Programación web con PHP

Patrones: Command

Temario

Tipo

De comportamiento (Nivel objetos)

Propósito

Permite definir las acciones que se quieren ejecutar como un objeto.

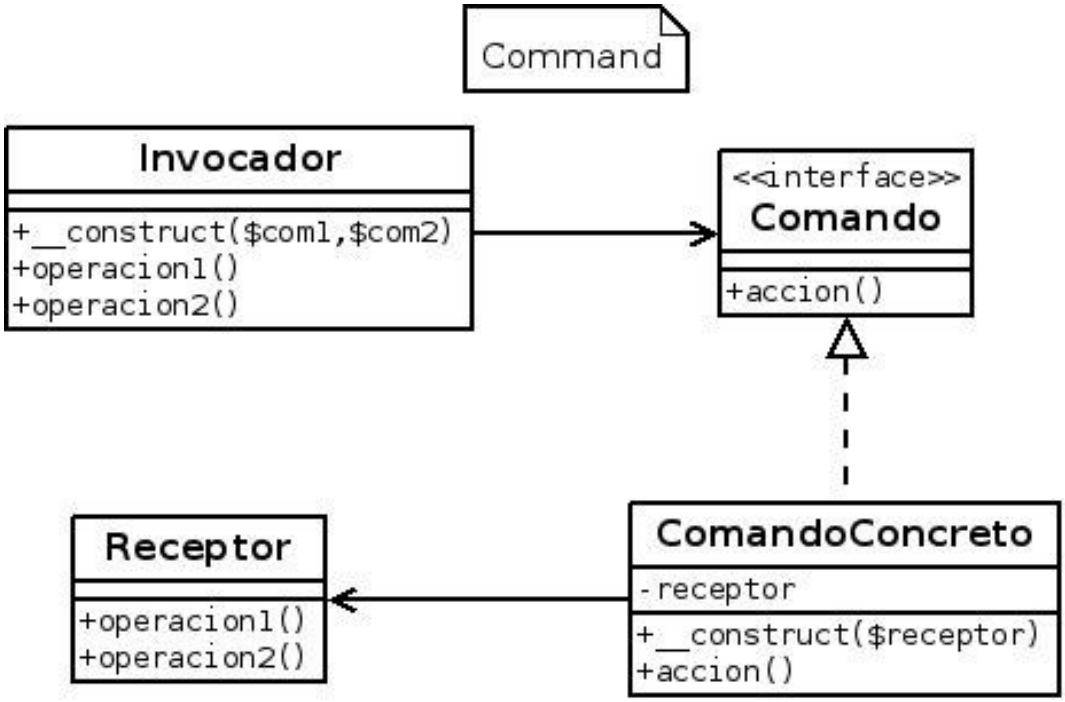
Utilización

- Cuando se desea realizar una acción sin saber que hace la operación o a quien va dirigida
- Cuando varios objetos pueden acceder a la misma acción

Ventajas

- Varios objetos pueden realizar la misma acción sin volver a implementarla
- Permite añadir nuevas acciones sin modificar clases existentes
- Permite colocar las peticiones de acción en una cola y ejecutarlas después.
- Permite deshacer una operación colocando las operaciones realizadas en una cola.

Estructura



¿Cómo funciona?

- Definimos tantos receptores como sea necesario, ellos tendrán las implementaciones de las operaciones.
- Definimos la interfaz comando que contendrá la definición del método acción.
- Definimos tantos comandos concretos que implementen la interfaz comando como operaciones existan en los receptores, , estos tendrán un atributo privado donde se guardara un receptor. Al constructor se le pasará este receptor para ser guardado. También contendrá un método acción que ejecutará el método correspondiente de receptor guardado.
- Definimos la clase invocador, tendrá tantos atributos privados como comandos concretos . Su constructor tomará por parámetro estos comandos concretos y los guardará. Cada método llamará a la función acción de alguno de los comandos guardados en el atributo.
- En el contexto instanciamos un receptor, luego creamos un invocador pasándoles por parámetro los comandos concretos correspondientes a los cuales a su vez le pasamos el receptor. Al llamar a un método del invocador este ejecutará el método acción del comando concreto correspondiente y este a su vez ejecutará la operación correspondiente del receptor
- Si quisiéramos agregar otro receptor simplemente lo definimos y le pasaríamos como parámetro al invocador los comandos concretos y a estos les pasaríamos el nuevo receptor

Código

Receptor A

```php
<?php
/*Receptor*/
class ReceptorA {

    public function operacion1() {
      echo "Receptor A operación 1<br>";
    }

    public function operacion2() {
      echo "Receptor A operación 2<br>";
    }
}
```

Receptor B

```php
<?php
//*Receptor*/
class ReceptorB {

    public function operacion1() {
      echo "Receptor B operacion 1<br>";
    }

    public function operacion2() {
     echo "Receptor B operacion 2<br>";
    }
}
```

Comando

```php
<?php
/*Comando*/
interface Comando {
   public function accion();
}
```

Comando Concreto 1

```php
<?php
/*Comando Concreto*/
class ComandoConcreto1 implements Comando {

  private $receptor;

  public function __construct($receptor) {
      $this>receptor=$receptor;
  }

  public function accion(){
    $this>receptor>operacion1();
  }
}
```

Invocador

```php
<?php
/*Invocador*/
class Invocador {

   private $comando1;
   private $comando2;

   public function __construct($c1,$c2){
          $this>comando1=$c1;
          $this>comando2=$c2;
   }
   public function operacion1() {
       $this>comando1>accion();
   }

   public function operacion2() {
       $this>comando2>accion();
   }
}
```

index.php

```php
<?php
//Creo un recetptor
        $receptorA = new ReceptorA();
   //Creo un invocador y le paso dos comandos concretos pasando a su
vez el receptor
        $invocador = new Invocador(new ComandoConcreto1($receptorA),new
ComandoConcreto2($receptorA));
        //LIamo a las operaciones
   $invocador>operacion1();//Receptor A operacion 1
   $invocador>operacion2();//Receptor A operacion 2

   //Creo otro receptor
   $receptorB = new ReceptorB();
   //Creo el invocador solo que esta vez paso el nuevo receptor
        $invocador = new Invocador(new ComandoConcreto1($receptorB),new
ComandoConcreto2($receptorB));
        //Lamo a las operaciones
   $invocador>operacion1();//Receptor B operacion 1
   $invocador>operacion2();//Receptor B operacion 2
```