

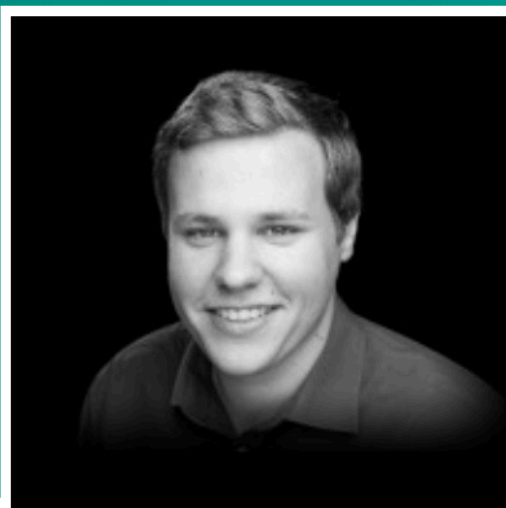
Laravel Design Patterns



About Me

Bobby Bouwmann
@bobbybouwmann
Enrise - Code Cuisine
markdownmail.com
Laracasts





bobbybouwmann  

DEVELOPER AT ENRISE
Apeldoorn, The Netherlands 

[HIRE ME](#)

EXPERIENCE
1,062,865
1259 BEST REPLY AWARDS

Member Since **3 Years Ago**

1241 Lessons Completed

14 Favorites

[EDIT PROFILE](#)

11TH MAY, 2018

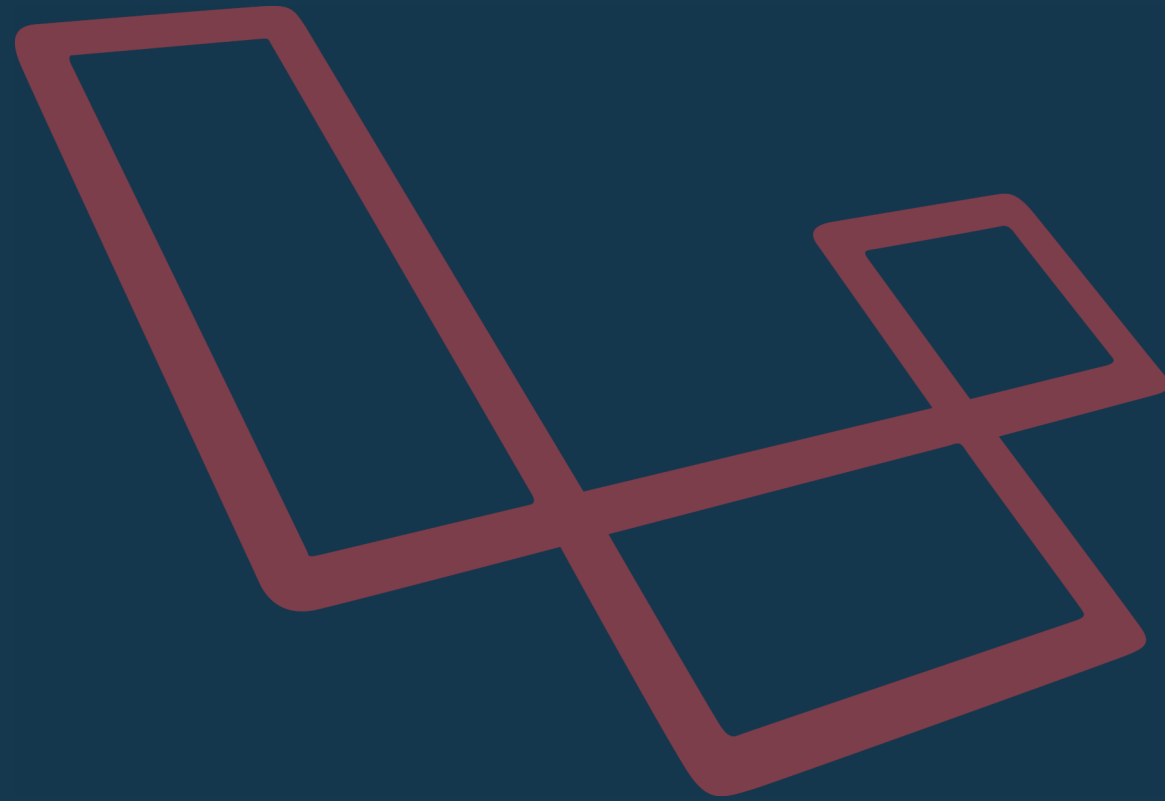


bobbybouwmann left a reply on [Passport And Auth::logout\(\) Produce An Error](#) • 15 HOURS AGO

So it seems that `Auth::` is returning the `RequestGuard` class. If you look in the code, this class indeed doesn't have a `logout` method.

Also when using password, you work with tokens right? You can't really logout, you can only invalidate the token or let the token expire (e.g. with `setTTL` or `setExpireAt`).

What is Laravel?



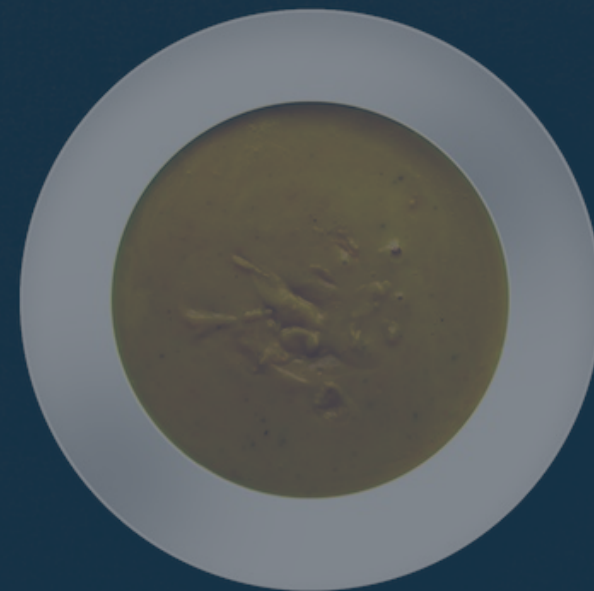
Laravel Design Patterns



Daily Design Patterns



What are Design Patterns?



Design Patterns are ...

Recipes for building maintainable code



Be your own Chef

Using the boundaries of Design Pattern



Why use Design Patterns?

- Provides a way to solve issues related to software development using a proven solution
- Makes communication between developer more efficient

Why use Design Patterns?

- There is no "silver bullet" for using design pattern, each project and situation is different

Maintainable Code

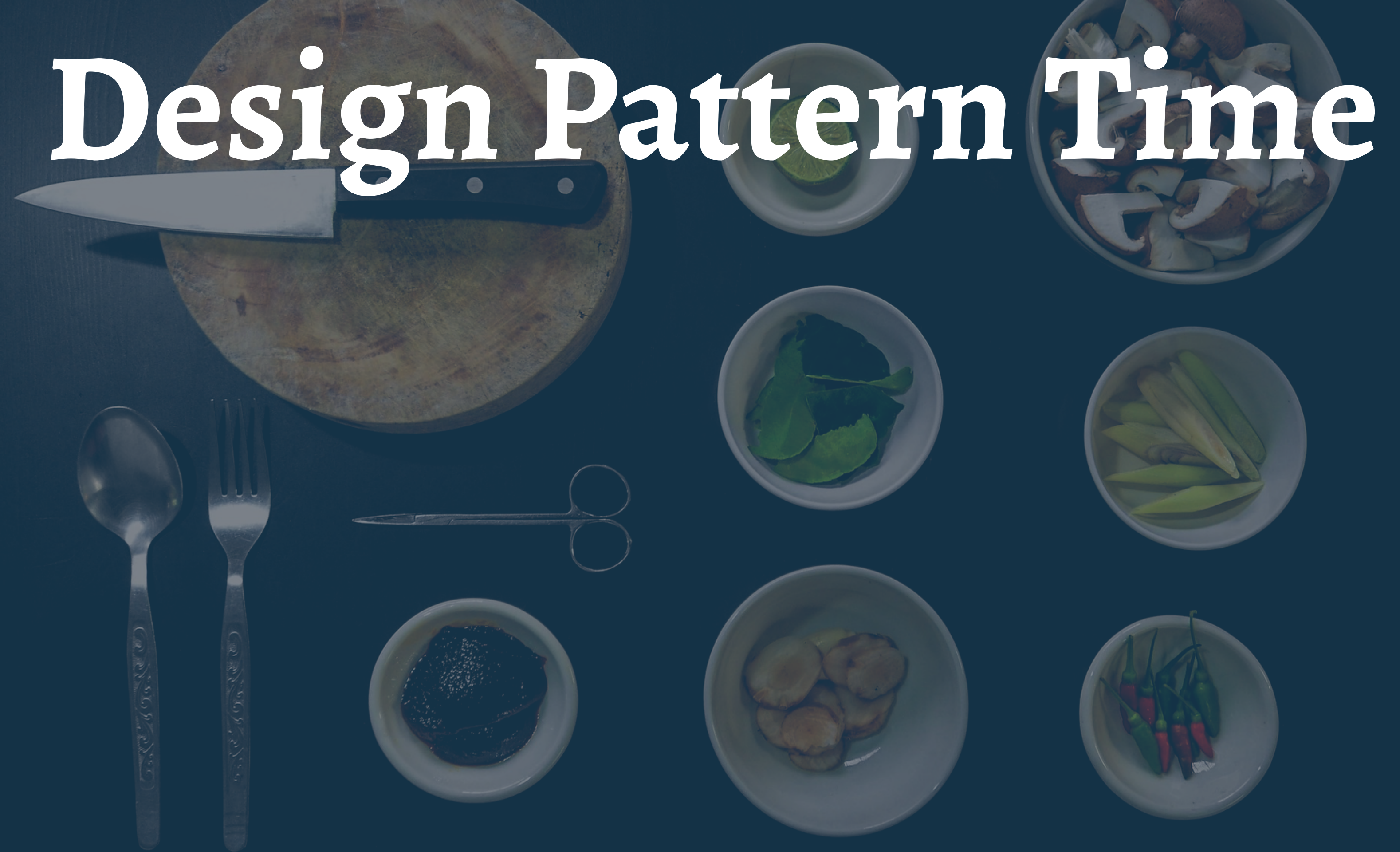
The background image shows a row of glass vials or test tubes on a wooden surface. The vials contain liquids of different colors: red, purple, and orange. The lighting is soft, and the focus is slightly blurred, giving it a vintage or scientific feel. The title 'Maintainable Code' is written in a large, white, serif font across the top of the image.

Maintainable Code

- Understandable
- Intuitive
- Adaptable
- Extendable
- Debuggable

Maintainable Code

- Don't Repeat Yourself
- Keep It Simple Stupid



Design Pattern Time

Design Pattern types

- Creational Patterns
- Structural Patterns
- Behavioural Patterns

Creational Patterns

- Class instantiation
- Hiding the creation logic
- Examples: Factory, Builder, Prototype, Singleton

Structural Patterns

- Composition between objects
- Usages of interfaces, abstract classes
- Examples: Adapter, Bridge, Decorator, Facade, Proxy

Behavioural Patterns

- Communication between objects
- Usages of mostly interfaces
- Examples: Command, Iterator, Observer, State, Strategy

Design Patterns in Laravel

- Factory Pattern
- Builder (Manager) Pattern
- Strategy Pattern
- Provider Pattern
- Repository Pattern
- Iterator Pattern
- Singleton Pattern
- Presenter Pattern

Design Patterns in Laravel

- Factory Pattern
- Builder (Manager) Pattern
- Strategy Pattern
- Provider Pattern

Factory Pattern

The background of the slide is an abstract composition of overlapping, translucent shapes in warm, earthy tones. The colors range from deep reds and oranges to lighter yellows and browns. The shapes are irregular and layered, creating a sense of depth and movement. The overall effect is a textured, painterly background that complements the title text.







ITALIA
08 1041
CE

ITALIA
08 1041
CE

16
GEN
ITALIA
08 1041
CE

GEN 16

ITALIA
08 1041
CE

GEN 16



Factory Pattern

Provides an interface for creating objects without specifying their concrete classes.

Factory Pattern

- Decouples code
- Factory is responsible for creating objects, not the client
- Multiple clients call the same factory, one place for changes
- Easier to test, easy to mock and isolate

Factory Pattern Example

```
interface PizzaFactoryContract
{
    public function make(array $toppings = []): Pizza;
}

class PizzaFactory implements PizzaFactoryContract
{
    public function make(array $toppings = []): Pizza
    {
        return new Pizza($toppings);
    }
}

$pizza = (new PizzaFactory)->make(['chicken', 'onion']);
```


Factory Pattern Laravel Example

```
class PostsController
{
    public function index(): \Illuminate\View\View
    {
        $posts = Post::all();

        return view('posts.index', ['posts' => $posts]);
    }
}
```

```
// Illuminate\Foundation\helpers.php

/**
 * @return \Illuminate\View\View|\Illuminate\Contracts\View\Factory
 */
function view($view = null, $data = [], $mergeData = [])
{
    $factory = app(ViewFactory::class);

    if (func_num_args() === 0) {
        return $factory;
    }

    return $factory->make($view, $data, $mergeData);
}
```



```
/**
 * Get the evaluated view contents for the given view.
 *
 * @return \Illuminate\Contracts\View\View
 */
public function make($view, $data = [], $mergeData = [])
{
    $path = $this->finder->find(
        $view = $this->normalizeName($view)
    );

    $data = array_merge($mergeData, $this->parseData($data));

    return new View($this, $this->getEngineFromPath($path), $view, $path, $data);
}
```

```
/**
 * Create a new Validator instance.
 *
 * @return \Illuminate\Validation\Validator
 */
public function make(array $data, array $rules, array $messages = [], array $customAttributes = [])
{
    $validator = $this->resolve(
        $data, $rules, $messages, $customAttributes
    );

    $this->addExtensions($validator);

    return $validator;
}

/**
 * Resolve a new Validator instance.
 *
 * @return \Illuminate\Validation\Validator
 */
protected function resolve(array $data, array $rules, array $messages, array $customAttributes)
{
    if (is_null($this->resolver)) {
        return new Validator($this->translator, $data, $rules, $messages, $customAttributes);
    }

    return call_user_func($this->resolver, $this->translator, $data, $rules, $messages, $customAttributes);
}
```


Builder (Manager) Pattern



Today's Specials!



\$9.99

♥ Create your
Own Pizza

\$11.99
each

♥ Chicken Marsala
asparagus, soup of
the day, mashed potatoes

♥ Salmon Asparagus



LET'S MAKE PIZZA!

KEEPING IT SIMPLE (AKA MARGHERITA)

Top a Bakers Delight Pizza Base with:

- Leggos Sugo di Pomodoro
- Perfect Italiano Pizza Plus cheese
- Handful of cherry tomatoes, halved
- Chopped parsley

Throw in the oven and cook for 4 -5 minutes then garnish with shaved parmesan, fresh basil and some rocket

GETTING A LITTLE BIT FANCY

Top a Bakers Delight Pizza Base with:

- Perfect Italiano ricotta
- Perfect Italiano parmesan
- Leggos sundried tomato pesto
- Handful of sautéed portobello mushrooms
- Perfect Italiano Pizza Plus cheese

Throw into the oven and cook for 4-5 minutes then garnish with pancetta and fresh basil

WHAT'S FOR EATS SPECIAL

Top a Bakers Delight Pizza Base with:

- Leggos Vine Ripe tomato and basil flavour
- Handful of caramelised onions
- Handful of sautéed portobello mushrooms
- Perfect Italiano Pizza Plus cheese

Throw into the oven and cook for 4-5 minutes then garnish with pancetta and fresh basil



Builder (Manager) Pattern

Builds complex objects step by step. It can return different objects based on the given data.

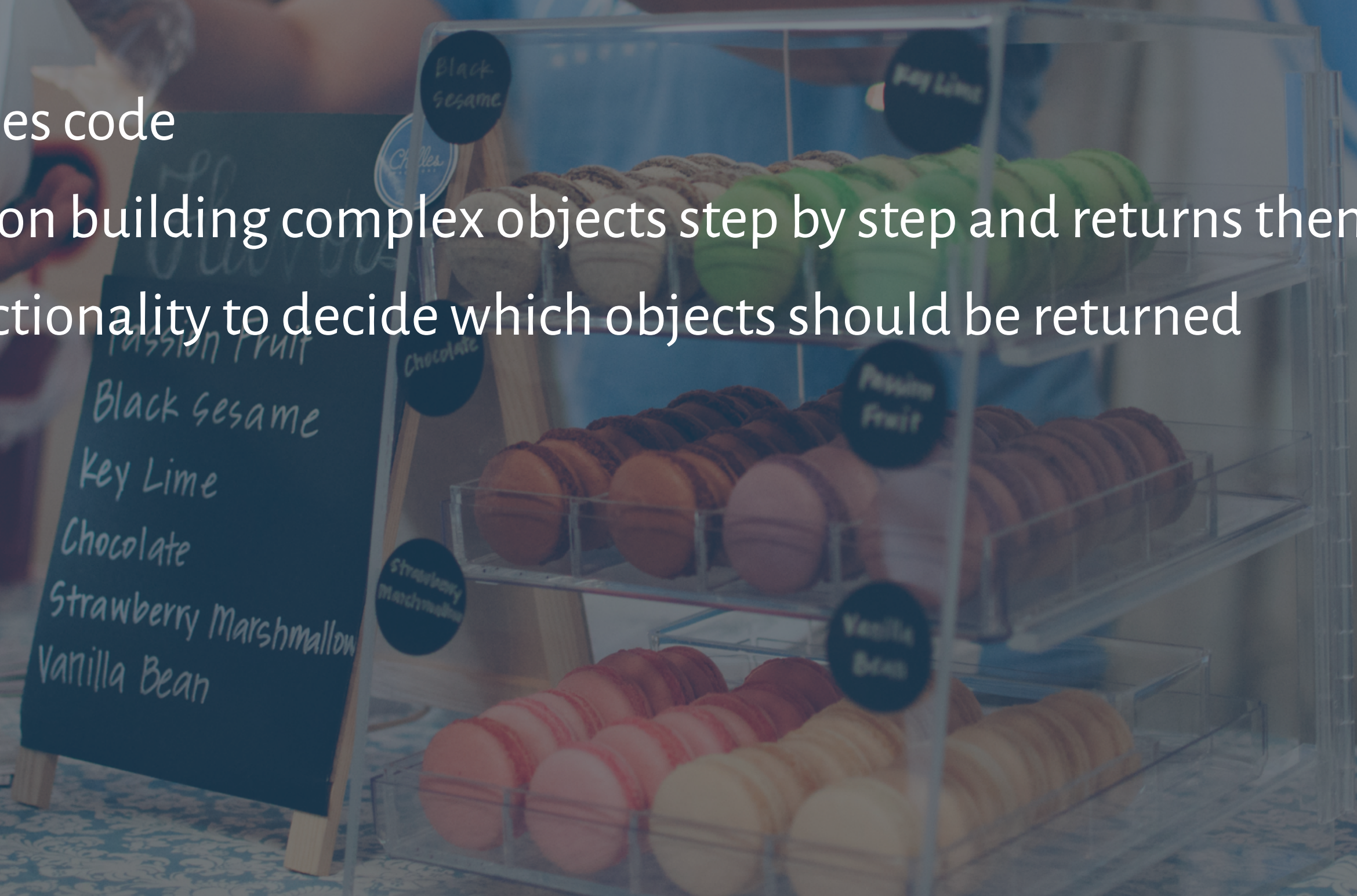
A photograph of a bakery display featuring a three-tiered clear acrylic stand filled with macarons in various colors. To the left of the stand is a small black chalkboard menu on a wooden stand. The background shows a person in a blue uniform and a blue sign with the word 'Che' in cursive. The entire image has a semi-transparent dark blue overlay.

Flavors

- Passion Fruit
- Black Sesame
- Key Lime
- Chocolate
- Strawberry Marshmallow
- Vanilla Bean

Builder (Manager) Pattern

- Decouples code
- Focuses on building complex objects step by step and returns them
- Has functionality to decide which objects should be returned



Builder Pattern Example

```
class PizzaBuilder
{
    public function make(PizzaBuilderInterface $pizza): Pizza
    {
        // Returns object of type Pizza
    }
}
```



```
class PizzaBuilder
{
    public function make(PizzaBuilderInterface $pizza): Pizza
    {
        return $pizza
            ->prepare()
            ->applyToppings()
            ->bake();
    }
}
```



```
interface PizzaBuilderInterface
{
    public function prepare(): Pizza;

    public function applyToppings(): Pizza;

    public function bake(): Pizza;
}
```



```
class MargarithaBuilder implements PizzaBuilderInterface
{
    protected $pizza;

    public function prepare(): Pizza
    {
        $this->pizza = new Pizza();

        return $this->pizza;
    }

    public function applyToppings(): Pizza
    {
        $this->pizza->setToppings(['cheese', 'tomato']);

        return $this->pizza;
    }

    public function bake(): Pizza
    {
        $this->pizza->setBakingTemperature(180);
        $this->pizza->setBakingMinutes(8);

        return $this->pizza;
    }
}
```



```
class PizzaBuilder
{
    public function make(PizzaBuilderInterface $pizza): Pizza
    {
        return $pizza
            ->prepare();
            ->applyToppings();
            ->bake();
    }
}
```

```
// Create a PizzaBuilder
$pizzaBuilder = new PizzaBuilder;
```

```
// Create pizza using Builder which returns Pizza instance
$pizza = $pizzaBuilder->make(new MargarithaBuilder());
```

```
class ChickenBuilder implements PizzaBuilderInterface
{
    protected $pizza;

    public function prepare(): Pizza
    {
        $this->pizza = new Pizza();

        return $this->pizza;
    }

    public function applyToppings(): Pizza
    {
        $this->pizza->setToppings(['cheese', 'tomato', 'chicken', 'corn']);

        return $this->pizza;
    }

    public function bake(): Pizza
    {
        $this->pizza->setBakingTemperature(200)
        $this->pizza->setBakingMinutes(8);

        return $this->pizza;
    }
}
```



```
class PizzaBuilder
{
    public function make(PizzaBuilderInterface $pizza): Pizza
    {
        return $pizza
            ->prepare()
            ->applyToppings()
            ->bake();
    }
}
```

```
$pizzaBuilder = new PizzaBuilder;
```

```
$pizzaOne = $pizzaBuilder->make(new MargarithaBuilder());
$pizzaTwo = $pizzaBuilder->make(new ChickenBuilder());
```


Builder Pattern Laravel

Flavors

- Passion Fruit
- Black Sesame
- Key Lime
- Chocolate
- Strawberry Marshmallow
- Vanilla Bean




```
class PizzaManager extends Illuminate\Support\Manager
{
    public function getDefaultDriver()
    {
        return 'margaritha';
    }

    public function createMargarithaDriver(): PizzaBuilderInterface
    {
        return new MargarithaBuilder();
    }

    public function createChickenPizzaDriver(): PizzaBuilderInterface
    {
        return new ChickenPizzaBuilder();
    }
}
```

```
public function driver($driver = null)
{
    $driver = $driver ?: $this->getDefaultDriver();

    // If the given driver has not been created before, we will create the instances
    // here and cache it so we can return it next time very quickly. If there is
    // already a driver created by this name, we'll just return that instance.
    if (! isset($this->drivers[$driver])) {
        $this->drivers[$driver] = $this->createDriver($driver);
    }

    return $this->drivers[$driver];
}
```



```
protected function createDriver($driver)
{
    $method = 'create' . Str::studly($driver) . 'Driver';

    if (method_exists($this, $method)) {
        return $this->$method();
    }

    throw new InvalidArgumentException("Driver [$driver] not supported.");
}
```

```
class PizzaManager extends Illuminate\Support\Manager
{
    public function getDefaultDriver()
    {
        return 'margaritha';
    }

    public function createMargarithaDriver(): PizzaBuilderInterface
    {
        return new MargarithaBuilder();
    }
}
```

```
$manager = new PizzaManager();
```

```
// MargarithaBuilder implementing PizzaBuilderInterface
$builder = $manager->driver('margaritha');
```

```
$pizza = $builder->make(); // Pizza
```



```
// Illuminate\Mail\TransportManager
class TransportManager extends Manager
{
    protected function createSmtplibDriver()
    {
        // Code for building up a SmtplibTransport class
    }

    protected function createMailgunDriver()
    {
        // Code for building up a MailgunTransport class
    }

    protected function createSparkpostDriver()
    {
        // Code for building up a SparkpostTransport class
    }

    protected function createLogDriver()
    {
        // Code for building up a LogTransport class
    }
}
```

```
class TransportManager extends Manager
{
    public function getDefaultDriver()
    {
        return $this->app['config']['mail.driver'];
    }

    // All create{$driver}Driver methods
}
```



```
// config/mail.php  
'driver' => env('MAIL_DRIVER', 'smtp'),
```

```
// config/mail.php  
'driver' => env('MAIL_DRIVER', 'smtp'),
```

```
// Illuminate\Mail\TransportManager  
createSmtpDriver()
```



```
// config/mail.php
'driver' => env('MAIL_DRIVER', 'smtp'),

// Illuminate\Mail\TransportManager
createSmtpDriver()

// app/Http/Controllers/RegisterController
Mail::to($user)
    ->send(new \App\Mail\UserRegistered($user));
```

Laravel Manager Pattern Examples

Illuminate\Auth\AuthManager

Illuminate\Broadcasting\BroadcastManager

Illuminate\Cache\CacheManager

Illuminate\Filesystem\FilesystemManager

Illuminate\Hashing\HashManager

Illuminate\Notifications\ChannelManager

Illuminate\Queue\QueueManager

Illuminate\Session\SessionManager


```
class HashManager extends Manager implements Hasher
{
    public function getDefaultDriver()
    {
        return $this->app['config']['hashing.driver'] ?? 'bcrypt';
    }

    public function createBcryptDriver()
    {
        return new BcryptHasher($this->app['config']['hashing.bcrypt'] ?? []);
    }

    public function createArgonDriver()
    {
        return new ArgonHasher($this->app['config']['hashing.argon'] ?? []);
    }
}
```

Strategy Pattern





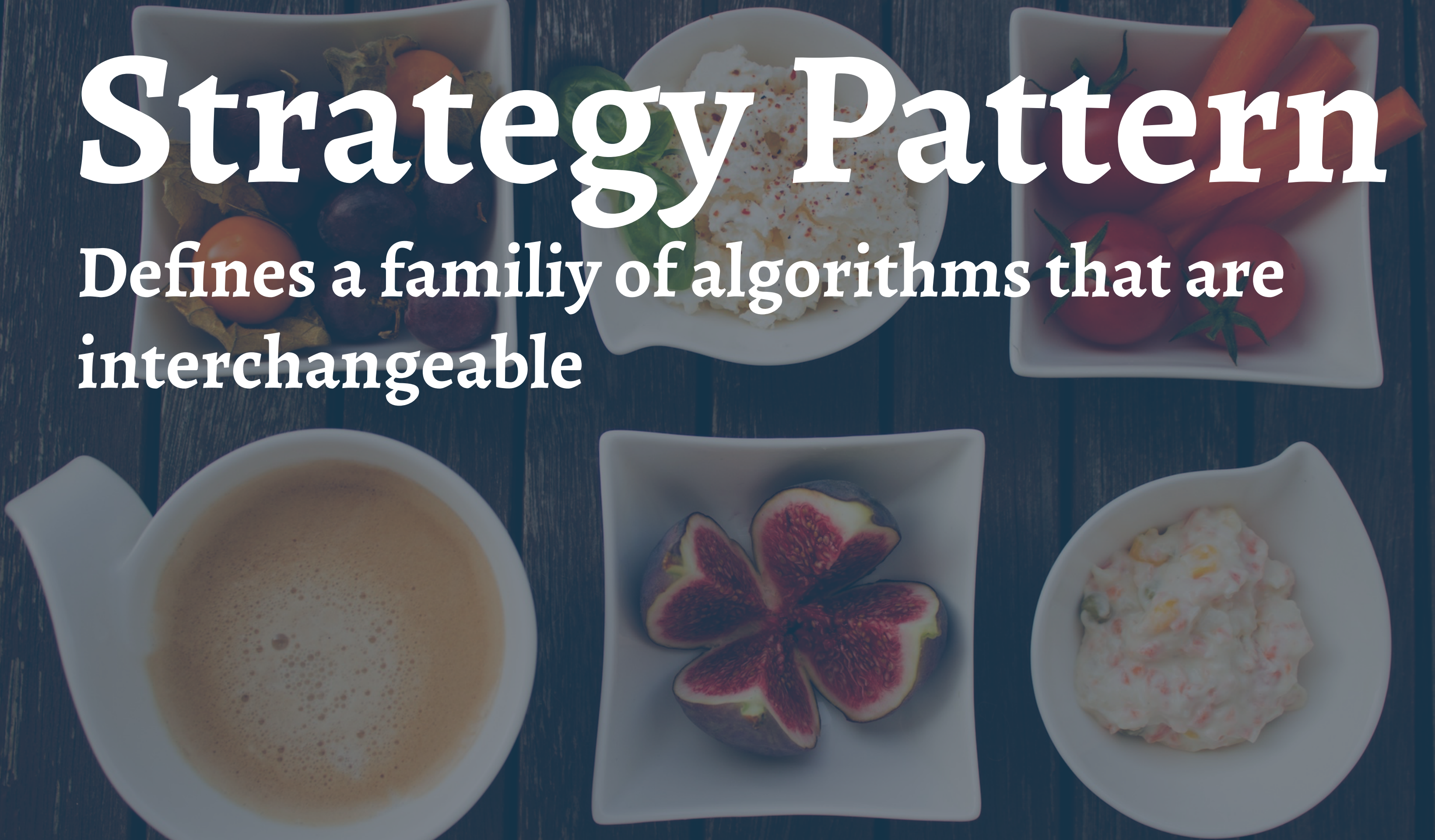






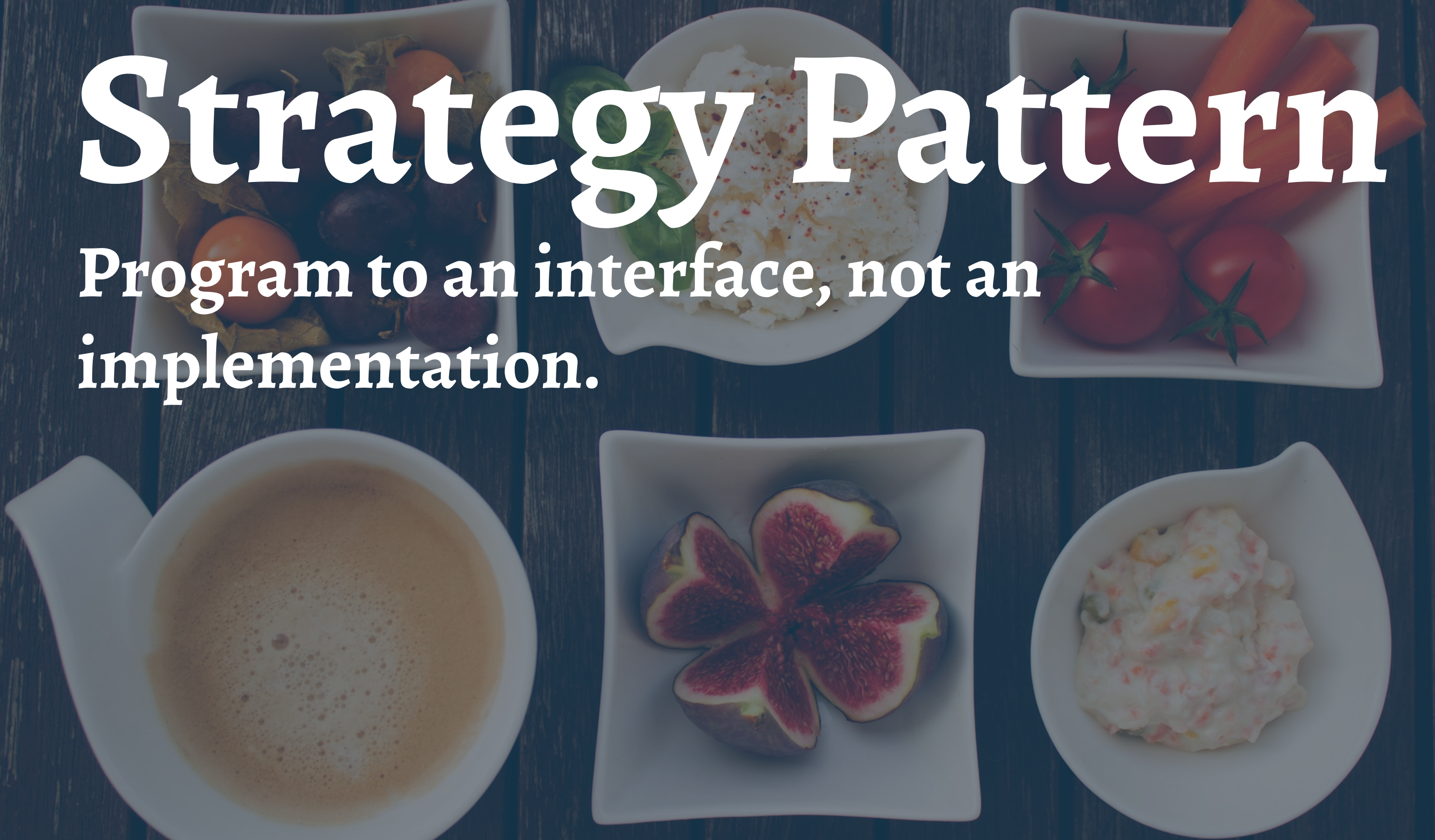
Strategy Pattern

Defines a family of algorithms that are interchangeable



Strategy Pattern

Program to an interface, not an implementation.



Strategy Pattern Example

```
interface DeliveryStrategy
{
    public function deliver(Address $address);
}
```

```
interface DeliveryStrategy
{
    public function deliver(Address $address);
}

class BikeDelivery implements DeliveryStrategy
{
    /**
     * Perform the delivery
     *
     * @param Address $address
     */
    public function deliver(Address $address):
    {
        $route = new BikeRoute($address);

        echo $route->calculateCosts();
        echo $route->calculateDeliveryTime();
    }
}
```



```
class PizzaDelivery
{
    public function deliverPizza(DeliveryStrategy $strategy, Address $address)
    {
        return $strategy->deliver($address);
    }
}
```

```
$address = new Address('Kalvarijų gatvė 137, Vilnius');
$delivery = new PizzaDelivery();
```

```
$delivery->deliver(new BikeDelivery(), $address);
```

```
class DroneDelivery implements DeliveryStrategy
{
    public function deliver(Address $address):
    {
        $route = new DroneRoute($address);

        echo $route->calculateCosts();
        echo $route->calculateFlyTime();
    }
}
```

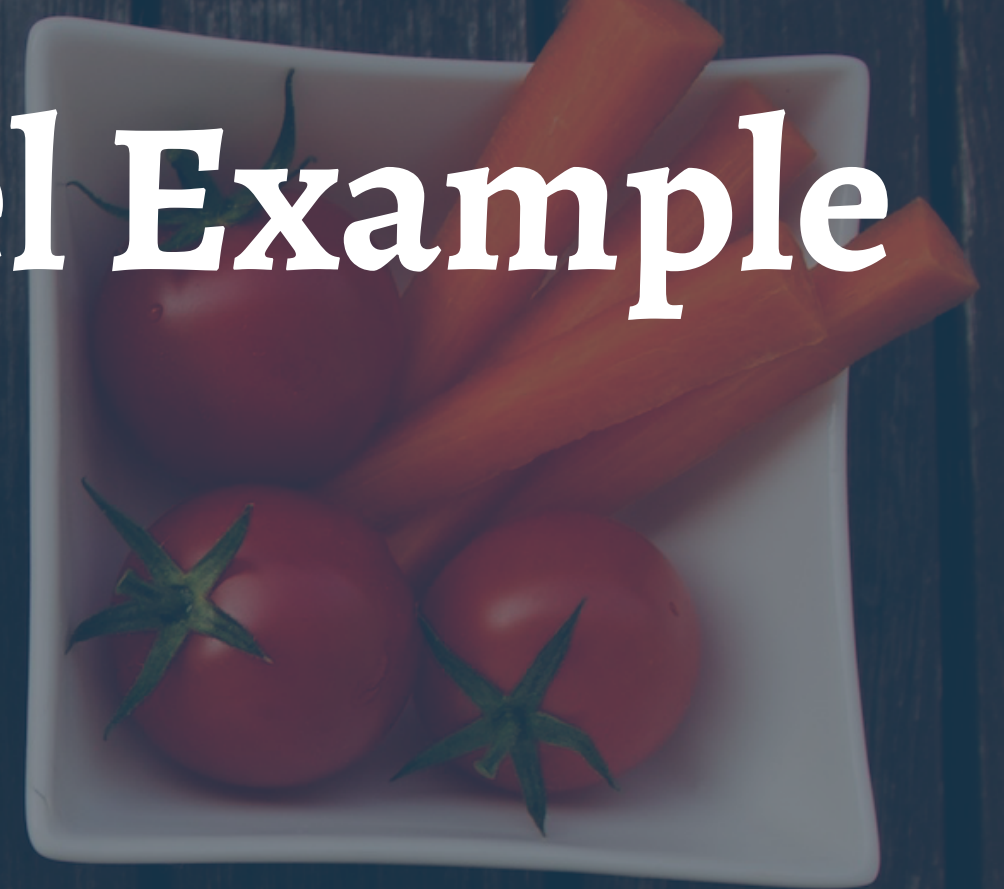


```
class PizzaDelivery
{
    public function deliverPizza(DeliveryStrategy $strategy, Address $address)
    {
        return $strategy->deliver($address);
    }
}
```

```
$address = new Address('Meer en Vaart 300, Amsterdam');
$delivery = new PizzaDelivery();
```

```
$delivery->deliver(new BikeDelivery(), $address);
$delivery->deliver(new DroneDelivery(), $address);
```

Strategy Pattern Laravel Example




```
<?php

namespace Illuminate\Translation;

use Illuminate\Support\ServiceProvider;

class TranslationServiceProvider extends ServiceProvider
{
    /**
     * Register the service provider.
     *
     * @return void
     */
    public function register()
    {
        $this->app->singleton('translator', function ($app) {
            $loader = new FileLoader($app['files'], $app['path.lang']);

            // When registering the translator component, we'll need to set the default
            // locale as well as the fallback locale. So, we'll grab the application
            // configuration so we can easily get both of these values from there.
            $locale = $app['config']['app.locale'];

            $trans = new Translator($loader, $locale);

            $trans->setFallback($app['config']['app.fallback_locale']);

            return $trans;
        });
    }
}
```

```
$address = new Address('Meer en Vaart 300, Amsterdam');  
$bikeDelivery = new BikeDelivery();
```

```
$delivery = new PizzaDelivery();  
$delivery->deliver($bikeDelivery, $address);
```

```
$loader = new FileLoader($app['files'], $app['path.lang']);  
$locale = $app['config']['app.locale'];
```

```
$trans = new Translator($loader, $locale);  
$trans->trans('pagination.next');
```



```
$fileLoader = new \Illuminate\Translation\FileLoader(  
    app('files'), app('app.lang')  
);
```

```
$translator = new \Illuminate\Translation\Translator(  
    $fileLoader, 'en'  
);
```

```
$arrayLoader = new \Illuminate\Translation\ArrayLoader();  
$translator = new \Illuminate\Translation\Translator(  
    $arrayLoader, 'en'  
);
```

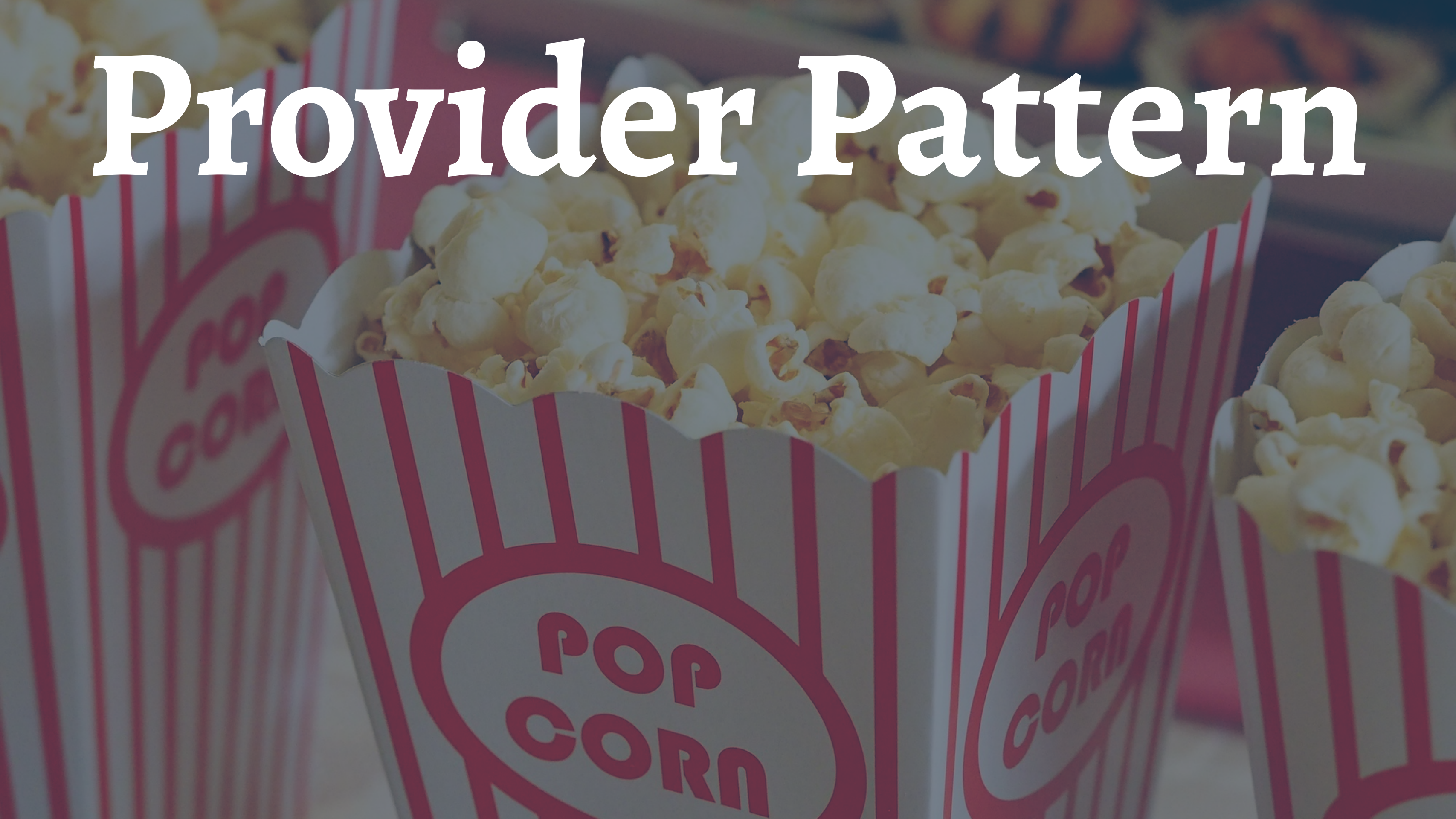
```
public function testFileUploads()
{
    $trans = new \Illuminate\Translation\Translator(
        new \Illuminate\Translation\ArrayLoader, 'en'
    );

    $file = new File(__FILE__, false);

    $validator = new Validator($trans, ['file' => $file], ['file' => 'Required|mimes:xls']);

    $this->assertFalse($validator->passes());
}
```


Provider Pattern



OAK
PIZZA

We Make People Happy
with Pizza

SUPER
fresh
SUPER
fast

Fresh thickened Corn

Crushed Tomato Sauce





green olives

black olives

artichokes

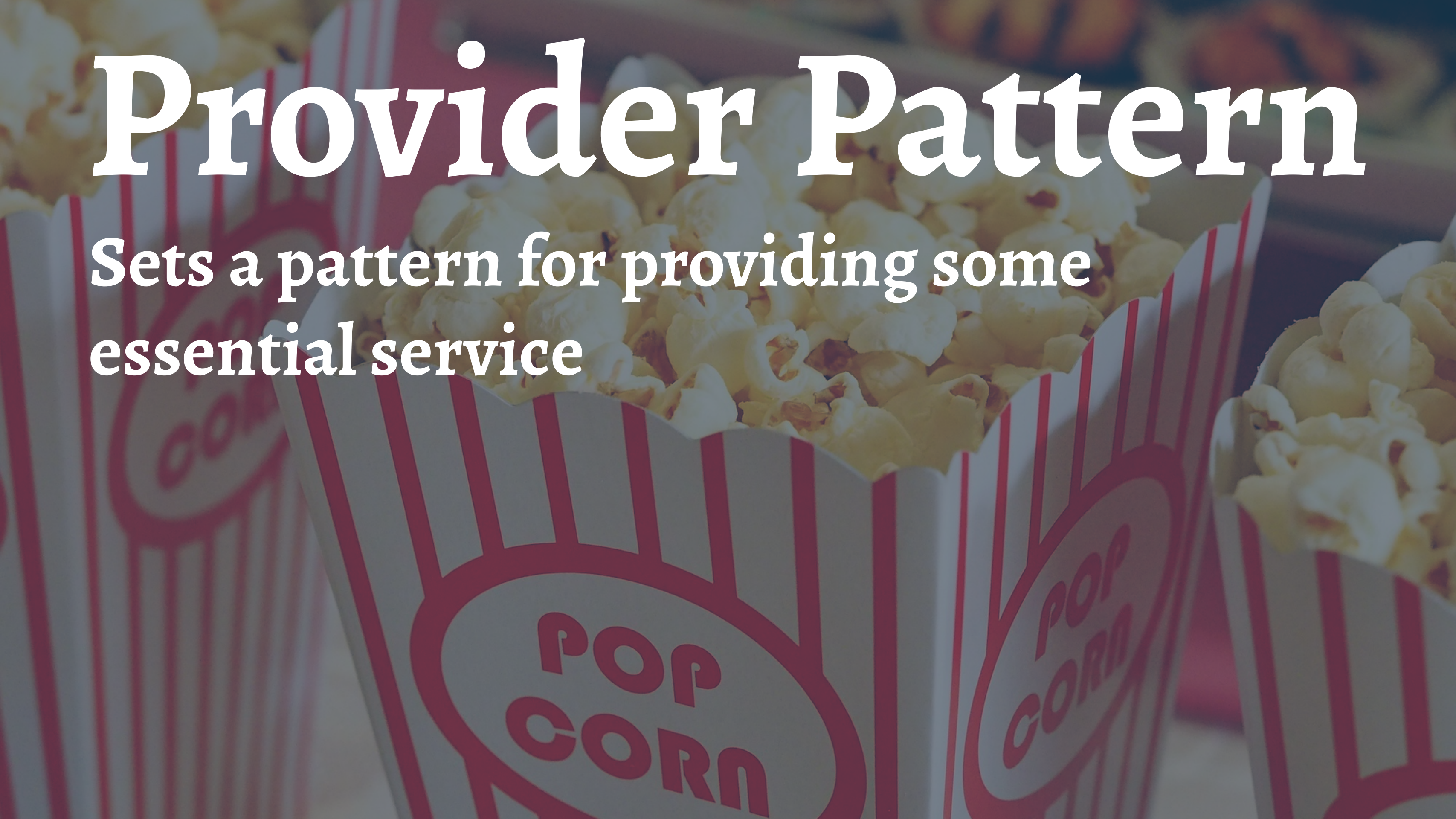
red peppers

green bell peppers

chopped garlic

Provider Pattern

Sets a pattern for providing some essential service



Provider Pattern Example

```
class DominosServiceProvider extends ServiceProvider
{
    public function register()
    {
        // Register your services here
    }
}
```

```
use App\Dominos\PizzaManager;

class PizzaServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->bind('pizza-manager', function ($app) {
            return new PizzaManager();
        });
    }
}
```



```
// Using instantiation
$pizzaManager = new \App\Dominos\PizzaManager();

// Using the provider pattern and the container
$pizzaManager = app('pizza-manager');

$margaritha = $pizzaManager->driver('margaritha');
$chicken = $pizzaManager->driver('chicken-pizza');
```

Provider Pattern Laravel Example




```
class DebugbarServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->singleton('debugbar', function ($app) {
            $debugbar = new LaravelDebugbar($app);

            if ($app->bound(SessionManager::class)) {
                $sessionManager = $app->make(SessionManager::class);
                $httpDriver = new SymfonyHttpDriver($sessionManager);
                $debugbar->setHttpDriver($httpDriver);
            }

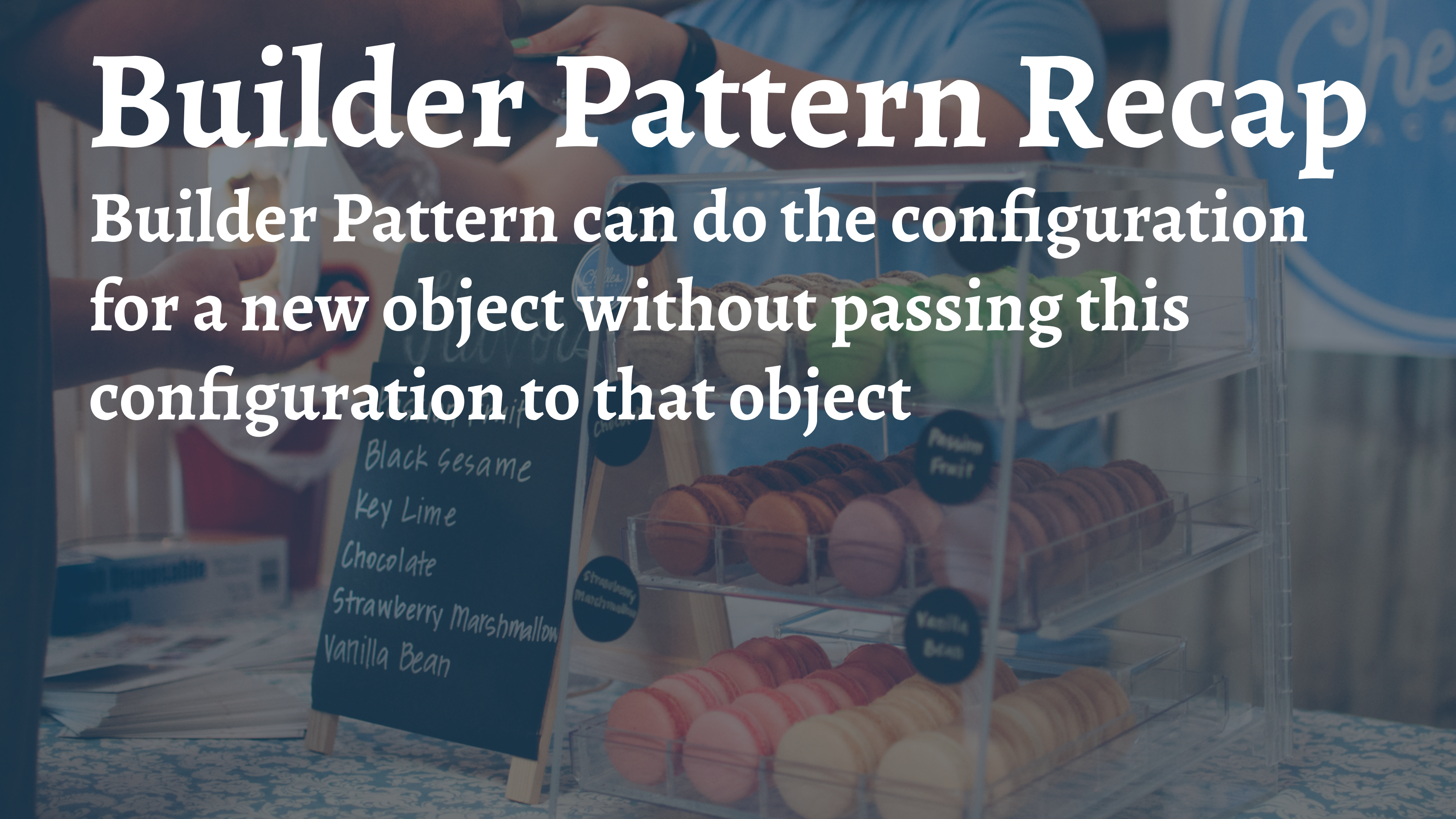
            return $debugbar;
        });
    }
}
```

Factory Pattern Recap

Factory translates interface to an instance

Builder Pattern Recap

Builder Pattern can do the configuration for a new object without passing this configuration to that object

A background image of a bakery display. A three-tiered clear acrylic stand holds rows of macarons in various colors. To the left, a small black chalkboard on wooden legs lists flavors: Black Sesame, Key Lime, Chocolate, Strawberry Marshmallow, and Vanilla Bean. Circular labels are placed near the macaron trays, including 'Passion Fruit', 'Vanilla Bean', and 'Strawberry Marshmallow'. In the background, a person's hands are visible holding a macaron, and a blue sign with 'The' is partially seen.

Black Sesame
Key Lime
Chocolate
Strawberry Marshmallow
Vanilla Bean

Strategy Pattern Recap

Strategy pattern provides one place where the correct strategy (algorithm) is called



Provider Pattern Recap

Provider Pattern let you extend the framework by registering new classes in the container

Wrapping up

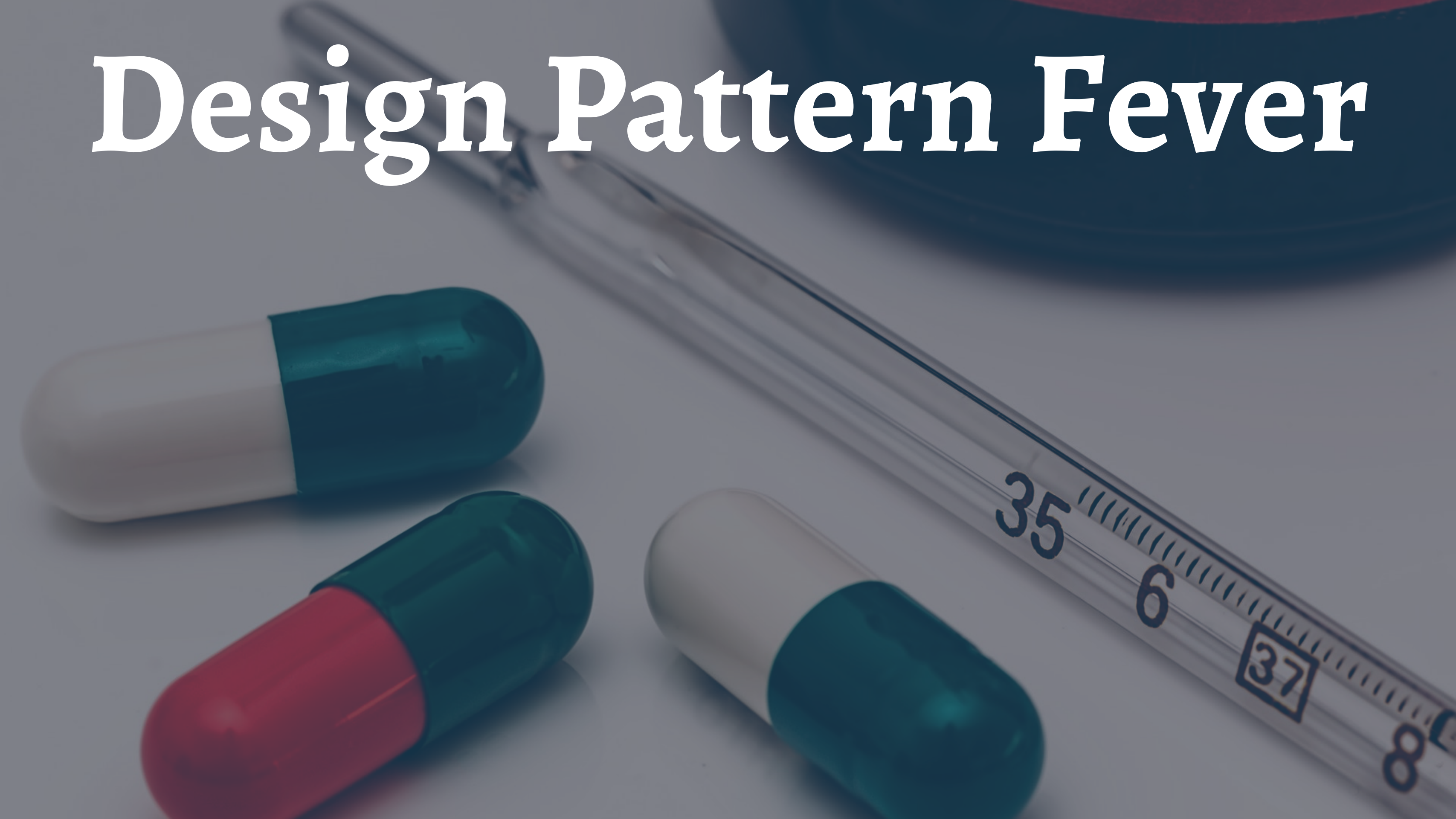


Wrapping up

```
use App\Dominos\PizzaManager;
```

```
class PizzaServiceProvider extends ServiceProvider
{
    public function register()
    {
        $this->app->bind('pizza-manager', function ($app) {
            return new PizzaManager();
        });
    }
}
```

Design Pattern Fever



Want to learn more?

More Design Patterns: <https://goo.gl/sHT3xK>

Thank you

Bobby Bouwmann
@bobbybouwmann

Thank you

Bobby Bouwmann
@bobbybouwmann