PHP Design Patterns

Tristan Lins & Christian Schiffler

Contao Nordtag 2014

8. Feb 2014

C-C-A Power - „TheTril“ & „Xtra“

Wers'n das?

Tristan „TheTril“ Lins

- Seit 2010 selbständig
- Seit 2009 bei ~~TYPOlight~~ Contao
- Hauptentwickler Avisota.

t.lins@c-c-a.org
http://bit3.de
http://twitter.com/tristanlins
http://github.com/tristanlins

Christian „Xtra“ Schiffler

- Seit 2004 selbständig
- Seit 2008 bei ~~TYPOlight~~ Contao
- Hauptentwickler MetaModels

c.schiffler@c-c-a.org
http://cyberspectrum.de
http://twitter.com/discordier
http://github.com/discordier

Zusammen

- DCGeneral
- Composer Integration
- CCA Komponenten

Part 1

Part 1

- Fluent Interfaces
- Hooks vs. Events
- Dependency Injection (DI)

Part 1

Part 1

- Fluent Interfaces
- Hooks vs. Events
- Dependency Injection (DI)

Part 1

Part 1

- Fluent Interfaces
- Hooks vs. Events
- Dependency Injection (DI)

Warum das ganze?

Decoupling!

Was sind Design Pattern?

Was sind Design Pattern

Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software, ISBN 3-8273-2199-9, 1994 Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides.

- Originaltitel: Design Patterns. Elements of Reusable Object-Oriented Software.
- Autoren oft als Viererbande (en. Gang of Four, GoF) bezeichnet.
- Oft nur als „GoF book“ abgekürzt.

http://de.wikipedia.org/wiki/Entwurfsmuster

„Der erfolgreiche Einsatz von Entwurfsmustern in der Vergangenheit kann dazu verleiten, die Entwurfsmuster als Wunderwaffe und Garant für gutes Design anzusehen. Unerfahrene Entwickler können geneigt sein, möglichst viele bekannte Muster zu verwenden, und dabei übersehen, dass in ihrem Fall vielleicht eine elegantere Lösung ohne den Einsatz von Mustern möglich wäre. Entwurfsmuster garantieren nicht, dass der Entwurf gut ist.“

Was sind Design Pattern

Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software, ISBN 3-8273-2199-9, 1994 Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides.

- Originaltitel: Design Patterns. Elements of Reusable Object-Oriented Software.
- Autoren oft als Viererbande (en. Gang of Four, GoF) bezeichnet.
- Oft nur als „GoF book“ abgekürzt.

http://de.wikipedia.org/wiki/Entwurfsmuster

„Der erfolgreiche Einsatz von Entwurfsmustern in der Vergangenheit kann dazu verleiten, die Entwurfsmuster als Wunderwaffe und Garant für gutes Design anzusehen. **Unerfahrene Entwickler können** *geneigt sein,* **möglichst viele bekannte Muster** *zu verwenden, und dabei übersehen, dass in ihrem Fall vielleicht eine elegantere Lösung ohne den Einsatz von Mustern möglich wäre. Entwurfsmuster* **garantieren nicht, dass der Entwurf gut ist.**“

Was sind design pattern

Bekannte Design pattern Kategorien

- algorithm strategy patterns.
- computational design patterns.
- execution patterns.
- implementation strategy patterns.
- structural design patterns

Was sind design pattern

Bekannte Design pattern Kategorien

- algorithm strategy patterns.
- computational design patterns.
- execution patterns.
- implementation strategy patterns.
- structural design patterns

Was sind design pattern

Bekannte Design pattern Kategorien

- algorithm strategy patterns.
- computational design patterns.
- execution patterns.
- implementation strategy patterns.
- structural design patterns

Was sind design pattern

Bekannte Design pattern Kategorien

- algorithm strategy patterns.
- computational design patterns.
- execution patterns.
- implementation strategy patterns.
- structural design patterns

Was sind design pattern

Bekannte Design pattern Kategorien

- algorithm strategy patterns.
- computational design patterns.
- execution patterns.
- implementation strategy patterns.
- structural design patterns

Was sind design pattern

Bekannte Design pattern Kategorien

- algorithm strategy patterns.
- computational design patterns.
- execution patterns.
- implementation strategy patterns.
- structural design patterns

Fluent Interface

Fluent Interface

Bekannte Design pattern Kategorien

- Als Begründer des Konzepts gelten Eric Evans und Martin Fowler.

Fluent Interface

Codebeispiel - Non fluent


```php
class Person
{
    /**
     * Sets the name.
     * @param string $name The name.
     * @return void
     */
    public function setName($name)
    {
        $this->name = $name;
    }

    /**
     * Sets the age.
     * @param string $age The age.
     * @return void
     */
    public function setAge($age)
    {
        $this->age = $age;
    }

    /**
     * Get the introduction.
     * @return string
     */
    public function getIntroduction()
    {
        return sprintf(
            'Hello, my name is %s and I am %d years old.',
            $this->name,
            $this->age
        );
    }
}
```

Fluent Interface

Codebeispiel - Now fluent


```php
class Person
{
    /**
     * [...]
     * @return Person
     */
    public function setName($name)
    {
        $this->name = $name;
        return $this;
    }

    /**
     * [...]
     * @return Person
     */
    public function setAge($age)
    {
        $this->age = $age;
        return $this;
    }

    /**
     * Get the introduction.
     * @return string
     */
    public function getIntroduction()
    {
        return sprintf(
            'Hello, my name is %s and I am %d years old.',
            $this->name,
            $this->age
        );
    }
}
```

Fluent Interface
Codebeispiel - Verwendung

Non fluent

```php
$person = new Person();
$person->setName('Zhapod Beeblebrox');
$person->setAge(42);
$introduction = $person->getIntroduction();
```

Fluent

```php
$person = new Person();
$introduction = $person
    ->setName('Guybrush Threepwood')
    ->setAge(16)
    ->getIntroduction();
```

Fluent Interface
Ein Beispiel aus der Praxis

Doctrine Mongo DB example

```
1  $qb = $dm->createQueryBuilder('Person')
2      ->select('username', 'age')
3      ->sort('createdAt', 'desc')
4      ->sort('featured', 'desc')
5      ->field('hometown')
6          ->equals('South Park')
7      ->limit(20)
8      ->skip(40);
9
10 $query = $qb->getQuery();
11 $persons = $query->execute();
```

Fluent Interface
Vor- und Nachteile

Vorteile

- Leichtere Entwicklung und bessere Lesbarkeit des Codes.
- Kommt natürlicher Sprache sehr nahe. Wenig kommentieren.
- Klare Vorstellungen über Funktionalitäten und Gebrauch.
- IDE autocompletion zeigt weitere Möglichkeiten.

Fluent Interface
Vor- und Nachteile

Vorteile

- Leichtere Entwicklung und bessere Lesbarkeit des Codes.
- Kommt natürlicher Sprache sehr nahe. Wenig kommentieren.
- Klare Vorstellungen über Funktionalitäten und Gebrauch.
- IDE autocompletion zeigt weitere Möglichkeiten.

Fluent Interface
Vor- und Nachteile

Vorteile

- Leichtere Entwicklung und bessere Lesbarkeit des Codes.
- Kommt natürlicher Sprache sehr nahe. Wenig kommentieren.
- Klare Vorstellungen über Funktionalitäten und Gebrauch.
- IDE autocompletion zeigt weitere Möglichkeiten.

Fluent Interface
Vor- und Nachteile

Vorteile

- Leichtere Entwicklung und bessere Lesbarkeit des Codes.
- Kommt natürlicher Sprache sehr nahe. Wenig kommentieren.
- Klare Vorstellungen über Funktionalitäten und Gebrauch.
- IDE autocompletion zeigt weitere Möglichkeiten.

Fluent Interface
Vor- und Nachteile

Vorteile

- Leichtere Entwicklung und bessere Lesbarkeit des Codes.
- Kommt natürlicher Sprache sehr nahe. Wenig kommentieren.
- Klare Vorstellungen über Funktionalitäten und Gebrauch.
- IDE autocompletion zeigt weitere Möglichkeiten.

Nachteile

- Aufwand für das Fluid Interface selbst.
- Realisierung einer Grammatik ist sehr aufwendig.
- Lange Kette von Aufrufen erschwert Debugging.

Fluent Interface
Vor- und Nachteile

Vorteile

- Leichtere Entwicklung und bessere Lesbarkeit des Codes.
- Kommt natürlicher Sprache sehr nahe. Wenig kommentieren.
- Klare Vorstellungen über Funktionalitäten und Gebrauch.
- IDE autocompletion zeigt weitere Möglichkeiten.

Nachteile

- Aufwand für das Fluid Interface selbst.
- Realisierung einer Grammatik ist sehr aufwendig.
- Lange Kette von Aufrufen erschwert Debugging.

Fluent Interface
Vor- und Nachteile

Vorteile

- Leichtere Entwicklung und bessere Lesbarkeit des Codes.
- Kommt natürlicher Sprache sehr nahe. Wenig kommentieren.
- Klare Vorstellungen über Funktionalitäten und Gebrauch.
- IDE autocompletion zeigt weitere Möglichkeiten.

Nachteile

- Aufwand für das Fluid Interface selbst.
- Realisierung einer Grammatik ist sehr aufwendig.
- Lange Kette von Aufrufen erschwert Debugging.

Hooks vs. Events

Hooks vs. Events
Was sind Hooks?

http://de.wikipedia.org/wiki/Hook_(Informatik)

„In der Programmierung bezeichnet der Begriff Hook (auch Einschubmethode genannt) eine Schnittstelle, mit der fremder Programmcode in eine bestehende Anwendung integriert werden kann, um diese zu erweitern, deren Ablauf zu modifizieren oder um bestimmte Ereignisse abzufangen.“

Zusammenfassung

- Hooks dienen dazu, auf Ereignisse zu reagieren.
- Hooks dienen dazu, das Verhalten der Software zu verändern.

Hooks vs. Events
Was sind Hooks?

http://de.wikipedia.org/wiki/Hook_(Informatik)

„In der Programmierung bezeichnet der Begriff Hook (auch Einschubmethode genannt) eine Schnittstelle, mit der fremder Programmcode in eine bestehende Anwendung integriert werden kann, um diese zu erweitern, deren Ablauf zu modifizieren oder um bestimmte Ereignisse abzufangen.“

Zusammenfassung

- Hooks dienen dazu, auf Ereignisse zu reagieren.
- Hooks dienen dazu, das Verhalten der Software zu verändern.

Hooks vs. Events
Was sind Hooks?

http://de.wikipedia.org/wiki/Hook_(Informatik)

„In der Programmierung bezeichnet der Begriff Hook (auch Einschubmethode genannt) eine Schnittstelle, mit der fremder Programmcode in eine bestehende Anwendung integriert werden kann, um diese zu erweitern, deren Ablauf zu modifizieren oder um bestimmte Ereignisse abzufangen.“

Zusammenfassung

- Hooks dienen dazu, auf Ereignisse zu reagieren.
- Hooks dienen dazu, das Verhalten der Software zu verändern.

Hooks vs. Events
Was sind Events?

http://de.wikipedia.org/wiki/Ereignis_(Programmierung)

„Das Programm wird nicht linear durchlaufen, sondern es werden spezielle Ereignisbehandlungsroutinen immer dann ausgeführt, wenn ein bestimmtes Ereignis auftritt. Ereignisorientierte Programmierung gehört zu den parallelen Programmiertechniken, hat also deren Vor- und Nachteile.“

Zusammenfassung

- Events dienen dazu, auf Ereignisse zu reagieren.
- Events dienen dazu, das Verhalten der Software zu verändern.

Hooks vs. Events
Was sind Events?

http://de.wikipedia.org/wiki/Ereignis_(Programmierung)

„Das Programm wird nicht linear durchlaufen, sondern es werden spezielle Ereignisbehandlungsroutinen immer dann ausgeführt, wenn ein bestimmtes Ereignis auftritt. Ereignisorientierte Programmierung gehört zu den parallelen Programmiertechniken, hat also deren Vor- und Nachteile.“

Zusammenfassung

- Events dienen dazu, auf Ereignisse zu reagieren.
- Events dienen dazu, das Verhalten der Software zu verändern.

Hooks vs. Events
Was sind Events?

http://de.wikipedia.org/wiki/Ereignis_(Programmierung)

„Das Programm wird nicht linear durchlaufen, sondern es werden spezielle Ereignisbehandlungsroutinen immer dann ausgeführt, wenn ein bestimmtes Ereignis auftritt. Ereignisorientierte Programmierung gehört zu den parallelen Programmiertechniken, hat also deren Vor- und Nachteile.“

Zusammenfassung

- Events dienen dazu, auf Ereignisse zu reagieren.
- Events dienen dazu, das Verhalten der Software zu verändern.

Hooks vs. Events
Gegenüberstellung

Hooks

- Hooks dienen dazu, auf Ereignisse zu reagieren.
- Hooks dienen dazu, das Verhalten der Software zu verändern.

Events

- Events dienen dazu, auf Ereignisse zu reagieren.
- Events dienen dazu, das Verhalten der Software zu verändern.

Hooks vs. Events
Zusammenfassung

Zusammenfassung

- Hooks sind nahe Verwandte von Events und werden oft auch als *Untergruppe* von Events betrachtet.
- Hooks sind speziell dafür gedacht, den Programmablauf zu **verändern**.
- Events **bilden** den Programmablauf und **verändern** diesen.
- Hooks und Events unterscheiden sich in den meisten Fällen lediglich in der Implementierung.
- Hooks werden nicht selten über Events realisiert.
- Es gibt noch Signale, Actions und Commands, welche lediglich andere/speziellere Formen/Begriffe von/für Events sind.

Hooks vs. Events
Zusammenfassung

Zusammenfassung

- Hooks sind nahe Verwandte von Events und werden oft auch als *Untergruppe* von Events betrachtet.
- Hooks sind speziell dafür gedacht, den Programmablauf zu **verändern**.
- Events **bilden** den Programmablauf und **verändern** diesen.
- Hooks und Events unterscheiden sich in den meisten Fällen lediglich in der Implementierung.
- Hooks werden nicht selten über Events realisiert.
- Es gibt noch Signale, Actions und Commands, welche lediglich andere/speziellere Formen/Begriffe von/für Events sind.

Zusammenfassung

- Hooks sind nahe Verwandte von Events und werden oft auch als *Untergruppe* von Events betrachtet.
- Hooks sind speziell dafür gedacht, den Programmablauf zu **verändern**.
- Events **bilden** den Programmablauf und **verändern** diesen.
- Hooks und Events unterscheiden sich in den meisten Fällen lediglich in der Implementierung.
- Hooks werden nicht selten über Events realisiert.
- Es gibt noch Signale, Actions und Commands, welche lediglich andere/speziellere Formen/Begriffe von/für Events sind.

Hooks vs. Events
Zusammenfassung

Zusammenfassung

- Hooks sind nahe Verwandte von Events und werden oft auch als *Untergruppe* von Events betrachtet.
- Hooks sind speziell dafür gedacht, den Programmablauf zu **verändern**.
- Events **bilden** den Programmablauf und **verändern** diesen.
- Hooks und Events unterscheiden sich in den meisten Fällen lediglich in der Implementierung.
- Hooks werden nicht selten über Events realisiert.
- Es gibt noch Signale, Actions und Commands, welche lediglich andere/speziellere Formen/Begriffe von/für Events sind.

Hooks vs. Events
Zusammenfassung

Zusammenfassung

- Hooks sind nahe Verwandte von Events und werden oft auch als *Untergruppe* von Events betrachtet.
- Hooks sind speziell dafür gedacht, den Programmablauf zu **verändern**.
- Events **bilden** den Programmablauf und **verändern** diesen.
- Hooks und Events unterscheiden sich in den meisten Fällen lediglich in der Implementierung.
- Hooks werden nicht selten über Events realisiert.
- Es gibt noch Signale, Actions und Commands, welche lediglich andere/speziellere Formen/Begriffe von/für Events sind.

Hooks vs. Events
Zusammenfassung

Zusammenfassung

- Hooks sind nahe Verwandte von Events und werden oft auch als *Untergruppe* von Events betrachtet.
- Hooks sind speziell dafür gedacht, den Programmablauf zu **verändern**.
- Events **bilden** den Programmablauf und **verändern** diesen.
- Hooks und Events unterscheiden sich in den meisten Fällen lediglich in der Implementierung.
- Hooks werden nicht selten über Events realisiert.
- Es gibt noch Signale, Actions und Commands, welche lediglich andere/speziellere Formen/Begriffe von/für Events sind.

Hooks vs. Events
Bekannte Vertreter

Bekannte Vertreter

- Java bringt ein ausgereiftes Event Dispatcher / Event Listener Framework mit (und zählt zu den Programmiersprachen, die Events am stärksten *ausleben*).
- JavaScript und node.js.
- Jede grafische Benutzeroberfläche.
- Computerhardware (hier in Form von Signalen/Interrupts).

Hooks vs. Events
Bekannte Vertreter

Bekannte Vertreter

- Java bringt ein ausgereiftes Event Dispatcher / Event Listener Framework mit (und zählt zu den Programmiersprachen, die Events am stärksten *ausleben*).
- JavaScript und node.js.
- Jede grafische Benutzeroberfläche.
- Computerhardware (hier in Form von Signalen/Interrupts).

Hooks vs. Events
Bekannte Vertreter

Bekannte Vertreter

- Java bringt ein ausgereiftes Event Dispatcher / Event Listener Framework mit (und zählt zu den Programmiersprachen, die Events am stärksten *ausleben*).
- JavaScript und node.js.
- Jede grafische Benutzeroberfläche.
- Computerhardware (hier in Form von Signalen/Interrupts).

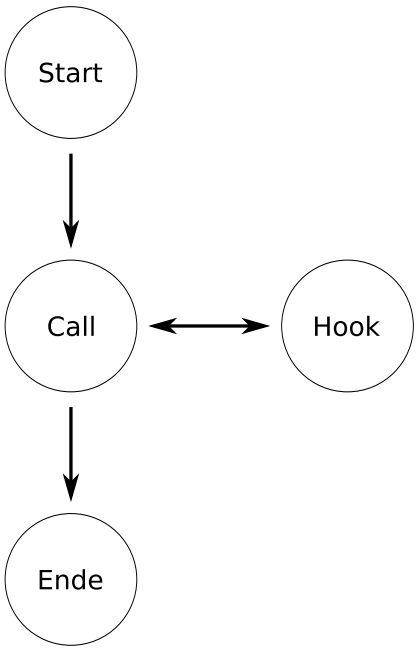
Hooks vs. Events
Bekannte Vertreter

Bekannte Vertreter

- Java bringt ein ausgereiftes Event Dispatcher / Event Listener Framework mit (und zählt zu den Programmiersprachen, die Events am stärksten *ausleben*).
- JavaScript und node.js.
- Jede grafische Benutzeroberfläche.
- Computerhardware (hier in Form von Signalen/Interrupts).

Programmablauf mit Hooks und Events

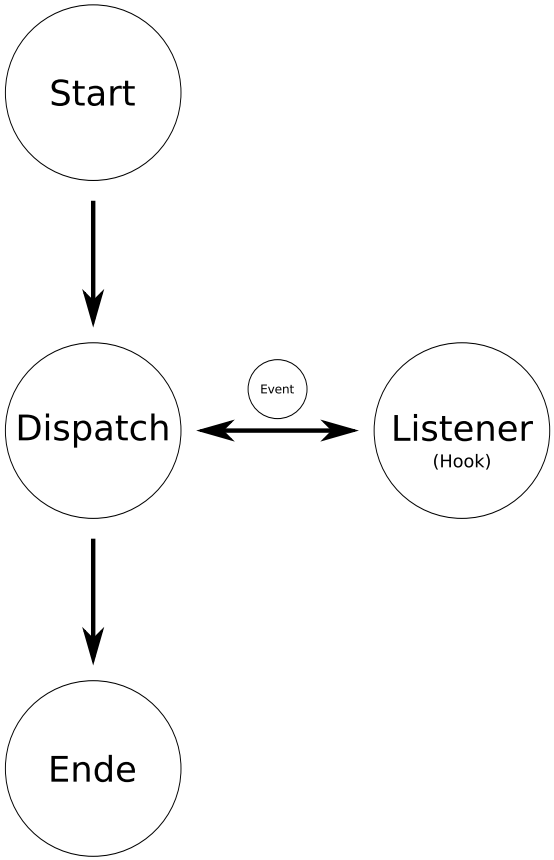
Hooks vs. Events
Programmablauf mit Hooks




```php
$message = 'Hello Mr. X';

if (isset($GLOBALS['TL_HOOKS']['manipulate']) &&
    is_array($GLOBALS['TL_HOOKS']['manipulate']))
  {
  foreach ($GLOBALS['TL_HOOKS']['manipulate'] as $callback)
  {
    $message = static::importStatic($callback[0])
      ->$callback[1]($message);
  }
}

echo $message;
```

Hooks vs. Events

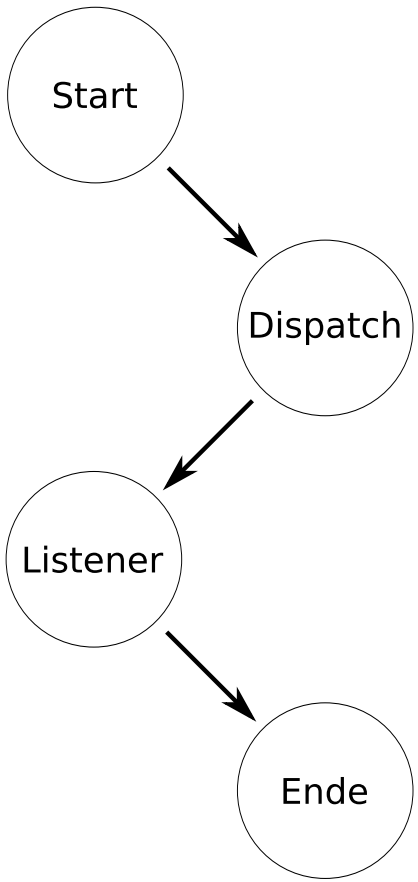
Programmablauf mit Events als Hooks




```php
$message = 'Hello Mr. X';

$event = new ManipulateMessageEvent();
$event->setMessage($message);

$eventDispatcher->dispatch(
  'manipulate',
  $event
);

$message = $event->getMessage();

echo $message;



class ManipulateMessageEvent
{
  protected $message;

  public function setMessage($message)
  {
    $this->message = $message;
    return $this;
  }

  public function getMessage()
  {
    return $this->message;
  }
}
```

Hooks vs. Events

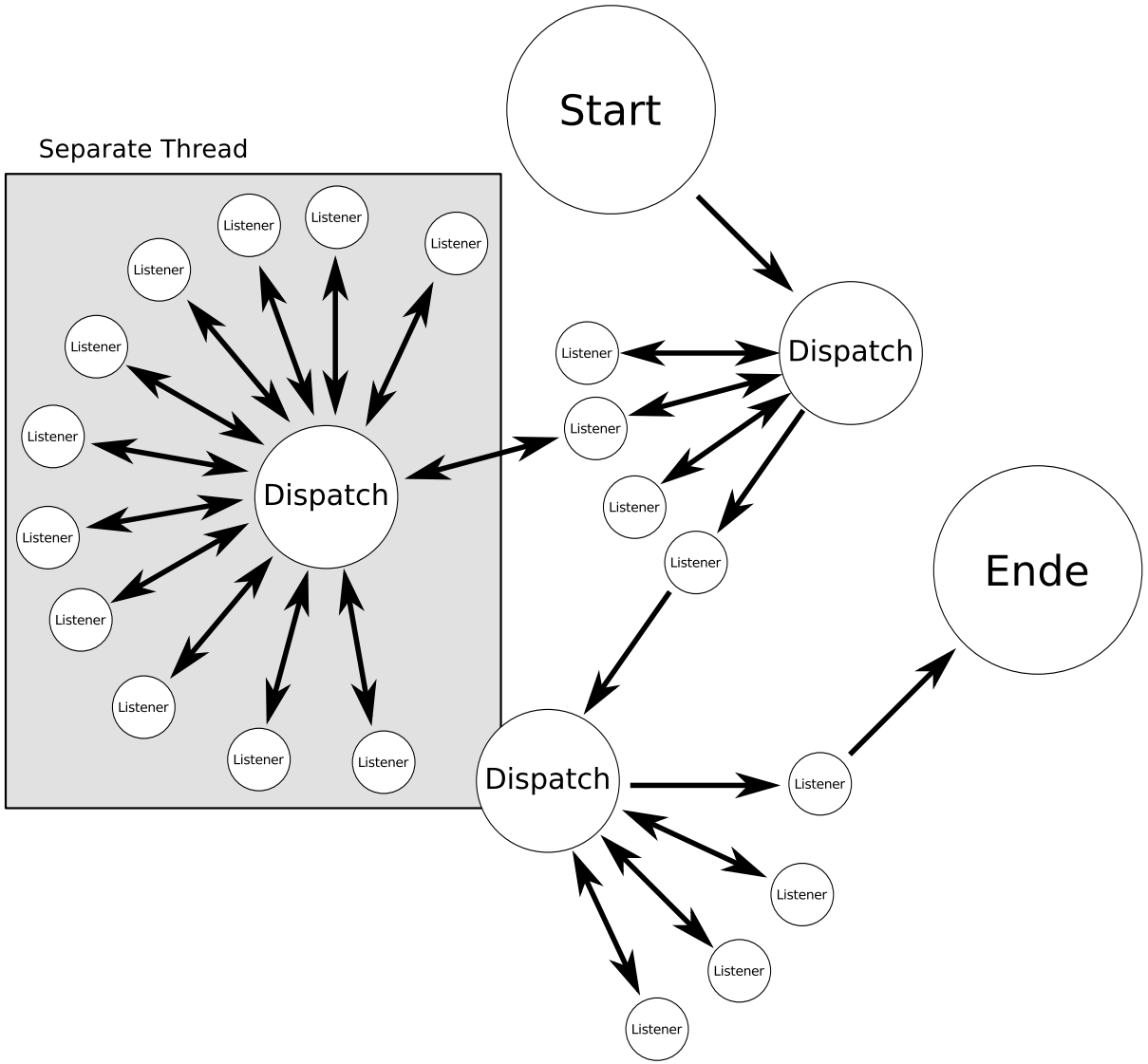
Eventgesteuerter Programmablauf




```php
$eventDispatcher->dispatch('start');



class Listener
{
  public function listen($event)
  {
    if ($event->getName() == 'start') {
      $message = 'Hello Mr. X';

      $manipulateMessageEvent = new ManipulateMe
      $manipulateMessageEvent->setMessage($messa

      $eventDispatcher->dispatch(
        'manipulate',
        $manipulateMessageEvent
      );

      $message = $manipulateMessageEvent->getMes

      echo $message;

      $event->stopPropagation();
    }
  }
}
```

Hooks vs. Events

Eventgesteuerter Programmablauf



Anwendungsbeispiele

Hooks vs. Events

Hooks in Contao


```php
class Listener {
  public function loadLanguageFile($strName, $strLanguage, $strCacheKey)
  {
    [...]
  }
}

// Listener registrieren
$GLOBALS['TL_HOOKS']['loadLanguageFile'][] = array('Listener', 'loadLanguageFile');

// Hooks aufrufen
if (isset($GLOBALS['TL_HOOKS']['loadLanguageFile']) &&
    is_array($GLOBALS['TL_HOOKS']['loadLanguageFile'])) {
  foreach ($GLOBALS['TL_HOOKS']['loadLanguageFile'] as $callback) {
    static::importStatic($callback[0])->$callback[1](
      $strName, $strLanguage, $strCacheKey);
  }
}
```

Hooks vs. Events

Events in Contao


```php
class Listener {
  public function loadLanguageFile(LoadLanguageFileEvent $event)
  {
    [...]
  }
}

// Listener registrieren
$eventDispatcher->addListener('load-language-file', array(new Listener(), 'loadLanguageFile'));

// Event dispatchen
$event = new LoadLanguageFileEvent();
$event->setName($strName)
      ->setLanguage($strLanguage)
      ->setCacheKey($strCacheKey);
$eventDispatcher->dispatch('load-language-file', $event);

.
```

Hooks vs. Events
Hooks in Contao


```php
class LoadLanguageFileEvent extends \Symfony\Component\EventDispatcher\Event
{
    protected $strName;
    protected $strLanguage;
    protected $strCacheKey;

    public function setName($strName)
    {
        $this->strName = $strName;
    }

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

    public function setLanguage($strLanguage)
    {
        $this->strLanguage = $strLanguage;
    }

    public function getLanguage()
    {
        return $this->strLanguage;
    }

    public function setCacheKey($strCacheKey)
    {
        $this->strCacheKey = $strCacheKey;
    }

    public function getCacheKey()
    {
        return $this->strCacheKey;
    }
}
```

Hooks vs. Events

Eventgesteuerte Programmierung

Pro Events

- Feingranulare Programmflusskontrolle.
- Leichtes Manipulieren der Programmlogik (nur neuer Listener).
- Leicht erweiterbar (nur neue Event Objekt Eigenschaft).
- Reduktion der Abhängigkeiten.
- *Inversion of Control*.

Kontra Events

- Steigende Komplexität des Gesamtsystems.
- Abhängigkeiten schwer oder gar nicht (aus Code) nachvollziehbar.
- *Inversion of Control*.

Hooks vs. Events

Eventgesteuerte Programmierung

Pro Events

- Feingranulare Programmflusskontrolle.
- Leichtes Manipulieren der Programmlogik (nur neuer Listener).
- Leicht erweiterbar (nur neue Event Objekt Eigenschaft).
- Reduktion der Abhängigkeiten.
- *Inversion of Control.*

Kontra Events

- Steigende Komplexität des Gesamtsystems.
- Abhängigkeiten schwer oder gar nicht (aus Code) nachvollziehbar.
- *Inversion of Control.*

Hooks vs. Events

Eventgesteuerte Programmierung

Pro Events

- Feingranulare Programmflusskontrolle.
- Leichtes Manipulieren der Programmlogik (nur neuer Listener).
- Leicht erweiterbar (nur neue Event Objekt Eigenschaft).
- Reduktion der Abhängigkeiten.
- *Inversion of Control.*

Kontra Events

- Steigende Komplexität des Gesamtsystems.
- Abhängigkeiten schwer oder gar nicht (aus Code) nachvollziehbar.
- *Inversion of Control.*

Hooks vs. Events

Eventgesteuerte Programmierung

Pro Events

- Feingranulare Programmflusskontrolle.
- Leichtes Manipulieren der Programmlogik (nur neuer Listener).
- Leicht erweiterbar (nur neue Event Objekt Eigenschaft).
- Reduktion der Abhängigkeiten.
- *Inversion of Control.*

Kontra Events

- Steigende Komplexität des Gesamtsystems.
- Abhängigkeiten schwer oder gar nicht (aus Code) nachvollziehbar.
- *Inversion of Control.*

Hooks vs. Events

Eventgesteuerte Programmierung

Pro Events

- Feingranulare Programmflusskontrolle.
- Leichtes Manipulieren der Programmlogik (nur neuer Listener).
- Leicht erweiterbar (nur neue Event Objekt Eigenschaft).
- Reduktion der Abhängigkeiten.
- *Inversion of Control.*

Kontra Events

- Steigende Komplexität des Gesamtsystems.
- Abhängigkeiten schwer oder gar nicht (aus Code) nachvollziehbar.
- *Inversion of Control.*

Hooks vs. Events
Eventgesteuerte Programmierung

Pro Events

- Feingranulare Programmflusskontrolle.
- Leichtes Manipulieren der Programmlogik (nur neuer Listener).
- Leicht erweiterbar (nur neue Event Objekt Eigenschaft).
- Reduktion der Abhängigkeiten.
- *Inversion of Control*.

Kontra Events

- Steigende Komplexität des Gesamtsystems.
- Abhängigkeiten schwer oder gar nicht (aus Code) nachvollziehbar.
- *Inversion of Control*.

Hooks vs. Events

Eventgesteuerte Programmierung

Pro Events

- Feingranulare Programmflusskontrolle.
- Leichtes Manipulieren der Programmlogik (nur neuer Listener).
- Leicht erweiterbar (nur neue Event Objekt Eigenschaft).
- Reduktion der Abhängigkeiten.
- *Inversion of Control*.

Kontra Events

- Steigende Komplexität des Gesamtsystems.
- Abhängigkeiten schwer oder gar nicht (aus Code) nachvollziehbar.
- *Inversion of Control*.

Hooks vs. Events

Eventgesteuerte Programmierung

Pro Events

- Feingranulare Programmflusskontrolle.
- Leichtes Manipulieren der Programmlogik (nur neuer Listener).
- Leicht erweiterbar (nur neue Event Objekt Eigenschaft).
- Reduktion der Abhängigkeiten.
- *Inversion of Control*.

Kontra Events

- Steigende Komplexität des Gesamtsystems.
- Abhängigkeiten schwer oder gar nicht (aus Code) nachvollziehbar.
- *Inversion of Control*.

Hooks vs. Events

Eventgesteuerte Programmierung

Pro Events

- Feingranulare Programmflusskontrolle.
- Leichtes Manipulieren der Programmlogik (nur neuer Listener).
- Leicht erweiterbar (nur neue Event Objekt Eigenschaft).
- Reduktion der Abhängigkeiten.
- *Inversion of Control*.

Kontra Events

- Steigende Komplexität des Gesamtsystems.
- Abhängigkeiten schwer oder gar nicht (aus Code) nachvollziehbar.
- *Inversion of Control*.

Dependency Injection

Static Singletons

Singleton (selten auch Einzelstück genannt)

- Gehört zur Kategorie der Erzeugungsmuster.
- Stellt sicher, dass von einer Klasse genau ein Objekt existiert.
- Singleton ist darüber hinaus üblicherweise global verfügbar.

Static Singletons

Singleton (selten auch Einzelstück genannt)

- Gehört zur Kategorie der Erzeugungsmuster.
- Stellt sicher, dass von einer Klasse genau ein Objekt existiert.
- Singleton ist darüber hinaus üblicherweise global verfügbar.

Static Singletons

Singleton (selten auch Einzelstück genannt)

- Gehört zur Kategorie der Erzeugungsmuster.
- Stellt sicher, dass von einer Klasse genau ein Objekt existiert.
- Singleton ist darüber hinaus üblicherweise global verfügbar.

Static Singletons
Vorteile

Vorteile (gegenüber globalen Variablen)

- Zugriffskontrolle kann realisiert werden.
- Kann durch Unterklassenbildung spezialisiert werden.
- Unterklasse kann zur Laufzeit bestimmt werden.
- Instanz muss nur erzeugt werden, wenn sie benötigt wird.
- Sollten später mehrere benötigt werden, ist eine Änderung leichter.

Static Singletons
Vorteile

Vorteile (gegenüber globalen Variablen)

- Zugriffskontrolle kann realisiert werden.
- Kann durch Unterklassenbildung spezialisiert werden.
- Unterklasse kann zur Laufzeit bestimmt werden.
- Instanz muss nur erzeugt werden, wenn sie benötigt wird.
- Sollten später mehrere benötigt werden, ist eine Änderung leichter.

Static Singletons
Vorteile

Vorteile (gegenüber globalen Variablen)

- Zugriffskontrolle kann realisiert werden.
- Kann durch Unterklassenbildung spezialisiert werden.
- Unterklasse kann zur Laufzeit bestimmt werden.
- Instanz muss nur erzeugt werden, wenn sie benötigt wird.
- Sollten später mehrere benötigt werden, ist eine Änderung leichter.

Static Singletons
Vorteile

Vorteile (gegenüber globalen Variablen)

- Zugriffskontrolle kann realisiert werden.
- Kann durch Unterklassenbildung spezialisiert werden.
- Unterklasse kann zur Laufzeit bestimmt werden.
- Instanz muss nur erzeugt werden, wenn sie benötigt wird.
- Sollten später mehrere benötigt werden, ist eine Änderung leichter.

Static Singletons
Vorteile

Vorteile (gegenüber globalen Variablen)

- Zugriffskontrolle kann realisiert werden.
- Kann durch Unterklassenbildung spezialisiert werden.
- Unterklasse kann zur Laufzeit bestimmt werden.
- Instanz muss nur erzeugt werden, wenn sie benötigt wird.
- Sollten später mehrere benötigt werden, ist eine Änderung leichter.

Static Singletons
Nachteile

Nachteile

- Exzessive Verwendung stellt ein Äquivalent zu globalen Variablen dar.
- Abhängigkeiten werden verschleiert, nur in Implementierung ersichtlich.
- Kopplung wird erhöht. Übersichtlichkeit einschränkt.
- Das Testen eines Singleton kann kompliziert sein.
  - Mocken ist aufwändig und teils unmöglich (Abhilfe: Reflection API).
- Konfiguration nur über weitere Singletons möglich (Environment, Registry, „well-known“ Files o. Ä).
- Ressource-Deallokation ist schwierig.

Static Singletons
Nachteile

Nachteile

- Exzessive Verwendung stellt ein Äquivalent zu globalen Variablen dar.
- Abhängigkeiten werden verschleiert, nur in Implementierung ersichtlich.
- Kopplung wird erhöht. Übersichtlichkeit einschränkt.
- Das Testen eines Singleton kann kompliziert sein.
  - Mocken ist aufwändig und teils unmöglich (Abhilfe: Reflection API).
- Konfiguration nur über weitere Singletons möglich (Environment, Registry, „well-known“ Files o. Ä).
- Ressource-Deallokation ist schwierig.

Static Singletons
Nachteile

Nachteile

- Exzessive Verwendung stellt ein Äquivalent zu globalen Variablen dar.
- Abhängigkeiten werden verschleiert, nur in Implementierung ersichtlich.
- Kopplung wird erhöht. Übersichtlichkeit einschränkt.
- Das Testen eines Singleton kann kompliziert sein.
  - Mocken ist aufwändig und teils unmöglich (Abhilfe: Reflection API).
- Konfiguration nur über weitere Singletons möglich (Environment, Registry, „well-known“ Files o. Ä).
- Ressource-Deallokation ist schwierig.

Static Singletons
Nachteile

Nachteile

- Exzessive Verwendung stellt ein Äquivalent zu globalen Variablen dar.
- Abhängigkeiten werden verschleiert, nur in Implementierung ersichtlich.
- Kopplung wird erhöht. Übersichtlichkeit einschränkt.
- Das Testen eines Singleton kann kompliziert sein.
  - Mocken ist aufwändig und teils unmöglich (Abhilfe: Reflection API).
- Konfiguration nur über weitere Singletons möglich (Environment, Registry, „well-known“ Files o. Ä).
- Ressource-Deallokation ist schwierig.

Static Singletons
Nachteile

Nachteile

- Exzessive Verwendung stellt ein Äquivalent zu globalen Variablen dar.
- Abhängigkeiten werden verschleiert, nur in Implementierung ersichtlich.
- Kopplung wird erhöht. Übersichtlichkeit einschränkt.
- Das Testen eines Singleton kann kompliziert sein.
  - Mocken ist aufwändig und teils unmöglich (Abhilfe: Reflection API).
- Konfiguration nur über weitere Singletons möglich (Environment, Registry, „well-known“ Files o. Ä).
- Ressource-Deallokation ist schwierig.

Static Singletons
Nachteile

Nachteile

- Exzessive Verwendung stellt ein Äquivalent zu globalen Variablen dar.
- Abhängigkeiten werden verschleiert, nur in Implementierung ersichtlich.
- Kopplung wird erhöht. Übersichtlichkeit einschränkt.
- Das Testen eines Singleton kann kompliziert sein.
  - Mocken ist aufwändig und teils unmöglich (Abhilfe: Reflection API).
- Konfiguration nur über weitere Singletons möglich (Environment, Registry, „well-known“ Files o. Ä).
- Ressource-Deallokation ist schwierig.

Static Singletons
Nachteile

Nachteile

- Exzessive Verwendung stellt ein Äquivalent zu globalen Variablen dar.
- Abhängigkeiten werden verschleiert, nur in Implementierung ersichtlich.
- Kopplung wird erhöht. Übersichtlichkeit einschränkt.
- Das Testen eines Singleton kann kompliziert sein.
  - Mocken ist aufwändig und teils unmöglich (Abhilfe: Reflection API).
- Konfiguration nur über weitere Singletons möglich (Environment, Registry, „well-known“ Files o. Ä).
- Ressource-Deallokation ist schwierig.

Static Singletons
Fazit

Fazit - Singletons:

- werden wegen der vielen Nachteile mitunter als Anti-Pattern bewertet.
- können sinnvoll sein wenn sie sich z.B. auf eine Abstract Factory beziehen.
- sind in der Regel schwieriger korrekt zu designen als Designs ohne Singletons.

Static Singletons
Fazit

Fazit - Singletons:

- werden wegen der vielen Nachteile mitunter als Anti-Pattern bewertet.
- können sinnvoll sein wenn sie sich z.B. auf eine Abstract Factory beziehen.
- sind in der Regel schwieriger korrekt zu designen als Designs ohne Singletons.

Static Singletons
Fazit

Fazit - Singletons:

- werden wegen der vielen Nachteile mitunter als Anti-Pattern bewertet.
- können sinnvoll sein wenn sie sich z.B. auf eine Abstract Factory beziehen.
- sind in der Regel schwieriger korrekt zu designen als Designs ohne Singletons.

Dependency Injection
Definition

Ursprung

2004 von Martin Fowler eingeführt um den damaligen Begriff Inversion of Control zu präzisieren.

Definition

Benötigt ein Objekt bei seiner Initialisierung ein anderes Objekt, ist diese Abhängigkeit hinterlegt und wird nicht vom initialisierten Objekt erzeugt („Hollywood-Prinzip“).

Dependency Injection
Varianten von DI

Varianten:

- Constructor Injection
- Setter Injection
- Interface Injection

Dependency Injection
Varianten von DI

Varianten:

- Constructor Injection
- Setter Injection
- Interface Injection

Dependency Injection
Varianten von DI

Varianten:

- Constructor Injection
- Setter Injection
- Interface Injection

Dependency Injection
Constructor Injection


```php
interface Dependency
{
  /* ... */
}



class Dependant
{
  /** @var Dependency */
  private $dependency;

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

class Injector
{
  public function method()
  {
    /** @var Dependency $dependency */
    $dependency = /* ... irgendwoher holen ... */;
    /** @var Dependant $injectable */
    $injectable = new Dependant($dependency);

  }
}
```

Dependency Injection
Setter Injection


```php
interface Dependency
{
  /* ... */
}



class Dependant
{
  /** @var Dependency */
  private $dependency;

  public function setDependency(Dependency $dependency)
  {
    $this->dependency = $dependency;
  }
}

class Injector
{
  public function method()
  {
    /** @var Dependency $dependency */
    $dependency = /* ... irgendwoher holen ... */;
    /** @var Dependant $injectable */
    $injectable = new Dependant();
    $injectable->setDependency($dependency);
  }
}
```

Dependency Injection
Interface Injection


```php
interface Dependency
{
  /* ... */
}

interface Injectable
{
  public function inject(Dependency $dependency);
}

class Dependant implements Injectable
{
  /** @var Dependency */
  private $dependency;

  public function inject(Dependency $dependency)
  {
    $this->dependency = $dependency;
  }
}

class Injector
{
  public function method()
  {
    /** @var Dependency $dependency */
    $dependency = /* ... irgendwoher holen ... */;
    /** @var Injectable $injectable */
    $injectable = /* ... irgendwoher holen ... */;
    $injectable->inject($dependency);
  }
}
```

Dependency Injection
Container

Vorteile:

- Reduzierung von Code (Abhängigkeiten kommen von Provider).
- Flexibilität in Konfiguration - Komponentenaustausch möglich.
- Leichtes Unit testing da MockUps „injected“ werden können.

Nachteile:

- Abhängigkeiten nach aussen hin „Black Boxen“ und schwer durchschaubar.
- Verschleiert Klassenhierarchien, RuntimeError statt ClassNotFound.
- Anfälliger für BC breaks. Wann ändert sich ein Interface im Container?
- In Anwendungen mit eng umrissenen Umfeld sinnlos.

Dependency Injection
Container

Vorteile:

- Reduzierung von Code (Abhängigkeiten kommen von Provider).
- Flexibilität in Konfiguration - Komponentenaustausch möglich.
- Leichtes Unit testing da MockUps „injected“ werden können.

Nachteile:

- Abhängigkeiten nach aussen hin „Black Boxen“ und schwer durchschaubar.
- Verschleiert Klassenhierarchien, RuntimeError statt ClassNotFound.
- Anfälliger für BC breaks. Wann ändert sich ein Interface im Container?
- In Anwendungen mit eng umrissenen Umfeld sinnlos.

Dependency Injection
Container

Vorteile:

- Reduzierung von Code (Abhängigkeiten kommen von Provider).
- Flexibilität in Konfiguration - Komponentenaustausch möglich.
- Leichtes Unit testing da MockUps „injected“ werden können.

Nachteile:

- Abhängigkeiten nach aussen hin „Black Boxen“ und schwer durchschaubar.
- Verschleiert Klassenhierarchien, RuntimeError statt ClassNotFound.
- Anfälliger für BC breaks. Wann ändert sich ein Interface im Container?
- In Anwendungen mit eng umrissenen Umfeld sinnlos.

Dependency Injection
Container

Vorteile:

- Reduzierung von Code (Abhängigkeiten kommen von Provider).
- Flexibilität in Konfiguration - Komponentenaustausch möglich.
- Leichtes Unit testing da MockUps „injected“ werden können.

Nachteile:

- Abhängigkeiten nach aussen hin „Black Boxen“ und schwer durchschaubar.
- Verschleiert Klassenhierarchien, RuntimeError statt ClassNotFound.
- Anfälliger für BC breaks. Wann ändert sich ein Interface im Container?
- In Anwendungen mit eng umrissenen Umfeld sinnlos.

Dependency Injection
Container

Vorteile:

- Reduzierung von Code (Abhängigkeiten kommen von Provider).
- Flexibilität in Konfiguration - Komponentenaustausch möglich.
- Leichtes Unit testing da MockUps „injected“ werden können.

Nachteile:

- Abhängigkeiten nach aussen hin „Black Boxen“ und schwer durchschaubar.
- Verschleiert Klassenhierarchien, RuntimeError statt ClassNotFound.
- Anfälliger für BC breaks. Wann ändert sich ein Interface im Container?
- In Anwendungen mit eng umrissenen Umfeld sinnlos.

Dependency Injection
Container

Vorteile:

- Reduzierung von Code (Abhängigkeiten kommen von Provider).
- Flexibilität in Konfiguration - Komponentenaustausch möglich.
- Leichtes Unit testing da MockUps „injected“ werden können.

Nachteile:

- Abhängigkeiten nach aussen hin „Black Boxen“ und schwer durchschaubar.
- Verschleiert Klassenhierarchien, RuntimeError statt ClassNotFound.
- Anfälliger für BC breaks. Wann ändert sich ein Interface im Container?
- In Anwendungen mit eng umrissenen Umfeld sinnlos.

Dependency Injection
Container

Vorteile:

- Reduzierung von Code (Abhängigkeiten kommen von Provider).
- Flexibilität in Konfiguration - Komponentenaustausch möglich.
- Leichtes Unit testing da MockUps „injected“ werden können.

Nachteile:

- Abhängigkeiten nach aussen hin „Black Boxen“ und schwer durchschaubar.
- Verschleiert Klassenhierarchien, RuntimeError statt ClassNotFound.
- Anfälliger für BC breaks. Wann ändert sich ein Interface im Container?
- In Anwendungen mit eng umrissenen Umfeld sinnlos.

Factory

Factory Pattern

- Gehört zu der Kategorie der Erzeugungsmuster (engl. creational patterns).
- Kapselung der Erzeugung von Objekten.
- Kapselung der (initialen) Konfiguration von Objekten.

Factory Pattern

- Gehört zu der Kategorie der Erzeugungsmuster (engl. creational patterns).
- Kapselung der Erzeugung von Objekten.
- Kapselung der (initialen) Konfiguration von Objekten.

Factory Pattern

- Gehört zu der Kategorie der Erzeugungsmuster (engl. creational patterns).
- Kapselung der Erzeugung von Objekten.
- Kapselung der (initialen) Konfiguration von Objekten.

Factory Pattern
getInstance() Fabrikmethode


```php
class Database
{
    public static function getInstance(array $arrCustom = null)
    {
        $arrConfig = array
        (
            'dbDriver'   => $GLOBALS['TL_CONFIG']['dbDriver'],
            'dbHost'     => $GLOBALS['TL_CONFIG']['dbHost'],
            'dbUser'     => $GLOBALS['TL_CONFIG']['dbUser'],
            'dbPass'     => $GLOBALS['TL_CONFIG']['dbPass'],
            'dbDatabase' => $GLOBALS['TL_CONFIG']['dbDatabase'],
            'dbPconnect' => $GLOBALS['TL_CONFIG']['dbPconnect'],
            'dbCharset'  => $GLOBALS['TL_CONFIG']['dbCharset'],
            'dbPort'     => $GLOBALS['TL_CONFIG']['dbPort'],
            'dbSocket'   => $GLOBALS['TL_CONFIG']['dbSocket']
        );

        if (is_array($arrCustom)) {
            $arrConfig = array_merge($arrConfig, $arrCustom);
        }

        // Sort the array before generating the key
        ksort($arrConfig);
        $strKey = md5(implode('', $arrConfig));

        if (!isset(static::$arrInstances[$strKey])) {
            $strClass = 'Database\\' . str_replace(
                    ' ',
                    '_',
                    ucwords(str_replace('_', ' ', strtolower($arrConfig['dbDriver'])))
                );
            static::$arrInstances[$strKey] = new $strClass($arrConfig);
        }

        return static::$arrInstances[$strKey];
    }
}
```

Factory Pattern
Fahrzeug Factory Mockup


```php
class CarFactory
{
    public function setManufacturer($manufacturer)
    {
        // [...]
    }

    public function setModel($model)
    {
        // [...]
    }

    /**
     * @return Car
     */
    public function createNewCar()
    {
        $car = new Car();
        $car->setManufacturer($this->manufacturer);
        $car->setModel($this->model);
        return $car;
    }
}
```

Factory Pattern
Fahrzeug Factory in Action


```php
// global code
$porscheBoxterFactory = new CarFactory();
$porscheBoxterFactory->setManufacturer('Porsche');
$porscheBoxterFactory->setModel('Boxter');

// local code
$porscheBoxter = $porscheBoxterFactory->createNewCar();
```

Factory Pattern
Factory im DI

- DI weiß nicht, wie ein Objekt initialisiert wird
- DI weiß nicht, wie ein Objekt konfiguriert wird
- Unverzichtbar für *lazy initialisation*

Factory Pattern
Factory im DI

- DI weiß nicht, wie ein Objekt initialisiert wird
- DI weiß nicht, wie ein Objekt konfiguriert wird
- Unverzichtbar für *lazy initialisation*

Factory Pattern
Factory im DI

- DI weiß nicht, wie ein Objekt initialisiert wird
- DI weiß nicht, wie ein Objekt konfiguriert wird
- Unverzichtbar für *lazy initialisation*

Fragen v.1.0

Fragen?

Pause

Part 2

Part 2

- Decoupling
- Dependency Injection Praxisbeispiel in Contao
- Event Dispatcher Praxisbeispiel in Contao

Part 2

Part 2

- Decoupling
- Dependency Injection Praxisbeispiel in Contao
- Event Dispatcher Praxisbeispiel in Contao

Part 2

Part 2

- Decoupling
- Dependency Injection Praxisbeispiel in Contao
- Event Dispatcher Praxisbeispiel in Contao

Decoupling

Decoupling
Hardcoding nightmare!


```php
class PorscheFactory
{
    /**
     * @return Car
     */
    public function createNewCar()
    {
        $bodyShop  = new BodyShop911Carrera();
        $paintShop = new PaintShop('red');
        $saddlery  = new Saddlery();

        $car = $bodyShop->createBody();
        $paintShop->paint($car);
        $saddlery->putSeats($car);

        $car->setManufacturer('Porsche');
        $car->setModel('911 Carrera');

        return $car;
    }
}
```

Decoupling
Aufgelöst.


```php
class CarFactory
{
    public function __construct($bodyShop, $paintShop)
    {
        // Karosseriewerkstatt und Lackiererei merken.
        $this->bodyShop  = $bodyShop;
        $this->paintShop = $paintShop;
    }

    public function setSaddlery($saddlery)
    {
        // Sattlerei merken.
        $this->saddlery = $saddlery;
    }

    /**
     * @return Car
     */
    public function createNewCar()
    {
        // Auto bauen, lackieren und Sitze rein!
        $car = $this->bodyShop->createBody();
        $this->paintShop->paint($car);
        $this->saddlery->putSeats($car);

        $car->setManufacturer($this->manufacturer);
        return $car;
    }
}
```

Decoupling
Aufgelöst.


```php
// Diverse Karosseriewerkstaetten erzeugen.
$container['carreraBodyShop'] = new BodyShop911Carrera();
$container['cayenneBodyShop'] = new BodyShopCayenne();

// Lackierereien brauchen wir auch.
$container['blackPaintShop']  = new PaintShop('black');
$container['silverPaintShop'] = new PaintShop('silver');

// Und eine Sattlerei waer auch ned schlecht.
$container['saddlery']  = new Saddlery();
```

Decoupling
Aufgelöst.


```php
// In dieser Fabrik stellt man nur schwarze Carreras her.
$container['blackCarreraFactory'] = function($container) {
    $carreraFactory = new CarFactory(
        $container['carreraBodyShop'],
        $container['blackPaintShop']
    );
    $carreraFactory->setSaddlery($container['saddlery']);
    return $carreraFactory;
}

// In dieser nur silberne Carreras.
$container['silverCarreraFactory'] = function($container) {
    $carreraFactory = new CarFactory(
        $container['carreraBodyShop'],
        $container['silverPaintShop']
    );
    $carreraFactory->setSaddlery($container['saddlery']);
    return $carreraFactory;
}

// in dieser nur schwarze Cayennes.
$container['blackCayenneFactory'] = function($container) {
    $cayenneFactory = new CarFactory(
        $container['cayenneBodyShop'],
        $container['blackPaintShop']
    );
    $cayenneFactory->setSaddlery($container['saddlery']);
    return $cayenneFactory;
}

// in dieser die silbernen Cayennes.
$container['silverCayenneFactory'] = function($container) {
    $cayenneFactory = new CarFactory(
        ...
```

Decoupling
Aufgelöst.


```php
// Einen schwarzen Carrera bitte damit ich schnell heim komme.
$blackCarreraFactory = $container['blackCarreraFactory'];
$blackCarrera = $blackCarreraFactory->createNewCar();

// und tschuess...
$blackCarrera->fahrenNach('Zuhause');
```

Dependency Injection in Contao

Dependency Injection in Contao

Der Dependency Injection Container


```php
/** @var \Pimple $container */
global $container;
```

Extension

- Composer: contao-community-alliance/dependency-container
- Extension Repository: dependency-container (meist veraltet!)

Referenz

- Verwendet den Pimple Dependency Container
- Composer: pimple/pimple
- Source: https://github.com/fabpot/Pimple

Dependency Injection in Contao

Der Dependency Injection Container


```php
/** @var \Pimple $container */
global $container;
```

Extension

- Composer: contao-community-alliance/dependency-container
- Extension Repository: dependency-container (meist veraltet!)

Referenz

- Verwendet den Pimple Dependency Container
- Composer: pimple/pimple
- Source: https://github.com/fabpot/Pimple

Dependency Injection in Contao

Der Dependency Injection Container


```php
/** @var \Pimple $container */
global $container;
```

Extension

- Composer: contao-community-alliance/dependency-container
- Extension Repository: dependency-container (meist veraltet!)

Referenz

- Verwendet den Pimple Dependency Container
- Composer: pimple/pimple
- Source: https://github.com/fabpot/Pimple

Dependency Injection in Contao

Definieren von Parametern und Services


```php
/** @var \Pimple $container */
global $container;

// (Konfigurations-)Parameter setzen
$container['manufacturer'] = 'Porsche';
$container['model']        = 'Boxter';
```

.

Dependency Injection in Contao

Definieren von Parametern und Services


```php
/** @var \Pimple $container */
global $container;

// (Konfigurations-)Parameter setzen
$container['manufacturer'] = 'Porsche';
$container['model']        = 'Boxter';

// Service definieren
$porscheBoxterFactory = new CarFactory();
$porscheBoxterFactory->setManufacturer($container['manufacturer']);
$porscheBoxterFactory->setModel($container['model']);

$container['porscheBoxterFactory'] = $porscheBoxterFactory;
```

.

Dependency Injection in Contao

Definieren von Parametern und Services


```php
/** @var \Pimple $container */
global $container;

// (Konfigurations-)Parameter setzen
$container['manufacturer'] = 'Porsche';
$container['model']        = 'Boxter';

// Service definieren
$porscheBoxterFactory = new CarFactory();
$porscheBoxterFactory->setManufacturer($container['manufacturer']);
$porscheBoxterFactory->setModel($container['model']);

$container['porscheBoxterFactory'] = $porscheBoxterFactory;

// Lazy Service definieren
$container['porscheBoxterFactory'] = function($container) {
    $porscheBoxterFactory = new CarFactory();
    $porscheBoxterFactory->setManufacturer($container['manufacturer']);
    $porscheBoxterFactory->setModel($container['model']);
    return $porscheBoxterFactory;
};
```

.

Dependency Injection in Contao

Definieren von Parametern und Services


```php
/** @var \Pimple $container */
global $container;

// (Konfigurations-)Parameter setzen
$container['manufacturer'] = 'Porsche';
$container['model']        = 'Boxter';

// Service definieren
$porscheBoxterFactory = new CarFactory();
$porscheBoxterFactory->setManufacturer($container['manufacturer']);
$porscheBoxterFactory->setModel($container['model']);

$container['porscheBoxterFactory'] = $porscheBoxterFactory;

// Lazy Service definieren
$container['porscheBoxterFactory'] = function($container) {
    $porscheBoxterFactory = new CarFactory();
    $porscheBoxterFactory->setManufacturer($container['manufacturer']);
    $porscheBoxterFactory->setModel($container['model']);
    return $porscheBoxterFactory;
};

// Singleton Lazy Service definieren
$container['porscheBoxterFactory'] = $container->share(
    function($container) {
        $porscheBoxterFactory = new CarFactory();
        $porscheBoxterFactory->setManufacturer($container['manufacturer']);
        $porscheBoxterFactory->setModel($container['model']);
        return $porscheBoxterFactory;
    }
);



.
```

Dependency Injection in Contao

Definieren von Parametern und Services


```php
/** @var \Pimple $container */
global $container;

// (Konfigurations-)Parameter setzen
$container['manufacturer'] = 'Porsche';
$container['model']        = 'Boxter';

// Service definieren
$porscheBoxterFactory = new CarFactory();
$porscheBoxterFactory->setManufacturer($container['manufacturer']);
$porscheBoxterFactory->setModel($container['model']);

$container['porscheBoxterFactory'] = $porscheBoxterFactory;

// Lazy Service definieren
$container['porscheBoxterFactory'] = function($container) {
    $porscheBoxterFactory = new CarFactory();
    $porscheBoxterFactory->setManufacturer($container['manufacturer']);
    $porscheBoxterFactory->setModel($container['model']);
    return $porscheBoxterFactory;
};

// Singleton Lazy Service definieren
$container['porscheBoxterFactory'] = $container->share(
    function($container) {
        $porscheBoxterFactory = new CarFactory();
        $porscheBoxterFactory->setManufacturer($container['manufacturer']);
        $porscheBoxterFactory->setModel($container['model']);
        return $porscheBoxterFactory;
    }
);

// Alias
$container['defaultPorscheBoxterFactory'] = function($container) {
    return $container['porscheBoxterFactory'];
};
```

Dependency Injection in Contao

Abfragen von Parametern und Services


```php
class CustomElement extends \ContentElement
{
    protected function compile()
    {
        /** @var \Pimple $container */
        global $container;

        $this->Template->manufacturer = $container['manufacturer'];
        $this->Template->model        = $container['model'];

        /** @var CarFactory $factory */
        $factory = $container['porscheBoxterFactory'];

        $this->Template->car = $factory->createNewCar();
    }
}
```

Event Dispatcher in Contao

Event Dispatcher in Contao

Der Event Dispatcher


```php
/** @var \Pimple $container */
global $container;

/** @var \Symfony\Component\EventDispatcher\EventDispatcher $eventDispatcher */
$eventDispatcher = $container['event-dispatcher'];
```

Extension

- Composer: contao-community-alliance/event-dispatcher
- Extension Repository: event-dispatcher (meist veraltet!)

Referenz

- Verwendet den Symfony2 EventDispatcher
- Composer: symfony/event-dispatcher
- Source: https://github.com/symfony/EventDispatcher

Event Dispatcher in Contao

Der Event Dispatcher


```php
/** @var \Pimple $container */
global $container;

/** @var \Symfony\Component\EventDispatcher\EventDispatcher $eventDispatcher */
$eventDispatcher = $container['event-dispatcher'];
```

Extension

- Composer: contao-community-alliance/event-dispatcher
- Extension Repository: event-dispatcher (meist veraltet!)

Referenz

- Verwendet den Symfony2 EventDispatcher
- Composer: symfony/event-dispatcher
- Source: https://github.com/symfony/EventDispatcher

Event Dispatcher in Contao

Der Event Dispatcher


```php
/** @var \Pimple $container */
global $container;

/** @var \Symfony\Component\EventDispatcher\EventDispatcher $eventDispatcher */
$eventDispatcher = $container['event-dispatcher'];
```

Extension

- Composer: contao-community-alliance/event-dispatcher
- Extension Repository: event-dispatcher (meist veraltet!)

Referenz

- Verwendet den Symfony2 EventDispatcher
- Composer: symfony/event-dispatcher
- Source: https://github.com/symfony/EventDispatcher

Dependency Injection in Contao

Registrieren von Listenern


```php
// als Closure
$GLOBALS['TL_EVENTS']['manipulate'][] = function($event) {
    // [...]
};
```

Dependency Injection in Contao

Registrieren von Listenern


```php
// als Closure
$GLOBALS['TL_EVENTS']['manipulate'][] = function($event) {
    // [...]
};

// als statische Methode
$GLOBALS['TL_EVENTS']['manipulate'][] = 'Listener::manipulate';
```

.

Dependency Injection in Contao

Registrieren von Listenern


```php
// als Closure
$GLOBALS['TL_EVENTS']['manipulate'][] = function($event) {
    // [...]
};

// als statische Methode
$GLOBALS['TL_EVENTS']['manipulate'][] = 'Listener::manipulate';

// als Instanz-Methode
$GLOBALS['TL_EVENTS']['manipulate'][] = array(new Listener(), 'manipulate');
```

.

Dependency Injection in Contao

Registrieren von Listenern


```php
// als Closure
$GLOBALS['TL_EVENTS']['manipulate'][] = function($event) {
    // [...]
};

// als statische Methode
$GLOBALS['TL_EVENTS']['manipulate'][] = 'Listener::manipulate';

// als Instanz-Methode
$GLOBALS['TL_EVENTS']['manipulate'][] = array(new Listener(), 'manipulate');

// als Service
$GLOBALS['TL_EVENTS']['manipulate'][] = function($event) {
    global $container;
    $container['manipulate-listener']->manipulate($event);
};
```

Dependency Injection in Contao

Registrieren von Listenern


```php
// als Closure
$GLOBALS['TL_EVENTS']['manipulate'][] = function($event) {
    // [...]
};

// als statische Methode
$GLOBALS['TL_EVENTS']['manipulate'][] = 'Listener::manipulate';

// als Instanz-Methode
$GLOBALS['TL_EVENTS']['manipulate'][] = array(new Listener(), 'manipulate');

// als Service
$GLOBALS['TL_EVENTS']['manipulate'][] = function($event) {
    global $container;
    $container['manipulate-listener']->manipulate($event);
};

// via Code
$container['event-dispatcher']->addListener('manipulate', [callable]);
```

.

Dependency Injection in Contao

Registrieren von Listenern


```php
// als Closure
$GLOBALS['TL_EVENTS']['manipulate'][] = function($event) {
    // [...]
};

// als statische Methode
$GLOBALS['TL_EVENTS']['manipulate'][] = 'Listener::manipulate';

// als Instanz-Methode
$GLOBALS['TL_EVENTS']['manipulate'][] = array(new Listener(), 'manipulate');

// als Service
$GLOBALS['TL_EVENTS']['manipulate'][] = function($event) {
    global $container;
    $container['manipulate-listener']->manipulate($event);
};

// via Code
$container['event-dispatcher']->addListener('manipulate', [callable]);



class Listener
{
    [static] public function manipulate(ManipulateEvent $event)
    {
        // [...]
    }
}
```

Dependency Injection in Contao

Registrieren von Subscribern


```php
// via Factory
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = function($eventDispatcher) {
    return new Subscriber();
};
```

Dependency Injection in Contao

Registrieren von Subscribern


```php
// via Factory
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = function($eventDispatcher) {
    return new Subscriber();
};

// via Klasse
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = 'Subscriber';
```

.

Dependency Injection in Contao

Registrieren von Subscribern


```php
// via Factory
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = function($eventDispatcher) {
    return new Subscriber();
};

// via Klasse
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = 'Subscriber';

// via Instanz
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = new Subscriber();
```

Dependency Injection in Contao

Registrieren von Subscribern


```php
// via Factory
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = function($eventDispatcher) {
    return new Subscriber();
};

// via Klasse
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = 'Subscriber';

// via Instanz
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = new Subscriber();

// via Service
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = function($eventDispatcher) {
    global $container;
    return $container['subscriber'];
};
```

.

Dependency Injection in Contao

Registrieren von Subscribern


```php
// via Factory
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = function($eventDispatcher) {
    return new Subscriber();
};

// via Klasse
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = 'Subscriber';

// via Instanz
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = new Subscriber();

// via Service
$GLOBALS['TL_EVENT_SUBSCRIBERS'][] = function($eventDispatcher) {
    global $container;
    return $container['subscriber'];
};



class Subscriber implements \Symfony\Component\EventDispatcher\EventSubscriberInterface
{
    public static function getSubscribedEvents()
    {
        return array(
            'manipulate' => 'doManipulation',
        );
    }

    public function doManipulation(ManipulateEvent $event)
    {
        // [...]
    }
}
```

Dependency Injection in Contao

Senden von Events


```php
class CustomElement extends \ContentElement
{
    protected function compile()
    {
        /** @var \Pimple $container */
        global $container;

        /** @var \Symfony\Component\EventDispatcher\EventDispatcher $eventDispatcher */
        $eventDispatcher = $container['event-dispatcher'];

        $event = new CreateCarEvent();
        $event->setManufacturer($container['manufacturer']);
        $event->setModel($container['model']);

        $eventDispatcher->dispatch('create-car', $event);

        $car = $event->getCar();

        $this->Template->manufacturer = $car->getManufacturer();
        $this->Template->model        = $car->getModel();
        $this->Template->car          = $car;
    }
}
```

Fertige Events für Contao

Fertige Events für Contao

Cron Events


```php
$GLOBALS['TL_EVENTS']['cron.monthly'][] = function($event) {
    // run monthly
};

$GLOBALS['TL_EVENTS']['cron.weekly'][] = function($event) {
    // run weekly
};

$GLOBALS['TL_EVENTS']['cron.daily'][] = function($event) {
    // run daily
};

$GLOBALS['TL_EVENTS']['cron.hourly'][] = function($event) {
    // run hourly
};

// only support in Contao 3.0 or newer
$GLOBALS['TL_EVENTS']['cron.minutely'][] = function($event) {
    // run minutely
};
```

Extension

- Composer: contao-community-alliance/events-cron
- Extension Repository: *nicht verfügbar*
- Source: https://github.com/contao-community-alliance/events-cron

Fertige Events für Contao

Cron Events


```php
$GLOBALS['TL_EVENTS']['cron.monthly'][] = function($event) {
    // run monthly
};

$GLOBALS['TL_EVENTS']['cron.weekly'][] = function($event) {
    // run weekly
};

$GLOBALS['TL_EVENTS']['cron.daily'][] = function($event) {
    // run daily
};

$GLOBALS['TL_EVENTS']['cron.hourly'][] = function($event) {
    // run hourly
};

// only support in Contao 3.0 or newer
$GLOBALS['TL_EVENTS']['cron.minutely'][] = function($event) {
    // run minutely
};
```

Extension

- Composer: contao-community-alliance/events-cron
- Extension Repository: *nicht verfügbar*
- Source: https://github.com/contao-community-alliance/events-cron

Fertige Events für Contao

Create Options Event (Options Callback Ersatz)


```php
// DCA
use ContaoCommunityAlliance\Contao\Events\CreateOptions\CreateOptionsEventCallbackFactory;

$GLOBALS['TL_DCA']['tl_car']['fields']['manufacturer'] = array(
    ...
    'options_callback' => CreateOptionsEventCallbackFactory::createCallback(
        'tl_car.manufacturer.create-options'
    ),
);



// Listener
$GLOBALS['TL_EVENTS']['tl_car.manufacturer.create-options'][] = function($event) {
    $options = $event->getOptions();

    $options['value1'] = 'Porsche';
    $options['value2'] = 'Jaguar';
    $options['value3'] = 'Infinity';
};
```

Extension

- Composer: contao-community-alliance/events-create-options
- Extension Repository: *nicht verfügbar*
- Source: https://github.com/contao-community-alliance/events-create-options

Fertige Events für Contao

Create Options Event (Options Callback Ersatz)


```php
// DCA
use ContaoCommunityAlliance\Contao\Events\CreateOptions\CreateOptionsEventCallbackFactory;

$GLOBALS['TL_DCA']['tl_car']['fields']['manufacturer'] = array(
    ...
    'options_callback' => CreateOptionsEventCallbackFactory::createCallback(
        'tl_car.manufacturer.create-options'
    ),
);



// Listener
$GLOBALS['TL_EVENTS']['tl_car.manufacturer.create-options'][] = function($event) {
    $options = $event->getOptions();

    $options['value1'] = 'Porsche';
    $options['value2'] = 'Jaguar';
    $options['value3'] = 'Infinity';
};
```

Extension

- Composer: contao-community-alliance/events-create-options
- Extension Repository: *nicht verfügbar*
- Source: https://github.com/contao-community-alliance/events-create-options

Fragen v.2.0

Fragen?