Guía Práctica de Ejercicios Nº 10

En nuestra décima práctica seguiremos desarrollando la programación orientada a objetos, definiendo algunos conceptos fundamentales del paradigma POO, mencionando algunas de las características del mismo y luego pasaremos a la implementación de la POO en PHP versión 5 con la implementación de patrones de diseño.

Objetivo: Comprender y profundizar los conceptos más avanzados de la programación orientada a objetos, definiendo las clases y objetos con sus atributos y métodos. Definir, desarrollar y utilizar los métodos constructores, destructores, setter, getter y toString. Entender y usar parámetros por defecto en los métodos de una clase. Utilizar objetos como parámetros en los métodos de una clase. Herencia. Clases abstractas. Polimorfismo e Interfaces con PHP. Más el desarrollo de patrones de diseño con PHP.

Introducción: En nuestra actualidad vemos que el desarrollo de Internet ha sido inminente y con ello las aplicaciones Web, por lo tanto, se hace indispensable el uso de un lenguaje que permita desarrollar aplicaciones Web como PHP, entre otros. Teniendo en cuenta la situación anterior es que veremos la como desarrollar aplicaciones Web que se ejecutarán del lado del servidor utilizando uno de los mejores lenguajes del ambiente del Software Libre. Teniendo en cuenta la magnitud del proyecto a desarrollar veremos la necesidad de emplear el paradigma orientado a objetos con PHP; que nos dará la posibilidad de diseñar, desarrollar y mantener el Software de manera profesional utilizando un paradigma antes mencionado.

Construcciones (palabras clave), variables globales y funciones que estudiaremos en esta unidad.

Construcciones del lenguaje con las que trabajaremos.

class nos permite crear una clase en PHP que es una colección de atributos y métodos.

public nos permite definir atributos y métodos para que sean accedidos desde cualquier lugar.

private nos permite definir atributos y métodos para que sean accedidos desde la clase donde fueron definidos.

protected nos permite definir atributos y métodos para que sean accedidos desde la clase donde fueron definidos y sus descendientes.

final nos permite definir atributos y métodos para que sean accedidos desde cualquier lugar pero no pueden ser reescritos por sus descendientes.

new nos permite instanciar un objeto de acuerdo a una clase.

extends nos permite definir una clase que extiende de otra.

abstract nos permite definir una clase abstracta que se utilizará de molde.

implements nos permite implementar el uso de interfaces.

const nos permite definir constantes en una clase.

Ejercicios

Utilizar los conceptos de programación Web y de base de datos

1) – Escriba los scripts PHP necesarios para desarrollar una conexión a un servidor de base de datos utilizando el patrón Singleton. Puntualmente, realice la conexión a la base de datos denominada phpn1_db_clase10 utilizando un objeto de la clase PDO.

Agregue un método getConexion() que permita retornar el objeto PDO.

Database.php

```
<?php

class DataBase
{
	private static $_singleton;
	private static $_dsn = 'mysql:dbname=phpn1_db_clase10;host=127.0.0.1;port=3306';
	private static $_user = 'root';
	private static $_pass = '';
	private $_conexion;

	private function __construct()
	{
		$this->_conexion = new PDO(self::$_dsn, self::$_user, self::$_pass);
	}

	public static function getInstancia()
	{
		if (is_null(self::$_singleton)) {
			self::$_singleton = new DataBase();
		}
		return self::$_singleton;
	}

	public function getConexion()
	{
		return $this→_conexion;
	}
}
```

index.php

```
<?php

require_once 'DataBase.php';

class Index
```

```
{
	public function ejecutar()
	{
		$oDataBase = DataBase::getInstancia();
		$pdo = $oDatabase→getConexion();
		var_dump($pdo);

		$oDataBase1 = DataBase::getInstancia();
		$pdo1 = $oDatabase1→getConexion();
		var_dump($pdo1 );
	}
}

$oIndex = new Index();
$oIndex->ejecutar();
```

2) – Teniendo en cuenta el ejemplo de la clase del patrón Factory realice los scripts necesarios para crear el desarrollo del patrón con base de datos MySQL, PostgreSQL.

ActiveRecordFactory.php

```
<?php

require_once 'MysqlActiveRecordFactory.php';
require_once 'PgsqlActiveRecordFactory.php';

class ActiveRecordFactory
{
	const MYSQL = 1;
	const PGSQL = 2;

	public static function getActiveRecordFactory($motor = selft::MYSQL)
	{
		switch ($motor) {
			case selft::MYSQL:
				return new MysqlActiveRecordFactory();
			case selft::PGSQL:
				return new PgsqlActiveRecordFactory();
		}
	}
}
```

MysqlActiveRecordFactory.php

```
<?php

class MysqlActiveRecordFactory
{

}
```

PgsqlActiveRecordFactory.php

```
<?php

class PgsqlActiveRecordFactory
{

}
```

index.php

```
<?php

class Index
{
	public function ejecutar()
	{
		$oMysql =
ActiveRecordFactory::getActiveRecordFactory(ActiveRecordFactory::MYSQL);
	}
}
```

3) – Escriba los script necesarios para desarrollar el patrón Active Record con la tabla Persona de la base de datos sgp.

Ademas deberá modificar el script del ejercicio 1 para conectar a la base de datos de este enunciado. Incluya el script y utilice los métodos de la clase para obtener la instancia PDO.

ActiveRecordPersona.php

```
<?php
require_once 'Database.php';

class ActiveRecordPersona
{
	private $_id;
	private $_tipoDocumento;
	private $_numDocumento;
	private $_apellidos;
	private $_nombres;
	private $_domicilio;
	private $_telefono;
	private $_email;

	public function getId()
	{
		return $this->_id;
	}
	public function getTipoDocumento()
	{
		return $this->_tipoDocumento;
	}
```

```
public function getNumDocumento()
{
        return $this->_numDocumento;
}
public function getApellidos()
{
        return $this->_apellidos;
}
public function getNombres()
{
        return $this->_nombres;
}
public function getDomicilio()
{
        return $this->_domicilio;
}
public function getTelefono()
{
        return $this->_telefono;
}
public function getEmail()
{
        return $this->_email;
}
public function setId($id)
{
        $this->_id = $id;
}
public function setTipoDocumento($tipoDocumento)
{
        $this->_tipoDocumento = $tipoDocumento;
}
public function setNumDocumento($numDocumento)
{
        $this->_numDocumento = $numDocumento;
}
public function setApellidos($apellidos)
{
        $this->_apellidos = $apellidos;
}
public function setNombres($nombres)
{
        $this->_nombres = $nombres;
}
public function setDomicilio($domicilio)
{
        $this->_domicilio = $domicilio;
}
public function setTelefono($telefono)
```

```
{
        $this->_telefono = $telefono;
}
public function setEmail($email)
{
        $this->_email = $email;
}

public function fetch($id)
{
        $oDb = Database::getInstancia();
        $pdo = $oDb→getConexion();

        $sql = “SELECT * FROM persona WHERE id = $id”;
        $stmt = $pdo→prepare($sql);
        $fila = $stmt→fetch(PDO::FETCH_ASSOC);

        if (!fila) {
                echo “error al obtener la fila”;
        }
        return $fila;

        unset($oDb,$pdo, $stmt);
}

public function insert()
{
        $oDb = Database::getInstancia();
        $pdo = $oDb→getConexion();

        $sql = “INSERT INTO persona (tipo_documento, num_documento, apellidos,
                nombres, domicilio, telefono, email)
                VALUES (
                        '$this→_tipoDocumento',
                        '$this->_numDocumento',
                        '$this->_apellidos',
                        '$this->_nombres',
                        '$this->_domicilio',
                        '$this->_telefono',
                        '$this→_email'
                )”;
        $stmt = $pdo→prepare($sql);

        if (!$stmt->execute()) {
                echo “error al insertar la fila”;
        }

        unset($oDb,$pdo, $stmt);
}
```

```
	public function update($id)
	{
		$oDb = Database::getInstancia();
		$pdo = $oDb→getConexion();

		$sql = “UPDATE persona SET tipo_documento = $this->_tipoDocumento,
			numero_documento = $this->_numDocumento, apellidos = $this->_apellidos,
			nombres = $this->_nombres, domicilio = $this->_domicilio, telefono = $this-
			>_telefono, email = $this->_email WHERE id = $id”;

		$stmt = $pdo→prepare($sql);

		if (!$stmt->execute()) {

			echo “error al actualizar la fila”;
		}

		unset($oDb,$pdo, $stmt);
	}

	public function delete($id)
	{
		$oDb = Database::getInstancia();
		$pdo = $oDb→getConexion();

		$sql = “DELETE FROM persona WHERE id = $id”;
		$stmt = $pdo→prepare($sql);

		if (!$stmt->execute()) {
			echo “error al borrar la fila”;
		}

		unset($oDb,$pdo, $stmt);
	}

}
```

Anotaciones de los Ejercicios


Gugler