IUT de Montpellier - Programmation Web - TD2
La persistance des données en PHP

Rémi Coletta, Bastien Vialla

Semaine du 22 Septembre 2014

Dans le TD n° 1 vous avez appris à créer des classes et à instancier des objets de ces classes. Mais, comme vous l'avez constaté, la durée de vie des objets ainsi crées ne dépassait pas la durée de l'exécution du programme (soit quelques ms).

Dans ce TD, nous allons apprendre à rendre les objets persistants, en les sauvegardant dans une base de données. Ainsi il sera possible de retrouver les objets d'une visite à l'autre du site web.

Rappel : Les copier/coller ne fonctionnent pas correctement.
Rappel : PHP est un langage interprété, les erreurs ne sont pas détectées lors d'une compilation. Il est donc conseillé de tester régulièrement chaque nouvelle partie de code.

1 Connexion à la base de données

Connectez vous à votre base de données MySQL, à l'aide de l'interface PhpMyAdmin `http://infolimon.iutmontp.univ-montp2.fr/my` changez votre mot de passe.

Créez une table `voiture` possédant 3 champs :

— `immatriculation` de type `VARCHAR` et de longueur maximale 8, défini comme la clé primaire
— `marque` de type `VARCHAR` est de longueur maximale 25.
— `couleur` de type `VARCHAR` est de longueur maximale 12.

Attention : Les noms des champs sont comme des noms de variables, ne jamais utiliser d'accents pour le nom des champs.

Insérer des données en utilisant l'onglet `insérer` de PhpMyAdmin.

1.1 Configuration

Pour avoir un code portable, il est préférable de ne pas utiliser les informations du serveur directement dans le code.

Commencez par créez un fichier `Conf.php`. Ce fichier contiendra une classe `Conf`, et avec `$databases` en attribut statique.

```
1  <?php
2    class Conf{
3      static private $databases = array(
4               'hostname' => 'serveur_base_donnee',   // infolimon
5               'database' => 'test',                  // login IUT
6               'login'    => 'root',                  // login IUT
7               'password' => 'root'                   // votre mot de passe
8        );
9
10       static public function getLogin() {
11           //en PHP l'indice d'un tableau n'est pas forcement un chiffre.
12           return self::$databases['login'];  //idem self::$databases[1];
13        }
14 }
15 ?>
```

On peut accéder aux données de cette façon dans le fichier `testConf.php`

```
<?php
  require 'Conf.php';     //equivalent du import en Java
  // affiche le login de la base de donnees
  echo Conf::getLogin();
?>
```

Vous remarquerez qu'en PHP, on appelle une méthode statique à partir du nom de la classe comme en Java, mais en utilisant " : :" au lieu du "." en Java et au lieu du "− > " pour les méthodes classiques (c'est à dire pas static).

1.2 L'objet *PDO*

En PHP pour se connecter à une base de données on utilise PDO. Commençons par établir une connexion à la base. Créez un fichier `Model.php`.

```
<?php

 require 'Conf.php';

 class Model {

        protected $pdo;

        public function __construct() {
                $host =  Conf::getHostname();
                $dbname =  Conf::getDatabase();
                $login =  Conf::getLogin();
                $pass =  Conf::getPassword();

                $this->pdo = new PDO("mysql:host=$host;dbname=$dbname",
                                                        "$login",
                                                       "$pass");
        }
  }
?>
```

Lorsqu'une erreur se produit, PDO n'affiche pas de message d'erreur, à la place, il lève une exception. Il faut donc la récupérer et la traiter :

```
<?php
  try{
      $pdo = new PDO("mysql:host=$host;dbname=$dbname",
                 "$login",
                 "$password");
    }catch(PDOException $e){
      echo $e->getMessage(); // affiche un message d'erreur
      die();
    }
?>
```

Vous remarquerez que la syntaxe des exceptions en PHP est très semblable à celle de Java.

Dans cet exemple, la gestion est très brutale : En effet, l'instruction `die();` équivaut à un système `System.exit(-1);` en Java.

Dans un vrai site web en "production" il faudrait indiquer à l'utilisateur qu'il a fait une erreur de saisie ou que le site est actuellement indisponible, ceci en fonction du détail de l'exception qui est levée.

Il est important que tout bout de codes utilisant PDO soit dans un `try - catch` afin de capturer les exceptions.

1.3 Gestion des erreurs

Dans un site en production, pour des raisons de sécurité et de confort d'utilisation, il déconseillé d'afficher directement un message d'erreur. Pour cela on va créer une variable pour activer ou désactiver l'affichage des messages d'erreurs.

1. Dans la classe `Conf`, ajouter un attribut static `debug` et son getter public.

```php
<?php
  class Conf{
   ...

    // la variable debug est un boolean
    static private $debug = True;

    static public function getDebug() {
        return self::$debug;
    }
}
?>
```

Ainsi on peut modifier les messages d'erreurs dans les `catch`.

```php
<?php
  try{

    $pdo = new PDO("mysql:host=$host;dbname=$dbname",
                              "$login",
                               "$password");
    }catch(PDOException $e){
      if(Conf::getDebug()){
        echo $e->getMessage(); // affiche un message d'erreur
      }
      else{
           echo 'Une erreur est survenue <a href="">retour a la page d\'accueil </a>';
      }
      die();
    }
?>
```

2 Opérations sur la base de données

2.1 Requêtes préparées et insertion d'éléments dans la base

PDO fonctionne uniquement que par l'utilisation de requêtes préparées. Pour insérer une données dans la base, au lieu d'écrire la requête en utilisant les variables, on va utiliser des tags.

```php
<?php
  $req = $pdo->prepare('INSERT INTO voiture (immatriculation, marque, couleur) VALUE
                (:immatriculation, :marque, :couleur)');
?>
```

On exécute le requête en appelant la méthode `execute` de `$req` et en lui donnant un tableau contenant les données. Il est important que les clés du tableau soit les mêmes que les tags utilisés dans la construction de la requête.

Au final, on obtient le code suivant :

```php
<?php
  try{
    // Preparation de la requete
    $req = $pdo->prepare('INSERT INTO voiture (immatriculation, marque, couleur) VALUES
          (:immatriculation, :marque, :couleur)');

    // Donnees a inserer
    $data = array(
      'immatriculation'    => '256AB34',
      'marque' => 'Renaut',
      'couleur'  => 'Bleu'
      );
```

```
13
14      // execution de la requete
15      $req->execute($data);
16
17    } catch(PDOException $e){
18          echo $e->getMessage();
19    }
20  ?>
```

L'utilisation des requêtes préparées est importante d'un point de vue sécurité, on se prévient ainsi des attaques par injections SQL (qui seront abordées plus tard dans le TD sur la sécurisation des sites).

1. Ajouter une méthode `save()` à la classe `voiture`
2. Modifier la page `creerVoiture.php` du TD précédent de sorte qu'elle sauvegarde l'objet voiture crée.
3. Testez l'insertion grâce au formulaire du TD n°1.
4. Vérifiez dans MYSQL que les voitures sont bien sauvegardées.

2.2 Consulter la base de données

Testez le code suivant dans `lireVoiture.php`.

```
1  <?php
2    try{
3      $pdo = new PDO('mysql:host=serveur_base_donnee;dbname=nom_base_donnee','login', '
           ↪ password');
4      //$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
5     }catch(PDOException $e){
6        echo $e->getMessage(); // affiche un message d'erreur
7        die();
8     }
9
10    $sql = 'SELECT␣*␣FROM␣voiture'; // requete SQL
11
12    try{
13       $req = $pdo->query($sql); // execute le requete sql
14
15       // On teste si le resultat est non vide
16       if($req->rowCount() != 0){
17         while($data = $req->fetch(PDO::FETCH_OBJ)){
18           echo 'immatriculation␣:␣'.$data->immatriculation.'␣;␣marque␣:␣'.$data->marque.'␣;␣
                 ↪ couleur␣:␣'.$data->couleur;
19         }
20       }
21    }catch(PDOException $e){
22       echo $e->getMessage();
23    }
24  ?>
```

Ce code fonctionne mais ne crée pas d'objets de la classe voiture sur lesquelles l'on pourrait appeler des méthodes.

1. Vous allez donc créer une méthode statique (voir la Section 1.1 pour plus de détails sur les méthodes statiques) `getAllVoitures()` dans la classe `Voiture` qui retourne un tableau contenant toutes les voitures présentes dans la base.
2. Créer une méthode statique `getVoiture($immatriculation)` qui renvoie une voiture suivant le numéro d'immatriculation donné en paramètre.

2.3 Héritage

Toutes les classes de votre site de covoiturage doivent désormais implémenter la persistance. Vous noterez quelles partagent toutes l'accès à une base de données. Pour factoriser au maximum notre code, nous allons créer une classe `Model` qui va gérer la connexion à la base de données. Toutes les classes voulant se connecter à la base devront hériter de `Model`.

Exemple d'héritage en PHP

```php
<?php
  class Vehicule {
    private $nbRoue;

    public function getNbRoue(){
      echo $this->nbRoue;
    }
  }

  class Voiture extends Vehicule {

    __construct(){
     //appelle le constructeur de la classe mere
      parent::__construct();  //equivalent du super() en Java
      $this->nbRoue = 4;
    }
  }

  $v = new Voiture();


  // Affiche le noubre de roue d'une voiture en utilisant la methode de la classe mere.
  public function affiche() {
          echo $v->getNbRoue().' roues.';
  }
?>
```

1. Créer une classe `Model` qui se connecte à la base de données lors de sa construction et stocke la connexion dans un attribut publique statique.
2. Modifier votre code pour que la classe `Voiture` hérite de `Model`.

2.4 Site de covoiturage

Reprendre les classes du TP précédent sur le covoiturage et y ajouter l'utilisation d'une base de données.

Chaque utilisateur sera identifié par un id, de même pour chaque trajet. Utiliser une table de jonction pour lier les utilisateurs aux trajets.