constructeurs parents ne sont pas appelés implicitement si la classe enfant définit un constructeur. Si vous voulez utiliser un constructeur parent, il sera nécessaire de faire appel à parent::__construct() depuis le constructeur enfant. Si l'enfant ne définit pas un constructeur alors il peut être hérité de la classe parent, exactement de la même façon qu'une méthode le serait (si elle n'a pas été déclarée comme privée).

Objet : exemples

```
class User{
        private $name;
        private $pass;

        public function getNom(){
            return $this->name;
        }
    }

$u = new User ("Jean");
echo "<pre>";
var_dump($u);
echo "</pre>";
var_dump($u->getNom());
```

```
object(User)#1 (2) {
 ["name":"User":private]=> NULL
["pass":"User":private]=> NULL }
NULL
```

Objet : exemples

```
class User{
        private string $name;
        private string $pass;

    public function getNom():string{
            return $this->name;
        }
}

$u = new User ("Jean");
echo "<pre>";
var_dump($u);
echo "</pre>";
var_dump($u->getNom());
```

object(User)#1 (0)
{ ["name":"User":private]=>
uninitialized(string)
["pass":"User":private]=>
uninitialized(string) }

Fatal error: Uncaught Error: Typed property User::$name must not be accessed before initialization

=> Penser à typer

Objet : exemples

```
public function __construct(string
$name, string $pass="")
{
    $this->name = $name;
    $this->pass = $pass;
}
```

```
object(User)#1 (2)
{ ["name":"User":private]=>
string(4) "Jean"
["pass":"User":private]=> string(0)
"" }

string(4) "Jean"
```

Objet : exemples

```
class Student extends User {
    private int $note;
}

$s = new Student("Jean");
var_dump($s);
```

```
object(Student)#1 (2)
{ ["name":"User":private]=>
string(4) "Jean"
["pass":"User":private]=> string(0)
"" ["note":"Student":private]=>
uninitialized(int) }  »
```

=> Soyez explicites

Objet : exemples

```
class Student extends User {
    private int $note;

    public function __construct(string $name, $pass = "")
    {
        parent::__construct($name, $pass);
        $this->note = 0;
    }
}
```

Objet : exemples

//variables statiques aussi nommées variables de classe

```
class Student extends User {
    private int $note;
    private static $nbUser = 0;
    public function __construct(string $name, $pass = "")
    {
        parent::__construct($name, $pass);
        $this->note = 0;
        //self::$nbUser++;
        Student::$nbUser++;
    }
}
```

Objet : interface

```
interface Mortel{
   pubic function meurt();
}


class Student extends User implements Mortel {
   private int $note;

   public function __construct(string $name, $pass = "")
   {
      parent::__construct($name, $pass);
      $this->note = 0;
   }

   function meurt()
   {
      echo "mort";
   }
}
```

Objet : singleton

Un Singleton (une instance unique)

```
class Singleton
{
    private static ?Singleton $instance = null;
    private function __construct() {
    }

    public static function getInstance():Singleton {

        if(is_null(self::$instance)) {
            self::$instance = new Singleton();
        }

        return self::$instance;
    }
}
```

1. Une variable statique pour conserver l'instance qui par définition sera unique car liée à la classe
2. Un constructeur privé pour interdire une instanciation à l'extérieur de la classe
3. Une méthode statique qui retourne l'instance si elle existe et qui la crée sinon.

Objet : Singleton

```
class Singleton
{
    private static ?Singleton $instance = null;
    private string $date;

    private function __construct() {
        $this->date = date(DATE_RFC2822);
    }

    public static function getInstance():Singleton {

        if(is_null(self::$instance)) {
            self::$instance = new Singleton();
        }

        return self::$instance;
    }
    public function getDate():string {
        return $this->date;
    }
}
```

```
$s = Singleton::getInstance();
echo $s->getDate().'<br>';
echo $s->getDate().'<br>';
$s = Singleton::getInstance();
echo $s->getDate().'<br>';
echo $s->getDate().'<br>';
```

Fri, 09 Sep 2022 14:10:49 +0000
Fri, 09 Sep 2022 14:10:49 +0000
Fri, 09 Sep 2022 14:10:49 +0000
Fri, 09 Sep 2022 14:10:49 +0000

Exceptions

```
class MyException extends Exception { }

//lever
throw une exception

//capturer
Try{
//danger
}
catch(MyException $e) {}
catch(Exception $e) {}
Finally {}
```

Exceptions : Exemple

```
class PasDeChanceException extends
Exception{}

function danger(int $nb){
   if ($nb == 0)
      throw new
PasDeChanceException();
   if ($nb == 1)
      throw new Exception();
}

danger(0);
danger(1);
```

Fatal error: Uncaught PasDeChanceException

Exceptions : Exemple

```
try {
    danger(0);
    danger(1);
} catch (PasDeChanceException $e) {
    echo $e;
} catch (Exception $e) {
    echo $e;
} finally {
    echo "finally";
}
```

Inclure des scripts

- require, require_once, include, include_once

-rep

-req.php (echo 0;$a = "hello";include "rep1/req1.php";echo $a;)

-rep1

-req1.php (echo 1;echo $a;$a =44;)

⇒01hello44

Rem : le premier script utilisé constitue la racine

Inclure des classes

Nous pouvons inclure une classes via son script au moment de l'instanciation en utilisant l'autoload.

-rep

-test.php

-class1

-Class1.php

-class2

-Class2.php

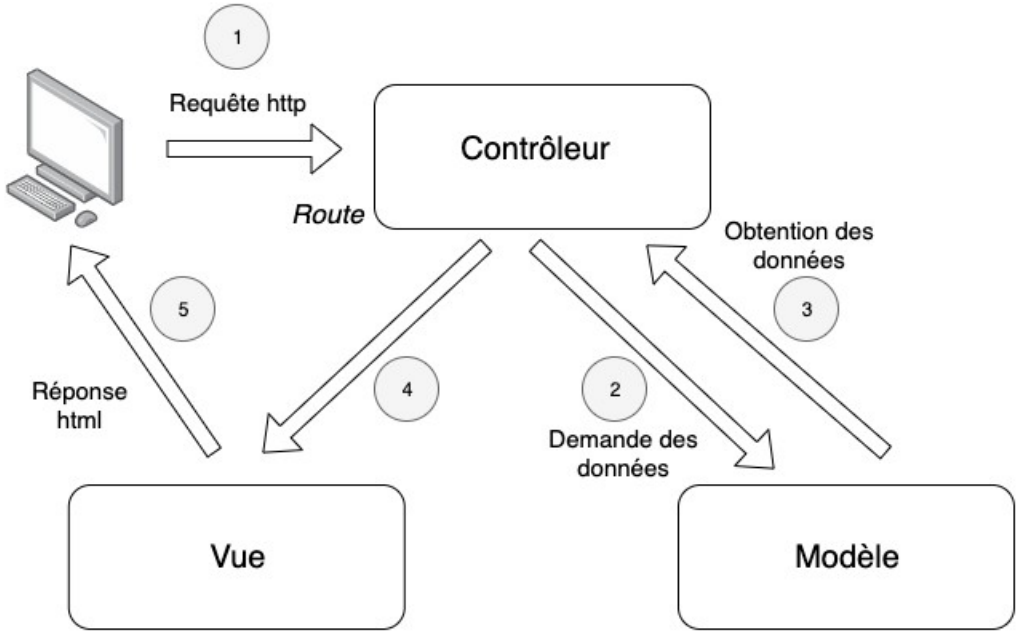
```
//test.php


function appel($classe){
   include(strtolower($classe).DIRE
CTORY_SEPARATOR.
$classe.".php");
}
spl_autoload_register("appel");
$c1 = new Class1();
$c2 = new Class2();
```

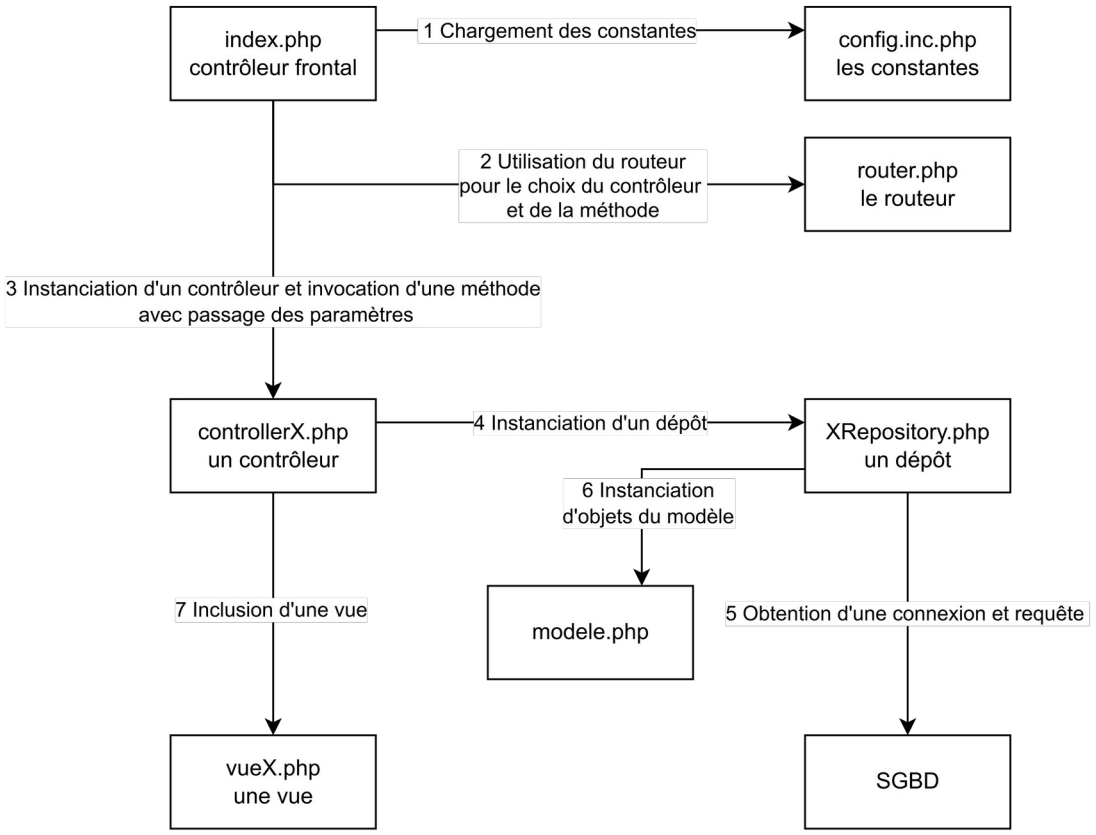
Les espaces de nommage

```php
require "User0.php";
require "User1.php";
require "User2.php";

$u0 = new User();
$u0 = new \User();
$u1 = new \good\User();
$u2 = new \bad\User();

use \good\User;
$u1 = new User();

use \bad\User as B;
$u2 = new B();
```

```php
//User0.php
class User
{

}
//User1.php
namespace good;

class User
{

}
//User2.php
namespace bad;


class User
{

}
```

MVC : Modèle Vue Controleur (Web)





Seul le contrôleur voit la vue et le modèle

1. Une URL déclenche une méthode d'un contrôleur
2. La méthode sollicite le modèle et obtient des données
3. La méthode appel une vue et lui passe des données, la vue est la réponse HTTP.

Le contrôleur frontal : index.php

On force l'utilisation d'index.php avec un .htaccess dans le répertoire qui le contient.

```
RewriteEngine On
FallbackResource /~jub/php/td2/app/index.php
```

On interdit les autres répertoires contenant des scripts

```
order deny,allow
deny from all
```

Le contrôleur frontal : index.php

Charger les constantes

```
require "config".DIRECTORY_SEPARATOR."config.inc.php";
```

Définir l'autoload

```
spl_autoload_register(function($class) {…}
```

Instancier le routeur

Traiter la route

```
$router = new Router();
$router->route();
```

Le routeur

Routes par nommage :

Controleur

Controleur/methode

Controleur/methode/param1/param2

Méthode index de Controleur

Méthode methode de Controleur

Idem avec params sous forme de tableau

```
$controllerinstance = new $controller();
$controllerinstance->$controllerMethod($params);
```

Le modèle : Entité et Dépôt

Entité

- Les classes représentant le modèle
- Attribut private + getter et setter
- Pas de constructeur ou constructeur sans paramètre

Dépôt

- Accède à la BD pour fournir des entités
- Utilisations du SQL

Le modèle : Entité et Dépôt

Entité

```
class ProductEntity
{
    private int $id;
    private string $name;
    private float $price;
    private int $quantity;

//getters setters
}
```

Dépôt

```
Interface ProductRepositoryInterface
{
    public function findAll(): array;
    public function findById(int $id):?ProductEntity;
    public function add(ProductEntity $product):ProductEntity;
    public function update(int $id, ProductEntity
$product):ProductEntity;
    public function delete(int $id): bool;

}
```

Et des implémentations

Le modèle : dépôt

```
class DbProductRepository implements ProductRepositoryInterface
{
   private $connexion; //La connexion à la BD

   public function __construct()
   {
      $dsn = "sqlite:".CFG["db"]["host"].CFG["db"]["database"]; //Le chemain de connexion
      $this->connexion = SPDO::getInstance($dsn,CFG["db"]["login"],CFG["db"]["password"],CFG["db"]["options"],CFG["db"]["exec"])
         ->getConnexion(); //obtention de la connexion avec un singleton
   }

   public function findAll(): array //renvoie un tableau de produit
   {
      $statement = $this->connexion->prepare("SELECT * FROM product");
      $statement->execute();
      return $statement->fetchAll(PDO::FETCH_CLASS, "ProductEntity");
   }
…
}
```

Contrôleur

```
//voit le modèle
//affiche la vue


class Home
{
   private ProductRepositoryInterface $repository;
   public function __construct()
   {
      $this->repository = new DbProductRepository();
   }

   function index(){
      $data = $this->repository->findAll();
      require "view".DIRECTORY_SEPARATOR."home.php";

   }


}
```

Vue

```
<!doctype html>
<html lang="fr">
<head>
   <meta charset="utf-8">
   <title>Mon magasin</title>
</head>
<body>
<nav>
   <ul>
      <li> <a href="<?=CFG['siteURL'].'Product/add'?>"> Add product </a> </li> <!–lien vers la page d’accueil -->
   </ul>
</nav>
<table>
<tbody>
   <?php foreach ($data as $product):?>
      <tr>
       <td> <?= $product->getId() ?> </td>
       <td> <?= $product->getName() ?> </td>
       <td> <?= $product->getPrice() ?> </td>
       <td> <?= $product->getQuantity() ?> </td>
       <td> <a href="<?=CFG['siteURL'].'Product/update/'.$product->getId() ?>"> update </a></td>
         <td> <a href="<?=CFG['siteURL'].'Product/delete/'.$product->getId() ?>"> delete </a></td>
      </tr>
   <?php endforeach;?>
   </tbody>
</table>
</body>
</html>
```