Are service decorators an anti-pattern in Shopware?

Shopware United Connect 2025

YIREO

Jisse Reitsma

- Founder of Yireo
- Main developer of Yireo Loki Checkout for Magento
- Trainer of Magento & Shopware developers
- Organizer of MageUnconference NL
- Former:
  - creator of Shopware official videos
  - board member of Mage-OS Nederland
  - organizer of MageTestFest, Reacticon
  - Magento Master

YIREO

Me and Shopware

- Shopware developer training
- Symfony developer of internal Yireo applications
- Experimental extension developer

YIREO

What is a Service Decorator?

YIREO

YIREO

What is a service?

- PHP class (so: an object)
- Registered as service in the Symfony service container
- Ideally a class with a single responsibility
- Ideally stateless objects
- Ideally driven by a PHP interface

YIREO

PHP interface

File `src/Components/FoobarInterface.php`:

```
namespace Swag\Example\Components;

interface FoobarInterface
{
    public function doSomething();
}
```


YIREO

PHP class

File `src/Components/Foobar.php`:

```
namespace Swag\Example\Components;

class Foobar implements FoobarInterface
{
    public function doSomething() {}
}
```


YIREO

Service declaration

File `src/Resources/config/services.xml`:

```
<?xml version="1.0" ?>
<container . . .>
    <services>
        <service
           id="Swag\Example\Components\FoobarInterface"
           class="Swag\Example\Components\Foobar" />

        <service
           id="foobar"
           class="Swag\Example\Components\FoobarInterface" />
    </services>
</container>
```


YIREO

Or write the service declaration in …

- PHP configuration
- YAML configuration
- PHP Attribute + autowiring + autoconfiguring

YIREO

YIREO

Changing services

- Override configurations
  - Constructor arguments
  - Factory patterns
  - Tagging
  - Setting as public or not
  - Lazy loading
  - Autowiring

YIREO

Changing services

- Override configurations
  - Via configuration files (like `services.xml`)
  - Via compiler passes

YIREO

Changing services

- Override configurations
- Rewrite services
  - Via aliases

YIREO

Concrete alias example: Logger

- `psr/log` standard is implemented by Monolog
- Service ID `Psr\Log\LoggerInterface` has an alias `logger`
- Service ID `logger` has an alias `monolog.logger`
- Service ID `monolog.logger` has parent `monolog.logger_prototype`
- Service ID `monolog.logger_prototype` has class `Monolog\Logger`

YIREO

Changing services

- Override configurations
- Rewrite services
  - Via aliases
  - Via decoration

YIREO

YIREO

Decorator design pattern

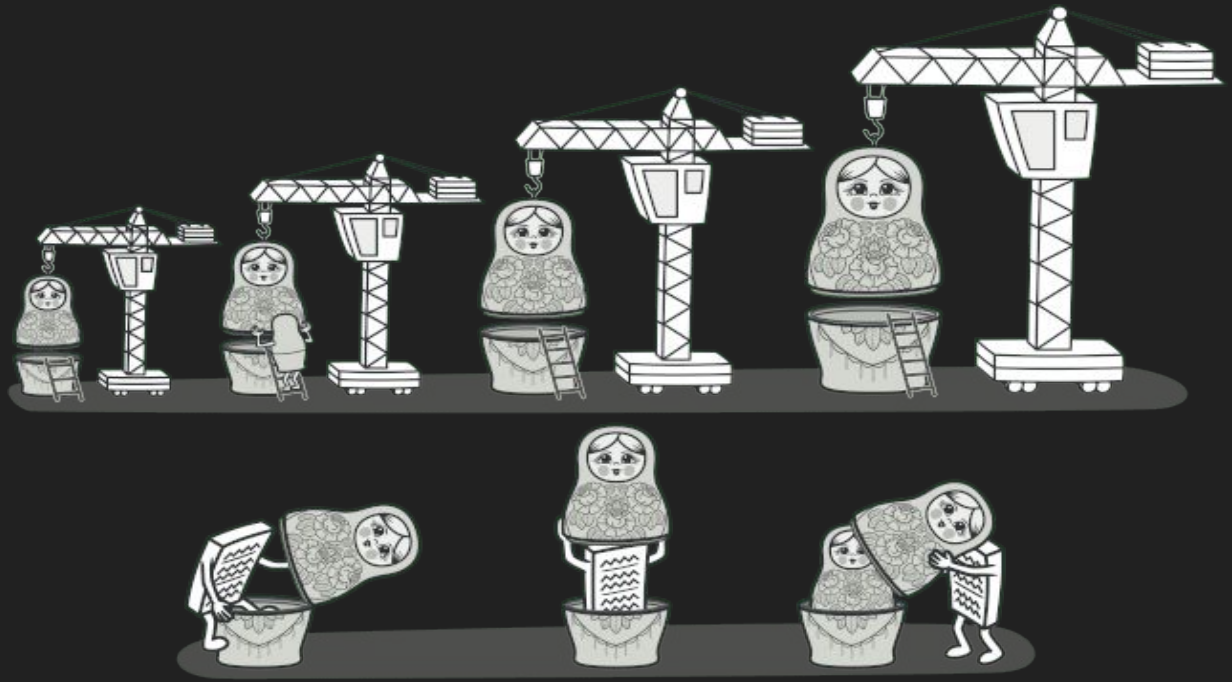


Image taken from
refactoring.guru

YIREO

Service decoration declaration

File `src/Resources/config/services.xml`:

```
<services>
    <service
        class="Swag\Example\Components\FoobarDecorator"
        decorates="Swag\Example\Components\FoobarInterface">
        <argument type=”service” id=”.inner”/>
    </service>
</services>
```


YIREO

Decorator class (implementing the interface)

File `src/Components/FoobarDecorator.php`:

```
namespace Swag\Example\Components;

class FoobarDecorator implements FoobarInterface
{
    public function __construct(
        private FoobarInterface $inner
    ){}

    public function doSomething() {
        return $this->inner->doSomething();
    }
}
```


YIREO

Decorator class (extending the original class)

File `src/Components/FoobarDecorator.php`:

```
namespace Swag\Example\Components;

class FoobarDecorator extends Foobar
{
    public function __construct(
        private Foobar $inner
    ){}

    public function doSomething() {
        return $this->inner->doSomething();
    }
}
```


YIREO

Benefits of service decorators vs aliases

- Multiple decorators can decorate the same service
- Versus alias rewrites that only work once

YIREO

With a service decorator …

- You can do something before and/or after the original methods
- Or you can skip the original methods
- …

YIREO

With a service decorator ...

- You can do something before, after or around the original methods
- Or you can override the original methods
- ... for any service in the container!

YIREO


Let’s decorate
Shopware United Connect 2025

Example: Change the logo

- In Twig, the logo is printed with `theme_config('sw-logo-desktop')`
- Calls `Shopware\Storefront\Framework\Twig\Extension\ConfigExtension`
- Calls `Shopware\Storefront\Framework\Twig\TemplateConfigAccessor`
- Which we can decorate

YIREO

Service decoration declaration

File `src/Resources/config/services.xml`:

```
<services>
    <service
        class="Swag\Example\Components\LogoDecorator"
        decorates="Shopware\Storefront\Theme\ThemeConfigValueAccessor">
        <argument type=”service” id=”.inner”/>
    </service>
</services>
```


YIREO

Decorator class

File `src/Components/LogoDecorator.php`:

```
class FoobarDecorator extends Foobar
{
    public function __construct(
        private ThemeConfigValueAccessor $inner
    ){}

    public function get(string $key, SalesChannelContext $context,
?string $themeId) {
        return $this->inner->get($key, $context, $themeId);
    }
}
```


YIREO

Decorator class

File `src/Components/LogoDecorator.php`:

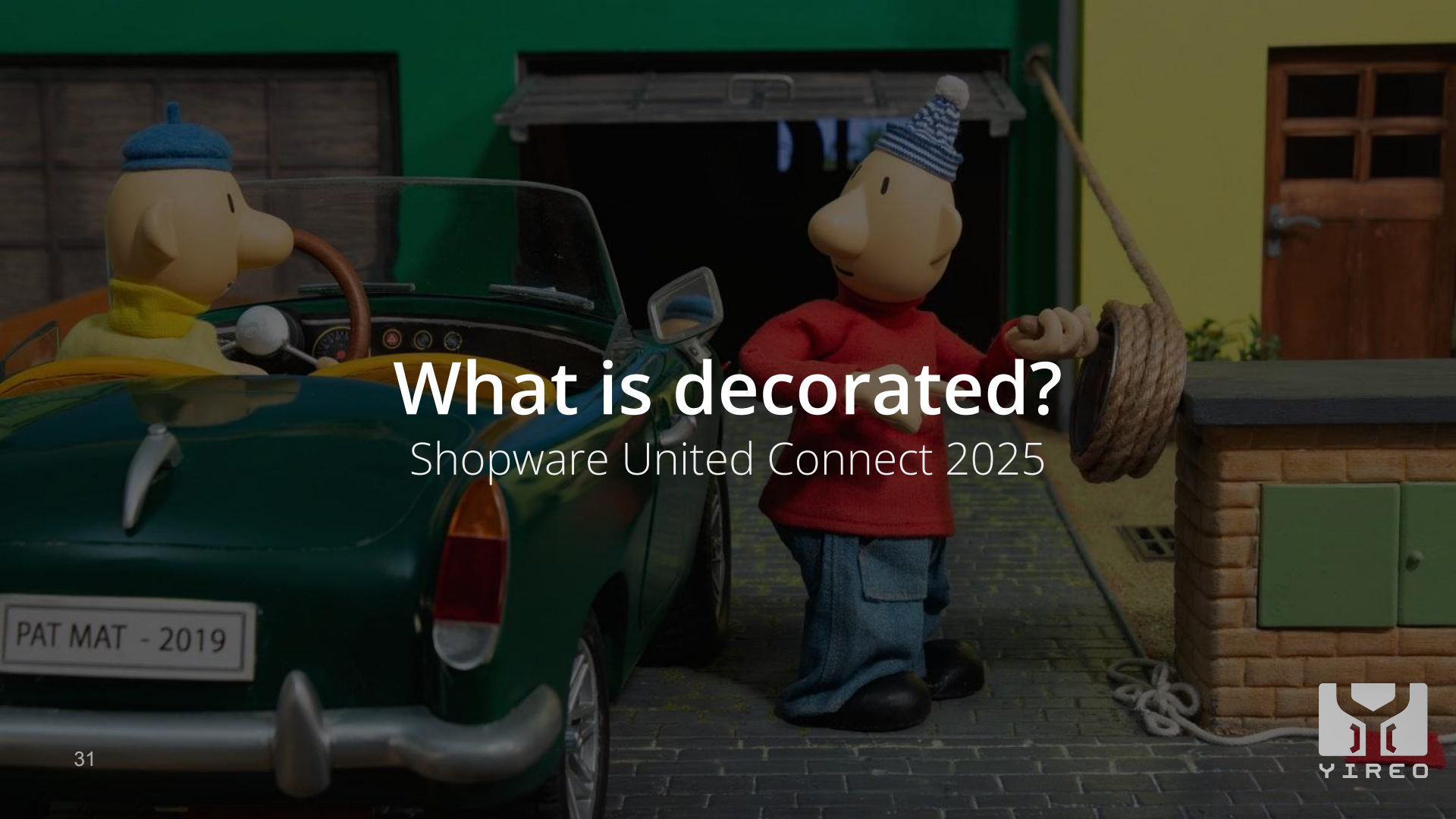
```
    public function get(string $key, SalesChannelContext $context,
?string $themeId) {
        if ($key === 'sw-logo-desktop') {
            return 'https://www.yireo.com/images/logos/yireo.svg';
        }

        return $this->inner->get($key, $context, $themeId);
    }
```


YIREO

The point?

- Yes, you can decorate anything
- But you should have just changed the logo via Shopware Administration

YIREO



What is decorated?

Shopware United Connect 2025

YIREO

How many decorators are there?

- Get tagged iterator `container.decorator`
- Use https://github.com/yireo-shopware6/shopware6-list-decorators
- `bin/console debug:decorators`

Answer: 62

YIREO

Why decorators in the core?

- Caching
- Sorting of payment / shipping methods
- Switch between ElasticSearch and MySQL search

YIREO

Why decorators in 3rd party extensions?

- Because it is handy?
- Because the core lacks a certain extensibility?

YIREO

How I have used decorators in the past?

- Replacing original service with something else
- Add hacks because the original architecture didn’t allow for modification
- Huge changes of logic in checkout, product loader, CMS logic, etc

YIREO

Are decorators an anti-pattern?

Shopware United Connect 2025

YIREO

Some anti-patterns

- Spaghetti Code
- God Object (like the Singleton)
- Dead Code
- Golden Hammer
- …

YIREO

Decorators are often used as Golden Hammer

- Forget about a screwdriver & a wrench: Just use the hammer.
- You can apply it to any service
- You are not limited by the Shopware core
- Almost every scenario is possible by adding a decorator

YIREO

Conclusion

- First try to extend the system as it was intended to be extended
  - Service configuration
  - Proper constructor DI
  - Event Listeners & Event Subscribers
  - Tags & tag iterators

YIREO

Conclusion

- First try to extend the system as it was intended to be extended
- And perhaps discuss this with the Shopware team
- Or create a Pull Request to refactor
- Only use service decorators as a final resort
- Document your decorator purpose with care

YIREO

Thanks

Shopware United Connect 2025

YIREO