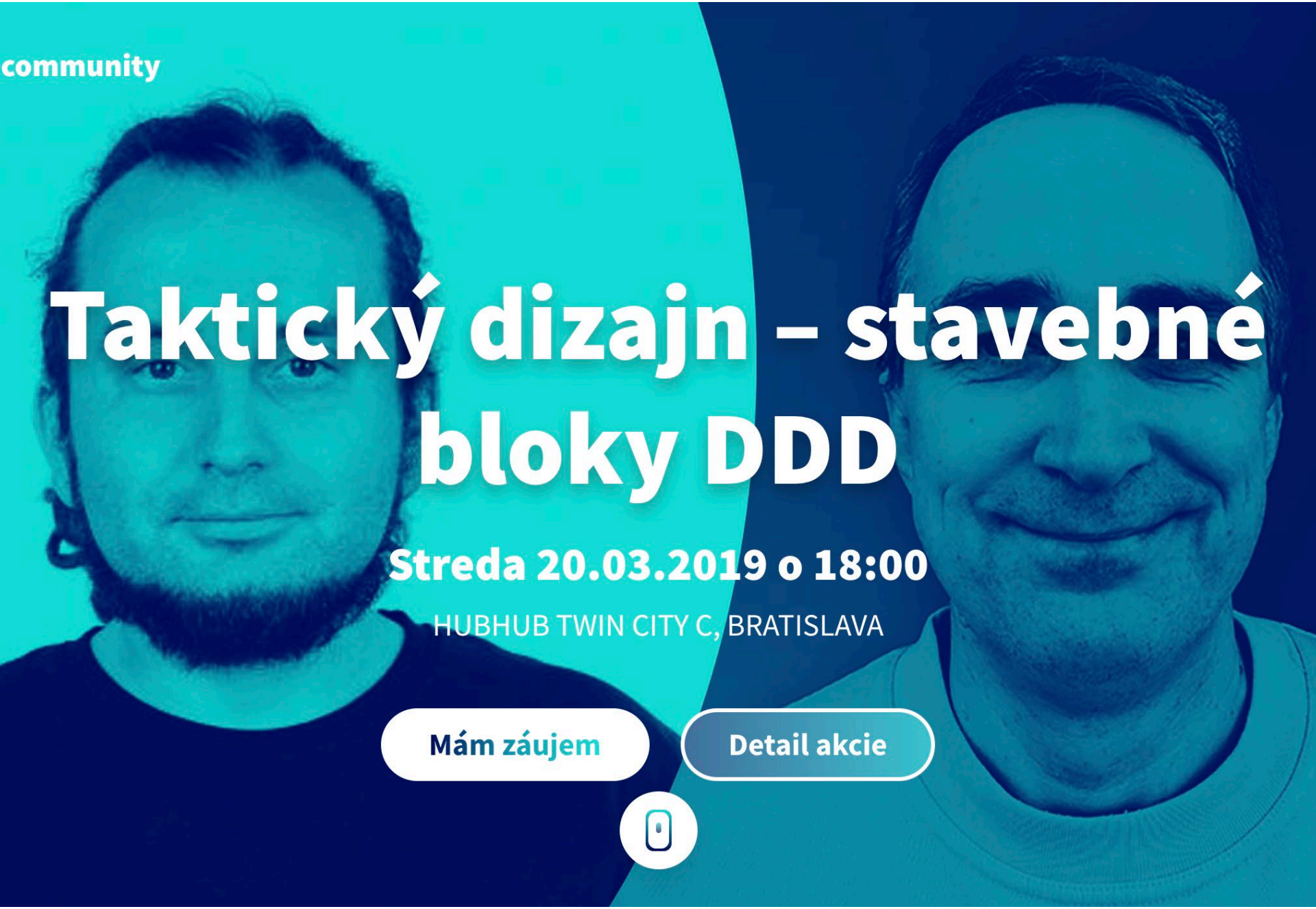

community
Taktický dizajn – stavebné bloky DDD
Streda 20.03.2019 o 18:00
HUBHUB TWIN CITY C, BRATISLAVA
Mám záujem
Detail akcie

PIXELFEDERATION

DDD - Tactical Patterns & Value Objects

Martin Fris

Martin Fris

Backend Sw. Architect In Pixelfederation

Interested In:

Programming, Design Patterns, SOLID, DDD, Infrastructure As Code (Kubernetes…) DevOps, Beer

Contact:

 mfris@pixelfederation.com

 @Rastusik

Plan

1. Projects Introduction
2. Tactical Patterns In DDD
3. Value Objects Design And Implementation
    - Problems & Solutions We Needed To Handle

Projects Using DDD Patterns

TrainStation 2

Projects Using DDD Patterns

TrainStation 2

Lines of code: **82 393**
Classes: **1 351**

Lines of code in the Domain: **38 060**
Classes: **594**

Projects Using DDD Patterns

Steam Cats

Projects Using DDD Patterns

Steam Cats

Lines of code: **105 881**
Classes: **2 027**

Lines of code in the Domain: **42 048**
Classes: **703**

Tactical Patterns In DDD

The Important Part In DDD Is NOT The Implementation

DDD evangelists and influencers say:

- Most of the companies only use "**DDD Lite**", not proper DDD
- Tactical patterns only help in the implementation phase
- Implementation is not the most important thing

Tactical Patterns In DDD

The Interesting Parts (Strategic Patterns)

- Domain Discovery & Distillation
- Context Mapping
- Interaction with Domain Experts
- Tighter Cooperation With Business

Tactical Patterns In DDD

BUT! The Boring Parts Aren't That Boring Or Useless

- Software eventually needs to get implemented
- It seems that DDD Lite patterns are a good way to prepare a project for longer life cycle and change
- **If understood properly and applied correctly** :)

Tactical Patterns In DDD

What Are The Tactical Patterns?

- Guidance to implementation
- Goal:
  - High quality code
  - Maintainable application
  - Scalable application

Tactical Patterns In DDD

How Are They Trying To Achieve The Goals?

Separation of application into layers:

- **Domain**
- Application
- Infrastructure

Tactical Patterns In DDD

Domain Layer

- The most important of them all
- Separated from the infrastructural details
  - => Easier infrastructural changes
  - => The most important information is completely isolated => easier changeable

Tactical Patterns In DDD

Implementation Building Blocks

- Value Objects
- Repositories
- Entities
- Aggregates
- Services
- Events
- Process Managers (“Sagas”)

Tactical Patterns In DDD

Our Design Decisions With:

- **Value Objects**
- Repositories
- Entities
- Aggregates
- Services
- Events
- Process Managers

Value Objects

Definition

- Value represented by object
- Distinguishable by it’s value
- Don’t have identifiers
- Immutable
- Contain basic logic

Value Objects

Example: Email

```php
final class EmailAddress
{
    /**
     * @var string
     */
    private $email;

    public function __construct(string $email)
    {
        if (!filter_var($email, FILTER_VALIDATE_EMAIL)) {
            throw new DomainException('Invalid email.');
        }

        $this->email = $email;
    }
}
```

Value Objects

Practical Benefits

- Constraints to the wrapped primitive values
  - => The value in the object is always valid
- Contains all valid operations bounded to the value/concept
- Concept naming
- Usable in threaded environments because of immutability

Value Objects

Code Without Using Value Objects Everywhere

```php
final class SpeedUpPriceCalculator
{
    /** @var ArticleDefinitions */
    private $articleDefinitions;

    public function calculate(
        DateTimeImmutable $originEndTime,
        DateTimeImmutable $newEndTime
    ): ArticleAmount {
        // @todo - toto by sa zisla dat do nejakeho VO, aby sa takto v sluzbe
        // nepocitalo so scalarnymi cislami
        $diffInSeconds = abs($newEndTime->getTimestamp() - $originEndTime->getTimestamp());
        $minutes = $diffInSeconds / 60;
        $hours = $diffInSeconds / 3600;
        $amountValue = (int)ceil($minutes ** 0.6 + $hours);

        return new ArticleAmount(
            $this->articleDefinitions->getGem(),
            new Amount($amountValue)
        );
    }
}
```

Value Objects

Working With Primitive Values Directly

```php
final class SpeedUpPriceCalculator
{
    /** @var ArticleDefinitions */
    private $articleDefinitions;

    public function calculate(
        DateTimeImmutable $originEndTime,
        DateTimeImmutable $newEndTime
    ): ArticleAmount {
        // @todo - toto by sa zisla dat do nejakeho VO, aby sa takto v sluzbe
        // nepocitalo so scalarnymi cislami
        $diffInSeconds = abs($newEndTime->getTimestamp() - $originEndTime->getTimestamp());
        $minutes = $diffInSeconds / 60;
        $hours = $diffInSeconds / 3600;
        $amountValue = (int)ceil($minutes ** 0.6 + $hours);

        return new ArticleAmount(
            $this->articleDefinitions->getGem(),
            new Amount($amountValue)
        );
    }
}
```

Time difference

Value Objects

Working With Primitive Values Directly

```php
final class SpeedUpPriceCalculator
{
    /** @var ArticleDefinitions */
    private $articleDefinitions;

    public function calculate(
        DateTimeImmutable $originEndTime,
        DateTimeImmutable $newEndTime
    ): ArticleAmount {
        // @todo - toto by sa zisla dat do nejakeho VO, aby sa takto v sluzbe
        // nepocitalo so scalarnymi cislami
        $diffInSeconds = abs($newEndTime->getTimestamp() - $originEndTime->getTimestamp());
        $minutes = $diffInSeconds / 60;
        $hours = $diffInSeconds / 3600;
        $amountValue = (int)ceil($minutes ** 0.6 + $hours);

        return new ArticleAmount(
            $this->articleDefinitions->getGem(),
            new Amount($amountValue)
        );
    }
}
```

Minutes from seconds

Value Objects

Working With Primitive Values Directly

```php
final class SpeedUpPriceCalculator
{
    /** @var ArticleDefinitions */
    private $articleDefinitions;

    public function calculate(
        DateTimeImmutable $originEndTime,
        DateTimeImmutable $newEndTime
    ): ArticleAmount {
        // @todo - toto by sa zisla dat do nejakeho VO, aby sa takto v sluzbe
        // nepocitalo so scalarnymi cislami
        $diffInSeconds = abs($newEndTime->getTimestamp() - $originEndTime->getTimestamp());
        $minutes = $diffInSeconds / 60;
        $hours = $diffInSeconds / 3600;
        $amountValue = (int)ceil($minutes ** 0.6 + $hours);

        return new ArticleAmount(
            $this->articleDefinitions->getGem(),
            new Amount($amountValue)
        );
    }
}
```

Hours from seconds

Value Objects

Working With Primitive Values Directly

```php
final class SpeedUpPriceCalculator
{
    /** @var ArticleDefinitions */
    private $articleDefinitions;

    public function calculate(
        DateTimeImmutable $originEndTime,
        DateTimeImmutable $newEndTime
    ): ArticleAmount {
        // @todo - toto by sa zisla dat do nejakeho VO, aby sa takto v sluzbe
        // nepocitalo so scalarnymi cislami
        $diffInSeconds = abs($newEndTime->getTimestamp() - $originEndTime->getTimestamp());
        $minutes = $diffInSeconds / 60;
        $hours = $diffInSeconds / 3600;
        $amountValue = (int)ceil($minutes ** 0.6 + $hours);

        return new ArticleAmount(
            $this->articleDefinitions->getGem(),
            new Amount($amountValue)
        );
    }
}
```

SpeedUp calculation

Value Objects

Working With Primitive Values Directly

```php
final class SpeedUpPriceCalculator
{
    /** @var ArticleDefinitions */
    private $articleDefinitions;

    public function calculate(
        DateTimeImmutable $originEndTime,
        DateTimeImmutable $newEndTime
    ): ArticleAmount {
        // @todo - toto by sa zisla dat do nejakeho VO, aby sa takto v sluzbe
        // nepocitalo so scalarnymi cislami
        $diffInSeconds = abs($newEndTime->getTimestamp() - $originEndTime->getTimestamp());
        $minutes = $diffInSeconds / 60;
        $hours = $diffInSeconds / 3600;
        $amountValue = (int)ceil($minutes ** 0.6 + $hours);

        return new ArticleAmount(
            $this->articleDefinitions->getGem(),
            new Amount($amountValue)
        );
    }
}
```

Final result

Value Objects

Working With Primitive Values Directly - Concept Extraction

```php
final class SpeedUpPriceCalculator
{
    /** @var ArticleDefinitions */
    private $articleDefinitions;

    public function calculate(
        DateTimeImmutable $originEndTime,
        DateTimeImmutable $newEndTime
    ): ArticleAmount {
        $diffInSeconds = abs($newEndTime->getTimestamp() - $originEndTime->getTimestamp());

        $minutes = Minutes::fromSeconds($diffInSeconds);
        $hours = Hours::fromSeconds($diffInSeconds);

        $amountValue = (int)ceil($minutes->getValue() ** 0.6 + $hours->getValue());

        return new ArticleAmount(
            $this->articleDefinitions->getGem(),
            new Amount($amountValue)
        );
    }
}
```

Minutes and Hours

Value Objects

Working With Primitive Values Directly - Concept Extraction

```php
final class SpeedUpPriceCalculator
{
    /** @var ArticleDefinitions */
    private $articleDefinitions;

    public function calculate(
        DateTimeImmutable $originEndTime,
        DateTimeImmutable $newEndTime
    ): ArticleAmount {
        $diffInSeconds = abs($newEndTime->getTimestamp() - $originEndTime->getTimestamp());

        $minutes = Minutes::fromSeconds($diffInSeconds);
        $hours = Hours::fromSeconds($diffInSeconds);

        $amountValue = (int)ceil($minutes->getValue() ** 0.6 + $hours->getValue());

        return new ArticleAmount(
            $this->articleDefinitions->getGem(),
            new Amount($amountValue)
        );
    }
}
```

Primitive values access

Value Objects

Using Value Objects (Number Pattern)

```php
abstract class Number
{
    /** @var int|float */
    private $value;

    /** @param float|int $value */
    public function __construct($value)
    {
        $this->value = $value;
    }

    /** @return float|int */
    protected function getValue()
    {
        return $this->value;
    }
}
```

```php
abstract class WholeNumber extends Number
{
    /** @param integer $number */
    public function __construct(int $number)
    {
        parent::__construct($number);
    }

    /** @return integer */
    protected function getValue(): int
    {
        return (int)parent::getValue();
    }
}
```

Value Objects

Using Value Objects Using The Number Pattern (Version 1)

```php
final class SpeedUpPrice extends WholeNumber {

    public function __construct(
        DateTimeImmutable $originalEndTime,
        DateTimeImmutable $newEndTime
    ) {
        $diffInSeconds = abs($newEndTime->getTimestamp() - $originalEndTime->getTimestamp());
        $minutes = Minutes::fromSeconds($diffInSeconds); // extends WholeNumber
        $hours = Hours::fromSeconds($diffInSeconds); // extends WholeNumber

        $speedUpAmountValue = (int)ceil($minutes->getValue() ** 0.6 + $hours->getValue());

        parent::__construct($speedUpAmountValue);
    }

    public function toAmount(): Amount {
        return new Amount($this->getValue());
    }
}
```

Value Objects

Using Value Objects Using The Number Pattern (Version 1)

```php
final class SpeedUpPrice extends WholeNumber {

    public function __construct(
        DateTimeImmutable $originalEndTime,
        DateTimeImmutable $newEndTime
    ) {
        $diffInSeconds = abs($newEndTime->getTimestamp() - $originalEndTime->getTimestamp());
        $minutes = Minutes::fromSeconds($diffInSeconds); // extends WholeNumber
        $hours = Hours::fromSeconds($diffInSeconds); // extends WholeNumber

        $speedUpAmountValue = (int)ceil($minutes->getValue() ** 0.6 + $hours->getValue());

        parent::__construct($speedUpAmountValue);
    }

    public function toAmount(): Amount {
        return new Amount($this->getValue());
    }
}
```

BUT!

Two concepts combined in one place

Value Objects

Using Value Objects Using The Number Pattern (Version 2a)

```php
final class TimeDifference extends WholeNumber {

    public function __construct(
        DateTimeImmutable $startTime,
        DateTimeImmutable $endTime
    ) {
        if ($startTime > $endTime) {
            throw new DomainException('End time needs to occur later than start time.');
        }

        $diffInSeconds = $endTime->getTimestamp() - $startTime->getTimestamp();
        parent::__construct($diffInSeconds);
    }

    public function getMinutes(): Minutes {
        return Minutes::fromSeconds($this->getValue());
    }

    public function getHours(): Hours {
        return Hours::fromSeconds($this->getValue());
    }
}
```

Extracted concept
(time difference)

Value Objects

Using Value Objects Using The Number Pattern (Version 2b)

```php
final class SpeedUpPrice extends WholeNumber {

    private function __construct(int $speedUpPrice) {
        parent::__construct($speedUpPrice);
    }

    public static function fromTimeDifference(TimeDifference $timeDiff): self {
        $speedUpAmountValue = (int)ceil(
            $timeDiff->getMinutes()->getValue() ** 0.6 + $timeDiff->getHours()->getValue()
        );

        return new self($speedUpAmountValue);
    }

    public function toAmount(): Amount {
        return new Amount($this->getValue());
    }
}
```

Extracted concept (Speedup price from Time difference)

Value Objects

Using Value Objects In The Calculator Service

```
final class SpeedUpPriceCalculator {
    /**
     * @param DateTimeImmutable $originEndTime
     * @param DateTimeImmutable $newEndTime
     * @return ArticleAmount
     */
    public function calculate(
        DateTimeImmutable $originalEndTime,
        DateTimeImmutable $newEndTime
    ): ArticleAmount {
        $timeDiff = new TimeDifference($originalEndTime, $newEndTime);
        $speedUpPrice = SpeedUpPrice::fromTimeDifference($timeDiff);

        return new ArticleAmount(
            $this->articleDefinitions->getGem(),
            $speedUpPrice->toAmount()
        );
    }
}
```

Readability bonus!

Value Objects

Using Value Objects In The Calculator Service

```
final class SpeedUpPriceCalculator {
    /**
     * @param DateTimeImmutable $originEndTime
     * @param DateTimeImmutable $newEndTime
     * @return ArticleAmount
     */
    public function calculate(
        DateTimeImmutable $originalEndTime,
        DateTimeImmutable $newEndTime
    ): ArticleAmount {
        $timeDiff = new TimeDifference($originalEndTime, $newEndTime);
        $speedUpPrice = SpeedUpPrice::fromTimeDifference($timeDiff);

        return new ArticleAmount(
            $this->articleDefinitions->getGem(),
            $speedUpPrice->toAmount()
        );
    }
}
```

Value Objects

We Use It Everywhere In The Domain

Value Objects

Using Value Objects In The Calculator Service

```
final class SpeedUpPriceCalculator {
    /**
     * @param DateTimeImmutable $originEndTime
     * @param DateTimeImmutable $newEndTime
     * @return ArticleAmount
     */
    public function calculate(
        DateTimeImmutable $originalEndTime,
        DateTimeImmutable $newEndTime
    ): ArticleAmount {
        $timeDiff = new TimeDifference($originalEndTime, $newEndTime);
        $speedUpPrice = SpeedUpPrice::fromTimeDifference($timeDiff);

        return new ArticleAmount(
            $this->articleDefinitions->getGem(),
            $speedUpPrice->toAmount()
        );
    }
}
```

Value objects repository

Value Objects

Persistent Value Objects (Dynamically Generated Definitions)

- Immutable (or even read only)
- Identifier doesn’t exist in the domain layer
- Identifier is only visible in the infrastructure layer

Value Objects

Persistent Value Objects - ArticleDefinition Example

```
class ArticleDefinition {

    /** @var Name */
    private $name;

    /** @var Level */
    private $minLevel;

    /** @var Score */
    private $score;

    public function isEqual(ArticleDefinition $articleDefinition): bool {
        return $this->name->isEqual($articleDefinition->name);
    }

    public function getMinLevel(): Level {
        return $this->minLevel;
    }

    public function getScore(): Score {
        return $this->score;
    }
}
```

Hidden id (not exposed externally)

Value Objects

Persistent Value Objects - ArticleDefinition Repository

```
// Domain
interface ArticleDefinitions {
    public function getGem(): ArticleDefinition;
}

// Infrastructure
class DoctrineArticleDefinitions implements ArticleDefinitions {

    private const NAME_GEM = 'Gem';
    // ...

    public function getGem(): ArticleDefinition
    {
        return $this->entityManager->findBy(['name' => self::NAME_GEM]);
    }
}
```

Modeled as collection pattern - set

Value Objects As Entity Identifiers

Entity Identifier Possibilities

- Primitive type
  - string
  - integer
- UUID
- Value object

Value Objects As Entity Identifiers

Reasons

- Value object can wrap all of the previous types, which means that:
  - Identifiers become infrastructure agnostic
  - It's possible to change the underlying identifier types without changing the model

Value Objects As Entity Identifiers

Identifier Example

```
// domain
final class TrainId {
    /** @var PlayerId */
    private $playerId;

    /** @var SequenceId */
    private $sequenceId;

    public function __construct(PlayerId $playerId, SequenceId $sequenceId) {...}

    public function getPlayerId(): PlayerId {}

    public function getSequenceId(): SequenceId {}
}
```

Hidden primitive types

Value Objects As Entity Identifiers

Repository Example

```php
// Domain
interface Trains {
    public function find(TrainId $trainId): ?Train;
}

// Infrastructure
final class DoctrineTrains {
    public function find(TrainId $trainId): ?Train {
        return $this->repository->findOneBy([
            'playerId' => WholeNumberExtractor::extract($trainId->getPlayerId()),
            'sequenceId' => WholeNumberExtractor::extract($trainId->getSequenceId()),
        ]);
    }
}
```

Value Objects As Entity Identifiers

Command Handler Example

```
final class CreateTrainHandler {

    public function __invoke(CreateTrain $createTrain): void {
        $trainId = new TrainId(
                $createTrain->getPlayerId(),
                $createTrain->getTrainSeqenceId()
            );
        $trainDefinition = $createTrain->getTrainDefinition();

        $train = new Train($trainId, $trainDefinition);
        $this->trains->save($train);
    }
}
```

Id created directly, independently of db structure

Value Objects As Entity Identifiers

But Is The Identifier Really Independent?

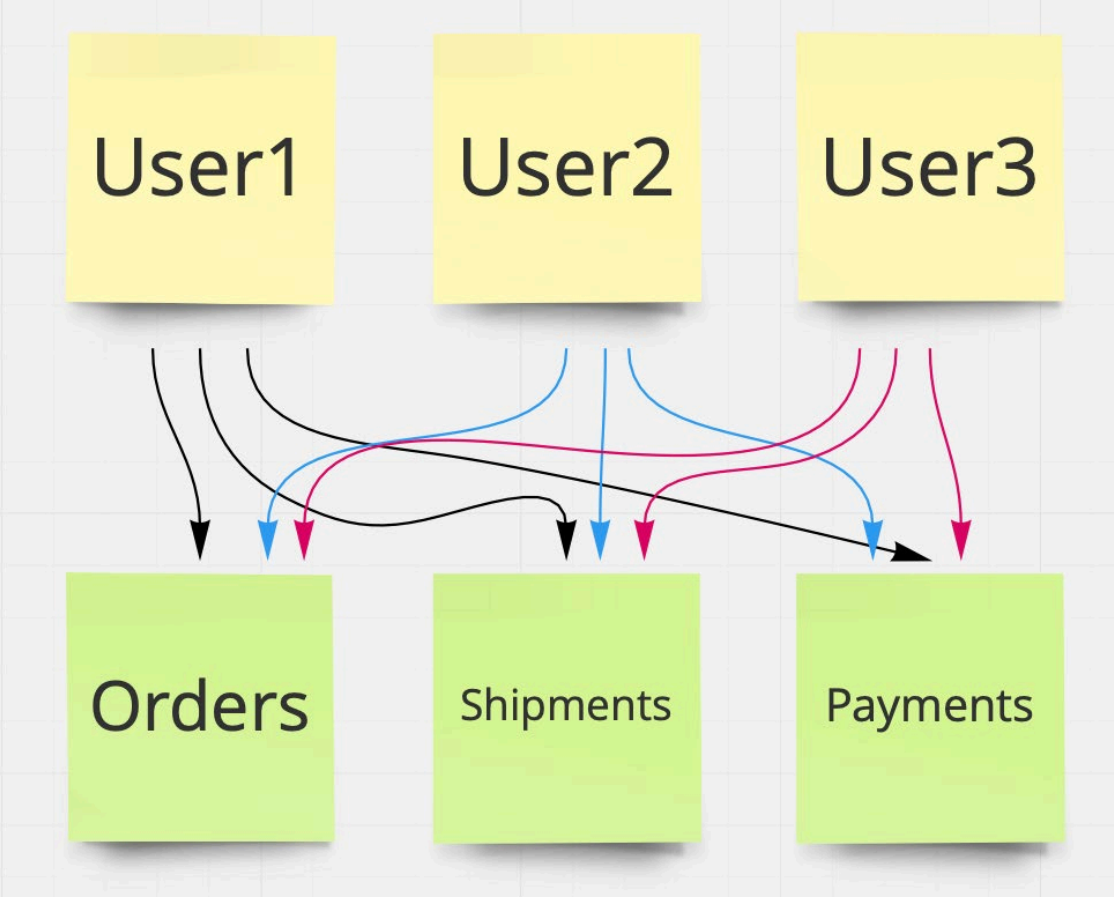
This is the `train` table:

player_id	sequence_id	level	definition_id
1	1	3	8
1	2	17	2
1	4	1	10

Is this structure significant for the domain?

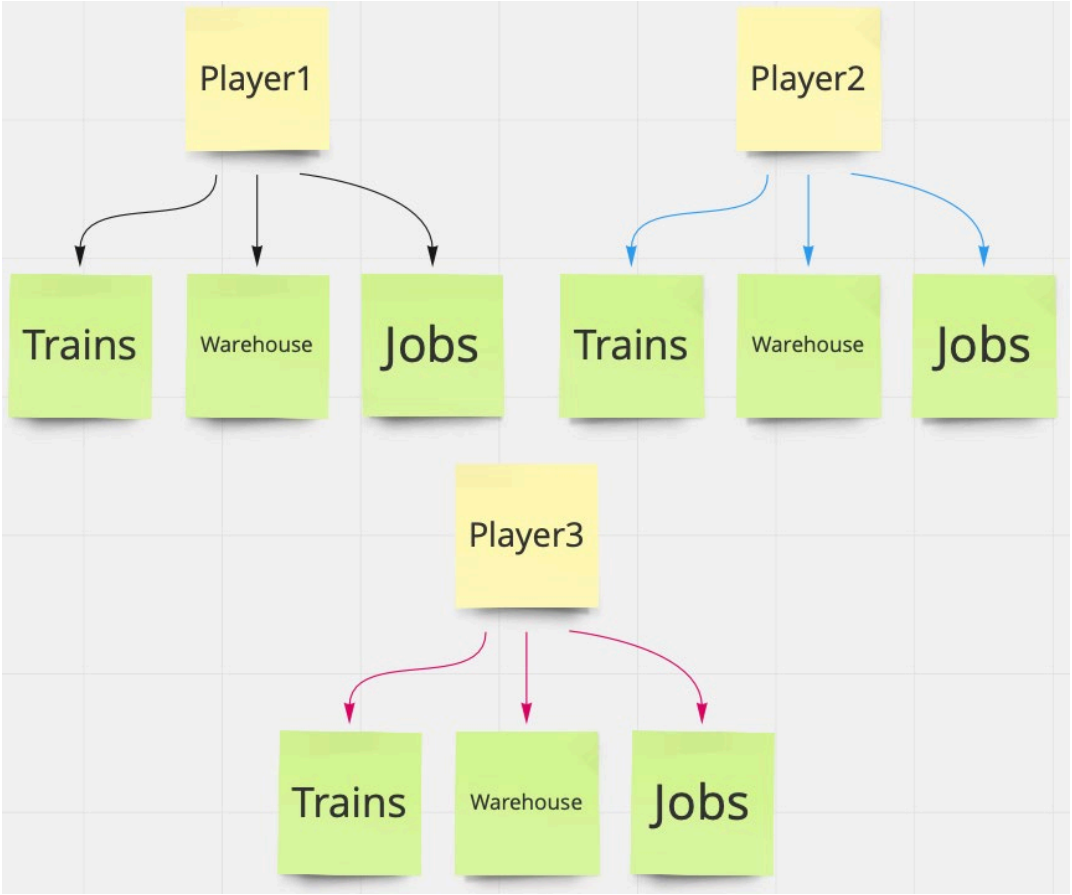
Value Objects As Entity Identifiers

Casual Domain Example (E-Shop)



Value Objects As Entity Identifiers

Game Domain Example (Player World)



Value Objects As Entity Identifiers

Removing The Identifier Dependence On Infrastructure

- **Pros**:
  - The domain is simpler (clearer - it's player's world with player abstracted as a 'director')
  - The model is more suited to be shared between client and server
- **Cons**:
  - Identifier instantiation becomes tightly coupled to repositories

Value Objects As Entity Identifiers

Identifier Example - Hidden Database Structure

```php
// Domain
interface TrainId {}

// Infrastructure
class DoctrineTraintId implements TrainId {
    /** @var PlayerId */
    private $playerId;

    /** @var SequenceId */
    private $sequenceId;

    public function __construct(PlayerId $playerId, SequenceId $sequenceId) {...}

    public function getPlayerId(): PlayerId {...}

    public function getSequenceId(): SequenceId {...}
}
```

Value Objects As Entity Identifiers

Repository Example - Identifier Coupling

```php
// Domain
interface Trains {

    public function newIdentifier(): TrainId;

    public function find(TrainId $trainId): ?Train;
}

// Infrastructure
final class DoctrineTrains {

    public function newIdentifier(): TrainId {
        return new DoctrineTrainId(
            $this->playerIdProvider->provide(),
            new SequenceId($this->sequencer->next())
        );
    }

    public function find(TrainId $trainId): ?Train {
        /* @var $trainId DoctrineTrainId */
        return $this->repository->findOneBy([
            'playerId' => WholeNumberExtractor::extract($trainId->getPlayerId()),
            'sequenceId' => WholeNumberExtractor::extract($trainId->getSequenceId()),
        ]);
    }
}
```

Single place for identifier creation

Value Objects As Entity Identifiers

Command Handler Example

```
final class CreateTrainHandler {

    public function __invoke(CreateTrain $createTrain): void {
        // id factory coupled to repository
        $trainId = $this->trains->newIdentifier();
        $trainDefinition = $createTrain->getTrainDefinition();

        $train = new Train($trainId, $trainDefinition);
        $this->trains->save($train);
    }
}
```

Db structure is hidden

Value Objects As Tactical Pattern

Summary

- Value object is a very powerful concept:
  - Always valid
  - Properly named
  - Strongly typed on domain level

Value Objects As Tactical Pattern

Practical Benefits

- They improve the implementation and makes it more transparent
- They make the design/implementation easily understandable
    - In theory even for less technical people

Value Objects As Tactical Pattern

What We Observed In Pixel

- Mastering value objects takes time, but makes better programmers and code as a result
- We started to use value objects even in infrastructure layer and in libraries
- Naming things is really really hard :)

PIXELFEDERATION

www.pixelfederation.com

Want To Play At Work With Us?

www.pixelfederation.com

Thank you