Professional Object Oriented Programming with Design Patterns.

PHPC3

SensioLabs

License

Professional Object Oriented Programming with Design Patterns

Copyright © 2011-2018 SensioLabs – All Rights Reserved

Contents are provided for strictly personal use. No part of these contents may be reproduced, stored in a retrieval system, publicly shared or transmitted in any form or by any means.

Disclaimer

Contents are distributed on an "as is" basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the authors nor SensioLabs shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

Introduction

Overview of OOP in PHP

Objects and classes

Overloading (a bit...)

Reflection

Single inheritance

Interfaces

Encapsulation

Garbage collector

Abstract & final

Type hinting

Object cloning

Magic methods

Classes autoloading

Abstract and final classes

Namespaces

Traits

SPL and more...

Object Calisthenics

Objects Calisthenics

Objects Calisthenics are a series of **nine** small **exercises** and rules to help improve the quality of a codebase.

There were invented by **Jeff Bay** in his book *ThoughtWorks Anthology* published under the *Pragmatic Programer* collection.

Object Calisthenics

1. One level of indendation per method
2. Don’t use the ELSE keyword
3. Wrap all primitives in classes
4. First class collections
5. One instance operator per line
6. Don’t abbreviate
7. Keep all classes less than 50 lines
8. No classes with more than two instance variables
9. No getters and setters

The STUPID Principle

Singleton

Tight Coupling

Untestability

Premature Optimizations

Indescriptive naming

Duplications

The SOLID Principle

Single Responsibility

Open / Close

Liskov Substitution

Interface Segregation

Dependency Inversion

Single Responsability (SRP)

Keep your classes small and focused on small responsabilities.

« A class should have one, and only one, reason to change. »

```php
class CsvDataImporter
{
    public function import($file)
    {
        $records = array();
        if (false !== $handle = fopen($file, 'r')) {
            while ($record = fgetcsv($handle)) {
                $records[] = $record;
            }
        }
        fclose($handle);

        try {
            $this->db->beginTransaction();
            foreach ($records as $record) {
                $stmt = $this->db->prepare('INSERT INTO ...');
                $stmt->execute($record);
            }
            $this->db->commit();
        } catch (PDOException $e) {
            $this->db->rollback();
            throw $e;
        }
    }
}
```

```php
class DataImporter
{
    private $loader;
    private $storage;

    function __construct(FileLoader $loader, Storage $storage)
    {
        $this->loader = $loader;
        $this->storage = $storage;
    }

    public function import($file)
    {
        $records = $this->loader->load($file);

        return $this->storage->store($records);
    }
}
```

Open Close (OCP)

Software entities should be *open* to extension and closed to *modifications*.

« You should be able to extend a class' behavior, without modifying it. »

Open Close

Using a different file loader is now very easy without changing the importer class.

```
$loader = new CsvFileLoader();
$loader = new XmlFileLoader();
$loader = new JsonFileLoader();

$importer = new DataImporter($loader, ...);
$importer->import('foo.csv|xml|json');
```

Liskov Substitution

A dependency of type T should be replaced by another subtype S of T without breaking the code.

« Derived classes must be substitutable for their base classes. »

Liskov Substitution

```php
class TextFileLoader implements FileLoader
{
    public function load($file)
    {
        // ...
        return [ ... ];
    }
}

class CsvFileLoader extends TextFileLoader
{
    public function load($file)
    {
        // ...
        return [ ... ];
    }
}
```

Interface Segregation (ISP)

Interface segregation splits larger interfaces into smaller ones so that clients will only know about the methods they need to work with.

« Make fine grained interfaces that are client specific. »

Interface Segregation

```
interface FileLoaderInterface
{
    public function load($file);
    public function supports($type);
}

interface StorageInterface
{
    public function store(array $records);
}
```

Interface Segregation

```php
class DataImporter
{
    function __construct(FileLoaderInterface $loader, StorageInterface $storage)
    {
        // ...
    }

    public function import($file)
    {
        $ext = pathinfo($file, PATHINFO_EXTENSION);
        if (!$this->loader->supports($ext)) {
            throw new UnsupportedDataFile();
        }
        $records = $this->loader->load($file);

        return $this->storage->store($records);
    }
}
```

Dependency Inversion (DIP)

D.I is where components are given their dependencies through their constructors, methods, or directly into fields.

« Depend on abstractions, not on concretions. »

DI – The Wrong Way

```
class DataImporter
{
    private $loader;
    private $storage;

    public function __construct()
    {
        $this->loader = new CsvFileLoader();
        $this->storage = new DatabaseStorage();
    }
}
```

DI – The Right Way

```
class DataImporter
{
    private $loader;
    private $storage;

    function __construct(FileLoader $loader, Storage $storage)
    {
        $this->loader = $loader;
        $this->storage = $storage;
    }
 }
```

DI – Injecting dependencies

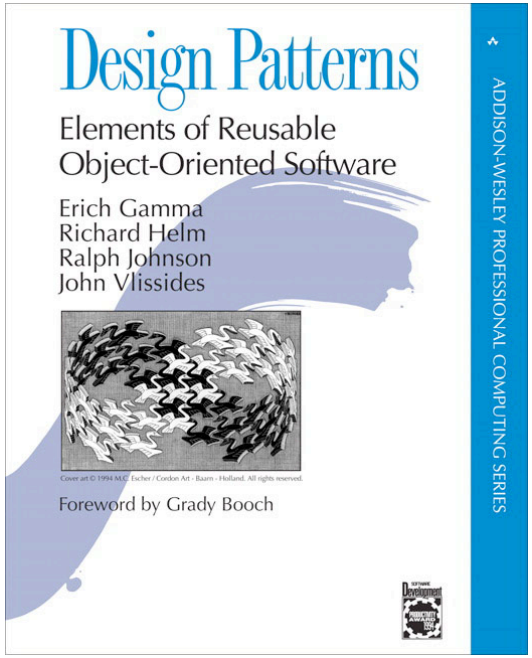
```
$importer = new DataImporter(
    new CsvFileLoader(),
    new DatabaseStorage('data', new PDO(…))
);

$importer->import('foo1.csv');
$importer->import('foo2.csv');
$importer->import('foo3.csv');
```

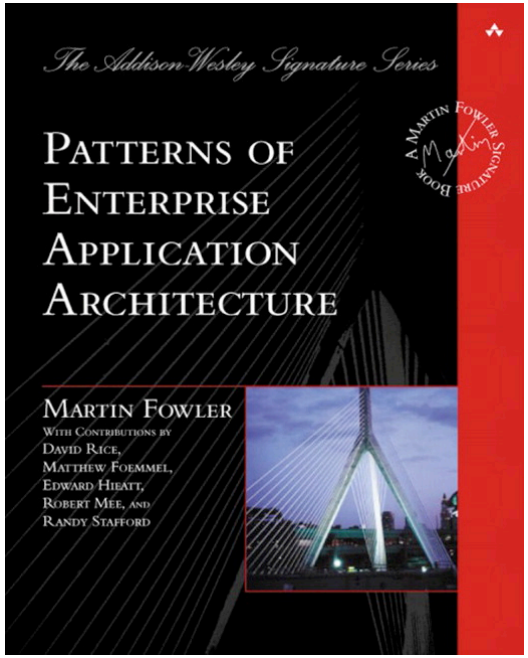
Disclaimer

Patterns are not always the holy grail!

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

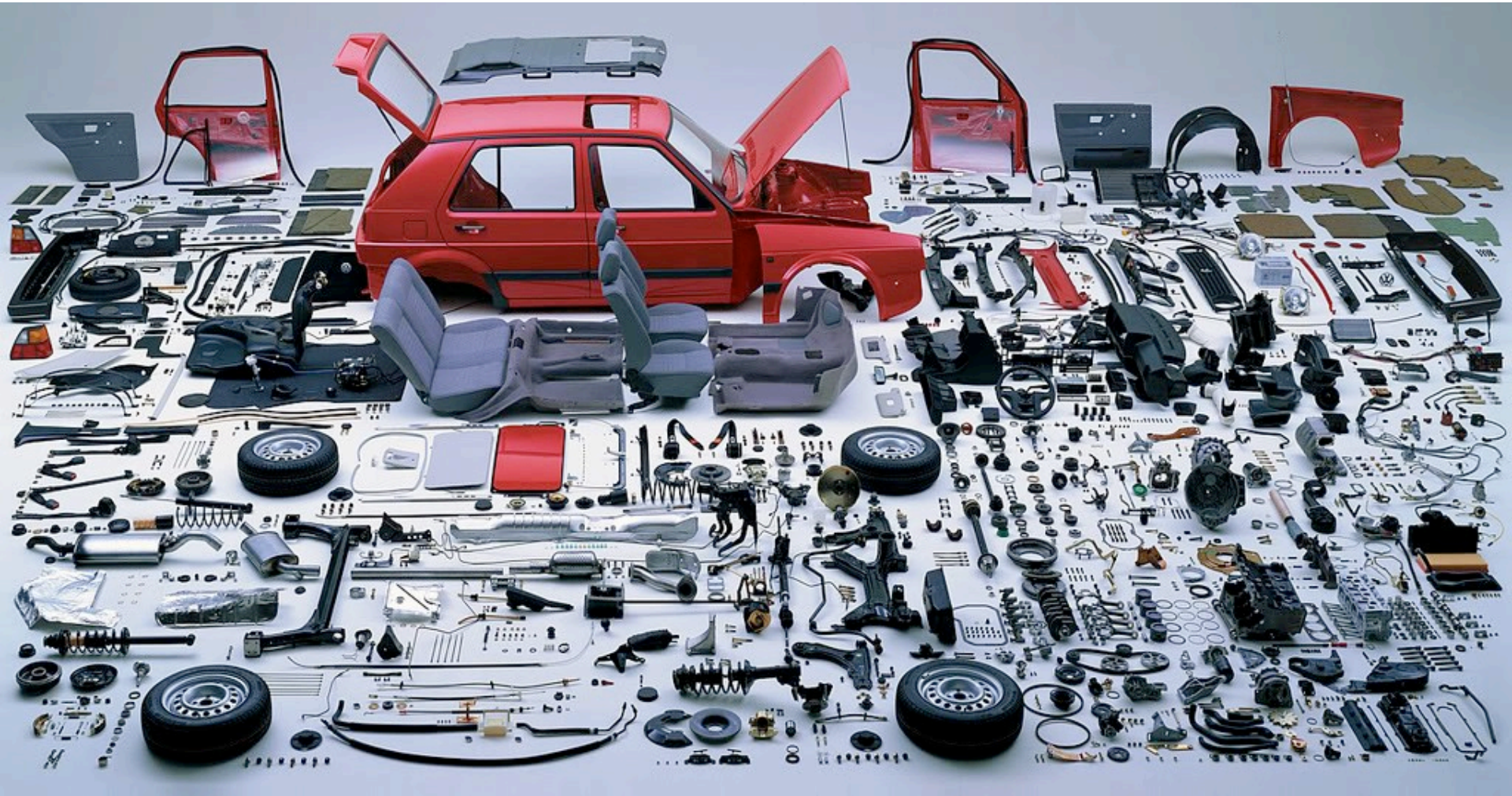
Recommended readings





Communication

Loose Coupling



Unit testability

```
Configuration read from /Users/ctheys/foo/d8/core/phpunit.xml.dist

.............................................................  63 / 557 ( 11%)
............................................................. 126 / 557 ( 22%)
............................................................. 189 / 557 ( 33%)
............................................................. 252 / 557 ( 45%)
............................................................. 315 / 557 ( 56%)
............................................................. 378 / 557 ( 67%)
............................................................. 441 / 557 ( 79%)
............................................................. 504 / 557 ( 90%)
.....................................................

Time: 15 seconds, Memory: 54.25Mb

OK (557 tests, 1135 assertions)

Generating code coverage report in HTML format ... done
[~/foo/d8/core]
01:30 PM [YesCT]  (8.x)
711 $
```

Maintenance

The Patterns Families

Creational Patterns

Creational patterns encapsulate the algorithms for creating and initializing objects.

Abstract Factory

Builder

Factory Method

Lazy Initialization

Prototype

Singleton

Structural Patterns

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

Adapter

Bridge

Composite

Decorator

Facade

Flyweight

Proxy

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

Singleton

Singleton ensures a class is always instantiated once.

Introduction

Ensure the uniqueness of an instance

```
Singleton
- static $instance : Singleton

+ static getInstance() : Singleton
- __construct()
- __clone()
```

- Static attribute to hold the unique instance
- Static method to create the instance on demand
- Class constructor is made private
- Object cloning must be disabled

```php
class Database
{
    private static $instance;
    private $dbh;

    private function __construct()
    {
        $this->dbh = new \PDO(
            'mysql:host=192.168.1.1;dbname=application',
            'root',
            'secret',
            [ \PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION ]
        );
    }

    public static function getInstance()
    {
        if (null === static::$instance) {
            static::$instance = new static();
        }

        return static::$instance;
    }
}
```

```php
class Database
{
    // …

    private function __clone()
    {
        // Avoid object cloning.
    }

    public function query($statement, array $params = array())
    {
        $stmt = $this->dbh->prepare($statement);
        $stmt->execute($params);

        return $stmt->fetchAll(\PDO::FETCH_ASSOC);
    }
}
```

Usage

```
$db = Database::getInstance();
$rs = $db->query('SELECT * FROM users');

$db = Database::getInstance();
$rs = $db->query('SELECT * FROM logs');
```

Singleton Limitations

Tight coupling

Global state

Hard to unit test

Variant

The Multiton Pattern

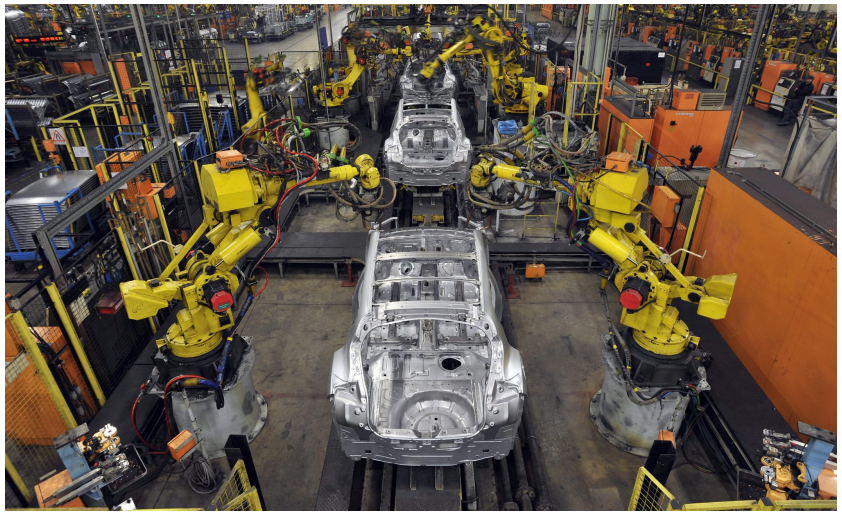
```php
// Set the databases connections configurations
Database::setConfiguration('master', 'mysql:host=...', 'root', 'secret');
Database::setConfiguration('slave1', 'mysql:host=...', 'root', 'secret');
Database::setDefaultConfiguration('slave');

// Returns the slave database connection (default)
$database1 = Database::getInstance();

// Returns the master database connection (explicit)
$database2 = Database::getInstance('master');

// Returns the slave database connection (explicit)
$database3 = Database::getInstance('slave');

var_dump($database1 === $database2);   // bool(false)
var_dump($database1 === $database3);   // bool(true)
```

Factory Method

Factory Method defines an interface for creating an object, but let subclasses decide which class to instantiate.

Introduction

The Concept of a Factory



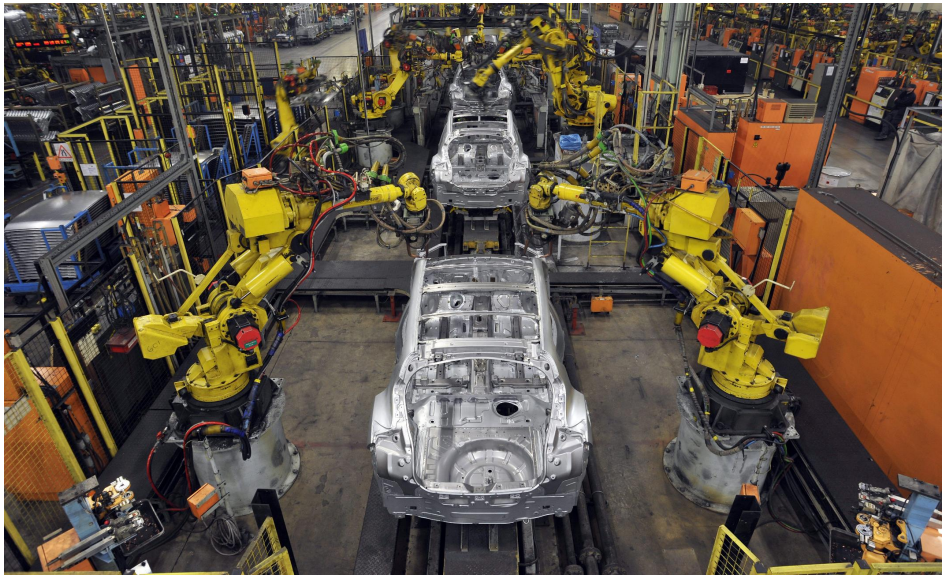
Car Factory

Bakery

The need for a factory

The factory object encapsulates the process for creating, assembling and initializing objects. It hides this whole process from the client code that just needs to get a new made object.

Having a Factory also allows to change the process for creating objects at any time without impacting the client code.



The Car Factory to assemble a new car.

The car factory can change the steps to assemble a new car in order to optimize the whole process, reduce costs and generate more benefits.

In the end, the final customer still buys an assembled car without taking care of how it was produced.

The bakery to produce bread based products.

This is the same for a baker who can slightly change the recipe for creating his bread and making more benefits. The final customer still buys a « baguette » in the end at the bakery.

The Static Factory Approach

Encapsulating Object Creation

The easiest way to go forward to a **Factory Method** pattern is at least to create a method in a class that is responsible for instanciating classes of objects to produce.

This method can be either an instance method or a static method. Even though it's still not a real Factory Method implementation, it remains a good practice to follow.

The Static Method Approach

```php
class MediaFactory
{
    public static function createMedia(string $path): string
    {
        $format = strtolower(pathinfo($path, PATHINFO_EXTENSION));

        if ('pdf' === $format) {
            return new PdfFile($path);
        }

        if (in_array($format, [ 'jpg', 'png', 'gif' ])) {
            return new ImageFile($path);
        }

        if ('txt' === $format) {
            return new TextFile($path);
        }

        throw new \UnexpectedValueException('Unexpected format '. $format);
    }
}
```

The Static Method Approach

```php
$file1 = MediaFactory::createMedia('/path/to/file.jpg');  // ImageFile
$file2 = MediaFactory::createMedia('/path/to/file.png');  // ImageFile
$file3 = MediaFactory::createMedia('/path/to/file.gif');  // ImageFile

$file4 = MediaFactory::createMedia('/path/to/file.pdf');  // PdfFile
$file5 = MediaFactory::createMedia('/path/to/file.txt');  // TextFile
```

This snippet shows how different types of objects are created based on the given file path extension. The **static factory method** either returns an ImageFile, PdfFile or TextFile object.

The Static Method in Action

```php
class DocumentRepository
{
    private $filesystem;
    private $imageManipulator;

    function __construct(Filesystem $filesystem, ImageManipulator $manipulator)
    {
        $this->filesystem = $filesystem;
        $this->imageManipulator = $nanipulator;
    }

    public function move(string $path, string $target): string
    {
        $file = MediaFactory::createMedia($path);

        if ($file instanceof ImageFile) {
            $file = $this->imageManipulator->resize($file, 120, 120);
        }

        return $this->filesystem->move($file->getRealPath(), $target);
    }
}
```

Pros & Cons of the Static Method

- Very simple,
- Very pragmatic,
- Easy to understand.

- Tight coupling between the factory and the client,
- Client code is more difficult to unit test,
- Impossible to switch to another factory,
- Difficulty to rely on dependencies in the factory.

The Simple Factory Approach

The Simple Factory Approach

```
class MediaFactory implements MediaFactoryInterface
{
    public function createMedia(string $path): string
    {
        $format = strtolower(pathinfo($path, PATHINFO_EXTENSION));

        if ('pdf' === $format) {
            return new PdfFile($path);
        }

        if (in_array($format, [ 'jpg', 'png', 'gif' ])) {
            return new ImageFile($path);
        }

        if ('txt' === $format) {
            return new TextFile($path);
        }

        throw new \UnexpectedValueException('Unexpected format '. $format);
    }
}
```

The Simple Factory Approach

```php
$factory = new MediaFactory();

$file1 = $factory->createMedia('/path/to/file.jpg');  // ImageFile
$file2 = $factory->createMedia('/path/to/file.png');  // ImageFile
$file3 = $factory->createMedia('/path/to/file.gif');  // ImageFile

$file4 = $factory->createMedia('/path/to/file.pdf');  // PdfFile
$file5 = $factory->createMedia('/path/to/file.txt');  // TextFile
```

This snippet shows how different types of objects are created based on the given file path extension. The factory is now an instance with a method that either returns an ImageFile, PdfFile or TextFile object.

The Simple Factory in Action

```php
class DocumentRepository
{
    private $mediaFactory;
    // ...

    public function __construct(
        FilesystemInterface $filesystem,
        MediaFactoryInterface $mediaFactory,
        ImageManipulatorInterface $imageManipulator
    )
    {
        $this->filesystem = $filesystem;
        $this->mediaFactory = $mediaFactory;
        $this->imageManipulator = $imageManipulator;
    }

    public function move(string $path, string $target): string
    {
        $file = $this->mediaFactory->createMedia($path);

        // ...
    }
}
```

Pros & Cons of the Simple Factory

- Simple & pragmatic approach
- Loose coupling between factory and client code
- Uses dependency injection
- Easy to unit test the client code
- Easy to replace the actual factory by another

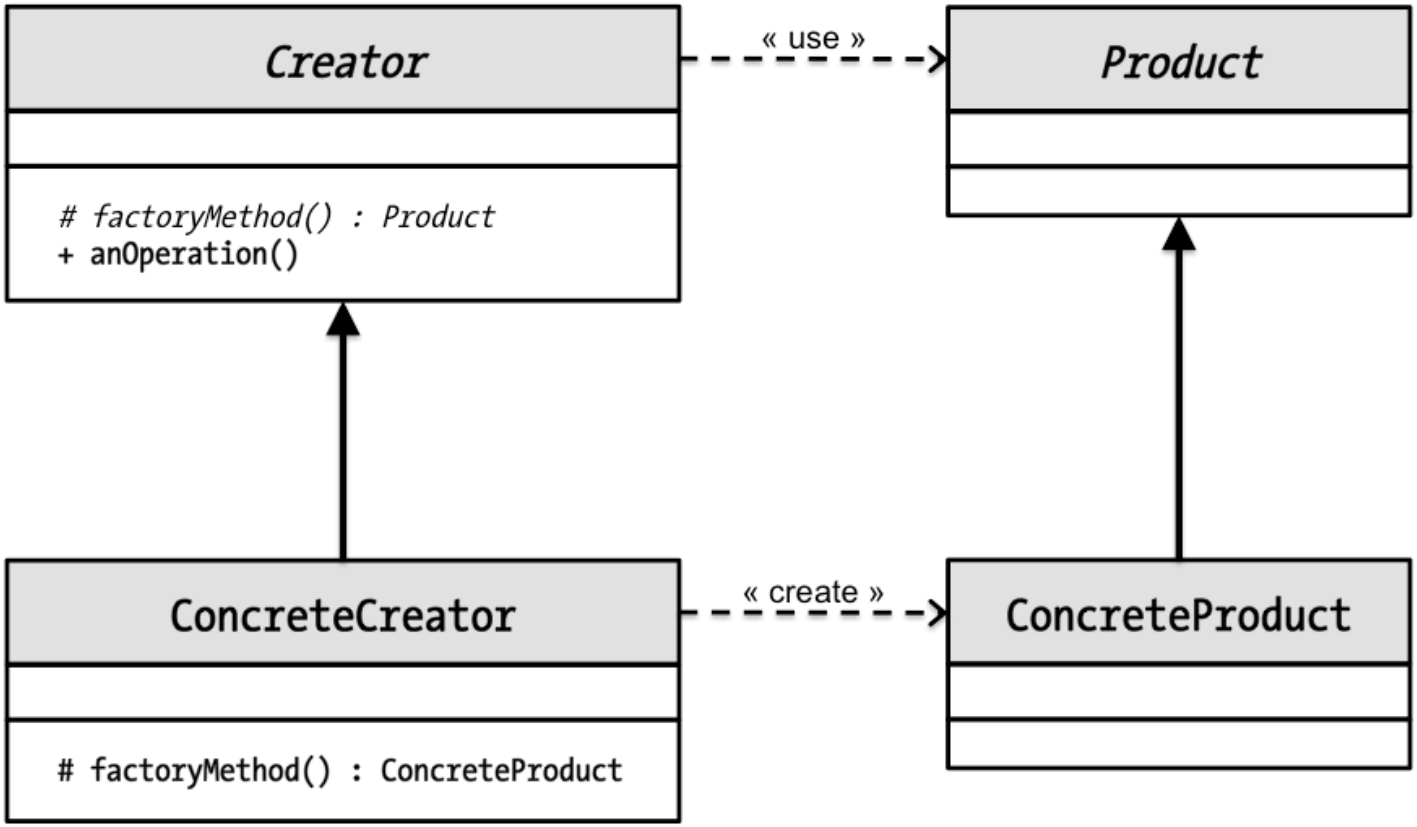
- It's still not a real factory method object.
- The factory produces multiple kinds of objects.

Introduction to the Factory Method Pattern

The Factory Method Pattern

- **Creational pattern**

- Encapsulate the creation of an object
- Algorithm to create object is partially implemented
- Creation of the specific object is delegated to subclasses
- Client code doesn’t know the type of the created object
- Each factory creates one kind of object




```
$factory = new ConcreteCreator();
$product = $factory->anOperation();
```

Abstract Creator Type

```
abstract class AbstractCreator
{
    /** @return AbstractProduct */
    abstract protected function factoryMethod();

    public function anOperation($foo, $bar)
    {
        $this->doSomethingFirst();

        $product = $this->factoryMethod();
        $product->setFooBar($foo, $bar);

        $this->doSomethingElse($product);

        return $product;
    }
}
```

Concrete Creator Type

```
class ConcreteCreator extends AbstractCreator
{
    protected function factoryMethod(): Product
    {
        $product = new ConcreteProduct();
        $product->initialize();

        return $product;
    }
}
```

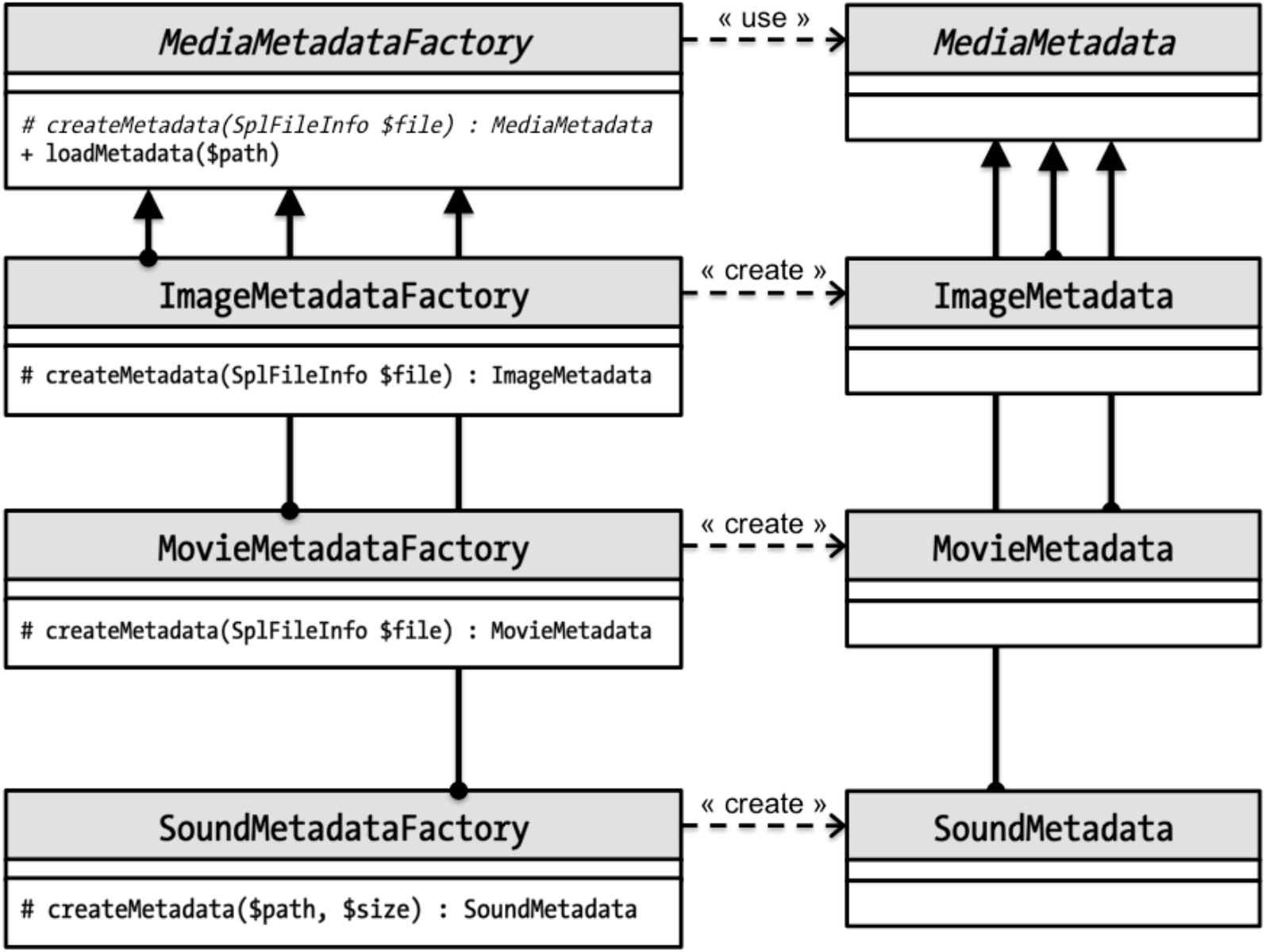
Implementing the Factory Method Design Pattern

Real world example

Media Gallery System

Media Metada Extractor

- We want to be able to extract metadata from media files
- Media files can be pictures, sounds or movies
- Media files have properties in common (filename, size…)
- They also have specific attributes (duration, dimensions…)
- We need a metadata factory for each type of media file.


MediaMetadataFactory
createMetadata(SplFileInfo $file) : MediaMetadata
+ loadMetadata($path)
« use »
MediaMetadata
ImageMetadataFactory
createMetadata(SplFileInfo $file) : ImageMetadata
« create »
ImageMetadata
MovieMetadataFactory
createMetadata(SplFileInfo $file) : MovieMetadata
« create »
MovieMetadata
SoundMetadataFactory
createMetadata($path, $size) : SoundMetadata
« create »
SoundMetadata

Splitting Responsabilities

```
src/MediaGallery/
 ├─── Analyzer/
 │     ├─── AnalysisFailedException.php
 │     ├─── AudioAnalyzer.php
 │     ├─── AudioVideoAnalyzer.php
 │     ├─── ImageAnalyzer.php
 │     ├─── MediaAnalyzerInterface.php
 │     ├─── VideoAnalyzer.php
 ├─── MediaMetadataException.php
 ├─── MediaMetadataFactoryInterface.php
 ├─── MediaMetadataInterface.php
 ├─── MediaNotFoundException.php
 └─── Metadata/
       ├─── ImageMetadata.php
       ├─── ImageMetadataFactory.php
       ├─── MediaMetadata.php
       ├─── MediaMetadataFactory.php
       ├─── MovieMetadata.php
       ├─── MovieMetadataFactory.php
       ├─── SoundMetadata.php
       └─── SoundMetadataFactory.php
```

The Abstract Metadata Factory

```
namespace MediaGallery;

interface MediaMetadataFactoryInterface
{
    /**
     * Loads the metadata of a given multimedia file.
     *
     * @throws MediaNotFoundException
     */
    public function loadMetadata(string $path): MediaMetadataInterface;
}
```

The Abstract Media Metadata Factory

```php
namespace MediaGallery\Metadata;

use MediaGallery\Analyzer\MediaAnalyzerInterface;
use MediaGallery\MediaMetadataFactoryInterface;
use MediaGallery\MediaMetadataInterface;
use MediaGallery\MediaNotFoundException;

abstract class MediaMetadataFactory implements MediaMetadataFactoryInterface
{
    private $analyzer;

    public function __construct(MediaAnalyzerInterface $analyzer)
    {
        $this->analyzer = $analyzer;
    }

    abstract protected function createMetadata(\SplFileInfo $file): MediaMetadataInterface;

    protected function analyze(\SplFileInfo $file): array
    {
        return $this->analyzer->analyze($file);
    }
}
```

The Abstract Media Metadata Factory

```
abstract class MediaMetadataFactory implements MediaMetadataFactoryInterface
{
    // ...

    public function loadMetadata(string $path): MediaMetadataInterface
    {
        if (!is_readable($path)) {
            throw new MediaNotFoundException(sprintf('%s not readable.', $path));
        }

        $file = new \SplFileInfo($path);

        $metadata = $this->createMetadata($file);
        $metadata->initialize($file);

        return $metadata;
    }
}
```

Summary so far…

- The **MediaMetadataFactory** receives an analyzer object
- The analyzer extracts information from the given file
- There is one analyzer instance per type of file
- The **loadMetadata()** method creates a **MediaMetadata**
- Each produced object is a subtype of the **MediaMetadata** class.

The Abstract MediaMetada Class

The abstract **MediaMetadata** class is designed to encapsulate the common shared attributes by all media files types.

For each media file, it stores its real path, its size and its creation date on the filesystem.

The Abstract MediaMetada Class

```php
namespace MediaGallery\Metadata;

use MediaGallery\MediaMetadataInterface;

abstract class MediaMetadata implements MediaMetadataInterface
{
    private $realPath;
    private $size;
    private $createdAt;

    public function initialize(\SplFileInfo $file): void
    {
        $this->size = $file->getSize();
        $this->realPath = $file->getRealPath();
        $this->createdAt = $file->getMTime();
    }

    // … plus one getter for each private property
}
```

The Image Metadata Factory Class

The **ImageMetadataFactory** class is responsible for producing **ImageMetadata** objects. To analyze a picture, the factory uses an **ImageAnalyzer** instance.

Each **ImageMetadata** instance has a real path, a size, a creation date but also a set of dimensions (width & height) and an orientation (portrait, landscape or square).

The ImageMetadata Class

```php
namespace MediaGallery\Metadata;

class ImageMetadata extends MediaMetadata
{
    private const SQUARE = 'square';
    private const PORTRAIT = 'portrait';
    private const LANDSCAPE = 'landscape';

    private $width;
    private $height;

    public function getHeight(): int
    {
        return $this->height;
    }

    public function getWidth(): int
    {
        return $this->width;
    }
}
```

The ImageMetadata Class

```
class ImageMetadata extends MediaMetadata
{
    // ...

    public function getOrientation(): string
    {
        if ($this->width === $this->height) {
            return self::SQUARE;
        }

        return $this->width > $this->height
            ? self::LANDSCAPE
            : self::PORTRAIT;
    }
}
```

```php
class ImageMetadata extends MediaMetadata
{
    // ...
    public function __construct(int $width, int $height)
    {
        \Assert\Assertion::greaterOrEqualThan($width, 1);
        \Assert\Assertion::greaterOrEqualThan($height, 1);

        $this->width  = $width;
        $this->height = $height;
    }
}
```

The ImageAnalyzer Class

```
namespace MediaGallery\Analyzer;

class ImageAnalyzer implements MediaAnalyzerInterface
{
    public function analyze(\SplFileInfo $file): array
    {
        $path = $file->getRealPath();
        if (!$path || !$metadata = getimagesize($path)) {
            throw new AnalysisFailedException(sprintf(
                'Unable to extract image metadata for path %s.',
                $path
            ));
        }

        return [ 'width'  => $metadata[0], 'height' => $metadata[1] ];
    }
}
```

The ImageMetadaFactory Class

```
namespace MediaGallery\Metadata;

class ImageMetadataFactory extends MediaMetadataFactory
{
    /**
     * Creates the specific ImageMetadata object.
     */
    protected function createMetadata(\SplFileInfo $file): MediaMetadataInterface
    {
        $metadata = $this->analyze($file);

        return new ImageMetadata($metadata['width'], $metadata['height']);
    }
}
```

Using the Image Metada Factory

```
use MediaGallery\Analyzer\ImageAnalyzer;
use MediaGallery\Metadata\ImageMetadataFactory;

$factory = new ImageMetadataFactory(new ImageAnalyzer());
$metadata = $factory->loadMetadata('/path/to/image.png');

echo 'Path:        ', $metadata->getRealPath()    ,"\n";
echo 'Date:        ', $metadata->getCreatedAt()   ,"\n";
echo 'Size:        ', $metadata->getSize()        ,"\n";
echo 'Width:       ', $metadata->getWidth()       ," px\n";
echo 'Height:      ', $metadata->getHeight()      ," px\n";
echo 'Orientation: ', $metadata->getOrientation() ,"\n";
```

The Movie Metadata Factory

The **MovieMetadataFactory** class is responsible for producing **MovieMetadata** objects. To analyze a movie, the factory uses an **VideoAnalyzer** instance.

Each **MovieMetadata** instance has a real path, a size, a creation date but also a set of specific attributes: a resolution in pixels (x & y axis), a duration, a frame rate and a frame count.

The MovieMetadata Class

```php
namespace MediaGallery\Metadata;

class MovieMetadata extends MediaMetadata
{
    private $xResolution;
    private $yResolution;
    private $frameRate;
    private $frameCount;
    private $duration;

    function __construct(int $xRes, int $yRes, int $duration, int $frameRate, int $frameCount)
    {
        \Assert\Assertion::greaterOrEqualThan($xRes, 1);
        // ...
        \Assert\Assertion::greaterOrEqualThan($frameCount, 1);

        $this->xResolution = $xRes;
        $this->yResolution = $yRes;
        $this->frameRate = round($frameRate);
        $this->frameCount = $frameCount;
        $this->duration = round($duration);
    }

    // ... getters for all attributes
}
```

The AudioVideoAnalyzer Class

```
namespace MediaGallery\Analyzer;

use GetId3\GetId3Core as MediaAnalyzer;

abstract class AudioVideoAnalyzer implements MediaAnalyzerInterface
{
    abstract protected function createAnalysisReport(
        \SplFileInfo $file, array $metadata): array;

    private function createAnalyzer(): MediaAnalyzer
    {
        $analyzer = new MediaAnalyzer();
        $analyzer
            ->setOptionMD5Data(true)
            ->setOptionMD5DataSource(true)
            ->setEncoding('UTF-8')
        ;

        return $analyzer;
    }
}
```

The AudioVideoAnalyzer Class

```php
abstract class AudioVideoAnalyzer implements MediaAnalyzerInterface
{
    // ...
    public function analyze(\SplFileInfo $file): array
    {
        $path = $file->getRealPath();
        $metadata = $this->createAnalyzer()->analyze($path);

        if (!$metadata || isset($metadata['error'])) {
            throw new AnalysisFailedException(sprintf(
                'Unable to get metadata from file %s.'
            ));
        }

        return $this->createAnalysisReport($file, $metadata);
    }
}
```

The AudioVideoAnalyzer Class

```php
namespace MediaGallery\Analyzer;

class VideoAnalyzer extends AudioVideoAnalyzer
{
    protected function createAnalysisReport(
        \SplFileInfo $file, array $metadata): array
    {
        $format = $metadata['video']['dataformat'];

        return [
            'resolution_x' => $metadata['video']['resolution_x'],
            'resolution_y' => $metadata['video']['resolution_y'],
            'duration'     => $metadata['playtime_seconds'],
            'frame_rate'   => $metadata['video']['frame_rate'],
            'frame_count'  => $metadata[$format]['video']['frame_count'],
        ];
    }
}
```

The MovieMetadataFactory Class

```php
namespace MediaGallery\Metadata;

class MovieMetadataFactory extends MediaMetadataFactory
{
    protected function createMetadata(\SplFileInfo $file): MediaMetadataInterface
    {
        // The $metadata is filled with information
        // provided by the awesome GetId3Core class
        // from the eponym library
        $metadata = $this->analyze($file);

        return new MovieMetadata(
            $metadata['resolution_x'],
            $metadata['resolution_y'],
            $metadata['duration'],
            $metadata['frame_rate'],
            $metadata['frame_count']
        );
    }
}
```

Using the Movie Metadata Factory

```
use MediaGallery\Analyzer\VideoAnalyzer;
use MediaGallery\Metadata\MovieMetadataFactory;

$factory  = new MovieMetadataFactory(new VideoAnalyzer());
$metadata = $factory->loadMetadata('/path/to/movie.mp4');

echo 'Path:          ', $metadata->getRealPath()    ,"\n";
echo 'Date:          ', $metadata->getCreatedAt()   ,"\n";
echo 'Size:          ', $metadata->getSize()        ,"\n";
echo 'Resolution X: ', $metadata->getXResolution() ,' px ', "\n";
echo 'Resolution Y: ', $metadata->getYResolution() ,' px ', "\n";
echo 'Duration:      ', $metadata->getDuration()    ," seconds\n";
echo 'Frame rate:    ', $metadata->getFrameRate()   ," fps\n";
echo 'Frames:        ', $metadata->getFrameCount()  ," frames\n";
```

The Sound Metadata Factory

The **SoundMetadataFactory** class is responsible for producing **SoundMetadata** objects. To analyze a sound file, the factory uses an **AudioAnalyzer** instance.

Each **SoundMetadata** instance has a real path, a size, a creation date but also a set of specific attributes: a bit rate, a playtime in seconds and whether or not the compression is lossless.

The SoundMetadata Class

```php
namespace MediaGallery\Metadata;

class SoundMetadata extends MediaMetadata
{
    private $lossless;
    private $bitrate;
    private $duration;

    function __construct(int $bitrate, int $duration, bool $lossless)
    {
        \Assert\Assertion::greaterOrEqualThan($bitrate, 1);
        \Assert\Assertion::greaterOrEqualThan($duration, 1);

        $this->lossless = $lossless;
        $this->bitrate = $bitrate;
        $this->duration = round($duration);
    }

    // ... getters for all attributes
}
```

The AudioAnalyzer Class

```php
namespace MediaGallery\Analyzer;

class AudioAnalyzer extends AudioVideoAnalyzer
{
    protected function createAnalysisReport(
        \SplFileInfo $file,
        array $metadata
    ): array {
        return [
            'lossless' => $metadata['audio']['lossless'],
            'duration' => $metadata['playtime_seconds'],
            'bitrate'  => $metadata['bitrate'],
        ];
    }
}
```

The SoundMetadataFactory Class

```php
namespace MediaGallery\Metadata;

class SoundMetadataFactory extends MediaMetadataFactory
{
    /** @return SoundMetadata */
    protected function createMetadata(\SplFileInfo $file): MediaMetadataInterface
    {
        $metadata = $this->analyze($file);

        return new SoundMetadata(
            $metadata['bitrate'],
            $metadata['duration'],
            $metadata['lossless']
       );
    }
}
```

Using the Sound Metadata Factory

```
use MediaGallery\Analyzer\VideoAnalyzer;
use MediaGallery\Metadata\SoundMetadataFactory;

$factory = new SoundMetadataFactory(new AudioAnalyzer());
$data = $factory->loadMetadata('/path/to/music.mp3');

echo 'Path:     ', $data->getRealPath()    ,"\n";
echo 'Date:     ', $data->getCreatedAt()   ,"\n";
echo 'Size:     ', $data->getSize()        ,"\n";
echo 'Duration: ', $data->getDuration()    ," s\n";
echo 'Bit rate: ', $data->getBitrate()     ," Hz\n";
echo 'Lossless: ', $data->isLossless() ? 'true' : 'false';
```

Going Further with the Factory Method Design Pattern

The Service Containers

Dependency Injection containers (or **service containers**) like the Symfony one are examples of implementing the **Factory Method** design pattern.

Their role is to encapsulate the process of creating and initializing objects on demand thanks to factory methods.

Service Container

```
$dic = new ServiceContainer();
$mailer  = $dic->get('mailer');
$logger  = $dic->get('logger');
$factory = $dic->get('metadata_factory');
```

A « Service Container » is an objects factory that creates services when they're requested by their names. It also ensures a service is only created once and shared in the application.

Implementing a Service Container

```
class ServiceContainer
{
    private $services = [];
    private $methodsMap = [];

    public function __construct()
    {
        $this->methodsMap = [
            'logger'  => 'getLoggerService',
            'mailer'  => 'getMailerService’,
            // ...
        ];
    }

    private function getLoggerService(): \Psr\Log\LoggerInterface
    {
        $this->services['logger'] = $instance = new \Logger();

        return $instance;
    }
}
```

Implementing a Service Container

```php
class ServiceContainer
{
    public function get(string $id)
    {
        $id = strtolower($id);
        if (isset($this->services[$id])) {
            return $this->services[$id];
        }

        if (!isset($this->methodsMap[$id])) {
            throw new ServiceNotFoundException();
        }

        // Get the "factory method" from the map
        // thanks to the service unique identifier
        $method = $this->methodsMap[$id];

        // Invoke the factory method to build
        // the requested service
        return call_user_func([ $this, $method ]);
    }
}
```

Lazy Initialization

+ Factory Method

= Service Container

Abstract Factory

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

Introduction

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

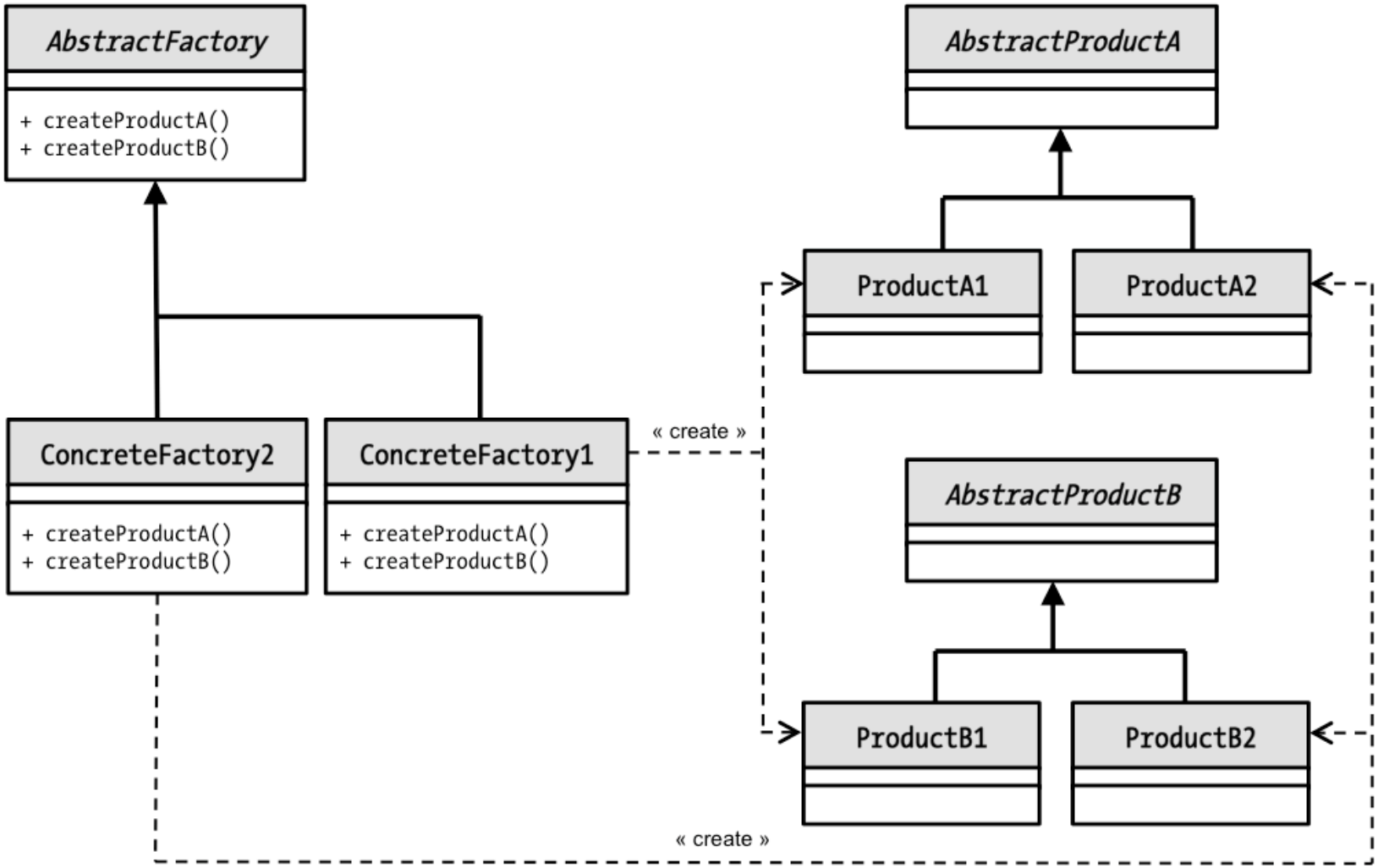
Refactoring context dependant code

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

    // ...

    public function handle(OrderCommand $command)
    {
        $price = $this->getUnitPrice($command)->getConvertedAmount();

        $order = new ConnectOrder();
        $order->setQuantity($command->getQuantity());
        $order->setUnitPrice($price);
        // ...
    }
}
```

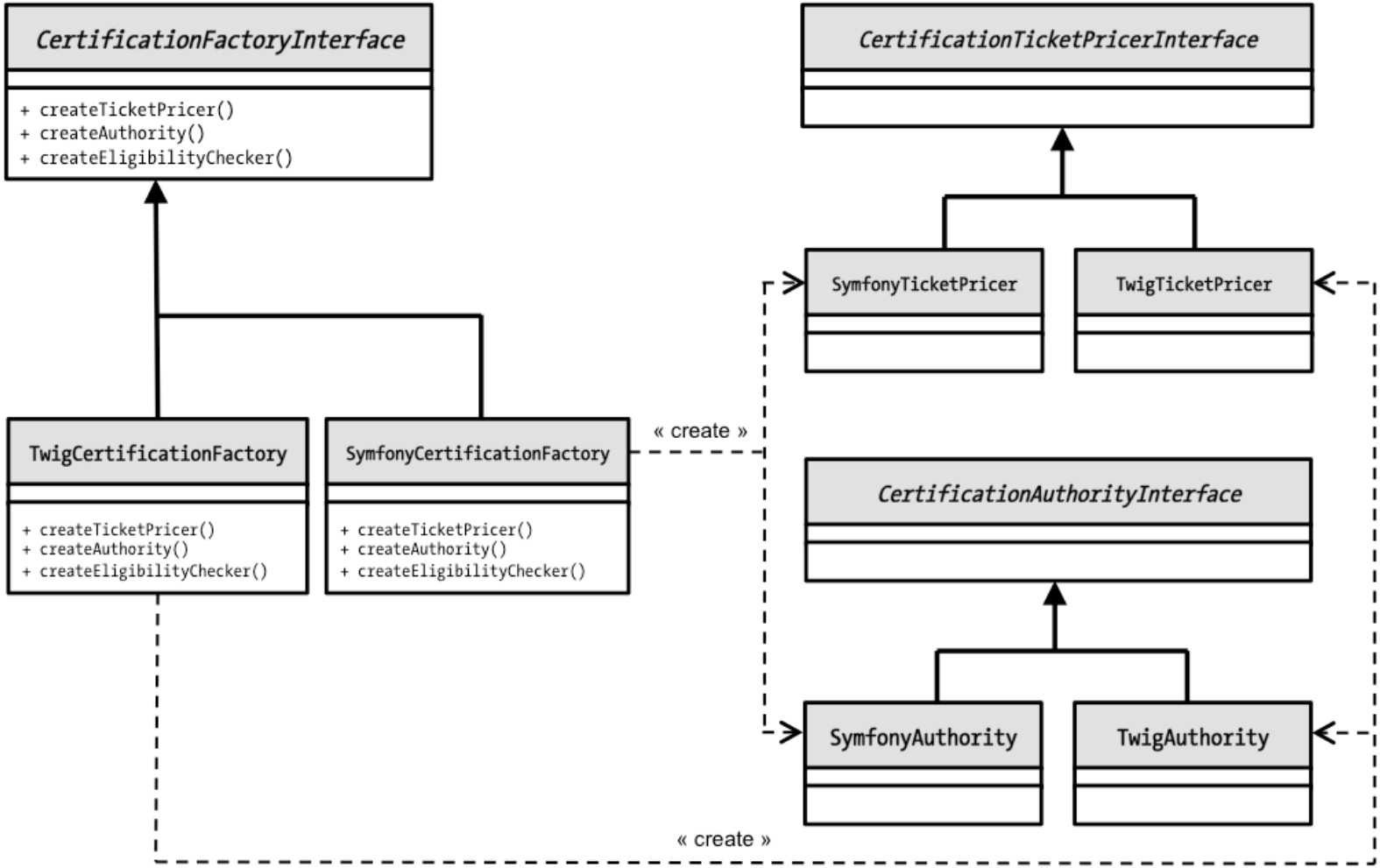
Refactoring context dependant code

```php
class PlaceOrderCommandHandler
{
    // ...

    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());
    }
}
```

UML Diagram


AbstractFactory
+ createProductA()
+ createProductB()
AbstractProductA
ProductA1
ProductA2
ConcreteFactory2
+ createProductA()
+ createProductB()
ConcreteFactory1
+ createProductA()
+ createProductB()
« create »
AbstractProductB
ProductB1
ProductB2
« create »

UML Diagram for Assessments Management



Implementing the Abstract Factory Design Pattern

```php
namespace SensioLabs\Certification;

interface CertificationFactoryInterface
{
    public function createEligibilityChecker()
        : CertificationEligibilityCheckerInterface;

    public function createTicketPricer()
        : CertificationTicketPricerInterface;

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

The Ticket Pricer

```
// 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 Ticket Pricer

```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 Ticket Pricer

```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('SL038744,SF3CE,35');
    $authority = new SymfonyCertificationAuthority(20, 10);
    $symfonyLevel = $authority->getCandidateLevel($assessment);

    $assessment = AssessmentResult::fromCSV('SL038744,TW1CE,17');
    $authority = new TwigCertificationAuthority(15);
    $twigLevel = $authority->getCandidateLevel($assessment);

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

The Certification Authority

```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 on the candidate's exam result.
     *
     * @throws CandidateNotCertifiedException
     * @throws UnsupportedAssessmentException
     */
    public function getCandidateLevel(AssessmentResult $result): string;
}
```

The Twig Certification Authority

```php
// ...
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

```
// ...
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

```
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

```
// ...
class SymfonyCertificationFactory implements CertificationFactoryInterface
{
    private $candidateRepository;
    private $assessmentRepository;

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

    // ...
}
```

```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

```
// ...
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

```
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);
    }
    // ...
}
```

Using the Factories

```
$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 Factories from a Higher 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;
    }

    // ...
}
```

```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);
    }
}
```

```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;
    }
}
```

```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 and 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.

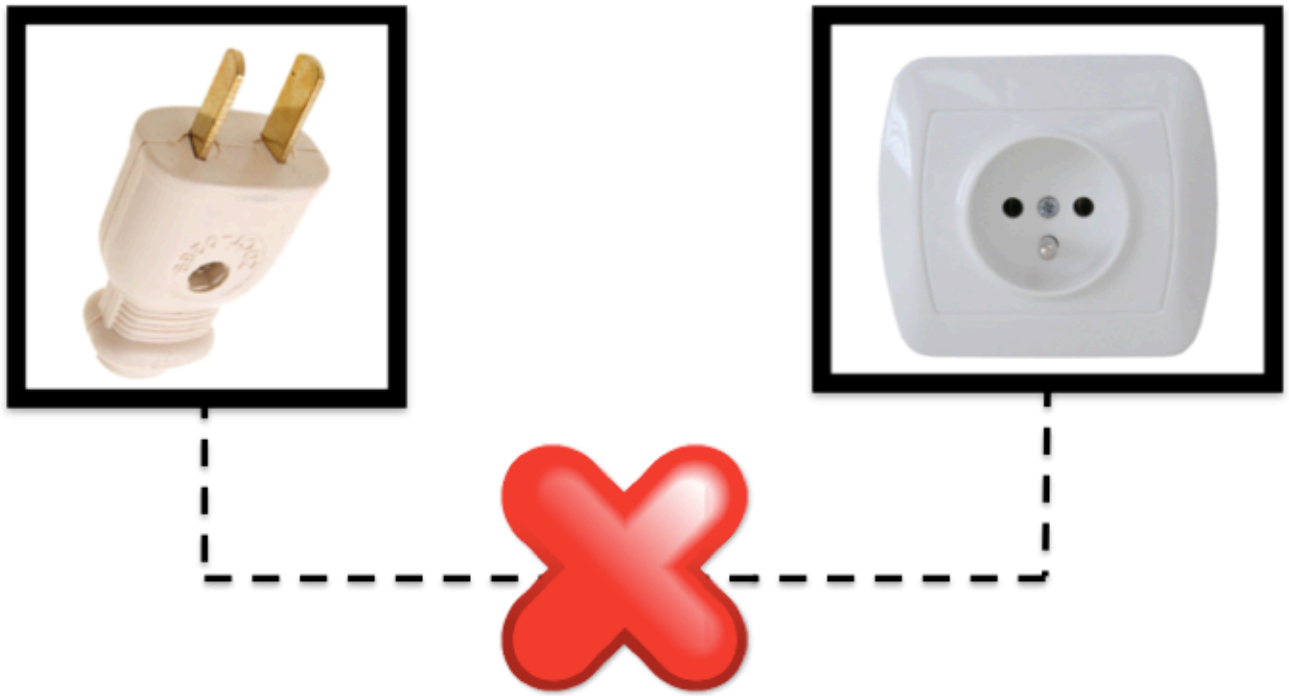
- 👎 High level of complexity and hard to implement,
- 👎 Involves lots of classes and interfaces,
- 👎 Requires the requirements to exactly fit the need for A.F.

Adapter

Making two incompatible objects work together without changing their interfaces.

Introduction

An example of incompatible interfaces

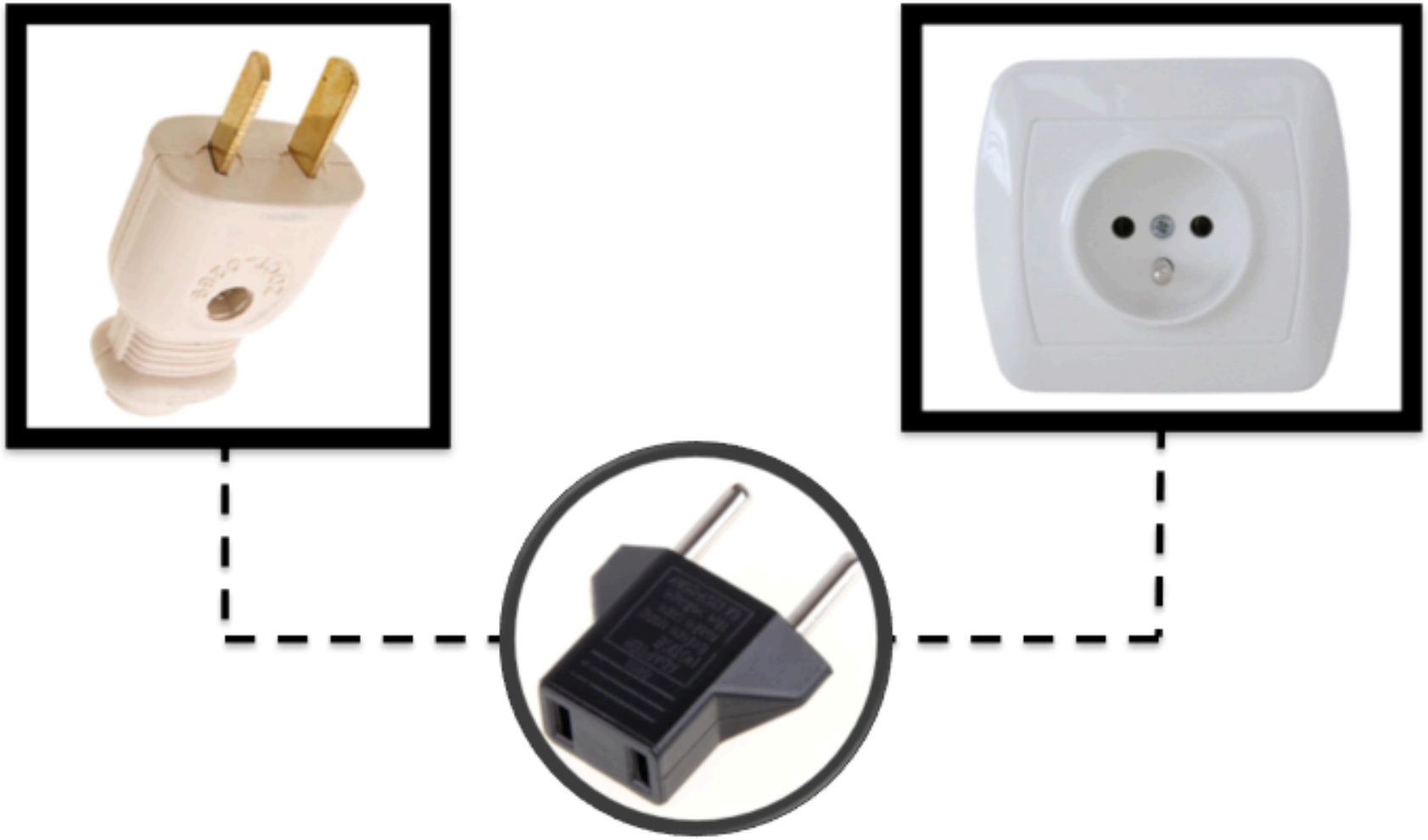


This male power plug is designed according to the American standards. It has two flat pins and supports 110V as power voltage.

This female power socket is designed according to the European standards. It has two cylindrical pins and supports a 220V power voltage.

This is a third party power plug **adapter** that converts a male American power plug interface into a female European power socket interface.

Adapting two incompatible interfaces



Concrete Adapters in the Real Life

Adaptateur VGA

Adaptateur DVI

Concrete Use Case

Real World Example

```
interface EngineInterface
{
    /** @return boolean */
    function exists($template);

    /** @return boolean */
    function supports($template);

    /** @return TemplateInterface */
    function loadTemplate($template, array $vars = []);

    /** @return string */
    function evaluate($template, array $vars = []);
}
```

Real World Example

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

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

Real World Example

```
class BlogController
{
    private $templating;

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

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

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

Real World Example

```
class BlogController
{
    // ...

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

The **render()** method invokes the public **evaluate()** method of the template engine dependency.

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?

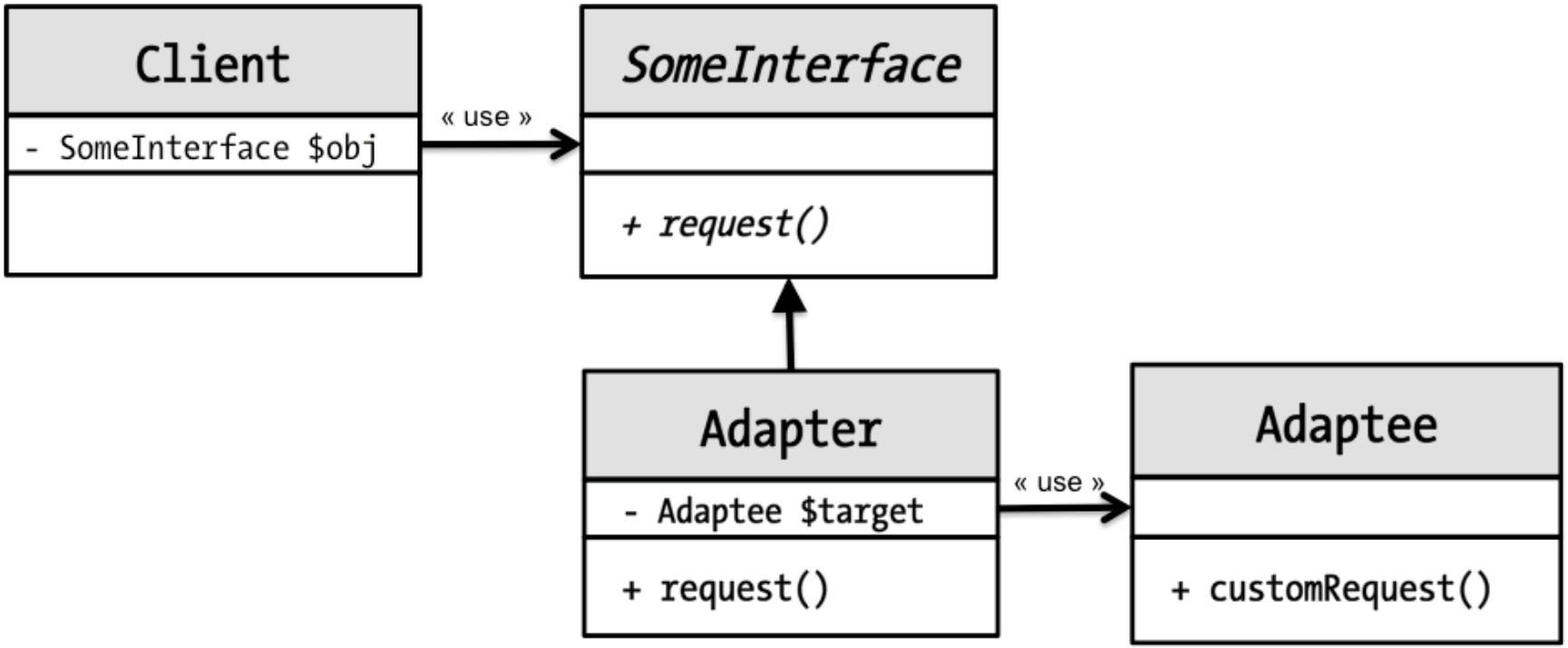
The Adapter Design Pattern

The Adapter Pattern

- **Structural pattern**

- Make two different interfaces compatible.
- Do not change any existing code.
- Add a new third party between the two APIs.

UML Diagram


Client
- SomeInterface $obj
« use »
SomeInterface
+ request()
Adapter
- Adaptee $target
+ request()
« use »
Adaptee
+ customRequest()

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 Target Interface

```
interface TargetInterface
{
    public function request();
}
```

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 Adaptee Class

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

The Adapter 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);
    }
}
```

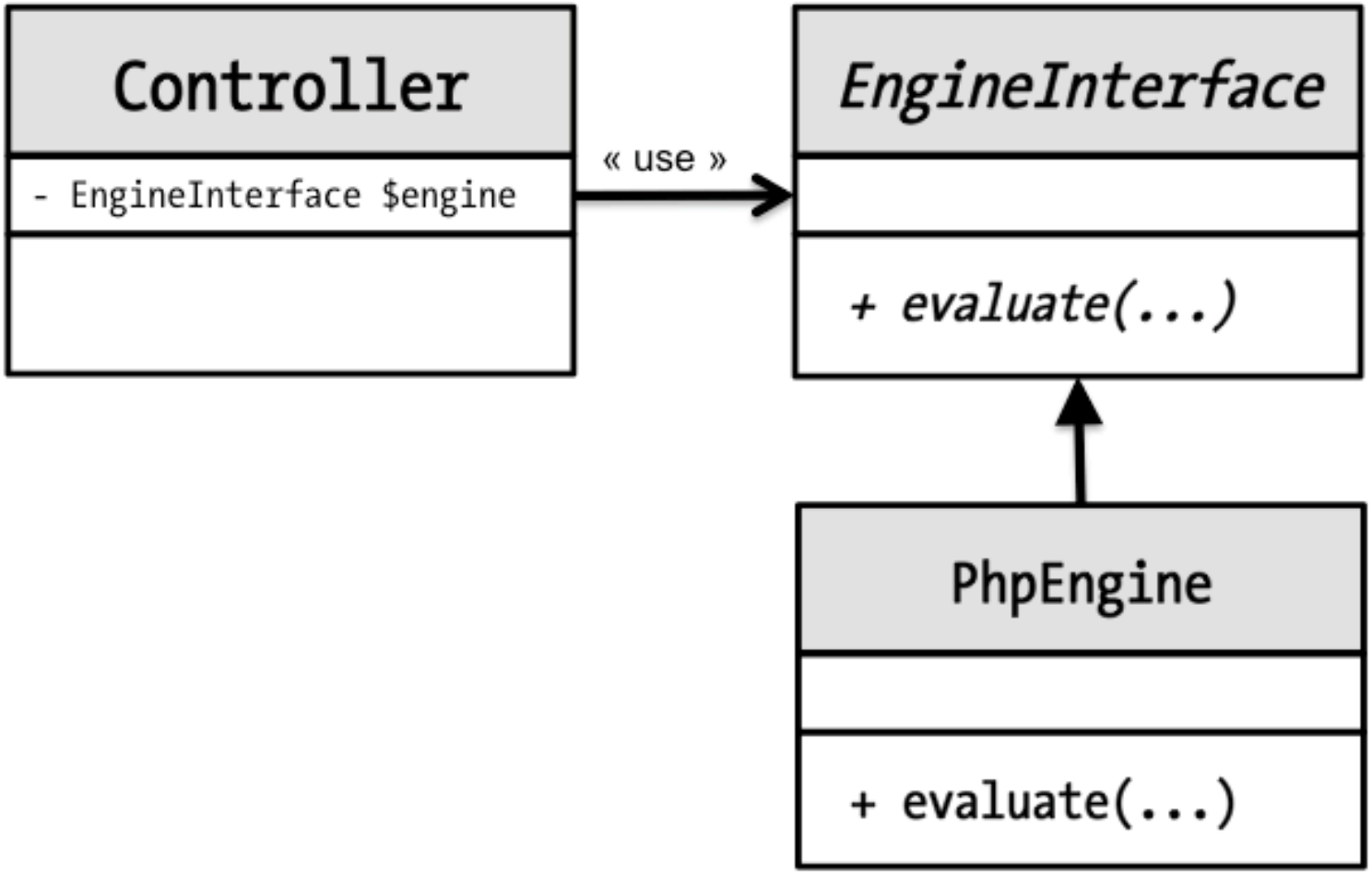
Use Cases for the Adapter Pattern

Uses Cases for the Adapter Pattern

- 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,
- …

Implementing the Adapter Design Pattern

The actual PHP engine implementation



The actual PHP engine class

```
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 engine class

```php
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 engine class

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

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

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

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

The actual PHP engine class

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

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

The actual PHP engine class

```php
class PhpEngine implements EngineInterface, \ArrayAccess
{
    // ...
    public function loadTemplate($template, array $vars = [])
    {
        if (!$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 engine class

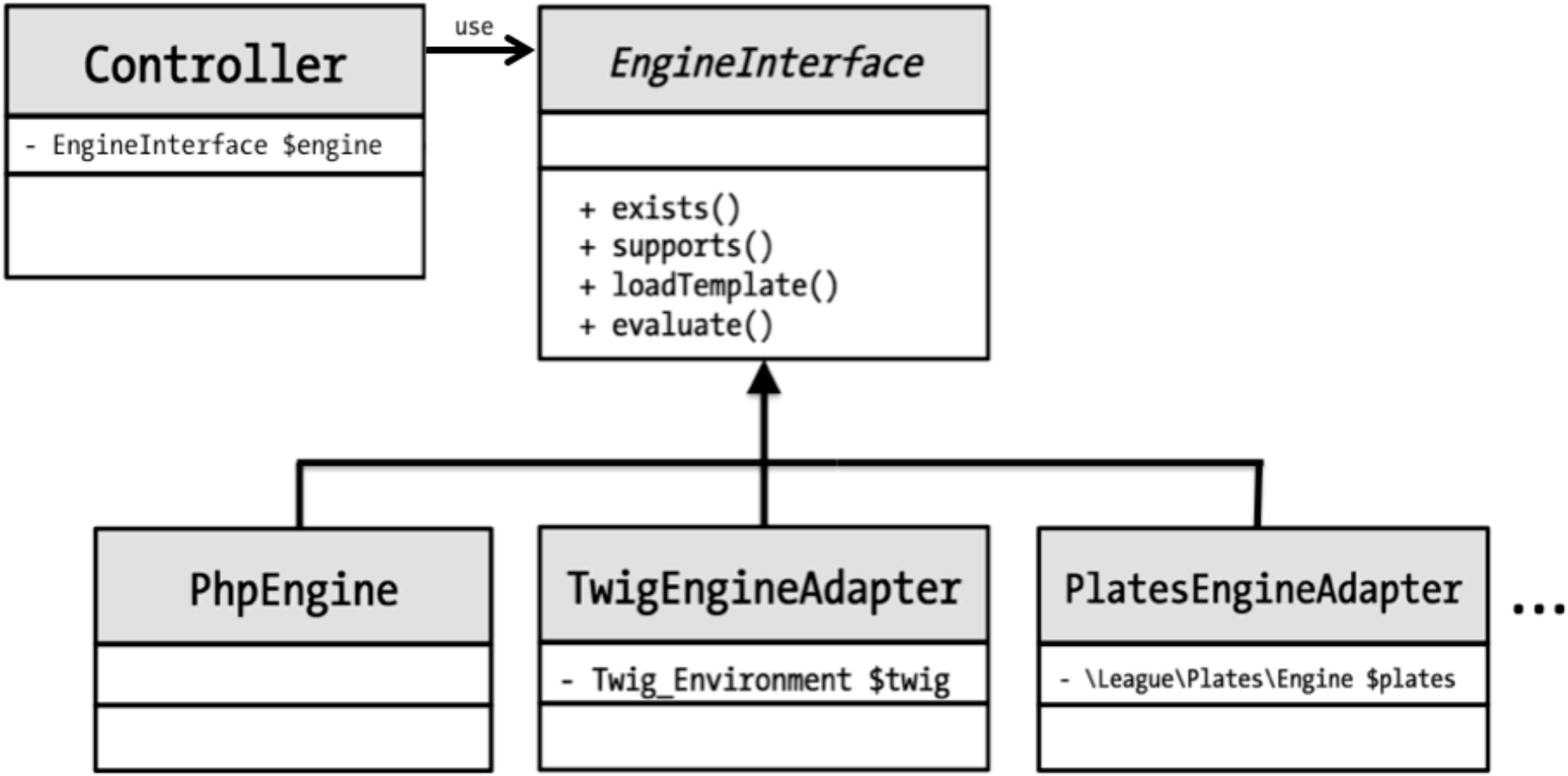
```php
class PhpEngine implements EngineInterface, \ArrayAccess
{
    // ...
    public function evaluate($template, array $vars = [])
    {
        $reference = $this->loadTemplate($template, $vars);

        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 third party libraries

We want to be able to support **Twig** or **Plates** or any template engine library.

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

UML Diagram


Controller
- EngineInterface $engine
use
EngineInterface
+ exists()
+ supports()
+ loadTemplate()
+ evaluate()
PhpEngine
TwigEngineAdapter
- Twig_Environment $twig
PlatesEngineAdapter
- \League\Plates\Engine $plates
...

Adapting the Twig Engine Library

The Twig Engine Adapter Class

```
$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 Twig Engine Adapter Class

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

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

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

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

The Twig Engine Adapter Class

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

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

        return true;
    }

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

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

The Twig Engine Adapter Class

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

        // ...
    }
}
```

```php
class TwigEngineAdapter implements EngineInterface
{
    // ...
    public function loadTemplate($template, array $vars = [])
    {
        // ...
        try {
            $reference = $this->twig->resolveTemplate($template);
        } catch (\Twig_Error_Loader $exception) {
            throw new TemplateNotFoundException(
                sprintf('Template %s does not exist.', $template),
                $exception
            );
        } catch (\Exception $exception) {
            throw new UnsupportedTemplateException(
                sprintf('Invalid template %s provided.', $template),
                $exception
            );
        }
        // ...
    }
}
```

The Twig Engine Adapter Class

```
class TwigEngineAdapter implements EngineInterface
{
    // ...
    public function loadTemplate($template, array $vars = [])
    {
        // ...

        return new Template(
            $reference->getTemplateName(),
            $this->twig->mergeGlobals($vars)
        );
    }
}
```

Adapting the Plates Engine Library

The Plates Engine Adapter

```
$plates = new \League\Plates\Engine(__DIR__.'/views');
$plates->setFileExtension(null);
$plates->addData([ 'foo' => 'bar' ]);

$engine = new PlatesEngineAdapter($plates);
$engine->evaluate('blog.tpl', ['posts' => 'posts']);
```

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

```php
class PlatesEngineAdapter implements EngineInterface
{
    private $plates;

    public function __construct(Plates $plates)
    {
        $this->plates = $plates;
    }

    public function exists($template)
    {
        return $this->plates->exists($template);
    }
}
```

```php
class PlatesEngineAdapter implements EngineInterface
{
    // ...
    public function supports($template)
    {
        return 'tpl' === pathinfo($template, PATHINFO_EXTENSION);
    }

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

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

```php
class PlatesEngineAdapter implements EngineInterface
{
    // ...
    public function loadTemplate($template, array $vars = [])
    {
        if (!$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 does not exist.',
                $template
            ));
        }

        return new Template(
            $this->plates->path($template),
            array_merge($this->plates->getData(), $vars)
        );
    }
}
```

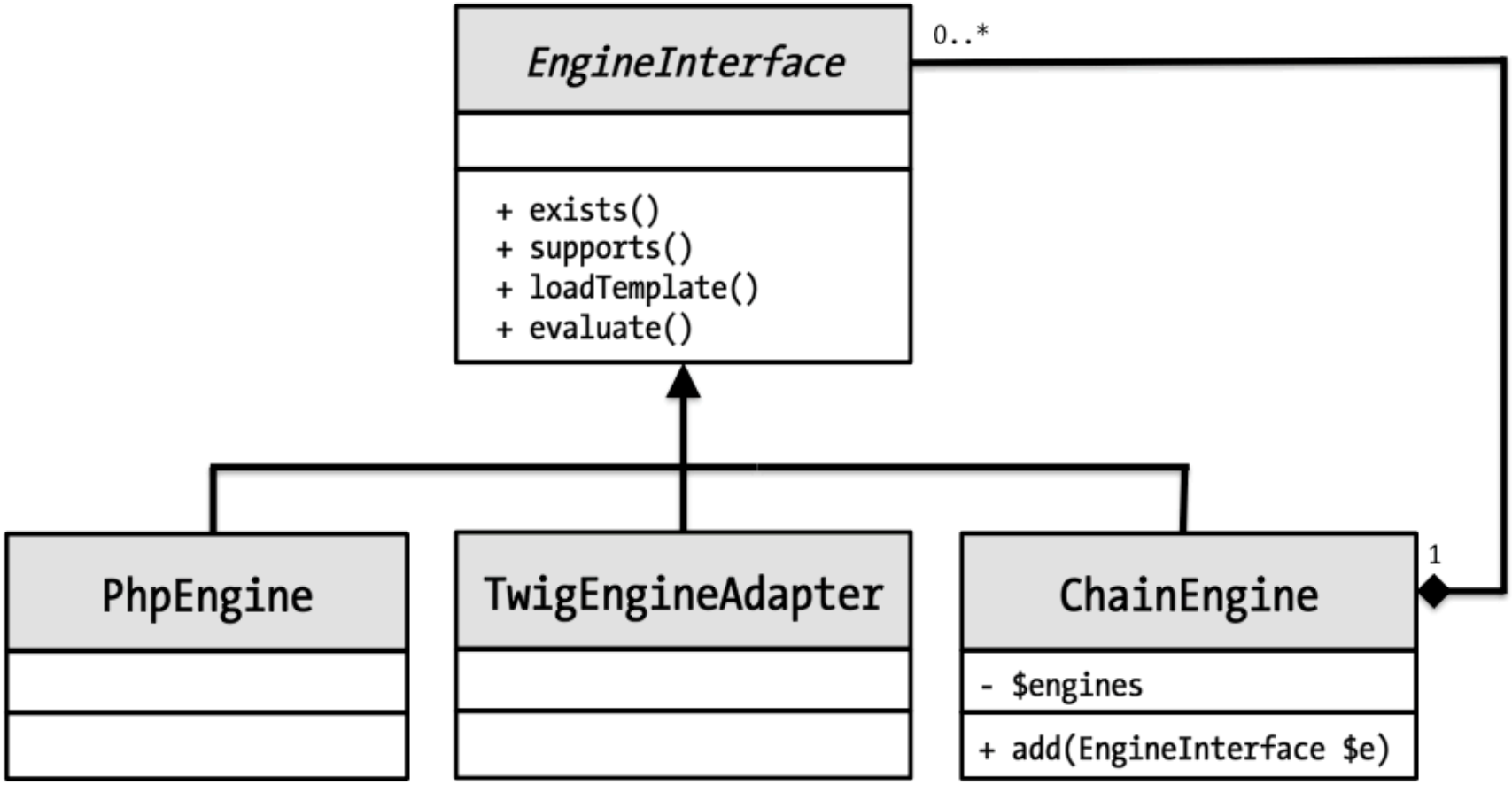
Now, you get the trick to support any new third party templating library!

Chaining Template Engines

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 correct one to use whenever the given view is supported.

UML Diagram


EngineInterface
+ exists()
+ supports()
+ loadTemplate()
+ evaluate()
0..*
1
PhpEngine
TwigEngineAdapter
ChainEngine
- $engines
+ add(EngineInterface $e)

The Chain Engine Composite

```
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());
```

The Chain Engine Composite

```
// ...

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

$engine->evaluate('blog.tpl', ['posts' => $posts]);  // Plates
$engine->evaluate('blog.twig', ['posts' => $posts]); // Twig
$engine->evaluate('blog.php', ['posts' => $posts]);  // PHP
```

The Chain Engine Composite

```
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 Chain Engine Composite

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

    /**
     * Returns the very first supported engine found for the provided template.
     *
     * @return EngineInterface
     */
    private function getEngine($template)
    {
        foreach ($this->engines as $engine) {
            if ($engine->supports($template)) {
                return $engine;
            }
        }
        throw new UnsupportedTemplateException(sprintf(
            'Template %s is not supported by any of the chained engines.',
            $template
        ));
    }
}
```

The Chain Engine Composite

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

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

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

The Chain Engine Composite

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

    public function exists($template)
    {
        return $this->getEngine($template)->exists($template);
    }

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

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

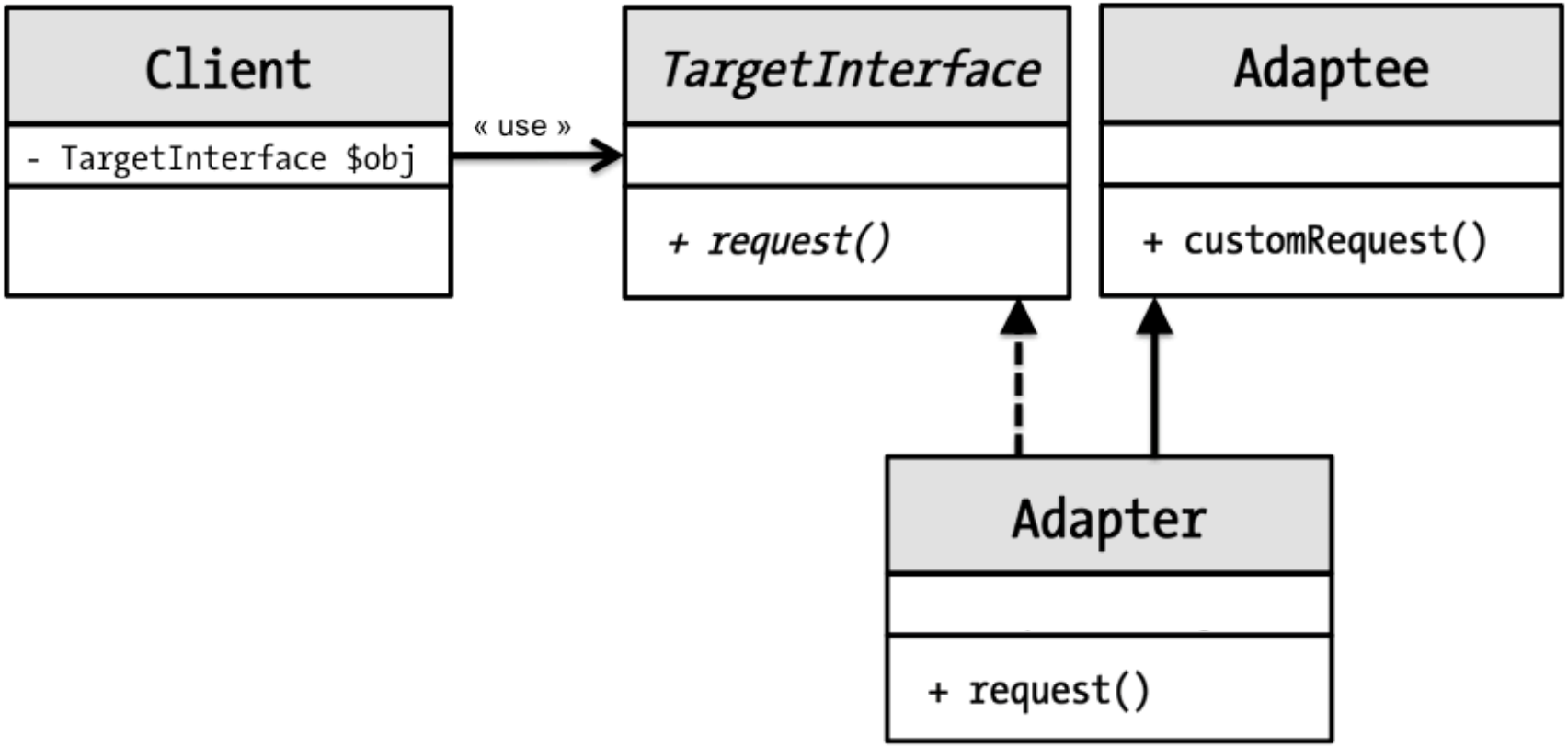
Adapter Pattern Variants

Adapter Pattern Variants

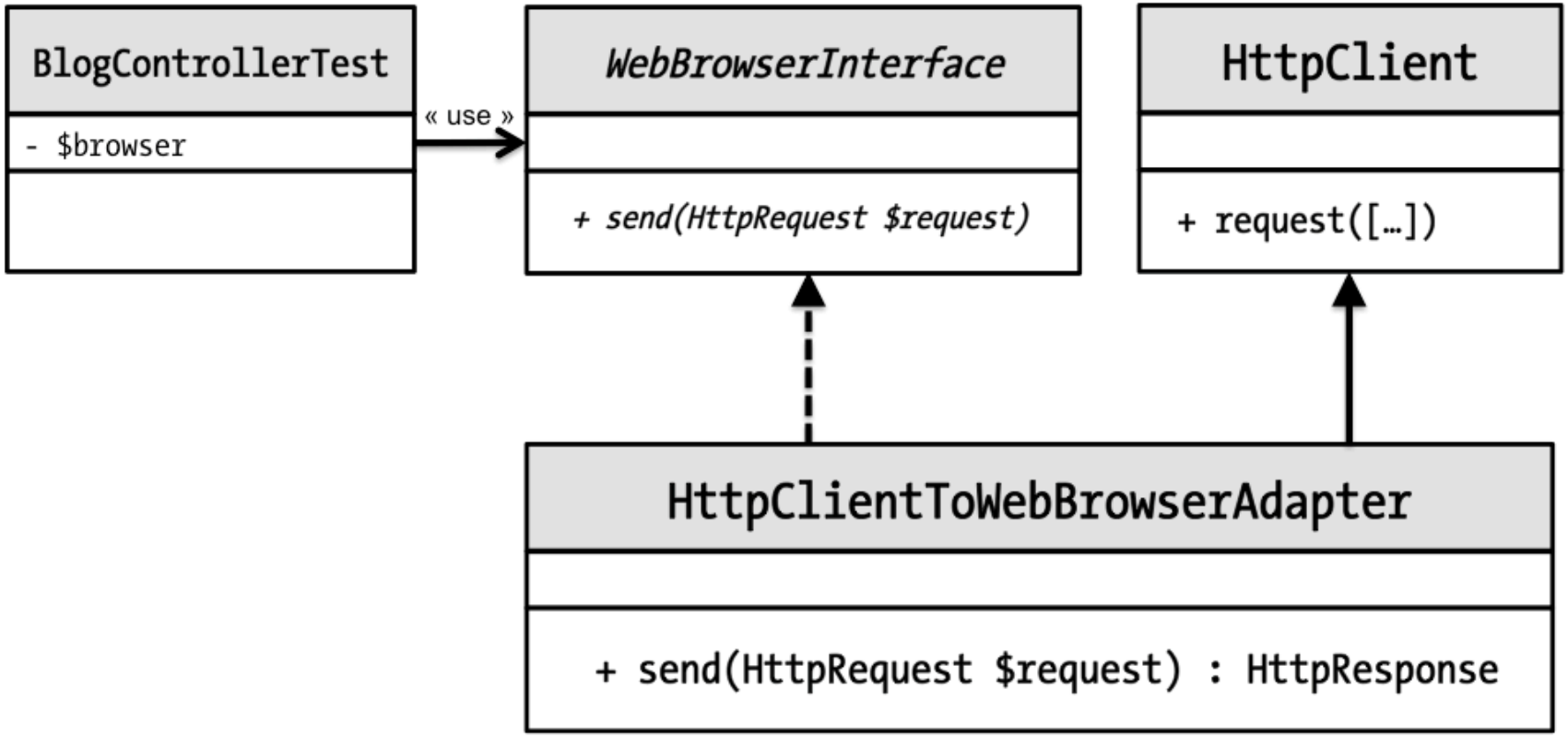
The **Adapter** design pattern is almost always used to adapt objects. In this case, it's said that the adapter is an **Object Adapter**.

Sometimes, it's also more convenient to adapt classes instead of instances. In this particular case, the adapter pattern takes the shape of a **Class Adapter**.

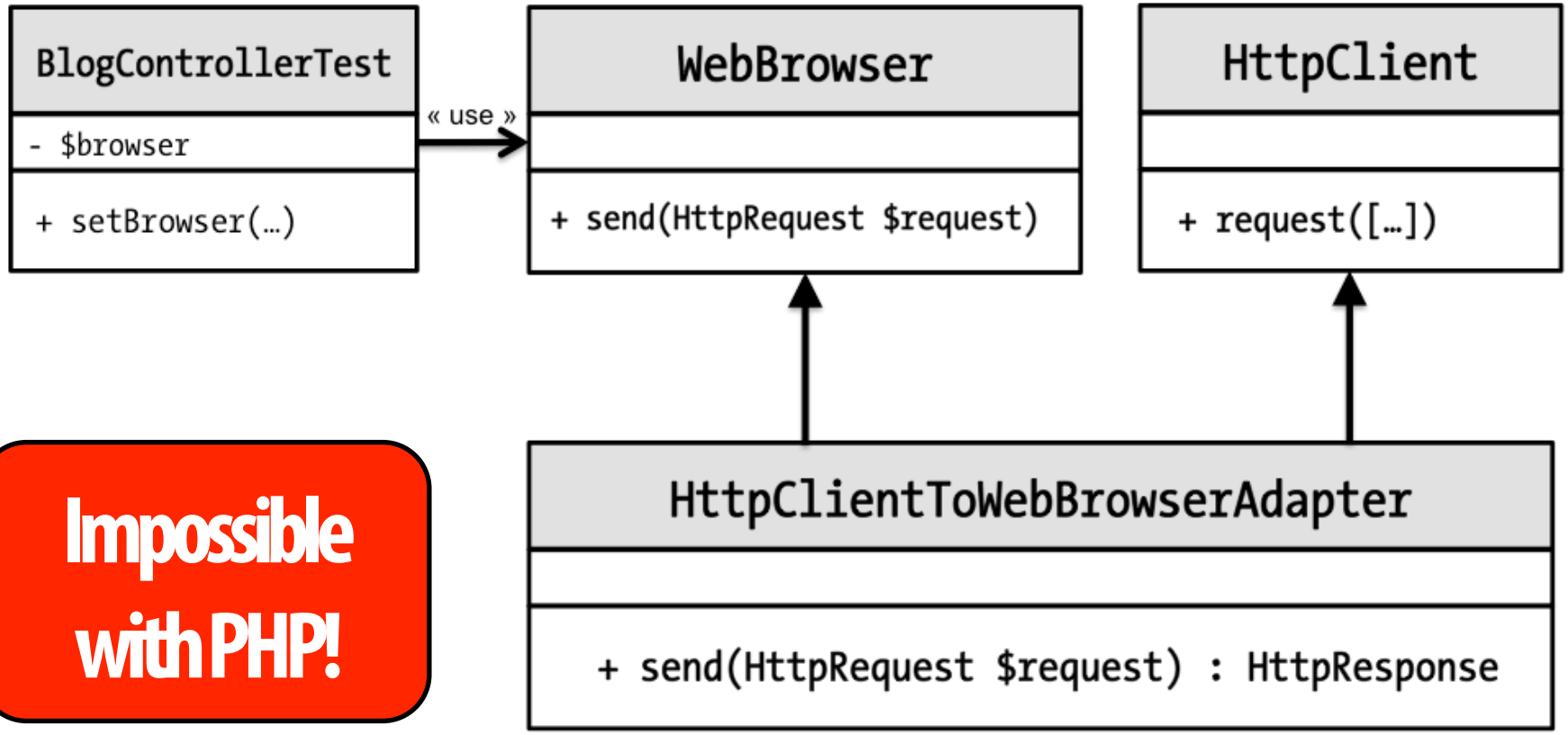
Single Class Inheritance Adapter



Single Class Inheritance Adapter



Multiple Class Inheritance Adapter



Impossible with PHP!

Composite

Composite lets clients treat individual objects and compositions of objects uniformly with a common interface.

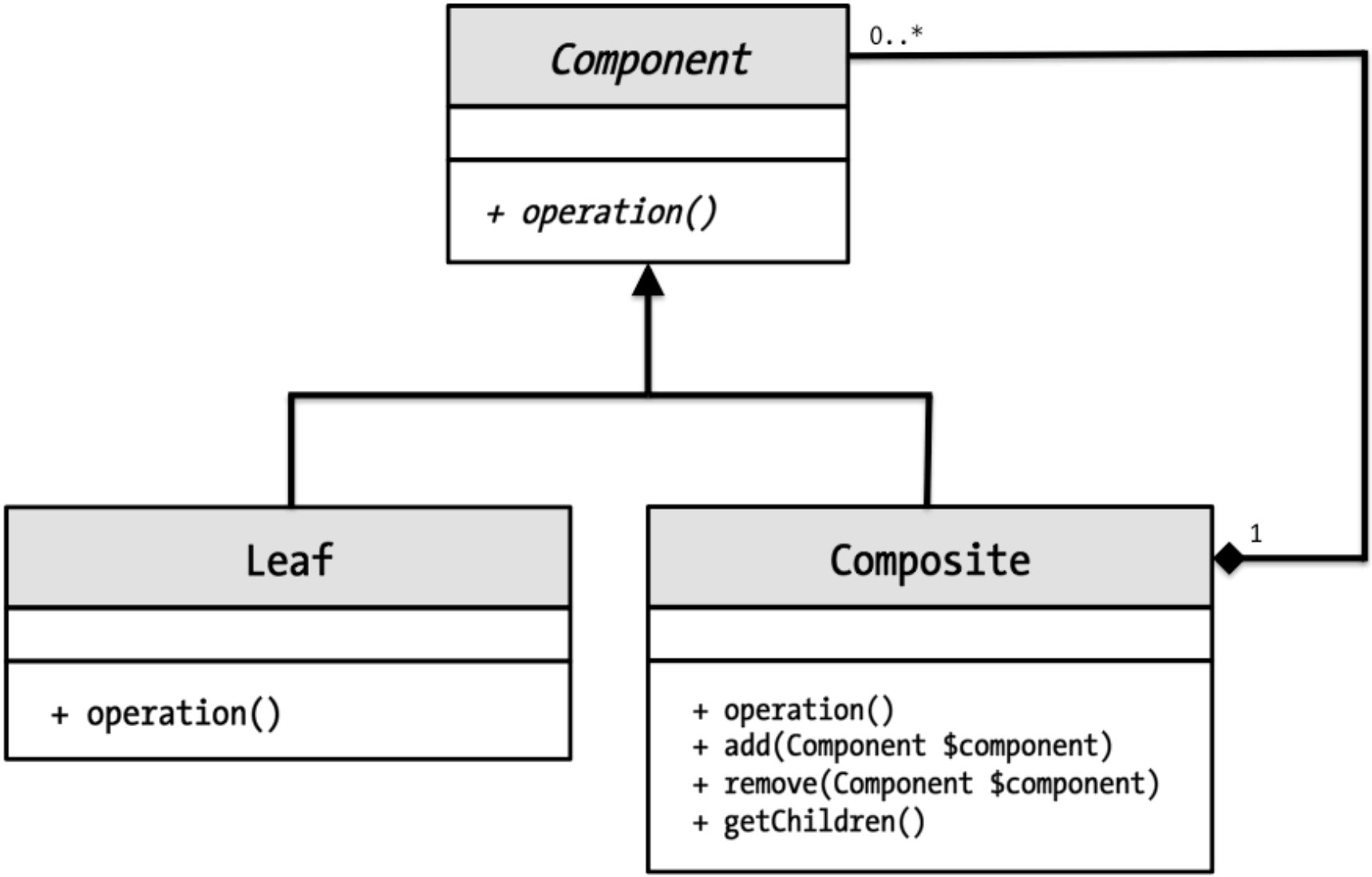
The Composite Design Pattern

The Composite Pattern

- **Structural pattern**

- Treating single objects and collections uniformly.
- Sharing a common unified interface.
- Combining objects easily.
- Enabling recursive operations.

UML Diagram


Component
+ operation()
0..*
1
Leaf
+ operation()
Composite
+ operation()
+ add(Component $component)
+ remove(Component $component)
+ getChildren()

Composite Pattern Usage

```
$nestedComposite = new ConcreteComposite();
$nestedComposite->add(new ConcreteLeaf());
$nestedComposite->add(new ConcreteLeaf());

$composite = new ConcreteComposite();
$composite->add(new ConcreteLeaf());
$composite->add(new ConcreteLeaf());
$composite->add($nestedComposite);
$composite->operation();

$leaf = new ConcreteLeaf();
$leaf->operation();
```

Use Cases for the Composite Pattern

Uses Cases for the Composite Pattern

- File explorer
- Organization chart
- Family tree
- Content management system
- XML file parsing
- Nested web forms
- Navigation bars
- …

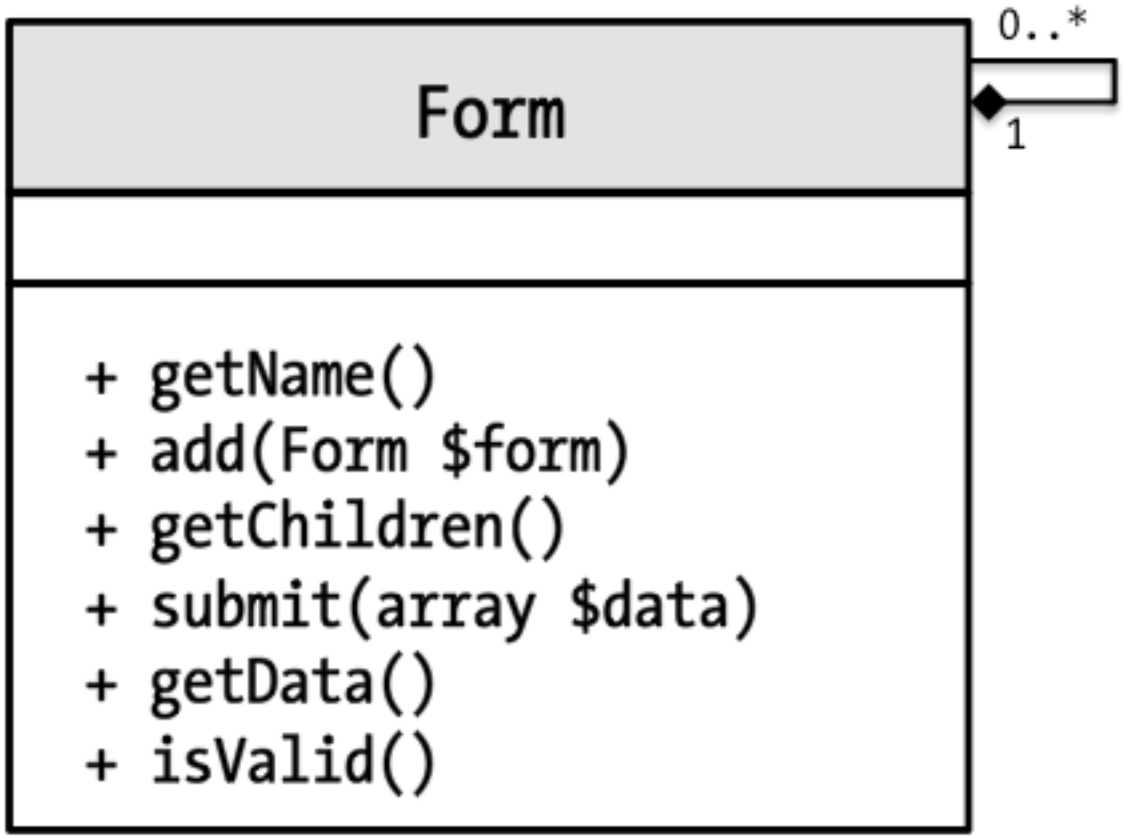
Implementing the Composite Design Pattern

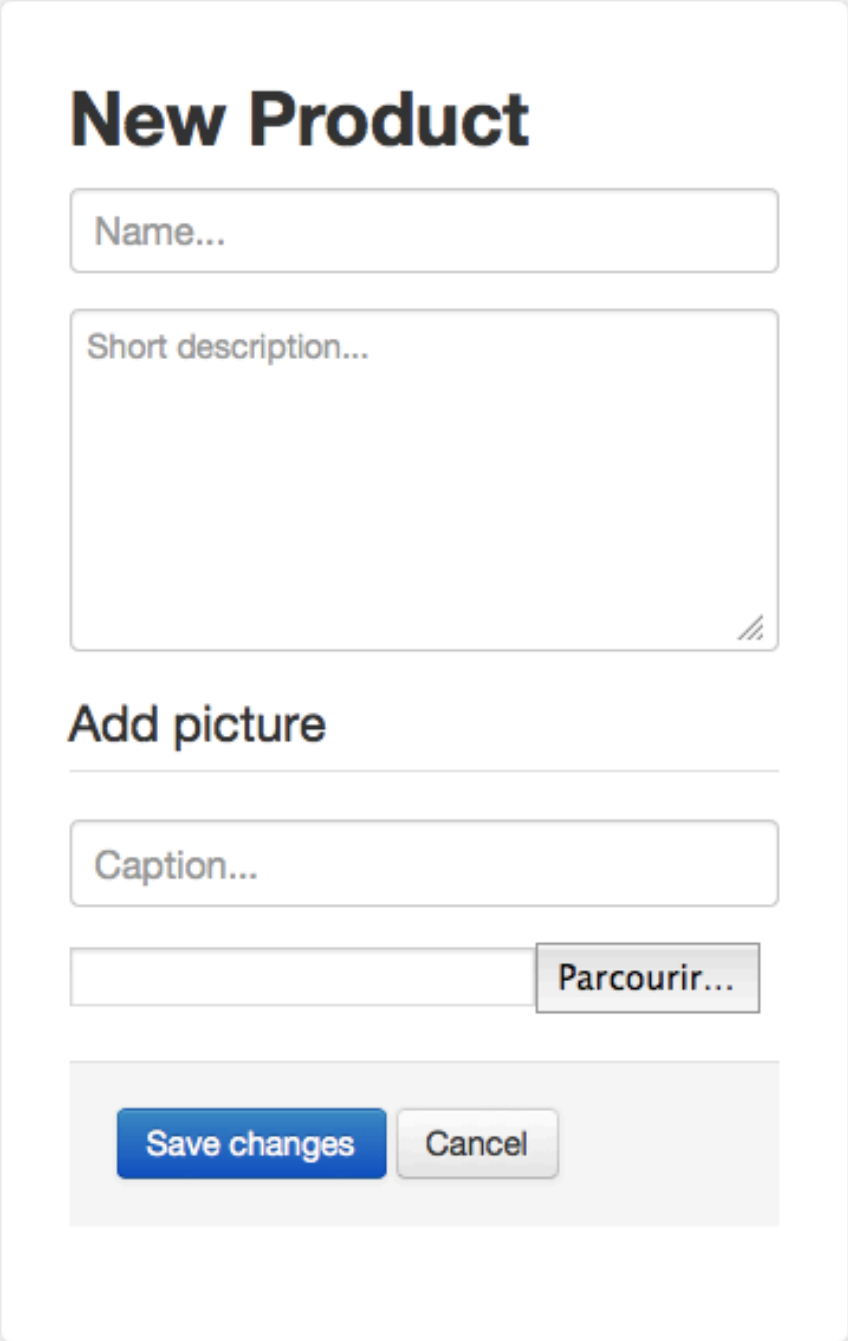
#1. Submitting a complex Form tree.

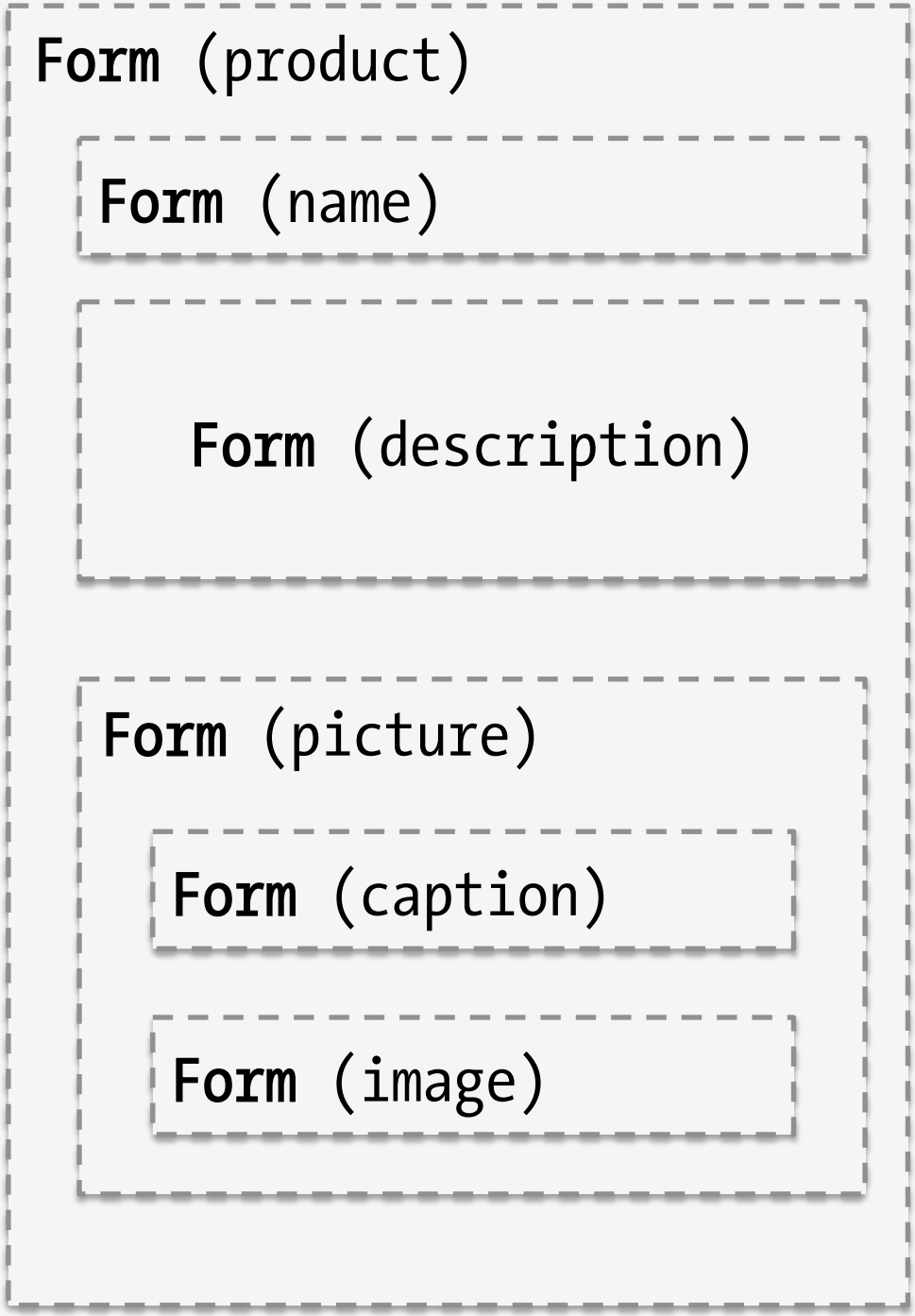
Every component is a Form

Each element that composes a form is an instance of a **Form** class. Thus, each **Form** instance keeps a reference to its parent **Form** object and a collection of its children **Form** references.

Each child **Form** instance can also reference its parent **Form** instance and combines nested **Form** children instances.


Form
0..*
1
+ getName()
+ add(Form $form)
+ getChildren()
+ submit(array $data)
+ getData()
+ isValid()


New Product
Name...
Short description...
Add picture
Caption...
Parcourir...
Save changes
Cancel


Form (product)
Form (name)
Form (description)
Form (picture)
Form (caption)
Form (image)

The Form Class

```
class Form implements FormInterface
{
    private $name;

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

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

The Form Class

```
class Form implements FormInterface
{
    private $parent;
    private $children;

    public function add(FormInterface $child): void
    {
        $this->children[$child->getName()] = $child;
        $child->parent = $this;
    }
}
```

Embedding forms into forms

```
$picture = new Form('picture');
$picture->add(new Form('caption'));
$picture->add(new Form('image'));

$form = new Form('product');
$form->add(new Form('name'));
$form->add(new Form('description'));
$form->add($picture);
```

Submitting & processing the form data

```
$form = new Form('product');
$form->add(new Form('name'));
// ...
$form->submit([
    'name' => 'MacBook Air 11 inches.',
    'description' => 'The thinest laptop',
    'picture' => [
        'caption' => 'The Macbook Air.',
    ],
]);
```

Recursive operation made simple!

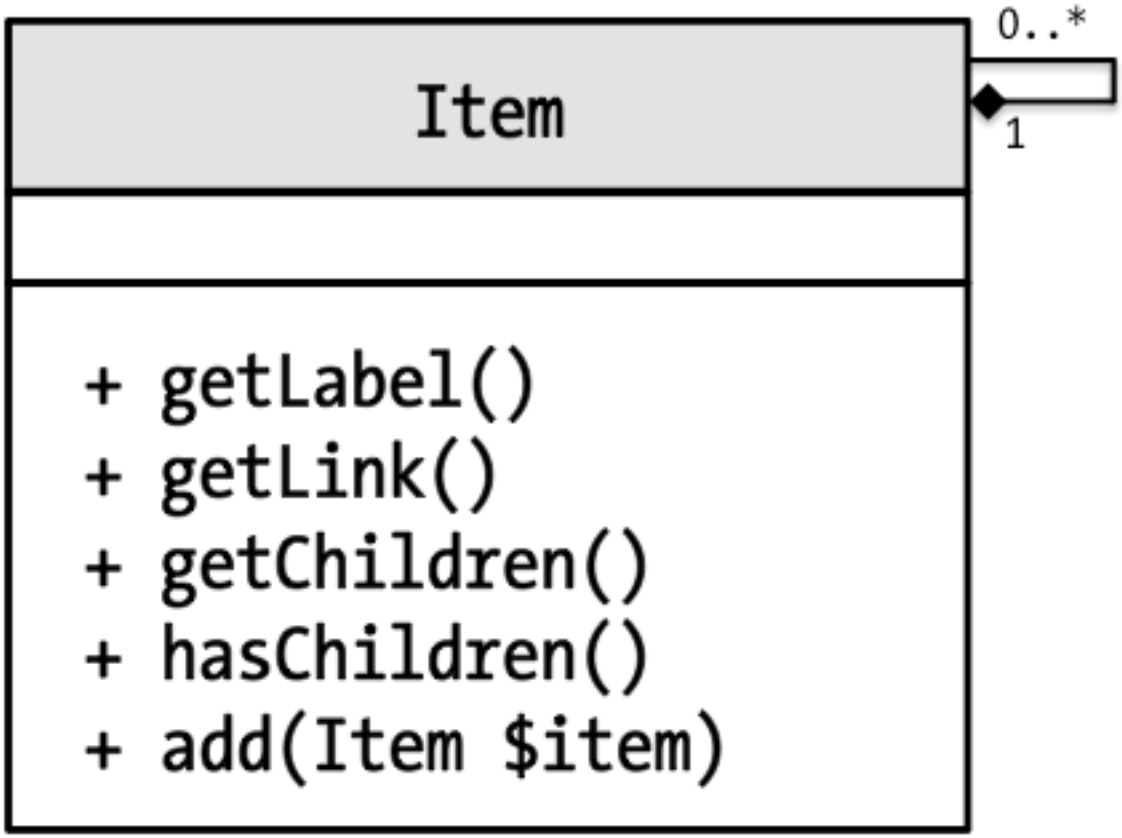
```php
class Form implements FormInterface
{
    // ...
    private $data;

    public function submit(array $data): void
    {
        $this->data = $data;
        foreach ($this->children as $child) {
            if (isset($data[$child->getName()])) {
                $childData = $data[$child->getName()];
                $child->submit($childData);
            }
        }
    }
}
```

#2.
The Navigation Bar.

The World Navigation Tree

We want to be able to modelize the World and render a navigation tree recursively with a minimum of efforts.

- The world is made of five continents.
- Each continent has countries.
- Each country may be split into provinces or states.
- Each country (or province / state) has cities.
- Each city may have districts…


Item
0..*
1
+ getLabel()
+ getLink()
+ getChildren()
+ hasChildren()
+ add(Item $item)

Modelizing the navigation tree

```
interface ItemInterface
{
    public function getLabel(): string;
    public function getLink(): ?string;
    public function getItems(): array;
    public function hasItems(): bool;
}
```

Modelizing the navigation tree

```
class Item implements ItemInterface
{
    private $label;
    private $link;
    private $items;

    public function __construct(
        string $label,
        string $link = null,
        array $items = []
    ) {
        $this->label = $label;
        $this->link = $link;
        $this->items = [];
        $this->addChildren($items);
    }
}
```

```php
class Item implements ItemInterface
{

    public function add(ItemInterface $item): void
    {
        if (!in_array($item, $this->items)) {
            $this->items[] = $item;
        }
    }

    public function getItems(): array
    {
        return $this->items;
    }

    public function hasItems(): bool
    {
        return count($this->items) > 0;
    }
}
```

```php
class Item implements ItemInterface
{
    // ...
    public function getLabel(): string
    {
        return $this->label;
    }

    public function getLink(): ?string
    {
        return $this->link;
    }
}
```

```php
$world = new Item('World', '/', [
    new Item('Africa', '/africa', [
        new Item('Mali', '/africa/mali', [
            new Item('Bamako', '/africa/mali/bamako'),
        ]),
        new Item('Tunisia', '/africa/tunisia', [
            new Item('Hammamet', '/africa/tunisia/tunis'),
            new Item('Sfax', '/africa/tunisia/sfax'),
            new Item('Tunis', '/africa/tunisia/tunis'),
        ]),
        new Item('Arab Emirates', '/africa/arab-emirates', [
            new Item('Abu Dhabi', '/africa/arab-emirates/abu-dhabi'),
            new Item('Dubai', '/africa/arab-emirates/dubai'),
        ]),
    ]),
    // ...
]);
```

```php
$world = new Item('World', '/', [
    new Item('America', '/america', [
        new Item('Canada', '/america/canada', [
            new Item('Quebec', '/america/canada/quebec', [
                new Item('Quebec', '/america/canada/quebec/quebec'),
                new Item('Montreal', '/america/canada/quebec/montreal'),
            ]),
            new Item('Ontario', '/america/canada/ontario', [
                new Item('Ottawa', '/america/canada/ontario/ottawa'),
                new Item('Toronto', '/america/canada/ontario/toronto'),
            ])
        ]),
        new Item('United States', '/america/united-states', [
            new Item('California', '/america/united-states/california', [
                new Item('Los Angeles', '/america/united-states/california/los-angeles'),
                new Item('San Francisco', '/america/united-states/california/san-francisco'),
            ]),
            new Item('Illinois', '/america/united-states/illinois', [
                new Item('Chicago', '/america/united-states/illinois/chicago'),
            ]),
        ]),
    ]),
    new Item('Europe', '/europe', [
        // ...
    ]),
]);
```

Rendering the navigation tree

```
function display_world(Item $item)
{
    $label = $item->getLabel();
    $link = $item->getLink();

    $output = '<ul>';
    $output.= '  <li>';
    $output.= $link ? sprintf('<a href="%s">%s</a>', $link, $label) : $label;
    foreach ($item->getItems() as $child) {
        $output.= display_world($child);
    }
    $output.= '  </li>';
    $output.= '</ul>';

    return $output;
}

if ($world->hasItems()) {
    echo display_world($world);
}
```

Rendering the navigation tree

- World
  - Africa
    - Mali
      - Bamako
    - Tunisia
      - Hammamet
      - Sfax
      - Tunis
    - Arab Emirates
      - Abu Dhabi
      - Dubai
  - America
    - Canada
      - Quebec
        - Quebec
        - Montreal
      - Ontario
        - Ottawa
        - Toronto
    - United States
      - California
        - Los Angeles
        - San Francisco
      - Illinois
        - Detroit
      - Michigan
        - Detroit
      - Nevada
        - Las Vegas
  - Asia
    - Indonesia

Hello World!

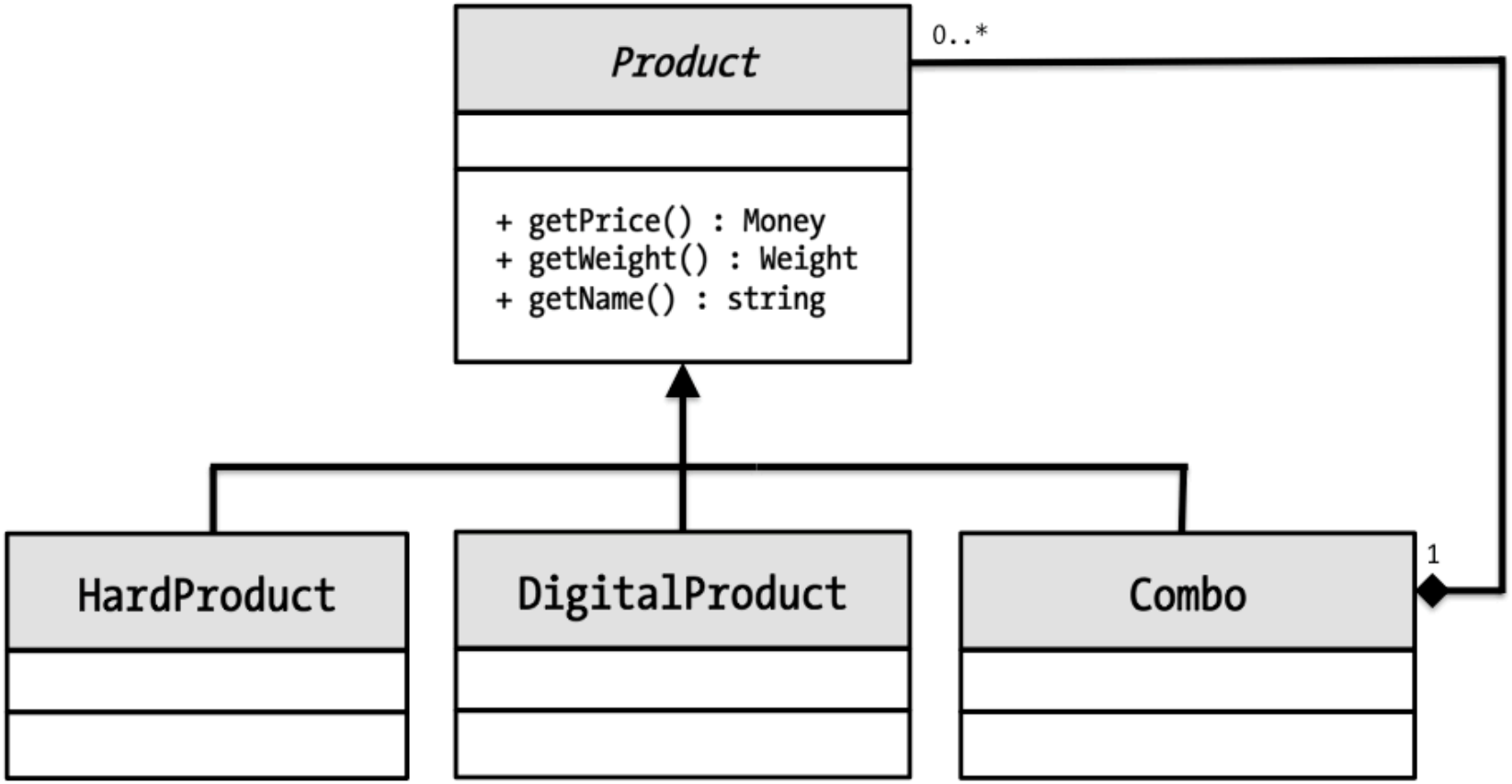
Norwegian Sea
Sverige
Sweden
Norge
Norway
Suomi
Finland
Россия
Russia
Danmark
Denmark
United
Kingdom
Беларусь
Belarus
Polska
Poland
Deutschland
Germany
Україна
Ukraine
Österreich
Austria
France
România
Romania
Italia
Italy
España
Spain
Ελλάς
Greece
Türkiye
Turkey
Қазақстан
Kazakhstan
Монгол Улс
Mongolia
O'zbekiston
Uzbekistan
Киргизия
Kyrgyzstan
Sea of Japan
中国
China
대한민국
South Korea
日本
Japan
East China Sea
تونس
Tunisia
المغرب
Morocco
الجزائر
Algeria
ليبيا
Libya
مصر
Egypt
العراق
Iraq
ایران
Iran
پاکستان
Pakistan
नेपाल
Nepal
India
السعودية
Saudi Arabia
عمان
Oman
Mali
Niger
تشاد
Tchad
Chad
السودان
Sudan
اليمن
Yemen
Gulf of Aden
Arabian Sea
Bay of Bengal
ประเทศไทย
Thailand
Việt Nam
Vietnam
Pilipinas
Philippines
Burkina
Faso

#3.
Dealing with combo products in a commerce application.

Selling single products & combos

- Each product can be a physical, digital or a **combo**.
- Digital products don't have a mass.
- Each product has a unit price.

For combo products, their price can be fixed or it can be the sum of the prices of the combined products. The mass of a combo is always the sum of the massess of its combined products.


Product
+ getPrice() : Money
+ getWeight() : Weight
+ getName() : string
0..*
1
HardProduct
DigitalProduct
Combo

Modelizing the products attributes

```
interface ProductInterface
{
    public function getName(): string;

    public function getPrice(): Money;

    public function getMass(): Mass;
}
```

```php
abstract class Product implements ProductInterface
{
    protected $name;
    protected $price;
    protected $mass;

    function __construct(string $name, Money $price, Mass $mass)
    {
        $this->name = $name;
        $this->price = $price;
        $this->mass = $mass;
    }

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

```php
abstract class Product implements ProductInterface
{
    // ...

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

    public function getMass(): Mass
    {
        return $this->mass;
    }
}
```

```php
class Combo extends Product
{
}

class HardProduct extends Product
{
}

class DigitalProduct extends Product
{
    function __construct(string $name, Money $price)
    {
        parent::__construct($name, $price, new Mass(0));
    }
}
```

```php
$paperBook = new HardProduct(
    'PHP 5 Design Patterns',
    new EUR(4900),
    new Mass(960)
);

$digitalBook = new DigitalProduct(
    'PHP 5 Design Patterns',
    new EUR(2200)
);
```

```php
class Combo extends Product
{
    private $products;

    function __construct(string $name, array $products, Money $price = null)
    {
        $this->setProducts($products);

        parent::__construct(
            $name,
            $price ?: $this->totalPrice(),
            $this->totalMass()
        );
    }

    // ...

    private function add(ProductInterface $product): void
    {
        $this->products[] = $product;
    }
}
```

```php
class Combo extends Product
{
    // ...

    private function setProducts(array $products): void
    {
        if (count($products) < 2) {
            throw new \LogicException('At least 2...');
        }

        foreach ($products as $product) {
            $this->add($product);
        }
    }
}
```

```php
class Combo extends Product
{
    // ...
    private function totalPrice(): Money
    {
        $total = $this->getPriceAt(0);
        for ($i = 1; $i < count($this->products); $i++) {
            $total = $total->add($this->getPriceAt($i));
        }

        return $total;
    }

    private function getPriceAt(int $index): Money
    {
        return $this->get($index)->getPrice();
    }

    private function get(int $index): ProductInterface
    {
        return $this->products[$index];
    }
}
```

Get the total mass of a combo

```php
class Combo extends Product
{
    // ...
    private function totalMass(): Mass
    {
        $total = $this->getMassAt(0);
        for ($i = 1; $i < count($this->products); $i++) {
            $total = $total->add($this->getMassAt($i));
        }

        return $total;
    }

    private function getMassAt(int $index): Mass
    {
        return $this->get($index)->getMass();
    }
}
```

The Combo with a fixed price

```
$products = [
  new HardProduct('Digital Camera', new EUR(78900), new Mass(855)),
  new HardProduct('Camera Bag', new EUR(3900), new Mass(220)),
  new HardProduct('Memory Card 128 Gb', new EUR(7900), new Mass(42)),
];

$combo = new Combo('Digital Camera & Bag', $products, new EUR(83900));

echo 'Name:   ', $combo->getName() ,"\n";
echo 'Mass:   ', $combo->getMass()->getValue() ,"\n";
echo 'Price:  ', $combo->getPrice()->getAmount() ,"\n";
```

```
Name: Digital Camera & Bag
Mass: 1117
Price: 83900
```

The Combo with a dynamic price

```
$products = [
  new HardProduct('Digital Camera', new EUR(78900), new Mass(855)),
  new HardProduct('Camera Bag', new EUR(3900), new Mass(220)),
  new HardProduct('Memory Card 128 Gb', new EUR(7900), new Mass(42)),
];

$combo = new Combo('Digital Camera & Bag', $products);

echo 'Name:   ', $combo->getName() ,"\n";
echo 'Mass:   ', $combo->getMass()->getValue() ,"\n";
echo 'Price:  ', $combo->getPrice()->getAmount() ,"\n";
```

```
Name: Digital Camera & Bag
Mass: 1117
Price: 90700
```

The Supper Dupper Combo!

```
$products = [
  new HardProduct('Digital Camera', new EUR(78900), new Mass(855)),
  new HardProduct('Camera Bag', new EUR(3900), new Mass(220)),
  new HardProduct('Memory Card 128 Gb', new EUR(7900), new Mass(42)),
];

$combo = new Combo('Digital Camera Combo Pack + Tripod', [
  new HardProduct('Lightweight Tripod', new EUR(2690), new Mass(570)),
  new Combo('Digital Camera & Bag', $products, new EUR(83900)),
]);

echo 'Name:   ', $combo->getName() ,"\n";
echo 'Mass:   ', $combo->getMass()->getValue() ,"\n";
echo 'Price:  ', $combo->getPrice()->getAmount() ,"\n";
```

```
Name: Digital Camera & Bag
Mass: 1117
Price: 90700
```

Decorator

Adding new responsabilities to an object without changing its class.

Composition over Inheritance

Modelizing a multifunctions pen

- Write in black
- Write in blue
- Write in italic
- Write in bold
- Write in reverse way
- Write in upper case
- Write in lower case
- Underline the text
- Ban bad words
- …

Modelizing a multifunctions pen

```
interface WriterInterface
{
    /**
     * Returns the formatted text.
     *
     * @param  string $text
     * @return string
     */
    public function write($text);
}
```

Modelizing a multifunctions pen

```
class Pen implements WriterInterface
{
    public function write($text)
    {
        return $text;
    }
}

$pen = new Pen();
echo $pen->write('Hello');
echo $pen->write(' ');
echo $pen->write('World !');
```

```php
class Pen implements WriterInterface
{
    public $bold = false;
    public $italic = false;

    public function write($text)
    {
        if ($this->bold) {
            $text = '<strong>'.$text.'</strong>';
        }

        if ($this->italic) {
            $text = '<em>'.$text.'</em>';
        }

        return $text;
    }
}
```

Modelizing a multifunctions pen

```php
$pen = new Pen();
$pen->bold = true;
$pen->italic = true;

echo $pen->write('Hello World !');
```

```
<em><strong>Hello World !</strong></em>
```

What's wrong with this design?

- Unable to extend the pen without changing the class.
- Unable to combine new features (bold + italic + blue).
- Mixing multiple responsabilities (bold & italic).
- Unable to change the order of formatting.
- Break SOLID principle!

What about using inheritance?

- `Pen`
- `BoldPen`
- `ItalicPen`
- `UnderlinePen`
- `UpperCasePen`
- `LowerCasePen`
- `ColorizedTextPen`
- …

What about using inheritance?

Inheritance doesn't work because it doesn't allow to combine pen's features at will. Each new formatting combination will lead to creating a new class:

- `ItalicBoldPen`
- `BoldItalicPen`
- `BoldItalicUnderlineBlueUpperCasePen`
- `ItalicRedBoldLowerCaseUnderlineRemoveInsultsPen`
- …

Use Cases for the Decorator Pattern

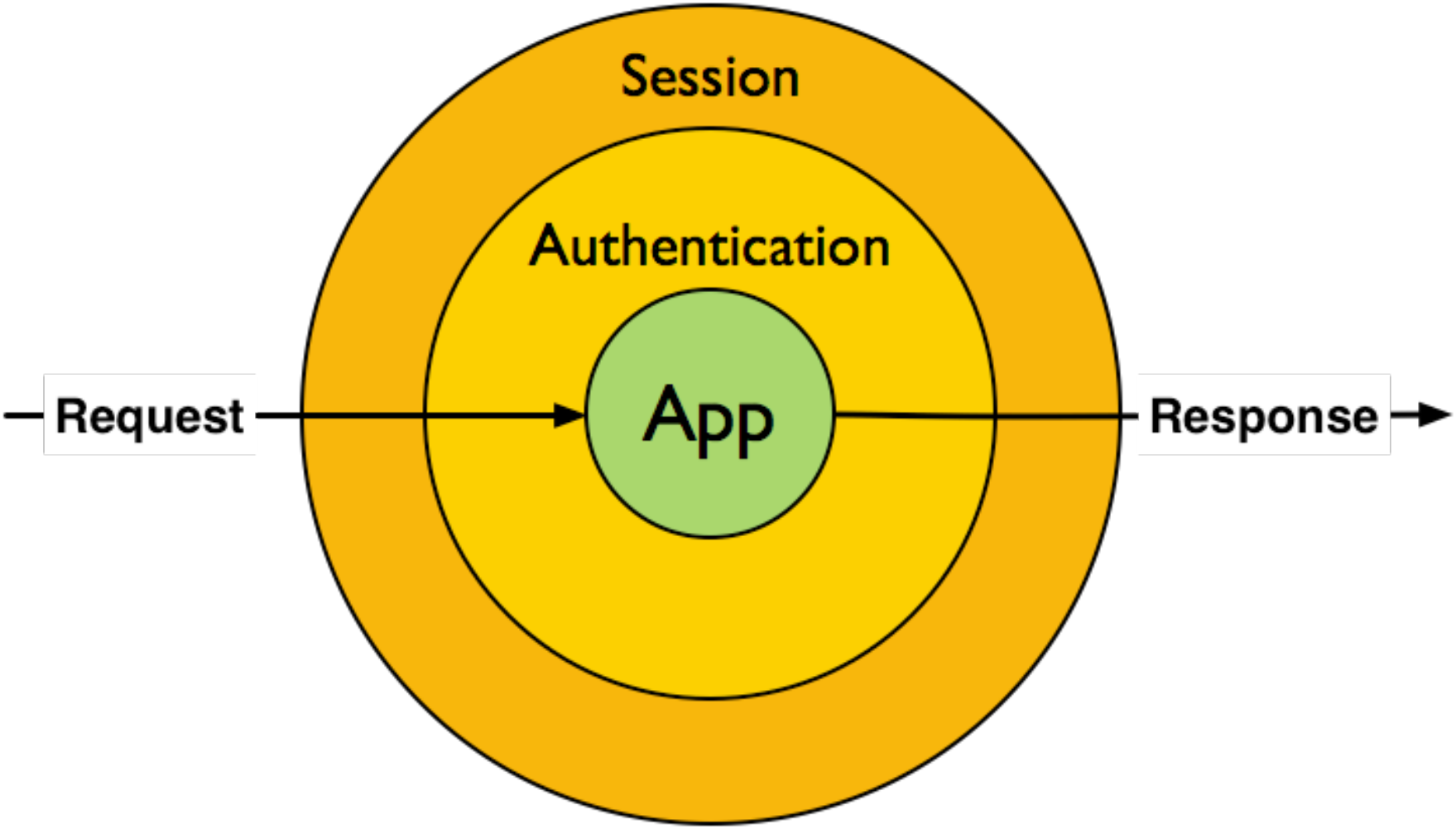
Logging

```
$handler = new Monolog\Handler\StreamHandler('/tmp/app.log');
$logger = new Monolog\Logger('app');
$logger->pushHandler($handler);

$ftp = new FTP\Client(\Net_SFTP('ftp.domain.tld'), 'user', 'secret');
$ftp = new FTP\VerboseClient($ftp, $logger);

$ftp->upload('/home/doc.pdf', '/var/data/doc.pdf');
$ftp->download('/var/data/summary.pdf', '/home/summary.pdf');
```

HTTP Caching

```
$kernel = new AppKernel('prod', false);
$kernel->loadClassCache();

$kernel = new AppCache($kernel);

$request = Request::createFromGlobals();
$response = $kernel->handle($request);
$response->send();
```

HTTP Middlewares



http://stackphp.com/

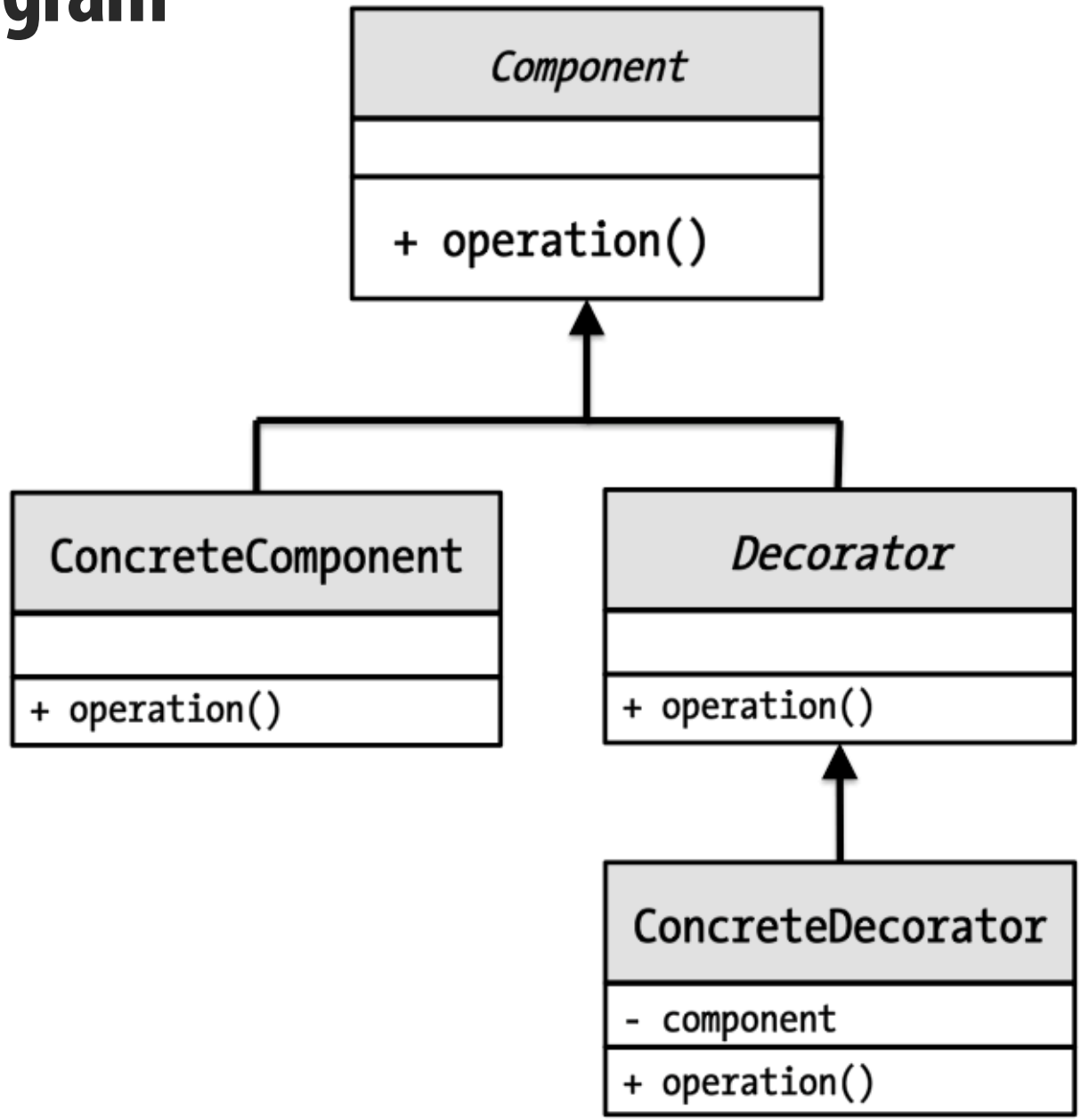
The Decorator Design Pattern

The Decorator Pattern

- **Structural pattern**

- Add new responsabilities to an object
- Don’t break or change the existing API
- Use composition over inheritance
- Allow infinite responsabilities combinations
- Respect SOLID principle

UML Diagram


Component
+ operation()
ConcreteComponent
+ operation()
Decorator
+ operation()
ConcreteDecorator
- component
+ operation()

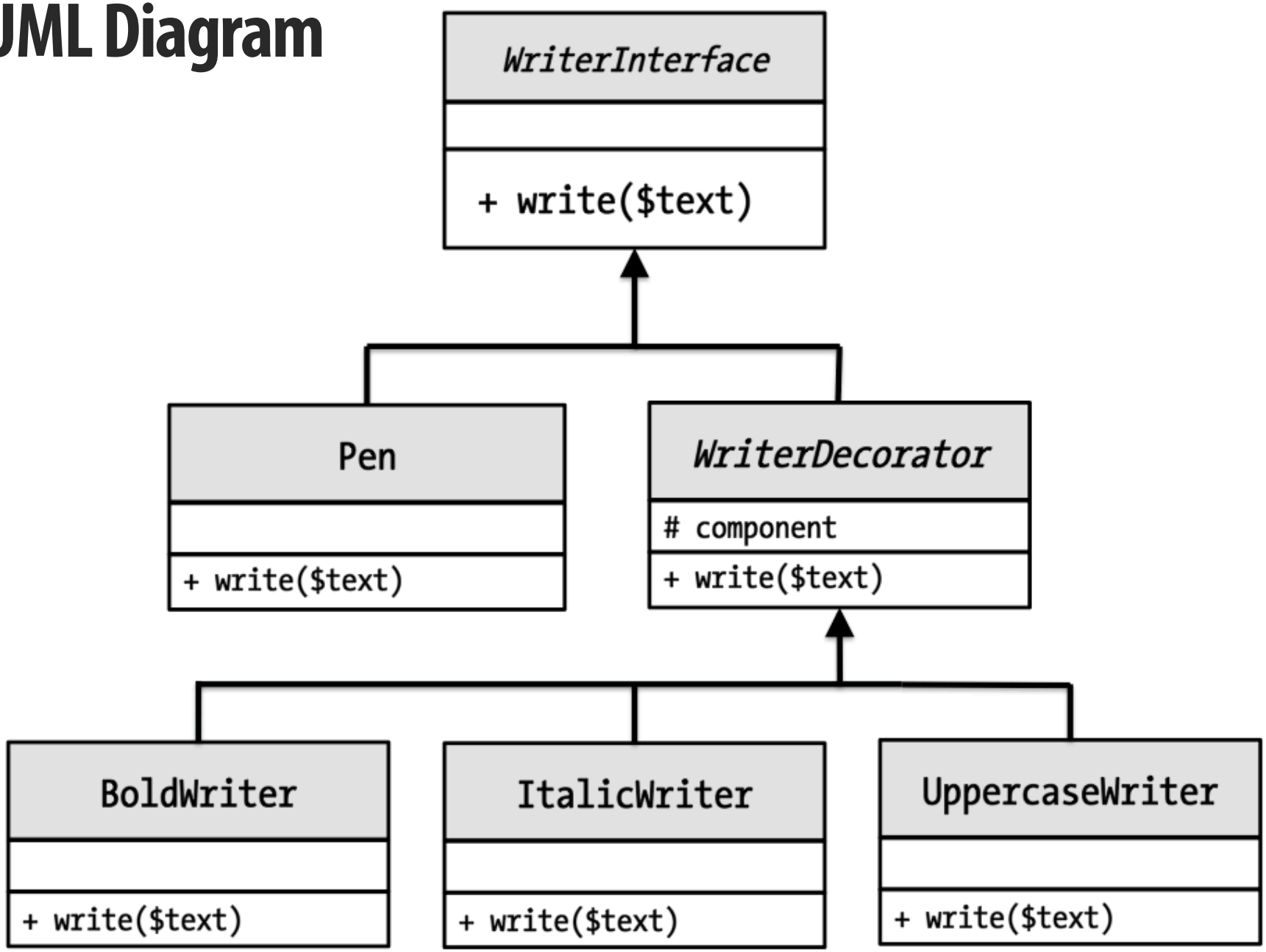
Decorator Pattern Usage

```
$instance = new OtherConcreteDecorator(
    new SomeConcreteDecorator(
        new ConcreteDecorator(
            new DecoratedComponent()
        )
    )
);

$instance->operation();
```

Implementing the Decorator Design Pattern

#1. Multifunction Pen

UML Diagram


WriterInterface
+ write($text)
Pen
+ write($text)
WriterDecorator
component
+ write($text)
BoldWriter
+ write($text)
ItalicWriter
+ write($text)
UppercaseWriter
+ write($text)

The abstract Decorator class

The abstract **Decorator** holds a reference to a decorated **Component** or **Decorator** object.

```
abstract class WriterDecorator implements WriterInterface
{
    protected $writer;

    public function __construct(WriterInterface $writer)
    {
        $this->writer = $writer;
    }
}
```

The BoldWriter Decorator Class

```
class BoldWriter extends WriterDecorator
{
    public function write($text)
    {
        return sprintf(
            '<strong>%s</strong>',
            $this->writer->write($text)
        );
    }
}
```

The ItalicWriter Decorator Class

```
class ItalicWriter extends WriterDecorator
{
    public function write($text)
    {
        return sprintf(
            '<em>%s</em>',
            $this->writer->write($text)
        );
    }
}
```

The UppercaseWriter Decorator Class

```
class UppercaseWriter extends WriterDecorator
{
    public function write($text)
    {
        return strtoupper(
            $this->writer->write($text)
        );
    }
}
```

The UppercaseWriter Decorator Class

```php
$pen = new ItalicWriter(new BoldWriter(new UppercaseWriter(new Pen())));
echo $pen->write('Hello World!');

$pen = new BoldWriter(new ItalicWriter(new UppercaseWriter(new Pen())));
echo $pen->write('Hello World!');

$pen = new UppercaseWriter(new ItalicWriter(new BoldWriter(new Pen())));
echo $pen->write('Hello World!');
```

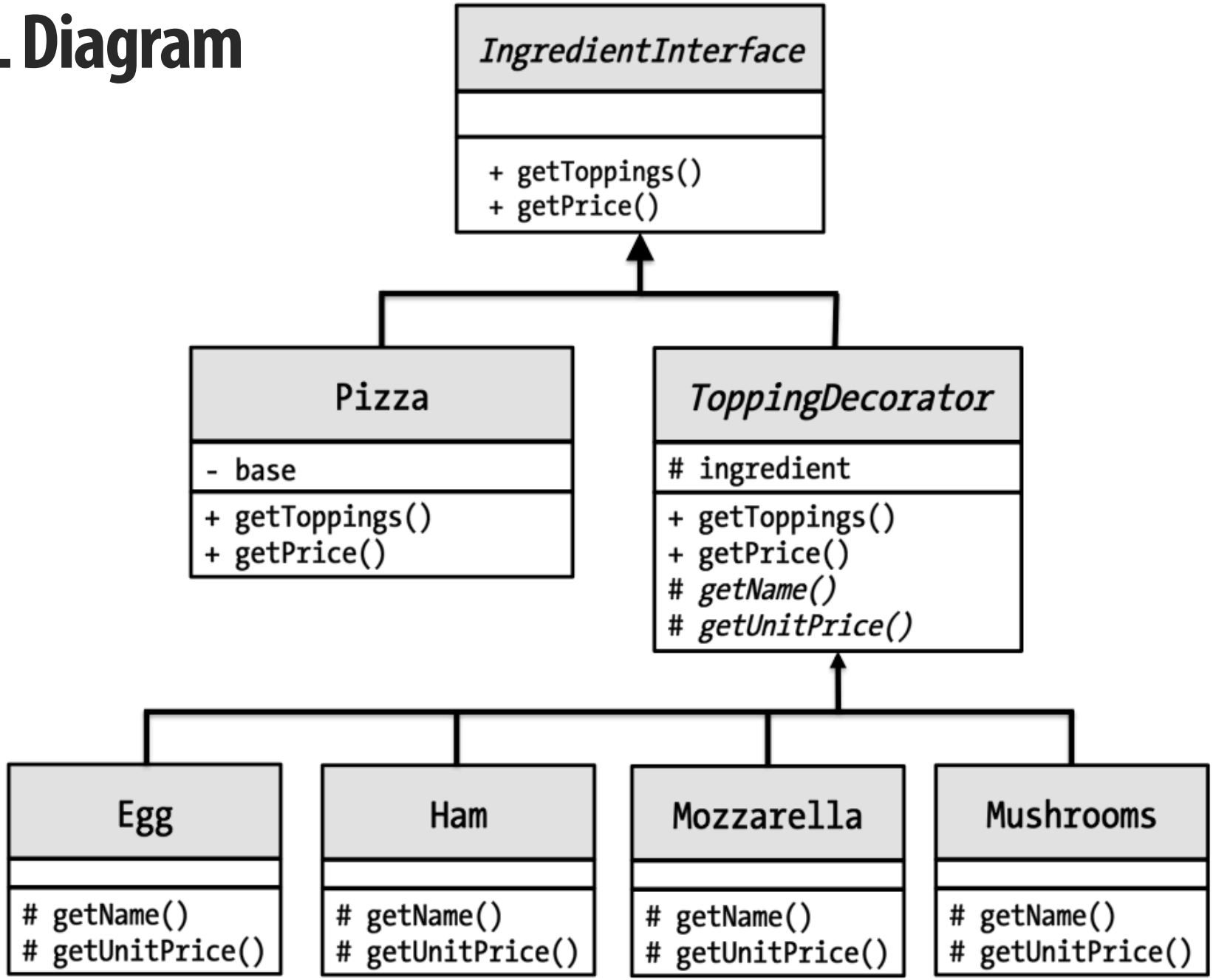
```
<em><strong>HELLO WORLD!</strong></em>
<strong><em>HELLO WORLD!</em></strong>
<EM><STRONG>HELLO WORLD!</STRONG></EM>
```

#2.
Pizza

The Pizza Example

- We want to compose any pizza preparation.
- We want to know how much a pizza.
- We want to create new toppings at will.
- We want to create custom pizzas.

UML Diagram


IngredientInterface
+ getToppings()
+ getPrice()
Pizza
- base
+ getToppings()
+ getPrice()
ToppingDecorator
ingredient
+ getToppings()
+ getPrice()
# getName()
# getUnitPrice()
Egg
getName()
getUnitPrice()
Ham
getName()
getUnitPrice()
Mozzarella
getName()
getUnitPrice()
Mushrooms
getName()
getUnitPrice()

The Ingredient Interface

```
interface IngredientInterface
{
    /**
     * Returns the list of toppings on the pizza.
     *
     * @return array
     */
    public function getToppings();

    /**
     * Returns the price of the ingredient.
     *
     * @return Money
     */
    public function getPrice();
}
```

The Pizza Class

```
class Pizza implements IngredientInterface
{
    const TOMATO = 'tomato';
    const CREAM  = 'cream';

    private $base;

    public function __construct($base)
    {
        $this->base = $base;
    }
}
```

The Pizza Class

```
class Pizza implements IngredientInterface
{
    // …
    public function getToppings()
    {
        return [ $this->base ];
    }

    public function getPrice()
    {
        return new Money(200, 'EUR');
    }
}
```

The Abstract ToppingDecorator Class

```
abstract class ToppingDecorator implements IngredientInterface
{
    protected $ingredient;
    private $price;

    function __construct(IngredientInterface $ingredient, Money $price)
    {
        $this->ingredient = $ingredient;
        $this->price = $price;
    }

    abstract protected function getName();
}
```

```php
abstract class ToppingDecorator implements IngredientInterface
{
    protected static function createPrice($price)
    {
        return new EUR((int) $price);
    }

    public function getPrice()
    {
        return $this->price->add($this->ingredient->getPrice());
    }

    public function getToppings()
    {
        $toppings   = $this->ingredient->getToppings();
        $toppings[] = $this->getName();

        return $toppings;
    }
}
```

The Concrete Egg Topping Class

```
class Egg extends ToppingDecorator
{
    function __construct(IngredientInterface $ingdt)
    {
        parent::__construct($ingdt, self::createPrice(60));
    }

    protected function getName()
    {
        return 'egg';
    }
}
```

The Concrete Ham Topping Class

```
class Ham extends ToppingDecorator
{
    function __construct(IngredientInterface $ingdt)
    {
        parent::__construct($ingdt, self::createPrice(120));
    }

    protected function getName()
    {
        return 'ham';
    }
}
```

The Concrete Mozzarella Topping Class

```
class Mozzarella extends ToppingDecorator
{
    function __construct(IngredientInterface $ingdt)
    {
        parent::__construct($ingdt, self::createPrice(70));
    }

    protected function getName()
    {
        return 'mozzarella';
    }
}
```

The Concrete Mushrooms Topping Class

```
class Mushrooms extends ToppingDecorator
{
    function __construct(IngredientInterface $ingdt)
    {
        parent::__construct($ingdt, self::createPrice(45));
    }

    protected function getName()
    {
        return 'mushrooms';
    }
}
```

Baking Pizzas!

```
$pizza = new Pizza(Pizza::TOMATO);
$pizza = new Egg(new Egg(new Mushrooms(new Mozzarella(new Ham($pizza)))));

$formatter = new IntlFormatter('fr_FR');
echo sprintf("Toppings: %s.\n", implode(', ', $pizza->getToppings()));
echo sprintf("Price: %s\n", $formatter->format($pizza->getPrice()));
```

```
Toppings: tomato, ham,
mozzarella, mushrooms, egg, egg.
Price: 5,55 €
```

#3.
Coupon Codes

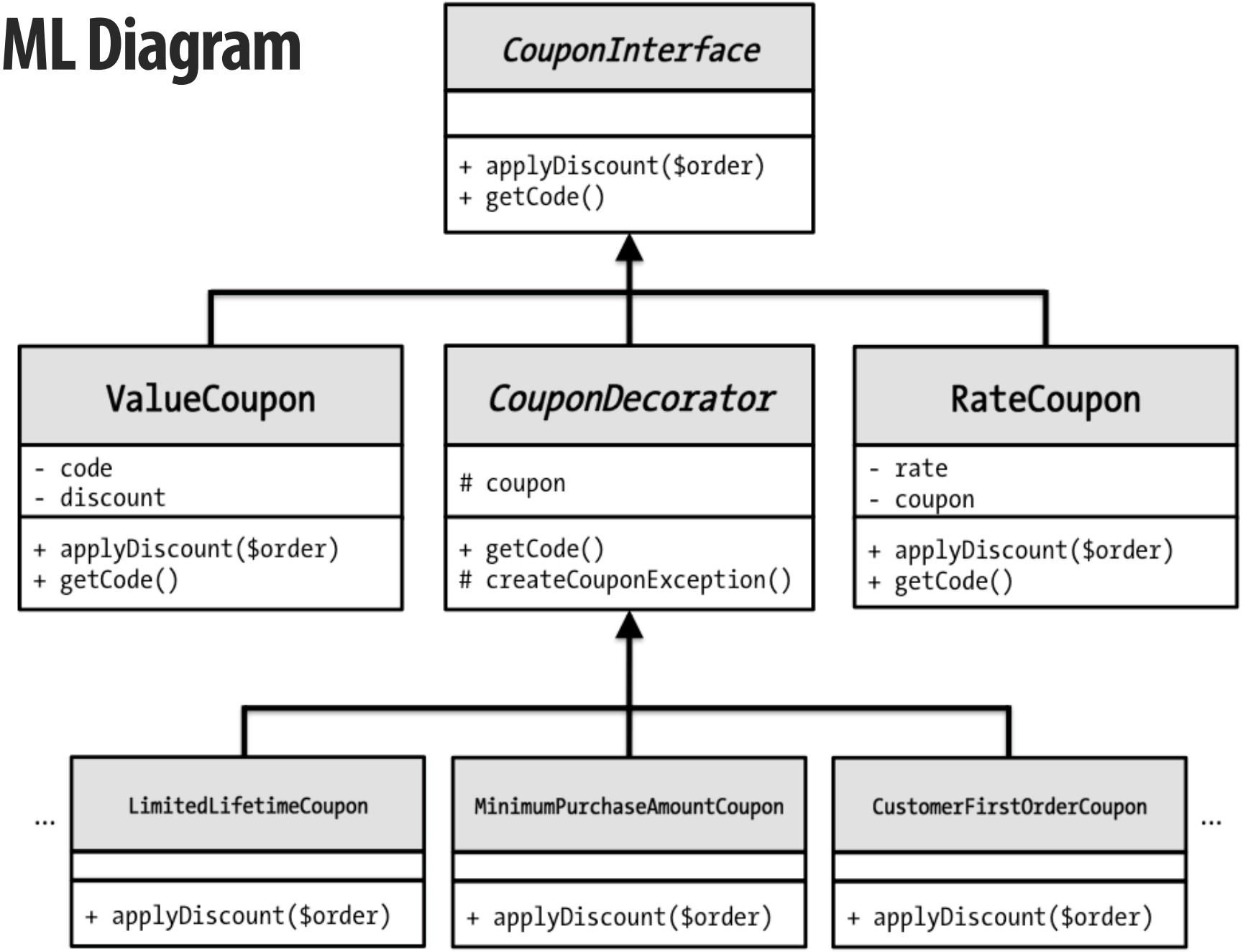
The Coupon System Example

- We want to offer discount codes.
- Each discount can be value (-5 EUR) or rate (-10%) based.
- Coupon codes must be combinable.
- Each coupon is applicable upon certain conditions.
- There must be no limits inventing new coupon codes.
- Coupons reduce an Order's total amount.

Coupon codes restrictions

- Valid for a specific period of time,
- Minimum total amount required,
- Minimum quantity of ordered items required,
- Valid for a specific geographical area,
- Customer must own the loyalty membership card,
- Coupon is valid for Premium / VIP customers only,
- Valid for the customer's very first order,
- Some products are not eligible for discounts,
- Valid only on some specific products areas (luxury, food…),
- etc.

UML Diagram


CouponInterface
+ applyDiscount($order)
+ getCode()
ValueCoupon
- code
- discount
+ applyDiscount($order)
+ getCode()
CouponDecorator
coupon
+ getCode()
# createCouponException()
RateCoupon
- rate
- coupon
+ applyDiscount($order)
+ getCode()
...
LimitedLifetimeCoupon
+ applyDiscount($order)
MinimumPurchaseAmountCoupon
+ applyDiscount($order)
CustomerFirstOrderCoupon
+ applyDiscount($order)
...

The CouponInterface interface

```
interface CouponInterface
{
    public function getCode();

    /**
     * Returns the new total amount after the coupon has been
     * applied on the given order.
     *
     * @param  OrderableInterface $order The order to discount
     * @return Money The new order total amount
     * @throws CouponException When coupon is not applicable
     */
    public function applyDiscount(OrderableInterface $order);
}
```

The ValueCoupon class

```
class ValueCoupon implements CouponInterface
{
    private $code;
    private $discount;

    public function __construct($code, Money $discount)
    {
        $this->code = $code;
        $this->discount = $discount;
    }

    public function getCode()
    {
        return $this->code;
    }

    public function applyDiscount(OrderableInterface $order)
    {
        return $order->getTotalAmount()->subtract($this->discount);
    }
}
```

The RateCoupon class

```
class RateCoupon implements CouponInterface
{
    private $code;
    private $rate;

    public function __construct($code, $rate)
    {
        if (!is_float($rate)) {
            throw new \InvalidArgumentException('$rate must be a float.');
        }

        if ($rate <= 0 || $rate > 1){
            throw new \InvalidArgumentException('$rate must be in ]0, 1].');
        }

        $this->code = $code;
        $this->rate = $rate;
    }
}
```

The RateCoupon class

```
class RateCoupon implements CouponInterface
{
    // ...

    public function getCode()
    {
        return $this->code;
    }

    public function applyDiscount(OrderableInterface $order)
    {
        $amount = $order->getTotalAmount();

        return $amount->subtract($amount->multiply($this->rate));
    }
}
```

The AbstractCouponDecorator class

```
abstract class CouponDecorator implements CouponInterface
{
    protected $coupon;

    public function __construct(CouponInterface $coupon)
    {
        $this->coupon = $coupon;
    }

    public function getCode()
    {
        return $this->coupon->getCode();
    }

    protected function createCouponException($message, \Exception $previous = null)
    {
        return new CouponException($message, 0, $previous);
    }
}
```

The LimitedLifetimeCoupon class

```
class LimitedLifetimeCoupon extends CouponDecorator
{
    private $startAt;
    private $expiresAt;

    public function __construct(CouponInterface $coupon, $startAt, $expiresAt)
    {
        if (!$startAt instanceof \DateTime) {
            $startAt = new \DateTime($startAt);
        }

        if (!$expiresAt instanceof \DateTime) {
            $expiresAt = new \DateTime($expiresAt);
        }

        if ($startAt > $expiresAt) {
            throw new \InvalidArgumentException('$startAt cannot be greater than $expiresAt.');
        }

        parent::__construct($coupon);

        $this->startAt = $startAt;
        $this->expiresAt = $expiresAt;
    }
}
```

The LimitedLifetimeCoupon class

```
class LimitedLifetimeCoupon extends CouponDecorator
{
    public function applyDiscount(OrderableInterface $order)
    {
        $now = new \DateTime('now');

        if ($this->startAt > $now) {
            throw $this->createCouponException(sprintf(
                'Coupon is usable from %s.',
                $this->startAt->format('Y-m-d H:i:s')
            ));
        }

        if ($now > $this->expiresAt) {
            throw $this->createCouponException(sprintf(
                'Coupon was valid until %s.',
                $this->expiresAt->format('Y-m-d H:i:s')
            ));
        }

        return $this->coupon->applyDiscount($order);
    }
}
```

The LimitedLifetimeCoupon class

```
// Coupon offers a discount value of 20 €.
$coupon1 = new LimitedLifetimeCoupon(
    new ValueCoupon('3s2h7pd65s', new EUR(2000)),
    '2014-05-01',
    '2014-08-31'
);

// Coupon offers 25% off.
$coupon2 = new LimitedLifetimeCoupon(
    new RateCoupon('76cqa6qr19', 0.25),
    '2014-05-01',
    '2014-08-31'
);
```

The MinimumPurchaseAmountCoupon class

```
class MinimumPurchaseAmountCoupon extends CouponDecorator
{
    private $minimumAmount;

    function __construct(CouponInterface $coupon, Money $minAmount)
    {
        parent::__construct($coupon);
        $this->minimumAmount = $minAmount;
    }

    public function applyDiscount(OrderableInterface $order)
    {
        $amount = $order->getTotalAmount();
        if ($amount->lessThan($this->minimumAmount)) {
            throw $this->createCouponException(sprintf(
                'Coupon requires a minimum amount of %u.',
                $this->minimumAmount->getAmount()
            ));
        }

        return $this->coupon->applyDiscount($order);
    }

}
```

The MinimumPurchaseAmountCoupon class usage

```
// Get 20 € off if total amount is greater than 300 €
$coupon1 = new MinimumPurchaseAmountCoupon(
    new ValueCoupon('3s2h7pd65s', new EUR(2000)),
    new EUR(30000)
);

// Get 25% off if total amount is greater than 300 €
$coupon2 = new MinimumPurchaseAmountCoupon(
    new RateCoupon('76cqa6qr19', 0.25),
    new EUR(30000)
);
```

The CustomerFirstOrderCoupon class

```
class CustomerFirstOrderCoupon extends CouponDecorator
{
    public function applyDiscount(OrderableInterface $order)
    {
        $customer = $order->getCustomer();
        if ($customer->hasPastOrders()) {
            throw $this->createCouponException(
                'Customer already has past orders.'
            );
        }

        return $this->coupon->applyDiscount($order);
    }
}
```

The CustomerFirstOrderCoupon class usage

```
// Get 20 € off on the first order
$coupon1 = new CustomerFirstOrderCoupon(
    new ValueCoupon('3s2h7pd65s', new EUR(2000))
);

// Get 25% off on the first order
$coupon2 = new CustomerFirstOrderCoupon(
    new RateCoupon('76cqa6qr19', 0.25)
);
```

Multiple restrictions coupon

```php
// Create the special coupon
$coupon = new ValueCoupon('3s2h7pd65s', new EUR(2000));
$coupon = new LimitedLifetimeCoupon($coupon, 'now', '+60 days');
$coupon = new MinimumPurchaseAmountCoupon($coupon, new EUR(17000));
$coupon = new CustomerFirstOrderCoupon($coupon);

// Create the order instance
$totalAmount = new EUR(20000);
$order = new Order(new Customer('jsmith@example.com'), $totalAmount);

// Apply discount coupon on the order
$discountedAmount = $coupon->applyDiscount($order);

// Display the new discounted total amount
$formatter = new IntlFormatter('fr_FR');
echo sprintf("Original Price: %s\n", $formatter->format($totalAmount));
echo sprintf("Discount Price: %s\n", $formatter->format($discountedAmount));
```

Proxy

The Proxy design pattern allows to define a substitute for an object.

Introduction

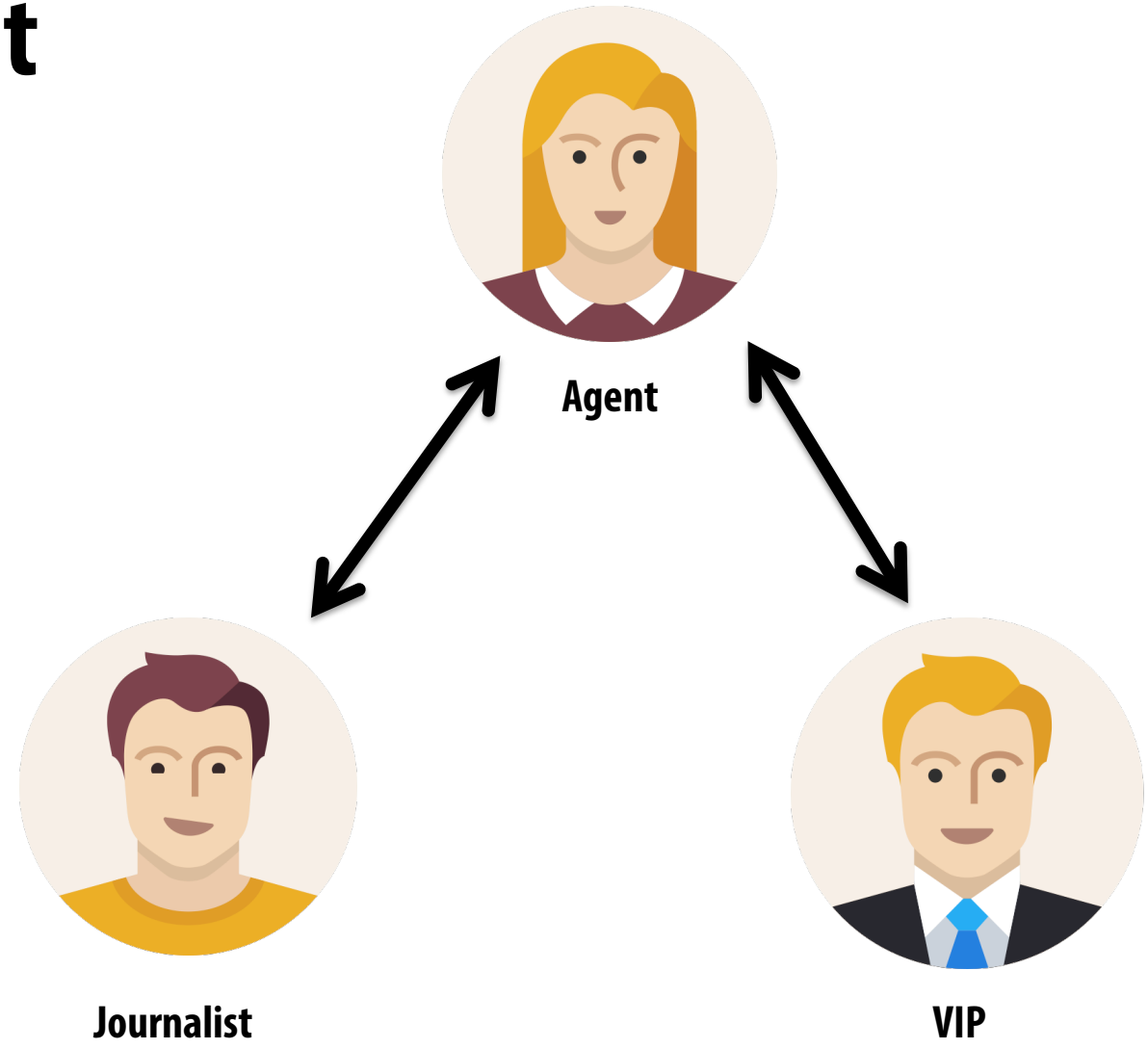
Introduction

Proxy (aka surrogate or delagate) is the concept of substituting something or someone to do something. This can be done for preserving resources, filtering information or controlling access to a resource.

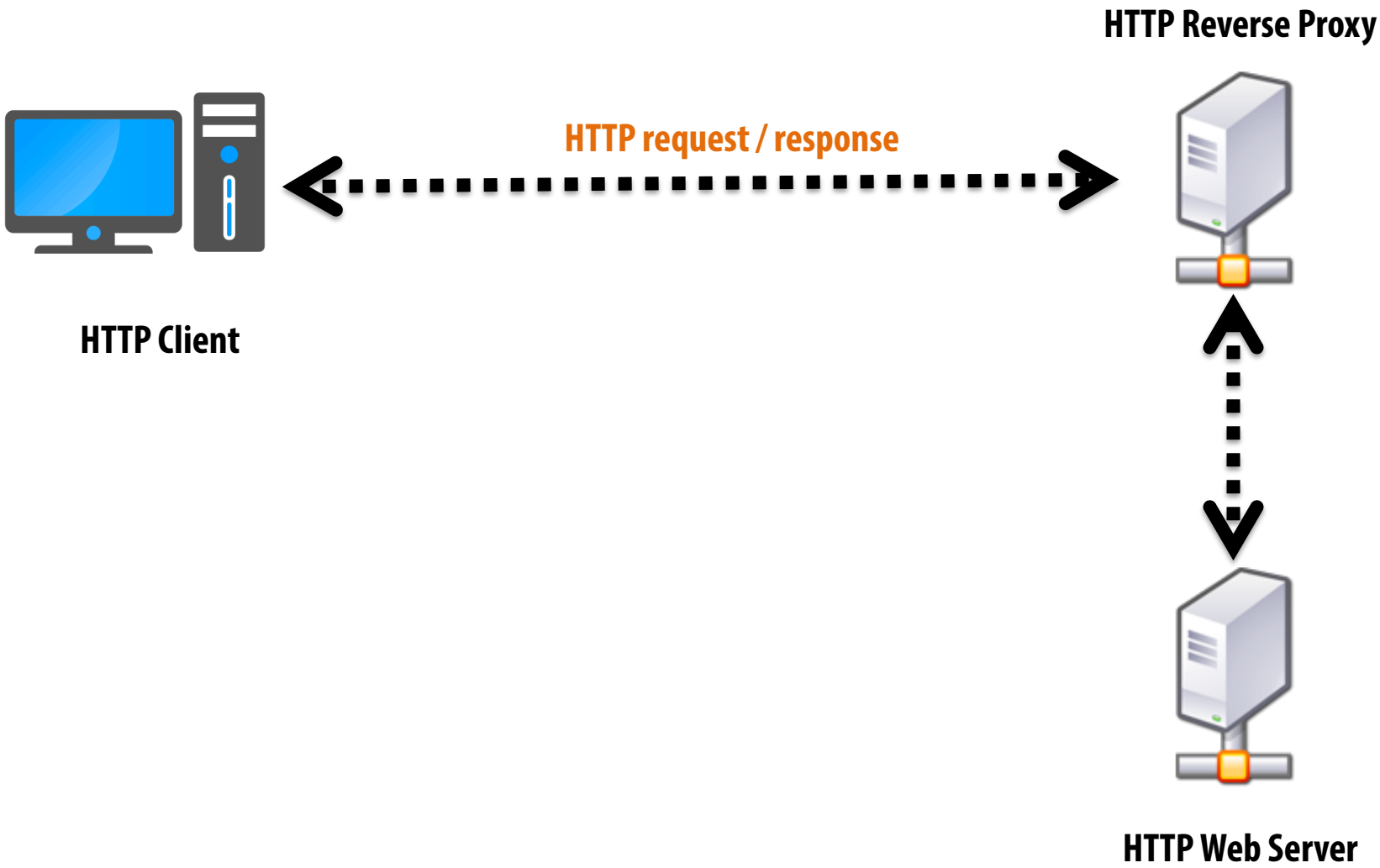
The **Proxy** design pattern aims at solving these problems that often occur in real world software applications.

Concrete Proxies in Real Life

VIP Agent



Reverse Proxy



The Proxy Design Pattern

The Proxy Pattern

- **Structural pattern**

- Provide a surrogate or placeholder for an object to control the access to it.
- Use an extra level of indirection to support distributed, controlled, or smart access.
- Add a wrapper and delegation to protect the real component from undue complexity.

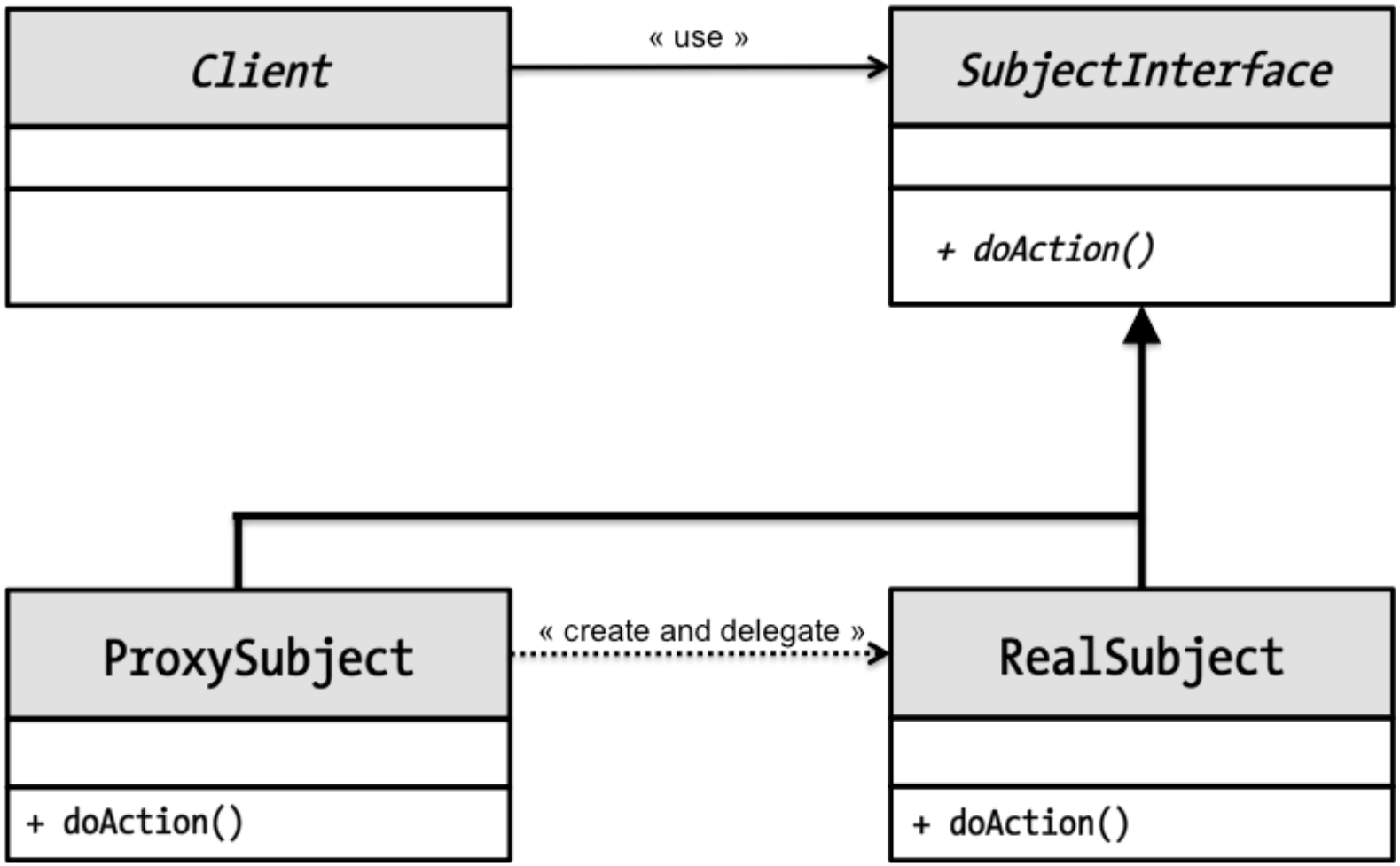
Types of Proxy Objects

- Remote proxies
- Virtual proxies
- Protection proxies
- Smart references

When to use the Proxy Design Pattern?

- Checking for permissions before accessing an object.
- Lazy loading an object that consumes lots of resources.
- Lazy loading a persistent object's relationships (Doctrine).
- Adding a lock to prevent multiple access to an object.
- Interacting with a remote object (Java RMI).

UML Diagram


Client
« use »
SubjectInterface
+ doAction()
ProxySubject
+ doAction()
« create and delegate »
RealSubject
+ doAction()

The Client class

```
class Client
{
    private $subject;

    public function __construct(SubjectInterface $subject)
    {
        $this->subject = $subject;
    }

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

The Subject Interface

```
interface SubjectInterface
{
    public function doAction();
}
```

The concrete RealSubject class

```
class RealSubject implements SubjectInterface
{
    public function doAction()
    {
        return 'real subject';
    }
}
```

The concrete ProxySubject class

```
class ProxySubject implements SubjectInterface
{
    public function doAction()
    {
        $real = new RealSubject();

        return $real->doAction();
    }
}
```

Implementing the Proxy Design Pattern

1. Remote Proxy

The Remote Proxy

The **Remote Proxy** acts as surrogate for an object that is stored on a remote server and often accessed over the wires.

```
class Tweet
{
    private $id;
    private $author;
    private $text;

    public function __construct($id, $author, $text)
    {
        $this->id = $id;
        $this->author = $author;
        $this->text = $text;
    }

    // ... + getter methods
}
```

```php
class TweetProxy extends Tweet
{
    private $api;
    private $id;

    public function __construct(TwitterApi $api, $id)
    {
        $this->api = $api;
        $this->id  = $id;
    }

    public function getText()
    {
        $tweet = $this->api->getTweet($this->id);

        return $tweet['message'];
    }
}
```

Remote Proxy Usage

```
$tweet = new Tweet('397161926', 'Hi World!');
$tweet->getText(); // Hi World!

$api = new TwitterApi('api_key', 'api_secret');

$remoteTweet = new TweetProxy($api, '8931691612');
$remoteTweet->getText(); // Some tweet text!

$remoteTweet = new TweetProxy($api, '73286343864');
$remoteTweet->getText(); // Some other tweet text!
```

Pros and Cons of Remote Proxies

+ Easy to share the object accross multiple systems
+ No need to store any subject in memory
+ Can act as an adapter for a different remote object

- Complex setup
- Fails when the remote adapter is broken
- Slow because of the network access

2. Lazy Loading

Lazy Loading Objects

Lazy loading is a technique to defer the creation of objects when they're really needed. Some proxy objects can implement this technique when real objects perform heavy and expensive tasks for the CPU and memory.

- Database queries
- Filesystem manipulations
- I/O and Streams
- Lexing and parsing documents
- …

```php
class Customer
{
    private $id;
    private $db;
    private $lastOrder;

    public function __construct(Database $db, $id)
    {
        $this->db = $db;
        $this->id = $id;
    }

    public function getLastOrder()
    {
        if ($this->lastOrder) {
            $this->lastOrder;
        }

        $sql = 'SELECT * FROM order WHERE customer = :cid ORDER BY id DESC LIMIT 1';
        $data = $this->db->fetch($query, ['cid' => $this->id]);
        if ($order) {
            $this->lastOrder = Order::fromArray($this->db, $data, $this);
        }

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

Lazy Loading Usage

```
$db = new Database('localhost', 'root', 'pass');

$customer = new Customer($db, 42);

// Order object will be magically loaded
// from the database and created in memory.

$order = $customer->getLastOrder();
$order->getReference();
```

Cons of the lazy loading approach

- Business logic and persistence logic are mixed.
- Code is more difficult to unit test.
- Not a real optimization.
- Code becomes more complex.

3. Virtual Proxy

Virtual Proxy

A **virtual proxy** is an object that looks like the **RealSubject** instance but only keeps a lazy reference to it.

The Real Subject to Proxy

```php
class Image
{
    private $image;

    public function __construct($path)
    {
        $this->image = imagecreatefromjpeg($path);
    }

    public function getDimensions()
    {
        return [
            imagesx($this->image),
            imagesy($this->image),
        ];
    }
}
```

The Virtual Proxy Object

```
class ImageProxy extends Image
{
    private $path;
    private $image;

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

    // ...
}
```

```php
class ImageProxy extends Image
{
    // ...

    public function getDimensions()
    {
        $this->load();

        return $this->image->getDimensions();
    }

    private function load()
    {
        if (!$this->image) {
            $this->image = new Image($this->path);
        }
    }
}
```

The Virtual Proxy Usage

```php
$img1 = new ImageProxy('/path/to/image1.jpg');
var_dump(memory_get_usage()); // ~200Kb

$img2 = new ImageProxy('/path/to/image2.jpg');
var_dump(memory_get_usage()); // ~200Kb

$img3 = new ImageProxy('/path/to/image3.jpg');
var_dump(memory_get_usage()); // ~200Kb

$size1 = $img1->getDimensions();
var_dump(memory_get_usage()); // ~4Mb
$size2 = $img2->getDimensions();
var_dump(memory_get_usage()); // ~8Mb
```

Pros and Cons of Virtual Proxies

+ Easy to implement
+ Initialization logic is moved away from the original class
+ Reduce CPU and memory usage
+ Remove some unneeded overhead

- Doesn’t work well for data that is always loaded
- Lazy loading also means lazy failing
- The virtual proxy is still not the same as the RealSubject
- They don’t have the same identity

4. Ghost Object

Ghost Object

A **ghost object** is an object that looks like the **RealSubject** instance. It also has all properties of the **RealSubject** set to null and initialized on demand with lazy loading when any of its methods is invoked.

This approach is better when the identity of the **RealSubject** object must be preserved. This is how the Doctrine ORM library handles data lazy fetching.

Refine the Real Subject to Proxy

```php
class Image
{
    protected $path;
    protected $image;

    public function __construct($path)
    {
        $this->path = $path;
        $this->image = imagecreatefromjpg($path);
    }
}
```

Attributes visibility must be turned to **protected** to be shared in the proxy child class.

Refine the Object to Proxy

```php
class ImageProxy extends Image
{
    private $initialized;

    public function __construct($path)
    {
        $this->path = $path;
        $this->initialized = false;
    }
}
```

The constructor must be fully overriden to skip accessing the image metadata information at construction time.

```php
class ImageProxy extends Image
{
    // ...

    public function getDimensions()
    {
        $this->load();

        return parent::getDimensions();
    }

    private function load()
    {
        if (!$this->initialized) {
            $image = new Image($this->path);
            $this->image = $image->image;
            $this->initialized = true;
        }
    }
}
```

Pros and Cons of Ghost Objects

+ Initialization logic is moved away from the original class
+ Reduce CPU and memory usage
+ Remove some unneeded overhead

- More complex to implement
- Properties must be protected or reflection must be used
- Lazy loading still means lazy failing
- Still not optimal for data that must be always loaded

5. Protection Proxy

Protection Proxy

A **protection proxy** is an object that controls access to a resource based on access rights. This pattern mostly takes the shape of a **Decorator** for the **RealSubject** instance.

Protection proxy pattern is often used to control access to some resources by filtering HTTP requests. For instance, preventing some blacklisted IP addresses to request a resource or handling an HTTP authentication on top of a REST API.

The Real Subject to Proxy

```php
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;

class Application implements HttpRequestHandlerInterface
{
    public function handle(Request $request)
    {
        return new Response('Welcome to the restricted area');
    }
}
```

This **Application** class represents an object able to convert a **Request** object into an **Response** object without doing any access controls.

The Protection Proxy Class

```
class IpAddressFilter implements HttpRequestHandlerInterface
{
    private $application;
    private $blacklist;

    public function __construct(
        HttpRequestHandlerInterface $app,
        array $blacklist
    )
    {
        $this->application = $app;
        $this->blacklist = $blacklist;
    }

    // ...
}
```

```php
class IpAddressFilter implements HttpRequestHandlerInterface
{
    // ...
    public function handle(Request $request)
    {
        if (in_array($request->getClientIp(), $this->blacklist)) {
            return new Response('Forbidden', 403);
        }

        return $this->application->handle($request);
    }
}
```

The **IpAddressFilter** class returns a **Forbidden** response object when it detects the client IP address in a list of blacklisted IP addresses.

Protection Proxy Usage

```
$blacklist = [
    '123.0.43.12',
    '144.11.0.8',
    '190.26.45.130',
];

$app = new Application();
$app = new IpAddressFilter($app, $blacklist);

$response = $app->handle($request);
$response->prepare($request);
$response->send();
```

Pros and Cons of Protection Proxy Objects

+ Easy to implement
+ No need to change the proxied object class

- The manipulated proxy is not the **RealSubject** instance
- The initial behavior of the subject is dynamically changed

6. Smart Reference Proxy

Smart Reference Proxy

A **Smart Reference Proxy** is an object that provides additional actions whenever a subject is referenced, like counting the number of references to an object.

Smart Reference proxy pattern is often used to cache the result of an heavy function call that is invoked several times. It's also often used to intercept function calls in the AOP* paradigm.

* Aspect Oriented Programming

Smart Reference Proxy Use Cases

- Caching slow method execution
- Logging information
- Event triggering
- Mocking
- AOP in general

The Real Subject to Proxy

```php
class OpenWeatherApi
{
    const API_URL = 'http://api.openweathermap.org/data/2.5';

    private $key;

    public function __construct($key, HttpClient $httpClient)
    {
        $this->key = $key;
        $this->httpClient = $httpClient;
    }

    public function getForecasts($city, $country)
    {
        $params['q'] = sprintf('%s,%s', $city, $country);

        return CurrentWeather::fromArray($this->get('/weather', $params));
    }
}
```

The Real Subject to Proxy

```php
class OpenWeatherApi
{
    const API_URL = 'http://api.openweathermap.org/data/2.5';

    // ...

    private function get($resource, array $params = [])
    {
        $params = array_merge(['appid' => $this->key, 'units' => 'metric'], $params);
        $uri = sprintf('%s%s?%s', self::API_URL, $resource, http_build_query($params));
        $response = $this->httpClient->request('GET', $uri);

        return json_decode($response->getBody(), true);
    }
}
```

The Smart Reference Proxy

```
class OpenWeatherApiProxy extends OpenWeatherApi
{
    private $cache;
    private $api;

    function __construct(OpenWeatherApi $api, CacheInterface $cache)
    {
        $this->api = $api;
        $this->cache = $cache;
    }

    public function getForecasts($city, $country)
    {
        // ...
    }
}
```

```php
class OpenWeatherApiProxy extends OpenWeatherApi
{
    // ...

    public function getForecasts($city, $country)
    {
        $city = strtolower($city);
        $country = strtolower($country);
        $key = md5(sprintf('weather:%s:%s', $city, $country));

        if ($data = $this->cache->fetch($key)) {
            return $data;
        }

        $data = $this->api->getForecasts($city, $country);
        $this->cache->store($key, $data, 3600);

        return $data;
    }
}
```

Smart Reference Proxy Usage

```php
define('API_KEY', '1a9e7bf41b93d2f343ac277156bd5865');

$api = new OpenWeatherApi(API_KEY, new HttpClient());
$cache = new RedisCacheDriver();

$api = new OpenWeatherApiProxy($api, $cache);
$weather = $api->getForecasts('Paris', 'FR');

echo $weather->getTemperature();
```

Thanks to the smart reference, the HTTP network call to the OpenWeather API is made only once every hour per city forecasts.

Things to consider when implementing proxies

Caveats when implementing proxies

Implementing proxies with PHP is not as easy as it can get. This is because of the numerous features, behaviors or special cases to handle with OOP in PHP.

- Public properties
- Serialization
- Magic getter methods
- Magic setter methods
- Magic methods invocations
- Cloning
- Fluent interfaces
- etc.

1. Fluent Interfaces

```php
class BankAccount
{
    protected $balance;
    protected $currency;

    public function __construct($balance, $currency)
    {
        $this->balance = $balance;
        $this->currency = $currency;
    }

    public function deposit($amount, $currency)
    {
        // ...
    }
}
```

```php
class BankAccount
{
    // ...
    public function deposit($amount, $currency)
    {
        if ($currency !== $this->currency) {
            throw new CurrencyMismatchException();
        }

        $this->balance += $amount;

        return $this;
    }
}
```

```php
$account = (new BankAccount(500, 'EUR'))
  ->deposit(50, 'EUR')
  ->deposit(30, 'EUR')
  ->deposit(20, 'EUR');
```

The **deposit()** method of the **BankAccount** instance always returns itself to provide a fluent interface. Through it seems to be very practical, returning the same instance makes it also much more difficult to design proxies as it usually breaks the LSP principle from SOLID.

Proxying fluent interfaces breaks virtual proxies, protection proxies and smart reference proxies.

```
class PoorBankAccountProxy extends BankAccount
{
    private $converter;
    private $account;

    function __construct(BankAccount $wrapped, CurrencyConverter $converter)
    {
        $this->account = $wrapped;
        $this->converter = $converter;
    }

    public function deposit($amount, $currency)
    {
        $curr = $this->account->currency;
        if ($currency !== $accountCurrency) {
            $amount = $this->converter->changeCurrencies($amount, $currency, $curr);
        }

        return $this->account->deposit($amount, $curr);
    }
}
```

```php
$converter = new CurrencyConverter();
$account1 = new BankAccount(500, 'EUR')
$account2 = new PoorBankAccountProxy($account1, $converter);
$account2
    ->deposit(50, 'USD')  // Returns $account1 proxied subject
    ->deposit(30, 'EUR')  // Throw a CurrencyMismatchException
    ->deposit(20, 'CHF'); // Never called!
```

The **deposit()** method of the **PoorBankAccountProxy** class always returns the real **BankAccount** subject on its first invocation instead returning itself. This leads to chaining the method call on the **BankAccount** real subject instead of the **PoorBankAccountProxy** object.

However, the following **dirty** solution can do the trick…

```
class BankAccountProxy extends BankAccount
{
    private $converter;

    public function __construct(BankAccount $wrapped, CurrencyConverter $converter)
    {
        $this->currency = $wrapped->currency;
        $this->balance = & $wrapped->balance;
        $this->converter = $converter;
    }

    public function deposit($amount, $currency)
    {
        if ($currency !== $this->currency) {
            $amount = $this->converter->changeCurrencies($amount, $currency, $this->currency);
        }

        return parent::deposit($amount, $this->currency);
    }
}
```

```php
$converter = new CurrencyConverter();
$account1 = new BankAccount(500, 'EUR')
$account2 = new BankAccountProxy($account1, $converter);
$account2
    ->deposit(50, 'USD')  // Returns $account2 proxy!
    ->deposit(30, 'EUR')  // Returns $account2 proxy!
    ->deposit(20, 'CHF'); // Returns $account2 proxy!
```

The **deposit()** method of the **BankAccountProxy** class now always returns the reference to itself without breaking the LSP principle from SOLID.

2. Public Properties

Accessing Public Attributes

```
$converter = new CurrencyConverter();
$account1 = new BankAccount()
$account2 = new PoorBankAccountProxy($account1, $converter);

$account2->balance = 500;    // Doesn’t update $account1->balance
$account2->currency = 'EUR'; // Doesn’t update $account1->currency
```

Using public attributes prevents the proxy object to update the related same attributes of the wrapped real subject they’re read or affected.

However, magic methods provides a **dirty** solution that does the trick… Use with caution!

```
class BankAccountProxy extends BankAccount
{
    private $account;

    public function __construct(BankAccount $wrapped)
    {
        unset($this->balance, $this->currency);
        $this->account = $wrapped;
    }

    private function validateProperty($name)
    {
        if (!property_exists($this->account, $name)) {
            throw new \RuntimeException('Expected attribute!');
        }
    }
}
```

```php
class BankAccountProxy extends BankAccount
{
    // ...

    public function __get($name)
    {
        $this->validateProperty($name);
        return $this->account->$name;
    }

    public function __set($name, $value)
    {
        $this->validateProperty($name);
        $this->account->$name = $value;
    }

    public function __isset($name)
    {
        $this->validateProperty($name);

        return true;
    }
}
```

Proxy with public attributes

```
$account1 = new BankAccount()
$account2 = new BankAccountProxy($account1);

// Update $account1->balance
$account2->balance = 500;

// Update $account1->currency
$account2->currency = 'EUR';
```

Useful Proxy Generator Libraries

Useful Proxies Generator Libraries

- **Doctrine Proxy Generator** (`doctrine/common`)
- **ProxyManager** (`ocramius/proxy-manager`)
- **Flow3** (from Typo3 framework)
- **Code Generator** (`jms/cg`)
- **GoAOP** (`goaop/framework`)

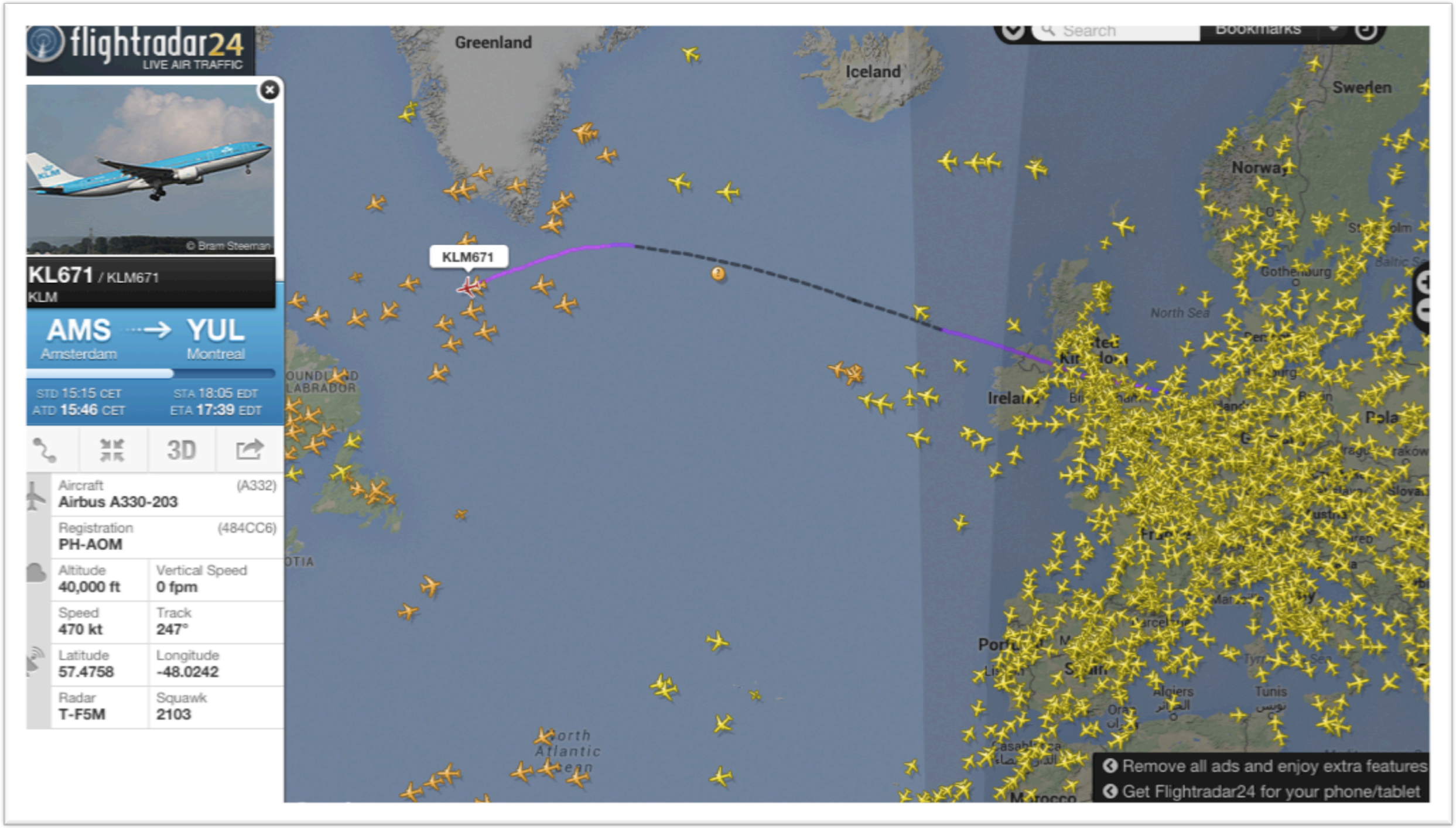
Flyweight

A flyweight is an object that minimizes memory usage by sharing as much data as possible with other similar objects

Introduction

Handling Large Data Sets

Handling Large Data Sets



Model Major Aircrafts

Model Major Aircrafts

```
class Aircraft
{
    private $manufacturer;
    private $aircraftSeries;
    private $registrationNumber;
    private $dateOfCommissioning;
    private $latitude;
    private $longitude;
    private $altitude;
    private $enginesCount;
    private $wingspan;
    private $fuselageLength;
    private $cruiseSpeed;
    private $distanceRange;
    // ...
}
```

```
$aircraft1 = new Aircraft('F-HJPA', '2009-12-02','Airbus', 'A380');
$aircraft1->setPosition(59.89, -28.22, 36000);
$aircraft1->setEnginesCount(4);
$aircraft1->setWingspan(80);
$aircraft1->setFuselageLength(73);
$aircraft1->setCruiseSpeed(945);
$aircraft1->setDistanceRange(15700);
```

```
$aircraft2 = new Aircraft('A6-EDJ', '2006-08-25', 'Airbus', 'A380');
$aircraft2->setPosition(25.412, 54.634, 15150);
$aircraft2->setEnginesCount(4);
$aircraft2->setWingspan(80);
$aircraft2->setFuselageLength(73);
$aircraft2->setCruiseSpeed(945);
$aircraft2->setDistanceRange(15700);
```

```
$aircraft3 = new Aircraft('9V-SKF', '2008-02-20', 'Airbus', 'A380');
$aircraft3->setPosition(4.8081, 99.1604, 40000);
$aircraft3->setEnginesCount(4);
$aircraft3->setWingspan(80);
$aircraft3->setFuselageLength(73);
$aircraft3->setCruiseSpeed(945);
$aircraft3->setDistanceRange(15700);
```

Each A380 aircraft is a unique instance but it also shares the same technical specifications with other aircrafts of the A380 family.

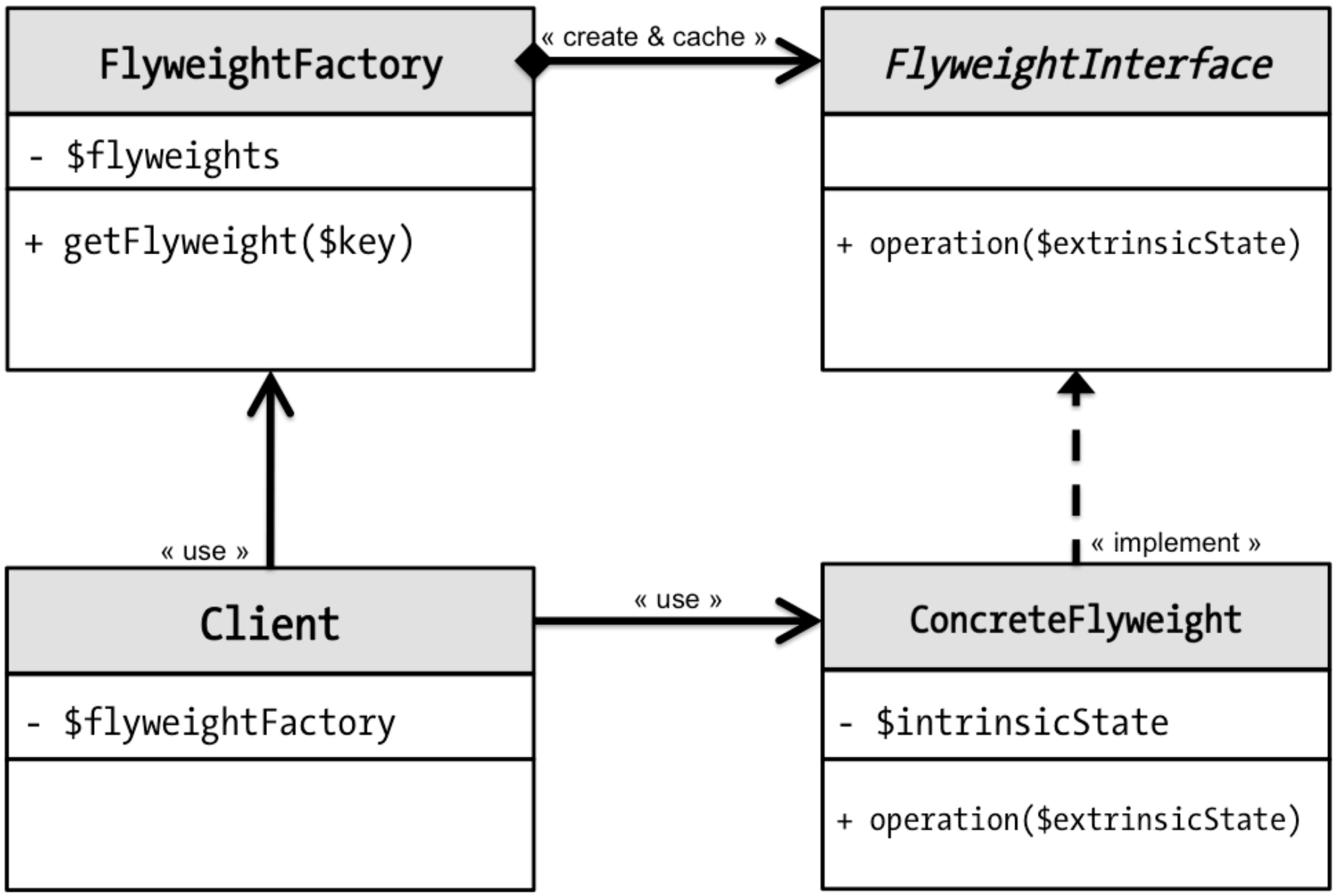
The Flyweight Design Pattrern

The Flyweight

Structural pattern
Inspired by Singleton & Factory Method

- Sharing instances to reuse them efficiently
- Creating new instances on-demand with a Factory
- Keeping memory usage as low as possible
- Fits the needs for handling huge amount of similar objects

UML Diagram


FlyweightFactory
- $flyweights
+ getFlyweight($key)
« create & cache »
FlyweightInterface
+ operation($extrinsicState)
« use »
« implement »
Client
- $flyweightFactory
« use »
ConcreteFlyweight
- $intrinsicState
+ operation($extrinsicState)

Intrinsic State

The Intrinsic State is defined as a simple immutable value object that encapsulates the common shared properties of all distinct entities.

Extrinsic State

The Extrinsic State refers to the attributes that distinguish the objects from each other. This state is always extracted and kept outside the flyweight object.

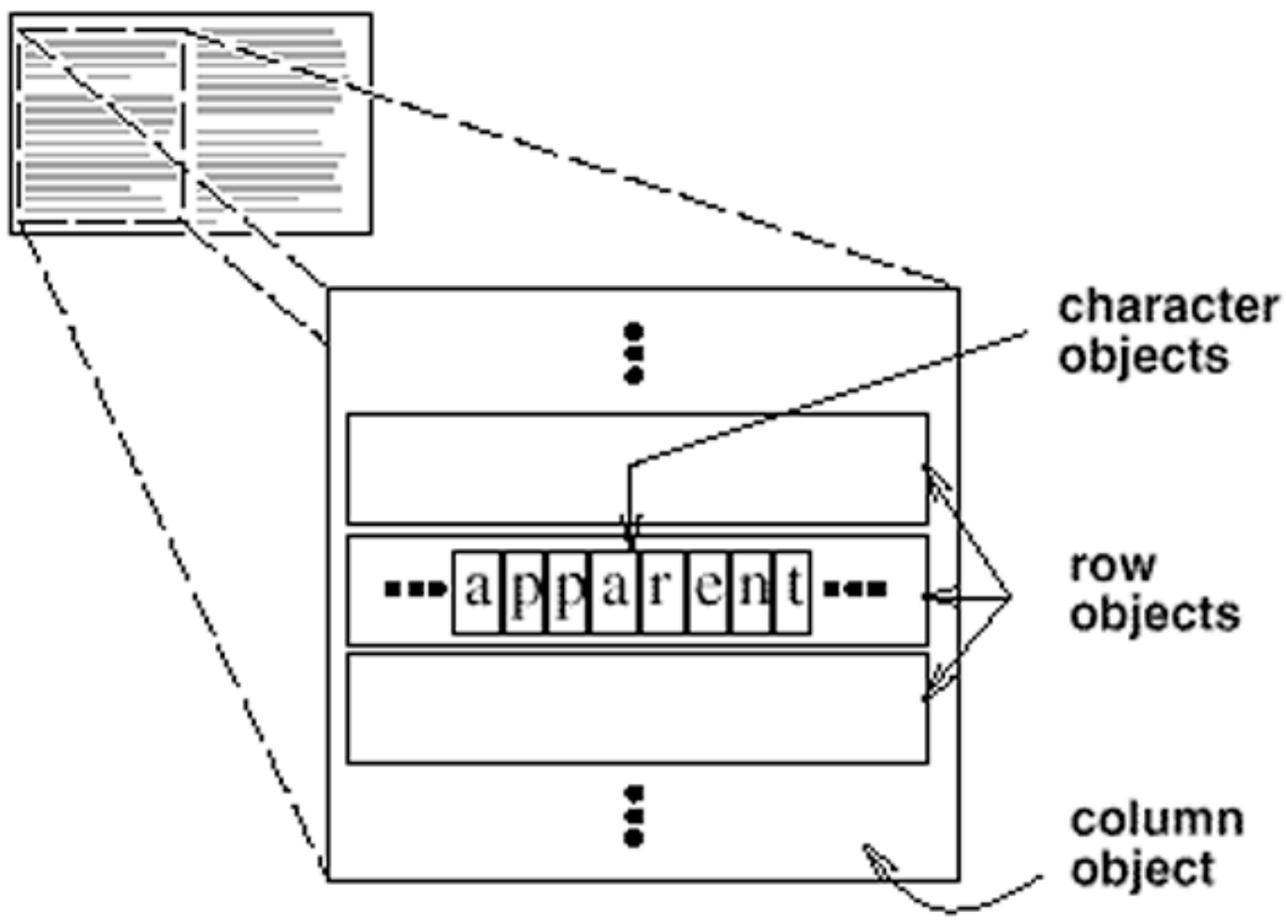
It can be kept in a separate entity or explicitly passed as parameters of the flyweight instance methods.

Value Objects

- They model a value (money, weight, atom…)
- They don’t have any identity
- They are immutable (state doesn’t change)
- They are interchangeable

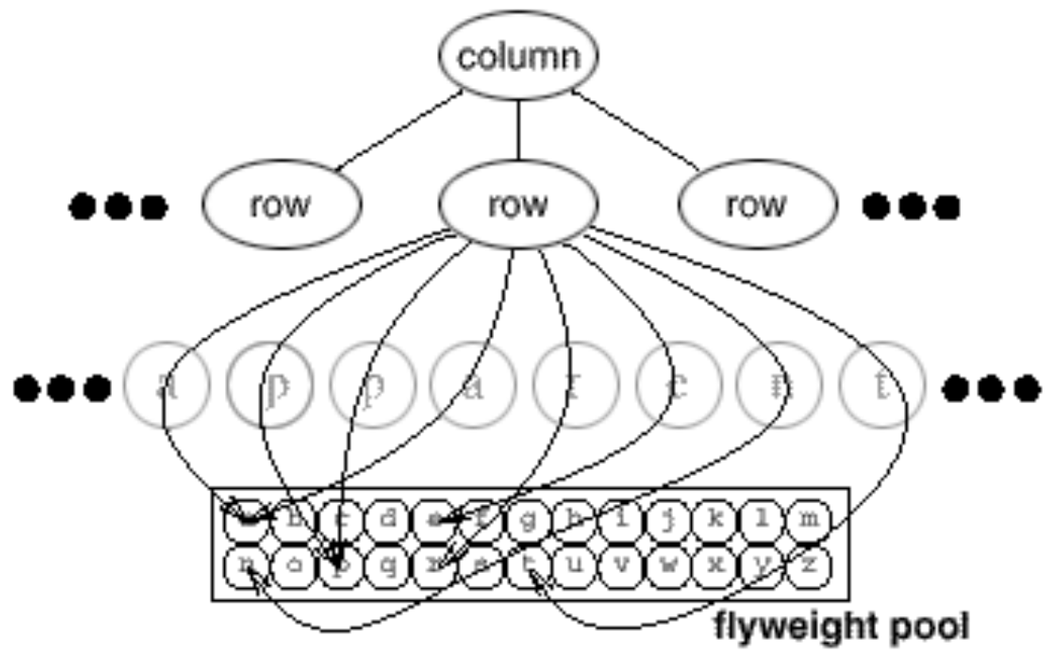
Use Cases for Flyweight Pattern Implementation

Word Processor



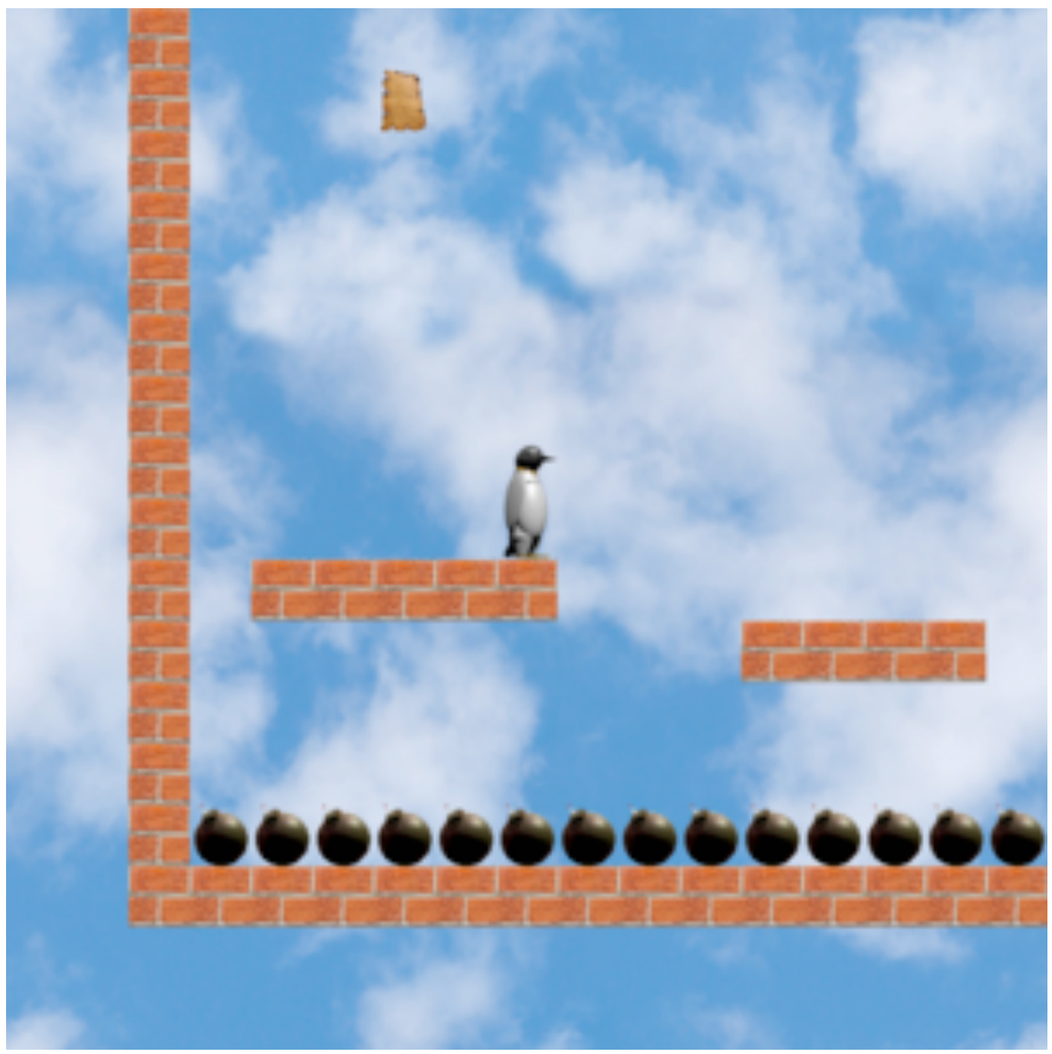
A text document is a large 2 dimensions matrix where each cell has a reference to a single character (aka glyph).

Word Processor



Each character instance holds a reference to a Glyph instance (the intrinsic state) and stores its position in the document as well as its formatting style (the extrinsic state).

Video Games



Rendering a single sprite many times in different locations on map (a tree forest for example).

Image Processing Software

Modeling a black image of 1000 x 1000 pixels.

```
$width  = 1000;
$height = 1000;
$colors = [];

for ($x = 0; $x < $width; $x++) {
    for ($y = 0; $y < $height; $y++) {
        $colors[$x][$y] = new Color(0, 0, 0);
    }
}
```

Image Processing Software

Modeling a black image of 1000 x 1000 pixels.

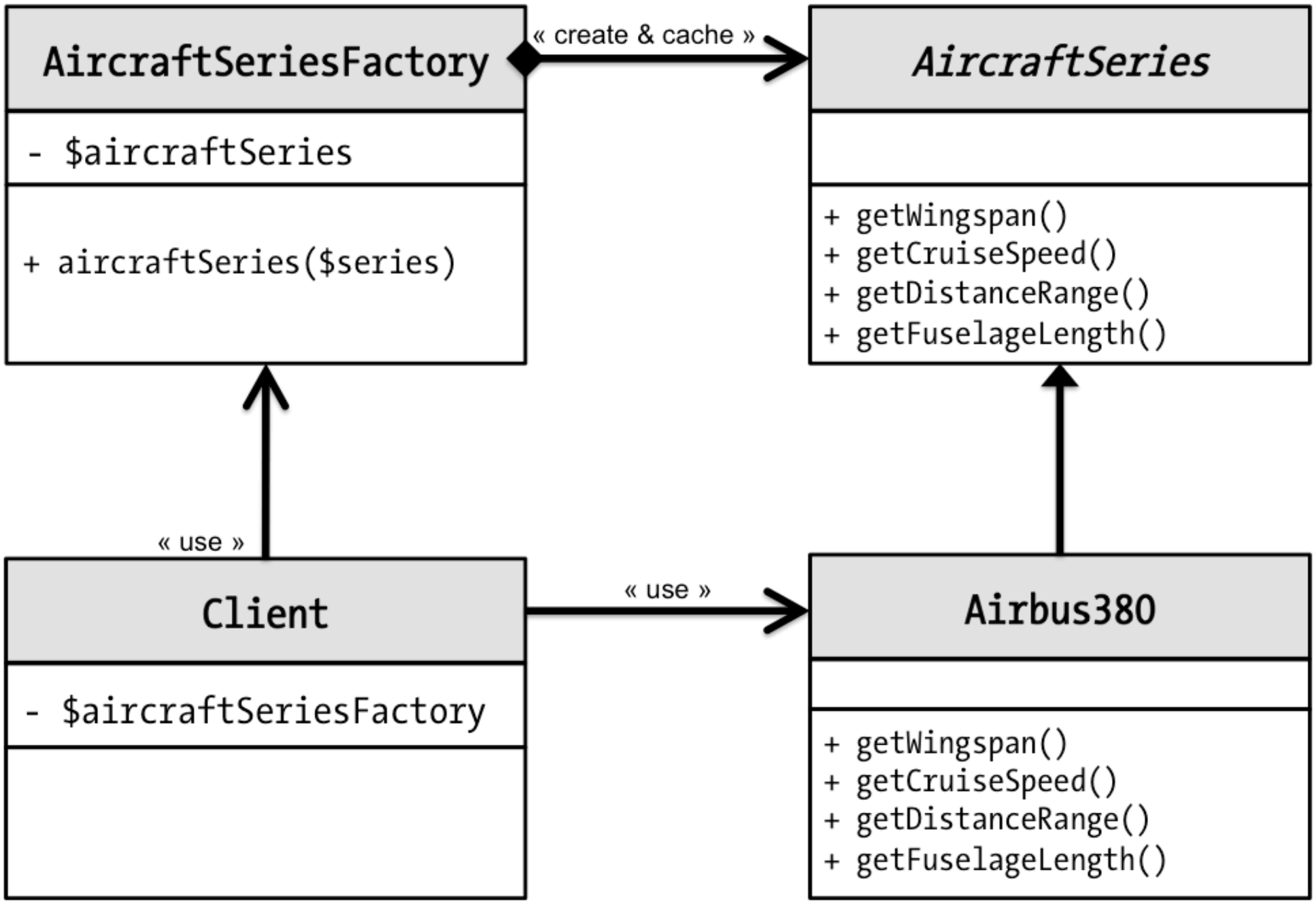
```
$width  = 1000;
$height = 1000;
$black  = new Color(0, 0, 0);
$colors = [];

for ($x = 0; $x < $width; $x++) {
    for ($y = 0; $y < $height; $y++) {
        $colors[$x][$y] = $black;
    }
}
```

Symfony Form Type Objects

```
namespace Symfony\Component\Form\Extension\Core\Type;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\OptionsResolver\OptionsResolverInterface;

class TextType extends AbstractType
{
    // ...

    public function getName()
    {
        return 'text';
    }
}
```

Implementing the Flyweight Design Pattrern

Aircraft Series Flyweight



Isolating the Intrinsic State

```
namespace FlightRadar;

interface AircraftSeriesInterface
{
    public function getManufacturer();
    public function getWingspan();
    public function getFuselageLength();
    public function getCruiseSpeed();
    public function getDistanceRange();
    public function getEnginesCount();
}
```

The A380 Flyweight Object

```php
namespace FlightRadar\AircraftSeries;

use FlightRadar\AircraftSeries;

final class Airbus380 extends AircraftSeries
{
    public function __construct()
    {
        $this->setManufacturer('Airbus');
        $this->setEnginesCount(4);
        $this->setFuselageLength(73);
        $this->setWingspan(80);
        $this->setCruiseSpeed(945);
        $this->setDistanceRange(15700);
    }
}
```

The B747 Flyweight Object

```php
namespace FlightRadar\AircraftSeries;

use FlightRadar\AircraftSeries;

final class Boeing747 extends AircraftSeries
{
    public function __construct()
    {
        $this->setManufacturer('Boeing');
        $this->setEnginesCount(4);
        $this->setFuselageLength(76);
        $this->setWingspan(64);
        $this->setCruiseSpeed(918);
        $this->setDistanceRange(14200);
    }
}
```

The B777 Flyweight Object

```php
namespace FlightRadar\AircraftSeries;

use FlightRadar\AircraftSeries;

final class Boeing777 extends AircraftSeries
{
    public function __construct()
    {
        $this->setManufacturer('Boeing');
        $this->setEnginesCount(2);
        $this->setFuselageLength(64);
        $this->setWingspan(65);
        $this->setCruiseSpeed(905);
        $this->setDistanceRange(14500);
    }
}
```

The Aircraft Object

```php
namespace FlightRadar;

class Aircraft
{
    private $aircraftSeries;
    private $registrationNumber;
    private $dateOfCommissioning;
    private $latitude;
    private $longitude;
    private $altitude;

    public function __construct(
        AircraftSeriesInterface $series,
        $registrationNumber,
        $dateOfCommissioning
    )
    {
        $this->aircraftSeries = $series;
        $this->registrationNumber = $registrationNumber;
        $this->dateOfCommissioning = $dateOfCommissioning;
    }

    // ...
}
```

Using the Flyweight Objects

```
use FlightRadar\Aircraft;
use FlightRadar\AircraftSeries\Airbus380;
use FlightRadar\AircraftSeries\Boeing747;
use FlightRadar\AircraftSeries\Boeing777;

// Flyweight immutable value objects (shared intrinsic state)
$airbusA380 = new Airbus380();
$boeingB747 = new Boeing747();
$boeingB777 = new Boeing777();

// A380 Aircraft objects (dependent extrinsic state)
$aircraft1 = new Aircraft($airbusA380, 'F-HJPA', '2009-12-02');
$aircraft1->setPosition(59.89, -28.22, 36000);

$aircraft2 = new Aircraft($airbusA380, 'A6-EDJ', '2006-08-25');
$aircraft2->setPosition(25.412, 54.634, 15150);

$aircraft3 = new Aircraft($airbusA380, '9V-SKF', '2008-02-20');
$aircraft3->setPosition(4.8081, 99.1604, 40000);
```

Using the Flyweight Objects

```
// B747 Aircraft objects (dependent extrinsic state)
$aircraft4 = new Aircraft($boeingB747, 'F-GITE', '1992-03-04');
$aircraft4->setPosition(49.0047, 2.5945, 0);

$aircraft5 = new Aircraft($boeingB747, 'N675NW', '2002-02-06');
$aircraft5->setPosition(39.9024, 15.2953, 36950);

// B77 Aircraft objects (dependent extrinsic state)
$aircraft6 = new Aircraft($boeingB777, 'A6-EMX', '2003-06-10');
$aircraft6->setPosition(31.8191, 49.6526, 34975);

$aircraft7 = new Aircraft($boeingB777, 'PH-BVG', '2012-05-21');
$aircraft7->setPosition(55.6094, 11.2096, 36000);

$aircraft8 = new Aircraft($boeingB777, 'G-VIIO', '1999-01-18');
$aircraft8->setPosition(51.76, -9.9, 36000);

// ... and thousands of other aircrafts
```

Going Further with a Flyweight Factory

The Flyweight Factory

Flyweight immutable value objects can be instantiated on-demand and easily shared thanks to a factory method instance.

The AircraftSeriesFactory Class

```php
namespace FlightRadar;

class AircraftSeriesFactory implements AircraftSeriesFactoryInterface
{
    private static $aircraftSeriesClasses = [
        'A380' => '\\FlightRadar\\AircraftSeries\\Airbus380',
        'B747' => '\\FlightRadar\\AircraftSeries\\Boeing747',
        'B777' => '\\FlightRadar\\AircraftSeries\\Boeing777',
    ];

    private static $aircraftSeries = [];

    public function aircraftSeries($series)
    {
        if (isset(static::$aircraftSeries[$series])) {
            return static::$aircraftSeries[$series];
        }

        if (!isset(static::$aircraftSeriesClasses[$series])) {
            throw new UnknownAircraftSeriesException($series);
        }

        $class = static::$aircraftSeriesClasses[$series];

        return static::$aircraftSeries[$series] = new $class();
    }
}
```

The AircraftSeriesFactory Object

```php
use FlightRadar\Aircraft;
use FlightRadar\AircraftSeriesFactory;

// Flyweight factory
$factory = new AircraftSeriesFactory();

// Aircraft objects (dependent extrinsic state)
$aircraft1 = new Aircraft($factory->aircraftSeries('A380'), 'F-HJPA', '2009-12-02');
$aircraft1->setPosition(59.89, -28.22, 36000);

$aircraft2 = new Aircraft($factory->aircraftSeries('A380'), 'A6-EDJ', '2006-08-25');
$aircraft2->setPosition(25.412, 54.634, 15150);

$aircraft4 = new Aircraft($factory->aircraftSeries('B747'), 'F-GITE', '1992-03-04');
$aircraft4->setPosition(49.0047, 2.5945, 0);

$aircraft5 = new Aircraft($factory->aircraftSeries('B747'), 'N675NW', '2002-02-06');
$aircraft5->setPosition(39.9024, 15.2953, 36950);

$aircraft6 = new Aircraft($factory->aircraftSeries('B777'), 'A6-EMX', '2003-06-10');
$aircraft6->setPosition(31.8191, 49.6526, 34975);

$aircraft7 = new Aircraft($factory->aircraftSeries('B777'), 'PH-BVG', '2012-05-21');
$aircraft7->setPosition(55.6094, 11.2096, 36000);


// ... + thousands of other aircrafts
```

Bridge

Decouple an abstraction from its implementation so that the two can vary independently

Introduction

Multiple payment methods and providers

A website wants to provides payment solutions for its customers. Customers are invited to pay with a **credit card**, a **money transfer** from their bank accounts to the merchant's one, a **check**, or even with some **voucher** or **gift card**.

Behind the scenes, the system also provides integration with several payment providers like **Stripe**, **Paypal**, or directly with the **website's bank payment** solution.

Whatever the payment method the customer chooses, the system must be able to use any suitable payment provider to process a transaction.

Inheritance is not a valid solution!

With a typical inheritance solution, solving this problem will lead to creating one new class for each potential combination of a payment method and payment provider.

- `StripeCreditCardPayment`
- `StripeWireTransferPayment`
- `PaypalCreditCardPayment`
- `PaypalWireTransferPayment`
- `Etc.`

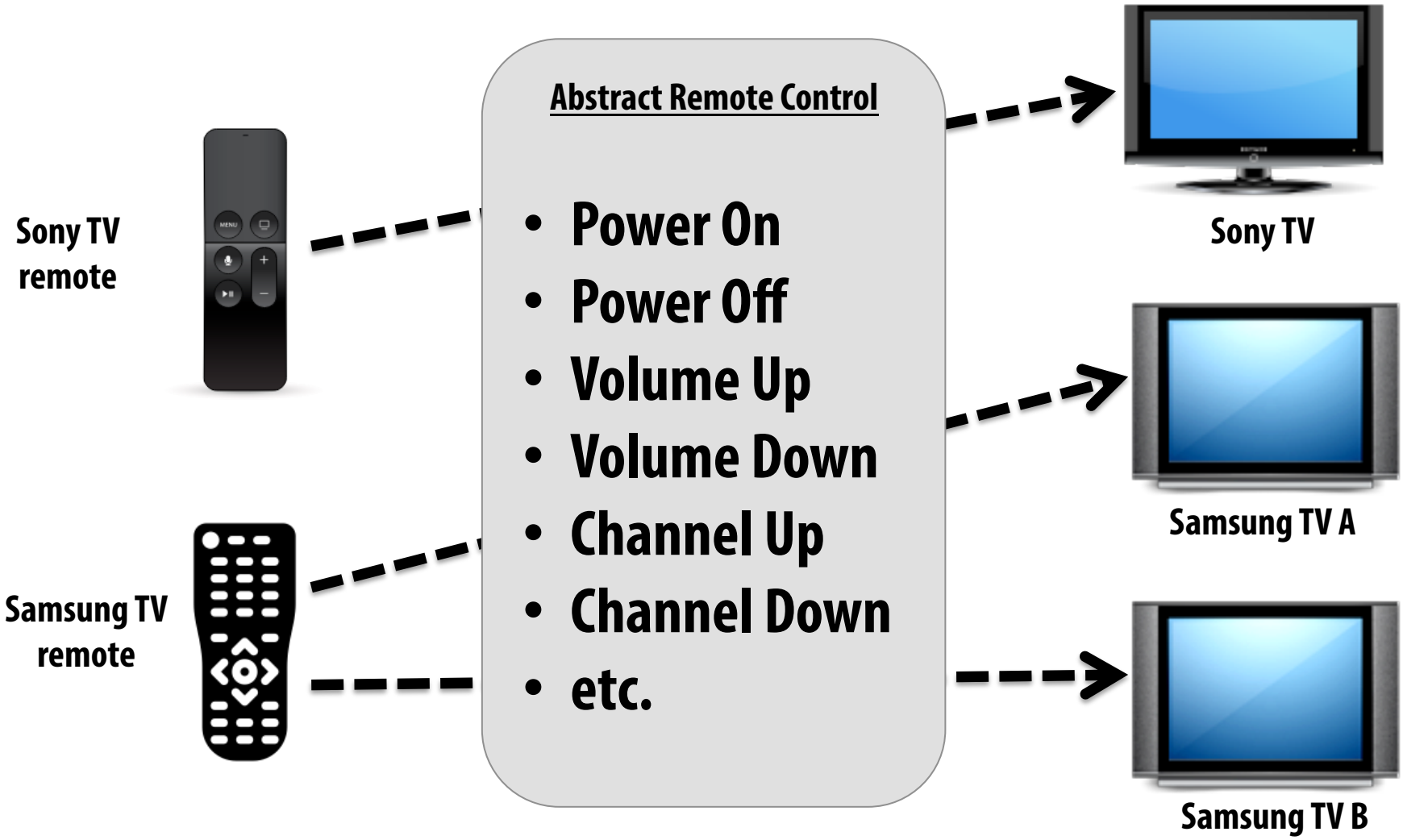
Composition over inheritance!

Once again the **composition over inheritance** principle is the key to solve this problem and support more complex combinations.

The **Bridge** design pattern enables to decouple the concept of a payment method (credit card, wire transfer, etc.) from its implementors (the payment providers), so that both can evolve independently.

Concrete Bridges in Real Life

The Remote Control



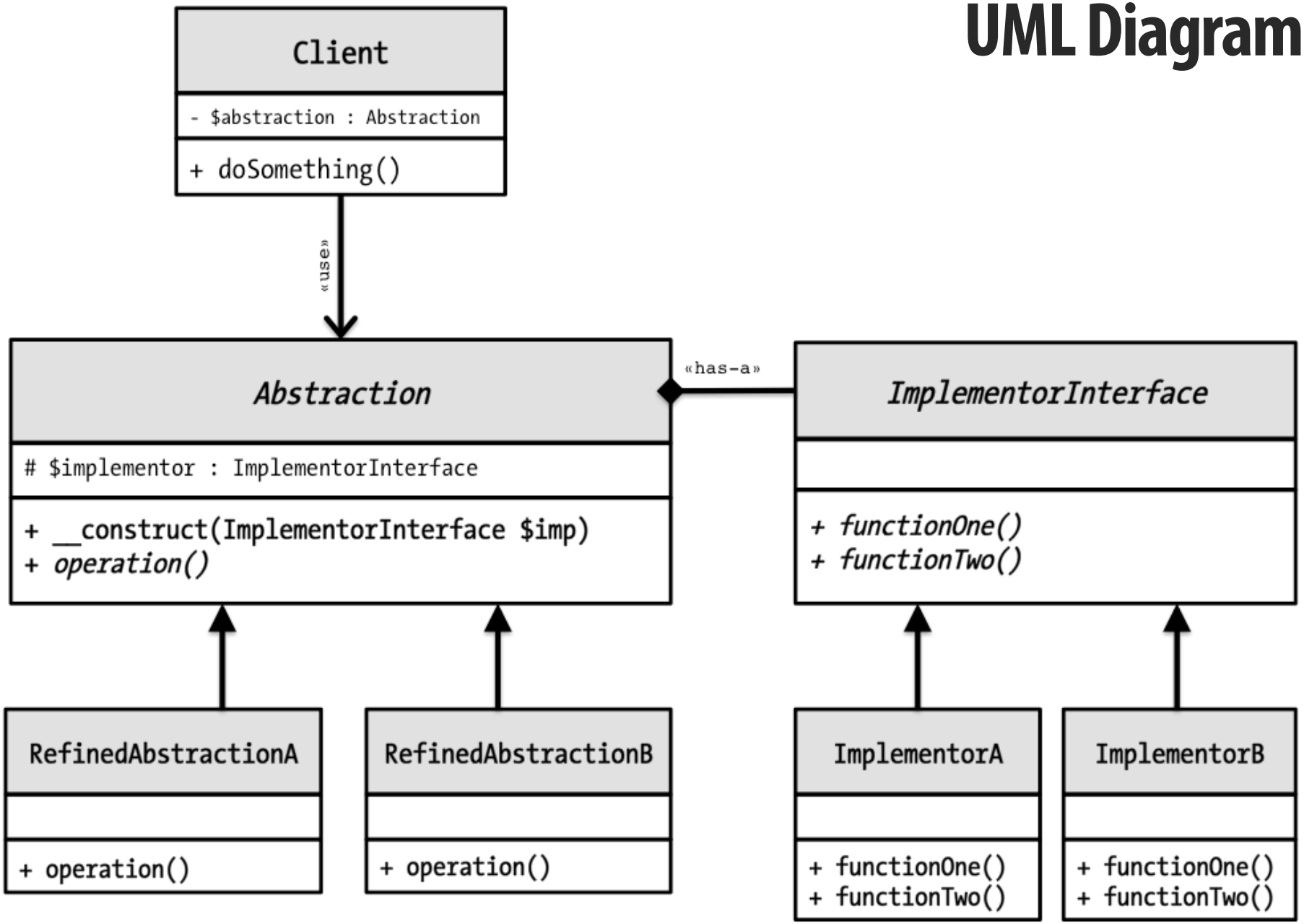
The Bridge Design Pattern

The Bridge Pattern

- **Structural pattern**

- Decouple an abstraction from its implementation so that the two can vary independently.
- Favor composition over inheritance.
- Introduce an « has-a » relationship.

UML Diagram


Client
- $abstraction : Abstraction
+ doSomething()
«use»
Abstraction
$implementor : ImplementorInterface
+ __construct(ImplementorInterface $imp)
+ operation()
«has-a»
ImplementorInterface
+ functionOne()
+ functionTwo()
RefinedAbstractionA
+ operation()
RefinedAbstractionB
+ operation()
ImplementorA
+ functionOne()
+ functionTwo()
ImplementorB
+ functionOne()
+ functionTwo()

Bridge Pattern Usage

```
$bridgeAA = new RefinedAbstractionA(new ImplementorA());
$bridgeAB = new RefinedAbstractionA(new ImplementorB());
$bridgeBA = new RefinedAbstractionB(new ImplementorA());
$bridgeBB = new RefinedAbstractionB(new ImplementorB());

$client1 = new Client($bridgeAA);
$client1->doSomething();

$client2 = new Client($bridgeAB);
$client2->doSomething();

$client3 = new Client($bridgeBA);
$client3->doSomething();

$client4 = new Client($bridgeBB);
$client4->doSomething();
```

The Client class

```php
class Client
{
    private $bridge;

    public function __construct(Abstraction $bridge)
    {
        $this->bridge = $bridge;
    }

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

The Implementor Interface

```
interface ImplementorInterface
{
    public function functionOne();
    public function functionTwo();
}
```

The concrete ImplementorA class

```
class ImplementorA implements ImplementorInterface
{
    public function functionOne()
    {
        return 'with implementor A and function One';
    }

    public function functionTwo()
    {
        return 'with implementor A and function Two';
    }
}
```

The concrete ImplementorB class

```
class ImplementorB implements ImplementorInterface
{
    public function functionOne()
    {
        return 'with implementor B and function One';
    }

    public function functionTwo()
    {
        return 'with implementor B and function Two';
    }
}
```

The abstract Abstraction class

```
abstract class Abstraction
{
    protected $implementor;

    public function __construct(ImplementorInterface $imp)
    {
        $this->implementor = $imp;
    }

    abstract public function operation();
}
```

The concrete RefinedAbstractionA class

```
class RefinedAbstractionA extends Abstraction
{
    public function operation()
    {
        return sprintf(
            "Bridging RefinedAbstractionA %s.\n",
            $this->implementor->functionOne()
        );
    }
}
```

The concrete RefinedAbstractionB class

```
class RefinedAbstractionB extends Abstraction
{
    public function operation()
    {
        return sprintf(
            "Bridging RefinedAbstractionB %s.\n",
            $this->implementor->functionTwo()
        );
    }
}
```

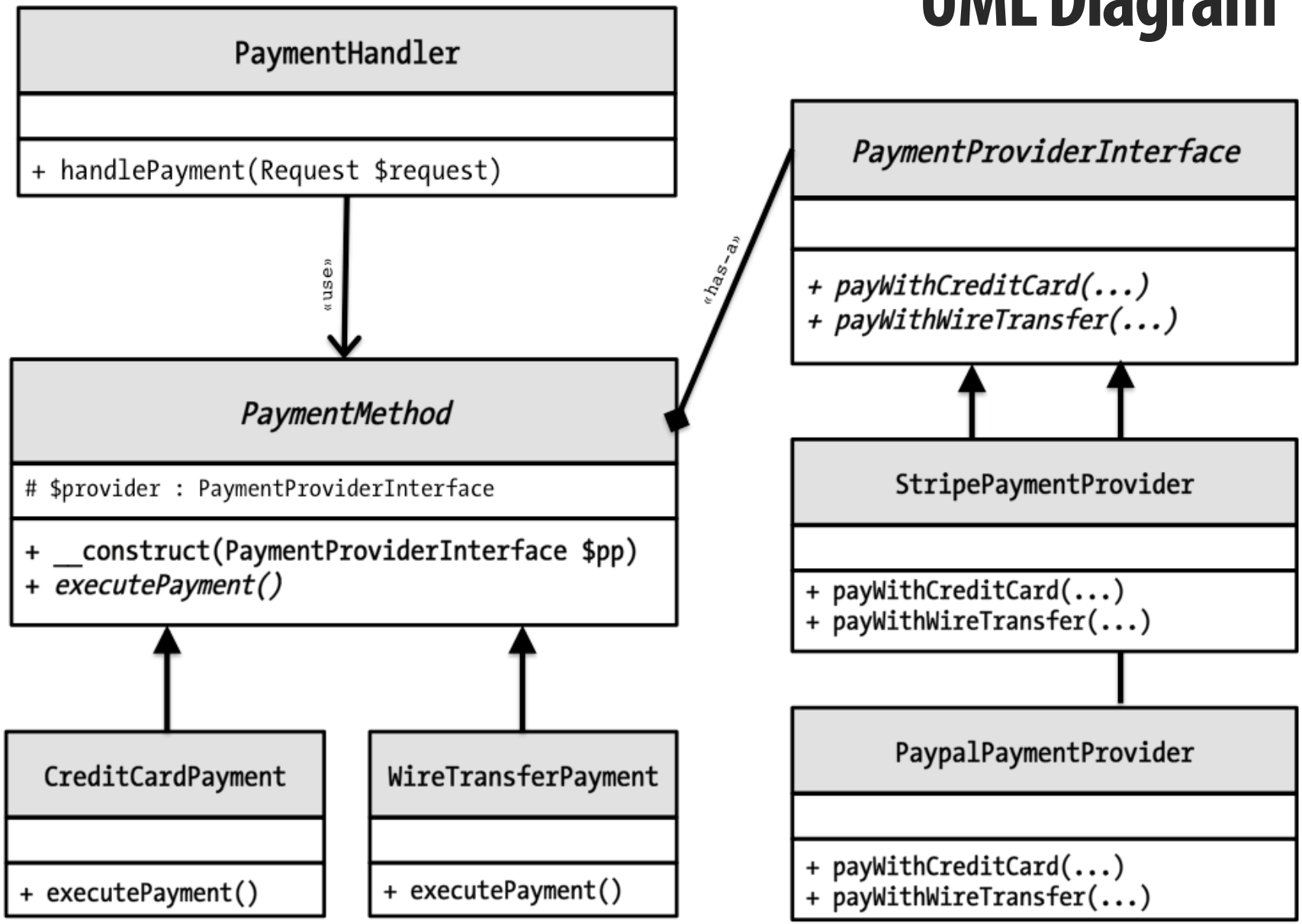
Bridge pattern usage

```php
$client1 = new Client(new RefinedAbstractionA(new ImplementorA()));
$client1->doSomething();

$client2 = new Client(new RefinedAbstractionA(new ImplementorB()));
$client2->doSomething();

$client3 = new Client(new RefinedAbstractionB(new ImplementorA()));
$client3->doSomething();

$client4 = new Client(new RefinedAbstractionB(new ImplementorB()));
$client4->doSomething();
```

```
Output:
Bridging RefinedAbstractionA with implementor A and function One.
Bridging RefinedAbstractionA with implementor B and function One.
Bridging RefinedAbstractionB with implementor A and function Two.
Bridging RefinedAbstractionB with implementor B and function Two.
```

Implementing the Bridge Design Pattern

UML Diagram


PaymentHandler
+ handlePayment(Request $request)
«use»
«has-a»
PaymentProviderInterface
+ payWithCreditCard(...)
+ payWithWireTransfer(...)
PaymentMethod
$provider : PaymentProviderInterface
+ __construct(PaymentProviderInterface $pp)
+ executePayment()
StripePaymentProvider
+ payWithCreditCard(...)
+ payWithWireTransfer(...)
CreditCardPayment
+ executePayment()
WireTransferPayment
+ executePayment()
PaypalPaymentProvider
+ payWithCreditCard(...)
+ payWithWireTransfer(...)

Implementing the Bridge Pattern

```php
abstract class PaymentMethod
{
    protected $provider;

    function __construct(PaymentProviderInterface $provider)
    {
        $this->provider = $provider;
    }

    abstract public function executePayment();
}
```

The PaymentMethod class represents the abstraction that must be refined and bridge the concrete payment providers.

Implementing the Bridge Pattern

```php
class CreditCardPayment extends PaymentMethod
{
    public function executePayment()
    {
        return sprintf(
            'Paying with a credit card via %s.',
            $this->provider->payWithCreditCard(
                Uuid::fromString('e4c5a064-8b01-498b-8761-a1ff10db0e8b'),
                new CreditCard()
            )
        );
    }
}
```

The PaymentMethod class represents the abstraction that must be refined and bridge the concrete payment providers.

Differences with other Design Patterns

Differences between Bridge pattern and others

- **Adapter** makes things work after they're designed; **Bridge** makes them work before they are. **Bridge** is designed up-front to let the abstraction and the implementation vary independently. **Adapter** is retrofitted to make unrelated classes work together.

- **State**, **Strategy**, **Bridge** (and to some degree **Adapter**) have similar solution structures. They all share elements of the "handle/body" idiom. They differ in intent - that is, they solve different problems.

Differences between Bridge pattern and others

The structure of **State** and **Bridge** are identical (except that **Bridge** admits hierarchies of envelope classes, whereas **State** allows only one). The two patterns use the same structure to solve different problems: **State** allows an object's behavior to change along with its state, while Bridge's intent is to decouple an abstraction from its implementation so that the two can vary independently.

If interface classes delegate the creation of their implementation classes (instead of creating/coupling themselves directly), then the design usually uses the **Abstract Factory** pattern to create the implementation objects.

Template Method

The Template Method pattern lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

Introduction

Following a protocol

In the civil aviation industry, passengers are required to check-in themselves and their luggages for the flights they're about to catch. The check-in protocol always includes the following steps:

1. **Finding** the passenger's booking with its booking number,
2. **Assigning** an available seat in the plane to the passenger,
3. **Checking** the passenger's luggages if he\she has any,
4. **Issuing** and printing his\her boarding pass.

Following a protocol

Nowadays, this check-in process can be achieved in many different ways. However, each available way to check-in a passenger onto a flight has to follow the same standardized « check-in » protocol.

1. **Airport check-in** with an airline employee
2. **Online check-in** on the airline website
3. **Online check-in** on the airline mobile application
4. **Automatic check-in** for national and regional flights

```php
abstract class FlightCheckIn implements CheckinServiceInterface
{
    public function checkin($bookingCode, Passenger $pax, Flight $flight)
    {
        if (!$booking = $this->retrievePassengerBooking($bookingCode)) {
            throw new BookingNotFoundException($bookingCode);
        }

        $bookedFlight = $booking->getBookedFlight();
        if (!$bookedFlight->equals($flight)) {
            throw new UnexpectedBookedFlightException($flight, $bookedFlight);
        }

        if (count($bags = $passenger->getBags())) {
            $this->checkPassengerBags($bags, $flight);
        }

        $seat = $this->assignSeat($pax, $flight);
        $boardingPass = $this->issueBoardingPass($booking, $pax, $flight, $seat);

        return $boardingPass;
    }
}
```

```php
abstract class FlightCheckIn implements CheckinServiceInterface
{
    /** @return Booking */
    private function retrievePassengerBooking($bookingCode)
    {
        // ...
    }

    /** @return Seat */
    private function assignSeat(Passenger $pax, Flight $flight)
    {
        // ...
    }

    /** @return BoardingPass */
    private function issueBoardingPass(Booking $b, Passenger $p, Flight $f, Seat $s)
    {
        // ...
    }

    /** @return CheckedBags */
    private function checkPassengerBags($bags, $flight)
    {
        // ...
    }
}
```

```
class AirportCheckIn extends FlightCheckIn
{
    // ...
}

class OnlineCheckIn extends FlightCheckIn
{
    // ...
}

class MobileCheckIn extends FlightCheckIn
{
    // ...
}

class AutomaticCheckIn extends FlightCheckIn
{
    // ...
}
```

What's wrong with the implementation?

In this example, the **FlightCheckIn** class has a public **checkin()** method and four private other methods for each steps of the check-in protocol.

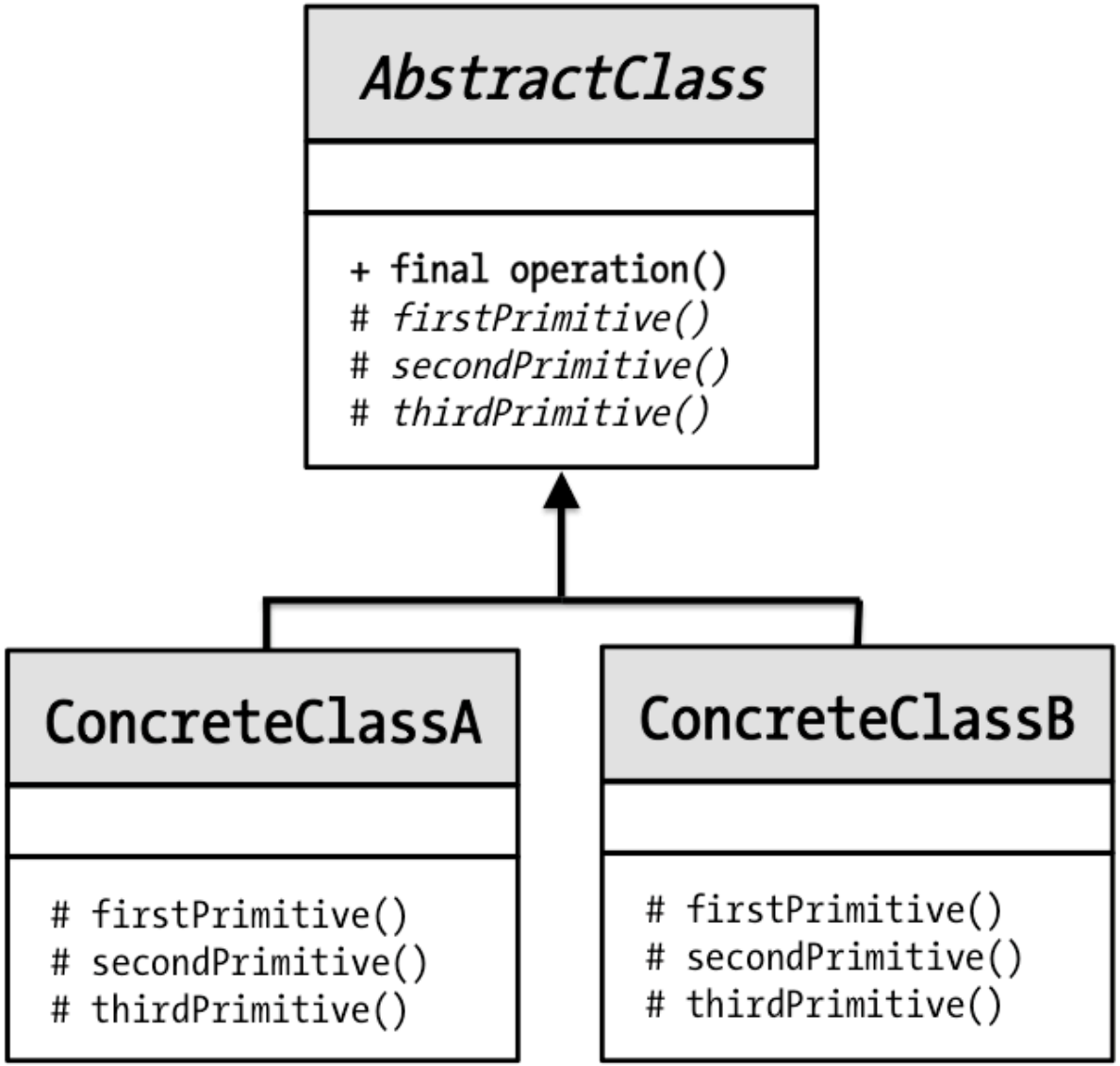
The main public **checkin()** method is public, which also means it may be fully overriden by a concrete class. This doesn't guarantee the protocol steps are always performed. Thus, the steps methods have been made private, which prevents them from being overriden by child concrete classes.

The Template Method Design Pattern

The Template Method Pattern

- **Behavioral pattern**

- Prevent an algorithm to be fully overriden by subclasses.
- Let subclasses override only some steps of an algorithm.
- Leverage the « Hollywood Principle ».


AbstractClass
+ final operation()
# firstPrimitive()
# secondPrimitive()
# thirdPrimitive()
ConcreteClassA
firstPrimitive()
secondPrimitive()
thirdPrimitive()
ConcreteClassB
firstPrimitive()
secondPrimitive()
thirdPrimitive()

```php
abstract class AbstractClass
{
    final public function operation()
    {
        $this->firstPrimitive();
        $this->secondPrimitive();

        return $this->thirdPrimitive();
    }

    abstract protected function firstPrimitive();
    abstract protected function secondPrimitive();
    abstract protected function thirdPrimitive();
}
```

The Hollywood Principle

http://sidvicious08.deviantart.com/art/Megaphone-31352732

```php
abstract class AbstractClass
{
    final public function operation()
    {
        $this->firstPrimitive();
        $this->secondPrimitive();

        return $this->thirdPrimitive();
    }
}
```

The **final public operation()** method is the only one responsible for calling the other **protected primitive** methods, which may be overriden by concrete subclasses.

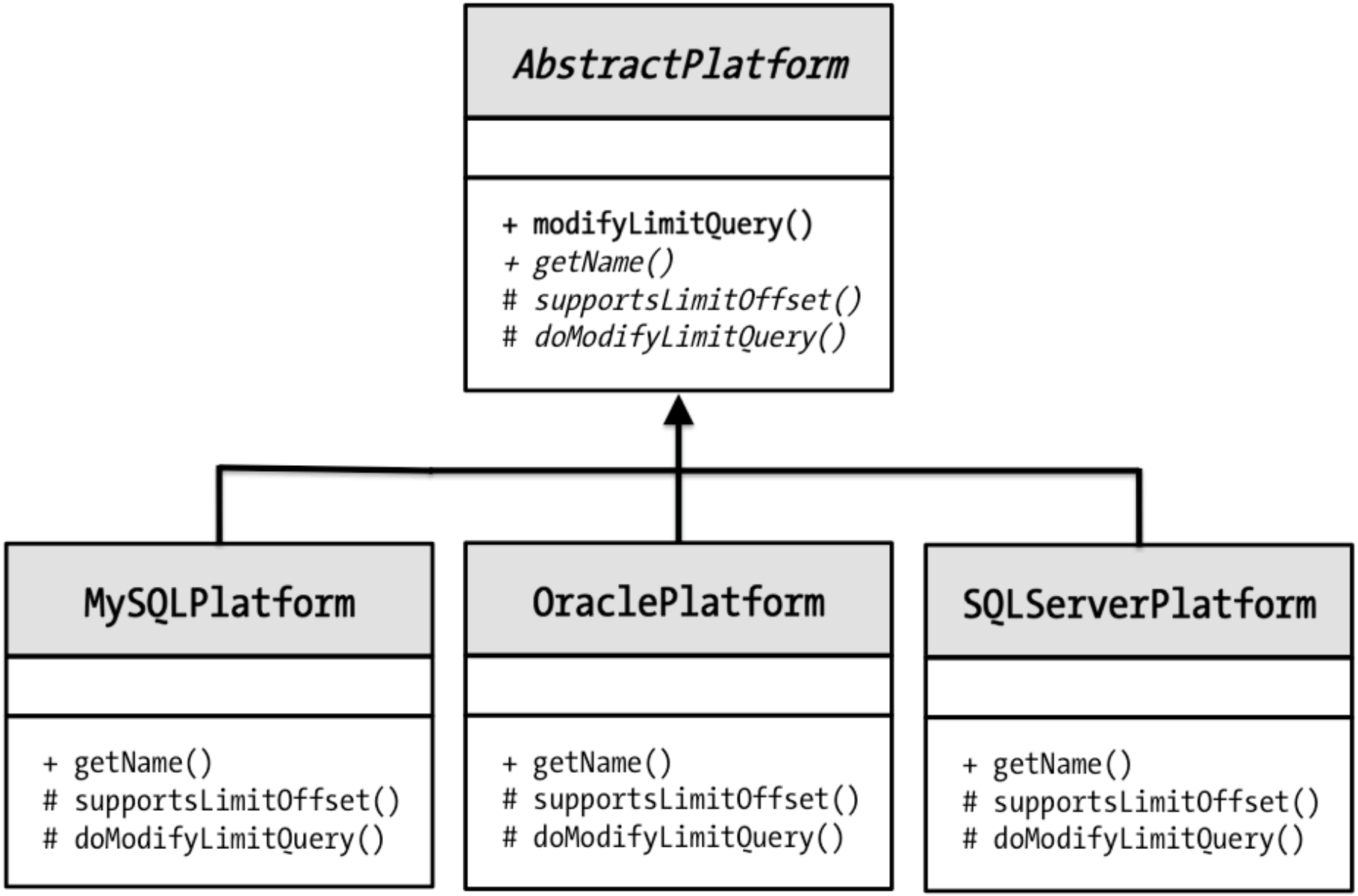
Implementing the Template Method Design Pattern

Paginating a SQL query in an ORM library

Object Relational Mapper libraries like Doctrine abstract the custom caracteristics of relational database vendors they support: MySQL, Mariadb, SQLite, Oracle, SQL Server, etc.

Although the SQL dialect is supposed to be standard and portable from a database engine to another, all vendors come with their custom implementation of SQL.

Some SQL keywords like **LIMIT** & **OFFSET** are not supported on every platform and must be converted to their counterparts.


AbstractPlatform
+ modifyLimitQuery()
+ getName()
# supportsLimitOffset()
# doModifyLimitQuery()
MySQLPlatform
+ getName()
# supportsLimitOffset()
# doModifyLimitQuery()
OraclePlatform
+ getName()
# supportsLimitOffset()
# doModifyLimitQuery()
SQLServerPlatform
+ getName()
# supportsLimitOffset()
# doModifyLimitQuery()

The Abstract Platform Class

```php
abstract class AbstractPlatform implements PlatformInterface
{
    /**
     * Appends the LIMIT clause to the SQL query.
     *
     * @param string $query  The SQL query to modify
     * @param int $limit  The max number of records to fetch
     * @param int $offset The offset from where to fetch records
     *
     * @return string The modified SQL query
     */
    final public function modifyLimitQuery($query, $limit, $offset = null)
    {
        // ...
    }

    abstract protected function doModifyLimitQuery($query, $limit, $offset);

    protected function supportsLimitOffset()
    {
        return true;
    }
}
```

```php
abstract class AbstractPlatform implements PlatformInterface
{
    final public function modifyLimitQuery($query, $limit, $offset = null)
    {
        if ($limit !== null) {
            $limit = (int) $limit;
        }

        if ($offset !== null) {
            $offset = (int) $offset;

            if ($offset < 0) {
                throw new PlatformException(sprintf(
                    'LIMIT offset must be greater or equal than 0, %u given.',
                    $offset
                ));
            }

            if ($offset > 0 && ! $this->supportsLimitOffset()) {
                throw new PlatformException(sprintf(
                    'Platform %s does not support offset values in limit queries.',
                    $this->getName()
                ));
            }
        }

        return $this->doModifyLimitQuery($query, $limit, $offset);
    }
}
```

```php
abstract class AbstractPlatform implements PlatformInterface
{
    final public function modifyLimitQuery($query, $limit, $offset = null)
    {
        if ($limit !== null) {
            $limit = (int) $limit;
        }

        if ($offset !== null) {
            $offset = (int) $offset;

            if ($offset < 0) {
                throw new PlatformException(sprintf(
                    'LIMIT offset must be greater or equal than 0, %u given.',
                    $offset
                ));
            }

            if ($offset > 0 && ! $this->supportsLimitOffset()) {
                throw new PlatformException(sprintf(
                    'Platform %s does not support offset values in limit queries.',
                    $this->getName()
                ));
            }
        }

        return $this->doModifyLimitQuery($query, $limit, $offset);
    }
}
```

The MySQL Platform Class

```php
class MySQLPlatform extends AbstractPlatform
{
    protected function doModifyLimitQuery($query, $limit, $offset)
    {
        if (null !== $limit) {
            $query .= ' LIMIT ' . $limit;
            if (null !== $offset) {
                $query .= ' OFFSET ' . $offset;
            }
        } elseif (null !== $offset) {
            $query .= ' LIMIT 18446744073709551615 OFFSET ' . $offset;
        }

        return $query;
    }

    public function getName()
    {
        return 'mysql';
    }
}
```

```php
$query = 'SELECT id, username FROM user';
$platform = new MySQLPlatform();
$platform->modifyLimitQuery($query, null);
$platform->modifyLimitQuery($query, 10);
$platform->modifyLimitQuery($query, 10, 50);
$platform->modifyLimitQuery($query, null, 50);
```

```
SELECT id, username FROM user
SELECT id, username FROM user LIMIT 10
SELECT id, username FROM user LIMIT 10 OFFSET 50
SELECT id, username FROM user LIMIT 18446744073709551615 OFFSET 50
```

The Oracle Platform Class

```
class OraclePlatform extends AbstractPlatform
{
    protected function doModifyLimitQuery($query, $limit, $offset = null)
    {
        if (!preg_match('/^\s*SELECT/i', $query)) {
            return $query;
        }

        if (!preg_match('/\sFROM\s/i', $query)) {
            $query .= ' FROM dual';
        }

        $limit  = (int) $limit;
        $offset = (int) $offset;

        if ($limit > 0) {
            $max = $offset + $limit;
            if ($offset > 0) {
                $min = $offset + 1;
                $query = sprintf(
                    'SELECT * FROM (SELECT a.*, ROWNUM AS dbal_rownum' .
                    ' FROM (%s) a WHERE ROWNUM <= %u) WHERE dbal_rownum >= %u)',
                    $query,
                    $max,
                    $min
                );
            } else {
                $query = sprintf('SELECT a.* FROM (%s) a WHERE ROWNUM <= %u', $query, $max);
            }
        }

        return $query;
    }
```

```php
$query = 'SELECT id, username FROM user';
$platform = new OraclePlatform();
$platform->modifyLimitQuery($query, null);
$platform->modifyLimitQuery($query, 10);
$platform->modifyLimitQuery($query, 10, 50);
```

```
SELECT id, username FROM user

SELECT a.* FROM (SELECT id, username FROM user) a WHERE ROWNUM <= 10

SELECT * FROM (SELECT a.*, ROWNUM AS dbal_rownum FROM (SELECT id,
username FROM user) a WHERE ROWNUM <= 60) WHERE dbal_rownum >= 51)
```

The MSSQL Platform Class

```php
abstract class SQLServerPlatform extends AbstractPlatform
{
    // ...

    public function supportsLimitOffset()
    {
        return false;
    }

    public function getName()
    {
        return 'mssql';
    }

    // ...
}
```

```php
class SQLServer2008Platform extends SQLServerPlatform
{
    public function supportsLimitOffset()
    {
        return true;
    }

    protected function doModifyLimitQuery($query, $limit, $offset = null)
    {
        if (null === $limit) {
            return $query;
        }

        $start = (int) $offset + 1;
        $end = (int) $offset + $limit;
        $orderBy = stristr($query, 'ORDER BY');

        // Remove ORDER BY from $query
        $query = preg_replace('/\s+ORDER\s+BY\s+([^\)]*)/', '', $query);
        $format = 'SELECT * FROM (%s) AS dbal_tbl WHERE dbal_rownum BETWEEN %d
AND %d';

        // ...
    }
}
```

```php
class SQLServer2008Platform extends SQLServerPlatform
{
    protected function doModifyLimitQuery($query, $limit, $offset = null)
    {
        // ...

        if (!$orderBy) {
            // Replace only first occurrence of FROM with OVER
            // to prevent changing FROM also in subqueries.
            $query = preg_replace(
                '/\sFROM\s/i', ', ROW_NUMBER() OVER (ORDER BY (SELECT 0)) AS
dbal_rownum FROM ',
                $query,
                1
            );

            return sprintf($format, $query, $start, $end);
        }

        // Clear ORDER BY
        $orderBy = preg_replace('/ORDER\s+BY\s+([^\)]*)(.*)/', '$1', $orderBy);
        $orderByParts = explode(',', $orderBy);
        $orderbyColumns = array();

        // ...
    }
}
```

```php
class SQLServer2008Platform extends SQLServerPlatform
{
    protected function doModifyLimitQuery($query, $limit, $offset = null)
    {
        // ...
        // Find alias for each colum used in ORDER BY
        $overColumns = [];
        if (!empty($orderbyColumns)) {
            foreach ($orderbyColumns as $col) {

                $pattern = sprintf(
                    '/%s\.(%s)\s*(AS)?\s*([^,\s\)]*)/i',
                    $col['table'],
                    $col['column']
                );

                $overColumn = preg_match($pattern, $query, $matches)
                    ? ($col['hasTable'] ? $col['table']  . '.' : '') . $col['column']
                    : $col['column'];

                if (isset($col['sort'])) {
                    $overColumn .= ' ' . $col['sort'];
                }

                $overColumns[] = $overColumn;
            }
        }
        // ...
    }
}
```

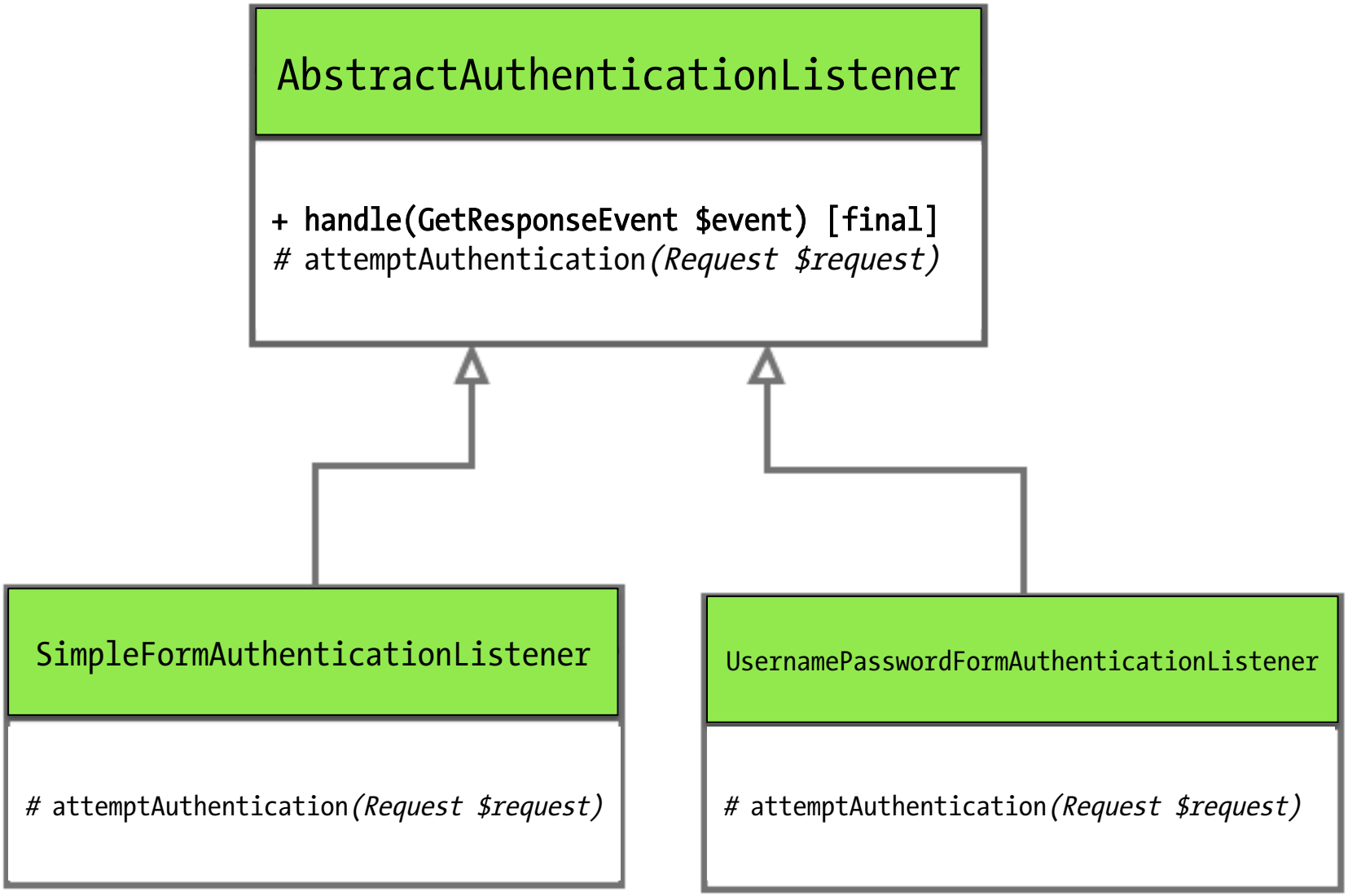
```
class SQLServer2008Platform extends SQLServerPlatform
{
    protected function doModifyLimitQuery($query, $limit, $offset = null)
    {
        // ...

        // Replace only first occurrence of FROM with $over
        // to prevent changing FROM also in subqueries.
        $over  = 'ORDER BY ' . implode(', ', $overColumns);
        $query = preg_replace('/\sFROM\s/i', ", ROW_NUMBER() OVER ($over) AS
dbal_rownum FROM ", $query, 1);

        return sprintf($format, $query, $start, $end);
    }
}
```

Finally! That was a big method override to support SQL query pagination on Microsoft SQL Server platforms.

```php
$query = 'SELECT id, username FROM user WHERE id >
170 ORDER BY username DESC';

$platform = new SQLServer2008Platform();
$platform->modifyLimitQuery($query, null);
$platform->modifyLimitQuery($query, 10);
$platform->modifyLimitQuery($query, 10, 50);
```

```
SELECT id, username FROM user WHERE id > 170 ORDER BY username DESC

SELECT * FROM (SELECT id, username, ROW_NUMBER() OVER (ORDER BY
username DESC) AS dbal_rownum FROM user WHERE id > 170) AS dbal_tbl
WHERE dbal_rownum BETWEEN 1 AND 10

SELECT * FROM (SELECT id, username, ROW_NUMBER() OVER (ORDER BY
username DESC) AS dbal_rownum FROM user WHERE id > 170) AS dbal_tbl
WHERE dbal_rownum BETWEEN 51 AND 60
```

Template Method in the Symfony framework


AbstractAuthenticationListener
+ handle(GetResponseEvent $event) [final]
# attemptAuthentication(Request $request)
SimpleFormAuthenticationListener
attemptAuthentication(Request $request)
UsernamePasswordFormAuthenticationListener
attemptAuthentication(Request $request)

```php
abstract class AbstractAuthenticationListener implements ListenerInterface
{
    final public function handle(GetResponseEvent $event)
    {
        // …
        try {
            // …
            $returnValue = $this->attemptAuthentication($request);
            if (null === $returnValue) {
                return;
            }
            // …
        } catch (AuthenticationException $e) {
            $response = $this->onFailure($event, $request, $e);
        }
        $event->setResponse($response);
    }

    abstract protected function attemptAuthentication(Request $request);
}
```

The Simple Form Authentication Listener

```php
namespace Symfony\Component\Security\Http\Firewall;

class SimpleFormAuthenticationListener extends AbstractAuthenticationListener
{
    protected function attemptAuthentication(Request $request)
    {
        // ...
        $token = $this->simpleAuthenticator->createToken(
            $request,
            trim($request->get('_username')),
            $request->get('_password')
        );

        return $this->authenticationManager->authenticate($token);
    }
}
```

Custom Authentication Listener

```
namespace Acme\SsoBundle\Security\Core\Authentication;

class SsoAuthenticationListener extends AbstractAuthenticationListener
{
    protected function attemptAuthentication(Request $request)
    {
        if (!$ssoToken = $request->query->get('ssotoken')) {
            return;
        }

        $token = new SSOToken($ssoToken);

        return $this->authenticationManager->authenticate($token);
    }
}
```

Strategy

The strategy pattern encapsulates algorithms of the same nature into dedicated classes to make them interchangeable.

Introduction

Identifying strategies

```php
class LocaleInitializer
{
    // ...

    public function initializeLocale(HttpRequest $request, $strategy = self::STRATEGY_PATH)
    {
        $locale = null;

        switch ($strategy) {
            case self::STRATEGY_PATH:
                preg_match('#^/([a-z]{2}([_-][a-zA-Z]{2})?)(?:/.+)?$#i', $request->getPath(), $matches);
                $locale = !empty($matches[1]) ? $matches[1] : null;
                break;
            case self::STRATEGY_COOKIE:
                $locale = $request->getCookie('lang');
                break;
            case self::STRATEGY_QUERY_STRING:
                $locale = $request->getQueryParameter('lang');
                break;
            case self::STRATEGY_DOMAIN_TLD:
                preg_match('#\.([a-z]{2})$#i', $request->getHostname(), $matches);
                $locale = !empty($matches[1]) ? $matches[1] : null;
                break;
            case self::STRATEGY_SUBDOMAIN:
                preg_match('#^([a-z]{2})\.#i', $request->getHostname(), $matches);
                $locale = !empty($matches[1]) ? $matches[1] : null;
                break;
            default:
                throw new \InvalidArgumentException(sprintf('Unsupported guessing strategy "%s".', $strategy));
        }

        \Locale::setDefault($this->findBestSupported($locale));
    }
```

```php
$initializer = new LocaleInitializer([ 'fr', 'en', 'de' ], 'en');

// Query string parameter strategy
$request = HttpRequest::create('GET', '/home?lang=fr');
$initializer->initializeLocale($request, LocaleInitializer::STRATEGY_QUERY_STRING);

// Prefixed path strategy
$request = HttpRequest::create('GET', '/fr/home');
$initializer->initializeLocale($request, LocaleInitializer::STRATEGY_PATH);

// Cookie header based strategy
$request = HttpRequest::create('GET', '/home');
$request->addHeader('Cookie', 'lang=fr');
$initializer->initializeLocale($request, LocaleInitializer::STRATEGY_COOKIE);

// Accept-Language header based strategy
$request = HttpRequest::create('GET', '/home');
$request->addHeader('Accept-Language', 'fr;q=1.0, en;q=0.6, de;q=0.4');
$initializer->initializeLocale($request, LocaleInitializer::STRATEGY_COOKIE);

// Domain name extension strategy
$request = HttpRequest::create('GET', 'http://wikipedia.fr/');
$initializer->initializeLocale($request, LocaleInitializer::STRATEGY_DOMAIN_TLD);

// Print guessed locale
echo \Locale::getDefault();
```

Problems with this code…

- The « switch/case » structure reduces code readability,
- Too many conditions leads to a higher class complexity,
- Higher complexity makes unit testing harder,
- Adding a new « strategy » implies to change the class,
- Class may have too many dependencies at some point.

The Need for a Strategy

Several ways to achieve the same goal

Which transportation system will be the best for a person who needs to travel from Paris to Lyon?

- **Airplane?** (short journey, expensive, airports outside downtown)
- **Train?** (short journey, city to city, affordable)
- **Car?** (long journey, not really affordable)
- **Bus or carpooling?** (long journey, cheap, many stops)
- **Boat?** (very slow, very long journey, ecological)
- **Bicycle?** (very slow, very cheap, very healthy, very ecological)

The chosen transportation mean won't be the same for every users profiles (business man, family of 4, athlete, etc.).

Several ways to achieve the same goal

Which banking solution is the best to save money and generate more interests?

- **Basic savings account?** (money is available, not risky, weak yield)
- **Life insurance?** (money not available, risky, better yield, long term)
- **Stock exchange?** (money not available, very risky, very good yield)

Again, the best option depends on every investors profiles and wishes. Someone with few savings will favor a basic savings account whereas someone with lots of money will invest it in riskier solutions.

Several ways to achieve the same goal

How should an airplane ticket be priced?

- **Fixed fare** for every seat?
- **Dynamic fare** depending on how full the aircraft is?
- **Pricing** depending on comfort (business class, economy, exit row, ...)?
- **Fare** based on ticket flexibility conditions?
- **Fare** based on the time period and events at destination?
- **Pricing** based on other competitors for the same destination?

Airlines have very complex strategies to price flight tickets and maximize the revenues of their flights. This is called the « *yield management* ».

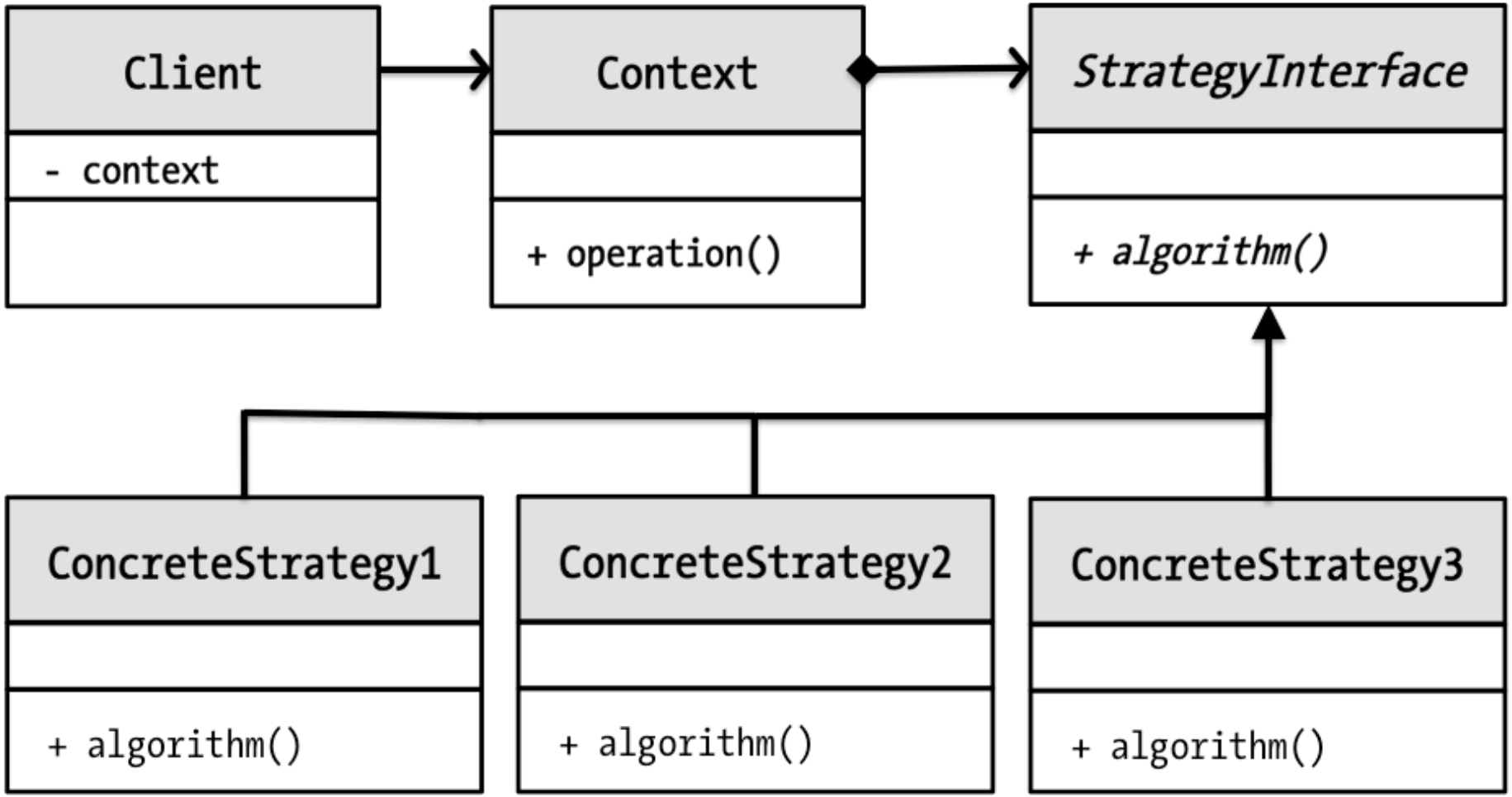
The Strategy Design Pattern

The Strategy Pattern

- **Behavioral pattern**

- Make algorithms of the same family interchangeable.
- Make the code open to future extensions.
- Make the code easier to unit test in isolation.

UML Diagram


Client
- context
Context
+ operation()
StrategyInterface
+ algorithm()
ConcreteStrategy1
+ algorithm()
ConcreteStrategy2
+ algorithm()
ConcreteStrategy3
+ algorithm()

Strategy Pattern Usage

```
$strategy1 = new ConcreteStrategy();
$strategy2 = new OtherConcreteStrategy();

$client = new Client($strategy1);
$client->doSomething();
$client->doSomethingElse($strategy2);
```

This client code delegates some parts of its algorithm to the **Strategy** object it receives in its construct method or into its methods. The behavior of the client code will change if an other **Strategy** instance is given to it.

The Strategy Interface

```
interface StrategyInterface
{
    public function algorithm();
}
```

The Client Code

```php
class Client
{
    private $strategy;

    public function __construct(StrategyInterface $strategy)
    {
        $this->strategy = $strategy;
    }

    public function doSomething()
    {
        $this->strategy->algorithm();

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

Use Cases for the Strategy Pattern

Uses Cases for the Strategy Pattern

- Sorting arrays or collections of objects,
- Compressing a data stream,
- Searching files with reduction criterias,
- Caching data into several kinds of datastores,
- Storing user session data in several datastores,
- Choosing the transport layer for an email,
- Detecting the best user's language,
- …

Session storage management

```
// Create a session storage
$storage = new NativeSessionStorage('PHPSESSID');
$storage = new NullSessionStorage();
$storage = new RedisSessionStorage();
$storage = new MemcachedSessionStorage();
$storage = new ApcSessionStorage();
$storage = new MySQLSessionStorage(new \PDO('...'));

// ...

// Make the shopping cart rely on the session storage
$cart = new ShoppingCart($storage);
$cart->addReference('ITEM-123-90', 2, 25.90);
$cart->addReference('ITEM-539-22', 1, 139.90);
```

Concrete Use Case for the Strategy Pattern

Guessing the user's locale

A web application guesses the locale for the current client based on the HTTP request information. There are several ways (strategies) to guess the user's locale:

- **Query string** parameter (`?lang=fr`)
- **Cookie** (`SetCookie: lang=fr`)
- **Accept-Language** header (`Accept-Language: fr`)
- **Domain name** (google.fr, google.de, etc.)
- **Subdomain name** (en.domain.com, fr.domain.com)
- **Prefix path** (/fr/home.html, /en/home.html, etc.)
- **Geo IP address**

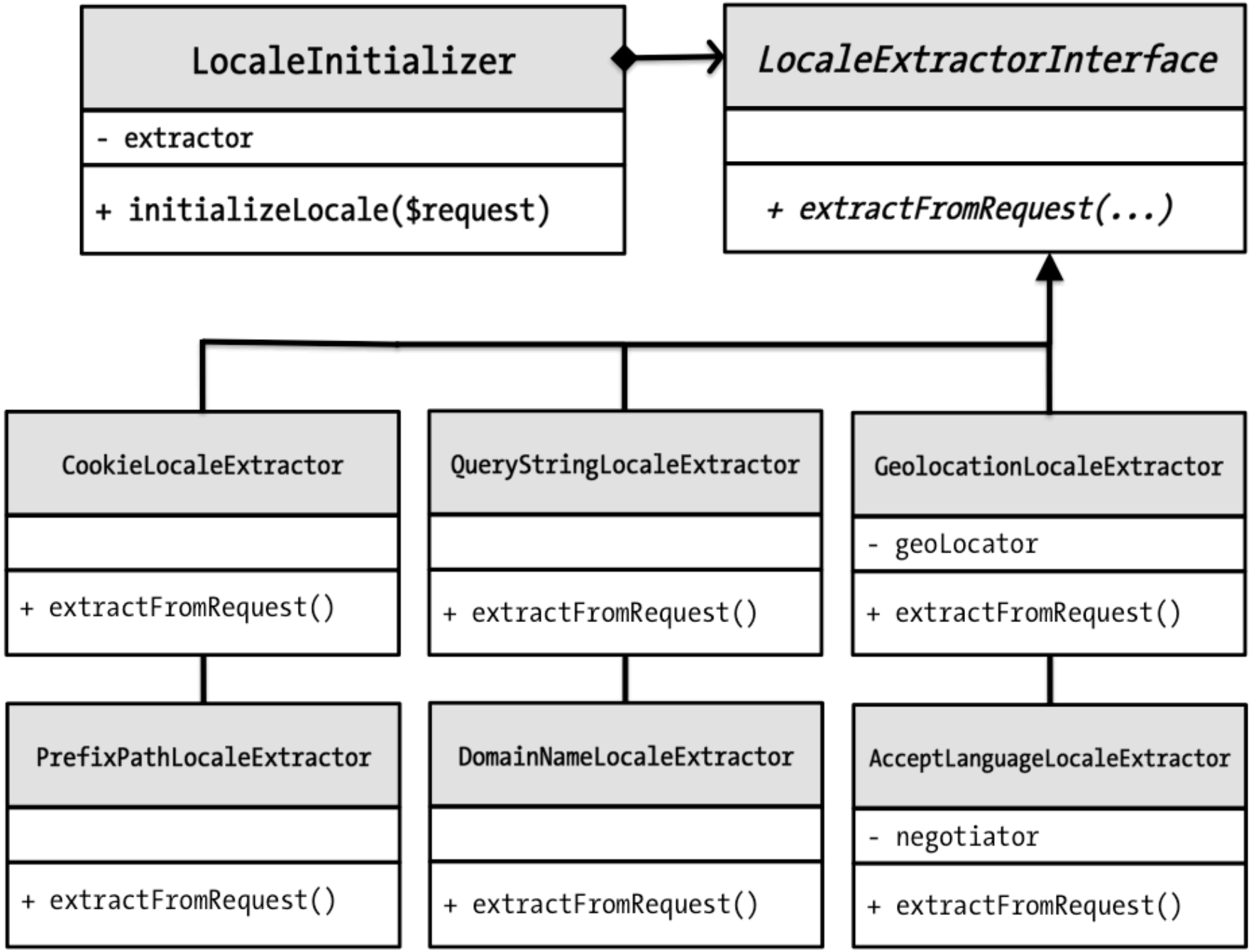
Implementing the Strategy Design Pattern

The LocaleInitializer example

Implementing the **Strategy** design pattern in the previous example of the **LocaleInitializer** class is as simple as breaking the long « switch/case » down into multiple smaller classes.

Each « **case** » code section is encapsulated into a smaller **Strategy** class, so that the **LocaleInitializer** class can depend on an abstract strategy type.

This way, adding a new concrete strategy only requires to create a new implementation of the abstract strategy without changing the client code or the other existing concrete strategies.


LocaleInitializer
- extractor
+ initializeLocale($request)
LocaleExtractorInterface
+ extractFromRequest(...)
CookieLocaleExtractor
+ extractFromRequest()
QueryStringLocaleExtractor
+ extractFromRequest()
GeolocationLocaleExtractor
- geoLocator
+ extractFromRequest()
PrefixPathLocaleExtractor
+ extractFromRequest()
DomainNameLocaleExtractor
+ extractFromRequest()
AcceptLanguageLocaleExtractor
- negotiator
+ extractFromRequest()

#1. Extracting the Strategy from the Client code

The LocaleExtractorInterface aka the Strategy

```
interface LocaleExtractorInterface
{
    /**
     * Extracts the locale based from an HTTP request object.
     *
     * @param HttpRequest $request The HTTP request object
     * @param array       $locales The supported locales
     *
     * @return string The extracted locale
     *
     * @throws LocaleException When not able to extract the locale
     */
    function extractFromRequest(HttpRequest $request, array $locales);
}
```

The LocaleInitializer class aka the client code

```
class LocaleInitializer
{
    private $extractor;
    private $supportedLocales;
    private $fallbackLocale;

    public function __construct(
        LocaleExtractorInterface $extractor,
        array $supportedLocales,
        $fallbackLocale
    )
    {
        $this->extractor = $extractor;
        $this->supportedLocales = $supportedLocales;
        $this->fallbackLocale = $fallbackLocale;
    }

    // ...
}
```

The LocaleInitializer class aka the client code

```php
class LocaleInitializer
{
    // ...

    private function getSupportedLocales()
    {
        $locales = [];
        foreach ($this->supportedLocales as $locale) {
            if (false !== strpos($locale, '_')) {
                $locales[] = str_replace('_', '-', $locale);
            }
        }

        return array_unique(array_merge($this->supportedLocales, $locales));
    }
}
```

```php
class LocaleInitializer
{
    // ...
    private function findBestSupported($locale)
    {
        $locale = str_replace('-', '_', $locale);

        // 1st outcome: locale is already supported, so take it!
        if (in_array($locale, $this->supportedLocales)) {
            return $locale;
        }

        // 2nd outcome: is there a locale_LOCALE locale available?
        $naiveLocale = sprintf('%s_%s', strtolower($locale), strtoupper($locale));
        if (in_array($naiveLocale, $this->supportedLocales)) {
            return $naiveLocale;
        }

        // 3rd outcome: find the best closest? (fr_FR becomes fr)
        $locale = substr($locale, 0, 2);
        foreach ($this->supportedLocales as $supportedLocale) {
            $supportedLocale = substr($supportedLocale, 0, 2);
            if ($locale === $supportedLocale) {
                return $locale;
            }
        }

        // 4th outcome: take the default locale if none matches
        return $this->fallbackLocale;
    }
}
```

Delegating locale guessing to the strategy

```
class LocaleInitializer
{
    // ...

    public function initializeLocale(HttpRequest $request)
    {
        $locales = $this->getSupportedLocales();

        try {
            $locale = $this->extractor->extractFromRequest($request, $locales);
            $locale = $this->findBestSupported($locale);
        } catch (\Exception $e) {
            $locale = $this->fallbackLocale;
        }

        \Locale::setDefault($locale);
    }

    // ...
}
```

Benefits of the LocaleInitializer class refactoring

1. Client code now depends on an interface,
2. Client code is loosely coupled to the strategy,
3. Strategy implementation can easily be changed,
4. Client code is simplier, smaller and easier to unit test,
5. Full SOLID principles compliance.

#2.
Query string parameter locale extraction strategy.

```php
class QueryStringLocaleExtractor implements LocaleExtractorInterface
{
    function extractFromRequest(HttpRequest $request, array $locales)
    {
        if (!$locale = $request->getQueryParameter('lang')) {
            throw new LocaleException('...');
        }

        return $locale;
    }
}
```

```php
$locales = [ 'fr', 'de', 'en' ];
$extractor = new QueryStringLocaleExtractor();
$request = HttpRequest::create('GET /home?lang=de HTTP/1.1');

$initializer = new LocaleInitializer($extractor, $locales, 'en');
$initializer->initializeLocale($request);
```

#3.
Cookie locale extraction strategy.

```php
class CookieLocaleExtractor implements LocaleExtractorInterface
{
    function extractFromRequest(HttpRequest $request, array $locales)
    {
        if (!$locale = $request->getCookieParameter('lang')) {
            throw new LocaleException('...');
        }

        return $locale;
    }
}
```

```php
$locales = [ 'fr', 'de', 'en' ];
$request = HttpRequest::create('GET /home HTTP/1.1');
$request->addHeader('Cookie: lang=fr');

$extractor = new CookieLocaleExtractor();
$initializer = new LocaleInitializer($extractor, $locales, 'en');
$initializer->initializeLocale($request);
```

#4.
Prefix path locale extraction strategy.

```php
class PrefixPathLocaleExtractor implements LocaleExtractorInterface
{
    const PATTERN = '#^/([a-z]{2}([_-][a-zA-Z]{2})?)(?:/.+)?$#i';

    public function extractFromRequest(HttpRequest $request, array $locales)
    {
        if (!preg_match(self::PATTERN, $request->getPath(), $matches)) {
            throw new LocaleException(sprintf(
                'Unable to extract locale from the resource path "%s". ',
                $request->getPath()
            ));
        }

        return $matches[1];
    }
}
```

```php
$locales = [ 'fr', 'de', 'en' ];
$request = HttpRequest::create('GET /fr/home HTTP/1.1');

$extractor = new PrefixPathLocaleExtractor();
$initializer = new LocaleInitializer($extractor, $locales, 'en');
$initializer->initializeLocale($request);
```

#5.
Domain name extraction strategy.

```php
class DomainNameLocaleExtractor implements LocaleExtractorInterface
{
    const PATTERN = '#\.([a-z]{2})$#i';

    public function extractFromRequest(HttpRequest $request, array $locales)
    {
        if (!preg_match(self::PATTERN, $request->getHostname(), $matches)) {
            throw new LocaleException('...');
        }

        return $matches[1];
    }
}
```

```php
$locales = [ 'fr', 'de', 'en' ];
$request = HttpRequest::create('GET /home HTTP/1.1');
$request->addHeader('Host: www.some-domain.fr');

$extractor = new DomainNameLocaleExtractor();
$initializer = new LocaleInitializer($extractor, $locales, 'en');
$initializer->initializeLocale($request);
```

#6.
Subdomain name extraction strategy.

```php
class SubDomainNameLocaleExtractor implements LocaleExtractorInterface
{
    const PATTERN = '#^([a-z]{2})\.#i';

    public function extractFromRequest(HttpRequest $request, array $locales)
    {
        if (!preg_match(self::PATTERN, $request->getHostname(), $matches)) {
            throw new LocaleException('...');
        }

        return $matches[1];
    }
}
```

```php
$locales = [ 'fr', 'de', 'en' ];
$request = HttpRequest::create('GET /home HTTP/1.1');
$request->addHeader('Host: fr.some-domain.org');

$extractor = new SubDomainNameLocaleExtractor();
$initializer = new LocaleInitializer($extractor, $locales, 'en');
$initializer->initializeLocale($request);
```

#7. Accept-Language HTTP header extraction strategy.

```php
class AcceptLanguageLocaleExtractor implements LocaleExtractorInterface
{
    private $negotiator;

    function __construct(\Negotiation\LanguageNegotiator $negotiator)
    {
        $this->negotiator = $negotiator;
    }

    function extractFromRequest(HttpRequest $request, array $locales)
    {
        if (!$header = $request->getHeader('Accept-Language')) {
            throw new LocaleException('...');
        }

        if (!$best = $this->negotiator->getBest($header, $locales)) {
            throw new LocaleException('...');
        }

        return $best->getValue();
    }
}
```

```php
$locales = [ 'fr', 'de', 'en' ];

$request = HttpRequest::create('GET /home HTTP/1.1');
$request->addHeader('Accept-Language', 'en;q=0.3,de;q=1.0');

$negotiator = new Negotiation\LanguageNegotiator();
$extractor = new AcceptLanguageLocaleExtractor($negotiator);
$initializer = new LocaleInitializer($extractor, $locales, 'en');
$initializer->initializeLocale($request);
```

The **LanguageNegotiator** class comes from a third party Composer package called **willdurand/negotiation** and developped by William Durand. This package provides a set of classes to ease HTTP content negotiation.

#8.
Geolocation extraction strategy.

```php
class GeolocationLocaleExtractor implements LocaleExtractorInterface
{
    private static $countries = [
        'FR' => [ 'fr-FR', 'fr' ],
        'US' => [ 'en-US', 'en' ],
        'UK' => [ 'en-GB', 'en' ],
        'DE' => [ 'de-DE', 'de' ],
        'AT' => [ 'de-AT', 'de' ],
        // ...
    ];

    private $locator;

    function __construct(\GeoIp2\Database\Reader $reader)
    {
        $this->locator = $reader;
    }

    private function createLocaleException(HttpRequest $request, \Exception $e = null)
    {
        $message = sprintf('Unable to geolocate IP: %s.', $request->getClientIp());

        return new LocaleException($message, $e);
    }
}
```

```php
class GeolocationLocaleExtractor implements LocaleExtractorInterface
{
    // ...

    private function findBestByCounty($countryCode, array $locales)
    {
        if (empty(static::$countries[$countryCode])) {
            return null;
        }

        foreach (static::$countries[$countryCode] as $locale) {
            if (in_array($locale, $locales)) {
                return $locale;
            }
        }
    }

    function extractFromRequest(HttpRequest $request, array $locales)
    {
        // ...
    }
}
```

```php
class GeolocationLocaleExtractor implements LocaleExtractorInterface
{
    // ...

    function extractFromRequest(HttpRequest $request, array $locales)
    {
        try {
            $data = $this->locator->country($request->getClientIp());
        } catch (\Exception $e) {
            throw $this->createLocaleException($request, $e);
        }

        $country = $data->country->isoCode;
        $locale = $this->findBestByCounty($country, $locales);
        if (!$locale) {
            throw $this->createLocaleException($request);
        }

        return $locale;
    }
}
```

```php
$_SERVER['REMOTE_ADDR'] = '162.221.145.28';

$locales = [ 'fr', 'de', 'en' ];

$request = HttpRequest::create('GET /home HTTP/1.1');
new \GeoIp2\Database\Reader('GeoLite2-Country.mmdb');
$locator =
$extractor = new GeolocationLocaleExtractor($locator);
$initializer = new LocaleInitializer($extractor, $locales, 'en');
$initializer->initializeLocale($request);
```

The **Reader** class and the **Geolite2-Country.mmdb** database file come from a third party Composer package called **geoip2/geoip2**. This package provides a database of localized IP addresses to ease the detection of a country and locale based on an client IP address.

Strategy variants

Passing the strategy at each client code method call

The actual implementation of the **LocaleInitializer** class may have a lacking point. The concrete **strategy** is indeed always passed to its construct method and thus cannot be changed. For more flexibility, the concrete **strategy** can be passed at each client code method call.

```
class LocaleInitializer
{
    // ...

    function initializeLocale(HttpRequest $request, LocaleExtractorInterface $le)
    {
        // ...
    }
}
```

Default strategy can be overriden by method call

Another design option is to force a default concrete strategy in the construct method of the client code. Thus, the default concrete strategy can be explicitly overriden by passing another concrete implementation as a method call argument.

```
class LocaleInitializer
{
    private $defaultExtractor;

    function __construct(LocaleExtractorInterface $extractor)
    {
        $this->defaultExtractor = $extractor;
    }

    function initializeLocale(..., LocaleExtractorInterface $extractor = null)
    {
        $extractor = $extractor ?: $this->defaultExtractor;
    }
}
```

Construct method can accept a list of strategies

A third option is to pass an array of concrete strategies to the **LocaleInitializer** class constructor method. Then, the main client code method call will return the first locale that is successfully extracted by one of the concrete **LocaleExtractorInterface** strategies. This implementation enables to decide in which order strategies are run.

```
class LocaleInitializer
{
    // ...
    private $extractors;

    public function __construct(array $extractors, ...)
    {
        // This method checks extractors implement the strategy interface.
        $this->setExtractors($extractors);
    }
}
```

```php
class LocaleInitializer
{
    // ...
    public function initializeLocale(HttpRequest $request)
    {
        $i = 0;
        $locale = null;
        $locales = $this->getSupportedLocales();
        while (null === $locale && isset($this->extractors[$i])) {
            $extractor = $this->extractors[$i];
            $locale = $this->tryExtractLocale($extractor, $request, $locales);
            $i++;
        }
        \Locale::setDefault($locale ?: $this->fallbackLocale);
    }

    private function tryExtractLocale(
        LocaleExtractorInterface $extractor,
        HttpRequest $request,
        array $locales
    )
    {
        try {
            $locale = $extractor->extractFromRequest($request, $locales);
            $locale = $this->findBestSupported($locale);
        } catch (\Exception $e) {
            $locale = null;
        }

        return $locale;
    }
}
```

Construct method can accept a list of strategies

```
$locales = [ 'fr', 'de', 'en' ];
$negotiator = new \Negotiation\LanguageNegotiator();
$locator = new \GeoIp2\Database\Reader('GeoLite2-Country.mmdb');

$extractors = [
    new AcceptLanguageLocaleExtractor($negotiator),
    new QueryStringLocaleExtractor(),
    new CookieLocaleExtractor(),
    new PrefixPathLocaleExtractor(),
    new DomainNameLocaleExtractor(),
    new SubDomainNameLocaleExtractor(),
    new GeolocationLocaleExtractor($locator),
];

$request = HttpRequest::create('GET /home?lang=de HTTP/1.1');
$initializer = new LocaleInitializer($extractors, $locales, 'en');
$initializer->initializeLocale($request);
```

Composite Strategy

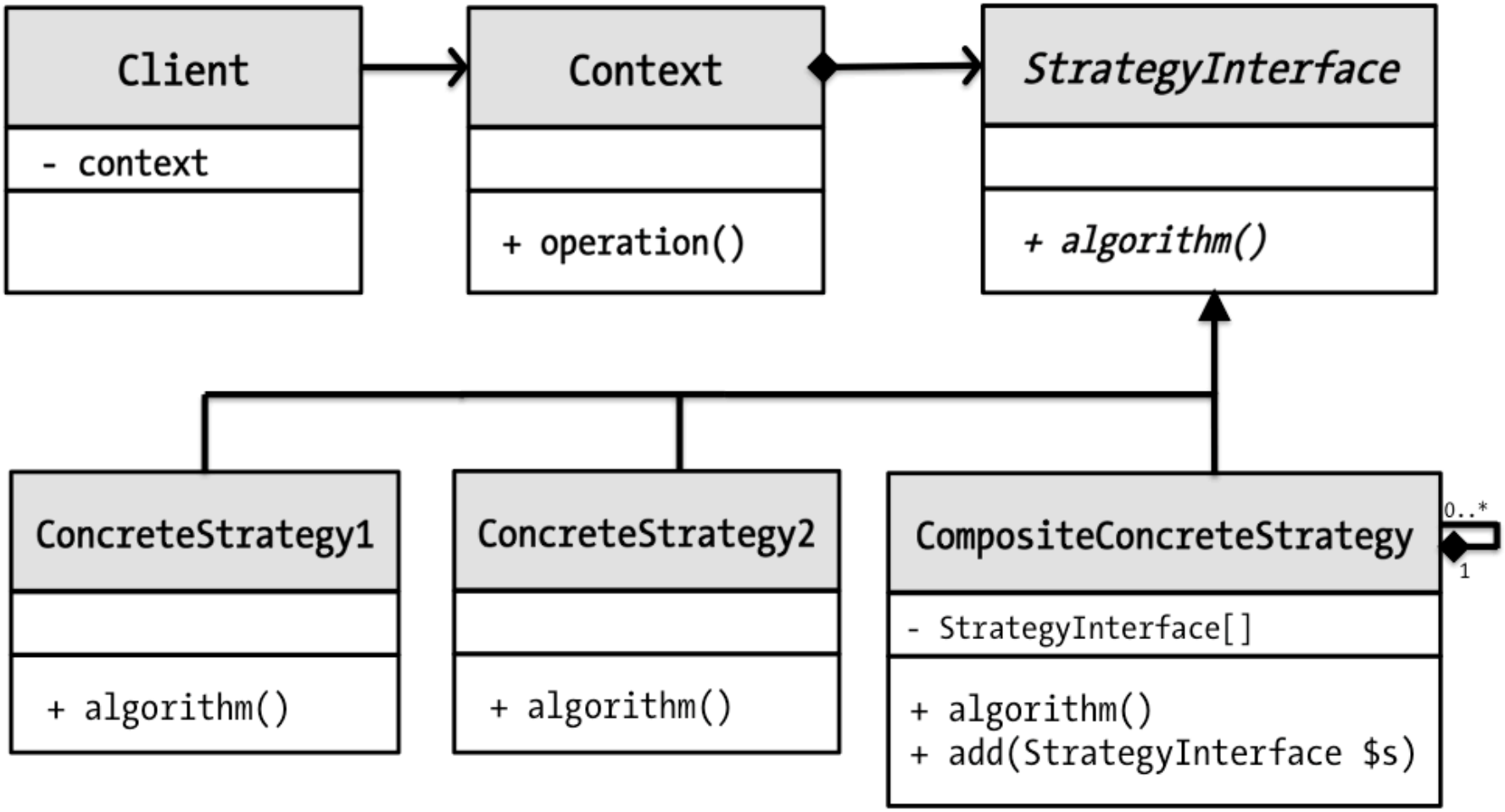
Combining strategies into a Composite object

The previous example was in fact lacking a major drawback! The client code had to be changed to support a chain of concrete locale extractors. The best option to avoid changing the working locale initializer class is to add a new **CompositeLocaleExtractor** concrete strategy class.

The structure of this new class will leverage the **Composite** structural design pattern, thus being employed as a concrete **Strategy** pattern implementation for the **LocaleInitializer** client code class.

Combining patterns together enables to create **greater** and **more powerful** design pattern implementations.

Combining strategies into a Composite object



```php
class CompositeLocaleExtractor implements LocaleExtractorInterface
{
    const PREPEND = 0;
    const APPEND = 1;

    /**
     * A collection of LocaleExtractorInterface instances.
     *
     * @var LocaleExtractorInterface[]
     */
    private $extractors;

    public function __construct(array $extractors)
    {
        $this->extractors = [];
        foreach ($extractors as $extractor) {
            $this->append($extractor);
        }
    }
}
```

```php
class CompositeLocaleExtractor implements LocaleExtractorInterface
{
    public function append(LocaleExtractorInterface $extractor)
    {
        return $this->add($extractor, self::APPEND);
    }

    public function prepend(LocaleExtractorInterface $extractor)
    {
        return $this->add($extractor, self::PREPEND);
    }

    private function add(LocaleExtractorInterface $extractor, $position)
    {
        if (in_array($extractor, $this->extractors)) {
            return $this;
        }

        $func = (self::APPEND === $position) ? 'array_push' : 'array_unshift';
        call_user_func_array($func, [&$this->extractors, $extractor]);

        return $this;
    }
}
```

```php
class CompositeLocaleExtractor implements LocaleExtractorInterface
{
    public function extractFromRequest(HttpRequest $request, array $locales)
    {
        foreach ($this->extractors as $extractor) {
            if ($locale = $this->tryExtractLocale($extractor, $request, $locales)) {
                return $locale;
            }
        }

        throw new LocaleException('Unable to extract any locale from any extractors.');
    }

    private function tryExtractLocale(LocaleExtractorInterface $extractor, ...)
    {
        $locale = null;
        try {
            $locale = $extractor->extractFromRequest($request, $locales);
        } catch (LocaleException $e) {
            // Skip and try the next one!
        } catch (\Exception $e) {
            // Something bad happened!
            throw new LocaleException('Unable to extract locale...', $e);
        }

        return $locale;
    }
}
```

```php
$reader = new Reader('GeoLite2-Country.mmdb');
$negotiator = new LanguageNegotiator();
$locales = [ 'en', 'fr', 'de' ];

// 1. Setup locale extractor
$extractor = new CompositeLocaleExtractor();
$extractor->append(new AcceptLanguageLocaleExtractor($negotiator));
$extractor->append(new PrefixPathLocaleExtractor());
$extractor->append(new QueryStringLocaleExtractor());
$extractor->append(new CookieLocaleExtractor());
$extractor->append(new GeolocationLocaleExtractor($reader));
$extractor->append(new DomainNameLocaleExtractor());
$extractor->append(new SubDomainNameLocaleExtractor());

// 2. Try to guess and fix the best client's locale
$initializer = new LocaleInitializer($extractor, $locales, 'en');
$initializer->initializeLocale(HttpRequest::createFromGlobals());

echo \Locale::getDefault();
```

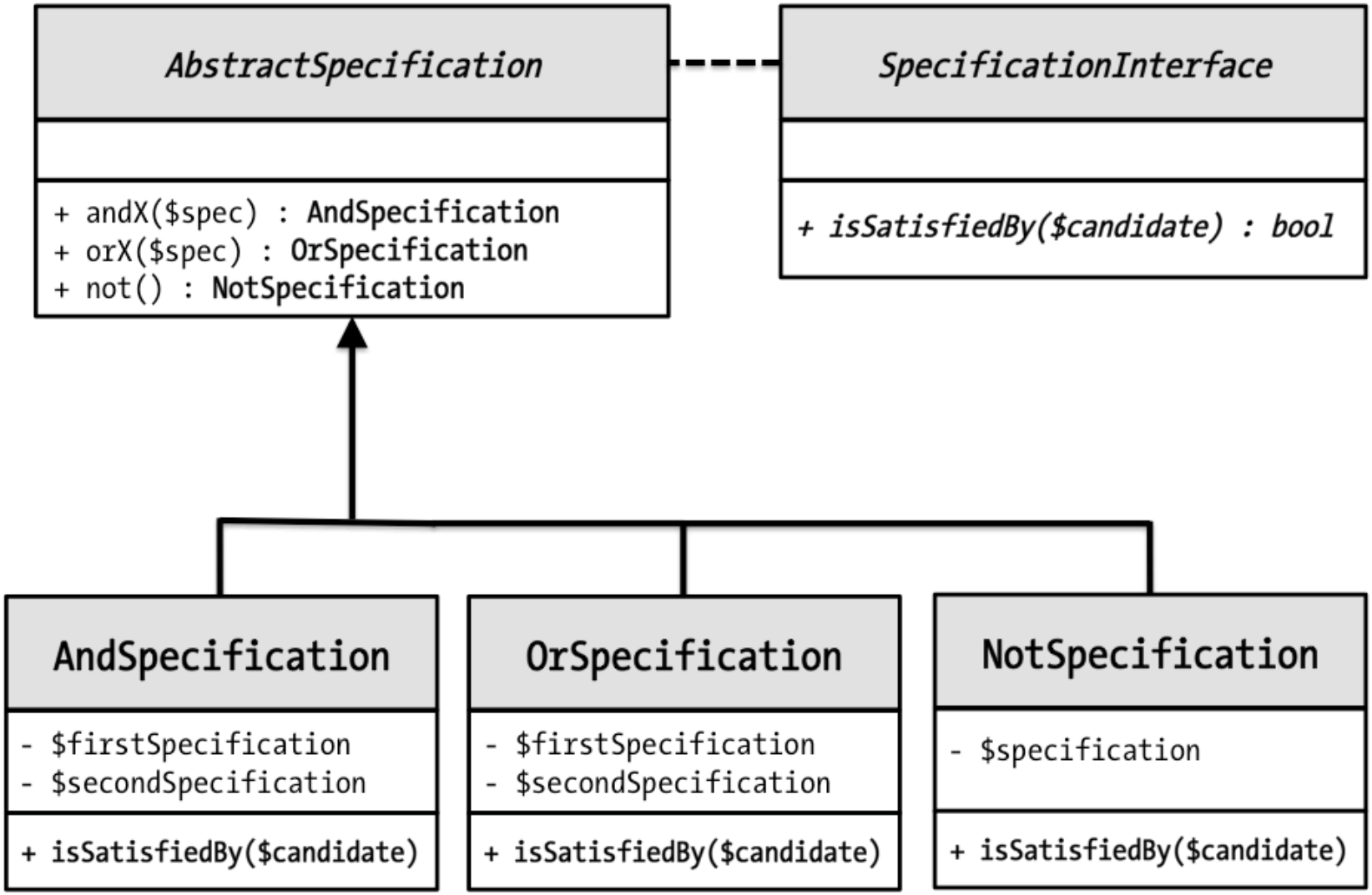
The Specification Pattern

The Specification Pattern

The **Strategy** design pattern is also often implemented as the **Specification** pattern. The **Specification** pattern allows to check if a candidate object satisfies a specific business rule encapsulated as a **Strategy** object.

More complex logical **specifications** can be easily defined by implementing the **Composite** pattern. This leads to a super pattern using **Composite** for the structure and **Strategy** for the behavior.

UML Diagram of the Specification Pattern



Real use case example

```
$candidate = new Candidate('DP2943074');
$candidate->addExamResult(new ExamResult('2015-10-12', 30));
$candidate->addExamResult(new ExamResult('2016-03-07', 37));
$candidate->addExamResult(new ExamResult('2016-04-22', 48));

$repository = new AuthorizationRepository($dbConnection);

$satisfies = (new CandidateIsAnObject())
  ->andX((new CandidateIsExpertCertified())->not())
  ->andX((new CandidateHasAlreadySatExamTwiceThisYear())->not())
  ->andX((new CandidateIsAlreadyAuthorizedToSeatExam($repository))->not())
  ->andX(new CandidateMustWaitOneWeekBetweenTwoExamSessions())
  ->isSatisfiedBy($candidate)
;
```

Iterator

The iterator pattern allows to traverse a container object and access its inner elements.

Introduction

Why using iterators?

Iterators are objects encapsulating list of information like arrays, lists or objects collections. They also provide a mean to traverse their inner list of data without letting the client code know how to do it. This way, this process can be changed in the iterator without changing the client code.

Iterators are also easily combinable with structural design patterns like **proxy**, **decorator** or **composite** to create more advanced and powerful implementations.

Examples

- Reading a file content line by line
- Recursively reading the content of directory
- Sorting items in a collection
- Applying filters on items of a collection
- Lazy loading data from a datastore into memory
- Implementing cursors for database result sets
- …

The Iterator Design Pattern

The Iterator interface

```
interface Iterator extends \Traversable
{
    public function current();
    public function next();
    public function rewind();
    public function valid();
    public function key();
}
```

The IteratorAggregate interface

```
interface IteratorAggregate extends \Traversable
{
    /**
     * Returns a traversable object.
     *
     * @return \Traversable
     */
    public function getIterator();
}
```

Implementing the Iterator Design Pattern

```php
class RouteCollection
{
    private $routes;

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

    function merge(RouteCollection $routes, $override = false)
    {
        if ($routes === $this) {
            throw new \LogicException('A routes collection...');
        }

        foreach ($routes as $name => $route) {
            $this->add($name, $route, $override);
        }
    }
}
```

```php
class RouteCollection
{
    // ...

    public function all()
    {
        return $this->routes;
    }

    public function add($name, Route $route, $override = false)
    {
        if (isset($this->routes[$name]) && !$override) {
            throw new \InvalidArgumentException(sprintf(
                'A route already exists for the name "%s".',
                $name
            ));
        }

        $this->routes[$name] = $route;
    }
}
```

```php
class RouteCollection implements \Iterator
{
    public function current()
    {
        return current($this->routes);
    }

    public function next()
    {
        next($this->routes);
    }

    public function key()
    {
        return key($this->routes);
    }

    public function valid()
    {
        return $this->current() instanceof Route;
    }

    public function rewind()
    {
        reset($this->routes);
    }
}
```

Iterating the RouteCollection object

```
$routes = new RouteCollection();
$routes->add('home', new Route('/', HomepageAction::class));
$routes->add('hello', new Route('/hello', HelloAction::class));
$routes->add('contact', new Route('/contact', ContactAction::class));

while ($routes->valid()) {
    $route = $routes->current();
    $name = $routes->key();
    // ...
    $routes->next();
}

foreach ($routes as $name => $route) {
    echo $route->getPath();
}
```

The Built-in PHP/SPL Iterators

AppendIterator, **ArrayIterator**, CachingIterator, CallbackFilterIterator, **DirectoryIterator**, EmptyIterator, FilesystemIterator, FilterIterator, **GlobIterator**, InfiniteIterator, IteratorIterator, LimitIterator, MultipleIterator, NoRewindIterator, ParentIterator, **RecursiveArrayIterator**, RecursiveCachingIterator, RecursiveCallbackFilterIterator, **RecursiveDirectoryIterator**, RecursiveFilterIterator, **RecursiveIteratorIterator**, RecursiveRegexIterator, RecursiveTreeIterator, RegexIterator

Real World Iterator Examples

The Symfony Finder

```
$iterator = Finder::create()
    ->files()
    ->name('*.php')
    ->depth(0)
    ->size('>= 1K')
    ->in(__DIR__);

foreach ($iterator as $file) {
    print $file->getRealpath()."\n";
}
```

```
├─── CustomFilterIterator.php
├─── DateRangeFilterIterator.php
├─── DepthRangeFilterIterator.php
├─── ExcludeDirectoryFilterIterator.php
├─── FilePathsIterator.php
├─── FileTypeFilterIterator.php
├─── FilecontentFilterIterator.php
├─── FilenameFilterIterator.php
├─── FilterIterator.php
├─── MultiplePcreFilterIterator.php
├─── PathFilterIterator.php
├─── RecursiveDirectoryIterator.php
├─── SizeRangeFilterIterator.php
└─── SortableIterator.php
```

Sorting a list of files

```
use Symfony\Component\Finder\Iterator\SortableIterator;
use Symfony\Component\Finder\Iterator\RecursiveDirectoryIterator;

$dirIterator = new \RecursiveIteratorIterator(
    new RecursiveDirectoryIterator(
        __DIR__,
        \RecursiveDirectoryIterator::SKIP_DOTS
    )
);
$dirIterator->setMaxDepth(0);

$iterator = new SortableIterator(
    $dirIterator,
    SortableIterator::SORT_BY_NAME
);
```

PHP Finder Adapter

```php
class PhpAdapter extends AbstractAdapter
{
    public function searchInDirectory($dir)
    {
        $iterator = new \RecursiveIteratorIterator(...);

        if ($this->minDepth > 0 || $this->maxDepth < PHP_INT_MAX) {
            $iterator = new DepthRangeFilterIterator($iterator, $this->minDepth, $this->maxDepth);
        }

        if ($this->mode) {
            $iterator = new FileTypeFilterIterator($iterator, $this->mode);
        }

        if ($this->exclude) {
            $iterator = new ExcludeDirectoryFilterIterator($iterator, $this->exclude);
        }

        // ... Other iterators are added

        return $iterator;
    }
}
```

Mediator

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

Introduction

Reducing communication complexity

In the civil aviation industry, all aircrafts pilots communicate with the air traffic controllers (ATC) who give pilots their clearance for takeoffs, landings and gate assignment.

This reduces communication complexity and accidents risks if pilots were all communicating at the same time on the ground and in the air.

Bruno Pisano
HABLO ESPAÑOL
REAL ESTATE
(213) 842-0774
NOW IS THE TIME TO
BUY or SELL

Reducing communication complexity

A real estate agent helps customers find a new house to purchase. He\she simplifies the communications between the customers willing to buy a new house and the people willing to sell their houses.

Thus, customers who want to contract a loan with a bank to pay for their house can contact a financial broker who can help them find the best loan offer. The broker agent simplifies the communications between customers and banking organizations.

NETGEAR
Auto 10/100 Mbps Switching
On = Link
Blink = Act
100M
10M
PWR
4

PP

Reducing communication complexity

A switch or hub facilitates the communications between several devices (computers, printers, etc.) on the same local network.

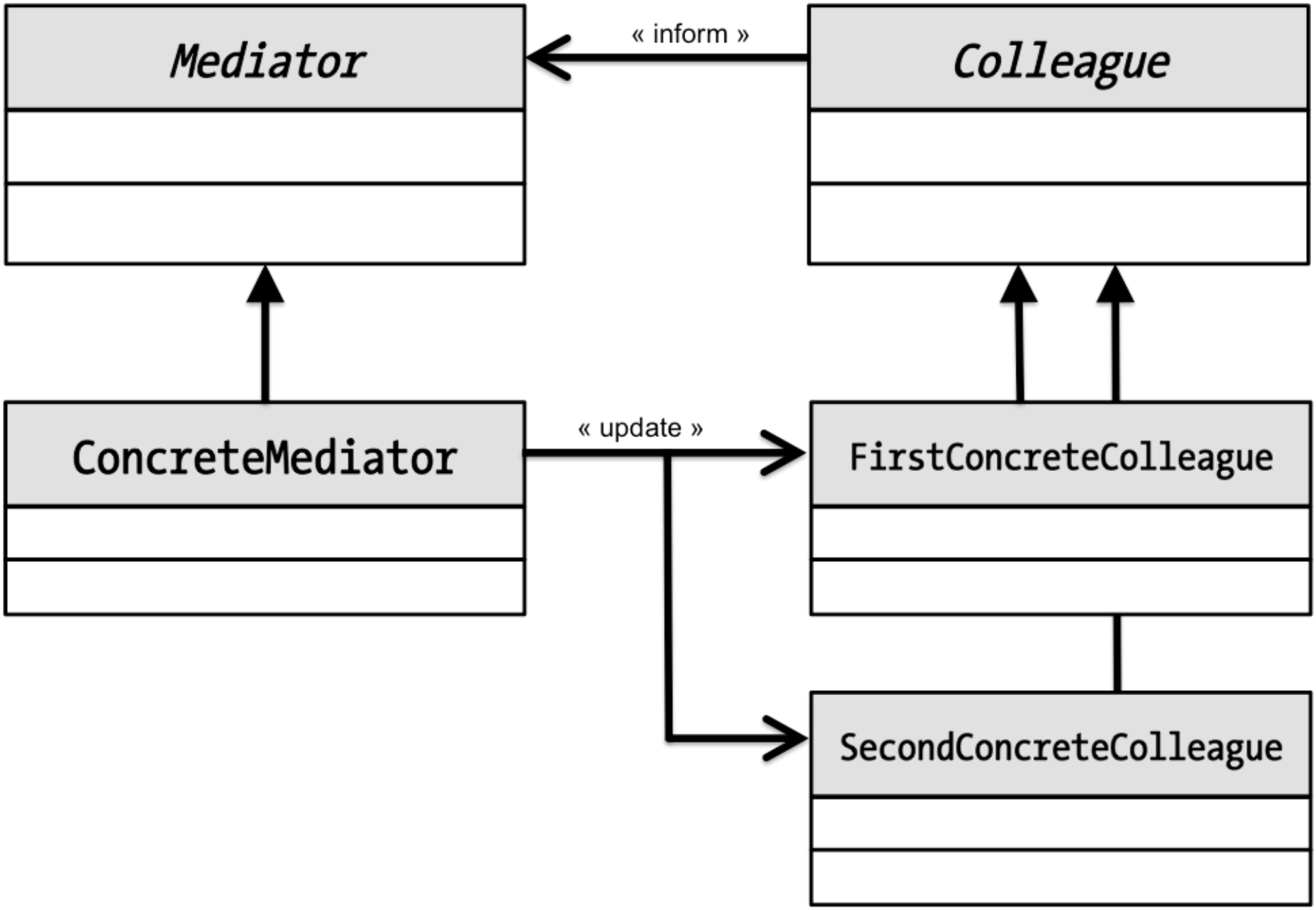
The switch can broadcast the data from a device to all connected devices of the network. It can also transfer the data from one point of the local network to another without broadcasting to the remaining devices.

The Mediator Design Pattern

The Mediator Pattern

- **Behavioral pattern**

- Ease communication between several objects.
- Reduce explicit coupling between objects.
- Make code truly extensible.


Mediator
« inform »
Colleague
ConcreteMediator
« update »
FirstConcreteColleague
SecondConcreteColleague

Implementing the Mediator Design Pattern

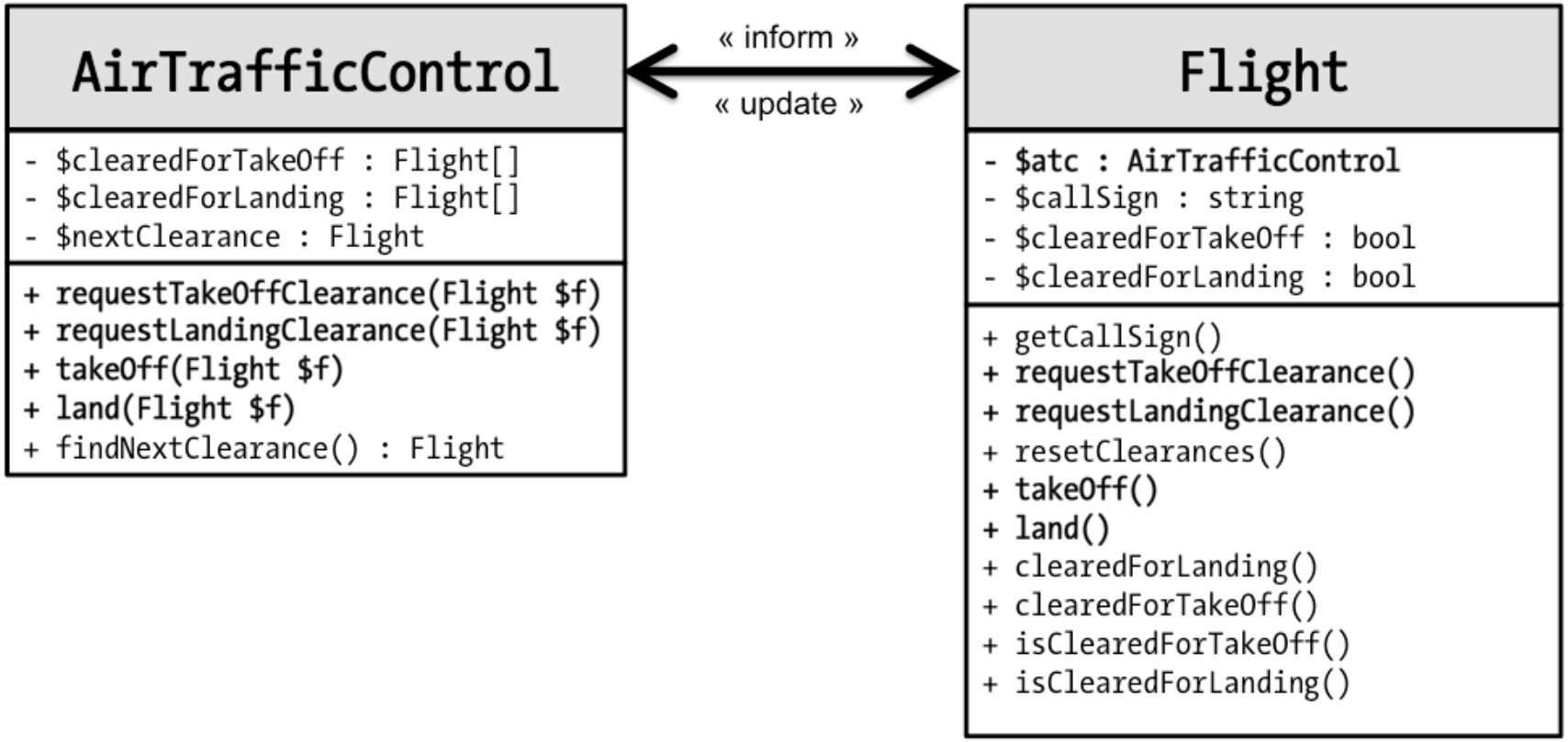
#1.
Air Traffic Control

Air Traffic Control

In order to take off or land, flights must request clearance to the air traffic control tower. The ATC keeps a list of flights cleared for landing and another list for flights cleared for taking off.

The ATC manages connections between all flights. It gives a priority clearance for landing flights. Each flight must always be cleared to land (or take off) by the ATC tower.

UML Diagram


AirTrafficControl
- $clearedForTakeOff : Flight[]
- $clearedForLanding : Flight[]
- $nextClearance : Flight
+ requestTakeOffClearance(Flight $f)
+ requestLandingClearance(Flight $f)
+ takeOff(Flight $f)
+ land(Flight $f)
+ findNextClearance() : Flight
« inform »
« update »
Flight
- $atc : AirTrafficControl
- $callSign : string
- $clearedForTakeOff : bool
- $clearedForLanding : bool
+ getCallSign()
+ requestTakeOffClearance()
+ requestLandingClearance()
+ resetClearances()
+ takeOff()
+ land()
+ clearedForLanding()
+ clearedForTakeOff()
+ isClearedForTakeOff()
+ isClearedForLanding()

```php
class AirTrafficControl
{
    private $clearedForTakeOff;
    private $clearedForLanding;
    private $nextClearance;

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

    public function land(Flight $flight)
    {
        $this->nextClearance = null;
        $this->findNextClearance();
    }

    public function requestLandingClearance(Flight $flight)
    {
        if (in_array($flight, $this->clearedForLanding)) {
            throw new \RuntimeException('Already waiting for landing clearance.');
        }

        $this->clearedForLanding[] = $flight;
    }
}
```

```php
class AirTrafficControl
{
    private $clearedForTakeOff;
    private $clearedForLanding;
    private $nextClearance;

    // ...

    public function requestTakeOffClearance(Flight $flight)
    {
        if (in_array($flight, $this->clearedForTakeOff)) {
            throw new \RuntimeException('Already waiting for takeoff clearance.');
        }

        $this->clearedForTakeOff[] = $flight;
    }

    public function takeOff(Flight $flight)
    {
        $this->nextClearance = null;
        $this->findNextClearance();
    }
}
```

```php
class AirTrafficControl
{
    // ...
    public function findNextClearance()
    {
        // Cleared flight hasn't used its clearance yet
        if (null !== $this->nextClearance) {
            return $this->nextClearance;
        }

        // Landing aircrafts have priority
        if (count($this->clearedForLanding)) {
            $flight = array_shift($this->clearedForLanding);
            $flight->clearedForLanding();
            $this->nextClearance = $flight;

            return $this->nextClearance;
        }

        if (count($this->clearedForTakeOff)) {
            $flight = array_shift($this->clearedForTakeOff);
            $flight->clearedForTakeOff();
            $this->nextClearance = $flight;
        }

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

```php
class Flight
{
    private $atc;
    private $callSign;
    private $clearedForTakeOff;
    private $clearedForLanding;

    function __construct(AirTrafficControl $atc, $callSign)
    {
        $this->atc = $atc;
        $this->callSign = $callSign;
        $this->resetClearances();
    }

    public function getCallSign()
    {
        return $this->callSign;
    }

    private function resetClearances()
    {
        $this->clearedForLanding = false;
        $this->clearedForTakeOff = false;
    }
}
```

```php
class Flight
{
    // ...
    public function requestTakeOffClearance()
    {
        $this->atc->requestTakeOffClearance($this);
    }

    public function takeOff()
    {
        if (!$this->clearedForTakeOff) {
            throw new \LogicException('Flight does not have clearance.');
        }
        $this->atc->takeOff($this);
        $this->resetClearances();
    }

    public function clearedForTakeOff()
    {
        $this->clearedForTakeOff = true;
    }

    public function isClearedForTakeOff()
    {
        return $this->clearedForTakeOff;
    }
}
```

```php
class Flight
{
    // ...
    public function requestLandingClearance()
    {
        $this->atc->requestLandingClearance($this);
    }

    public function land()
    {
        if (!$this->clearedForLanding) {
            throw new \LogicException('Flight does not have clearance.');
        }
        $this->atc->land($this);
        $this->resetClearances();
    }

    public function clearedForLanding()
    {
        $this->clearedForLanding = true;
    }

    public function isClearedForLanding()
    {
        return $this->clearedForLanding;
    }
}
```

Mediator usage

```
$atc = new AirTrafficControl();

$flight1 = new Flight($atc, 'AFR123');
$flight2 = new Flight($atc, 'AFR006');
$flight3 = new Flight($atc, 'AFR376');
$flight4 = new Flight($atc, 'AFR100');
$flight5 = new Flight($atc, 'AFR888');

$flight1->requestTakeOffClearance();
$flight5->requestTakeOffClearance();
$flight3->requestLandingClearance();
$flight2->requestLandingClearance();
$flight4->requestLandingClearance();
```

Mediator usage

```
while ($cleared = $atc->findNextClearance()) {

    $callSign = $cleared->getCallSign();

    if ($cleared->isClearedForLanding()) {
        echo sprintf("Flight %s is cleared for landing.\n", $callSign);
        echo sprintf("Flight %s is landing.\n", $callSign);
        $cleared->land();
        echo sprintf("Flight %s has landed.\n", $callSign);
        echo "----------------------------------------\n";
        continue;
    }

    echo sprintf("Flight %s is cleared for takeOff.\n", $callSign);
    echo sprintf("Flight %s is taking off.\n", $callSign);
    $cleared->takeOff();
    echo sprintf("Flight %s has taken off.\n", $callSign);
    echo "----------------------------------------\n";
}
```

```
Flight AFR376 is cleared for landing.
Flight AFR376 is landing.
Flight AFR376 has landed.
----------------------------------------
Flight AFR006 is cleared for landing.
Flight AFR006 is landing.
Flight AFR006 has landed.
----------------------------------------
Flight AFR100 is cleared for landing.
Flight AFR100 is landing.
Flight AFR100 has landed.
----------------------------------------
Flight AFR123 is cleared for takeOff.
Flight AFR123 is taking off.
Flight AFR123 has taken off.
----------------------------------------
Flight AFR888 is cleared for takeOff.
Flight AFR888 is taking off.
Flight AFR888 has taken off.
```

#2.
Event Manager

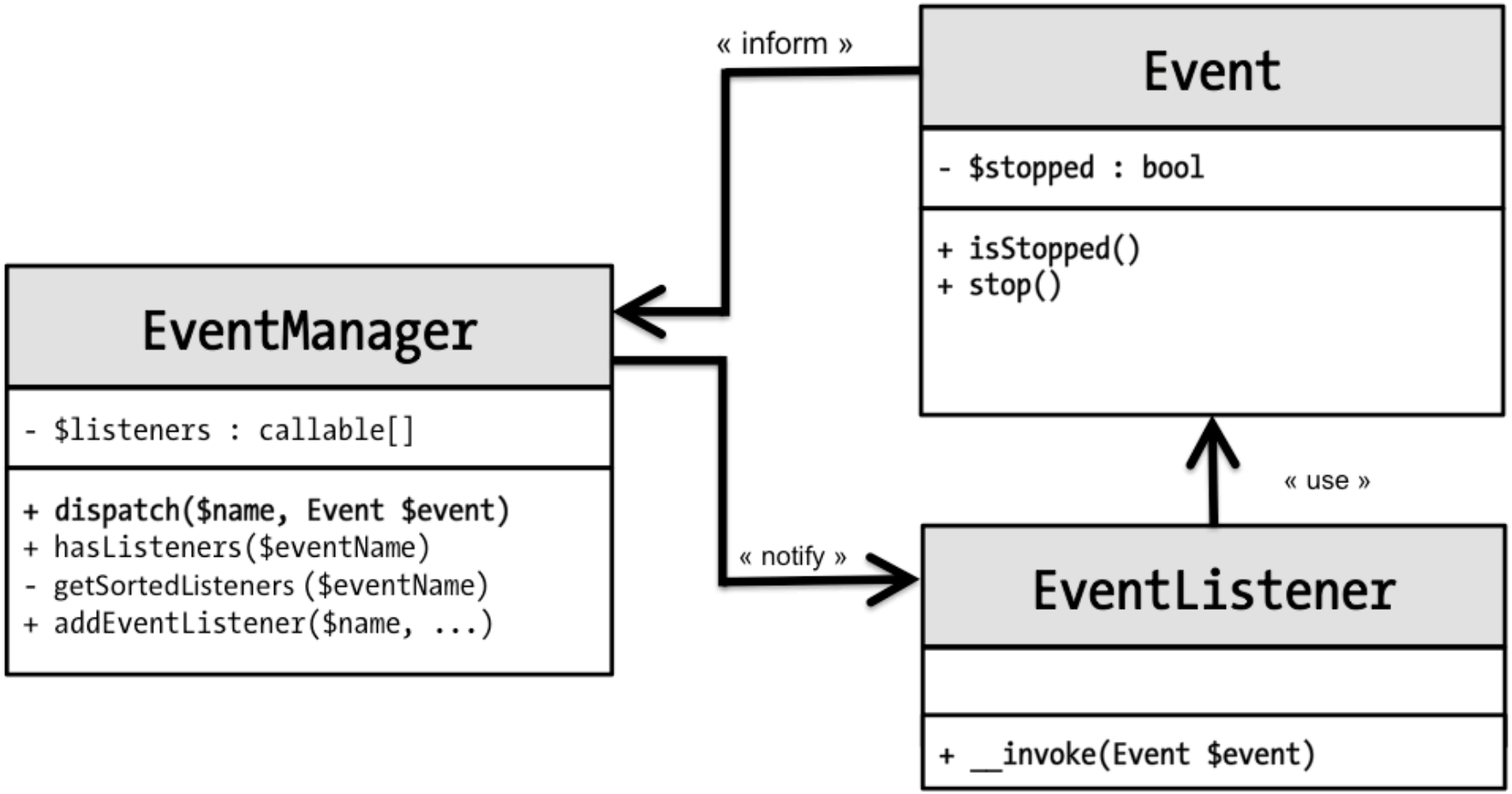
Event Manager

An **event manager** (or event dispatcher) is a **Mediator** pattern implementation, which allows to ease communications between different kind of objects.

The event dispatcher dispatches an **event**, an object that carries information, and passes it it each **listener** object that listens to the dispatched event.

An **event listener** is any callable object that is responsible for receiving the event object and processing it. An event listener can stop the event propagation. In this case, remaining listeners won't be invoked.

UML Diagram


« inform »
Event
- $stopped : bool
+ isStopped()
+ stop()
EventManager
- $listeners : callable[]
+ dispatch($name, Event $event)
+ hasListeners($eventName)
- getSortedListeners ($eventName)
+ addEventListener($name, ...)
« use »
« notify »
EventListener
+ __invoke(Event $event)

```php
abstract class Event
{
    private $stopped;

    public function __construct()
    {
        $this->stopped = false;
    }

    public function isStopped()
    {
        return $this->stopped;
    }

    public function stop()
    {
        $this->stopped = true;
    }
}
```

```php
class EventManager
{
    private $listeners;

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

    private function hasListeners($event)
    {
        return !empty($this->listeners[$event]);
    }

    /**
     * Attaches a new event listener.
     *
     * @param string   $event    The name of the event to listen to
     * @param callable $listener The listener to register on this event
     * @param int      $priority The execution priority of this listener
     *
     * @return void
     */
    public function addEventListener($event, callable $listener, $priority = 0)
    {
        $this->listeners[$event][] = [ $listener, $priority ];
    }
}
```

```php
class EventManager
{
    // ...

    private function getSortedListeners($event)
    {
        $listenersByPriority = [];
        foreach ($this->listeners[$event] as $listener) {
            $priority = $listener[1];
            $listenersByPriority[$priority][] = $listener[0];
        }

        krsort($listenersByPriority);

        $listeners = [];
        foreach ($listenersByPriority as $priorityListeners) {
            $listeners = array_merge($listeners, $priorityListeners);
        }

        return $listeners;
    }
}
```

```php
class EventManager
{
    // ...

    public function dispatch($name, Event $event)
    {
        if (!$this->hasListeners($name)) {
            return $event;
        }

        foreach ($this->getSortedListeners($name) as $listener) {
            call_user_func_array($listener, [ $event ]);
            if ($event->isStopped()) {
                // Stop looping if a listener
                // has stopped the event propagation.
                break;
            }
        }

        return $event;
    }
}
```

Event manager usage

```
$dic = new \Pimple\Container($settings['parameters']);

$dic->register('event_manager', function (Container $dic) {
    $manager = new EventManager();
    $manager->addEventListener('request', [new RouterListener(...), 'onKernelRequest']);
    $manager->addEventListener('request', [$dic['security.auth_listeners'], 'onKernelRequest']);
    $manager->addEventListener('controller', [new ControllerListener($dic), 'onKernelController']);
    $manager->addEventListener('exception', [new LoggerHandler(...), 'onKernelException']);
    $manager->addEventListener('exception', [$dic['security.error_handler’], 'onKernelException']);
    $manager->addEventListener('exception', [$dic['error_handler'], 'onKernelException']);

    return $manager;
});
```

Event listeners objects are registered first on the event manager object. Each event listener is configured to listen to an event (request, controller, exception, etc.).

```php
class HttpKernel implements KernelInterface
{
    /** @var EventManager */
    private $eventManager;
    private $factory;

    private function doHandleRequest(Request $request)
    {
        $event = $this->dispatch('request', new KernelEvent($request));
        if ($event->hasResponse()) {
            return $event->getResponse();
        }

        // Create the controller based on request attributes
        $controller = $this->factory->createController($request->getAttributes());
        $event = $this->dispatch('controller', new ControllerEvent($controller, $request));
        $response = call_user_func_array($event->getController(), [ $request ]);

        if (!$response instanceof Response) {
            throw new \RuntimeException('A controller must return a Response object.');
        }

        $event = $this->dispatch('response', new ResponseEvent($request, $response));

        return $event->getResponse();
    }

    // ...
}
```

```php
class RouterListener
{
    private $router;
    private $urlGenerator;

    public function __construct(
        UrlMatcherInterface $router,
        RequestContextAwareInterface $urlGenerator = null
    )
    {
        $this->router = $router;
        $this->urlGenerator = $urlGenerator;
    }

    public function onKernelRequest(KernelEvent $event)
    {
        $request = $event->getRequest();
        $context = RequestContext::createFromRequest($request);

        if ($this->urlGenerator) {
            $this->urlGenerator->setRequestContext($context);
        }

        $request->setAttributes($this->router->match($context));
    }
}
```

Visitor

Represent an operation to be performed on elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates.

Introduction

Getting Information about a Shopping Cart

Consider a **ShoppingCart** object which contains a list of **ShoppingCartItem** instance. Each of these instances holds a reference to a **Product** reference object and the ordered quantity.

At the checkout, we want to perform several operations on the shopping cart like computing the total price, sorting all products per categories or even issueing the receipt.

Example of a Shopping Cart

```
$cart = new ShoppingCart();
$cart->addItem(0.65, new Tomatoes('Vegetables', new EUR(189)));
$cart->addItem(1.38, new Salmon('Seafood', new EUR(2250)));
$cart->addItem(2, new ToothpasteTube('Wellness', new EUR(115)));
$cart->addItem(0.89, new Apples('Vegetables', new EUR(128)));
$cart->addItem(3, new Shampoo('Wellness', new EUR(123)));
$cart->addItem(0.47, new Tuna('Seafood', new EUR(1980)));
$cart->addItem(0.26, new GoatCheese('Creamery', new EUR(1723)));
```

Each product reference object has a category and a price. The price can be either a unit price (for a shampoo for instance) or a price per kilogram (for vegetables or seafood).

Performing Operations on a Shopping Cart

```
class ShoppingCart
{
    /** @var ShoppingCartItems[] */
    private $items;

    public function getTotalPrice() { ... }
    public function sortByCategories() { ... }
    public function issueReceipt(PDFGenerator $gen) { ... }
}
```

Implementing all these operations in the **ShoppingCart** class will bloat it at some point. Also, adding new operations requires to change the class body.

Concrete Visitors in Real Life

Doctors are Visitors for a Human Body!

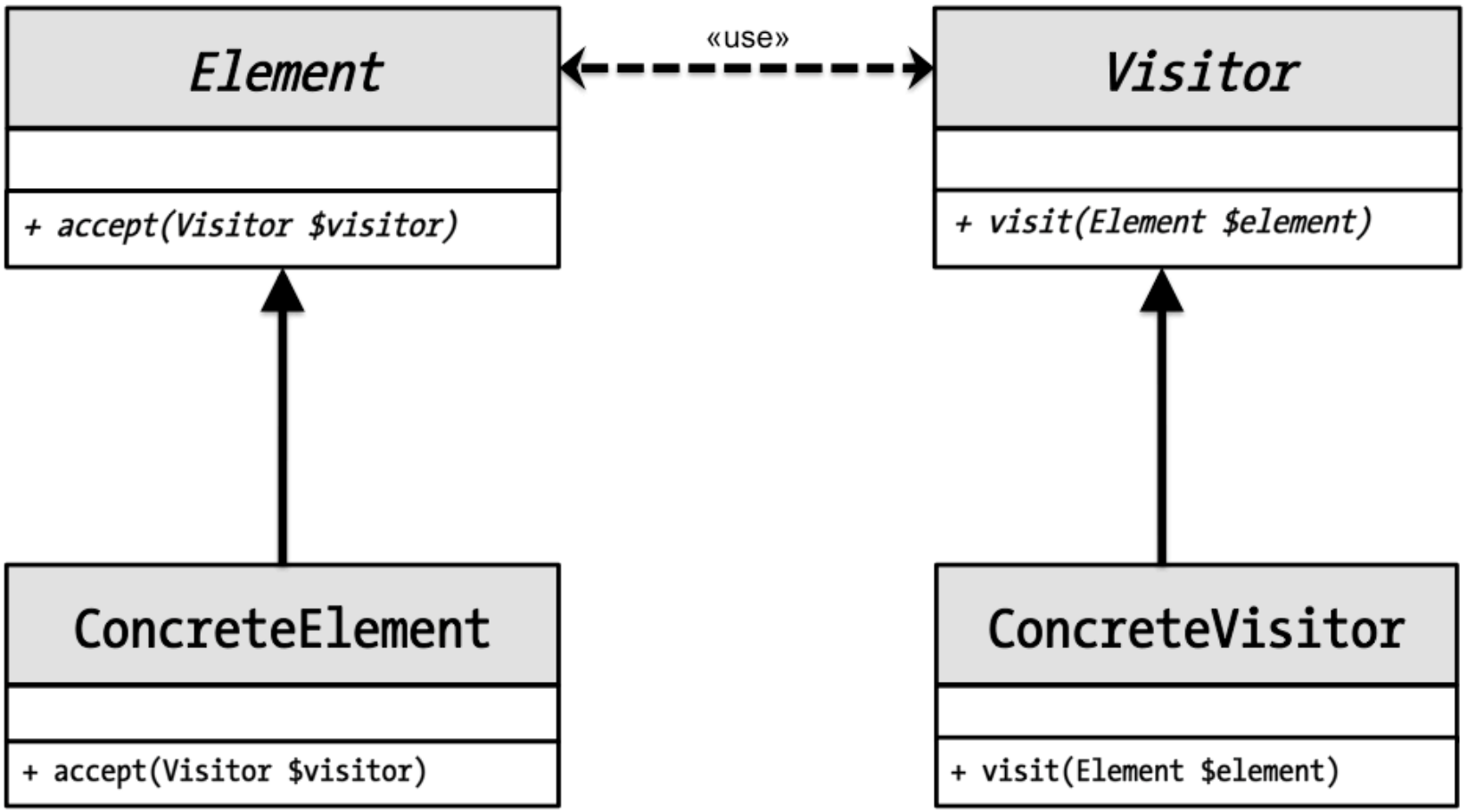
The Visitor Design Pattern

The Visitor Pattern

- **Behavioral pattern**

- Represent an operation to be performed on elements of an object structure.
- Visitor lets you define a new operation without changing the classes of the elements on which it operates.
- Prevent code duplications
- Guaratee the Open/Close principle
- Encourage the Single Responsability principle

UML Diagram


Element
+ accept(Visitor $visitor)
«use»
Visitor
+ visit(Element $element)
ConcreteElement
+ accept(Visitor $visitor)
ConcreteVisitor
+ visit(Element $element)

The Element and Visitor Interfaces

```
interface ElementInterface
{
    public function accept(VisitorInterface $visitor);
}

interface VisitorInterface
{
    public function visit(ElementInterface $element);
}
```

The concrete element and visitor classes

```
class ConcreteElement implement ElementInterface
{
    public function accept(VisitorInterface $visitor)
    {
        $visitor->visit($this);
    }
}

class ConcreteVisitor implement VisitorInterface
{
    public function visit(ElementInterface $element)
    {
        // do something with the element
    }
}
```

Visitor Pattern Usage

```
$element1 = new ConcreteElement();
$element1->accept(new ConcreteVisitorA());
$element1->accept(new ConcreteVisitorB());
$element1->accept(new ConcreteVisitorC());

$element2 = new OtherConcreteElement();
$element2->accept(new ConcreteVisitorD());
$element2->accept(new ConcreteVisitorC());
```

The **accept()** method of each concrete element receives a concrete instance of the **Visitor** type and invokes the its **visit()** method on it. The concrete element injects itself into the **visit()** method list of arguments.

Implementing the Visitor Design Pattern

#1.
Shopping Cart

Implementing Cart Operations as Cart Visitors

```
class ShoppingCart
{
    // ...

    public function accept(ShoppingCartVisitorInterface $visitor)
    {
        $visitor->visit($this);
    }
}
```

The first step is to make the **ShoppingCart** instance accept a **ShoppingCartVisitorInterface** implementation. The latter will then visit the shopping cart object that injects itself into the **visit()** method.

Computing the Shopping Cart Total Price

```
class TotalPriceCartVisitor implements ShoppingCartVisitorInterface
{
    private $total;

    public function __construct()
    {
        $this->total = new \SebastianBergmann\Money\EUR(0);
    }

    public function visit(ShoppingCart $cart)
    {
        foreach ($cart->getItems() as $item) {
            $this->total = $this->total->add($item->getTotalPrice());
        }
    }

    public function getTotalPrice()
    {
        return $this->total;
    }
}
```

Computing the Shopping Cart Total Price

```
$cart = new ShoppingCart();
// ...

$pricerVisitor = new TotalPriceCartVisitor();
$cart->accept($pricerVisitor);
$pricerVisitor->getTotalPrice()->getConvertedAmount();
```

Computing the total price of the whole shopping cart is as simple as passing the visitor to the shopping cart instance.

Sorting the Shopping Cart Items By Categories

```
class SortCartItemsVisitor implements ShoppingCartVisitorInterface
{
    private $sortedList;

    public function visit(ShoppingCart $cart)
    {
        foreach ($cart->getItems() as $item) {
            $this->sortedList[$item->getCategory()][] = $item;
        }

        ksort($this->sortedList);
    }

    public function getSortedCartItems()
    {
        return $this->sortedList;
    }
}
```

Sorting the Shopping Cart Items By Categories

```
$cart = new ShoppingCart();
// ...

$sortVisitor = new SortCartItemsVisitor();
$cart->accept($sortVisitor);

print_r($sortVisitor->getSortedCartItems());
```

Sorting the whole list of shopping cart items by product references categories is as simple as injecting the visitor into the **ShoppingCart** instance.

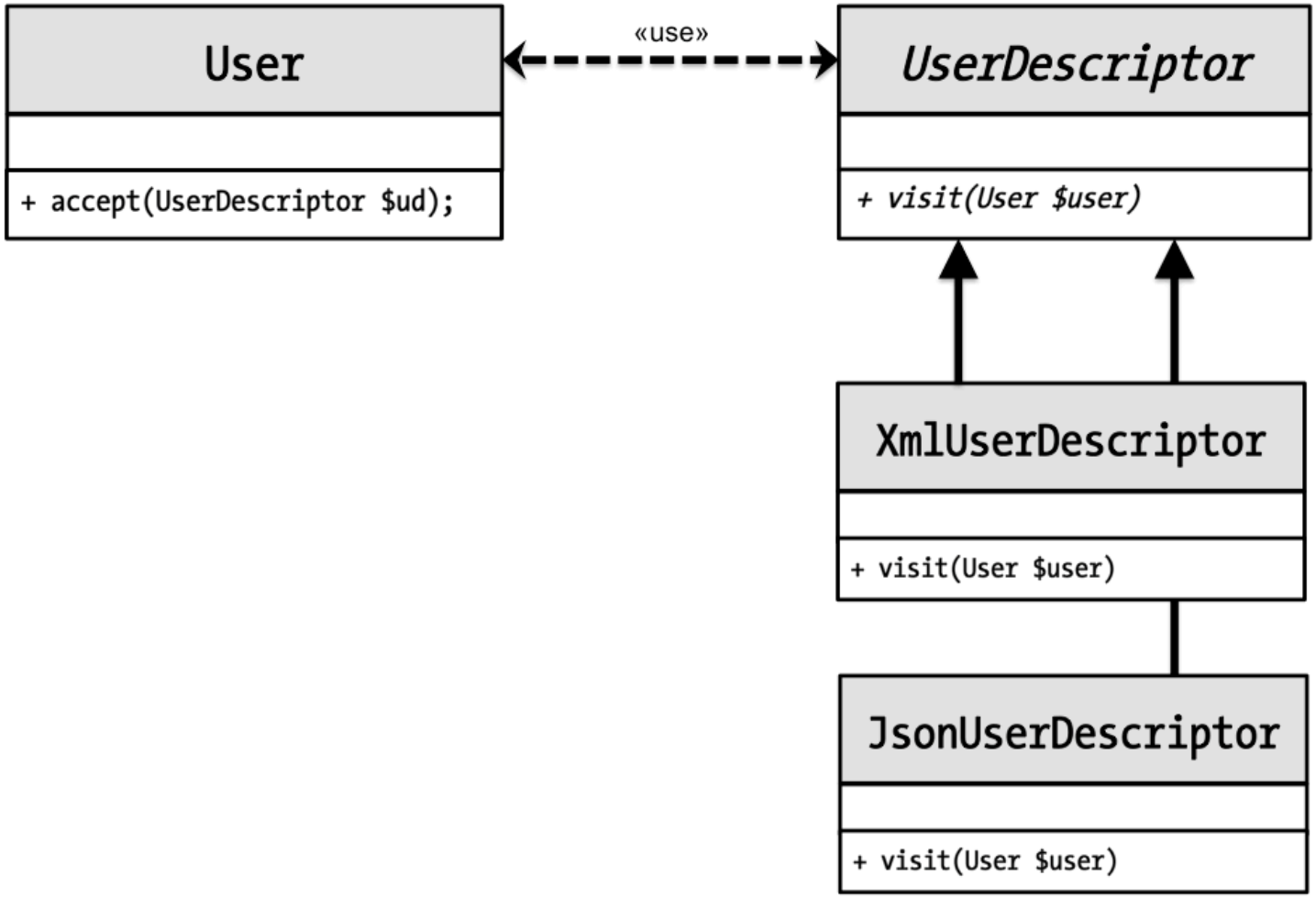
#2.
User Descriptors

Separating a data structure from its computations

The **Visitor** design pattern helps **separating** a **data structure** from its many **computations**.

For example, a User data structure object that must be easily exported to different output serialization formats like HTML, XML, CSV or JSON.

UML Diagram


User
+ accept(UserDescriptor $ud);
«use»
UserDescriptor
+ visit(User $user)
XmlUserDescriptor
+ visit(User $user)
JsonUserDescriptor
+ visit(User $user)

Implementation – The Visitable Object

```php
class User
{
    private $id;
    private $fullName;
    private $birthdate;
    private $biography;

    public function __construct($id, $fullName, $birthdate, $biography = null)
    {
        if (!$birthdate instanceof $birthdate) {
            $birthdate = new \DateTime($birthdate);
        }

        $this->id = $id;
        $this->fullName = $fullName;
        $this->birthdate = $birthdate;
        $this->biography = $biography;
    }

    // ... + getter methods for each attribute
}
```

Implementation – The Visitable Object

```php
class User
{
    // ...

    public function accept(UserDescriptorInterface $visitor)
    {
        $visitor->visit($this);
    }
}
```

Implementation – The Abstract Visitor

```
interface UserDescriptorInterface
{
    public function visit(User $user);
}



abstract class AbstractUserDescriptor implements UserDescriptorInterface
{
    protected $description;

    public function getDescription()
    {
        return $this->description;
    }
}
```

Implementation – The XML Descriptor Visitor

```
class XmlUserDescriptorVisitor extends AbstractUserDescriptor
{
    public function visit(User $user)
    {
        $dob = $user->getBirthdate();

        $doc = new \DOMDocument('1.0', 'utf-8');
        $doc->preserveWhiteSpace = true;

        $root = $doc->createElement('user');
        $root->setAttribute('id', $user->getId());
        $doc->appendChild($root);

        $root->appendChild($doc->createElement('name', $user->getFullName()));
        $root->appendChild($doc->createElement('birthdate', $dob->format('Y-m-d')));
        $root->appendChild($doc->createElement('biography', $user->getBiography()));

        $this->description = $doc->saveXML();
    }
}
```

Implementation – The JSON Descriptor Visitor

```
class JsonUserDescriptorVisitor extends AbstractUserDescriptor
{
    public function visit(User $user)
    {
        $dob = $user->getBirthdate();

        $dto = new \stdClass();
        $dto->id = (int) $user->getId();
        $dto->name = $user->getFullName();
        $dto->birthdate = $dob->format('Y-m-d');
        $dto->biography = $user->getBiography();

        $this->description = json_encode($dto);
    }
}
```

Usage

```php
$xmlDescriptor = new XmlUserDescriptorVisitor();
$jsonDescriptor = new JsonUserDescriptorVisitor();

$john = new User(42, 'John Smith', '1987-07-02', 'PHP expert');
$john->accept($xmlDescriptor);
$john->accept($jsonDescriptor);

echo $xmlDescriptor->getDescription();
echo $jsonDescriptor->getDescription();
```

Without changing the User class, the User instance can dynamically receive new virtual computation methods thanks to its visitors.

Single Dispatch versus Double Dispatch

Single Dispatch

```
class XmlUserDescriptorVisitor extends AbstractUserDescriptor
{
    public function visit(User $user)
    {
        // ...
    }
}
```

Single dispatch enables a visitor to visit only one single type of object at a time.

Multiple Dispatch

```
class XmlUserDescriptorVisitor extends AbstractUserDescriptor
{
    public function visit(User $user)
    {
        // ...
    }

    public function visit(Address $address)
    {
        // ...
    }
}
```

Multiple dispatch enables a visitor to visit multiple objects with a single method name thanks to polymorphism. The right method to invoke is chosen at runtime based on the given visited object type.

SensioLabs services

About us

We are the **creators of Symfony**. We know the framework and PHP inside out and we can help you.

Contact us

SensioLabs France
92-98, Boulevard Victor Hugo
92 115 Clichy Cedex
France
Tel: +33 (0)1 40 99 81 09
contact@sensiolabs.com

SensioLabs Deutschland
Neusser Str. 27-29
50670 Köln
Germany
Tel: +49 221 66 99 27 50
contact@sensiolabs.de

We also provide worldwide **on-site services**. Contact us at: sensiolabs.com/en/contact

Our products and services

SensioLabs Insight

Blackfire

Consulting

Training

SensioLabs Insight

insight.sensiolabs.com

SensioLabsInsight

Dashboard Help What we analyze Pricing Blog Account

fabpot / symfony #3812 Triggered by the API 2 hours ago, duration: 8 minutes

70/100

3 months

to get the Platinum Medal

Search

Severity

2 Critical
49 Major
262 Minor
16 Info

Category

38 Architecture
97 Bugrisk
17 Codestyle
171 Deadcode
4 Performance
2 Security

Developer

117 Nicolas Grekas
48 Bernhard Schussek

Violations (329) Fixed Ignored

▼ **Database queries should use parameter binding** 2 — Critical Security

in src/Symfony/Component/HttpKernel/Profiler/PdoProfilerStorage.php, line 59

```
54.
55.         list($criteria, $args) = $this->buildCriteria($ip, $url, $start, $enc
56.
57.         $criteria = $criteria ? 'WHERE '.implode(' AND ', $criteria) : '';
```

If provided by the user, the value of `implode(' AND ', $criteria)` *may allow an SQL injection attack. Avoid concatenating parameters to SQL query strings, and use parameter binding instead.*

Time to fix: about 1 hour — Comment | Open Issue | Permalink

Last edited 4 years ago by Jan Schumann

```
58.
```

▶ **Global variable or function should never be used** 11 — Major Architecture

▶ **Logical operators should be avoided** — Major Bugrisk

It analyzes the **quality** of your **code** continuously.

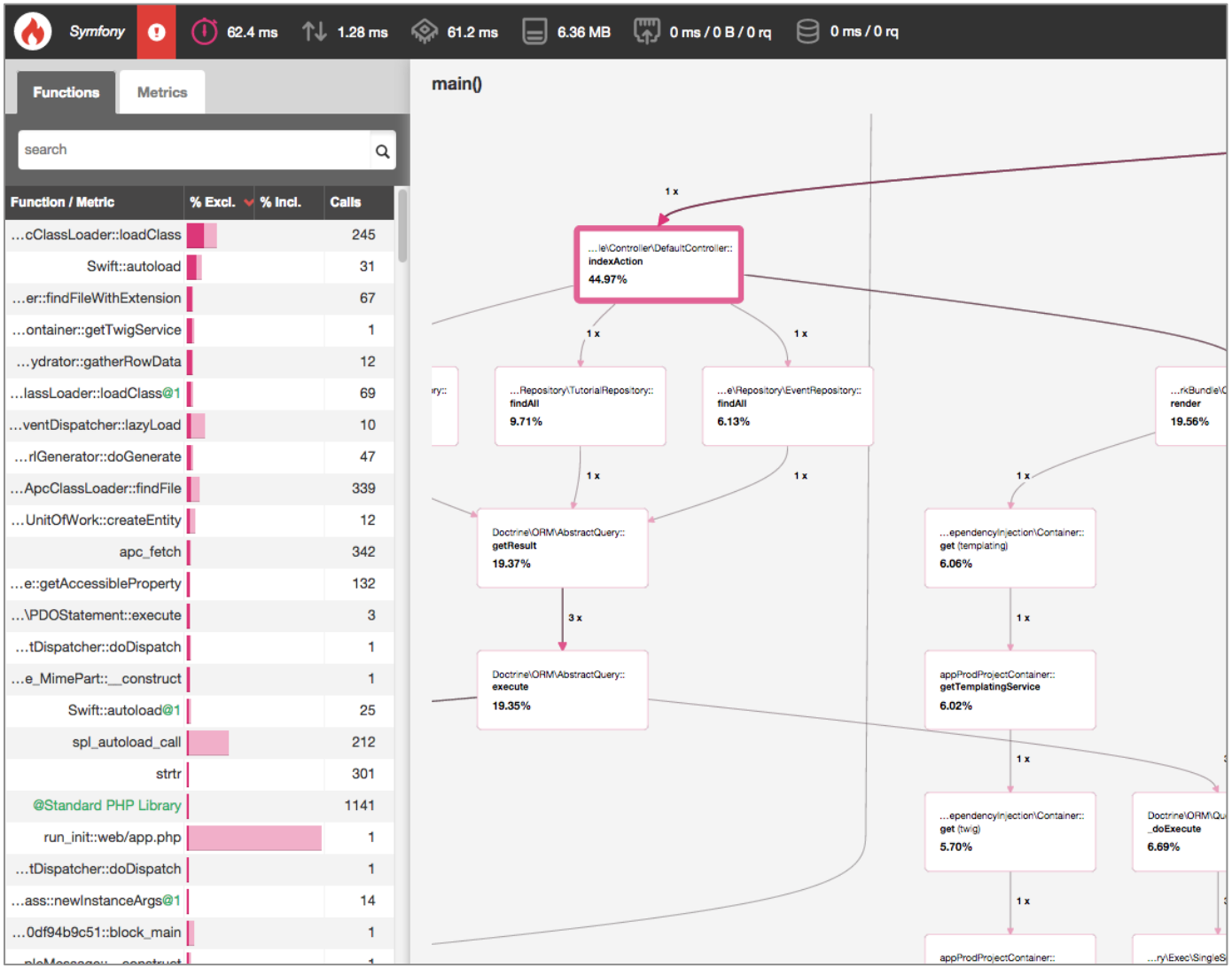
All problems are displayed on context and they provide **detailed solutions**.

It works for **any PHP application**, not only Symfony.

SensioLabs Insight helps you contain the **technical debt** of your projects.

Blackfire

blackfire.io



It analyzes the **performance** of your **application** to find bottlenecks.

It provides **useful metrics** for CPU, memory, I/O, SQL queries, HTTP requests, etc.

It works for **any PHP application**, not only Symfony.

Create faster applications with Blackfire.

SensioLabs Consulting

- We develop **proof of concept applications** to evaluate Symfony for your product.
- We **coach your team** and accompany your company through the development.
- We deliver **expert missions** to help you solve specific problems in your development.
- We help you **migrate** your **legacy** PHP applications.

Our full list of services: sensiolabs.com/en/packaged-solutions

SensioLabs Training

- We provide **general Symfony training** for all levels (Getting Started, Mastering and Hacking) and **specific Symfony training** for testing, performance and internals.
- We provide special Symfony training for your **developers**, including web development and Twig integration.
- We provide training for other **PHP** technologies such as Doctrine and Drupal.

Our full list of courses: training.sensiolabs.com

SensioLabs Training Department

Address

92-98 Boulevard Victor Hugo

92 115 Clichy Cedex

France

Phone

+33 1 40 99 82 05

Email

training@sensiolabs.com

SensioLabs