Singleton Pattern in PHP

Enrico Besenyei

Singleton est l'un des modèles les plus simples à implémenter. Son objectif est simple : **restreindre les capacités d'instanciation d'une classe**, le plus souvent à un seul objet. Son implémentation est également simple : il ne nécessite qu'une seule classe.

Sa création répond au besoin de ne pas dupliquer certaines instances de classe : accès à la base de données, gestion des threads, gestion du cache... Singleton permet de limiter l'usage des ressources. Plusieurs instances de différentes classes peuvent ainsi se reposer sur une seule instance de la classe appliquant ce modèle.

```
1.  <?php
2.  class MaClasse {
3.    private static $instance;
4.
5.    private function __construct() {
6.      echo "Instanciation de MaClasse";
7.      }
8.
9.    public static function getInstance {
10.     if ( !isset(self::instance) ) {
11.       self::$instance = new MaClasse();
12.       }
13.     return self::$instance;
14.     }
15.
16.   public function mangerCarotte() {
17.     echo "Gnip gnip !";
18.     }
19.   }
20. ?>
```

L'idée d'un modèle est donc de faire usage des principes Objet pour obtenir ce que l'on veut. Ici, on déclare une variable interne $instance, qui contiendra la référence à l'instance de la classe. Le constructeur est déclaré private, pour que seule cette classe puisse l'invoquer... et donc instancier cette classe. Cela rend a priori la classe impossible à instancier, vu qu'elle ne peut l'être que par elle-même, mais le tour de passe-passe est réalisé par une méthode publique getInstance(), qui nous fournit un moyen d'instancier la classe, et d'en retourner une instance.

getInstance() contient une instance unique de $instance (c'est une variable statique). Si $instance n'est pas affectée, c'est que l'instance n'est pas encore créée. On instancie donc la classe depuis sa méthode publique getInstance(), et on affecte $instance, que l'on renvoi. Si $instance était déjà affectée, l'instance était donc créée lors de l'appel de getInstance, donc on annule l'instanciation demandée. Les autres méthodes de la classe sont alors accessibles. La mise en place du modèle Singleton est encore plus simple avec PHP 5 : il suffit d'utiliser des méthodes statiques :

```
1.  <?php
2.  class MaClasse {
3.    public static function mangerCarotte() {
4.      echo "Gnip gnip !";
5.      }
6.    }
7.  ?>
```

```php
// BookSingleton.php
<?php
class BookSingleton {

    private $author = 'Gamma, Helm, Johnson, and Vlissides';
    private $title  = 'Design Patterns';
    private static $book = NULL;
    private static $isLoanedOut = FALSE;

    private function __construct() { }

    static function borrowBook() {
      if (FALSE == self::$isLoanedOut) {
        if (NULL == self::$book) {
           self::$book = new BookSingleton();
        }
        self::$isLoanedOut = TRUE;
        return self::$book;
      } else {
        return NULL;
      }
    }

    function returnBook(BookSingleton $bookReturned) {
        self::$isLoanedOut = FALSE;
    }

    function getAuthor() {return $this->author;}

    function getTitle() {return $this->title;}

    function getAuthorAndTitle() {
      return $this->getTitle() . ' by ' . $this->getAuthor();
    }

}//end class
?>

// BookBorrower.php
<?php
include_once('BookSingleton.php');

class BookBorrower {

    private $borrowedBook;
    private $haveBook = FALSE;

    function __construct() { }

    function getAuthorAndTitle() {
      if (TRUE == $this->haveBook) {
        return $this->borrowedBook->getAuthorAndTitle();
      } else {
        return "I don't have the book";
      }
    }

   function borrowBook() {
      $this->borrowedBook = BookSingleton::borrowBook();
      if ($this->borrowedBook == NULL) {
        $this->haveBook = FALSE;
```

```
      } else {
        $this->haveBook = TRUE;
      }
    }

    function returnBook() {
      $this->borrowedBook->returnBook($this->borrowedBook);
    }
}//end class
?>

// BookBorrower.php
<?php
include_once('BookSingleton.php');
include_once('BookBorrower.php');

$bookBorrower1 = new BookBorrower();
$bookBorrower2 = new BookBorrower();

$bookBorrower1->borrowBook();
writeln('BookBorrower1 asked to borrow the book');
writeln('BookBorrower1 Author and Title: ');
writeln($bookBorrower1->getAuthorAndTitle());
writeln('');

$bookBorrower2->borrowBook();
writeln('BookBorrower2 asked to borrow the book');
writeln('BookBorrower2 Author and Title: ');
writeln($bookBorrower2->getAuthorAndTitle());
writeln('');

$bookBorrower1->returnBook();
writeln('BookBorrower1 returned the book');
writeln('');

$bookBorrower2->borrowBook();
writeln('BookBorrower2 Author and Title: ');
writeln($bookBorrower1->getAuthorAndTitle());
writeln('');

function writeln($line_in) {
    echo $line_in.'<'.'BR'.'>';
  }
?>
```