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

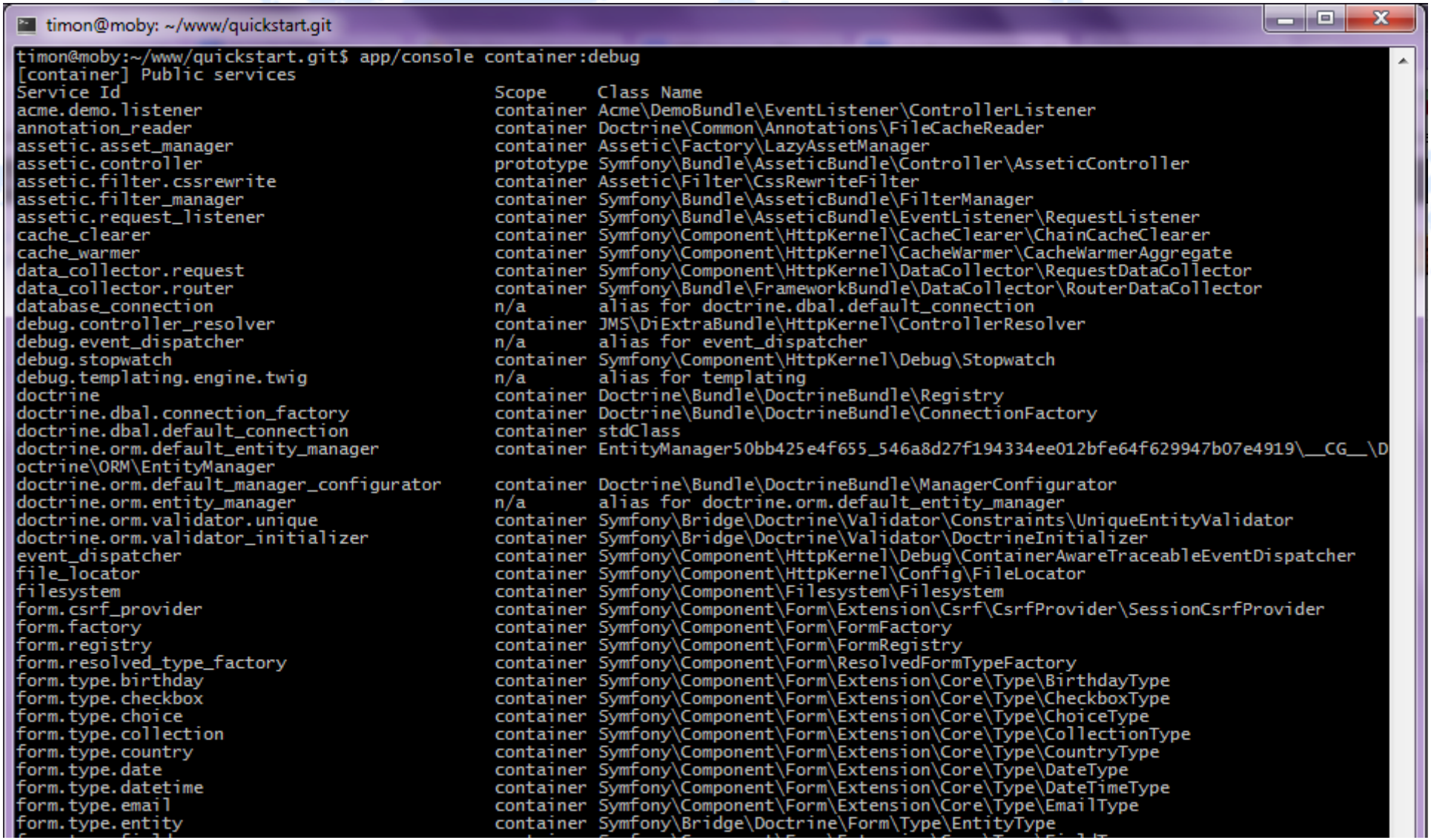
- Any ordinary PHP class can be managed by DIC
- Only 2 lines of configuration per class are needed
- Any class managed by the DIC is called a "Service"

```
# 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



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\...\meClass
        arguments:
            some_arg:     "string"
            another:
                - arry_member
                - arry_member
            even_more:    @another_service
```

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

Any other service can be injected as as argument

```php
<?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
<?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);
```

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

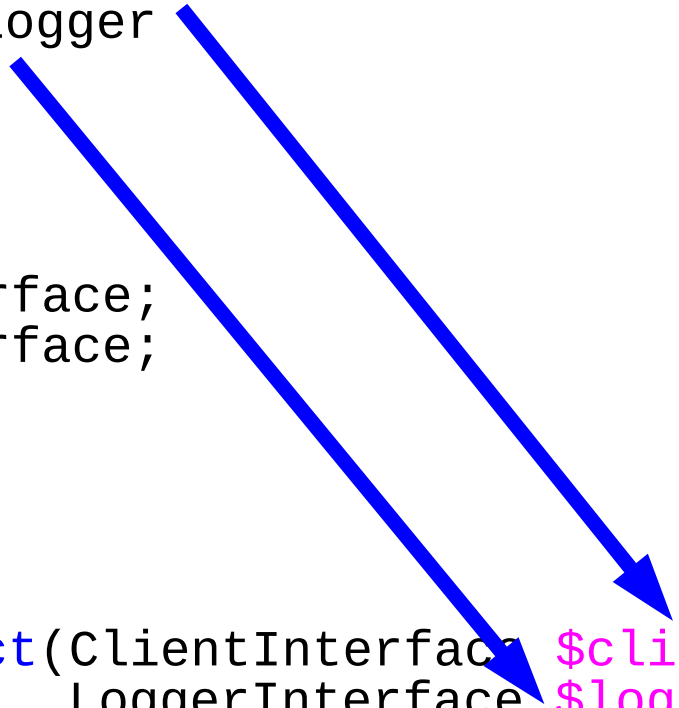
```php
<?php
use   Guzzle\Http\ClientInterface;
use   Acme\Logger\LoggerInterface;

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);
```

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



```php
<?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 instanciated and provided (injected) by the DIC

Summary

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

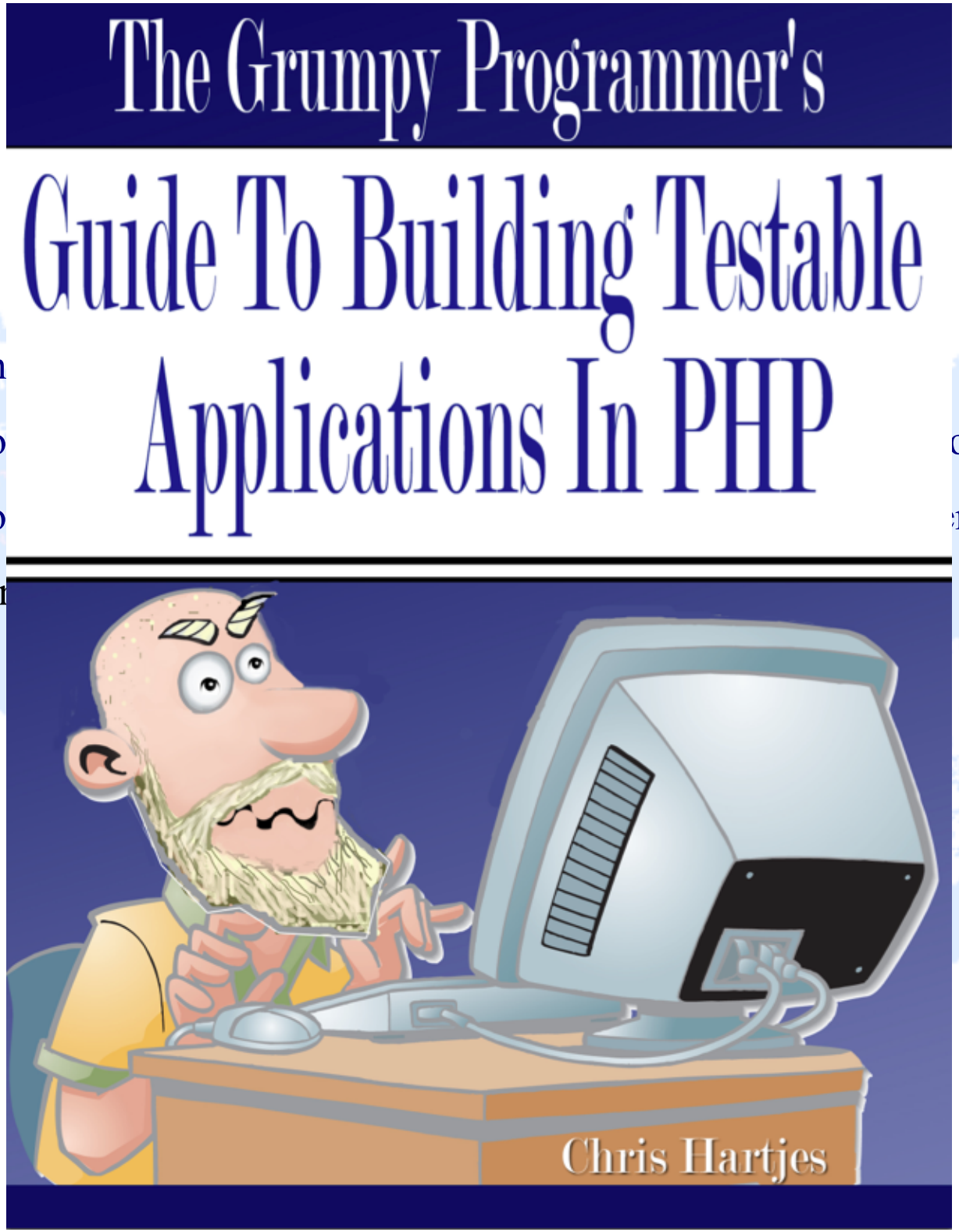
Summary

- Our classes only depend on interfaces
- All implementation classes are instanciated 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/cookbook/service_container/compiler_passes.html
- Use the source ...

- http://fabien
- http://symfo ompilation.html
- http://symfo r_passes.html
- Use the sour



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