Обучение Советы

Шаблоны проектирования по-человечески: структурные паттерны

От **Montgomeri** - 09.06.2017

22441 2

Добавить в избранное

Структурные паттерны связаны со структурами из объектов, с тем, как эти объекты взаимодействуют друг с другом.

Характеристика

Структурные паттерны рассматривают образование более крупных структур из объектов и классов. Шаблоны уровня класса полезны в тех случаях, когда необходимо объединить процессы нескольких библиотек. При этом затрагиваются такие механизмы:

1. Наследование. Определение реализации подклассами, а интерфейса – базовым классом.
2. Композиция, при которой структуры образуются через объединение нескольких объектов.

Структурные паттерны: классификация

Данные шаблоны делятся на:

1. Adapter
2. Bridge
3. Composite
4. Decorator
5. Facade
6. Flyweight
7. Proxy

1. Паттерн Адаптер

Представьте, что у вас есть несколько фото, которые хранятся на карте памяти. Вам нужно перенести их на компьютер, а чтобы это сделать, необходим адаптер, совместимый и с картой, и с портами ПК. Именно такую функцию выполняет паттерн Adapter: обеспечивает взаимосвязь классов и несовместимых интерфейсов.

В качестве другого примера рассмотрим игру, в которой есть охотник и львы. Для начала нам потребуется интерфейс Lion:

```
interface Lion
{
    public function roar();
}

class AfricanLion implements Lion
{
    public function roar()
    {
    }
}

class AsianLion implements Lion
{
    public function roar()
    {
    }
}
```

И есть охотник, который ожидает реализации интерфейса Lion для охоты:

```
class Hunter
{
    public function hunt(Lion $lion)
    {
    }
}
```

Теперь добавим в игру класс WildDog, на который охотник тоже должен охотиться. Вот только это нельзя сделать напрямую, так как у динго другой интерфейс. Чтобы сделать класс совместимым с классом охотника, нужен адаптер:

```
// Этот класс нужно добавить в игру
class WildDog
{
    public function bark()
    {
    }
}

// Создадим адаптер
class WildDogAdapter implements Lion
```

```php
{
    protected $dog;

    public function __construct(WildDog $dog)
    {
        $this->dog = $dog;
    }

    public function roar()
    {
        $this->dog->bark();
    }
}
```

Используем:

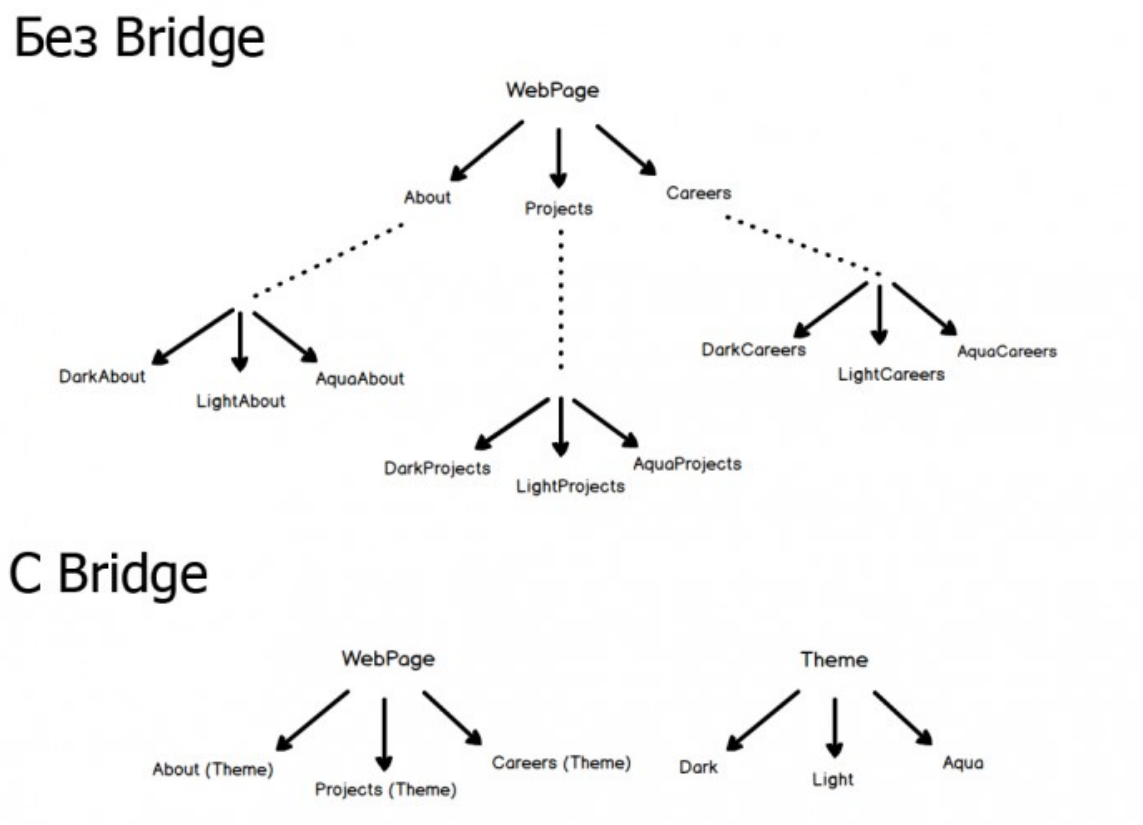
```php
$wildDog = new WildDog();
$wildDogAdapter = new WildDogAdapter($wildDog);

$hunter = new Hunter();
$hunter->hunt($wildDogAdapter);
```

Теперь рассмотрим другие структурные паттерны.

2. Паттерн Мост

Есть сайт с разными страницами, и пользователь должен иметь возможность изменять тему. Что делать? Можно создать копии страниц для каждой из тем или же просто загрузить темы отдельно. Структурные паттерны Bridge позволят реализовать второй вариант.



Шаблон отвечает за разделение одного или нескольких классов на 2 иерархии, которые включают в себя абстракцию и реализацию. Это позволяет работать с иерархиями независимо друг от друга.

Наш пример в виде кода, в котором представлена иерархия WebPage:

```
interface WebPage
{
    public function __construct(Theme $theme);
    public function getContent();
}

class About implements WebPage
{
    protected $theme;

    public function __construct(Theme $theme)
    {
        $this->theme = $theme;
    }

    public function getContent()
    {
        return "Страница About page в теме " . $this->theme->getColor();
    }
}

class Careers implements WebPage
{
    protected $theme;

    public function __construct(Theme $theme)
    {
        $this->theme = $theme;
    }

    public function getContent()
    {
        return "Страница Careers page в теме " . $this->theme->getColor();
    }
}
```

И отдельная иерархия тем:

```
interface Theme
{
    public function getColor();
}

class DarkTheme implements Theme
{
    public function getColor()
    {
        return 'Dark Black';
    }
}
class LightTheme implements Theme
{
    public function getColor()
    {
        return 'Off white';
    }
}
class AquaTheme implements Theme
{
    public function getColor()
    {
        return 'Light blue';
    }
}
```

Реализовываем их:

```
$darkTheme = new DarkTheme();

```

```php
$about = new About($darkTheme);
$careers = new Careers($darkTheme);

echo $about->getContent(); // "Страница "About page" в теме "Dark Black"";
echo $careers->getContent(); // "Страница "Careers page" в теме "Dark Black"";
```

3. Паттерн Компоновщик

Любая организация состоит из сотрудников, а сотрудники, в свою очередь, выполняют определенные обязанности, получают заработную плату, могут или не могут иметь в подчинении других сотрудников и т. д. Структурные паттерны Composite объединяют различные объекты в древовидные структуры, позволяя в дальнейшем работать с ними, как с одним объектом.

Здесь представлены разные типы сотрудников:

```php
interface Employee
{
    public function __construct(string $name, float $salary);
    public function getName(): string;
    public function setSalary(float $salary);
    public function getSalary(): float;
    public function getRoles(): array;
}

class Developer implements Employee
{
    protected $salary;
    protected $name;

    public function __construct(string $name, float $salary)
    {
        $this->name = $name;
        $this->salary = $salary;
    }

    public function getName(): string
    {
        return $this->name;
    }

    public function setSalary(float $salary)
    {
        $this->salary = $salary;
    }

    public function getSalary(): float
    {
        return $this->salary;
    }

    public function getRoles(): array
    {
        return $this->roles;
    }
}

class Designer implements Employee
{
    protected $salary;
    protected $name;

    public function __construct(string $name, float $salary)
    {
        $this->name = $name;
        $this->salary = $salary;
    }

    public function getName(): string
```

```
    {
        return $this->name;
    }

    public function setSalary(float $salary)
    {
        $this->salary = $salary;
    }

    public function getSalary(): float
    {
        return $this->salary;
    }

    public function getRoles(): array
    {
        return $this->roles;
    }
}
```

Далее представляем организацию, в которой состоят различные сотрудники:

```
class Organization
{
    protected $employees;

    public function addEmployee(Employee $employee)
    {
        $this->employees[] = $employee;
    }

    public function getNetSalaries(): float
    {
        $netSalary = 0;

        foreach ($this->employees as $employee) {
            $netSalary += $employee->getSalary();
        }

        return $netSalary;
    }
}
```

Используем:

```
// Подготовка сотрудников
$john = new Developer('Джон', 12000);
$jane = new Designer('Джейн', 15000);

// Добавляем их в организацию
$organization = new Organization();
$organization->addEmployee($john);
$organization->addEmployee($jane);

echo "Оклад: " . $organization->getNetSalaries(); // Оклад: 22000
```

4. Паттерн Декоратор

В отличие от статического механизма наследования, паттерн Decorator работает динамически. Он может добавлять объектам необходимую функциональность в процессе.

Рассмотрим в качестве примера кофе. Прежде всего, у нас есть простой кофе с соответствующим интерфейсом:

```
interface Coffee
{
```

```
    public function getCost();
    public function getDescription();
}

class SimpleCoffee implements Coffee
{
    public function getCost()
    {
        return 10;
    }

    public function getDescription()
    {
        return 'Простой кофе';
    }
}
```

Но мы хотим добавить дополнительные параметры:

```
class MilkCoffee implements Coffee
{
    protected $coffee;

    public function __construct(Coffee $coffee)
    {
        $this->coffee = $coffee;
    }

    public function getCost()
    {
        return $this->coffee->getCost() + 2;
    }

    public function getDescription()
    {
        return $this->coffee->getDescription() . ', молоко';
    }
}

class WhipCoffee implements Coffee
{
    protected $coffee;

    public function __construct(Coffee $coffee)
    {
        $this->coffee = $coffee;
    }

    public function getCost()
    {
        return $this->coffee->getCost() + 5;
    }

    public function getDescription()
    {
        return $this->coffee->getDescription() . ', сливки';
    }
}

class VanillaCoffee implements Coffee
{
    protected $coffee;

    public function __construct(Coffee $coffee)
    {
        $this->coffee = $coffee;
    }

    public function getCost()
    {
        return $this->coffee->getCost() + 3;
    }

    public function getDescription()
```

```
    {
        return $this->coffee->getDescription() . ', ваниль';
    }
}
```

Теперь сделаем наш кофе:

```
$someCoffee = new SimpleCoffee();
echo $someCoffee->getCost(); // 10
echo $someCoffee->getDescription(); // Простой кофе

$someCoffee = new MilkCoffee($someCoffee);
echo $someCoffee->getCost(); // 12
echo $someCoffee->getDescription(); // Простой кофе, молоко

$someCoffee = new WhipCoffee($someCoffee);
echo $someCoffee->getCost(); // 17
echo $someCoffee->getDescription(); // Простой кофе, молоко, сливки

$someCoffee = new VanillaCoffee($someCoffee);
echo $someCoffee->getCost(); // 20
echo $someCoffee->getDescription(); // Простой кофе, молоко, сливки, ваниль
```

5. Паттерн Фасад

Как включить компьютер? Все мы привыкли, что для этого достаточно нажать одну кнопку, и никто не задумывается, что при этом происходит на самом деле. Как и простое включение компьютера, Facade паттерн позволяет использовать максимально простой интерфейс для доступа к библиотеке, системе классов или фреймворку.

Итак, у нас есть класс «Computer»:

```
class Computer
{
    public function getElectricShock()
    {
        echo "Ouch!";
    }

    public function makeSound()
    {
        echo "Beep beep!";
    }

    public function showLoadingScreen()
    {
        echo "Loading..";
    }

    public function bam()
    {
        echo "Ready to be used!";
    }

    public function closeEverything()
    {
        echo "Bup bup bup buzzzz!";
    }

    public function sooth()
    {
        echo "Zzzzz";
    }

    public function pullCurrent()
    {
```

```
        echo "Haaah!";
    }
}
```

А здесь расположен фасад:

```
class ComputerFacade
{
    protected $computer;

    public function __construct(Computer $computer)
    {
        $this->computer = $computer;
    }

    public function turnOn()
    {
        $this->computer->getElectricShock();
        $this->computer->makeSound();
        $this->computer->showLoadingScreen();
        $this->computer->bam();
    }

    public function turnOff()
    {
        $this->computer->closeEverything();
        $this->computer->pullCurrent();
        $this->computer->sooth();
    }
}
```

Теперь мы можем использовать фасад:

```
$computer = new ComputerFacade(new Computer());
$computer->turnOn(); // Ouch! Beep beep! Loading.. Ready to be used!
$computer->turnOff(); // Bup bup buzzz! Haah! Zzzzz
```

6. Паттерн Flyweight

Вы когда-нибудь заказывали чай в кафе? Они зачастую делают больше одной чашки, приносят вам ваш чай, а остальное сохраняют для других клиентов. Структурные паттерны Flyweight как раз этим и занимаются, чем экономят память.

У нас есть чай и тот, кто его готовит:

```
class KarakTea
{
}

// Работает как производитель и сохраняет чай
class TeaMaker
{
    protected $availableTea = [];

    public function make($preference)
    {
        if (empty($this->availableTea[$preference])) {
            $this->availableTea[$preference] = new KarakTea();
        }

        return $this->availableTea[$preference];
    }
}
```

Также есть кафе, где принимаются и выполняются заказы:

```php
class TeaShop
{
    protected $orders;
    protected $teaMaker;

    public function __construct(TeaMaker $teaMaker)
    {
        $this->teaMaker = $teaMaker;
    }

    public function takeOrder(string $teaType, int $table)
    {
        $this->orders[$table] = $this->teaMaker->make($teaType);
    }

    public function serve()
    {
        foreach ($this->orders as $table => $tea) {
            echo "Принести чай, столик# " . $table;
        }
    }
}
```

Используем:

```php
$teaMaker = new TeaMaker();
$shop = new TeaShop($teaMaker);

$shop->takeOrder('меньше сахара', 1);
$shop->takeOrder('больше молока', 2);
$shop->takeOrder('без сахара', 5);

$shop->serve();
```

7. Паттерн Proxy

Пользуетесь карточкой доступа, чтобы открыть дверь? Есть несколько вариантов открытия такой двери: это можно сделать с помощью карты доступа или же через нажатие кнопки, которая обходит безопасность. Главная функция двери – открываться, но также есть и дополнительный функционал.

Есть интерфейс двери:

```php
interface Door
{
    public function open();
    public function close();
}

class LabDoor implements Door
{
    public function open()
    {
        echo "Открывание";
    }

    public function close()
    {
        echo "Закрывание";
    }
}
```

Ее защита:

```php
class Security
{
```

```
    protected $door;

    public function __construct(Door $door)
    {
        $this->door = $door;
    }

    public function open($password)
    {
        if ($this->authenticate($password)) {
            $this->door->open();
        } else {
            echo "Это невозможно.";
        }
    }

    public function authenticate($password)
    {
        return $password === '$ecr@t';
    }

    public function close()
    {
        $this->door->close();
    }
}
```

Используем:

```
$door = new Security(new LabDoor());
$door->open('invalid'); // Это невозможно.

$door->open('$ecr@t'); // Открывание
$door->close(); // Закрывание
```

Также рекомендуем Вам посмотреть:

Шаблоны проектирования по-человечески: 6 порождающих паттернов, которые упростят жизнь
Шаблоны проектирования по-человечески: поведенческие паттерны в примерах
Лучший видеокурс по шаблонам проектирования
4 лучших книг о шаблонах проектирования
20 полезных навыков, которые можно освоить за 3 дня

Хотите получать больше интересных материалов с доставкой?

Подпишитесь на нашу рассылку:

Ваш Email

Подписаться

И не беспокойтесь, мы тоже не любим спам. Отписаться можно в любое время.

Читайте наши статьи в Telegram

Теги Разное

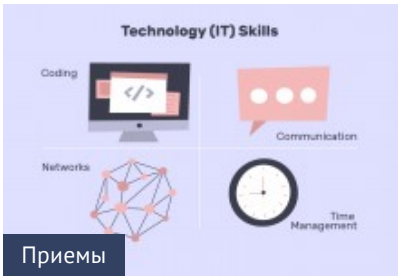
Предыдущая статья
Видеокурс по работе с MySQL

Следующая статья
164 крутых опенсорс проекта для новичков

Похожие статьи | Больше от автора

Как перестать писать плохой код на Bash: практические советы

ТОП-5 историй о том, как IT-консалтинг ускоряет бизнес



10 советов: как подтянуть разговорный английский язык

Комментариев: 2

Дмитрий Духнич 29.03.2019, в 11:33 ▲ 0 ▼

Я не шарю. Но похоже шаблон «Компоновщик» не верен. Тут у вас просто наследование.

Ответить

Mms99 Maslennikova 10.12.2018, в 19:42 ▲ 0 ▼

Шаблон «Компоновщик».
Я не php программист, мне не понятно, а где protected roles классов Designer и Developer?

Ответить

Комментарий:

Добавить

Свежие вакансии

Разместить вакансию

Plarium Krasnodar

Иллюстратор

Краснодар | полный день

В наш рекламный отдел мы ищем талантливого Иллюстратора, который умеет решать сложные задачи в установленные сроки.
Присоединяйся к нам и стань героем геймдева!

ОТКЛИКНУТЬСЯ НА ВАКАНСИЮ

Мел

Редактор спецпроектов

Москва | полный день

«Мелу» очень нужен человек с редакторским опытом, который будет придумывать, продюсировать и писать рекламные материалы на лучшем сайте об образовании и воспитании.

ОТКЛИКНУТЬСЯ НА ВАКАНСИЮ

dt group

CG / Пост-продакшн продюсер

Москва | полный день

з/п: 80000 ₽

Для создания новой команды, специализирующейся на производстве графических роликов в корпоративном сегменте, ищем голову — продюсера пост-продакшн — человека, умеющего управлять CG-производством, находить решения внутри и на аутсорсе, чувствовать клиента и производить качественный контент.

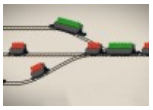
ОТКЛИКНУТЬСЯ НА ВАКАНСИЮ

Темы

Android Blockchain C# C++ Data Science Frontend Go Hacking iOS Java JavaScript Junior Linux Middle Mobile Python Senior Web Алгоритмы Базы данных Математика Новичку Общее Разное Разработка игр Советы Трудоустройство

Случайные статьи

Пошаговое руководство по взлому страницы ВКонтакте



Асинхронное программирование: концепция, реализация, примеры

ИИ шахматы на JavaScript в 5 этапов

Путь Data Scientist'а в 2017

Логические и математические задачи с собеседований

Популярные материалы

ТОП-8 трендов web-разработки, обязательных в 2019 году

14 советов, с которыми ты начнёшь мыслить как программист

ТОП-13 крутых идей веб-проектов для прокачки навыков

Хороший, спорный, злой Vue.js: опыт перехода с React

Твиттер на Vue.js: руководство для начинающих

Загрузить больше

О нас

Библиотека программиста — ваш источник образовательного контента в IT-сфере. Мы публикуем обзоры книг, видеолекции и видеоуроки, дайджесты и образовательные статьи, которые помогут вам улучшить процесс познания в разработке.

Подпишись

ВКонтакте | Telegram | Facebook | Instagram | Яндекс.Дзен

Медиакит | Пользовательское соглашение | Политика конфиденциальности

Связаться с нами

По вопросам рекламы: matvey@proglib.io

Для обратной связи: hello@proglib.io
123022, Москва, Рочдельская ул., 15, к. 17-18, +7 (995) 114-98-90

© Библиотека программиста, 2016-2019. При копировании материала ссылка на источник обязательна.