Design Pattern

Nội dung

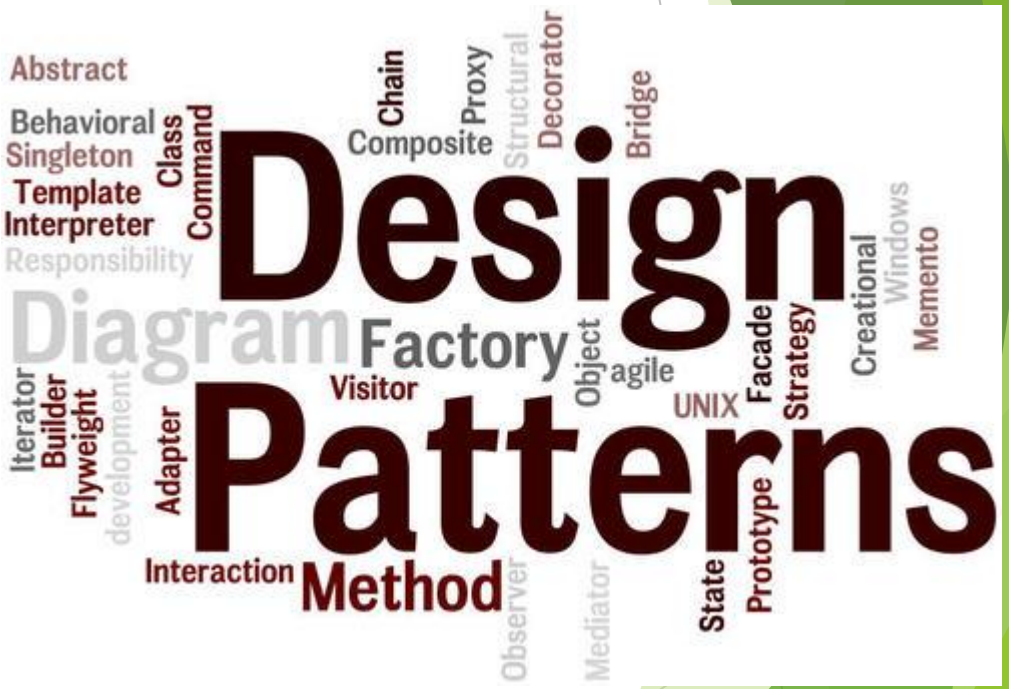
1. Giới thiệu Design Pattern
2. Các Nhóm Design Pattern
3. Design Pattern trong Laravel

1

Giới thiệu Design Pattern

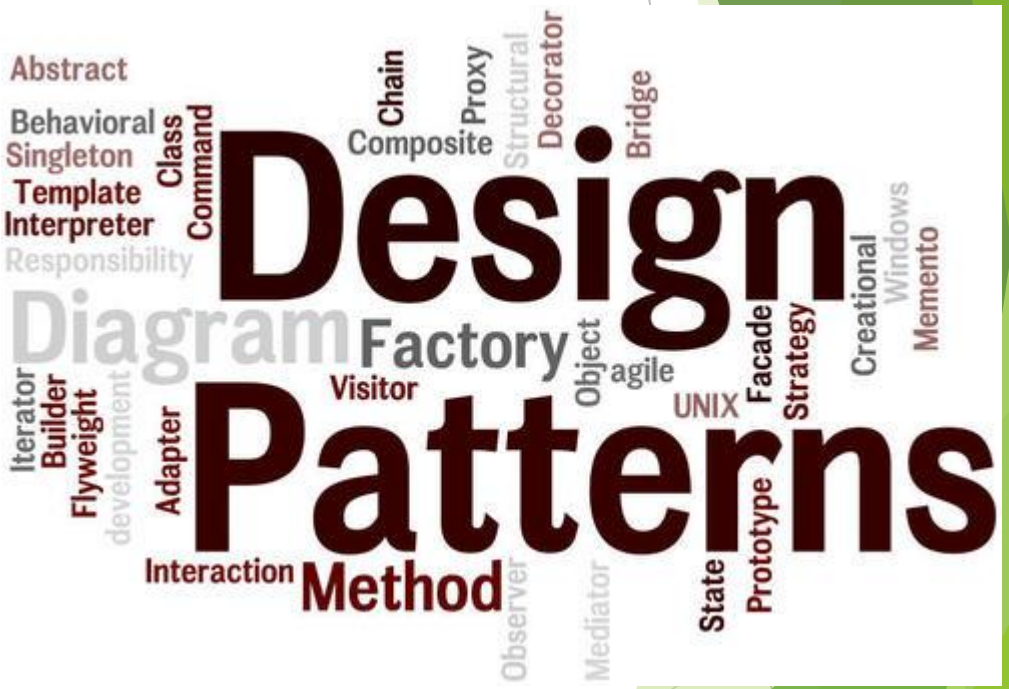
Giới thiệu Design Pattern

Năm 1994, bốn tác giả của GoF (Erich Gamma, Richard Helm, Ralph Johnson và John Vlissides) đã cho xuất bản một cuốn sách với tiêu đề Design Patterns – Elements of Reusable Object-Oriented Software, đây là khởi nguồn của khái niệm design pattern trong lập trình phần mềm.



Giới thiệu PHP

Design Pattern là một giải pháp tổng thể cho các vấn đề chung trong thiết kế phần mềm. Một mẫu thiết kế không phải là một thiết kế hoàn thiện để mà có thể được chuyển đổi trực tiếp thành mã; nó chỉ là template mô tả cách giải quyết một vấn đề mà có thể được dùng trong nhiều tình huống khác nhau.



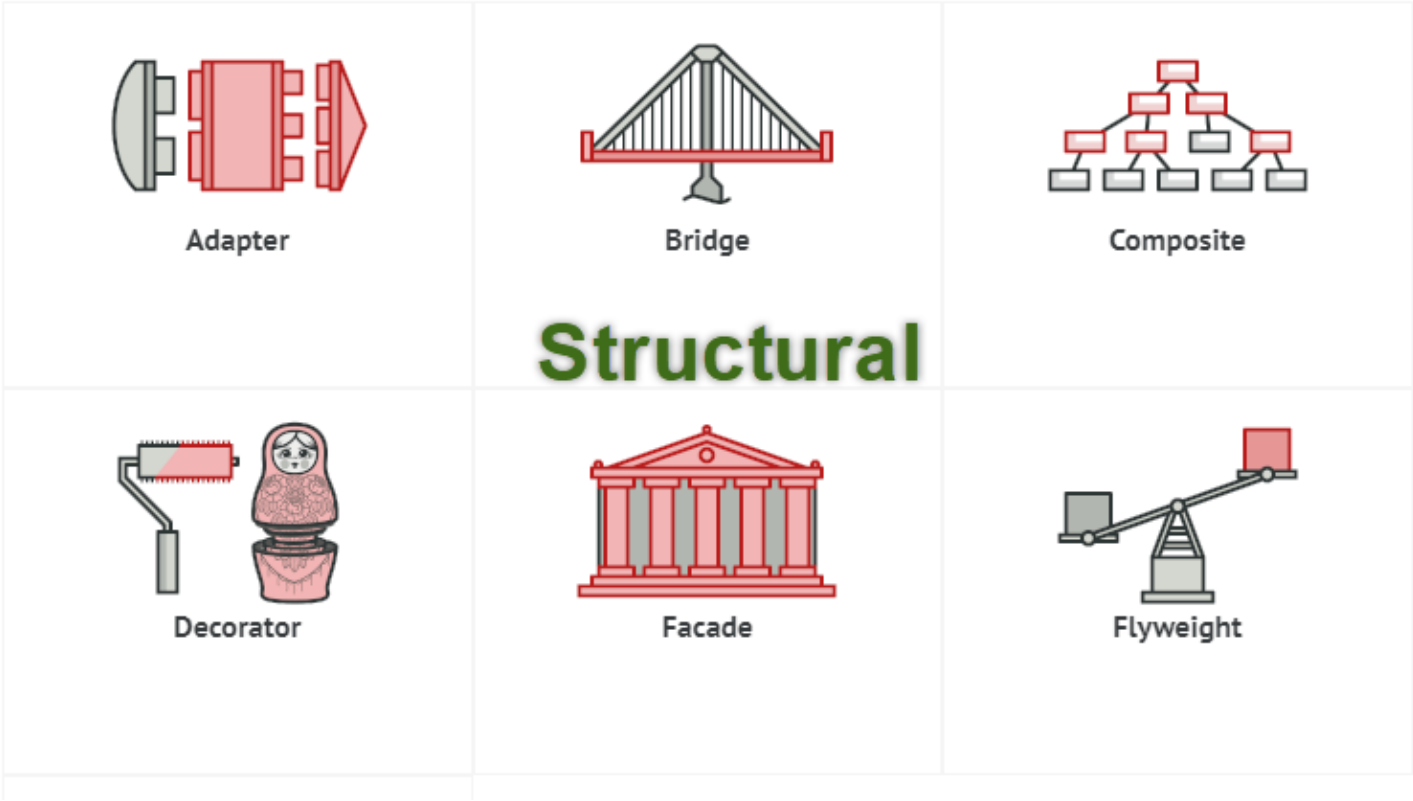
2

Các nhóm Design Pattern

Creational (Nhóm tạo)

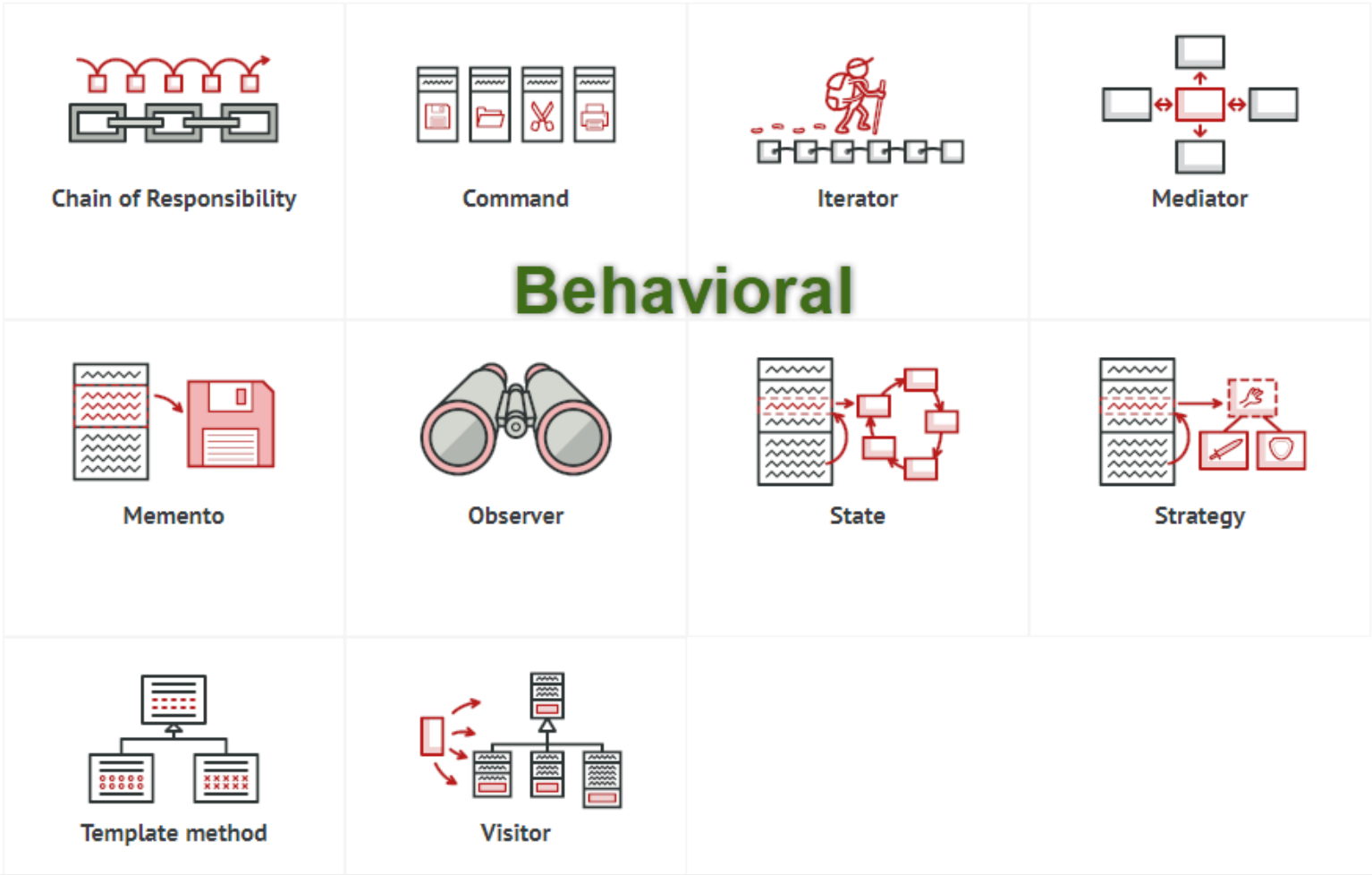
Giúp khởi tạo các đối tượng, cung cấp các thủ thuật để khởi tạo đối tượng linh hoạt và hiệu quả (Không cần đến từ khóa **new** như thông thường).

Structural (nhóm cấu trúc)



Liên quan đến xử lý mối quan hệ thành phần (cấu trúc) giữa các đối tượng, giúp tương tác với tập hợp đối tượng linh hoạt và dễ dàng hơn.

Behavioral (nhóm hành vi/ tương tác)



Liên quan đến xử lý mối quan hệ giao tiếp/tương tác giữa các đối tượng, giúp tương tác các đối tượng linh hoạt và dễ dàng hơn.

3

Các design pattern trong Laravel

Một số pattern chính

1. Dependency Injection (Service Container/Service Provider)
2. Repository pattern
3. Contract, Facade, Singleton,…

Dependency Injection

```php
// Non Dependency Injection
Class A
{
    protected $classB;
    public function __construct()
    {
        $this->classB = new ClassB();
    }
}

// Dependency Injection
Class A
{
    protected $classB;
    public function __construct(ClassB $classB)
    {
        $this->classB = $classB;
    }
}
```

Dependency Injection

```php
class UserController extends Controller
{
    protected $users;

    public function __construct(UserRepository $users)
    {
        $this->users = $users;
    }

    public function show($id)
    {
        $user = $this->users->find($id);

        return view('user.profile', ['user' => $user]);
    }
}
```

With out Repository (Use Model directly)

```php
class PostController extends BaseController {

  public function index()
  {
    $posts = Post::paginate(20);
    return View::make('post.index', compact('posts'));
  }

  public function show($id)
  {
    $post = Post::findOrFail($id);
    return View::make('post.show', compact('post'));
  }

  // ... etc

}
```

Nhược điểm ?

With Repository

```php
class PostController extends BaseController {

  private $postRepository;

  public function __construct(PostRepository $postRepository = null)
  {
          // Dependancy Injection
  }

  public function index()
  {
    $posts = $this->postRepository->paginate(20);
    return View::make('post.index', compact('posts'));
  }

  public function show($id)
  {
    $posts = $this->postRepository->findOrFail($id);
    return View::make('post.show', compact('post'));
  }

  // ... etc

}
```

```php
class PostRepository {
  public function paginate($perPage = null, $columns = array('*'))
  {
    return Post::paginate($perPage, $columns);
  }

  public function findOrFail($id, $columns = array('*'))
  {
    return Post::findOrFail($id, $columns);
  }
  // ...etc
}
```

Tài liệu tham khảo

- https://refactoring.guru/design-patterns
- https://www.sitepoint.com/how-laravel-facades-work-and-how-to-use-them-elsewhere/
- https://www.script-tutorials.com/design-patterns-in-php/