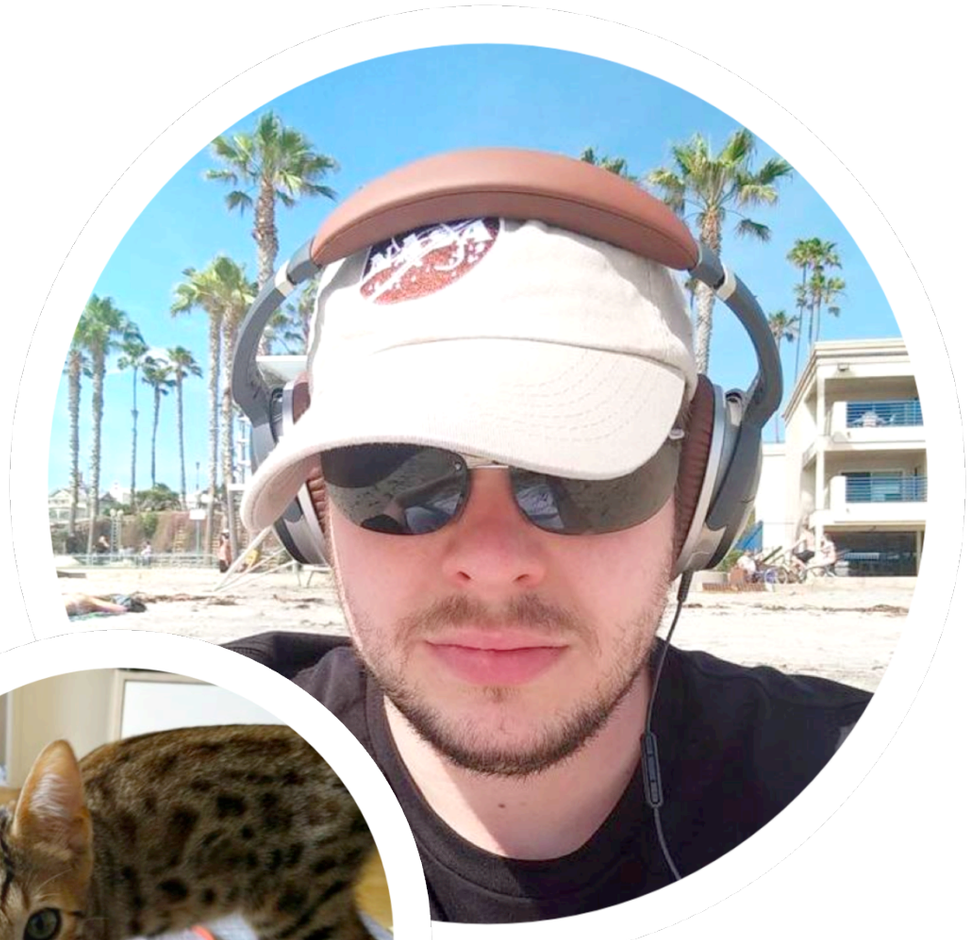
Solving real world problems with design patterns!

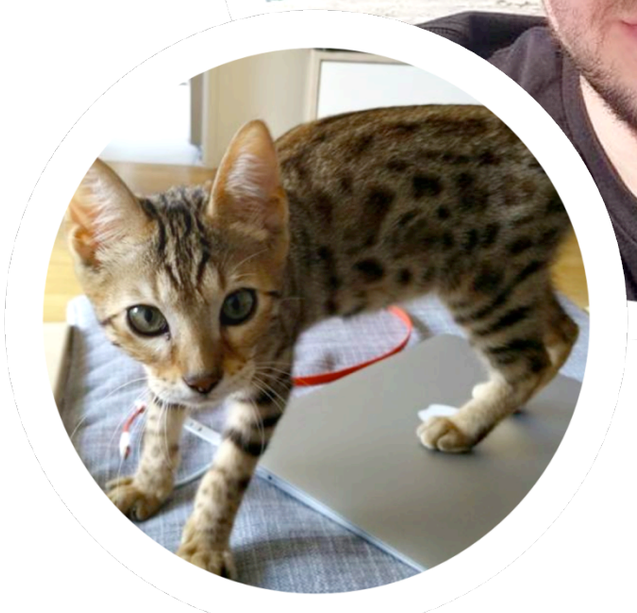
https://www.flickr.com/photos/moritzlino

SensioLabs

Confoo 2017 – Montréal - Hugo Hamon

Hugo Hamon





Software Architect
SensioLabs

Book author
Conferences speaker
Symfony contributor
Bengal cat owner

@hhamon / @catlannister

More PHP Presentations on Speakerdeck

https://speakerdeck.com/hhamon

Implementing Design Patterns with PHP

Mar 2, 2017 by Hugo Hamon

Implementing Design Patterns with PHP

Sep 28, 2016 by Hugo Hamon

Persister des Value Objects avec Doctrine

Apr 8, 2016 by Hugo Hamon

Practical Approach of Using Design Patterns

Data (De)Serialization 101

Sep 3, 2015 by Hugo Hamon

Design Patterns, the practical approach in PHP

Hugo Hamon

Hugo Hamon is a PHP and Symfony fan who works with PHP since 2003. After five years of professional PHP web development in web agencies for famous french customers, he now works as a consultant and head of training at SensioLabs. On his free time, Hugo contributes to the Symfony2 project (source code and documentation) and gets involved in the french PHP association as an organization member. Hugo also wrote and contributed to french and english books related to PHP and the Symfony framework.

Introduction

What are design patterns?

In software design, a **design pattern** is an abstract **generic solution** to **solve** a particular **redundant problem**.

Design Patterns Classification

23 « **Gang of Four** » Design Patterns

- **Creational**
- **Structural**
- **Behavioral**

Benefits of Design Patterns

- Communication & vocabulary
- Testability
- Maintainance
- Extensibility
- Loose coupling

Downsides of Design Patterns

- Hard to teach and learn
- Hard to know when to apply
- Require good knowledge of OOP
- **Not always the holly grail!**

Design Patterns encourage SOLID code

SRP / **S**ingle Responsability Principle

OCP / **O**pen / Closed Principle

LSP / **L**iskov Substitution Principle

ISP / **I**nterface Segregation Principle

DIP / **D**ependency Inversion Principle

Creational Patterns

Creational design patterns encapsulate and isolate the algorithms to create and initialize objects.

Abstract Factory – Builder – Factory Method
Lazy Initialization – Prototype – ~~Singleton~~

Abstract Factory

https://www.flickr.com/photos/cortomaltes/4640745131/

Abstract Factory provides an interface for creating families of related or dependent objects without specifying their concrete classes.

– GoF

Towards context independency

2012 – Symfony 2.3

2015 – Symfony 3.0

2016 - Twig 1.x

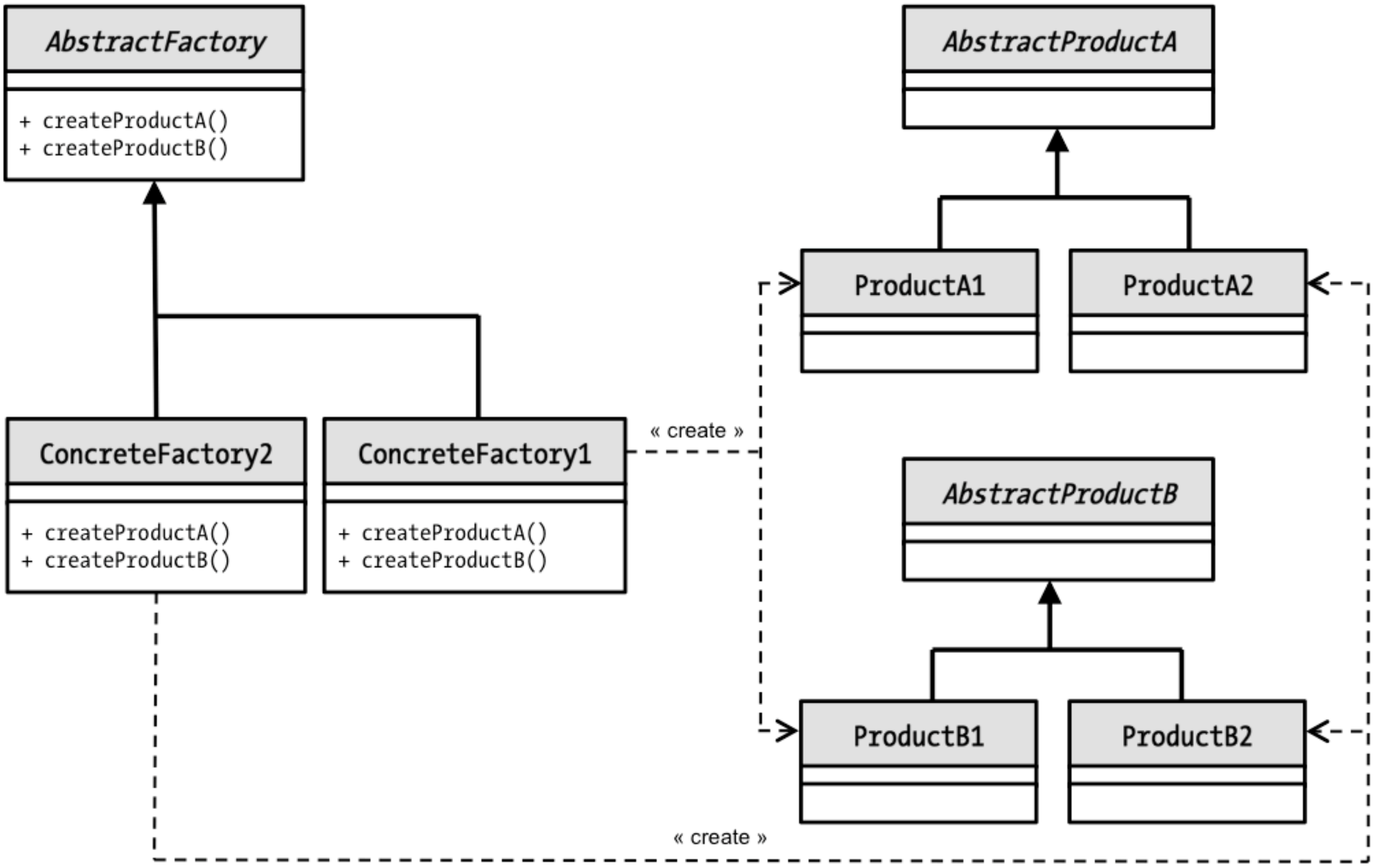
Both assessments have similarities

-	Symfony	Twig
Pricing	**€250** (USA, Europe, UAE, Japan, etc.) **€200** (Brazil, Tunisia, India, etc.)	**€149** (no country restriction)
Eligibility Conditions	Candidate must be at least 18 y.o Candidate must not be Expert Certified Up to 2 sessions max per civil year No active exam registration 1 week blank period between 2 exams	Candidate must be at least 18 y.o Candidate must not be Expert Certified No active exam registration 1 month blank period between 2 exams
Levels	**Expert** Symfony Developer **Advanced** Symfony Developer	**Expert** Twig WebDesigner

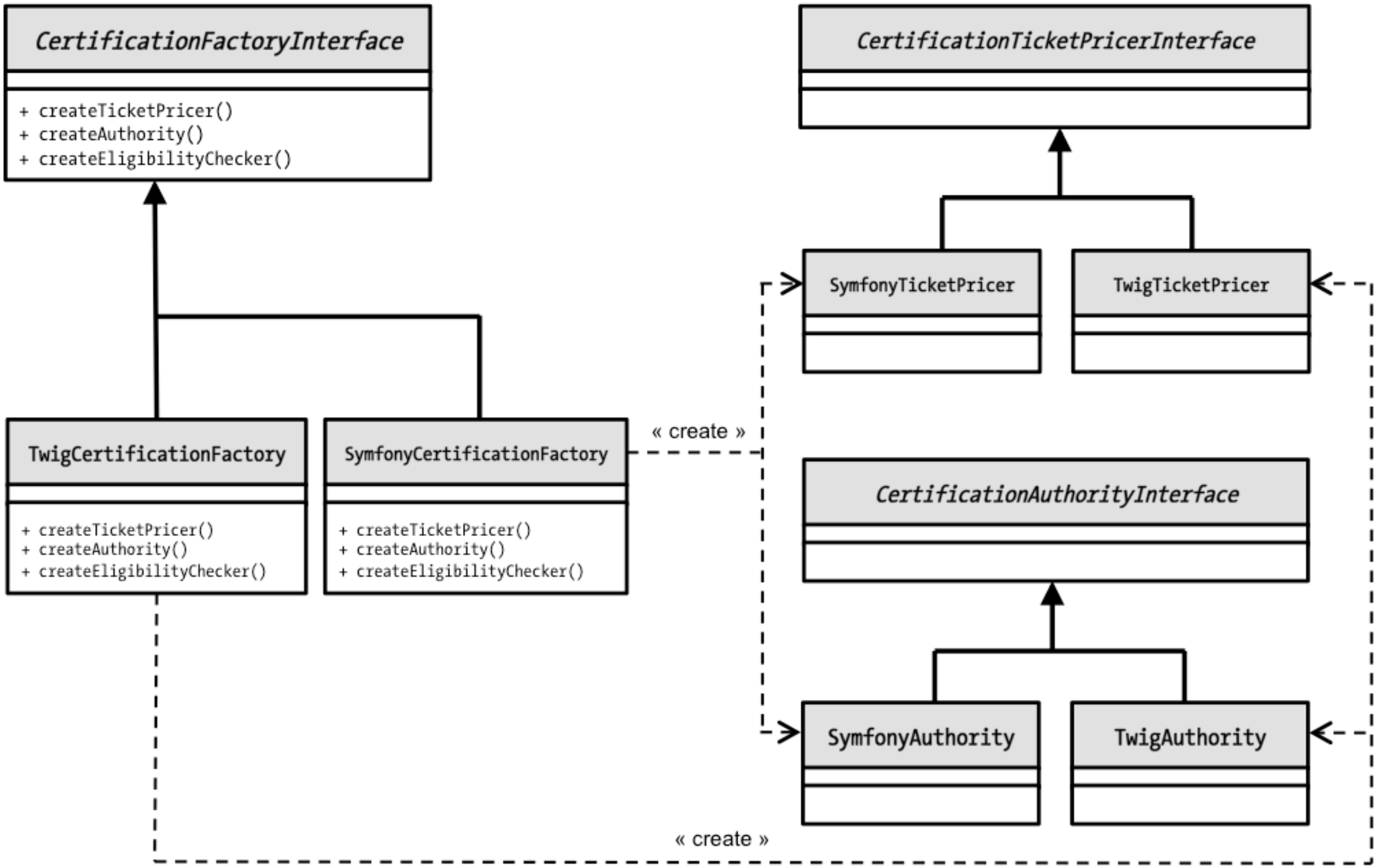
Refactoring context dependant code

```php
class PlaceOrderCommandHandler
{
    private $symfonyPricer;
    private $twigPricer;

    // ...

    public function handle(OrderCommand $command)
    {
        $order = new ConnectOrder();
        $order->setQuantity($command->getQuantity());
        $order->setUnitPrice($this->getUnitPrice($command)->getConvertedAmount());
        // ...
    }

    private function getUnitPrice(OrderCommand $command): Money
    {
        $country = $command->getCountry();
        switch ($command->getExamSeriesType()) {
            case 'TWXCE':
                return $this->twigPricer->getUnitPrice($country);
            case 'SFXCE':
                return $this->symfonyPricer->getUnitPrice($country);
        }

        throw new UnsupportedAssessmentException($command->getExamSeriesType());
    }
}
```

Abstract Factory - UML Diagram



Abstract Factory for Assessments Management



Applying the Abstract Factory Pattern

```php
namespace SensioLabs\Certification;

interface CertificationFactoryInterface
{
    public function createEligibilityChecker()
        : CertificationEligibilityCheckerInterface;

    public function createTicketPricer()
        : CertificationTicketPricerInterface;

    public function createAuthority()
        : CertificationAuthorityInterface;
}
```

The ticket pricer

```php
// Symfony certifications
$regular = Money::fromString('250', 'EUR');
$discount = Money::fromString('200', 'EUR');

$pricer = new SymfonyCertificationTicketPricer($regular, $discount);
$price = $pricer->getUnitPrice('FR');
$price = $pricer->getUnitPrice('TN');

// Twig certification
$regular = Money::fromString('149', 'EUR');

$pricer = new TwigCertificationTicketPricer($regular);
$price = $pricer->getUnitPrice('FR');
$price = $pricer->getUnitPrice('TN');
```

The generic TicketPricerInterface

```php
namespace SensioLabs\Certification;

use SebastianBergmann\Money\Money;

interface CertificationTicketPricerInterface
{
    /**
     * Returns unit retail price of a ticket.
     */
    public function getUnitPrice(string $country): Money;

    /**
     * Returns the total price for the given quantity.
     */
    public function getTotalPrice(string $country, int $quantity): Money;
}
```

The abstract TicketPricer implementation

```php
namespace SensioLabs\Certification;

use SebastianBergmann\Money\Money;

abstract class AbstractTicketPricer implements CertificationTicketPricerInterface
{
    /**
     * Returns the total price for the given quantity.
     */
    public function getTotalPrice(string $country, int $quantity): Money
    {
        if ($quantity < 1) {
            throw new \InvalidArgumentException('Invalid quantity.');
        }

        return $this->getUnitPrice($country)->multiply($quantity);
    }
}
```

The Twig Ticket Pricer

```php
namespace SensioLabs\Certification\Twig;

use SebastianBergmann\Money\Money;
use SensioLabs\Certification\AbstractTicketPricer;

class TwigCertificationTicketPricer extends AbstractTicketPricer
{
    private $price;

    public function __construct(Money $price)
    {
        $this->price = $price;
    }

    public function getUnitPrice(string $country): Money
    {
        return $this->price;
    }
}
```

The Symfony Ticket Pricer

```php
class SymfonyCertificationTicketPricer extends AbstractTicketPricer
{
    private $regularPrice;
    private $discountPrice;

    public function __construct(Money $regularPrice, Money $discountPrice)
    {
        $this->regularPrice = $regularPrice;
        $this->discountPrice = $discountPrice;
    }

    public function getUnitPrice(string $country): Money
    {
        if (in_array($country, ['FR', 'US', 'DE', 'IT', 'CH', 'BE', /* … */])) {
            return $this->regularPrice;
        }

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

The certification authority

```
try {

    $assessment = AssessmentResult::fromCSV('SLO38744,SF3CE,35');
    $authority = new SymfonyCertificationAuthority(20, 10);
    $symfonyLevel = $authority->getCandidateLevel($assessment);

    $assessment = AssessmentResult::fromCSV('SLO38744,TW1CE,17');
    $authority = new TwigCertificationAuthority(15);
    $twigLevel = $authority->getCandidateLevel($assessment);

} catch (CandidateNotCertifiedException $e) {
    // ...
} catch (UnsupportedAssessmentException $e) {
    // ...
} catch (\Exception $e) {
    // ...
}
```

The certification authority interface

```php
namespace SensioLabs\Certification;

use SensioLabs\Certification\Entity\AssessmentResult;
use SensioLabs\Certification\Exception\CandidateNotCertifiedException;
use SensioLabs\Certification\Exception\UnsupportedAssessmentException;

interface CertificationAuthorityInterface
{
    /**
     * Returns the candidate's certification level
     * based the candidate's exam result.
     *
     * @throws CandidateNotCertifiedException
     * @throws UnsupportedAssessmentException
     */
    public function getCandidateLevel(AssessmentResult $result): string;
}
```

The Twig certification authority

```php
namespace SensioLabs\Certification\Twig;

use SensioLabs\Certification\CertificationAuthorityInterface;
use SensioLabs\Certification\Entity\AssessmentResult;
use SensioLabs\Certification\Exception\CandidateNotCertifiedException;
use SensioLabs\Certification\Exception\UnsupportedAssessmentException;

class TwigCertificationAuthority implements CertificationAuthorityInterface
{
    private $passingScore;

    public function __construct(int $passingScore)
    {
        $this->passingScore = $passingScore;
    }

    public function getCandidateLevel(AssessmentResult $result): string
    {
        if ('TW1CE' !== $assessmentID = $result->getAssessmentID()) {
            throw new UnsupportedAssessmentException($assessmentID);
        }

        if ($result->getScore() < $this->passingScore) {
            throw new CandidateNotCertifiedException($result->getCandidateID(), $assessmentID);
        }

        return 'expert';
    }
}
```

The Symfony certification authority

```php
namespace SensioLabs\Certification\Symfony;

use SensioLabs\Certification\CertificationAuthorityInterface;
use SensioLabs\Certification\Entity\AssessmentResult;
use SensioLabs\Certification\Exception\CandidateNotCertifiedException;
use SensioLabs\Certification\Exception\UnsupportedAssessmentException;

class SymfonyCertificationAuthority implements CertificationAuthorityInterface
{
    private $expertLevelPassingScore;
    private $advancedLevelPassingScore;

    public function __construct(int $expertScore, int $advancedScore)
    {
        $this->expertLevelPassingScore = $expertScore;
        $this->advancedLevelPassingScore = $advancedScore;
    }

    // ...
}
```

The Symfony certification authority

```php
class SymfonyCertificationAuthority implements CertificationAuthorityInterface
{
    public function getCandidateLevel(AssessmentResult $result): string
    {
        $assessmentID = $result->getAssessmentID();
        if (!in_array($assessmentID, ['SF2CE', 'SF3CE'])) {
            throw new UnsupportedAssessmentException($assessmentID);
        }

        $score = $result->getScore();
        if ($score >= $this->expertLevelPassingScore) {
            return 'expert';
        }

        if ($score >= $this->advancedLevelPassingScore) {
            return 'advanced';
        }

        throw new CandidateNotCertifiedException($result->getCandidateID(), $assessmentID);
    }

    // ...
}
```

The Symfony certification factory

```php
namespace SensioLabs\Certification\Symfony;

use SebastianBergmann\Money\Money;
use SensioLabs\Certification\CertificationEligibilityCheckerInterface;
use SensioLabs\Certification\CertificationAuthorityInterface;
use SensioLabs\Certification\CertificationFactoryInterface;
use SensioLabs\Certification\Repository\AssessmentRepositoryInterface;
use SensioLabs\Certification\Repository\CandidateRepositoryInterface;

class SymfonyCertificationFactory implements CertificationFactoryInterface
{
    private $candidateRepository;
    private $assessmentRepository;

    public function __construct(
        CandidateRepositoryInterface $candidateRepository,
        AssessmentRepositoryInterface $assessmentRepository
    )
    {
        $this->candidateRepository = $candidateRepository;
        $this->assessmentRepository = $assessmentRepository;
    }

    // ...
}
```

The Symfony certification factory

```php
class SymfonyCertificationFactory implements CertificationFactoryInterface
{
    public function createEligibilityChecker(): CertificationEligibilityCheckerInterface
    {
        return new SymfonyCertificationEligibilityChecker(
            $this->candidateRepository,
            $this->assessmentRepository
        );
    }

    public function createTicketPricer(): CertificationTicketPricerInterface
    {
        return new SymfonyCertificationTicketPricer(
            Money::fromString('250', 'EUR'),
            Money::fromString('200', 'EUR')
        );
    }

    public function createAuthority(): CertificationAuthorityInterface
    {
        return new SymfonyCertificationAuthority(20, 10);
    }

    // ...
}
```

The Twig certification factory

```php
namespace SensioLabs\Certification\Twig;

use SebastianBergmann\Money\Money;
use SensioLabs\Certification\CertificationEligibilityCheckerInterface;
use SensioLabs\Certification\CertificationAuthorityInterface;
use SensioLabs\Certification\CertificationFactoryInterface;
use SensioLabs\Certification\Repository\AssessmentRepositoryInterface;
use SensioLabs\Certification\Repository\CandidateRepositoryInterface;

class TwigCertificationFactory implements CertificationFactoryInterface
{
    private $candidateRepository;
    private $assessmentRepository;

    public function __construct(
        CandidateRepositoryInterface $candidateRepository,
        AssessmentRepositoryInterface $assessmentRepository
    )
    {
        $this->candidateRepository = $candidateRepository;
        $this->assessmentRepository = $assessmentRepository;
    }

    // ...
}
```

The Twig certification factory

```php
class TwigCertificationFactory implements CertificationFactoryInterface
{
    public function createEligibilityChecker(): CertificationEligibilityCheckerInterface
    {
        return new TwigCertificationEligibilityChecker(
            $this->candidateRepository,
            $this->assessmentRepository
        );
    }

    public function createTicketPricer(): CertificationTicketPricerInterface
    {
        return new TwigCertificationTicketPricer(Money::fromString('149', 'EUR'));
    }

    public function createAuthority(): CertificationAuthorityInterface
    {
        return new TwigCertificationAuthority(20);
    }
    // ...
}
```

Certification Factories Usage

```php
$assessmentFamily = 'SFXCE';

// Choose the right factory
switch ($assessmentFamily) {
    case 'TWXCE':
        $factory = new TwigCertificationFactory(...);
        break;
    case 'SFXCE':
        $factory = new SymfonyCertificationFactory(...);
        break;
    default:
        throw new UnsupportedAssessmentFamilyException($assessmentFamily);
}

// Request and use the factory services
$pricer = $factory->createTicketPricer();
$checker = $factory->createEligibilityChecker();
$authority = $factory->createAuthority();
```

Producing the factories from a factory

```php
namespace SensioLabs\Certification;

use SensioLabs\Certification\Exception\UnsupportedAssessmentException;
use SensioLabs\Certification\Repository\AssessmentRepositoryInterface;
use SensioLabs\Certification\Repository\CandidateRepositoryInterface;
use SensioLabs\Certification\Symfony\SymfonyCertificationFactory;
use SensioLabs\Certification\Twig\TwigCertificationFactory;

class CertificationFactoryFactory implements CertificationFactoryFactoryInterface
{
    private $factories;
    private $candidateRepository;
    private $assessmentRepository;

    public function __construct(
        CandidateRepositoryInterface $candidateRepository,
        AssessmentRepositoryInterface $assessmentRepository
    )
    {
        $this->factories = [];
        $this->candidateRepository = $candidateRepository;
        $this->assessmentRepository = $assessmentRepository;
    }

    // ...
}
```

Producing the factories from a factory

```php
class CertificationFactoryFactory implements CertificationFactoryFactoryInterface
{
    // ...

    public function getFactory(string $assessment): CertificationFactoryInterface
    {
        if (isset($this->factories[$assessment])) {
            return $this->factories[$assessment];
        }

        if ('TW1CE' === $assessment) {
            return $this->createTwigCertificationFactory($assessment);
        }

        if (in_array($assessment, ['SF2CE', 'SF3CE'], true)) {
            return $this->createSymfonyCertificationFactory($assessment);
        }

        throw new UnsupportedAssessmentException($assessment);
    }
}
```

Producing the factories from a factory

```php
class CertificationFactoryFactory implements CertificationFactoryFactoryInterface
{
    // ...
    private function createTwigCertificationFactory(string $assessment)
    : CertificationFactoryInterface
    {
        $this->factories[$assessment] = $factory = new TwigCertificationFactory(
            $this->candidateRepository,
            $this->assessmentRepository
        );

        return $factory;
    }

    private function createSymfonyCertificationFactory(string $assessment)
    : CertificationFactoryInterface
    {
        $this->factories[$assessment] = $factory = new SymfonyCertificationFactory(
            $this->candidateRepository,
            $this->assessmentRepository
        );

        return $factory;
    }
}
```

Producing the factories from a factory

```php
class CertificationFactoryFactory implements CertificationFactoryFactoryInterface
{
    // ...
    public function createCertificationTicketPricer(string $assessment)
    : CertificationTicketPricerInterface
    {
        return $this->getFactory($assessment)->createTicketPricer();
    }

    public function createCertificationAuthority(string $assessment)
    : CertificationAuthorityInterface
    {
        return $this->getFactory($assessment)->createCertificationAuthority();
    }

    public function createCertificationEligibilityChecker(string $assessment)
    : CertificationEligibilityCheckerInterface
    {
        return $this->getFactory($assessment)->createEligibilityChecker();
    }
}
```

Refactoring context dependant code

```php
class PlaceOrderCommandHandler
{
    private $factory;

    function __construct(CertificationFactoryFactoryInterface $factory)
    {
        $this->factory = $factory;
    }

    // ...

    private function getUnitPrice(OrderCommand $command)
    {
        return $this
            ->factory
            ->getCertificationTicketPricer($command->getExamSeriesType())
            ->getUnitPrice($command->getCountry());
    }
}
```

Pros & Cons of Abstract Factories

- 👍 Provide context / platform / system independance,
- 👍 Extreme modularity, reusability and interchangeabilty,
- 👍 Swapping a factory by another is very easy,
- 👍 Objects of the same family are centralized,
- 👍 Very useful for generic systems like frameworks.

- 👎 High level of complexity and hard to implement,
- 👎 Involves lots of classes and interfaces,
- 👎 Requires the requirements to exactly fit the need for an A.F.

Structural Patterns

Structural patterns organize classes and help separating the implementations from their interfaces.

Adapter – Bridge – **Composite** – Decorator
Facade – Flyweight – Proxy

The Adapter design pattern helps two incompatible interfaces work together without changing them.

An example of incompatible interfaces

Adapting two incompatible interfaces

Real world example – Template engines

```php
interface EngineInterface
{
    public function exists(string $template): bool;

    public function supports(string $template): bool;

    public function loadTemplate(
      string $template,
      array $vars = []
    ): TemplateInterface;

    public function evaluate(
        string $template,
        array $vars = []
    ): string;
}
```

The PHP template engine

```
class PhpEngine implements EngineInterface
{
    // ... implement methods of the interface.
}
```

The **PhpEngine** class represents a template engine that enables to render raw PHP templates.

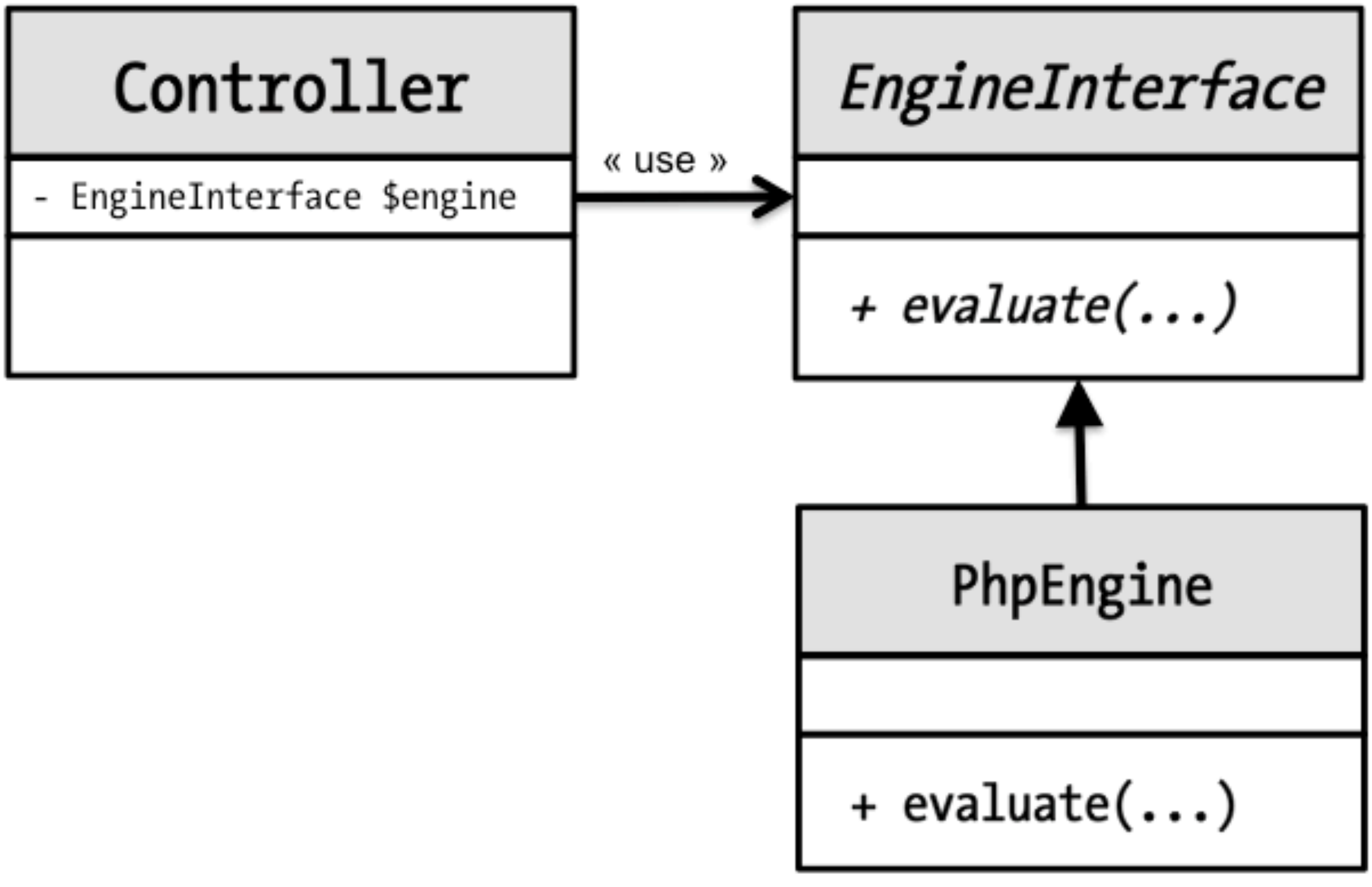
Using the PHP template engine

```
class BlogController
{
    private $templating;

    public function __construct(..., EngineInterface $templating)
    {
        $this->templating = $templating;
    }

    public function indexAction(Request $request): Response
    {
        $posts = ... ;

        return $this->render('blog/index.tpl', [ 'posts' => $posts ]);
    }
}
```

Using the PHP template engine

```
class BlogController
{
    // ...

    protected function render(
        string $view,
        array $context = []
    ): Response
    {
        return new Response(
            $this->templating->evaluate($view, $context)
        );
    }
}
```

Uses cases for adapters

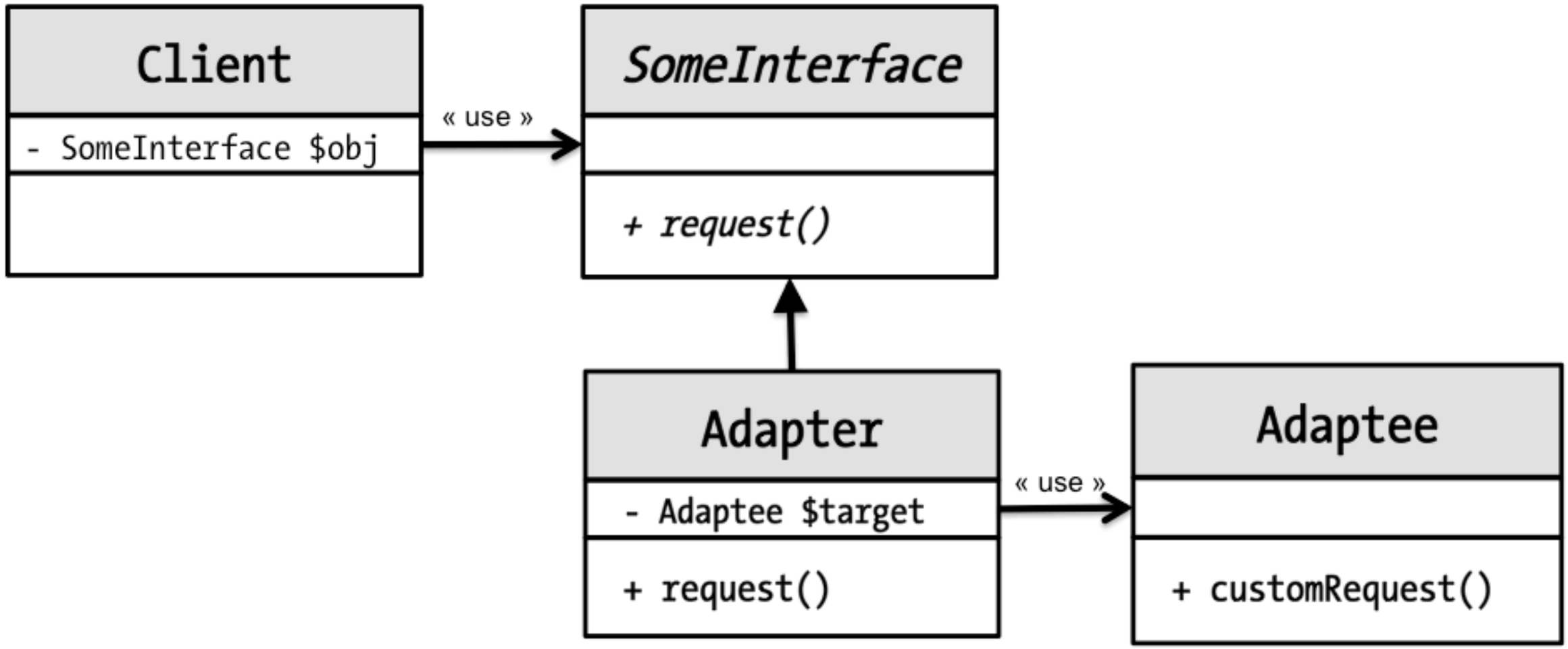


Several years later… a new challenge!

Many years have passed and the IT team decides to modernize the application using a new template engine like **Twig** or **Plates**. However, both third party libraries don't implement the initial homemade **EngineInterface** interface.

Challenge: how to use any of these two libraries without changing any single line of their code or the actual application code?

The Adapter pattern – UML Diagram



The Adapter pattern - Usage

```
$someDependency = new SomeDependency();
$adaptee = new Adaptee($someDependency);
$adapter = new Adapter($adaptee);

$client = new Client($adapter);
$client->doSomething();
```

This client code receives an Adapter dependency. The latter translates the public incompatible API of the Adaptee object into a public compatible API that is expected by the client code.

The Client code

```
class Client
{
    private $target;

    public function __construct(TargetInterface $target)
    {
        $this->target = $target;
    }

    public function doSomething()
    {
        $request = $this->target->request();

        // ... do other things
    }
}
```

The Object to Adapt

```
class Adaptee
{
    public function customRequest()
    {
        return 'custom request';
    }
}
```

The Adapter to class

```
class Adapter implements TargetInterface
{
      private $adaptee;

      public function __construct(Adaptee $adaptee)
      {
            $this->adaptee = $adaptee;
       }

     public function request()
     {
          $request = $this->adaptee->customRequest();

          return strtolower($request);
     }
}
```

Uses cases for adapters

- Modernizing a legacy code,
- Normalizing a public API,
- Exposing / consuming a web service,
- Maintaining a backward compatibility layer,
- Replacing an existing library by a third party one,
- …

The actual PHP template engine

```
class PhpEngine implements EngineInterface
{
    private $helpers;
    private $directory;

    public function __construct($directory)
    {
        $this->addHelper(new EscapeHelper());
        $this->directory = $directory;
    }

    public function addHelper(HelperInterface $helper)
    {
        $this->helpers[$helper->getName()] = $helper;
    }
}
```

The actual PHP template engine

```
class PhpEngine implements EngineInterface
{
    // …

    private function getHelper($name)
    {
        if (!isset($this->helpers[$name])) {
            throw new UnsupportedHelperException($name, array_keys(
                $this->helpers
            ));
        }

        return $this->helpers[$name];
    }
}
```

The actual PHP template engine

```php
class PhpEngine implements EngineInterface
{
    // ...

    public function escape(string $content): string
    {
        return $this->getHelper('escape')->escape($content);
    }

    public function exists(string $template): bool
    {
        return is_readable($this->getTemplatePath($template));
    }

    private function getTemplatePath(string $template): string
    {
        return realpath($this->directory.'/'.ltrim($template, '/'));
    }
}
```

The actual PHP template engine

```php
class PhpEngine implements EngineInterface
{
    // ...
    public function supports($template): bool
    {
        $ext = pathinfo($template, PATHINFO_EXTENSION)
        $ext = strtolower($ext);

        return in_array($ext, ['php', 'tpl']);
    }
}
```

The actual PHP template engine

```php
class PhpEngine implements EngineInterface, \ArrayAccess
{
    // ...
    public function loadTemplate(string $template, array $vars = []): TemplateInterface
    {
        if (!$this->supports($template)) {
            throw new UnsupportedTemplateException(sprintf(
                'Template %s is not supported by this engine.',
                $template
            ));
        }

        if (!$this->exists($template)) {
            throw new TemplateNotFoundException(sprintf(
                'Template %s cannot be found under %s directory.',
                $template,
                $this->directory
            ));
        }

        return new Template($this->getTemplatePath($template), $vars);
    }
}
```

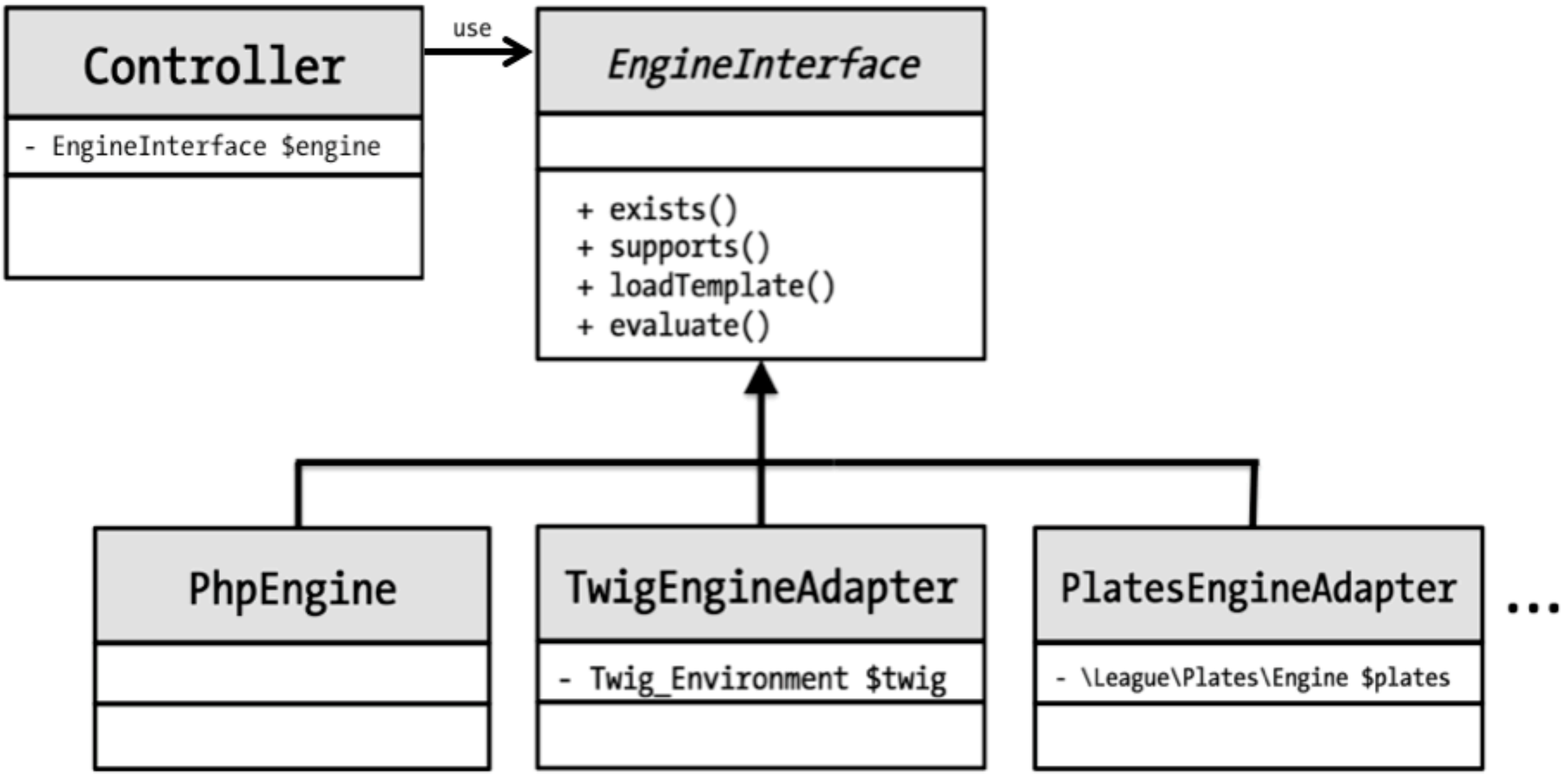
The actual PHP template engine

```php
class PhpEngine implements EngineInterface, \ArrayAccess
{
    // ...
    public function evaluate(string $template, array $vars = []): string
    {
        $reference = $this->loadTemplate($template, $vars);

        if ($reference->has('view')) {
            throw new ReservedKeywordException(sprintf(
                'Template %s cannot have a variable called "view".',
                $template
            ));
        }

        $reference->set('view', $this);

        extract($reference->getVariables());
        ob_start();
        include $reference->getPath();

        return ob_get_clean();
    }
}
```

Supporting a new template engine

Supporting a new template engine like **Twig** must not require to change any line of code in the actual application or in the external libraries source code.

The Adapter pattern to the rescue



Adapting the Twig engine

```
$loader = new \Twig_Loader_Filesystem(__DIR__.'/views');
$twig = new \Twig_Environment($loader);

$engine = new TwigEngineAdapter($twig);
$engine->evaluate('blog.twig', ['posts' => $posts]);
```

The **TwigEngineAdapter** object decorates the **Twig_Environment** object and adapts its specific API for the client code that expects a valid implementation of the **EngineInterface** interface.

The TwigEngineAdapter Class

```php
class TwigEngineAdapter implements EngineInterface
{
    private $twig;

    public function __construct(\Twig_Environment $twig)
    {
        $this->twig = $twig;
    }

    public function supports($template): bool
    {
        $extension = strtolower(pathinfo($template, PATHINFO_EXTENSION));

        return in_array($extension, ['twig', 'tpl']);
    }
}
```

The TwigEngineAdapter Class

```php
class TwigEngineAdapter implements EngineInterface
{
    // ...

    public function exists(string $template): bool
    {
        try {
            $this->twig->loadTemplate($template);
        } catch (\Exception $e) {
            return false;
        }

        return true;
    }

    public function evaluate(string $template, array $vars = []): string
    {
        $reference = $this->loadTemplate($template, $vars);

        return $this->twig->render($template, $reference->getVariables());
    }
}
```

The TwigEngineAdapter Class

```php
class TwigEngineAdapter implements EngineInterface
{
    // ...
    public function loadTemplate(string $template, array $vars = [])
    : TemplateInterface
    {
        if (!$this->supports($template)) {
            throw new UnsupportedTemplateException(sprintf(
                'Template %s is not supported by this engine.',
                $template
            ));
        }

        // ...
    }
}
```

The TwigEngineAdapter Class

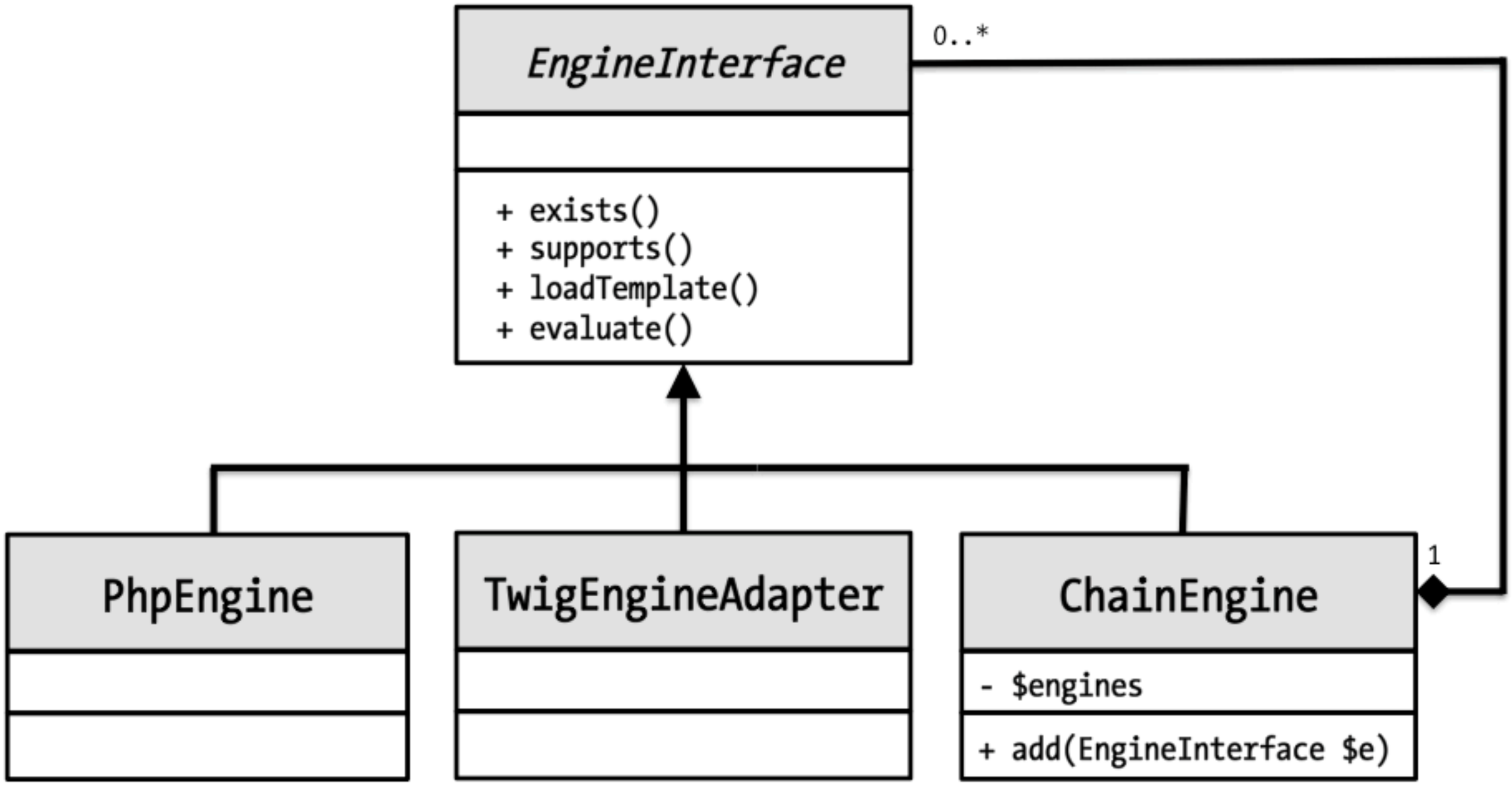
```
class TwigEngineAdapter implements EngineInterface
{
    // ...
    public function loadTemplate(string $template, array $vars = [])
    : TemplateInterface
    {
        // ...
        try {
            $ref = $this->twig->resolveTemplate($template);
        } catch (\Twig_Error_Loader $exception) {
            throw new TemplateNotFoundException('…');
        } catch (\Exception $exception) {
            throw new UnsupportedTemplateException('…');
        }

        $vars = $this->twig->mergeGlobals($vars);

        return new Template($ref->getTemplateName(), $vars);
    }
}
```

Choosing the best template engine

To use as many different template engines in the same application, a special custom **ChainEngine** class can be added to the system.

The **ChainEngine** object is a **Composite** object that embeds all supported template engines and chooses the most fitted one to use whenever the given view is supported.

The ChainEngine Composite Design



The ChainEngine Composite Class

```
define('VIEWS_DIR', __DIR__.'/views');

$twigLoader = new \Twig_Loader_Filesystem(VIEWS_DIR);
$twig = new \Twig_Environment($twigLoader);
$plates = new \League\Plates\Engine(VIEWS_DIR, null);

$php = new PhpEngine(VIEWS_DIR);
$php->addHelper(new DateHelper());
$php->addHelper(new TextHelper());

$engine = new ChainEngine();
$engine->add(new PlatesEngineAdapter($plates));
$engine->add(new TwigEngineAdapter($twig));
$engine->add($php);
```

The ChainEngine Composite Class

```php
// ChainEngine will choose Plates engine
$engine->evaluate('blog.tpl');

// ChainEngine will choose Twig engine
$engine->evaluate('blog.twig');

// ChainEngine will choose PHP default engine
$engine->evaluate('blog.php');
```

The ChainEngine Composite Class

```
class ChainEngine implements EngineInterface
{
    private $engines;

    public function __construct(array $engines = [])
    {
        $this->engines = [];
        foreach ($engines as $engine) {
            $this->add($engine);
        }
    }

    public function add(EngineInterface $engine)
    {
        $this->engines[] = $engine;
    }
}
```

The ChainEngine Composite Class

```
class ChainEngine implements EngineInterface
{
    // ...

    private function getEngine(string $template)
    : EngineInterface
    {
        foreach ($this->engines as $engine) {
            if ($engine->supports($template)) {
                return $engine;
            }
        }
        throw new UnsupportedTemplateException('...');
    }
}
```

The ChainEngine Composite Class

```
class ChainEngine implements EngineInterface
{
    // ...

    public function supports(string $template): bool
    {
        $engine = null;
        try {
            $engine = $this->getEngine($template);
        } catch (UnsupportedTemplateException $e) {
            // Nothing to do here
        }

        return null !== $engine;
    }
}
```

The ChainEngine Composite Class

```php
class ChainEngine implements EngineInterface
{
    // ...

    function exists(string $template): bool
    {
        return $this->getEngine($template)->exists($template);
    }

    function loadTemplate(string $template, array $vars = [])
    : TemplateInterface
    {
        return $this->getEngine($template)->loadTemplate($template, $vars);
    }

    function evaluate(string $template, array $vars = []): string
    {
        return $this->getEngine($template)->evaluate($template, $vars);
    }
}
```

Behavioral Patterns

Behavioral patterns simplify the communication between objects in order to increase flexibility and decoupling.

Chain of Responsability – Command – Interpreter
Iterator – Mediator – Memento – Observer – State
Strategy – Template Method – Visitor

Mediator

The **Mediator** pattern defines an object that encapsulates how a set of objects interact in order to reduce communication complexity and modules coupling.

Avoiding code coupling

```php
class OrderManager
{
    private $logger;
    private $mailer;
    private $repository;
    private $templating;

    public function __construct(
        OrderRepositoryInterface $repository,
        MailerInterface $mailer,
        TemplatingInterface $templating,
        LoggerInterface $logger = null
    )
    {
        $this->repository = $repository;
        $this->templating = $templating;
        $this->mailer = $mailer;
        $this->logger = $logger;
    }
}
```

Avoiding code coupling

```php
class OrderManager
{
    // ...
    public function confirmOrder(Order $order, Payment $payment)
    {
        $order->pay($payment->getPaidAmount());
        $this->repository->save($order);

        if ($this->logger) {
            $this->logger->log('New order...');
        }

        $mail = new Email();
        $mail->recipient = $order->getCustomer()->getEmail();
        $mail->subject = 'Your order!';
        $mail->message = $this->templating->render('...');
        $this->mailer->send($mail);

        $mail = new Email();
        $mail->recipient = 'sales@acme.com';
        $mail->subject = 'New order to ship!';
        $mail->message = $this->templating->render('...');
        $this->mailer->send($mail);
    }
}
```

What are the problems with this code?

- Tight coupling between classes
- **OrderService** class has too many responsibilities
- Evolutivity and extensibility are limited
- Maintaining the code becomes difficult
- Unit testing will be harder

This code doesn’t follow the SOLID principles!

Responsabilities mixup...

```php
class OrderManager
{
    // ...
    public function confirmOrder(Order $order, Payment $payment)
    {
        $order->pay($payment->getPaidAmount());
        $this->repository->save($order);

        if ($this->logger) {
            $this->logger->log('New order...');
        }

        $mail = new Email();
        $mail->recipient = $order->getCustomer()->getEmail();
        $mail->subject = 'Your order!';
        $mail->message = 'Thanks for ordering...';
        $this->mailer->send($mail);

        $mail = new Email();
        $mail->recipient = 'sales@acme.com';
        $mail->subject = 'New order to ship!';
        $mail->message = '...';
        $this->mailer->send($mail);
    }
}
```

Main Responsability

Secondary Responsabilities

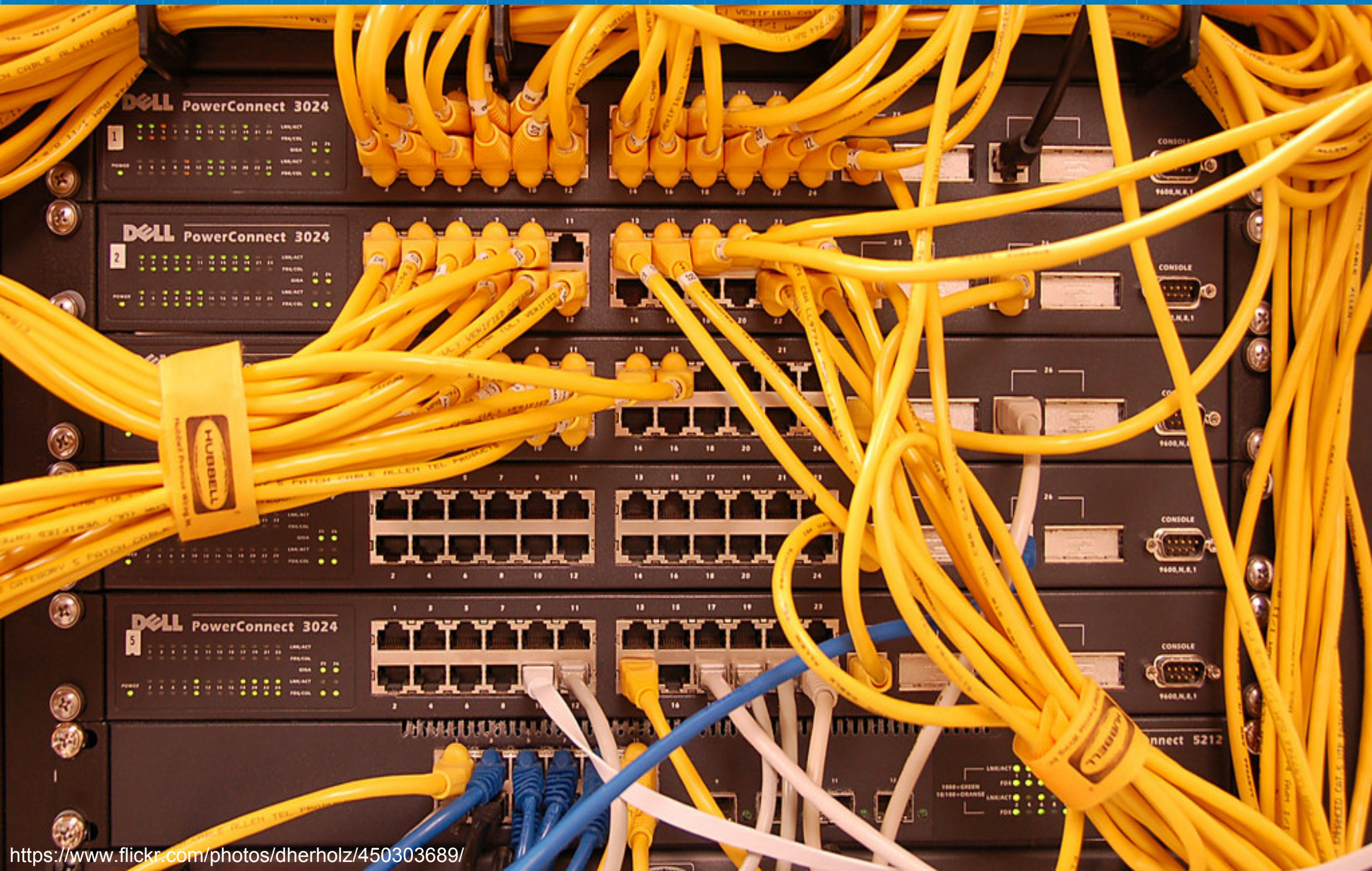
Mediators in real life – Air Traffic Control

https://www.flickr.com/photos/savannahcorps

Mediators in real life – Real Estate Broker

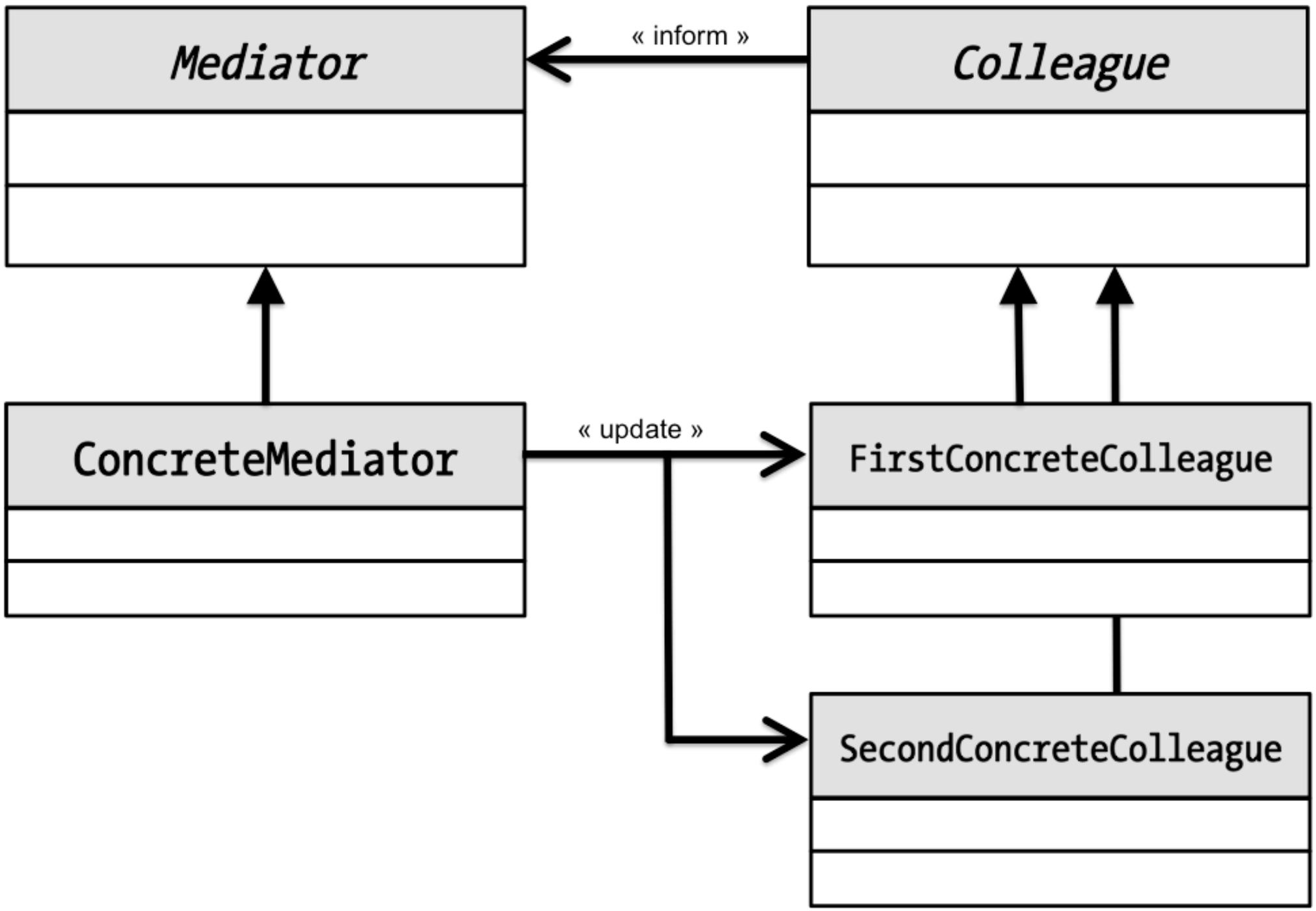
https://www.flickr.com/photos/neubie/729419705/

Mediators in real life – Hub/Switch



https://www.flickr.com/photos/dherholz/450303689/

UML Diagram


Mediator
« inform »
Colleague
ConcreteMediator
« update »
FirstConcreteColleague
SecondConcreteColleague

Implementing a Mediator object

```php
class Mediator implements MediatorInterface
{
    private $observers;

    public function __construct()
    {
        $this->observers = [];
    }

    public function register(string $event, MediatorCallable $mapping)
    {
        $this->observers[$event][] = $mapping;
    }

    public function trigger(string $event, MediatorCallableContext $context)
    {
        $callables = $this->observers[$event] ?? [];
        foreach ($callables as $callable) {
            call_user_func_array($callable->getCallable(), [$context]);
        }
    }
}
```

Encapsulating some contextual data

```php
class OrderWasPaidContext extends ObserverContext
{
    private $order;
    private $payment;

    public function __construct(Order $order, Payment $payment)
    {
        $this->order = $order;
        $this->payment = $payment;
    }

    public function getOrder(): Order
    {
        return $this->order;
    }

    public function getPayment(): Payment
    {
        return $this->payment;
    }
}
```

Decoupling dependencies with a Mediator

```php
class OrderManager
{
    private $mediator;
    private $repository;

    public function __construct(
        OrderRepositoryInterface $repository,
        MediatorInterface $mediator
    )
    {
        $this->repository = $repository;
        $this->mediator = $mediator;
    }

    public function confirmOrder(Order $order, Payment $payment)
    {
        $order->pay($payment->getPaidAmount());
        $this->repository->save($order);

        $context = new OrderWasPaidContext($order, $payment);
        $this->mediator->trigger('order.paid', $context);
    }
}
```

Separating concerns and responsabilities

```php
class OrderLogger
{
    private $logger;

    public function __construct(Prs\Log\LoggerInterface $logger)
    {
        $this->logger = $logger;
    }

    public function onOrderWasPaid(OrderWasPaidContext $context)
    {
        $this->logger->log($context->getOrder()->getReference());
        $this->logger->log($context->getPayment()->getPaidAmount());
    }
}
```

Separating concerns and responsabilities

```php
class OrderLogger
{
    private $logger;

    public function __construct(Prs\Log\LoggerInterface $logger)
    {
        $this->logger = $logger;
    }

    public function onOrderWasPaid(OrderWasPaidContext $context)
    {
        $this->logger->log($context->getOrder()->getReference());
        $this->logger->log($context->getPayment()->getPaidAmount());
    }
}
```

Separating concerns and responsabilities

```php
class CustomerNotifier
{
    private $mailer;
    private $templating;

    // ...
    public function onOrderWasPaid(OrderWasPaidContext $context)
    {
        $order = $context->getOrder();
        $payment = $context->getPayment();

        $mail = new Email();
        $mail->recipient = $order->getCustomer()->getEmail();
        $mail->subject = 'Your order is now confirmed!';
        $mail->message = $this->templating->render('mail/order.txt', [
            'order' => $order,
            'payment' => $payment,
        ]);

        $this->mailer->send($mail);
    }
}
```

Separating concerns and responsabilities

```php
class SalesDepartmentNotifier
{
    private $mailer;
    private $templating;

    // ...
    public function onOrderWasPaid(OrderWasPaidContext $context)
    {
        $order = $context->getOrder();
        $payment = $context->getPayment();

        $mail = new Email();
        $mail->recipient = 'sales@acme.com';
        $mail->subject = 'New placed order to ship!';
        $mail->message = $this->templating->render('mail/order_ship.txt', [
            'order' => $order,
            'payment' => $payment,
        ]);

        $this->mailer->send($mail);
    }
}
```

Wiring all the things together

```php
// Configure the listeners / observers
$listener1 = new OrderLogger($logger);
$listener2 = new CustomerNotifier($mailer, $templating);
$listener3 = new SalesDepartmentNotifier($mailer, $templating);

// Configure the mediator
$mediator = new Mediator();
$mediator->register('order.paid', new MediatorCallable($listener1, 'onOrderWasPaid'));
$mediator->register('order.paid', new MediatorCallable($listener3, 'onOrderWasPaid'));
$mediator->register('order.refunded', new MediatorCallable($listener2, 'onOrderWasRefunded'));

// Process the paid order
$order = new Order($customer, 150.90);
$payment = Payment::fromHttpRequest($request);

$service = new OrderService($repository, $mediator);
$service->confirmOrder($order, $payment);
```

Only the mediator holds the listeners.

Benefits of the new refactored code

- 👍 OrderService doesn't know anything about its collaborators but only keeps a reference to the Mediator. Coupling was drastically reduced.

- 👍 The Mediator can fire multiple kinds of events and pass any context to the notified observers.

- 👍 Each observer encapsulates one single responsability.

Drawbacks of the new refactored code

- It requires more code.
- Debug can be harder.
- Small performance overhead.

Mediator Design Pattern in Symfony

```php
use Symfony\Component\EventDispatcher\Event;
use Symfony\Component\EventDispatcher\EventDispatcher;

$dp = new EventDispatcher();

$dp->addListener('event.name', function ($event) {
    // do whatever you want...
});

$dp->addListener('event.name', function ($event) {
    // do whatever you want...
});


$dp->dispatch('event.name', new Event());
```

Make the Symfony Core Truly Extensible

```php
class HttpKernel implements HttpKernelInterface
{
    protected $dispatcher;
    // ...

    private function handleRaw(Request $request, ...)
    {
        $this->requestStack->push($request);

        // request
        $event = new GetResponseEvent($this, $request, $type);
        $this->dispatcher->dispatch(KernelEvents::REQUEST, $event);

        if ($event->hasResponse()) {
            return $this->filterResponse($event->getResponse(), $request, $type);
        }

        // ...
        return $this->filterResponse($response, $request, $type);
    }
}
```

Thank You!

https://www.flickr.com/photos/moritzlino

SensioLabs

Confoo 2017 – Montréal - Hugo Hamon