Singleton

Padrões Criacionais

O padrão *Singleton* garante que uma classe tenha apenas uma instância e fornece um ponto global de acesso a ela.

Motivação (Por que utilizar?)

O padrão *Singleton* garante que uma classe tenha apenas uma instância ,ou seja, um único objeto. Deste modo este padrão deve ser utilizado quando tal necessidade existe.

Por exemplo, uma conexão com banco de dados. Vamos supor que em uma mesma execução uma conexão com o banco de dados será chamada várias vezes em um código. Se a classe for instanciada todas vezes que for chamada haverá perda de desempenho pois a conexão será refeita todas as vezes.

Considere que temos as classes abaixo já implementadas no sistema:

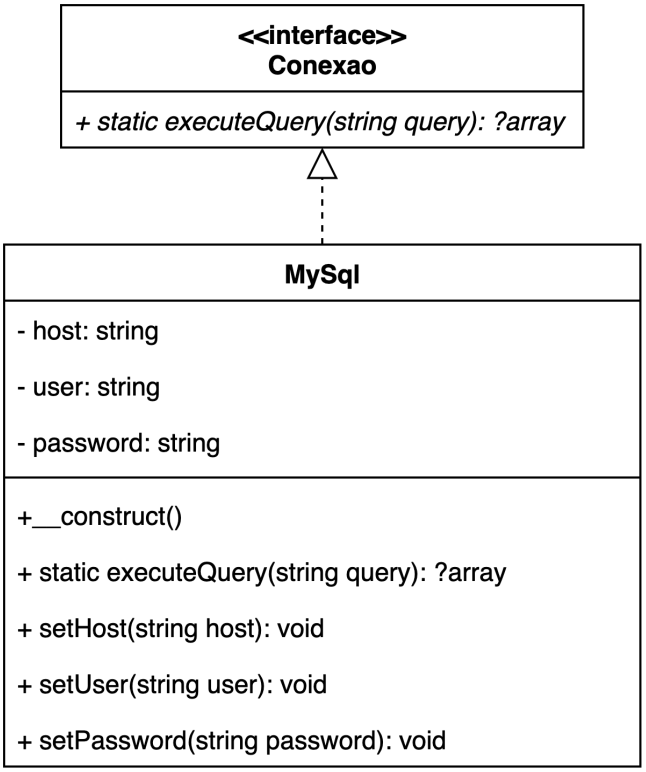

Diagrama das classes de conexão do sistema

Em cada contexto distinto de um sistema, sempre que o método `executeQuery()` é necessário, uma nova instância de `MySql` precisa ser criada previamente.

Uma instância de `MySQL` pode ser compartilhada entre todos os contextos de um código e por meio do *Singleton* podemos ter apenas um objeto criado.

É importante dizer que quando mal utilizado, o *Singleton* pode trazer muitos problemas. A seguir estão descritos alguns dos motivos que levam o *Singleton* a ser considerado por muitos um ***anti-pattern*** (antipadrão).

- *Singletons* são usados da mesma forma que variáveis globais, ele é acessível no escopo geral do sistema. Se um objeto *Singleton* for modificado de forma indevida pode causar efeitos colaterais no sistema como um todo.

- O uso de *Singletons* dificulta a realização de testes unitários, pois as classes devem ser levemente ligadas, permitindo que sejam testadas individualmente.

- *Singletons* violam o Princípio de Responsabilidade Única, pois além de sua função principal ele ainda tem a responsabilidade de gerenciar sua única instância.

- A implementação padrão do *Singleton* não é *thread safe*, ou seja, não é segura quando executada paralelamente em dois processos distintos.

- Muitas linguagens de programação fornecem operações como clonagem ou serialização de objetos, quando isso não é tratado corretamente o *Singleton* perde sua confiabilidade.

- Usar um método estático para inicializar objetos *Singleton* é considerado uma boa abordagem para implementar *Singletons*. Porém essa abordagem força os programadores a conhecerem a estrutura interna de código da classe, pois métodos estáticos apenas podem ser chamados a partir de nomes de classes.

Voltando ao nosso exemplo, vejamos algumas características do código abaixo que transforma a classe MySQL em um *Singleton* de modo que apenas uma instância dela seja criada em todo o sistema.

```php
class MySql implements Conexao
{
    private static ?MySql $instance = null; //Inicializado como null.

    private string $host;    private string $user;
    private string $password;

    //O construtor de um Singleton deve ser privado.
    //Desta forma, somente a própria classe pode instanciar seu objeto.
    private function __construct()
    {
        //Simulação de uma conexão com um banco de dados
        //No PHP poderia ser utilizado o PDO ou MySqli
        $this->host = 'mysql:host=localhost;dbname=bancoDeDados';
        $this->user = 'pedrosilva';
        $this->password = 'pedro123';
    }

    //O PHP possui dois métodos mágicos que quebram as garantias de que apenas um objeto
    //será instanciado, por isso além do construtor eles também precisam ser privados.
    private function __clone() {}

    private function __wakeup() {}

    /* Este método será utilizado fora da classe BancoDeDados, porém somente ela pode
       instanciar seu objeto (construtor privado), portanto este método deve ser estático,
       de modo que que possa ser utilizado sem uma instância de BancoDeDados. */
    public static function getInstance(): MySql
    {
        /* Se a classe BancoDeDados ainda não foi instanciada, uma instância será criada e
           atribuída a $instance. (Será true apenas na primeira vez que o método
           getInstance() for chamado, então, somente uma instância de BancodeDados será
           criada). */
        if (self::$instance === null) {
            self::$instance = new MySql();
        }
        return self::$instance;
    }

    //Responsável executar uma query no banco de dados. Simulação de execução de query.
    public static function executeQuery(string $query): ?array
    {
        echo "A query: <br>'" . $query . "'<br>foi executada com sucesso<br><br>";
        return null;
    }

    public function setHost(string $host): void
    {
        $this->host = $host;
    }

    public function setUser(string $user): void
    {
        $this->user = $user;
    }

    public function setPassword(string $password): void
    {
        $this->password = $password;
    }
```

```
    //Formata a impressão da classe
    public function __toString(): string
    {
        $saida['host'] = $this->host;
        $saida['user'] = $this->user;
        $saida['password'] = $this->password;

        return json_encode($saida);
    }
}
```

Vamos fazer um teste:

```
echo "<pre>"; //Apenas para tornar a saída mais legível.

//Criação de uma instância de MySQL
$bancoMySql1 = MySql::getInstance();

$query = 'CREATE TABLE usuario (
            id INT(11) AUTO_INCREMENT PRIMARY KEY,
            nome VARCHAR(100) NOT NULL,
            email VARCHAR(60),
            senha VARCHAR(100)
          )';

//Execução de uma query MySQL
$bancoMySql1::executeQuery($query);

//Impressão da instância $bancoMySql1.
echo "<br>==== bancoMySql1 ====<br>";
echo $bancoMySql1;

//Criação de outra instância de MySQL
$bancoMySql2 = MySql::getInstance();

//Impressão da instância $bancoMySql2.
echo "<br><br>==== bancoMySql2 ====<br>";
echo$bancoMySql2;

//Mudança de usuário de $bancoMySql1
echo "<br><br>==== MUDANÇA DE USUÁRIO EM bancoMySql1 ====<br>";
$bancoMySql1->setUser('mariaDaSilva');

//Impressão de $bancoMySql1 e $bancoMySql2
echo "<br>==== bancoMySql1 ====<br>";
echo $bancoMySql1;
echo "<br><br>==== bancoMySql2 ====<br>";
echo $bancoMySql2;

echo "</pre>"; //Apenas para tornar a saída mais legível.
```

Saída:

```
A query:
'CREATE TABLE usuario (
            id INT(11) AUTO_INCREMENT PRIMARY KEY,
            nome VARCHAR(100) NOT NULL,
            email VARCHAR(60),
            senha VARCHAR(100)
           )'
foi executada com sucesso


==== bancoMySql1 ====
{"host":"mysql:host=localhost;dbname=bancoDeDados","user":"pedrosilva","password":"pedro123"}

==== bancoMySql2 ====
{"host":"mysql:host=localhost;dbname=bancoDeDados","user":"pedrosilva","password":"pedro123"}

==== MUDANÇA DE USUÁRIO EM bancoMySql1 ====

==== bancoMySql1 ====
{"host":"mysql:host=localhost;dbname=bancoDeDados","user":"mariaDaSilva","password":"pedro123"}

==== bancoMySql2 ====
{"host":"mysql:host=localhost;dbname=bancoDeDados","user":"mariaDaSilva","password":"pedro123"}
```

Repare na saída que ao editar o valor do atributo `user` de `bancoMySql1` ele também foi editado em `bancoMySql2`. Isso ocorre devido ao fato de `bancoMySql1` e `bancoMySql2` serem exatamente a mesma instância de `MySQL`.

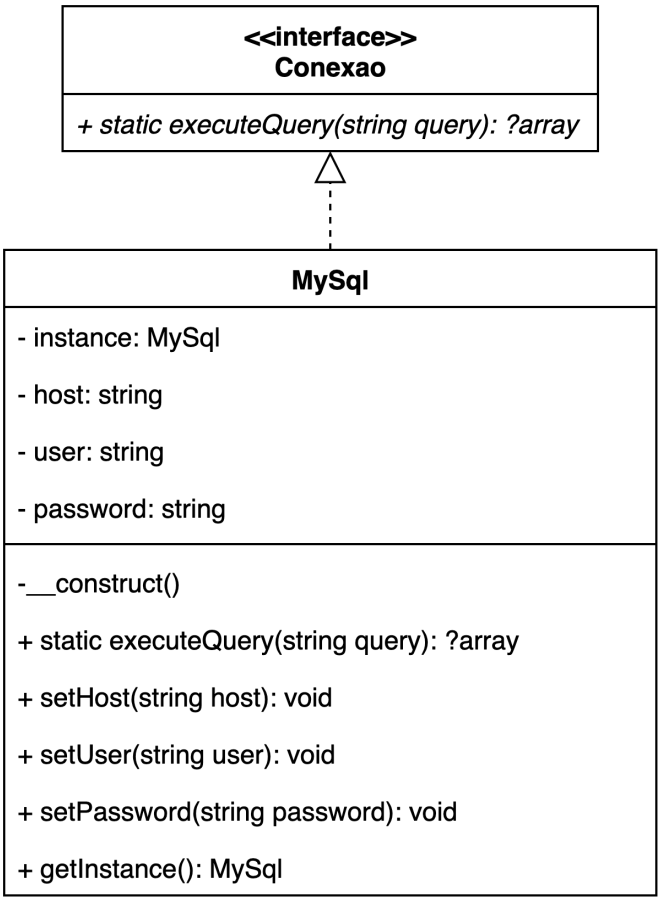

Diagrama de Classes do Exemplo Completo

Aplicabilidade (Quando utilizar?)

- Quando deve existir exatamente uma instância de uma classe e ela deve estar acessível aos clientes a partir de um ponto de acesso global conhecido.

Componentes

- **Singleton:** Esse diagrama de classes é composto por apenas uma classe.
    - O atributo uniqueInstance contém a única instância de Singleton.
    - O método getInstance() é estático, o que significa que é um método de classe, então pode ser acessado a partir de qualquer lugar do código utilizando Singleton::getInstance;

Note que temos no diagrama informações dizendo "Demais Atributos" e "Demais Métodos". Isso se dá pelo fato de que não faz sentido uma classe existir apenas para implementar o padrão *Singleton*, ela deve ser uma classe de fins gerais com seu conjunto de atributos e métodos.

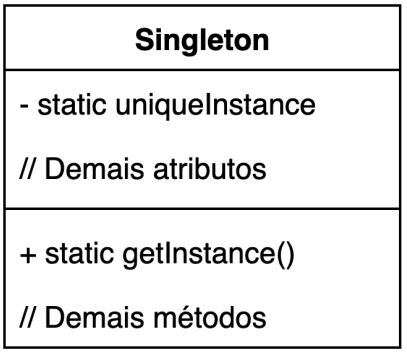

Diagrama de Classes

Consequências

- Acesso controlado a uma única instância. Como a classe *Singleton* encapsula sua única instância, ela pode ter um controle estrito sobre como e quando os clientes a acessam.
- Permite o refinamento de operações e representação. A classe *Singleton* pode ser subclassificada e é fácil configurar uma aplicação com uma instância dessa classe estendida.
- O padrão pode mudar de estratégia e passar a permitir mais de uma instância da classe *Singleton*. Controlando o número de instâncias que o sistema pode instanciar. Somente a operação que concede acesso à instância *Singleton* precisa ser alterada.