Advanced OO Patterns

OSI Days 2011

Tobias Schlitt (@tobySen)

November 21, 2011

Outline

About me

- Degree in computer sience
- Prof. PHP since 1999
- Open source enthusiast
- FLOSS contributor

About me

- Degree in computer sience
- Prof. PHP since 1999
- Open source enthusiast
- FLOSS contributor

Co-founder of

About me

- Degree in computer sience
- Prof. PHP since 1999
- Open source enthusiast
- FLOSS contributor

Co-founder of

Helping teams to create high quality PHP software.

About me

- Degree in computer sience
- Prof. PHP since 1999
- Open source enthusiast
- FLOSS contributor

Co-founder of

Helping teams to create high quality PHP software.

`http://qafoo.com`

Disclaimer

- This talk cannot be in depth

Disclaimer

- This talk cannot be in depth
- This talk shall inspire you

Disclaimer

- This talk cannot be in depth
- This talk shall inspire you
- This talk will not show UML diagrams

Disclaimer

- This talk cannot be in depth
- This talk shall inspire you
- This talk will not show UML diagrams
- This talk will show you quite some code

Disclaimer

- This talk cannot be in depth
- This talk shall inspire you
- This talk will not show UML diagrams
- This talk will show you quite some code
- This talk can seriously harm your coding habbits

Patterns are . . .

. . . names for proven ideas how a certain class of problems can be solved.

Patterns are **not** . . .

- . . . appliable to every problem.
- . . . directly transferable to code.
- . . . written in stone.
- . . . always the best solution.

Pattern classification

- Creational
- Structural
- Behavioural
- Architectural

Well known patterns

Signal / Observer

Iterator

Visitor

Adapter

Singleton

Factory

Well known patterns

Signal / Observer

Iterator

Visitor

Adapter

~~Singleton~~

Factory

Outline

Goals of good OO design

- Modular

Goals of good OO design

- Modular
- Flexible

Goals of good OO design

- Modular
- Flexible
- Reusable

Goals of good OO design

- Modular
- Flexible
- Reusable
- Testable

Goals of good OO design

- Modular
- Flexible
- Reusable
- Testable
- S.O.L.I.D.

  (`http://butunclebob.com/ArticleS.UncleBob.PrinciplesOfOod`)

What's the problem here?

```
1   <?php
2
3   class MessageDispatcher
4   {
5       public function __construct()
6       {
7           $this->messengers[] = new
                JabberMessenger();
8           $this->messengers[] = new
                MailMessenger();
9       }
10  }
11
12  class MailMessenger implements Messenger
13  {
14      public function sendMessage( $text )
15      {
16          myLogger::getInstance()->log( $text
                );
17          $this->sendMail( /*...*/ );
18      }
19  }
```

What's the problem here?

```php
<?php

class MessageDispatcher
{
    public function __construct()
    {
        $this->messengers[] = new
            JabberMessenger();
        $this->messengers[] = new
            MailMessenger();
    }
}

class MailMessenger implements Messenger
{
    public function sendMessage( $text )
    {
        myLogger::getInstance()->log( $text
            );
        $this->sendMail( /*...*/ );
    }
}
```

- Inflexible

What's the problem here?

```php
<?php

class MessageDispatcher
{
    public function __construct()
    {
        $this->messengers[] = new
            JabberMessenger();
        $this->messengers[] = new
            MailMessenger();
    }
}

class MailMessenger implements Messenger
{
    public function sendMessage( $text )
    {
        myLogger::getInstance()->log( $text
            );
        $this->sendMail( /*...*/ );
    }
}
```

- Inflexible
- Not reusable

What's the problem here?

```php
<?php

class MessageDispatcher
{
    public function __construct()
    {
        $this->messengers[] = new
            JabberMessenger();
        $this->messengers[] = new
            MailMessenger();
    }
}

class MailMessenger implements Messenger
{
    public function sendMessage( $text )
    {
        myLogger::getInstance()->log( $text
            );
        $this->sendMail( /*...*/ );
    }
}
```

- Inflexible
- Not reusable
- Hardly testable

Injecting dependencies

```
1  <?php
2
3  $messenger = new MessageDispatcher(

19 );
```

Injecting dependencies

```
1  <?php
2
3  $messenger = new MessageDispatcher(
4      array(
5          new JabberMessenger( 'jabber.example.org', 'user', 'pass' ),
6          new MailMessenger(
16         )
17     ),
19 );
```

Injecting dependencies

```
1   <?php
2
3   $messenger = new MessageDispatcher(
4       array(
5           new JabberMessenger( 'jabber.example.org', 'user', 'pass' ),
6           new MailMessenger(
7               new MailSmtpTransport( 'mail.example.org', 'user', 'pass' ),
16          )
17      ),
19  );
```

Injecting dependencies

```
1  <?php
2
3  $messenger = new MessageDispatcher(
4      array(
5          new JabberMessenger( 'jabber.example.org', 'user', 'pass' ),
6          new MailMessenger(
7              new MailSmtpTransport( 'mail.example.org', 'user', 'pass' ),
8              $logger = new Logger(
15             )
16         )
17     ),
18     $logger
19 );
```

Injecting dependencies

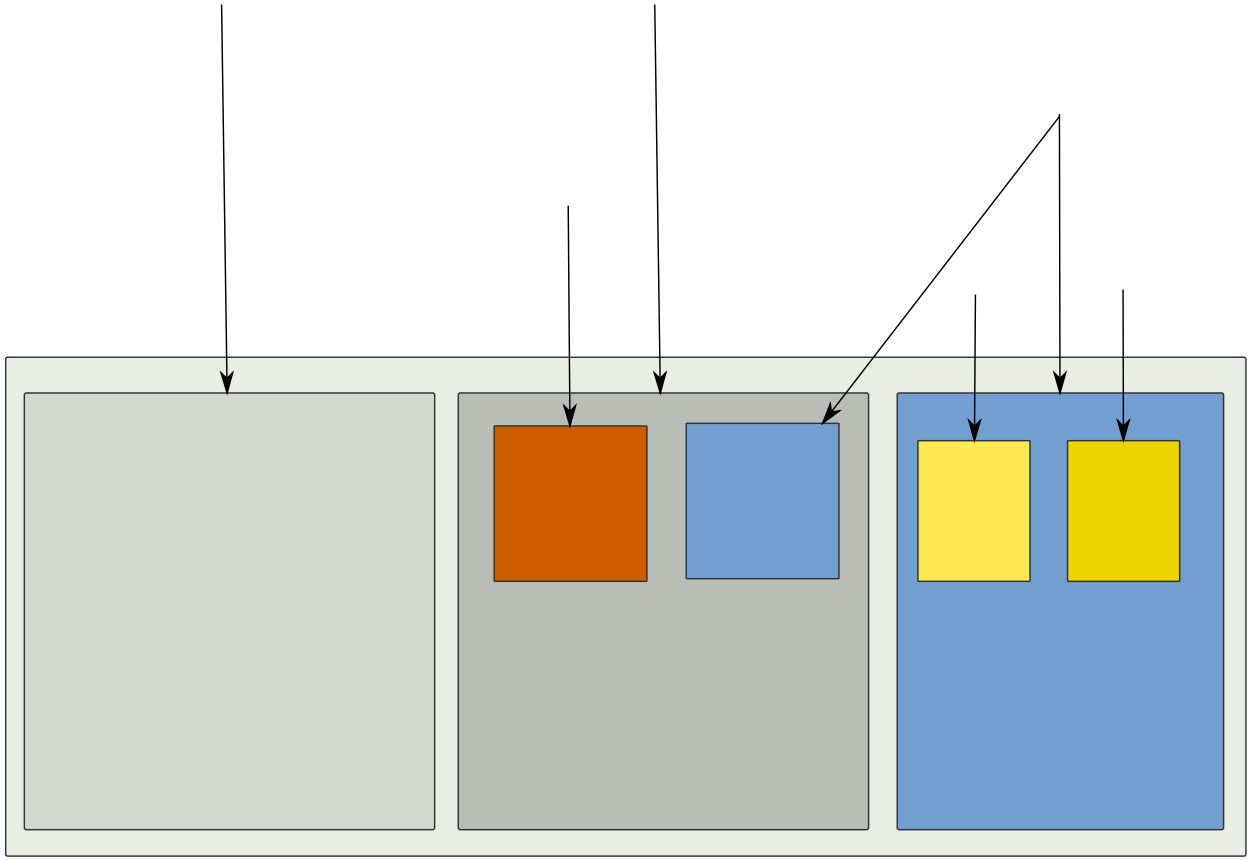
```
1  <?php
2
3  $messenger = new MessageDispatcher(
4      array(
5          new JabberMessenger( 'jabber.example.org', 'user', 'pass' ),
6          new MailMessenger(
7              new MailSmtpTransport( 'mail.example.org', 'user', 'pass' ),
8              $logger = new Logger(
9                  new LoggingDispatcher(
14                 )
15             )
16         )
17     ),
18     $logger
19 );
```

Injecting dependencies

```php
<?php

$messenger = new MessageDispatcher(
    array(
        new JabberMessenger( 'jabber.example.org', 'user', 'pass' ),
        new MailMessenger(
            new MailSmtpTransport( 'mail.example.org', 'user', 'pass' ),
            $logger = new Logger(
                new LoggingDispatcher(
                    array(
                        new SyslogLogger(),
                        new FileSystemLogger( 'log/errors.log' )
                    )
                )
            )
        )
    ),
    $logger
);
```

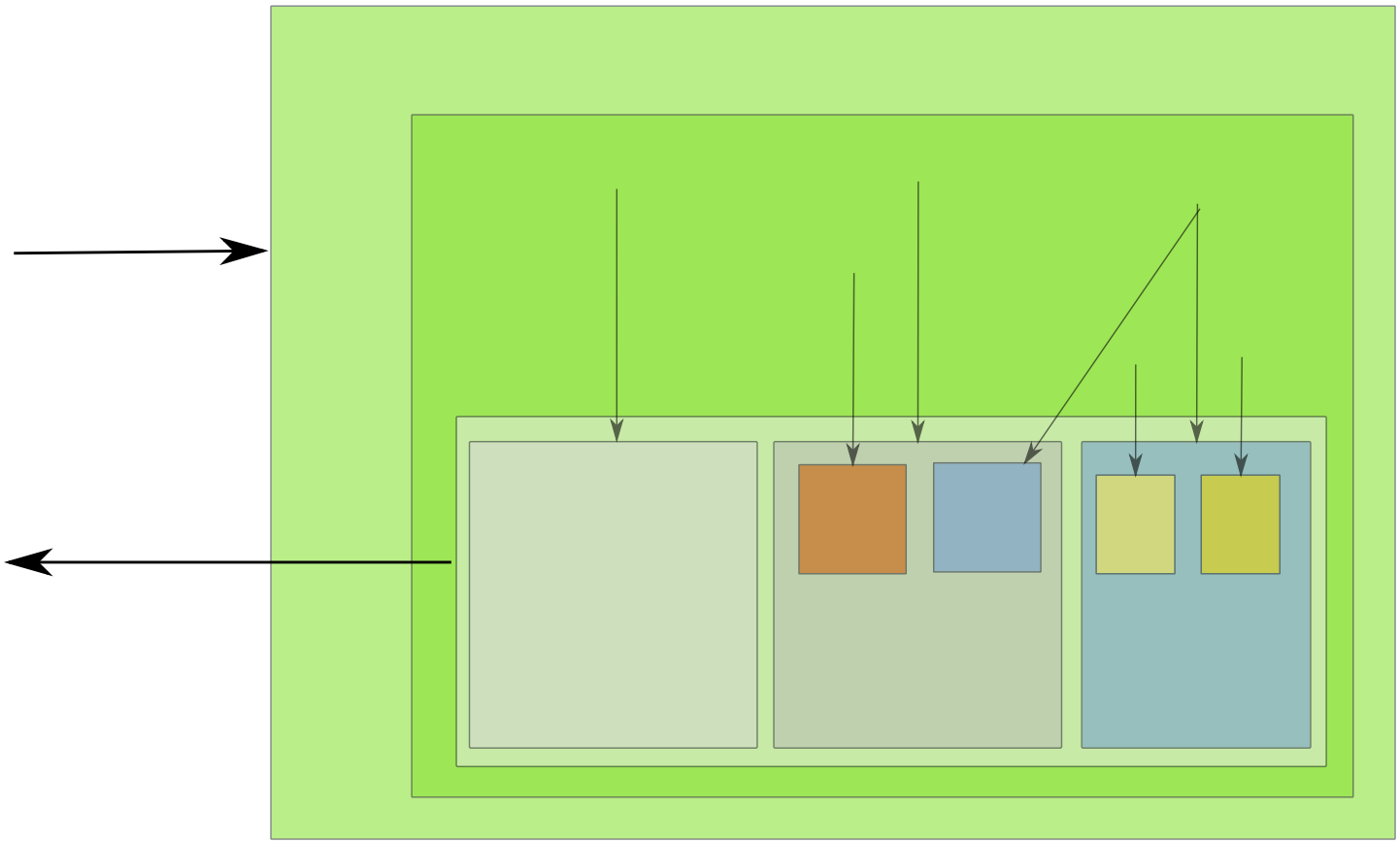
DI visualized



What is desired?

- Inject all dependencies
- Separate object construction from logic

DI visualized



What is desired?

- Don't have ctors all over the place
- Manage dependencies
- Replace dependencies fine grained
- Lazy initialization

Builders

```php
<?php

class MessageDispatcherBuilder
{
    protected $logBuilder;
    protected $messageDispatcher;

    public function __construct( LogBuilder $logBuilder )
    { /* ... */ }

    public function createMessageDispatcher()
    {
        if ( !isset( $this->messageDispatcher ) )
        {
            $this->messageDispatcher = new MessageDispatcher(
                array(
                    new JabberMessenger( /* ... */ ),
                    new MailMessenger(
                        new MailSmtpTransport( /* ... */ ),
                        $this->logBuilder->createLogger()
                    ),
                    $this->logBuilder->createLogger() );
        }
        return $this->messageDispatcher;
    }
}
```

Builders

```php
<?php

class MessageDispatcherBuilder
{
    protected $logBuilder;
    protected $messageDispatcher;

    public function __construct( LogBuilder $logBuilder )
    { /* ... */ }

    public function createMessageDispatcher()
    {
        if ( !isset( $this->messageDispatcher ) )
        {
            $this->messageDispatcher = new MessageDispatcher(
                array(
                    new JabberMessenger( /* ... */ ),
                    new MailMessenger(
                        new MailSmtpTransport( /* ... */ ),
                        $this->logBuilder->createLogger()
                    ),
                    $this->logBuilder->createLogger() );
        }
        return $this->messageDispatcher;
    }
}
```

Builders

```php
<?php

class MessageDispatcherBuilder
{
    protected $logBuilder;
    protected $messageDispatcher;

    public function __construct( LogBuilder $logBuilder )
    { /* ... */ }

    public function createMessageDispatcher()
    {
        if ( !isset( $this->messageDispatcher ) )
        {
            $this->messageDispatcher = new MessageDispatcher(
                array(
                    new JabberMessenger( /* ... */ ),
                    new MailMessenger(
                        new MailSmtpTransport( /* ... */ ),
                        $this->logBuilder->createLogger()
                    ),
                    $this->logBuilder->createLogger() );
        }
        return $this->messageDispatcher;
    }
}
```

Builders

```php
<?php

class MessageDispatcherBuilder
{
    protected $logBuilder;
    protected $messageDispatcher;

    public function __construct( LogBuilder $logBuilder )
    { /* ... */ }

    public function createMessageDispatcher()
    {
        if ( !isset( $this->messageDispatcher ) )
        {
            $this->messageDispatcher = new MessageDispatcher(
                array(
                    new JabberMessenger( /* ... */ ),
                    new MailMessenger(
                        new MailSmtpTransport( /* ... */ ),
                        $this->logBuilder->createLogger()
                    ),
                    $this->logBuilder->createLogger() );
        }
        return $this->messageDispatcher;
    }
}
```

Builders

```php
<?php

class MessageDispatcherBuilder
{
    protected $logBuilder;
    protected $messageDispatcher;

    public function __construct( LogBuilder $logBuilder )
    { /* ... */ }

    public function createMessageDispatcher()
    {
        if ( !isset( $this->messageDispatcher ) )
        {
            $this->messageDispatcher = new MessageDispatcher(
                array(
                    new JabberMessenger( /* ... */ ),
                    new MailMessenger(
                        new MailSmtpTransport( /* ... */ ),
                        $this->logBuilder->createLogger()
                    ) ),
                $this->logBuilder->createLogger() );
        }
        return $this->messageDispatcher;
    }
}
```

Builders

```php
<?php

class MessageDispatcherBuilder
{
    protected $logBuilder;
    protected $messageDispatcher;

    public function __construct( LogBuilder $logBuilder )
    { /* ... */ }

    public function createMessageDispatcher()
    {
        if ( !isset( $this->messageDispatcher ) )
        {
            $this->messageDispatcher = new MessageDispatcher(
                array(
                    new JabberMessenger( /* ... */ ),
                    new MailMessenger(
                        new MailSmtpTransport( /* ... */ ),
                        $this->logBuilder->createLogger()
                    ) ),
                $this->logBuilder->createLogger() );
        }
        return $this->messageDispatcher;
    }
}
```

Builders

```php
<?php

class MessageDispatcherBuilder
{
    protected $logBuilder;
    protected $messageDispatcher;

    public function __construct( LogBuilder $logBuilder )
    { /* ... */ }

    public function createMessageDispatcher()
    {
        if ( !isset( $this->messageDispatcher ) )
        {
            $this->messageDispatcher = new MessageDispatcher(
                array(
                    new JabberMessenger( /* ... */ ),
                    new MailMessenger(
                        new MailSmtpTransport( /* ... */ ),
                        $this->logBuilder->createLogger()
                    ),
                    $this->logBuilder->createLogger() );
        }
        return $this->messageDispatcher;
    }
}
```

Dependency Injection Container

- Takes care of object creation
- Manages dependencies
- Allows fine grained replacement

Manual DIC

```
1   <?php
2
3   class DependencyInjectionContainer
4   {
5       protected $cache;
10      public function getCache()
11      {
12          if ( !isset( $this->cache ) )
13          {
14              $this->cache = new FilesystemCache( /* ... */ );
15          }
16          return $this->cache;
17      }
42  }
```

Manual DIC

```php
<?php

class DependencyInjectionContainer
{

    protected $messenger;

    public function getMessenger()
    {
        if ( !isset( $this->messenger ) )
        {
            $this->messenger = new Messenger( array(
                'email' => $this->getMailMessenger(),
            ) );
        }
        return $this->messenger;
    }

}
```

Manual DIC

```php
<?php

class DependencyInjectionContainer
{

    protected $mailMessenger;

    public function getMailMessenger()
    {
        if ( !isset( $this->mailMessenger ) )
        {
            $this->mailMessenger = new MailMessenger(
                $this->getMailTransport()
            );
        }
        return $this->mailMessenger;
    }

    /* ... */

}
```

The Arbit DIC

- Exemplary
  - Shared objects (initialized once)
  - Closures for lazy initialization
  - Inherent dependency resolution

The Arbit DIC

- Exemplary
  - Shared objects (initialized once)
  - Closures for lazy initialization
  - Inherent dependency resolution
- Base class for custom DICs

The Arbit DIC

- Exemplary
  - Shared objects (initialized once)
  - Closures for lazy initialization
  - Inherent dependency resolution
- Base class for custom DICs
- Heavy use of interceptors

The Arbit DIC

- Exemplary
  - Shared objects (initialized once)
  - Closures for lazy initialization
  - Inherent dependency resolution
- Base class for custom DICs
- Heavy use of interceptors
- No implementation details,
  see here: `http://bit.ly/arbitDIC`

The Arbit DIC

```
3   class arbitEnvironmentDIC extends arbitDependencyInjectionContainer
4   {


13      public function initialize()
14      {


38      }
39  }
```

The Arbit DIC

```
3   class arbitEnvironmentDIC extends arbitDependencyInjectionContainer
4   {


13      public function initialize()
14      {


38      }
39  }
```

The Arbit DIC

```
3   class arbitEnvironmentDIC extends arbitDependencyInjectionContainer
4   {


13      public function initialize()
14      {


15          $this->cache = function( $dic )
16          {
17              return new arbitFilesystemCache( $dic->request->controller );
18          };


38      }
39  }
```

Resolving the Cache

```php
<?php

$dic = new arbitEnvironmentDIC();
$cache = $dic->cache;

// Inside $dic this is similar to:

if ( !isset( $this->shared['cache'] ) )
{
    $this->shared['cache'] = $this->cache( $this );
}

// Triggering the closure in $this->cache:

function cacheClosure( $dic )
{
    return new arbitFilesystemCache( $dic->request->controller );
}
```

Resolving the Cache

```php
<?php

$dic = new arbitEnvironmentDIC();
$cache = $dic->cache;

// Inside $dic this is similar to:

if ( !isset( $this->shared['cache'] ) )
{
    $this->shared['cache'] = $this->cache( $this );
}

// Triggering the closure in $this->cache:

function cacheClosure( $dic )
{
    return new arbitFilesystemCache( $dic->request->controller );
}
```

Resolving the Cache

```php
<?php

$dic = new arbitEnvironmentDIC();
$cache = $dic->cache;

// Inside $dic this is similar to:

if ( !isset( $this->shared['cache'] ) )
{
    $this->shared['cache'] = $this->cache( $this );
}

// Triggering the closure in $this->cache:

function cacheClosure( $dic )
{
    return new arbitFilesystemCache( $dic->request->controller );
}
```

Resolving the Cache

```php
<?php

$dic = new arbitEnvironmentDIC();
$cache = $dic->cache;

// Inside $dic this is similar to:

if ( !isset( $this->shared['cache'] ) )
{
    $this->shared['cache'] = $this->cache( $this );
}

// Triggering the closure in $this->cache:

function cacheClosure( $dic )
{
    return new arbitFilesystemCache( $dic->request->controller );
}
```

Resolving the Cache

```php
<?php

$dic = new arbitEnvironmentDIC();
$cache = $dic->cache;

// Inside $dic this is similar to:

if ( !isset( $this->shared['cache'] ) )
{
    $this->shared['cache'] = $this->cache( $this );
}

// Triggering the closure in $this->cache:

function cacheClosure( $dic )
{
    return new arbitFilesystemCache( $dic->request->controller );
}
```

The Arbit DIC

```
3   class arbitEnvironmentDIC extends arbitDependencyInjectionContainer
4   {


13      public function initialize()
14      {


20          $this->messenger = function( $dic )
21          {
22              return new arbitMessenger( array(
23                  'email' => $dic->mailMessenger,
24              ) );
25          };


38      }
39  }
```

The Arbit DIC

```
3   class arbitEnvironmentDIC extends arbitDependencyInjectionContainer
4   {


13      public function initialize()
14      {


27          $this->mailMessenger = function( $dic )
28          {
29              return new arbitMailMessenger(
30                  $dic->mailTransport,
31                  $dic->configuration->main,
32                  $dic->configuration->project( $dic->request->controller ),
33                  $dic->views->decorator
34              );
35          };


38      }
39  }
```

Dependency Injection Approaches

- Constructor injection
  - We just saw that

Dependency Injection Approaches

- Constructor injection
  - We just saw that
- Setter injection
  - Use `set*()`
  - Default instances?

Dependency Injection Approaches

- Constructor injection
  - We just saw that
- Setter injection
  - Use `set*()`
  - Default instances?
- Interface injection
  - Inject by interface
  - Specify instance for interface

More about DIC

- Phemto - A dependency injector for PHP 5
  `http://phemto.sourceforge.net/`
- Bucket - Basic di-container for php
  `https://github.com/troelskn/bucket`
- Symfony Dependency Injection
  `http://components.symfony-project.org/dependency-injection/`
- Pimple - A small PHP 5.3 DIC (used in Silex)
  `https://github.com/fabpot/Pimple`

More about DIC

- Phemto - A dependency injector for PHP 5
  `http://phemto.sourceforge.net/`
- Bucket - Basic di-container for php
  `https://github.com/troelskn/bucket`
- Symfony Dependency Injection
  `http://components.symfony-project.org/dependency-injection/`
- Pimple - A small PHP 5.3 DIC (used in Silex)
  `https://github.com/fabpot/Pimple`
- **Martin Fowler on Dependency Injection**
  `http://martinfowler.com/articles/injection.html`

Outline

Motivation

- Central Registry
- Clients ask for service (abstraction)

Motivation

- Central Registry
- Clients ask for service (abstraction)
- Knows how to ”assemble” services
- Orginates from compiled languages

Motivation

- Central Registry
- Clients ask for service (abstraction)
- Knows how to "assemble" services
- Orginates from compiled languages
- Sometimes used similar to DI
- Often used to reduce parameter lists

Service locator example

```php
<?php

class ServiceLocator
{
    public function getUserGateway()
    { /* ... */ }

    public function getProductGateway()
    { /* ... */ }

    // ...
}

class UserController
{
    public function __construct( ServiceLocator $serviceLocator )
    { /* ... */ }
}

?>
```

Service locator example

```php
<?php

class ServiceLocator
{
    public function getUserGateway()
    { /* ... */ }

    public function getProductGateway()
    { /* ... */ }

    // ...
}

class UserController
{
    public function __construct( ServiceLocator $serviceLocator )
    { /* ... */ }
}

?>
```

Service locator example

```php
<?php

class ServiceLocator
{
    public function getUserGateway()
    { /* ... */ }

    public function getProductGateway()
    { /* ... */ }

    // ...
}

class UserController
{
    public function __construct( ServiceLocator $serviceLocator )
    { /* ... */ }
}

?>
```

Service locator example

```php
<?php

class ServiceLocator
{
    public function getUserGateway()
    { /* ... */ }

    public function getProductGateway()
    { /* ... */ }

    // ...
}

class UserController
{
    public function __construct( ServiceLocator $serviceLocator )
    { /* ... */ }
}

?>
```

Service locator example

```php
<?php

class ServiceLocator
{
    public function getUserGateway()
    { /* ... */ }

    public function getProductGateway()
    { /* ... */ }

    // ...
}

class UserController
{
    public function __construct( ServiceLocator $serviceLocator )
    { /* ... */ }
}

?>
```

Problems

- Hides dependencies
- Hides code smells
- Enables usage of arbitrary objects

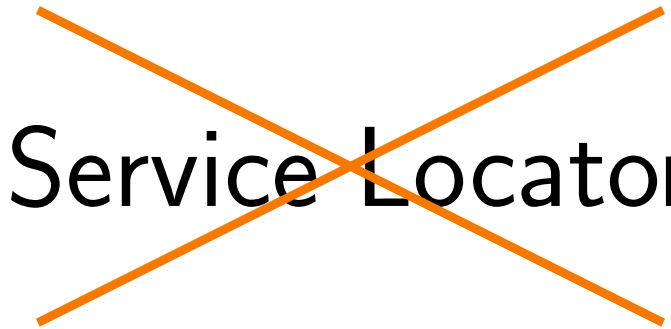
Universal Constructors

- Ctor expects only single parameter
- An array with everything the object needs
- Potentially merged with default values

Universal Constructors Example

```
1  <?php
2
3  class UserController
4  {
5      public function __construct( $config = null )
6      { /* ... */ }
7  }
```

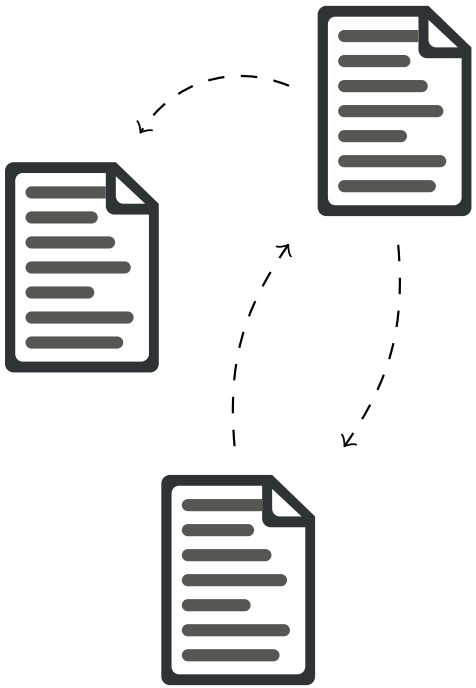
The essence



Outline

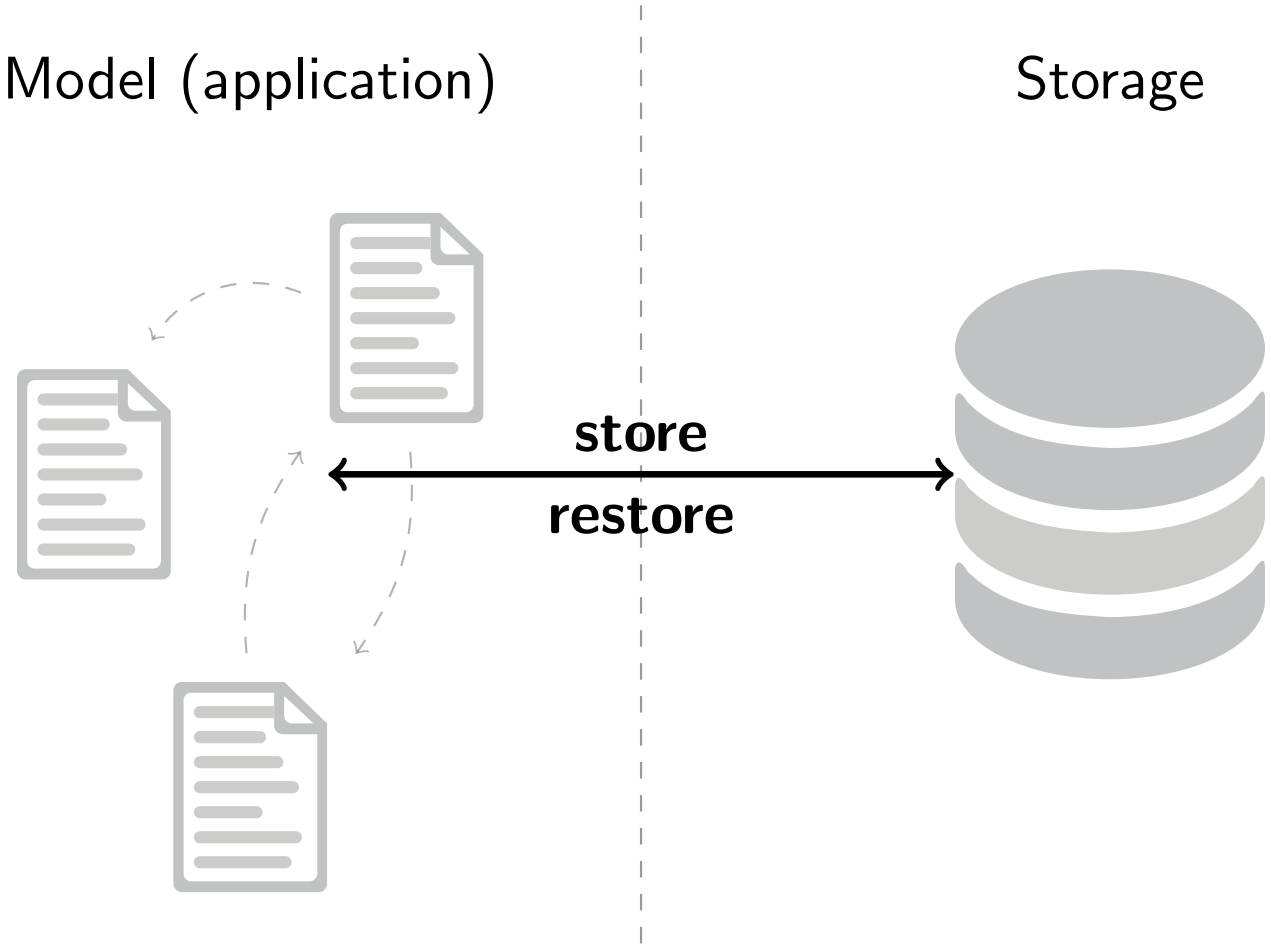
The situation

Model (application)



Storage

The situation



Challenges

- Model and storage structure differ

Challenges

- Model and storage structure differ
  - Object relational impedance mismatch

Challenges

- Model and storage structure differ
  - Object relational impedance mismatch
- Different access approaches

Challenges

- Model and storage structure differ
  - Object relational impedance mismatch
- Different access approaches
  - Query language differences

Challenges

- Model and storage structure differ
  - Object relational impedance mismatch
- Different access approaches
  - Query language differences
- Storage back end might change

Challenges

- Model and storage structure differ
  - Object relational impedance mismatch
- Different access approaches
  - Query language differences
- Storage back end might change
- Back ends could even be mixed

Challenges

- Model and storage structure differ
  - Object relational impedance mismatch
- Different access approaches
  - Query language differences
- Storage back end might change
- Back ends could even be mixed
- ...

Active Record

```php
<?php

class Invoice extends ActiveRecord
{
    protected $id;
    protected $positions;
    protected $vat;
    // ...

    public function calculateValue()
    { /* business logic */ }
}

class ActiveRecord
{
    public function insert()
    { /* ... */ }
    public function update()
    { /* ... */ }
}
```

Active Record

```php
<?php

class Invoice extends ActiveRecord
{
    protected $id;
    protected $positions;
    protected $vat;
    // ...

    public function calculateValue()
    { /* business logic */ }
}

class ActiveRecord
{
    public function insert()
    {  /* ... */  }
    public function update()
    { /* ... */ }
}
```

Active Record

```php
<?php

class Invoice extends ActiveRecord
{
    protected $id;
    protected $positions;
    protected $vat;
    // ...

    public function calculateValue()
    { /* business logic */ }
}

class ActiveRecord
{
    public function insert()
    { /* ... */ }
    public function update()
    { /* ... */ }
}
```

Active Record

```php
<?php

class Invoice extends ActiveRecord
{
    protected $id;
    protected $positions;
    protected $vat;
    // ...

    public function calculateValue()
    { /* business logic */ }
}

class ActiveRecord
{
    public function insert()
    {  /* ... */  }
    public function update()
    { /* ... */ }
}
```

Active Record

- Combines storage and business logic
- Commonly storage logic in base class
- Model class corresponds to database record

Evaluation

Pros

- Very easy to use
- Few code to write

Cons

Evaluation

Pros

- Very easy to use
- Few code to write

Cons

Evaluation

Pros

- Very easy to use
- Few code to write

Cons

- Model structure = storage structure

Evaluation

Pros

- Very easy to use
- Few code to write

Cons

- Model structure = storage structure
- Classes become complex
  - Business logic
  - Storage logic

Evaluation

Pros

- Very easy to use
- Few code to write

Cons

- Model structure = storage structure
- Classes become complex
  - Business logic
  - Storage logic
- Broken object semantics

Evaluation

Pros

- Very easy to use
- Few code to write

Cons

- Model structure = storage structure
- Classes become complex
  - Business logic
  - Storage logic
- Broken object semantics
- Changes ripple over

Evaluation

Pros

- Very easy to use
- Few code to write

Cons

- Model structure = storage structure
- Classes become complex
  - Business logic
  - Storage logic
- Broken object semantics
- Changes ripple over
- Hard to exchange storage logic

Evaluation

Pros

- Very easy to use
- Few code to write

Cons

- Model structure = storage structure
- Classes become complex
  - Business logic
  - Storage logic
- Broken object semantics
- Changes ripple over
- Hard to exchange storage logic
- Really hard to test!

Lesson learned . . .

De-couple business and storage logic!

Other Approaches

- Table data gateway
- Row data gateway

Data Mapper

```php
<?php

class Invoice
{
    protected $positions;
    protected $vat;
    // ...

    public function calculateValue()
    { /* business logic */ }
}

interface InvoiceMapper
{
    public function store( Invoice $invoice );
    public function update( Invoice $invoice );
    public function findByCustomer( Customer $customer );
}

class DbInvoiceMapper implements InvoiceMapper
{
    // ...
}

?>
```

Data Mapper

```php
<?php

class Invoice
{
    protected $positions;
    protected $vat;
    // ...

    public function calculateValue()
    { /* business logic */ }
}

interface InvoiceMapper
{
    public function store( Invoice $invoice );
    public function update( Invoice $invoice );
    public function findByCustomer( Customer $customer );
}

class DbInvoiceMapper implements InvoiceMapper
{
    // ...
}

?>
```

Data Mapper

```php
<?php

class Invoice
{
    protected $positions;
    protected $vat;
    // ...

    public function calculateValue()
    { /* business logic */ }
}

interface InvoiceMapper
{
    public function store( Invoice $invoice );
    public function update( Invoice $invoice );
    public function findByCustomer( Customer $customer );
}

class DbInvoiceMapper implements InvoiceMapper
{
    // ...
}

?>
```

Data Mapper

```php
<?php

class Invoice
{
    protected $positions;
    protected $vat;
    // ...

    public function calculateValue()
    { /* business logic */ }
}

interface InvoiceMapper
{
    public function store( Invoice $invoice );
    public function update( Invoice $invoice );
    public function findByCustomer( Customer $customer );
}

class DbInvoiceMapper implements InvoiceMapper
{
    // ...
}

?>
```

Data Mapper

- Decouple storage from model
- No OO modelling of DB structure
- Model does not even know a database exists

Evaluation

Pros

- Complete decoupling

Cons

Evaluation

Pros

- Complete decoupling
- Model is not aware of storage

Cons

Evaluation

Pros

- Complete decoupling
- Model is not aware of storage
- Clean storage interface

Cons

Evaluation

Pros

- Complete decoupling
- Model is not aware of storage
- Clean storage interface
  - Implement different storages!

Cons

Evaluation

Pros

- Complete decoupling
- Model is not aware of storage
- Clean storage interface
  - Implement different storages!
- DB changes only affect mapping layer
- Model changes only affect mapping layer

Cons

Evaluation

Pros

- Complete decoupling
- Model is not aware of storage
- Clean storage interface
  - Implement different storages!
- DB changes only affect mapping layer
- Model changes only affect mapping layer
- Nice for testing

Cons

Evaluation

Pros

- Complete decoupling
- Model is not aware of storage
- Clean storage interface
  - Implement different storages!
- DB changes only affect mapping layer
- Model changes only affect mapping layer
- Nice for testing

Cons

- Quite some code to write

Evaluation

Pros

- Complete decoupling
- Model is not aware of storage
- Clean storage interface
  - Implement different storages!
- DB changes only affect mapping layer
- Model changes only affect mapping layer
- Nice for testing

Cons

- Quite some code to write
- Mapping can become complex

Attribution

Large parts of this talk are inspired by

Patterns of Enterprise Application Architecture

by Martin Fowler

ISBN 978-0321127426 — http://amzn.to/PofEAA

Highy recommended!

Conclusion

- Patterns are not the holy grail!

Conclusion

- Patterns are not the holy grail!
- They assign names to good ideas
- They help you to talk about concepts
- They can inspire you

Q / A

Are there any questions left?

Thanks for listening

Slides will be online at
`http://talks.qafoo.com`

Thanks for listening

Slides will be online at
`http://talks.qafoo.com`

Stay in touch

- Tobias Schlitt
- toby@qafoo.com
- @tobySen / @qafoo

Rent a PHP quality expert:
`http://qafoo.com`