PHP – Design Patterns

In the theory of software engineering, the term "Design patterns" generally refers to a reusable solution that can be used as a template for developing applications to address commonly occurring problems. You can consider the software design patterns as formalized best practices when developing software solutions.

Most of the standard design patterns can be very effectively implemented in developing applications in PHP. In this chapter, we shall learn how to apply some of the popular design patterns in developing PHP applications.

Singleton Pattern

The singleton design pattern is useful when you want to restrict the instantiation of an object of a certain class to only one instance. The name "singleton pattern" comes from the concept of singleton in Mathematics. Singleton pattern ensures that there will be only one instance, having a global access to it throughout the application.

Typical application of singleton pattern is creation of a database connection object, which must be created once in the lifetime of an application.

Example

In the following code, the DataBaseConnector class can be instantiated only once, otherwise a message that disallows duplicate object will be issued.

</> Open Compiler

```
<?php
   class DataBaseConnector {
      private static $obj;
      private final function __construct() {
         echo __CLASS__ . " object created for first time ". PHP_EOL;
      }
      public static function getConnect() {
         if (!isset(self::$obj)) {
            self::$obj = new DataBaseConnector();
            return self::$obj;
         } else {
            echo "connection object could not be created again" . PHP_EOL;
         }
```

```
      }
   }

   $obj1 = DataBaseConnector::getConnect();
   $obj2 = DataBaseConnector::getConnect();

   var_dump($obj1 == $obj2);
?>
```

It will produce the following output –

```
DataBaseConnector object created for first time
connection object could not be created again
bool(false)
```

Factory Pattern

This is one of the most commonly used design patterns. In this pattern, you don't declare the object of the desired class directly, but another class is provided whose static method creates the required object.

Example

The following example demonstrates how factory design pattern works –

</> Open Compiler

```
<?php
   class Automobile {
      private $bikeMake;
      private $bikeModel;

      public function __construct($make, $model) {
         $this->bikeMake = $make;
         $this->bikeModel = $model;
      }

      public function getMakeAndModel() {
         return $this->bikeMake . ' ' . $this->bikeModel;
      }
   }
```

```
   class AutomobileFactory {
      public static function create($make, $model) {
         return new Automobile($make, $model);
      }
   }

   $pulsar = AutomobileFactory::create('ktm', 'Pulsar');
   print_r($pulsar->getMakeAndModel());
?>
```

It will produce the following output –

```
ktm Pulsar
```


Strategy Pattern

The strategy pattern recommends an approach where you encapsulate specific families of algorithms allowing the client class responsible for instantiating a particular algorithm. The class that implements the pattern has no knowledge of the actual implementation.

Example

Here is a code that demonstrates the use of strategy pattern. We have an interface whose case() method is implemented differently by two different classes. The object of testdata class calls the respective case() methods indirectly through its own process() method.

</> Open Compiler

```
<?php
   interface example {
      public function case($str);
   }

   class ucase implements example {
      public function case($str) {
         return strtoupper($str);
```

```
      }
   }

   class lcase implements example {
      public function case($str) {
         return strtolower($str);
      }
   }

   class testdata {
      private $data;

      public function __construct($input) {
         $this->data = $input;
      }
      public function process(example $type) {
         return $this->data = $type->case($this->data);
      }
   }
   $str = "hello";
   $obj = new testdata($str);
   echo $obj->process(new ucase) . PHP_EOL;
   $str = "HELLO";
   echo $obj->process(new lcase);
?>
```

It will produce the following **output** –

```
HELLO
Hello
```

MVC Design Pattern

MVC, which stands for Model, View and Controller, is a very popular softeware architecture pattern. Most of the PHP networks such as Laravel, Symfony etc. implement the MVC architecture.

The separation of the role of each layer in an application is as follows –

- **Model** – Refers to the data structure. In this case, the database.
- **View** – Refers to the user interface. The HTML and CSS.

- **Controller** – The "middleman" doing the processing. Accepts input from the view, and works with the model. Self-explanatory, the PHP scripts and libraries themselves.

The View acts as the GUI, the Model acts as the back-end and the Control acts as an adapter. Here, three parts are interconnected with each other. It will pass the data and access the data between each other.

Example

Let us implement the MVC design pattern in pure PHP, JavaScript and HTML in the example below –

The presentation layer of the application is view.php, which renders a HTML form. The user submits the data to a controller script. The result returned by the controller is rendered on the web page with a bit of JavaScript

view.php

```
<!DOCTYPE html>
<html>
<head>
   <title>View (User Interface)</title>
   <link rel="stylesheet" href="style.css">
</head>
<body>
   <form id="mysearch" action="controller.php" method="POST">
      <input type="text" id = "nm" name="search" required>
      <input type="submit" value="Search">
   </form>
   <div id="results"></div>
   <script>
      let results = document.getElementById("results");
      results.innerHTML = "";
   </script>
   <?php
      session_start();
      if (isset($_SESSION['result'])) {
         $arr=$_SESSION['result'];

         foreach ($arr as $obj) {?>
            <script>
               results.innerHTML += "<div><?php echo $obj['id'] . "-" .
```

```
                $obj['name'] . "</div>"; ?>";
            </script>
            <?php
         }
      }
   ?>
</body>
</html>
```

The controller script requires model.php, and uses the database object, calls the select method to fetch data from the database. The result is stored in the current session so that it can be accessed on the view page.

controller.php

```
<?php
   session_start();
   require "model.php";
   $results = $_DB->select(
      "SELECT * FROM `users` WHERE `name` LIKE ?",
      ["%{$_POST["search"]}%"]
   );
   $_SESSION['search'] = $_POST['search'];
   $_SESSION['result'] = $results;
   Header("Location: view.php", true);
?>
```

The model layer of the application is coded in "model.php". It establishes connection with mysql database named mydb, using PDO extension.

model.php

```
<?php
   class DB {
      public $error = "";
      private $pdo = null;
      private $stmt = null;
      var $dsn="localhost";
      var $dbName="myDB";
      var $username="root";
      var $password="";
      function __construct () {
```

```
            $this->pdo = new PDO("mysql:host=$this->dsn;dbname=$this->
               dbName",$this->username,$this->password);
         }
         function __destruct () {
            if ($this->stmt!==null) { $this->stmt = null; }
            if ($this->pdo!==null) { $this->pdo = null; }
         }
         function select ($sql, $data=null) {
            $this->stmt = $this->pdo->prepare($sql);
            $this->stmt->execute($data);
            return $this->stmt->fetchAll();
         }
      }
      $_DB = new DB();
   ?>
```

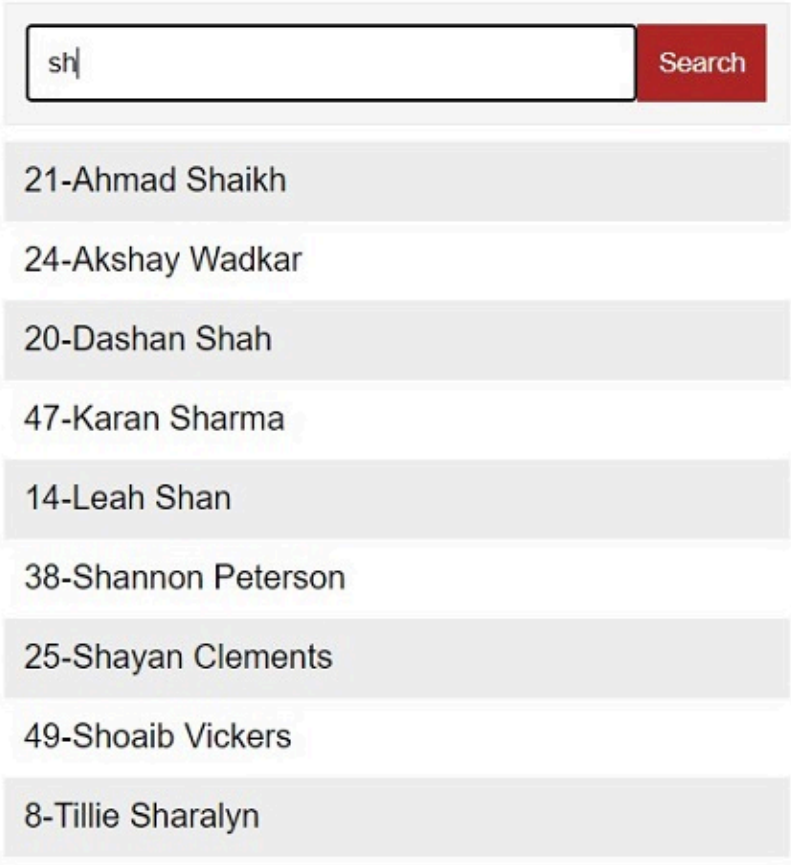
The backend mydb database must have a users table with ID and NAME fields.

```
-- Table structure for table `users`
--
CREATE TABLE `users` (
   `id` bigint(20) NOT NULL,
   `name` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;

--
-- Dumping data for table `users`
--
INSERT INTO `users` (`id`, `name`) VALUES
(21, 'Ahmad Shaikh'),
(24, 'Akshay Wadkar'),
(26, 'Bridget Wooten'),
(10, 'Coby Kelleigh'),
(20, 'Dashan Shah'),
(12, 'Elizabeth Taylor'),
(41, 'Issac Newton'),
(34, 'Julia Roberts'),
(31, 'Junior Mahmood'),
(32, 'Kailas Bende'),
(47, 'Karan Sharma'),
(16, 'Kenneth Sanders'),
(28, 'Kirstie Thomas'),
```

```
(33, 'Lawrence Murphy'),
(14, 'Leah Shan'),
(51, 'Marcus Best'),
(29, 'Maya Pande'),
(50, 'Nathaniel Khan'),
(6, 'Richard Breann'),
(54, 'Rowan Avalos'),
(3, 'Rusty Terry'),
(37, 'Sacha Gross'),
(27, 'Sally Castillo'),
(11, 'Sarah Sanders'),
(18, 'Seth Sonnel'),
(38, 'Shannon Peterson'),
(25, 'Shayan Clements'),
(49, 'Shoaib Vickers'),
(43, 'Simran Kaur'),
(35, 'Sulaiman Gilmour'),
(44, 'Taran Morin'),
(48, 'Taran Morin'),
(22, 'Thelma Kim'),
(8, 'Tillie Sharalyn'),
(36, 'Virgil Collier');
```

Start the application by visiting **"http://localhost/view.php"** in the browser. Enter a search term corresponding to the names having required letters.


sh
Search
21-Ahmad Shaikh
24-Akshay Wadkar
20-Dashan Shah
47-Karan Sharma
14-Leah Shan
38-Shannon Peterson
25-Shayan Clements
49-Shoaib Vickers
8-Tillie Sharalyn