



# Formation Symfony Avancée

Romain Bohdanowicz

Twitter : @bioub - Github : <https://github.com/bioub>

<http://formation.tech/>

# Sommaire





# Introduction



- Romain Bohdanowicz

Ingénieur EFREI 2008, spécialité en Ingénierie Logicielle

- Expérience

Formateur/Développeur Freelance depuis 2006

Plus de 8000 heures de formation animées

- Langages

Expert : HTML / CSS / JavaScript / PHP / Java

Notions : C / C++ / Objective-C / C# / Python / Bash / Batch

- Certifications

PHP 5 / PHP 5.3 / PHP 5.5 / Zend Framework 1

- Particularités

Premier site web à 12 ans (HTML/JS/PHP), Triathlète à mes heures perdues

- Et vous ?

Langages ? Expérience ? Utilité de cette formation ?



# Tests automatisés



## ▸ Vérification manuelle

- Ecrire une recette de tests et demander à une personne de la rejouer à des étapes clés (nouvelle version)
- Ecrire le test sous la forme de code, et vérifier visuellement que les résultats attendus soit les bons

## ▸ Tests automatisés

- Le test est codé, la vérification se fait dans un rapport

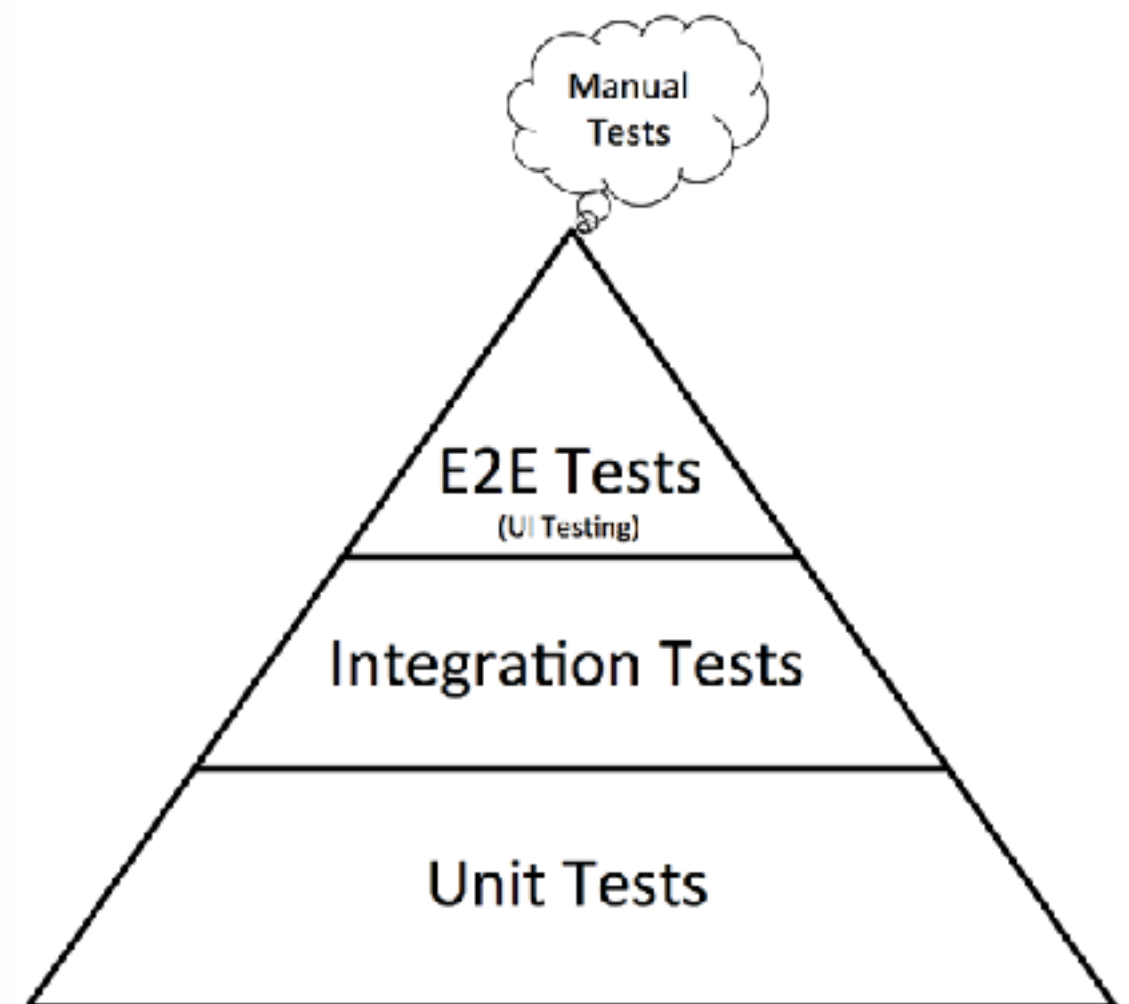
## ▸ Historique

- sUnit en 1994 (SmallTalk), JUnit en 1997 (Java)
- Les frameworks s'inspirant de sUnit sont catégorisés xUnit (PHPUnit, CUnit...)



## ► Types de tests

- **Unitaire** : tests des méthodes d'une classe
- **Intégration** : teste l'intégration entre plusieurs classes
- **Fonctionnels** : teste l'application du point de vue du client (HTTP dans le cas du web)
- **End-to-End (E2E)** : teste l'application dans le client (y compris JavaScript, CSS...)



Tests automatisés - 2 unit tests. 0 integration tests.





Tests automatisés - 2 unit tests. 0 integration tests.





# PHPUnit

# PHPUnit - Introduction

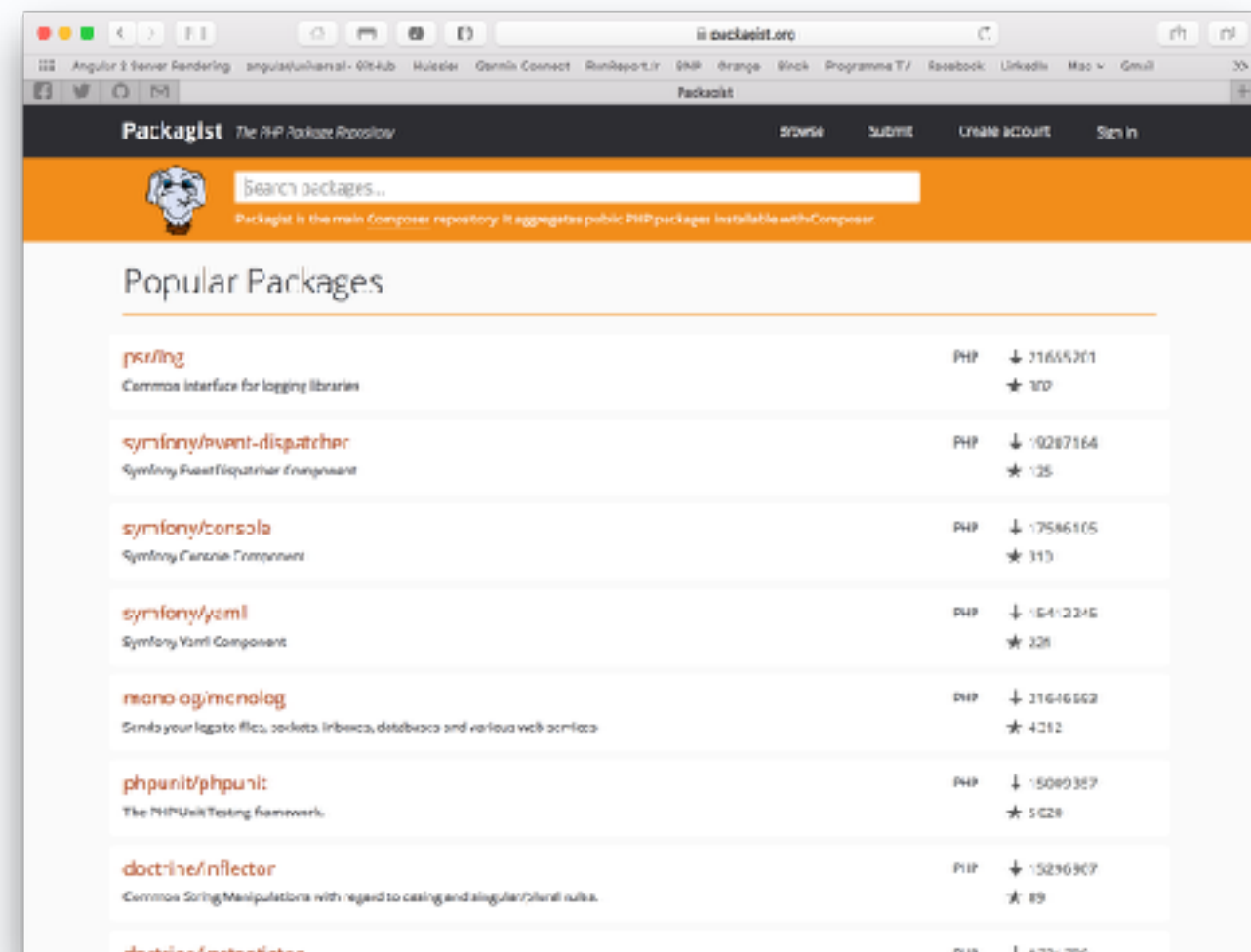


- ▶ Créé en 2001 par Sebastian Bergmann
- ▶ Framework de tests de référence en PHP  
Utilisé, même étendu par Symfony et Zend Framework

- ▶ Documentation  
<https://phpunit.de/documentation.html>

- ▶ Open Source  
Licence BSD Modifiée

- ▶ Concurrents :  
atoum (FR), Behat (BDD), SimpleTest





## ▸ PHAR

- Dernière Version  
<https://phar.phpunit.de/phpunit.phar>
- Version spécifique  
<https://phar.phpunit.de/phpunit-X.Y.Z.phar>

## ▸ Composer

- Dernière Version  
`composer global require phpunit/phpunit`
- Version spécifique  
`composer global require phpunit/phpunit:5.0.*`
- Penser à ajouter le répertoire bin global au PATH, sur UNIX :  
`~/ .composer/vendor/bin`



## ► Composer

- Dernière Version  
`composer require phpunit/phpunit --dev`
- Version spécifique  
`composer require phpunit/phpunit:5.0.* --dev`
- Ou en éditant directement le fichier `composer.json` puis `composer update`

```
{  
    "require-dev" : {  
        "phpunit/phpunit": "5.1.*"  
    }  
}
```

- Exécution depuis la racine du projet :  
`./vendor/bin/phpunit`



## ► Conventions

- Un test PHPUnit est une méthode dont le nom commence par test :  
`testMaFonction()`
- Cette méthode se trouve dans une classe dont le nom se termine par Test et qui hérite de `\PHPUnit_Framework_TestCase` ou `\PHPUnit\Framework\TestCase`

## ► Bonnes pratiques

- Ne pas hésiter à être le plus verbeux possible dans le nom des méthodes
- L'arborescence du répertoire test correspond au répertoire src (ex : `src/MonNamespace/MaClasse.php` -> `tests/MonNamespaceTest/MaClasseTest.php`)

# PHPUnit - Exemple



```
<?php

namespace FormationTechTest\Entity;

use FormationTech\Entity\CompteBancaire;

class CompteBancaireTest extends \PHPUnit_Framework_TestCase
{
    public function testCrediter()
    {
        $compte = new CompteBancaire(0);
        $compte->crediter(1000);
        $this->assertEquals(1000, $compte->getSolde());

        $compte->crediter(500);
        $this->assertEquals(1500, $compte->getSolde());
    }
}
```

```
MBP-de-Romain:PrepaFormationPHPUnit remain$ ./vendor/bin/phpunit tests/Entity/CompteBancaireTest.php --colors
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.
```

```
.                                                                    1 / 1 (100%)
```

```
Time: 39 ms, Memory: 1.50Mb
```

```
OK (1 test, 2 assertions)
```



- ▶ PHPUnit peut appeler des méthodes avant et après chaque test
  - setUp
  - tearDown
- ▶ Avant ou après chaque classe (méthodes statiques)
  - setUpBeforeClass
  - tearDownAfterClass



# PHPUnit - Ligne de commande



```
MBP-de-Romain:PrepaFormationPHPUnit romain$ ./vendor/bin/phpunit -h
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.
```

```
Usage: phpunit [options] UnitTest [UnitTest.php]
       phpunit [options] <directory>
```

## Code Coverage Options:

<code>--coverage-clover &lt;file&gt;</code>	Generate code coverage report in Clover XML format.
<code>--coverage-crap4j &lt;file&gt;</code>	Generate code coverage report in Crap4J XML format.
<code>--coverage-html &lt;dir&gt;</code>	Generate code coverage report in HTML format.
<code>--coverage-php &lt;file&gt;</code>	Export PHP_CodeCoverage object to file.
<code>--coverage-text=&lt;file&gt;</code>	Generate code coverage report in text format.
	Default: Standard output.
<code>--coverage-xml &lt;dir&gt;</code>	Generate code coverage report in PHPUnit XML format.
<code>--whitelist &lt;dir&gt;</code>	Whitelist <dir> for code coverage analysis.

## Logging Options:

<code>--log-junit &lt;file&gt;</code>	Log test execution in JUnit XML format to file.
<code>--log-tap &lt;file&gt;</code>	Log test execution in TAP format to file.
<code>--log-teamcity &lt;file&gt;</code>	Log test execution in TeamCity format to file.
<code>--log-json &lt;file&gt;</code>	Log test execution in JSON format.
<code>--testdox-html &lt;file&gt;</code>	Write agile documentation in HTML format to file.
<code>--testdox-text &lt;file&gt;</code>	Write agile documentation in Text format to file.
<code>--reverse-list</code>	Print defects in reverse order

# PHPUnit - Ligne de commande



## Test Selection Options:

<code>--filter &lt;pattern&gt;</code>	Filter which tests to run.
<code>--testsuite &lt;pattern&gt;</code>	Filter which testsuite to run.
<code>--group ...</code>	Only runs tests from the specified group(s).
<code>--exclude-group ...</code>	Exclude tests from the specified group(s).
<code>--list-groups</code>	List available test groups.
<code>--test-suffix ...</code>	Only search for test in files with specified suffix(es). Default: Test.php,.phpt

## Configuration Options:

<code>--bootstrap &lt;file&gt;</code>	A "bootstrap" PHP file that is run before the tests.
<code>-c --configuration &lt;file&gt;</code>	Read configuration from XML file.
<code>--no-configuration</code>	Ignore default configuration file (phpunit.xml).
<code>--no-coverage</code>	Ignore code coverage configuration.
<code>--include-path &lt;path(s)&gt;</code>	Prepend PHP's include_path with given path(s).
<code>-d key[=value]</code>	Sets a php.ini value.

## Miscellaneous Options:

<code>-h --help</code>	Prints this usage information.
<code>--version</code>	Prints the version and exits.
<code>--atleast-version &lt;min&gt;</code>	Checks that version is greater than min and exits.

# PHPUnit - Ligne de commande



## Test Execution Options:

<code>--report-useless-tests</code>	Be strict about tests that do not test anything.
<code>--strict-coverage</code>	Be strict about unintentionally covered code.
<code>--strict-global-state</code>	Be strict about changes to global state
<code>--disallow-test-output</code>	Be strict about output during tests.
<code>--disallow-resource-usage</code>	Be strict about resource usage during small tests.
<code>--enforce-time-limit</code>	Enforce time limit based on test size.
<code>--disallow-todo-tests</code>	Disallow @todo-annotated tests.
<code>--process-isolation</code>	Run each test in a separate PHP process.
<code>--no-globals-backup</code>	Do not backup and restore \$GLOBALS for each test.
<code>--static-backup</code>	Backup and restore static attributes for each test.
<code>--colors=&lt;flag&gt;</code>	Use colors in output ("never", "auto" or "always").
<code>--columns &lt;n&gt;</code>	Number of columns to use for progress output.
<code>--columns max</code>	Use maximum number of columns for progress output.
<code>--stderr</code>	Write to STDERR instead of STDOUT.
<code>--stop-on-error</code>	Stop execution upon first error.
<code>--stop-on-failure</code>	Stop execution upon first error or failure.
<code>--stop-on-warning</code>	Stop execution upon first warning.
<code>--stop-on-risky</code>	Stop execution upon first risky test.
<code>--stop-on-skipped</code>	Stop execution upon first skipped test.
<code>--stop-on-incomplete</code>	Stop execution upon first incomplete test.
<code>-v --verbose</code>	Output more verbose information.
<code>--debug</code>	Display debugging information during test execution.
<code>--loader &lt;loader&gt;</code>	TestSuiteLoader implementation to use.
<code>--repeat &lt;times&gt;</code>	Runs the test(s) repeatedly.
<code>--tap</code>	Report test execution progress in TAP format.
<code>--teamcity</code>	Report test execution progress in TeamCity format.
<code>--testdox</code>	Report test execution progress in TestDox format.
<code>--printer &lt;printer&gt;</code>	TestListener implementation to use.

# PHPUnit - phpunit.xml



```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit colors="true">

  <testsuites>
    <testsuite name="AllTests">
      <directory>tests/Mapper</directory>
    </testsuite>
  </testsuites>

  <filter>
    <blacklist>
      <directory suffix=".php"></directory>
      <file></file>
      <exclude>
        <directory suffix=".php"></directory>
        <file></file>
      </exclude>
    </blacklist>
    <whitelist processUncoveredFilesFromWhitelist="true">
      <directory suffix=".php">classes</directory>
      <file></file>
      <exclude>
        <directory suffix=".php"></directory>
        <file></file>
      </exclude>
    </whitelist>
  </filter>

  <logging>
    <log type="coverage-clover" target="logs/phpunit-coverage.xml"/>
    <log type="junit" target="logs/phpunit-log.xml" logIncompleteSkipped="false"/>
  </logging>

</phpunit>
```



- Un fichier de bootstrap peut être exécuté au démarrage de PHPUnit
- Intérêts :
  - Autochargement de classe (sauf si phpunit a été installé avec Composer et que l'autoloader est celui de composer)
  - Modification du `include_path`
  - Chargement de fichiers de configuration



## ► PHPStorm

The screenshot shows the PHPStorm IDE interface with the following components:

- Project Structure:** A tree view on the left showing the project hierarchy. The 'Entity' directory is expanded, showing files like 'Database.php' and 'DatabaseListTest.php'.
- Code Editor:** The main window displays the code for 'Database.php'. It shows a namespace declaration 'namespace FormationTech\Entity;' and a class 'class Database' with methods 'public function \_\_construct(\$name)' and 'public function getName()'.
- Coverage:** A panel on the right shows the code coverage for the 'Entity' directory. It indicates '50% files, 18% lines in 'Entity'' and lists the following statistics:

Element	Statistics, %
CompteBancaire.php	
Database.php	50% lines

- Test Results:** A panel at the bottom left shows the results of the PHPUnit tests. It indicates 'All 9 tests passed - 400ms' and lists the following tests:

Test Name	Time
testFindAllWithDummyProphecy	400ms
testFindAllWithFake	20ms
testFindAllWithMock	20ms
testFindAllWithMockProphecy	60ms
testFindAllWithMySQL	40ms
testFindAllWithSpyProphecy	50ms
testFindAllWithStubFromClass	30ms
testFindAllWithStubFromInterface	50ms
testFindAllWithStubProphecy	50ms

- Terminal:** A panel at the bottom right shows the output of the PHPUnit command. It includes the command path, the start time, the version of PHPUnit (5.1.3), the number of tests passed (19), and the exit code (0).

# PHPUnit - Intégration continue



## ► JUnit Plugin

A screenshot of a web browser displaying the Jenkins interface. The browser's address bar shows 'localhost'. The Jenkins page title is 'BuildApp #26 FormationTechTest\Mapper\DatabaseMapperTest [Jenkins]'. The main heading is 'Test Result : FormationTechTest\Mapper\DatabaseMapperTest'. Below the heading, it says '0 failures' and '9 tests Took 0.36 sec. add description'. A table titled 'All Tests' lists the test names, their durations, and their status (all 'Passed').

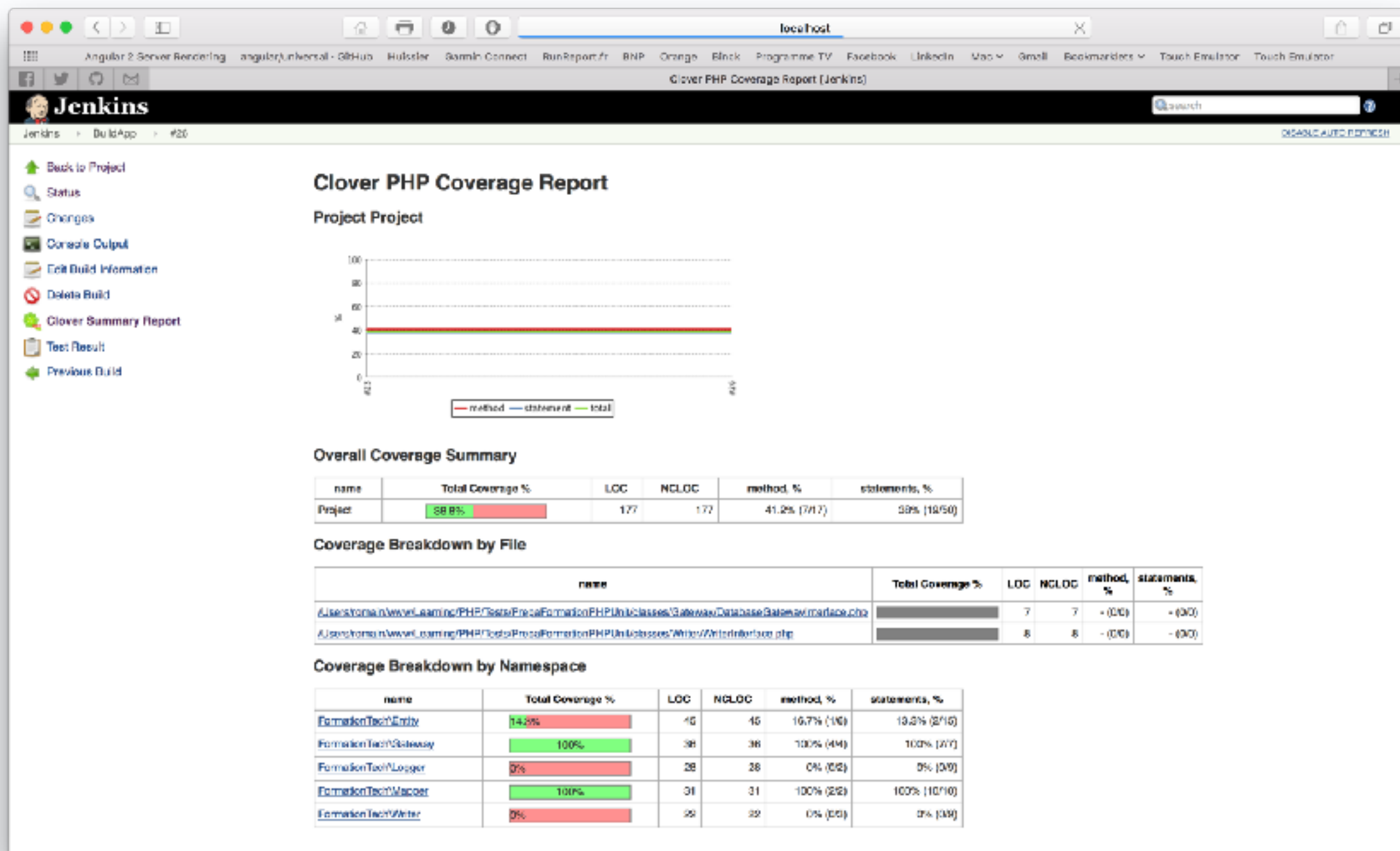
Test name	Duration	Status
<a href="#">testFindAllWithDummyProphecy</a>	0.1 sec	Passed
<a href="#">testFindAllWithFake</a>	12 ms	Passed
<a href="#">testFindAllWithMock</a>	15 ms	Passed
<a href="#">testFindAllWithMockProphecy</a>	52 ms	Passed
<a href="#">testFindAllWithMySQL</a>	33 ms	Passed
<a href="#">testFindAllWithSpyProphecy</a>	34 ms	Passed
<a href="#">testFindAllWithStubFromClass</a>	17 ms	Passed
<a href="#">testFindAllWithStubFromInterface</a>	53 ms	Passed
<a href="#">testFindAllWithStubProphecy</a>	43 ms	Passed



# PHPUnit - Intégration continue



## ► Clover PHP plugin







# Assertions PHPUnit



- Dans un framework xUnit, les assertions sont les méthodes qui vérifient qu'un résultat espéré corresponde au résultat attendu
- Le test échoue et s'arrête à la première assertion qui n'est pas vérifiée
- Bonnes pratiques :
  - Plusieurs assertions par test
  - Utiliser la méthode d'assertion la plus précise possible pour avoir un message d'erreur clair :  
Ex : `assertEmpty($tableau)`  
plutôt que `assertEquals(0, count($tableau))`
  - Si possible ajouter un message personnalisé

# Assertions PHPUnit - Basiques



- ▶ assertContains
- ▶ assertEquals
- ▶ assertFalse
- ▶ assertGreaterThan
- ▶ assertGreaterThanOrEqual
- ▶ assertInfinite
- ▶ assertInternalType
- ▶ assertLessThan
- ▶ assertLessThanOrEqual
- ▶ assertNan
- ▶ assertRegExp
- ▶ assertSame
- ▶ assertStringEndsWith
- ▶ assertStringMatchesFormat
- ▶ assertStringStartsWith
- ▶ assertThat
- ▶ assertTrue



- `assertArrayHasKey`
- `assertArraySubset`
- `assertCount`
- `assertContains`
- `assertContainsOnly`
- `assertContainsOnlyInstancesOf`
- `assertEmpty`



## ▸ Fichiers

- `assertFileEquals`
- `assertFileExists`
- `assertStringEqualsFile`
- `assertStringMatchesFormatFile`

## ▸ JSON

- `assertJsonFileEqualsJsonFile`
- `assertJsonStringEqualsJsonFile`
- `assertJsonStringEqualsJsonString`

## ▸ XML

- `assertEqualXMLStructure`
- `assertXmlFileEqualsXmlFile`
- `assertXmlStringEqualsXmlFile`
- `assertXmlStringEqualsXmlString`

# Assertions PHPUnit - Classes et Objets



- ▶ `assertClassHasAttribute`
- ▶ `assertClassHasStaticAttribute`
- ▶ `assertInstanceOf`
- ▶ `assertObjectHasAttribute`
- ▶ `assertNull`



# Types de tests PHPUnit

# Types de tests PHPUnit - Test unitaire



```
<?php

namespace FormationTech\Entity;

class CompteBancaire
{
    protected $solde;

    public function __construct($solde = 0)
    {
        $this->solde = (double) $solde;
    }
    public function getSolde()
    {
        return $this->solde;
    }

    public function debiter($montant)
    {
        $this->solde -= (double) $montant;
    }

    public function crediter($montant)
    {
        $this->solde += (double) $montant;
    }
}
```

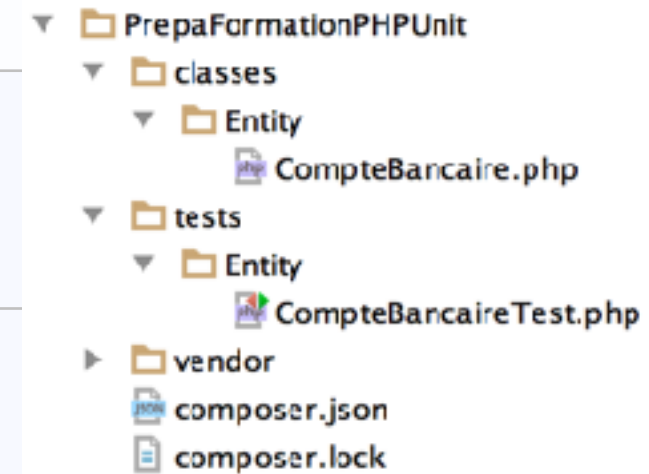
```
<?php

namespace FormationTechTest\Entity;

use FormationTech\Entity\CompteBancaire;

class CompteBancaireTest extends
    \PHPUnit_Framework_TestCase
{
    public function testCrediter()
    {
        $compte = new CompteBancaire(0);
        $compte->crediter(1000);
        $this->assertEquals(1000, $compte->getSolde());

        $compte->crediter(500);
        $this->assertEquals(1500, $compte->getSolde());
    }
}
```



```
MBP-de-Romain:PrepaFormationPHPUnit remain$ ./vendor/bin/phpunit tests/Entity/CompteBancaireTest.php --colors
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.
```

.

1 / 1 (100%)

Time: 39 ms, Memory: 1.50Mb

OK (1 test, 2 assertions)



# Types de tests PHPUnit - Test d'intégration



```
<?php

namespace FormationTech\Logger;

use FormationTech\Writer\WriterInterface;
use Psr\Log\LoggerInterface;
use Psr\Log\LoggerTrait;

class Logger implements LoggerInterface
{
    use LoggerTrait;

    protected $writer;

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

    public function log($level, $message, array $context = array())
    {
        $datetime = date('Y-m-d H:i:s');
        $logMessage = "[$level] - $datetime - $message";

        $this->writer->write($logMessage);
    }
}
```

```
<?php

namespace FormationTech\Writer;

class FileWriter implements WriterInterface
{
    protected $fic;

    public function __construct($filePath)
    {
        $this->fic = fopen($filePath, 'a');
    }

    public function write($message)
    {
        fwrite($this->fic, "$message\n");
    }

    public function __destruct()
    {
        fclose($this->fic);
    }
}
```

- Exemple de communication entre 2 classes :
  - Logger dépend de Writer (WriterInterface) et est compatible PSR-4
  - FileWriter implémente WriterInterface et sa méthode write

# Types de tests PHPUnit - Test d'intégration



```
<?php

namespace FormationTechTest\Logger;

use FormationTech\Logger\Logger;
use FormationTech\Writer\FileWriter;
use Psr\Log\LogLevel;

class LoggerTest extends \PHPUnit_Framework_TestCase
{
    public function testLogWithFileWriter()
    {
        $testFile = __DIR__ . '/../../tests.log';
        $fw = new FileWriter($testFile);
        $logger = new Logger($fw);

        $logger->log(LogLevel::NOTICE, 'Un message');
        $content = file_get_contents($testFile);

        $this->assertRegExp('/\[notice\] - \d{4}-\d{2}-\d{2} \d{2}:\d{2}:\d{2} - Un message\n/',
        $content);
    }
}
```

```
MBP-de-Romain:PrepaFormationPHPUnit romain$ ./vendor/bin/phpunit tests/Logger/LoggerTest.php --colors
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.
```

```
.                                                                    1 / 1 (100%)
```

```
Time: 38 ms, Memory: 1.50Mb
```

```
OK (1 test, 1 assertion)
```

# Types de tests PHPUnit - Test fonctionnel



```
<?php
require_once __DIR__ . '/vendor/autoload.php';

$pdo = new \PDO('mysql:host=localhost', 'root', '');
$gateway = new \FormationTech\Gateway\DatabaseGateway($pdo);
$dbList = $gateway->listDbs();
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Database list</title>
    </head>
    <body>
        <h2>Database list</h2>
        <ul>
            <?php foreach ($dbList as $db) : ?>
            <li><?=htmlspecialchars($db)?></li>
            <?php endforeach; ?>
        </ul>
    </body>
</html>
```

- Démarrage du PHP Built-in Server  
php -S localhost:8080

# Types de tests PHPUnit - Test fonctionnel



```
<?php

namespace FormationTechTest\Fonctionnal;

use Goutte\Client;

class DatabaseListTest extends \PHPUnit_Framework_TestCase
{
    public function testListDbs()
    {
        $client = new Client();
        $crawler = $client->request('GET', 'http://localhost:8080/database-list.php');

        $this->assertEquals(200, $client->getResponse()->getStatusCode());
        $this->assertEquals('Database list', $crawler->filter('h2')->text());
        $this->assertCount(13, $crawler->filter('ul > li'));
    }
}
```

```
MBP-de-Romain:PrepaFormationPHPUnit remain$ ./vendor/bin/phpunit tests/Logger/LoggerTest.php --colors
PHPUnit 5.1.3 by Sebastian Bergmann and contributors.
```

```
.                                                                    1 / 1 (100%)
```

```
Time: 38 ms, Memory: 1.50Mb
```

```
OK (1 test, 1 assertion)
```



# Tests Symfony basiques

# Tests Symfony basiques - Tests fonctionnels



## ► Test fonctionnels

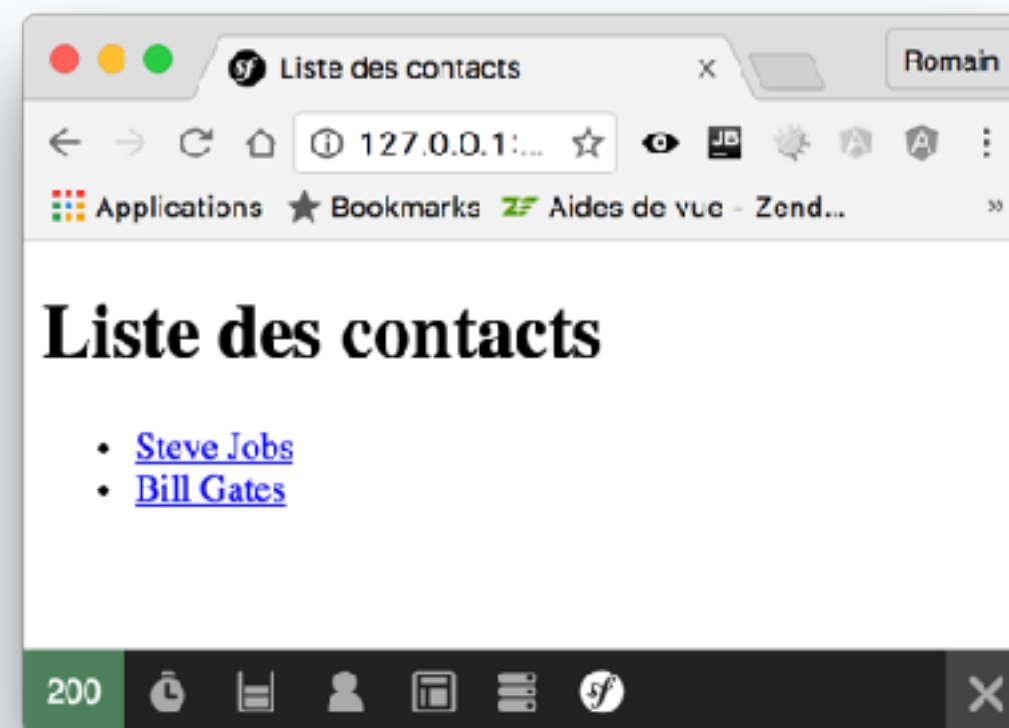
Symfony\Bundle\FrameworkBundle contient 2 classes pour faciliter les tests

- Symfony\Bundle\FrameworkBundle\Test\WebTestCase (et sa méthode `createClient` pour les tests fonctionnels)
- Symfony\Bundle\FrameworkBundle\Test\KernelTestCase (et sa méthode `bootKernel` pour les tests qui nécessitent un kernel)

```
public function listAction()
{
    $repo = $this->getDoctrine()
        ->getRepository('AppBundle:Contact');

    $contacts = $repo->findAll();

    return $this->render('list.html.twig', array(
        'contacts' => $contacts
    ));
}
```





## ► Test fonctionnels

```
<?php

namespace AppBundle\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class ContactControllerTest extends WebTestCase
{
    public function testListAvecMysql()
    {
        $client = static::createClient();

        $crawler = $client->request('GET', '/contacts/');
        $this->assertEquals(200, $client->getResponse()->getStatusCode());

        $this->assertContains('Liste des contacts', $crawler->filter('h1')->text());

        $this->assertCount(2, $crawler->filter('h1 + ul > li'));
    }
}
```

- Problème : le tests dépend d'un composant extérieur (base de données)
- Solution 1 : Réinitialiser la base de données entre chaque test (dans une méthode setup)
- Solution 2 : Utiliser les mocks



## ► Tests de formulaires

```
<?php

namespace AppBundle\Form;

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

class ContactType extends AbstractType
{
    /**
     * @param FormBuilderInterface $builder
     * @param array $options
     */
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('prenom')
            ->add('nom')
        ;
    }

    /**
     * @param OptionsResolver $resolver
     */
    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults(array(
            'data_class' => 'AppBundle\Entity>Contact'
        ));
    }
}
```



# Tests Symfony basiques - Tests de formulaires



```
<?php

namespace AppBundle\Tests\Form;

use AppBundle\Entity>Contact;
use AppBundle\Form\ContactType;
use Symfony\Component\Form\Test\TypeTestCase;

class ContactTypeTest extends TypeTestCase
{
    public function testSubmitValidData()
    {
        $formData = array(
            'prenom' => 'Romain',
            'nom' => 'Bohdanowicz',
        );

        $form = $this->factory->create(ContactType::class);

        $contact = (new Contact())->setPrenom('Romain')->setNom('Bohdanowicz');

        // submit the data to the form directly
        $form->submit($formData);

        $this->assertTrue($form->isSynchronized());
        $this->assertEquals($contact, $form->getData());

        $view = $form->createView();
        $children = $view->children;

        foreach (array_keys($formData) as $key) {
            $this->assertArrayHasKey($key, $children);
        }
    }
}
```



# Injection de Dépendance



## ► Composition

Une composition est un type d'association forte entre 2 objet. La destruction d'un objet entrainerait la destruction de l'objet associé.

Exemple : Un objet Tasse est composée de Café

```
<?php
namespace EspressoComposition;

class Cafe
{
    protected $variete;
    protected $provenance;

    public function __construct($provenance, $variete)
    {
        $this->provenance = $provenance;
        $this->variete = $variete;
    }
}
```

```
<?php
namespace EspressoComposition;

class Tasse
{
    protected $contenu;

    public function __construct() {
        $this->contenu = new Cafe("Arabica", "Mexique");
    }
}
```

```
<?php
require_once 'autoload.php';

$tasseDeCafe = new \EspressoComposition\Tasse();
```

## ► Mauvaise Pratique

La composition est désormais considéré comme une mauvaise pratique. Premièrement la classe Tasse n'est très réutilisable, elle ne peut contenir que du café. De plus il n'est pas possible d'écrire d'écrire un test unitaire de Tasse, puisqu'il faudrait en même temps tester Café.

Globalement il faut essayer de proscrire l'utilisation de new il l'intérieur d'une classe (à l'exception des Values Objects (DateTime, ArrayObject, etc...))

# Injection de Dépendance - Solution



## ► Solution

La solution est simple, pour éviter le new dans cette classe nous allons injecter la dépendance.

```
<?php
namespace EspressoInjection;

interface Liquide {
}
```

```
<?php
namespace EspressoInjection;

class Cafe implements Liquide
{
    protected $variete;
    protected $provenance;

    public function __construct($provenance, $variete)
    {
        $this->provenance = $provenance;
        $this->variete = $variete;
    }
}
```

```
<?php
namespace EspressoInjection;

class Tasse
{
    protected $contenu;

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

```
<?php
require_once 'autoload.php';

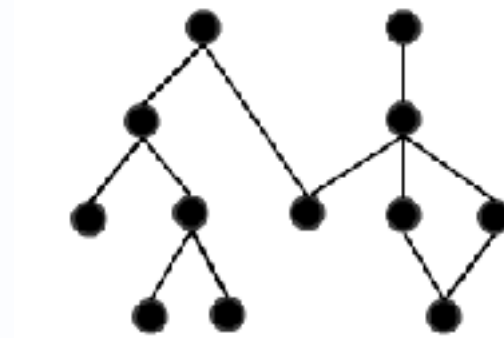
$cafe = new \EspressoInjection\Cafe();
$tasseDeCafe = new \EspressoInjection\Tasse($cafe);
```

- La classe Tasse peut désormais recevoir n'importe quel contenu qui implémente l'interface Liquide.



## ► Conteneur d'injection de dépendance (DIC)

Le problème lorsqu'on injecte les dépendances est qu'on peut parfois se retrouver avec des dépendances complexes :



► Dans ce cas il devient utile d'utiliser un conteneur d'injection de dépendance qu'on aura configuré au préalable. En PHP il existe quelques DIC connus :

- Pimple par Fabien Potencier
- Dice par Tom Butler
- PHP-DI par Matthieu Napoli
- Symfony\Container intégré Symfony2
- Zend\Di & Zend\ServiceManager intégrés ZF 2



# Conteneur Symfony



- Dans une application il arrive fréquemment que l'on repose sur des classes utilitaires, par exemple :
  - Envoi d'un email
  - Sérialisation d'un objet
  - Connexion à la base de données
  - ...
- On appelle ces objets des services (ils nous rendent des services)
- Par opposition, les autres objets qui contiennent nos données sont des Value Objects (Entity, DateTime, ArrayCollection...)



- Les services peuvent être configurées différemment
  - Génération d'un mot de passe plus ou moins sécurisé
  - Paramètres de connexion à une base de données
  - ...
- Selon l'environnement
  - En dev, on logue plus d'informations qu'en prod
  - Plus de cache sur la base de données en prod
  - ...
- Ils peuvent évoluer
  - Migration d'une base MySQL vers une base MongoDB
  - Envoi d'un mail en utilisant un API REST plutôt qu'un serveur SMTP
  - ...





- Pour permettre de faciliter leur changement dans l'application il faut :
  - qu'ils soient hautement configurables via un fichier de configuration
  - qu'ils soient eux même créés à partir de la configuration
- Autrement dit : il faut faire disparaître les 'new' dans votre code
- Le conteneur de Symfony est là pour ça



## ▸ Qu'est ce qu'un conteneur ?

- On peut voir un conteneur comme un annuaire d'objets
- C'est en quelque sorte un tableau associatif dans lequel seraient placés nos objets

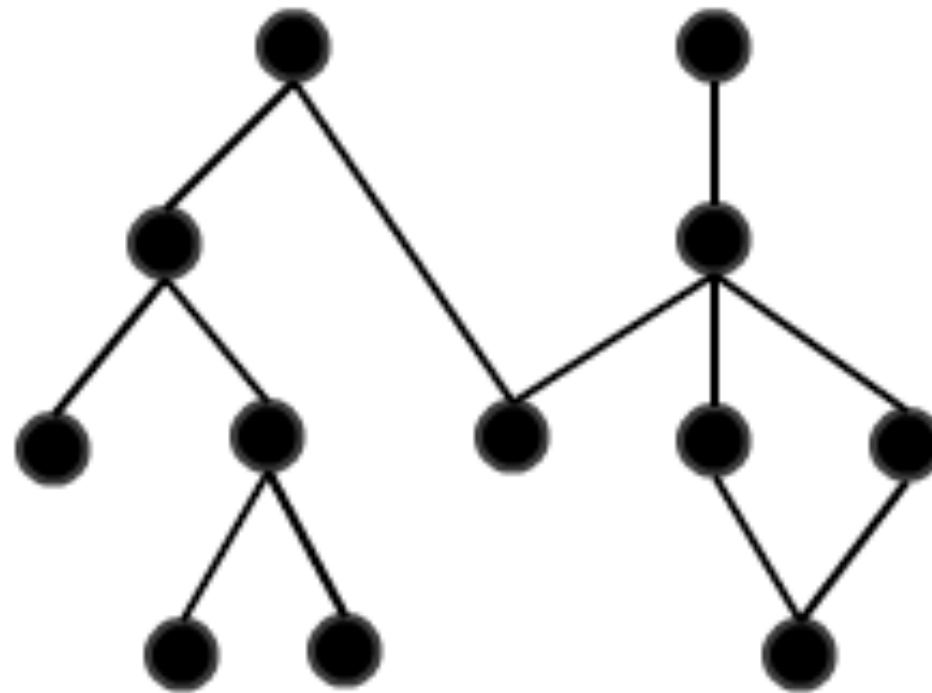
<code>doctrine.orm.default_entity_manager</code>	<code>Doctrine\ORM\EntityManager</code>
<code>form.type.text</code>	<code>Symfony\Component\Form\Extension\Core\Type\TextType</code>
<code>logger</code>	<code>Symfony\Bridge\Monolog\Logger</code>
<code>mailer</code>	<code>Swift_Mailer</code>

- En réalité les objets sont instanciés à la demande (sinon on parlerait de Registre)
- Pour y placer et récupérer nos objets on a des méthodes basiques : `get()`, `set()`, `has()`



## ▸ Dépendances

- Les objets peuvent dépendre les uns des autres et leur dépendance devrait être injectée (voir injection de dépendance)
- Symfony permet de configurer comment ses dépendances s'emboîtent les unes dans les autres, jusqu'à former un graphe





## ▸ Autowiring

- Depuis Symfony 3.3, l'ensemble des classes de AppBundle sont disponibles sous la forme de services, y compris les contrôleurs

```
services:
  _defaults:
    autowire: true
    autoconfigure: true
    public: false

  AppBundle\:
    resource: '../..'/src/AppBundle/*'
    exclude: '../..'/src/AppBundle/{Entity,Repository,Tests}'

  AppBundle\Controller\:
    resource: '../..'/src/AppBundle/Controller'
    public: true
    tags: ['controller.service_arguments']
```



## ▸ Autowiring

- Depuis Symfony 3.3, l'ensemble des classes de AppBundle sont disponibles sous la forme de services, y compris les contrôleurs

```
services:
  _defaults:
    autowire: true
    autoconfigure: true
    public: false

  AppBundle\:
    resource: '../..'/src/AppBundle/*'
    exclude: '../..'/src/AppBundle/{Entity,Repository,Tests}'

  AppBundle\Controller\:
    resource: '../..'/src/AppBundle/Controller'
    public: true
    tags: ['controller.service_arguments']
```



## ▸ Autowiring

- Depuis Symfony 3.3, l'ensemble des classes de AppBundle sont disponibles sous la forme de services, y compris les contrôleurs

```
services:
  _defaults:
    autowire: true
    autoconfigure: true
    public: false

  AppBundle\:
    resource: '../..'/src/AppBundle/*'
    exclude: '../..'/src/AppBundle/{Entity,Repository,Tests}'

  AppBundle\Controller\:
    resource: '../..'/src/AppBundle/Controller'
    public: true
    tags: ['controller.service_arguments']
```



## ► Autowiring

```
<?php

namespace AppBundle\Manager;

use AppBundle\Entity\Company;
use Doctrine\ORM\EntityManagerInterface;

class CompanyManager
{
    /**
     * @var EntityManagerInterface
     */
    protected $em;

    /**
     * ContactManager constructor.
     * @param EntityManagerInterface $em
     */
    public function __construct(EntityManagerInterface $em)
    {
        $this->em = $em;
    }
}
```



# Tests Symfony avec Mock



# Tests Symfony avec Mock - Tests Fonctionnels



```
public function testListAvecMock()
{
    $client = static::createClient();

    $contacts = [
        (new Contact())->setId(1)->setPrenom('A')->setNom('B'),
        (new Contact())->setId(2)->setPrenom('C')->setNom('D'),
        (new Contact())->setId(3)->setPrenom('E')->setNom('F'),
    ];

    $mockRepo = $this->prophesize(ContactRepository::class);
    $mockRepo->findAll()->willReturn($contacts)->shouldBeCalledTimes(1);

    $mockRegistry = $this->prophesize(Registry::class);
    $mockRegistry->getConnectionNames()->shouldBeCalledTimes(1);
    $mockRegistry->getManagerNames()->shouldBeCalledTimes(1);
    $mockRegistry->getRepository('AppBundle:Contact')->willReturn($mockRepo->reveal())->shouldBeCalledTimes(1);

    $client->getContainer()->set('doctrine', $mockRegistry->reveal());

    $crawler = $client->request('GET', '/contacts/');

    $this->assertCount(3, $crawler->filter('h1 + ul > li'));
}
```

- Problème : le contrôleur dépend de la classe Registry pour obtenir le Repository, 2 mocks à créer
- Solution : avoir une dépendance directe dans le Conteneur (couche Service par exemple)

# Tests Symfony avec Mock - Tests Fonctionnels



```
public function testListAvecMockEtServiceLayer()
{
    $client = static::createClient();

    $contacts = [
        (new Contact())->setId(1)->setPrenom('A')->setNom('B'),
        (new Contact())->setId(2)->setPrenom('C')->setNom('D'),
        (new Contact())->setId(3)->setPrenom('E')->setNom('F'),
    ];

    $mockRepo = $this->prophesize(ContactManager::class);
    $mockRepo->findAll()->willReturn($contacts)->shouldBeCalledTimes(1);

    $client->getContainer()->set('app.manager.contact', $mockRepo->reveal());

    $crawler = $client->request('GET', '/contacts/list-avec-manager');

    $this->assertCount(3, $crawler->filter('h1 + ul > li'));
}
```

```
# services.yml
services:
    app.manager.contact:
        class: AppBundle\Manager\ContactManager
        arguments: ["@doctrine.orm.entity_manager"]
```



# Doubles



- ▶ Le code PHP fait souvent appel à des composants externes :
  - Accès aux entrées/sorties
  - Accès à une base de données
  - Accès à un Service Web
- ▶ Certaines classes ne peuvent être testées de manières unitaires car elles dépendent d'autres classes.
- ▶ Solutions : les Doubles

Objets ou fonctions qui ressemblent et se comportent comme le composant qu'ils imitent, mais qui sont en réalité des versions simplifiées qui permettent de faciliter l'écriture du test.



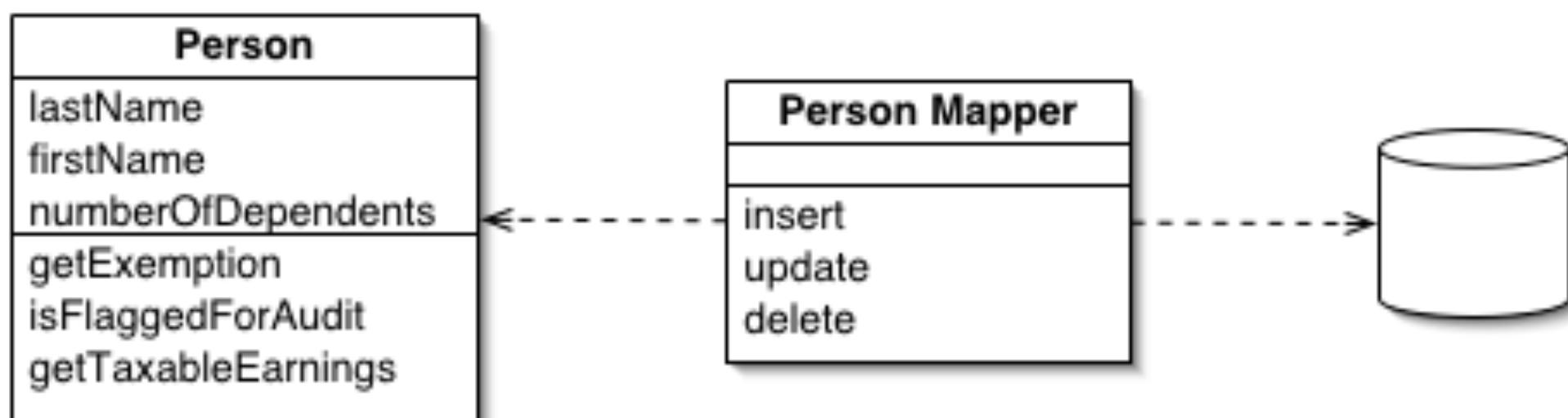
- ▶ 5 types de Doubles :
  - Fake (une classe créée par l'utilisateur qui fera les opérations en mémoire)
  - Dummy (une classe générée dont les méthodes ne font rien)
  - Stub (une classe générée dont les méthodes ont le même comportement)
  - Mock (Stub + vérification que les méthodes soient bien appelée)
  - Spy (Dummy + vérification que les méthodes soient bien appelée à posteriori)
- ▶ Bonnes pratiques :
  - Injection de Dépendance (pas de composition)
  - Registre ou Container d'injection de Dépendance

# Double - Classes d'exemple



- Un DataMapper

<http://martinfowler.com/eaCatalog/dataMapper.html>



# Double - Classes d'exemple



## ► Entité

```
<?php
namespace FormationTech\Entity;

class Database
{
    protected $name;

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

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

# Double - Classes d'exemple



## ► Gateway

```
<?php

namespace FormationTech\Gateway;

class DatabaseGateway implements DatabaseGatewayInterface
{
    protected $pdo;

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

    public function listDbs()
    {
        $stmt = $this->pdo->query('SHOW DATABASES');

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

```
<?php

namespace FormationTech\Gateway;

interface DatabaseGatewayInterface
{
    public function listDbs();
}
```



# Double - Classes d'exemple



- Mapper (classe à tester unitairement)

```
<?php

namespace FormationTech\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGatewayInterface;

class DatabaseMapper
{
    protected $gateway;

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

    public function findAll()
    {
        $dbsArray = $this->gateway->listDbs();
        $dbsObj = [];

        if (!$dbsArray) {
            return $dbsObj;
        }

        foreach ($dbsArray as $dbName) {
            $dbsObj[] = new Database($dbName);
        }

        return $dbsObj;
    }
}
```



## ► Test sans double

```
<?php

namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGateway;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{
    public function testFindAllWithMySQL()
    {
        $pdo = new \PDO('mysql:host=localhost', 'root', '');
        $gateway = new DatabaseGateway($pdo);
        $mapper = new DatabaseMapper($gateway);

        $dbs = $mapper->findAll();

        $this->assertCount(13, $dbs);
        $this->assertContainsOnlyInstancesOf(Database::class, $dbs);
    }
}
```

- Problème : changement dans la base de données ?
- Solution : fixture dans un setUp ? double ?

# Double - Fake



## ► Fake

```
<?php
namespace FormationTech\Gateway;

class DatabaseGatewayFake implements DatabaseGatewayInterface
{
    protected $dbs;

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

    public function listDbs()
    {
        return $this->dbs;
    }
}
```

```
<?php
namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGatewayFake;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{
    public function testFindAllWithFake()
    {
        $gateway = new DatabaseGatewayFake(['db1', 'db2', 'db3']);
        $mapper = new DatabaseMapper($gateway);

        $dbs = $mapper->findAll();

        $this->assertCount(3, $dbs);
        $this->assertContainsOnlyInstancesOf(Database::class, $dbs);
    }
}
```



- ▶ Sebastian Bergman à propos de l'API de Mock de PHPUnit :  
<https://thephp.cc/news/2015/02/phpunit-4-5-and-prophecy>
- ▶ L'ancien API continue d'exister pour rester compatible avec les anciens tests
- ▶ PHPUnit depuis la version 4.5 intègre un framework de test moderne : Prophecy
- ▶ Documentation  
<https://github.com/phpspec/prophecy>

# Double - Prophecy Dummy



```
<?php

namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGateway;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{
    // ...

    public function testFindAllWithDummyProphecy()
    {
        $dummy = $this->prophesize(DatabaseGateway::class);

        $mapper = new DatabaseMapper($dummy->reveal());

        $dbs = $mapper->findAll();

        $this->assertEmpty($dbs);
    }

    // ...
}
```

# Double - Prophecy Stub



```
<?php

namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGateway;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{
    // ...

    public function testFindAllWithStubProphecy()
    {
        $stub = $this->prophesize(DatabaseGateway::class);

        $stub->listDbs()->willReturn(['db1', 'db2', 'db3', 'db4']);

        $mapper = new DatabaseMapper($stub->reveal());

        $dbs = $mapper->findAll();

        $this->assertCount(4, $dbs);
        $this->assertContainsOnlyInstancesOf(Database::class, $dbs);
    }

    // ...
}
```

# Double - Prophecy Mock



```
<?php

namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGateway;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{
    // ...

    public function testFindAllWithMockProphecy()
    {
        $mock = $this->prophesize(DatabaseGateway::class);

        $mock->listDbs()->willReturn(['db1', 'db2']->shouldBeCalledTimes(1);

        $mapper = new DatabaseMapper($mock->reveal());

        $dbs = $mapper->findAll();

        $this->assertCount(2, $dbs);
        $this->assertContainsOnlyInstancesOf(Database::class, $dbs);
    }

    // ...
}
```

# Double - Prophecy Spy



```
<?php

namespace FormationTechTest\Mapper;

use FormationTech\Entity\Database;
use FormationTech\Gateway\DatabaseGateway;
use FormationTech\Mapper\DatabaseMapper;

class DatabaseMapperTest extends \PHPUnit_Framework_TestCase
{
    // ...

    public function testFindAllWithSpyProphecy()
    {
        $mock = $this->prophesize(DatabaseGateway::class);

        $mapper = new DatabaseMapper($mock->reveal());

        $dbs = $mapper->findAll();

        $this->assertEmpty($dbs);

        $mock->listDbs()->shouldHaveBeenCalledTimes(1);
    }

    // ...
}
```





- ▶ Mockery

<https://github.com/padraig/mockery>

<http://docs.mockery.io/en/latest/>

- ▶ Phake

<https://github.com/mlively/Phake>

<http://phake.readthedocs.org/en/2.1/>



# Commandes Symfony

# Commandes Symfony - Introduction



- Depuis Symfony 3, les scripts en lignes de commande d'une app sont disponible dans le répertoire `bin`
- `bin` est la valeur par défaut dans `DistributionBundle`, modifiable dans la config `composer` sous `"extra" > "symfony-bin-dir"`.
- A moins de modifier la config de `composer`, les scripts `composer` (`doctrine`, `phpunit`, etc...) s'installent sous `vendor/bin`

```
"config": {  
    "bin-dir": "bin"  
}
```

- Dans tous les cas il est important de lancer les programmes depuis la racine du projet :
  - Sous Windows : `php bin\console`
  - Sous Mac/Linux : `bin/console`

# Commandes Symfony - bin/symfony\_requirements



- Par défaut 2 programmes existent dans bin, console et symfony\_requirements
- symfony\_requirements est un programme en ligne de commande (CLI) qui vérifie que la config PHP est correcte (équivalent CLI de /config.php), une fois vérifié il est recommandé de le supprimer

```
$ bin/symfony_requirements
```

```
Symfony Requirements Checker
```

```
~~~~~
```

```
> PHP is using the following php.ini file:
```

```
  /usr/local/etc/php/7.1/php.ini
```

```
> Checking Symfony requirements:
```

```
.....
```

```
[OK]
```

```
Your system is ready to run Symfony projects
```

**Note** The command console could use a different php.ini file than the one used with your web server. To be on the safe side, please check the requirements from your web server using the `web/config.php` script.

# Commandes Symfony - bin/console



- console contient des commandes permettant d'interagir avec l'application dans un contexte Symfony (génération de code, configuration, démarrage du serveur...)

```
$ bin/console
Symfony 3.3.8 (kernel: app, env: dev, debug: true)

Usage:
  command [options] [arguments]

Options:
  -h, --help                Display this help message
  -q, --quiet               Do not output any message
  -V, --version             Display this application version
      --ansi                Force ANSI output
      --no-ansi             Disable ANSI output
  -n, --no-interaction      Do not ask any interactive question
  -e, --env=ENV             The environment name [default: "dev"]
      --no-debug            Switches off debug mode
  -v|vv|vvv, --verbose      Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and
3 for debug

Available commands:
  about                    Displays information about the current project
  help                    Displays help for a command
  list                    Lists commands
  assets
  assets:install           Installs bundles web assets under a public directory
  cache
  cache:clear              Clears the cache
  cache:pool:clear         Clears cache pools
  cache:warmup             Warms up an empty cache
  config
  config:dump-reference    Dumps the default configuration for an extension
```

# Commandes Symfony - bin/console



## debug

debug:config

Dumps the current configuration for an extension

debug:container

Displays current services for an application

debug:event-dispatcher

Displays configured listeners for an application

debug:router

Displays current routes for an application

debug:swiftmailer

[swiftmailer:debug] Displays current mailers for an application

debug:translation

Displays translation messages information

debug:twig

Shows a list of twig functions, filters, globals and tests

## doctrine

doctrine:cache:clear-collection-region

Clear a second-level cache collection region.

doctrine:cache:clear-entity-region

Clear a second-level cache entity region.

doctrine:cache:clear-metadata

Clears all metadata cache for an entity manager

doctrine:cache:clear-query

Clears all query cache for an entity manager

doctrine:cache:clear-query-region

Clear a second-level cache query region.

doctrine:cache:clear-result

Clears result cache for an entity manager

doctrine:database:create

Creates the configured database

doctrine:database:drop

Drops the configured database

doctrine:database:import

Import SQL file(s) directly to Database.

doctrine:ensure-production-settings

Verify that Doctrine is properly configured for a production environment.

doctrine:generate:crud

[generate:doctrine:crud] Generates a CRUD based on a Doctrine entity

doctrine:generate:entities

[generate:doctrine:entities] Generates entity classes and method stubs from your

## mapping information

doctrine:generate:entity

[generate:doctrine:entity] Generates a new Doctrine entity inside a bundle

doctrine:generate:form

[generate:doctrine:form] Generates a form type class based on a Doctrine entity

doctrine:mapping:convert

[orm:convert:mapping] Convert mapping information between supported formats.

doctrine:mapping:import

Imports mapping information from an existing database

doctrine:mapping:info

doctrine:query:dql

Executes arbitrary DQL directly from the command line.

doctrine:query:sql

Executes arbitrary SQL directly from the command line.

doctrine:schema:create

Executes (or dumps) the SQL needed to generate the database schema

doctrine:schema:drop

Executes (or dumps) the SQL needed to drop the current database schema

doctrine:schema:update

Executes (or dumps) the SQL needed to update the database schema to match the

## current mapping metadata.

doctrine:schema:validate

Validate the mapping files.

# Commandes Symfony - bin/console



<b>generate</b>	
generate:bundle	Generates a bundle
generate:command	Generates a console command
generate:controller	Generates a controller
<b>lint</b>	
lint:twig	Lints a template and outputs encountered errors
lint:xliff	Lints a XLIFF file and outputs encountered errors
lint:yaml	Lints a file and outputs encountered errors
<b>router</b>	
router:match	Helps debug routes by simulating a path info match
<b>security</b>	
security:check	Checks security issues in your project dependencies
security:encode-password	Encodes a password.
<b>server</b>	
server:log	Starts a log server that displays logs in real time
server:run	Runs a local web server
server:start	Starts a local web server in the background
server:status	Outputs the status of the local web server for the given address
server:stop	Stops the local web server that was started with the server:start
<b>command</b>	
<b>swiftmailer</b>	
swiftmailer:email:send	Send simple email message
swiftmailer:spool:send	Sends emails from the spool
<b>translation</b>	
translation:update	Updates the translation file

# Commandes Symfony - Lancer une commande



- ▶ Pour lancer une commande, on écrit dans un terminal :  
`bin/console NOM_DE_LA_COMMANDE`
- ▶ Exemple :
  - Sous Windows : `php bin\console server:start`
  - Sous Mac/Linux : `bin/console server:start`
- ▶ Pour afficher la doc d'une commande
  - `bin/console NOM_DE_LA_COMMANDE -h`
  - `bin/console help NOM_DE_LA_COMMANDE`



# Commandes Symfony - Lancer une commande



## ► Exemple

```
bin/console server:start -h
```

```
$ bin/console server:start -h
```

### Usage:

```
server:start [options] [--] [<addressport>]
```

### Arguments:

**addressport**                      The address to listen to (can be address:port, address, or port)

### Options:

<b>-d, --docroot=DOCR00T</b>	Document root
<b>-r, --router=ROUTER</b>	Path to custom router script
<b>--pidfile=PIDFILE</b>	PID file
<b>-h, --help</b>	Display this help message
<b>-q, --quiet</b>	Do not output any message
<b>-V, --version</b>	Display this application version
<b>--ansi</b>	Force ANSI output
<b>--no-ansi</b>	Disable ANSI output
<b>-n, --no-interaction</b>	Do not ask any interactive question
<b>-e, --env=ENV</b>	The environment name [default: "dev"]
<b>--no-debug</b>	Switches off debug mode
<b>-v vv vvv, --verbose</b>	Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

# Commandes Symfony - Lancer une commande



## ▸ Exemple (suite)

### Help:

`server:start` runs a local web server: By default, the server listens on `127.0.0.1` address and the port number is automatically selected as the first free port starting from `8000`:

```
php bin/console server:start
```

The server is run in the background and you can keep executing other commands. Execute `server:stop` to stop it.

Change the default address and port by passing them as an argument:

```
php bin/console server:start 127.0.0.1:8080
```

Use the `--docroot` option to change the default docroot directory:

```
php bin/console server:start --docroot=htdocs/
```

Specify your own router script via the `--router` option:

```
php bin/console server:start --router=app/config/router.php
```

See also: <http://www.php.net/manual/en/features.commandline.webserver.php>



- Arguments

Les arguments sont des paramètres principaux d'une commande (port pour `server:start`, chemin vers le controller à créer pour `generate:controller`)

- Options

Les options sont des paramètres secondaires, parfois globaux (environnements, verbosité...) parfois spécifiques à une commande (`docroot` pour `server:start`)

- Obligatoires, optionnels, interactives

Arguments comme options peuvent être obligatoires ou optionnels, les commandes sont parfois interactives, c'est à dire qu'elles vont vous poser des questions



## ▸ Où trouver les commandes ?

Les commandes sont définies dans des Bundles (WebServerBundle pour server ou bien DoctrineBundle pour Doctrine par exemple), il est possible que de nouvelles commandes apparaissent à l'installation de certains bundles

## ▸ Créer ses propres commandes

- Créer un répertoire Command dans un Bundle
- Y déposer une classe suffixée par Command
- La faire hériter de `Symfony\Component\Console\Command\Command` ou bien des dérivées comme `Symfony\Bundle\FrameworkBundle\Command\ContainerAwareCommand`
- Redéfinir la méthode `configure`, avec à minima un appel à `$this->setName('NOM_DE_LA_COMMANDE')`
- Redéfinir la méthode `execute`, avec le code à exécuter lors de l'appel à la commande

# Commandes Symfony - Custom Commands



## ► Exemple

```
<?php

namespace AppBundle\Command;

use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;

class HelloCommand extends Command
{
    public function configure()
    {
        $this->setName('hello');
    }

    public function execute(InputInterface $input, OutputInterface $output)
    {
        $output->writeln('Hello');
    }
}
```

```
$ bin/console hello
Hello
```

# Commandes Symfony - Custom Commands



## ► Ou en utilisant le générateur

```
$ bin/console generate:command
```

```
Welcome to the Symfony command generator
```

First, you need to give the name of the bundle where the command will be generated (e.g. **AppBundle**)

**Bundle name:** AppBundle

Now, provide the name of the command as you type it in the console (e.g. **app:my-command**)

**Command name:** hello

```
Summary before generation
```

You are going to generate a **hello** command inside **AppBundle** bundle.

**Do you confirm generation [yes]?**

```
created ./src/AppBundle/Command/
```

```
created ./src/AppBundle/Command/HelloCommand.php
```

Generated the **hello** command in **AppBundle**

```
Everything is OK! Now get to work :).
```

# Commandes Symfony - Custom Commands



- Configure avec description + argument + option

```
class HelloCommand extends ContainerAwareCommand
{
    protected function configure()
    {
        $this
            ->setName('hello')
            ->setDescription('Hello, world command')
            ->addArgument('first-name', InputArgument::OPTIONAL, 'Your first name')
            ->addOption('upper', 'u', InputOption::VALUE_NONE, 'Will output in uppercase')
        ;
    }
    // ...
}
```

# Commandes Symfony - Custom Commands



- Execute avec description + argument + option

```
class HelloCommand extends ContainerAwareCommand
{
    // ...
    protected function execute(InputInterface $input, OutputInterface $output)
    {
        $firstName = $input->getArgument('first-name') ? $input->getArgument('first-name')
: 'world';
        $outputMsg = "Hello $firstName !";

        if ($input->getOption('upper')) {
            $outputMsg = strtoupper($outputMsg);
        }

        $output->writeln($outputMsg);
    }
}
```

```
$ bin/console hello
Hello world !
$ bin/console hello Romain
Hello Romain !
$ bin/console hello Romain --upper
HELLO ROMAIN !
```



# Commandes Symfony - Custom Commands



- Accéder aux services et paramètres du container

```
<?php

class HelloCommand extends ContainerAwareCommand
{
    protected function execute(InputInterface $input, OutputInterface $output)
    {
        /** @var EntityManager $em */
        $em = $this->getContainer()->get('doctrine.orm.default_entity_manager');

        $databaseName = $this->getContainer()->getParameter('database_name');
    }
}
```



- ▶ Quand créer ses propres commandes ?
  - Besoin des arguments ou des options
  - Besoin d'afficher une doc via help ou -h
  - Besoin d'avoir accès au contexte de l'application Symfony (Container, Config...)
- ▶ Quand ne pas créer de commande ?
  - Programme serveur (envoi de mail, compression d'image ou de video), sans arguments ou options
  - Script de build, ne nécessitant pas de connaître le contexte Symfony



## ► Test de Commande

```
<?php

namespace AppBundle\Command;

use Symfony\Bundle\FrameworkBundle\Command\ContainerAwareCommand;
use Symfony\Component\Console\Input\InputArgument;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Input\InputOption;
use Symfony\Component\Console\Output\OutputInterface;

class HelloWorldCommand extends ContainerAwareCommand
{
    protected function configure()
    {
        $this->setName('hello:world')
            ->setDescription('A Hello command')
            ->addArgument('name', InputArgument::OPTIONAL, 'Your name')
            ->addOption('upper', 'u', InputOption::VALUE_NONE, 'Capitalize answer');
    }

    protected function execute(InputInterface $input, OutputInterface $output)
    {
        $name = $input->getArgument('name');
        $message = ($name) ? "Hello $name :)" : "Hello !";

        if ($input->getOption('upper')) {
            $message = strtoupper($message);
        }

        $output->writeln($message);
    }
}
```



## ► Test de Commande

```
<?php

namespace Tests\AppBundle\Command;

use AppBundle\Command\HelloWorldCommand;
use Symfony\Bundle\FrameworkBundle\Console\Application;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;
use Symfony\Component\Console\Tester\CommandTester;

class HelloWorldCommandTest extends KernelTestCase
{
    public function testExecute()
    {
        $kernel = $this->createKernel();
        $kernel->boot();

        $application = new Application($kernel);
        $application->add(new HelloWorldCommand());

        $command = $application->find('hello:world');
        $commandTester = new CommandTester($command);
        $exitCode = $commandTester->execute(array(
            'command' => $command->getName(),
            'name' => 'Romain',
            '-u' => true
        ));

        $output = $commandTester->getDisplay();
        $this->assertEquals(0, $exitCode, 'Returns 0 in case of success');
        $this->assertContains('HELLO ROMAIN :)', $output);
    }
}
```



## ► Aller plus loin :

- Documentation  
<https://symfony.com/doc/current/components/console.html>
- Ecrire en couleur  
<https://symfony.com/doc/current/console/coloring.html>
- Command Helpers (barre de progression, commandes interactives...)  
<https://symfony.com/doc/current/components/console/helpers/index.html>
- Enregistrer ses commandes comme services (injection de dépendance, tests...)  
[https://symfony.com/doc/current/console/commands\\_as\\_services.html](https://symfony.com/doc/current/console/commands_as_services.html)



- ▶ Créer une commande qui va insérer 5 contacts
- ▶ Utiliser pour cela l'EntityManager de Doctrine  
(voir slides précédentes)
- ▶ Ajouter une option --drop qui va supprimer et recréer la base de données en amont  
(voir exemple suivant :  
[https://github.com/bioub/Formation\\_PHP\\_Objets\\_Symfony\\_2016\\_08/blob/master/AddressBookSymfony/src/Prepavenir/AliceBundle/Command/AliceLoadCommand.php](https://github.com/bioub/Formation_PHP_Objets_Symfony_2016_08/blob/master/AddressBookSymfony/src/Prepavenir/AliceBundle/Command/AliceLoadCommand.php))



Doctrine\Common\Annotations



- ▶ Doctrine\Common\Annotations

Permet de manipuler des annotations personnalisés, celles qui seront utilisées plus tard dans les ORM et ODM.

- ▶ Installation

`composer require doctrine/annotations`

- ▶ Exemple

Création d'une bibliothèque permettant de générer un jeu de test et son annotation `FirstName` pour piocher aléatoire dans une liste de prénoms

```
<?php

namespace FormationTech\Entity;

use FormationTech\Annotation\FirstName;

class Contact
{
    /** @FirstName(language="en") */
    protected $prenom;

    // ... getter/setters ...
}
```





## ► Déclarer une annotation

Une annotation est une classe, qui est elle même annotée à minima avec l'annotation `@Annotation`

```
<?php

namespace FormationTech\Annotation;

/**
 * @Annotation
 */
class FirstName
{
}
```

```
<?php

namespace FormationTech\Entity;

use FormationTech\Annotation\FirstName;

class Contact
{
    /** @FirstName */
    protected $prenom;
}
```



## ▸ Annotation paramétrée

Pour paramétrer une annotation il suffit d'y déclarer des propriétés publiques

```
<?php

namespace FormationTech\Annotation;

/**
 * @Annotation
 */
class FirstName
{
    /** @var string */
    public $language = 'fr';
}
```

```
<?php

namespace FormationTech\Entity;

use FormationTech\Annotation\FirstName;

class Contact
{
    /** @FirstName(language="en") */
    protected $prenom;
}
```



- Annotation paramétrée avec un constructeur

```
/** @Annotation */
class LastName
{
    protected $languages = [];

    public function __construct(array $values)
    {
        // Example : @LastName(language="fr")
        if (isset($values['language']) && is_string($values['language'])) {
            $this->languages[] = $values['language'];
        }

        // Example : @LastName(languages={"fr", "en"})
        if (isset($values['languages']) && is_array($values['languages'])) {
            $this->languages += $values['languages'];
        }

        // Example : @LastName("fr")
        if (isset($values['value']) && is_string($values['value'])) {
            $this->languages[] = $values['value'];
        }

        // Example : @LastName({"fr", "en"})
        if (isset($values['value']) && is_array($values['value'])) {
            $this->languages += $values['value'];
        }
    }
}
```



## ► Cibler une annotation

Une annotation peut s'appliquer à une classe, une propriété, une méthode, etc... Pour cibler son utilisation on utilise l'annotation `@Target`, valeurs possibles : ALL, CLASS, METHOD, PROPERTY, ANNOTATION (pour les annotations imbriquées, cf `@JoinColumn`)

```
<?php

namespace FormationTech\Annotation;

/**
 * @Annotation
 * @Target("PROPERTY")
 */
class FirstName
{
}
```

```
<?php

namespace FormationTech\Annotation;

/**
 * @Annotation
 * @Target({"CLASS", "PROPERTY"})
 */
class Language
{
}
```



## ► Charger ses annotations

Les annotations Doctrine ne chargent pas automatiquement avec les autoloaders traditionnels (Composer, Zend, Symfony...), il faut soit les charger manuellement, soit utiliser :

`\Doctrine\Common\Annotations\AnnotationRegistry::registerAutoloadNamespace`  
(PSR-0 uniquement)

```
<?php  
  
require_once 'vendor/autoload.php';  
  
use Doctrine\Common\Annotations\AnnotationRegistry;  
  
AnnotationRegistry::registerAutoloadNamespace(  
    'FormationTech\Annotation\\', __DIR__ . '/src'  
);
```



## ▸ Lire ses annotations

On utilise une classe Reader : AnnotationReader ou SimpleAnnotationReader (permet d'éviter de faire les use), qui peuvent être décorées par FileCacheReader, CachedReader ou IndexedReader pour la performance.

```
$reader = new \Doctrine\Common\Annotations\AnnotationReader();
```

```
$reader = new \Doctrine\Common\Annotations\SimpleAnnotationReader();  
$reader->addNamespace('FormationTech\Annotation');
```

```
$reader = new CachedReader(new AnnotationReader(), new ArrayCache())
```



```
<?php

namespace FormationTech\Driver;

use Doctrine\Common\Annotations\Reader;
use FormationTech\Annotation\FirstName;
use FormationTech\Provider\FirstNameProvider;

class AnnotationDriver
{
    /** @var Reader */
    protected $reader;

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

    public function getFixture($className) {
        $class = new \ReflectionClass($className);

        $properties = $class->getProperties();

        $entity = new $className;

        foreach ($properties as $property) {
            $firstNameAnnotation = $this->reader->getPropertyAnnotation($property, FirstName::class);

            if ($firstNameAnnotation instanceof FirstName) {
                $provider = new FirstNameProvider();

                $setter = 'set' . $property->getName();

                if (!method_exists($entity, $setter)) {
                    throw new \Exception("Undefined setter $setter");
                }

                $entity->$setter($provider->getValue($firstNameAnnotation->language));
            }
        }

        return $entity;
    }
}
```

# Doctrine\Common\Annotations



```
<?php

require_once 'vendor/autoload.php';

use Doctrine\Common\Annotations\AnnotationRegistry;

AnnotationRegistry::registerAutoloadNamespace(
    'FormationTech\Annotation\\', __DIR__ . '/src'
);

$reader = new \Doctrine\Common\Annotations\SimpleAnnotationReader();
$reader->addNamespace('FormationTech\Annotation');
$driver = new \FormationTech\Driver\AnnotationDriver($reader);

$contact = $driver->getFixture(\FormationTech\Entity>Contact::class);

var_dump($contact);
```







Doctrine\Common\Cache



- ▶ Doctrine\Common\Cache  
Contient différentes implémentations de cache clés/valeurs
- ▶ Installation  
composer require doctrine/cache

```
<?php  
  
require_once 'vendor/autoload.php';  
  
$arrayCache = new \Doctrine\Common\Cache\ArrayCache();  
$arrayCache->
```

m	contains(id : string)	bool
m	delete(id : string)	bool
m	deleteAll()	bool
m	fetch(id : string)	false mixed
m	fetchMultiple(keys : array)	array mixed[]
m	flushAll()	bool
m	getNamespace()	string
m	getStats()	array null
m	save(id : string, data : mixed, [lifeTime : int = 0])	bool
m	saveMultiple(keysAndValues : array, [lifetime : int = 0])	bool
m	setNamespace(namespace : string)	void

Press ^. to choose the selected (or first) suggestion and insert a dot afterwards >> π



- Implémentations (version 1.6)
  - ApcCache (nécessite ext/apc)
  - ApcuCache (nécessite ext/apcu)
  - ArrayCache (en mémoire, durée de vie de la requête)
  - CouchbaseCache (nécessite ext/couchbase)
  - FilesystemCache (pas optimal en cas de forte concurrence)
  - MemcacheCache (nécessite ext/memcache)
  - MemcachedCache (nécessite ext/memcached)
  - MongoDBCache (nécessite ext/mongo)
  - PhpFileCache (pas optimal en cas de forte concurrence)
  - PRedisCache.php (nécessite predis/predis)
  - RedisCache.php (nécessite ext/phpredis)
  - RiakCache (nécessite ext/riak)
  - SQLite3Cache (nécessite ext/sqlite)
  - VoidCache (utile pour les tests)
  - WinCacheCache.php (nécessite ext/wincache)
  - XcacheCache.php (nécessite ext/xcache)
  - ZendDataCache.php (nécessite Zend Server Platform)



- ▶ **Norme PSR-6**

Depuis peu il existe une norme cherchant à uniformiser les systèmes de cache. Permettrait par exemple de rendre compatible zendframework/zend-cache avec doctrine/cache.

<http://www.php-fig.org/psr/psr-6/>

- ▶ **Doctrine Adapter**

Le projet PHP-Cache propose une compatibilité avec PSR-6 ce qui n'est pas le cache de doctrine + un adaptateur pour pouvoir utiliser les classes de doctrines sur une application PSR-6 et inversement.

<http://www.php-cache.com/en/latest/>



Doctrine\ORM



## ▶ Doctrine\ORM

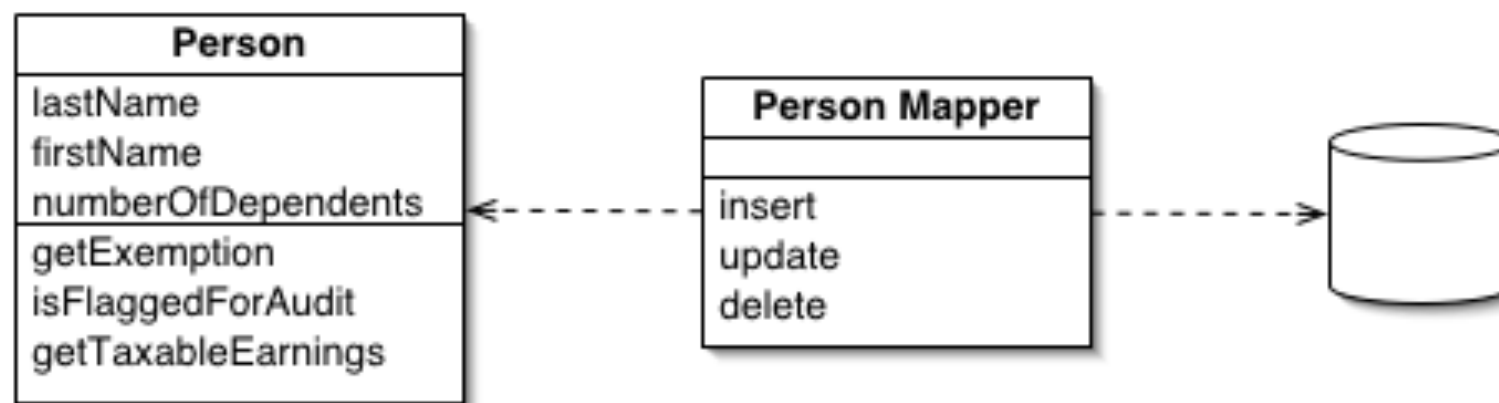
Doctrine ORM (object relational mapping) est un API permettant de faire traduire des manipulations d'objets en requêtes SQL.

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/>

## ▶ Data Mapper

Doctrine ORM est une implémentation du Design Pattern Data Mapper, c'est un composant qui permet de communiquer avec une base de données de manière objet. Il est inspiré de la bibliothèque Hibernate en Java.

<http://martinfowler.com/eaCatalog/dataMapper.html>



## ▶ Installation

`composer require doctrine/orm`



- ▶ **Doctrine\ORM\EntityManager**  
L'objet responsable de la persistance des entités.
- ▶ **Doctrine\ORM\EntityRepository**  
L'objet responsable de la récupération des entités.
- ▶ **Portabilité vers Doctrine\ODM**  
Pour être portable vers Doctrine\ODM, utiliser les interfaces Doctrine\Common\Persistence\ObjectManager et Doctrine\Common\Persistence\ObjectRepository dans les DocBlocks.



## ▸ Configuration du Mapping

Le mapping est la traduction entre le monde objet et le monde de la base de données.

## ▸ Formats possibles

- Annotations
- XML
- YAML





```
<?php

namespace AddressBook\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Contact
 *
 * @ORM\Table(name="contact", uniqueConstraints={@ORM\UniqueConstraint(name="email_UNIQUE", columns={"email"})},
indexes={@ORM\Index(name="fk_contact_societe_idx", columns={"societe_id"}), @ORM\Index(name="fk_contact_membre1_idx",
columns={"membre_id"})})
 * @ORM\Entity
 */
class Contact
{
}
}
```

- ▶ **@ORM\Entity**  
Pour déclarer la classe persistente
- ▶ **@ORM\Table(name="contact")**  
Pour déclarer à quelle table est associée cette entité  
Permet également de définir des index et contrainte unique.



## ▸ @ORM\Column

Pour lier une propriété à une colonne, attributs **name** id le nom de colonne est différent, **length** taille, **nullable** (true/false), **type** parmi :

- **string** (string <> VARCHAR)
- **integer** (int <> INT)
- **smallint** (int <> SMALLINT)
- **bigint** (string <> BIGINT)
- **boolean** (boolean)
- **decimal** (float <> DECIMAL avec des options precision et scale)
- **date** (\DateTime <> DATETIME)
- **time** (\DateTime <> TIME)
- **datetime** (\DateTime <> DATETIME/TIMESTAMP)
- **text** (string <> TEXT), object (serialize(object) <> unserialize(TEXT))
- **array** (serialize(object) <> unserialize(TEXT))
- **float** (float <> FLOAT (séparateur décimale .))

▸ Il est également possible de définir ses propres types (peut-être utiliser pour les ENUM) :

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/cookbook/mysql-enums.html>



- ▶ `@ORM\Id`

Si la colonne est la primaire.

- ▶ `@ORM\GeneratedValue(strategy="IDENTITY")`

Si la clé primaire est générée par le SGBDR (`strategy="IDENTITY"` pour MySQL/SQLite/MSSQL), (`strategy="SEQUENCE"` pour PostgreSQL/Oracle), (`strategy="AUTO"` pour être portable)



## ► Exemple :

```
/**
 * @var integer
 *
 * @ORM\Column(name="id", type="integer", nullable=false)
 * @ORM\Id
 * @ORM\GeneratedValue(strategy="IDENTITY")
 */
private $id;

/**
 * @var string
 *
 * @ORM\Column(name="prenom", type="string", length=45, nullable=false)
 */
private $prenom;

/**
 * @var \DateTime
 *
 * @ORM\Column(name="date_naissance", type="date", nullable=true)
 */
private $dateNaissance;

/**
 * @var int
 *
 * @ORM\Column(name="taille", type="integer", nullable=true)
 */
private $taille;
```



## ▸ Associations

Doctrine ORM permet de définir directement les associations entre les entités et s'occupera de faire la plupart des jointures et insertions multiples automatiquement.

- **OneToOne**

Relation 1..1

- **OneToMany**

Relation 1..n du côté 1

- **ManyToOne**

Relation 1..n du côté n (du côté de la clé étrangère)

- **ManyToMany**

Relation n..m

## ▸ Unidirectionnelle / Bidirectionnelle

La relation peut-être unidirectionnelle (une entité en connaît une autre mais l'inverse n'est pas vrai) ou bidirectionnelle (chaque entité connaît l'autre).

Les relations bidirectionnelles doivent déclarer la relation opposées (mappedBy/inversedBy).



```
class Contact
{
    /**
     * @var \Application\Entity\Membre
     *
     * @ORM\OneToOne(targetEntity="AddressBook\Entity\Membre")
     * @ORM\JoinColumns({
     *     @ORM\JoinColumn(name="membre_id", referencedColumnName="id")
     * })
     */
    private $membre;

    /**
     * @var \Application\Entity\Societe
     *
     * @ORM\ManyToOne(targetEntity="Application\Entity\Societe", inversedBy="contacts")
     * @ORM\JoinColumns({
     *     @ORM\JoinColumn(name="societe_id", referencedColumnName="id")
     * })
     */
    private $societe;

    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\ManyToMany(targetEntity="Application\Entity\Association")
     * @ORM\JoinTable(name="adhesion",
     *     joinColumns={
     *         @ORM\JoinColumn(name="contact_id", referencedColumnName="id")
     *     },
     *     inverseJoinColumns={
     *         @ORM\JoinColumn(name="association_id", referencedColumnName="id")
     *     }
     * )
     */
    private $associations;
}
```



## ► Exemple AddressBook\Entity\Societe :

```
class Societe
{
    // ...

    /**
     * @var \Doctrine\Common\Collections\Collection
     *
     * @ORM\OneToMany(targetEntity="Application\Entity>Contact", mappedBy="societe")
     */
    private $contacts;

    /**
     * Constructor
     */
    public function __construct()
    {
        $this->contacts = new \Doctrine\Common\Collections\ArrayCollection();
    }
}
```



## ► bootstrap.php

Création de l'EntityManager (\$isDevMode pour désactiver le cache)

```
<?php
// bootstrap.php
require_once "vendor/autoload.php";

// Create a simple "default" Doctrine ORM configuration for XML Mapping
$isDevMode = true;
$config =
\Doctrine\ORM\Tools\Setup::createAnnotationMetadataConfiguration(array(__DIR__."/
src"), $isDevMode);
// or if you prefer yaml or annotations
// $config = Setup::createXMLMetadataConfiguration(array(__DIR__."/config/xml"),
$isDevMode);
// $config = Setup::createYAMLMetadataConfiguration(array(__DIR__."/config/yaml"),
$isDevMode);

// database configuration parameters
$conn = array(
    'driver' => 'pdo_sqlite',
    'path' => __DIR__ . '/db.sqlite',
);

// obtaining the entity manager
$entityManager = \Doctrine\ORM\EntityManager::create($conn, $config);
```





## ► cli-config.php

Pour pouvoir utiliser les binaires de Doctrine\ORM il faut créer un fichier cli-config à la racine du projet ou dans un répertoire config.

```
<?php
// cli-config.php
require_once "bootstrap.php";

$helperSet = new \Symfony\Component\Console\Helper\HelperSet(array(
    'em' => new \Doctrine\ORM\Tools\Console\Helper\EntityManagerHelper($entityManager),
    'db' => new
\Doctrine\DBAL\Tools\Console\Helper\ConnectionHelper(\Doctrine\DBAL\DriverManager::getConnection($conn))
));

return $helperSet;
```



## ► Insertions

```
<?php
// create_user.php
require_once "bootstrap.php";

$user = new User();
$user->setName('Romain');

$entityManager->persist($user);
$entityManager->flush();

echo "Created User with ID " . $user->getId() . "\n";
```



## ► Recherches

```
<?php
// list_bugs.php
require_once "bootstrap.php";

$dql = "SELECT b, e, r FROM Bug b JOIN b.engineer e JOIN b.reporter r ORDER BY b.created
DESC";

$query = $entityManager->createQuery($dql);
$query->setMaxResults(30);
$bugs = $query->getResult();

foreach($bugs AS $bug) {
    echo $bug->getDescription()." - ".$bug->getCreated()->format('d.m.Y')."\\n";
    echo "    Reported by: ".$bug->getReporter()->getName()."\\n";
    echo "    Assigned to: ".$bug->getEngineer()->getName()."\\n";
    foreach($bug->getProducts() AS $product) {
        echo "        Platform: ".$product->getName()."\\n";
    }
    echo "\\n";
}
```



## ► extends Doctrine\ORM\EntityRepository

Parfois les requêtes ne peuvent pas s'écrire où ne sont pas bien optimisés, il faut alors créer une classe Repository qui héritera de Doctrine\ORM\EntityRepository  
Les requêtes peuvent alors s'écrire :

- En SQL

Il faudra alors expliquer à Doctrine comment cette requête se traduit en entité

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/native-sql.html>

- Avec le QueryBuilder

Equivalent de Zend\Db\Sql

<http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html>

- En DQL

Language propre à Doctrine proche de SQL mais manipulant des entités.

# Doctrine - Requêtes « complexes »



## ► Exemple

Dans `showAction` du contrôleur `Contact`, nous requérons une entité par sa clé primaire. Dans la vue nous demandons d'accéder à sa Société liée.

Par défaut Doctrine fait 2 requêtes SQL, nous aimerions en faire 1 seule requêtes avec une jointure.

```
// AddressBook/Controller/ContactController
public function showAction()
{
    $id = $this->params("id");

    $contact = $this->getRepository()->find($id);

    return new ViewModel(
        array(
            "contact" => $contact
        )
    );
}
```

```
<!-- view/address-book/contact/show.phtml -->
<?=$this->escapeHtml($contact->getSociete()->getNom())?>
```

» DoctrineORMModule

» Queries for doctrine.sql\_logger\_collector.orm\_default

SQL:

```
SELECT t0.id AS id1, t0.prenom AS prenom2, t0.nom AS nom3, t0.telephone AS
telephone4, t0.email AS email5, t0.membre_id AS membre_id6, t0.societe_id AS
societe_id7 FROM contact t0 WHERE t0.id = ?
```

Params: 0 => string '1' (length=1)

Types: 0 => string 'integer' (length=7)

Time: 0.00042605400085449

SQL:

```
SELECT t0.id AS id1, t0.nom AS nom2, t0.ville AS ville3 FROM societe t0
WHERE t0.id = ?
```

Params: 0 => int 3



2 queries in 876.19 µs



4 mappings

# Doctrine - Requêtes « complexes »



## ► Exemple

Avec le Repository, on obtient une seule requête (temps d'exécution divisé par 2).

```
// AddressBook/Entity/Repository/ContactRepository

class ContactRepository extends EntityRepository
{
    public function findWithSociete($id)
    {
        $dql = "SELECT c, s
                FROM AddressBook\Entity\Contact c
                LEFT JOIN c.societe s
                WHERE c.id = :id";

        return $this->getEntityManager()->createQuery($dql)
            ->setParameter("id", $id)
            ->getSingleResult();
    }
}
```

```
// AddressBook/Controller/ContactController

public function showAction()
{
    $id = $this->params("id");

    $contact = $this->getRepository()->findWithSociete($id);

    return new ViewModel(
        array(
            "contact" => $contact
        )
    );
}
```

```
» DoctrineORMModule
» Queries for doctrine.sql_logger_collector.orm_default

SQL:
SELECT c0_.id AS id0, c0_.prenom AS prenom1, c0_.nom AS nom2,
c0_.telephone AS telephone3, c0_.email AS email4, s1_.id AS id5, s1_.nom AS
nom6, s1_.ville AS ville7, c0_.membre_id AS membre_id8, c0_.societe_id AS
societe_id9 FROM contact c0_ LEFT JOIN societe s1_ ON c0_.societe_id =
s1_.id WHERE c0_.id = ?

Params:          0 => string '1' (length=1)
Types:           0 => int 2
Time:            0.0004878044128418

→ 1 queries in 487.80 µs → 4 mappings
```