

# DI is very easy in **Symfony 2**



# Symfony

# DI ist very easy in Symfony 2

- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

# DI ist very easy in Symfony 2

- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

```
# app/config/config.yml
# ...
services:
```

```
    my_service:
        class: Acme\MyBundle\Service\AwesomeClass
```

# DI ist very easy in Symfony 2

- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

```
# app/config/config.yml
# ...
services:
```

```
  my_service:
    class: Acme\MyBundle\Service\AwesomeClass
```

Class to manage

Name of the new service

# # php app/console container:debug

```
timon@moby: ~/www/quickstart.git
timon@moby:~/www/quickstart.git$ app/console container:debug
[container] Public services
Service Id      Scope   Class Name
acme.demo.listener  container  Acme\DemoBundle\EventListener\ControllerListener
annotation_reader  container  Doctrine\Common\Annotations\FileCacheReader
assetic.asset_manager  container  Assetic\Factory\LazyAssetManager
assetic.controller  prototype  Symfony\Bundle\AsseticBundle\Controller\AsseticController
assetic.filter.cssrewrite  container  Assetic\Filter\CssRewriteFilter
assetic.filter_manager  container  Symfony\Bundle\AsseticBundle\FilterManager
assetic.request_listener  container  Symfony\Bundle\AsseticBundle\EventListener\RequestListener
cache_clearer      container  Symfony\Component\HttpKernel\CacheClearer\ChainCacheClearer
cache_warmer       container  Symfony\Component\HttpKernel\CacheWarmer\CacheWarmerAggregate
data_collector.request  container  Symfony\Component\HttpKernel\DataCollector\RequestDataCollector
data_collector.router  container  Symfony\Bundle\FrameworkBundle\DataCollector\RouterDataCollector
database_connection  n/a       alias for doctrine.dbal.default_connection
debug.controller_resolver  container  JMS\DiExtraBundle\HttpKernel\ControllerResolver
debug.event_dispatcher  n/a       alias for event_dispatcher
debug.stopwatch    container  Symfony\Component\HttpKernel\Debug\Stopwatch
debug.templating.engine.twig  n/a       alias for templating
doctrine           container  Doctrine\Bundle\DoctrineBundle\Registry
doctrine.dbal.connection_factory  container  Doctrine\Bundle\DoctrineBundle\ConnectionFactory
doctrine.dbal.default_connection  container  stdClass
doctrine.orm.default_entity_manager  container  EntityManager50bb425e4f655_546a8d27f194334ee012bfe64f629947b07e4919\_CG\_D
doctrine\ORM\EntityManager
doctrine.orm.default_manager_configurator  container  Doctrine\Bundle\DoctrineBundle\ManagerConfigurator
doctrine.orm.entity_manager  n/a       alias for doctrine.orm.default_entity_manager
doctrine.orm.validator.unique  container  Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntityValidator
doctrine.orm.validator_initializer  container  Symfony\Bridge\Doctrine\Validator\DoctrineInitializer
event_dispatcher   container  Symfony\Component\HttpKernel\Debug\ContainerAwareTraceableEventDispatcher
file_locator       container  Symfony\Component\HttpKernel\Config\FileLocator
filesystem        container  Symfony\Component\Filesystem\Filesystem
form.csrf_provider  container  Symfony\Component\Form\Extension\CsrfProvider\SessionCsrfProvider
form.factory       container  Symfony\Component\Form\FormFactory
form.registry      container  Symfony\Component\Form\FormRegistry
form.resolved_type_factory  container  Symfony\Component\Form\ResolvedFormTypeFactory
form.type.birthday  container  Symfony\Component\Form\Extension\Core\Type\BirthdayType
form.type.checkbox  container  Symfony\Component\Form\Extension\Core\Type\CheckboxType
form.type.choice    container  Symfony\Component\Form\Extension\Core\Type\ChoiceType
form.type.collection  container  Symfony\Component\Form\Extension\Core\Type\CollectionType
form.type.country   container  Symfony\Component\Form\Extension\Core\Type\CountryType
form.type.date      container  Symfony\Component\Form\Extension\Core\Type\DateType
form.type.datetime  container  Symfony\Component\Form\Extension\Core\Type\DateTimeType
form.type.email     container  Symfony\Component\Form\Extension\Core\Type\EmailType
form.type.entity    container  Symfony\Bridge\Doctrine\Form\Type\EntityType
```

# DI ist very easy in Symfony 2

- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

```
# app/config/config.yml
# ...
services:
    my_service:
        class: Acme\MyBundle\Service\AwesomeClass
```

# DI ist very easy in Symfony 2

- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

```
# app/config/config.yml
# ...
services:
    my_service:
        class:      Acme\MyBundle\Service\AwesomeClass
        arguments:
            some_arg:      "string"
            another:
                - arry_member
                - arry_member
            even_more:    @another_service
```



# DI ist very easy in Symfony 2

- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a “Service”

```
# app/config/config.yml
# ...
services:
  my_service:
    class: Acme\MyService
    arguments:
      some_arg: "string"
      another:
        - array_member
        - array_member
    even_more: @another_service
```

Arguments can be strings, numbers, arrays, placeholders, and many more ...

Any other service can be injected as argument



```

<?php
use    Guzzle\Http\ClientInterface;
use    Acme\Logger\LoggerInterface;

class FeedAggregator {
    private $client;
    private $logger;

    public function __construct(ClientInterface $client,
                                LoggerInterface $logger) {
        $this->client = $client;
        $this->logger = $logger;
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $request->send();
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: '.$host.$path);
            return null;
        }

        return $response->getBody();
    }
    // ...
}

```

```
<?php
use Guzzle\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

class FeedAggregator {
    private $client;
    private $logger;

    public function __construct(ClientInterface $client,
                               LoggerInterface $logger) {
        $this->client = $client;
        $this->logger = $logger;
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $request->send();
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: ' . $host . $path);
        }
    }
}
```

```
# app/config/config.yml
# ...
services:
    feed_aggregator:
        class:      Acme\FeedBundle\Service\FeedAggregator
        arguments:
            client:      @http_client
            logger:      @logger
```

```
<?php
use Guzzle\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

class FeedAggregator {
    private $client;
    private $logger;

    public function __construct(ClientInterface $client,
                               LoggerInterface $logger) {
        $this->client = $client;
        $this->logger = $logger;
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $request->send();
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: ' . $host . $path);
        }
    }
}
```

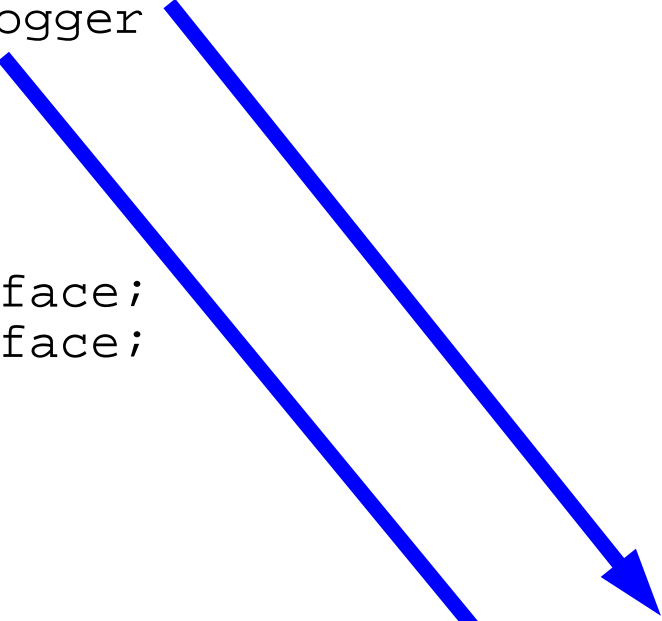
```
# app/config/config.yml
# ...
services:
    feed_aggregator:
        class:      Acme\FeedBundle\Service\FeedAggregator
        arguments:
            client:  @http_client
            logger:  @logger
```

```
<?php
use Guzzle\Http\ClientInterface;
use Acme\Logger\LoggerInterface;

class FeedAggregator {
    private $client;
    private $logger;

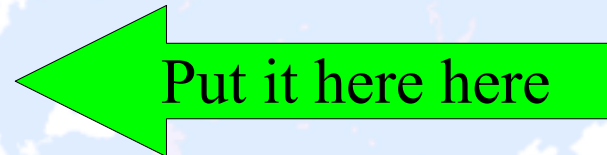
    public function __construct(ClientInterface $client,
                               LoggerInterface $logger) {
        $this->client = $client;
        $this->logger = $logger;
    }

    public function retrieveFeed ($baseurl, $path) {
        $request = $this->client->setBaseUrl($baseurl)->get($path);
        $response = $request->send();
        if (200 != $response->getStatusCode()) {
            $this->logger->log('Could not get: ' . $host . $path);
        }
    }
}
```



# Different Config for Testing?

- `app/config/config.yml`
- `app/config/config_dev.yml`
- `app/config/config_test.yml`



# Nachhaltige Architektur für große Web-Projekte

- Modularität
- Erweiterbarkeit
- Wartbarkeit
- ★ Testbarkeit
- Performance





# Summary

- Our classes only depend on interfaces



# Summary

- Our classes only depend on interfaces
- All implementation classes are instantiated and provided (injected) by the DIC

# Summary

- Our classes only depend on interfaces
- All implementation classes are instantiated and provided (injected) by the DIC
- Our classes create only domain entities, exceptions and value objects

# Summary

- Our classes only depend on interfaces
- All implementation classes are instantiated and provided (injected) by the DIC
- Our classes create only domain entities, exceptions and value objects
- The DIC is not passed to any model class, domain entity, exception or value object
- Controllers can access the DIC to obtain services

# Further Reading

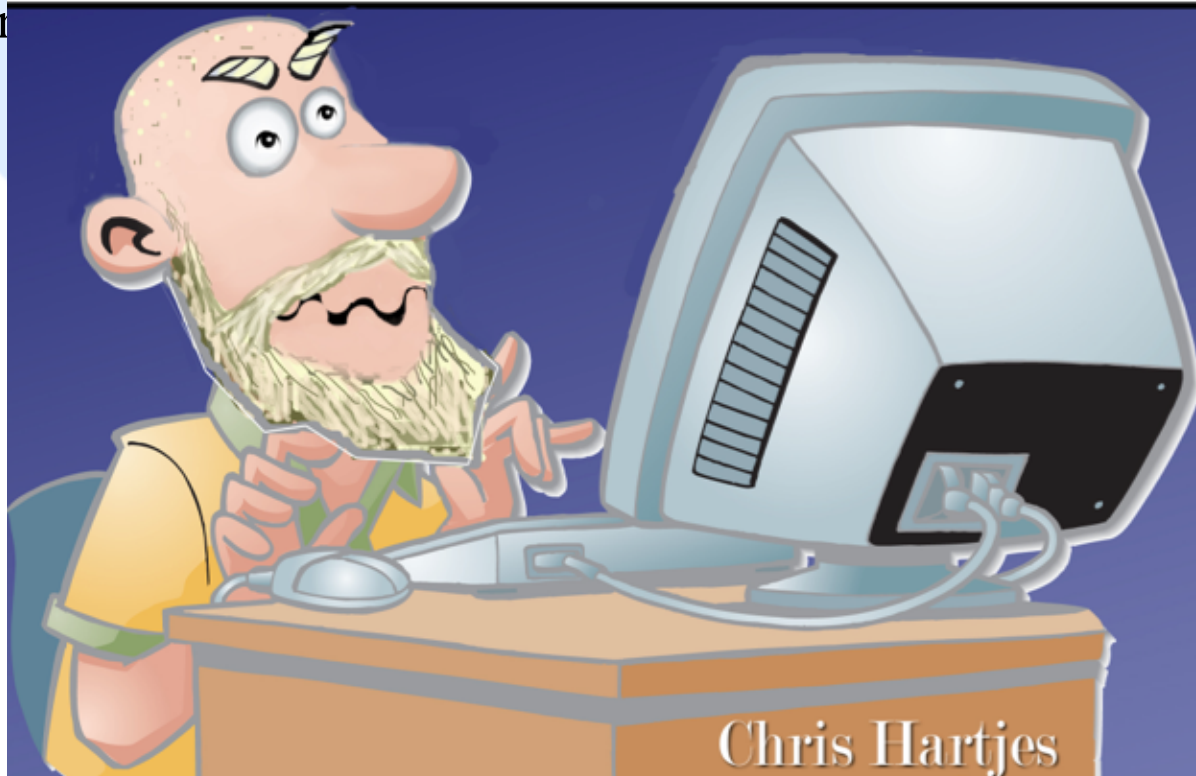
- <http://fabien.potencier.org/article/11/what-is-dependency-injection>
- [http://symfony.com/doc/current/components/dependency\\_injection/compilation.html](http://symfony.com/doc/current/components/dependency_injection/compilation.html)
- [http://symfony.com/doc/current/cookbook/service\\_container/compiler\\_passes.html](http://symfony.com/doc/current/cookbook/service_container/compiler_passes.html)
- Use the source ...

# The Grumpy Programmer's

# Guide To Building Testable Applications In PHP

- <http://fabien>
- <http://symfo>
- <http://symfo>
- Use the source

[ompileation.html](#)  
[er\\_passes.html](#)





**Vielen Dank  
für Eure Aufmerksamkeit!**

# SOLID

- S Single Responsibility Principle
- O Open / Close Principle
- L Liskov Substitution Principle
- I Interface Segregation Principle
- D Dependency Inversion Principle