BETTER ENTITIES

Getting more from your Domain Objects

a.k.a how to create Doctrine entities that don't suck!

RE-CAP…

- Presenter notes are on the slides
- What is an Entity?
- Why use entities?
- What is the role of an Entity?
- DDD FTW!

```php
/**
 * @ORM\Entity
 */
class Post
{
    const NUM_ITEMS = 10;

    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string")
     */
    private $title;

    public function __construct()
    {
        $this->publishedAt = new \DateTime();
        $this->comments    = new ArrayCollection();
    }

    public function setTitle($title)
    {
        $this->title = $title;

        return $this;
    }

    public function getSlug()
    {
        return $this->slug;
    }

    public function setSlug($slug)
    {
        $this->slug = $slug;

        return $this;
    }
}
```

TYPICAL ENTITY

- Protected or private properties
- Getters and setters
- Annotations
- Mostly empty constructor
- Life-cycle methods
- Many Symfony and Doctrine examples

```php
namespace AppBundle\Entity;

use Doctrine\ORM\Mapping as ORM;
use Doctrine\Common\Collections\ArrayCollection;

/**
 * @ORM\Entity
 */
class Post
{
    const NUM_ITEMS = 10;

    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string")
     */
    private $title;

    /**
     * @ORM\Column(type="string")
     */
    private $slug;

    /**
     * @ORM\Column(type="text")
     */
    private $content;

    /**
     * @ORM\Column(type="string")
     */
    private $authorEmail;

    /**
     * @ORM\Column(type="datetime")
     */
    private $publishedAt;
```

PROBLEMS…

- Purpose
- Validity
- Mutability
- Annotations
- Persistence knowledge
- We can do better!

```php
namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable
{

    use Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name', 'email', 'password',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password', 'remember_token',
    ];
}
```

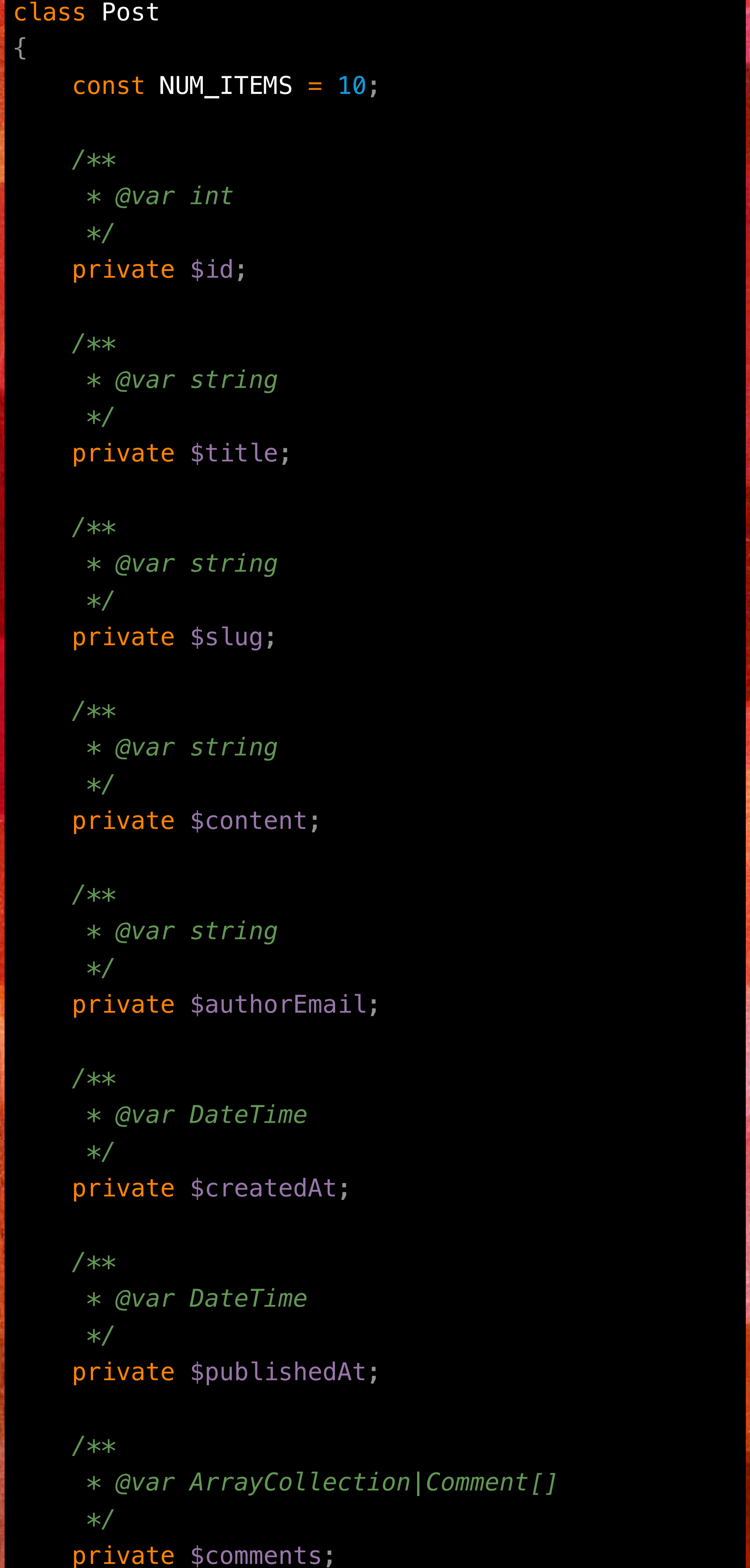
TYPICAL AR MODEL

- No clear API
- Lots of “magic”
- Object reflects CURRENT table design
- Attributes can be overloaded
- Difficult to use Domain language
- Difficult to ensure state

LETS MAKE IT BETTER!

And make a bunch of contentious statements…

ANNOTATIONS



```
class Post
{
    const NUM_ITEMS = 10;

    /**
     * @var int
     */
    private $id;

    /**
     * @var string
     */
    private $title;

    /**
     * @var string
     */
    private $slug;

    /**
     * @var string
     */
    private $content;

    /**
     * @var string
     */
    private $authorEmail;

    /**
     * @var DateTime
     */
    private $createdAt;

    /**
     * @var DateTime
     */
    private $publishedAt;

    /**
     * @var ArrayCollection|Comment[]
     */
    private $comments;
```

- Bad:
  - Annotations are not code!
  - Tightly couple framework
  - Decreases readability
  - Another config language to learn
- Instead:
  - Use config files
  - Free annotations
  - Easier to convey domain

```php
class Post
{
    const NUM_ITEMS = 10;

    /**
     * @var int
     */
    private $id;

    /**
     * @var string
     */
    private $title;

    /**
     * @var string
     */
    private $slug;

    /**
     * @var string
     */
    private $content;

    /**
     * @var string
     */
    private $authorEmail;

    /**
     * @var DateTime
     */
    private $createdAt;

    /**
     * @var DateTime
     */
    private $publishedAt;

    /**
     * @var ArrayCollection|Comment[]
     */
    private $comments;
```

SETTERS

- Remove ALL the setters
    - Need to control state*
    - No arbitrary changes
    - No partial changes
    - Cleaner interface

** state changes are coming up!*

```php
class Post
{



    . . .

    private function __construct($title, $slug, $content)
    {
        $this->title   = $title;
        $this->slug    = $slug;
        $this->content = $content;
    }

    public static function create($title, $slug, $content)
    {
        $entity = new static($title, $slug, $content);

        return $entity;
    }

    public static function createAndPublish($title, $slug, $content)
    {
        $entity = static::create($title, $slug, $content);
        $entity->publish();

        return $entity;
    }

}
```

INSTANTIATION

- Define required properties
- Enforce factory methods
- Name methods after domain terms
- Decide on identity scheme
  - Do you need identity now?

```php
class Post
{

    . . .

    public function publish(DateTime $publishedAt = null)
    {
        $this->publishedAt = ($publishedAt ?: new DateTime());
    }

    public function removeFromPublication()
    {
        $this->publishedAt = null;
    }

    public function changeTitleAndSlug(string $title, string $slug)
    {
        $this->title = $title;
        $this->slug = $slug;
    }

    public function replaceContentWith(string $content)
    {
        $this->content = $content;
    }

}
```

ENFORCE STATE CHANGES

- Define explicit methods
- Require ALL arguments
  - Enforce types
- Make methods statements
  - no return value
  - use the domain language

```
class UserAddress
{

    protected $type;

    public function __construct(Address $address, AddressType $type)
    {
        $this->address = $address;
        $this->type    = $type;
    }
}

/**
 * Class AddressType
 *
 * @method static AddressType HOME_ADDRESS()
 * @method static AddressType WORK_ADDRESS()
 */
final class AddressType extends AbstractEnumeration
{

    const HOME_ADDRESS  = 'home';
    const WORK_ADDRESS  = ‘work';

}
```

STRING CONSTANTS

- Use Enumerations
- Type hint class type
- Enumeration is already valid
- Works with refactoring tools
- Use domain language

```
class Post
{
    private function __construct(string $title, string $slug, string
$content)
    {
        Assert::lazy()->tryAll()
            ->that($title, 'title')->notEmpty()->maxLength(100)
            ->that($slug, 'slug')->notEmpty()->maxLength(64)
            ->that($content, ‘content')
                ->notEmpty()->minLength(100)->maxLength(65000)
            ->verifyNow()
        ;

        $this->title   = $title;
        $this->slug    = $slug;
        $this->content = $content;
    }

    public function publish(DateTime $publishedAt = null)
    {
        $this->publishedAt = ($publishedAt ?: new DateTime());
    }


    public function removeFromPublication()
    {
        $this->publishedAt = null;
    }

    public function changeTitleAndSlug(string $title, string $slug)
    {
        Assert::lazy()->tryAll()
            ->that($title, 'title')->notEmpty()->maxLength(100)
            ->that($slug, 'slug')->notEmpty()->maxLength(64)
            ->verifyNow()
        ;

        $this->title = $title;
        $this->slug = $slug;
    }
}
```

VALIDATION

- Core concept of the entity
- Always check data
- Use type hints
- Use scalar type hints
- Use Exceptions
- Change assertions when needed
- Avoid framework dependencies
- Entity validity !== Application validity

```
class EnquiryFormRequest extends AppRequest
{

    /**
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * @return array
     */
    public function rules()
    {
      return [
        'email' => 'email|max:255|required_without_all:phone_number',
        'phone_number' => 'numeric|required_without_all:email',
        'name'          => 'nullable|max:255',
        'notes'         => 'nullable|max:10000',
      ];
    }

    /**
     * @return array
     */
    public function messages()
    {
      return [
       'email.required_without_all' => ‘Required if no phone',
       'phone_number.required_without_all' => ‘Required if no email',
      ];
    }
}
```

APPLICATION VALIDATION

- Use framework validators
- Validate to application specs
- Enforce access checks
- Enforce uniqueness
- Enforce complex rules
- Transform data for the domain

VALUE OBJECTS

Better separation of responsibilities

```
class EmailAddress
{

    private $value;

    public function __construct($value)
    {
        Assert::that($value, null, ‘email')
            ->email()->notEmpty()->maxLength(100);

        $this->value = $value;
    }

    public function __set($name, $value)
    {
        // don't allow setting anything
    }

    public function __toString()
    {
        return $this->toString();
    }

    public function toString()
    {
        return (string)$this->value;
    }

    public function equals($test)
    {
        if (__CLASS__ === get_class($test)) {
            return ((string)$test === (string)$this);
        }

        return false;
    }
}
```

VALUE OBJECTS (VO)

- ➤ Object has identity through its properties
- ➤ Compare based on properties
- ➤ Immutable
- ➤ Move validation to VO
- ➤ Add domain logic to VO
- ➤ Group related properties

```php
class EmailAddress
{

    private $value;

    public function __construct($value)
    {
        Assert::that($value, null, 'email')
            ->email()->notEmpty()->maxLength(100);

        $this->value = $value;
    }

    public function __set($name, $value)
    {
        // don't allow setting anything
    }

    public function __toString()
    {
        return $this->toString();
    }

    public function toString()
    {
        return (string)$this->value;
    }

    public function equals($test)
    {
        if (__CLASS__ === get_class($test)) {
            return ((string)$test === (string)$this);
        }

        return false;
    }
}
```

VO DANGERS!

- VOs are part of your domain
  - Use domain language
- VOs should be immutable
  - Don't pass in entities
  - Don't change state
- Don't share between domains
- Don't share between projects
  - Username is not always the same thing
  - Validations can change

```php
class PostAuthor
{
    private $name;
    private $email;

    public function __construct(string $name, EmailAddress $email)
    {
        Assert::that($name, null, ‘name')
            ->notEmpty()->maxLength(100);

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

    public function __toString()
    {
        return $this->toString();
    }

    public function toString()
    {
        return (string)$this->name;
    }

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

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

    public function equals($test)
    {
        if (__CLASS__ === get_class($test)) {
            return (
                $this->name === $test->name() &&
                $this->email->equals($test->email())
            );
        }

        return false;
    }
}
```

POST AUTHOR

- Replace AuthorEmail with VO
- VO reflects domain language
- VO can limit data access
- Enforce validity
  - Consistent validation
  - Simplifies entity

```php
class Post
{

    private function __construct(PostAuthor $author, string $title,
string $slug, string $content)
    {
        Assert::lazy()->tryAll()
            ->that($title, 'title')->notEmpty()->maxLength(100)
            ->that($slug, 'slug')->notEmpty()->maxLength(64)
            ->that($content, ‘content')
                ->notEmpty()->minLength(100)->maxLength(65000)
            ->verifyNow()
        ;

        $this->title   = $title;
        $this->slug    = $slug;
        $this->content = $content;
    }

    public static function create(PostAuthor $author, string $title,
string $slug, string $content)
    {
        $entity = new static($author, $title, $slug, $content);

        return $entity;
    }

    public static function createAndPublish(PostAuthor $author,
string $title, string $slug, string $content)
    {
        $entity = new static($author, $title, $slug, $content);
        $entity->publish();

        return $entity;
    }

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

}
```

USING POST AUTHOR

- Author now required
    - Enforce in methods
- Conveys domain language
- Change accessor name
    - post->author->name
    - post->author->email

```php
class PostContent
{
    private $content;

    public function __construct($content)
    {
        Assert::that($content, null, 'content')
            ->notEmpty()->minLength(100)->maxLength(65000)
        ;

        $this->content = $content;
    }

    public function __toString()
    {
        return $this->toString();
    }

    public function toString()
    {
        return (string)$this->content;
    }

    public function html()
    {
        return $this->content;
    }

    public function text()
    {
        return strip_tags($this->html());
    }

    public function summary($length = 40)
    {
        return implode(' ', array_slice(explode(' ', $this->text()),
0, $length)) . '...';
    }

    public function equals($test)
    {
        if (__CLASS__ === get_class($test)) {
            return ((string)$test === (string)$this);
        }

        return false;
    }
}
```

MORE VALUE OBJECTS

- What about other properties?
- Title
  - Group title and slug
  - Protect slug from user
- Content
  - Encapsulate the body
  - Add transforms e.g.:
    - summary
    - text (plain text)

TOWARDS A CLEAN API

Steps to make our Entity even better

```
class Post
{

    public function author(): PostAuthor
    {
        return $this->author;
    }

    public function title(): PostTitle
    {
        return $this->title;
    }

    public function content(): PostContent
    {
        return $this->content;
    }

    public function createdAt(): DateTimeImmutable
    {
        return $this->createdAt;
    }

    public function publishedAt(): ?DateTimeImmutable
    {
        return $this->publishedAt;
    }

    public function isPublished(): bool
    {
        return $this->publishedAt instanceof DateTimeImmutable;
    }

    public function isRecentlyPublished(): bool
    {
        return $this->isPublished() && $this->publishedAt->diff(new 
DateTimeImmutable())->days < 10;
    }
}
```

BETTER DATA METHODS

- Drop “get” prefixes
- Use domain language
- Return useful data
- Add state methods
  - isPublished
  - isRecentlyPublished
  - hasComments
  - leaveComment

```
class Post
{


    public function comments()
    {
        return new ArrayCollection($this->comments->toArray());
    }

    public function reverseCommentOrder()
    {
        return new ArrayCollection(
            array_reverse($this->comments->toArray())
        );
    }

}
```

SPECIAL CASE

- Internal collections
- Mutable
- Don’t expose mutability
- Wrap collections
  - Very important with Doctrine
  - Don’t allow changes outside of the entity

BROADCASTING CHANGES

De-coupling processing

```
class Post
{
    private $events = [];

    public static function create(PostAuthor $author, PostTitle
$title, PostContent $content)
    {
        $entity = new static($author, $title, $content);
        $entity->raise(new PostCreated([
            'author' => $author,
            'title' => $title,
            'summary' => $content->summary(),
            'created_at' => $entity->createdAt(),
        ]));

        return $entity;
    }

    protected function raise(Event $event)
    {
        $this->events[] = $event;
    }

    public function releaseEvents()
    {
        $events = $this->events;

        $this->events = [];

        return $events;
    }
}
```

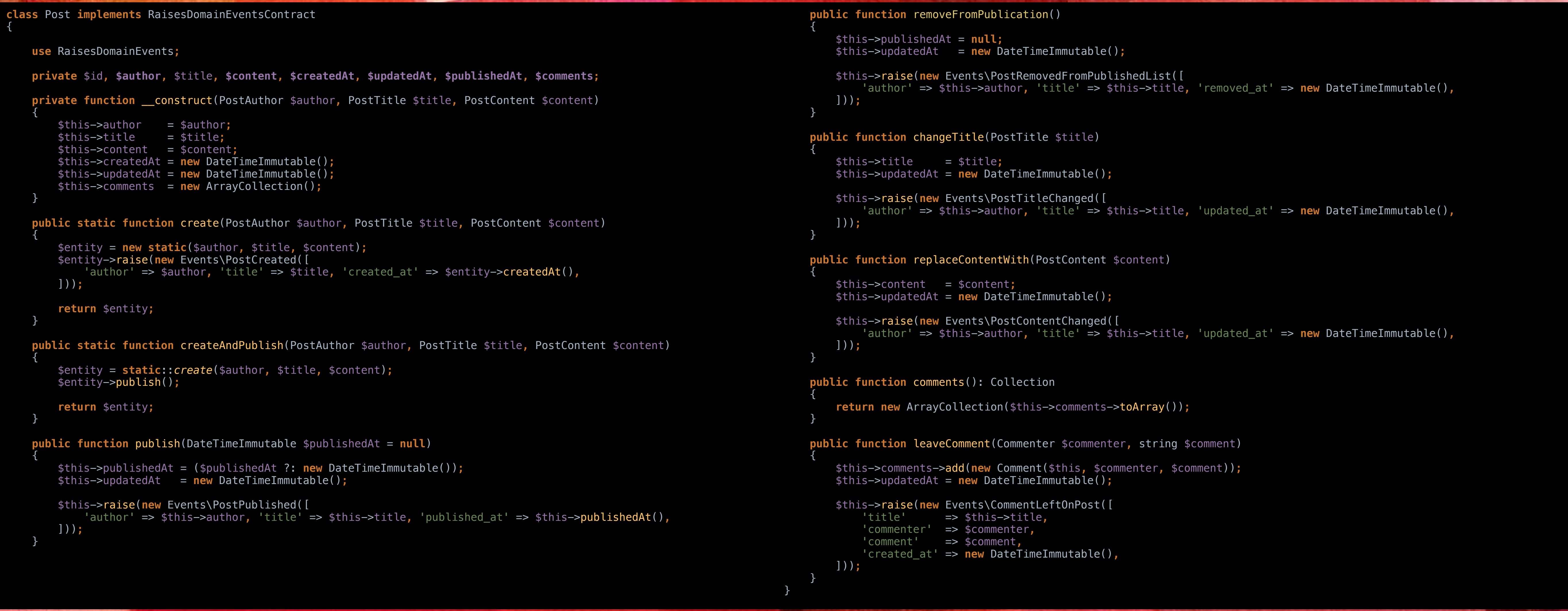
DOMAIN EVENTS

- Reflect important changes
- Propagate those changes
- Separate responsibilities
- Cross process data sharing
  - update search indexes
  - send messages
  - more domain changes
- First step to Event Sourcing

```php
class DomainEventPublisher implements EventSubscriber
{

    private $entities;

    // abbreviated code…
    public function __construct()
    {
        $this->entities = new Collection();
    }

    public function getSubscribedEvents()
    {
        return [
            Events::prePersist, Events::preFlush, Events::postFlush
        ];
    }

    public function prePersist(LifecycleEventArgs $event)
    {
        $entity = $event->getEntity();

        if ($entity instanceof RaisesDomainEvents) {
            $this->entities->add($entity);
        }
    }

    public function preFlush(PreFlushEventArgs $event)
    {
        $uow = $event->getEntityManager()->getUnitOfWork();

        foreach ($uow->getIdentityMap() as $class => $entities) {
            foreach ($entities as $entity) {
                $this->entities->add($entity);
            }
        }
    }

    public function postFlush(PostFlushEventArgs $event)
    {
       // dispatch events, add other channels here
       $events->call(function ($event) use ($em, $evm) {
            $evm->dispatchEvent('on' . $event->name(), $event);
        });

        $this->entities->reset();
    }
}
```

DISPATCHING EVENTS

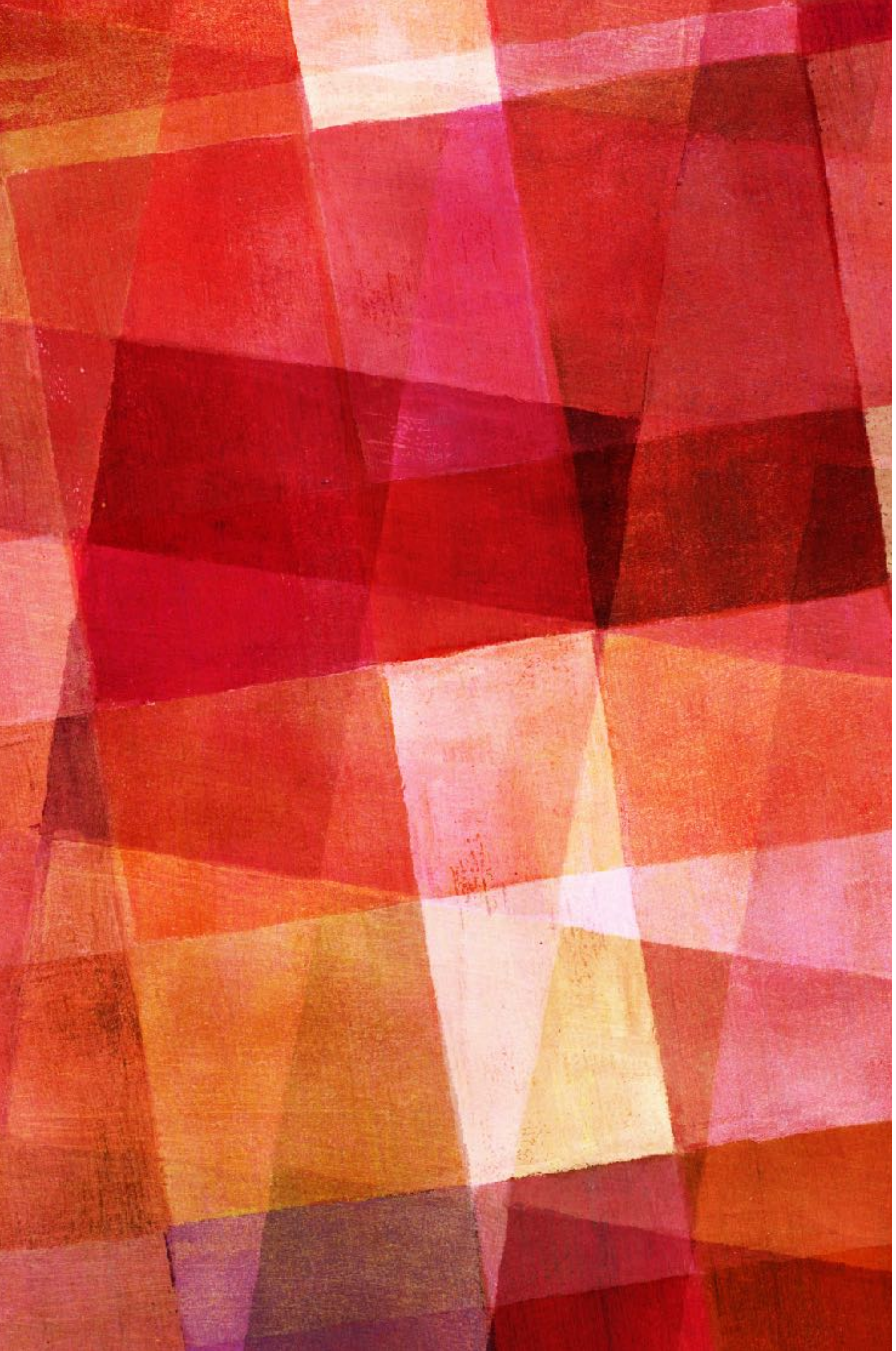
- Use framework dispatcher
- Can use Doctrine
  - Listen for:
    - prePersist
    - preFlush
    - postFlush
  - Publish events post flush
- Can broadcast via RabbitMQ



```php
class Post implements RaisesDomainEventsContract
{

    use RaisesDomainEvents;

    private $id, $author, $title, $content, $createdAt, $updatedAt, $publishedAt, $comments;

    private function __construct(PostAuthor $author, PostTitle $title, PostContent $content)
    {
        $this->author    = $author;
        $this->title     = $title;
        $this->content   = $content;
        $this->createdAt = new DateTimeImmutable();
        $this->updatedAt = new DateTimeImmutable();
        $this->comments  = new ArrayCollection();
    }

    public static function create(PostAuthor $author, PostTitle $title, PostContent $content)
    {
        $entity = new static($author, $title, $content);
        $entity->raise(new Events\PostCreated([
            'author' => $author, 'title' => $title, 'created_at' => $entity->createdAt(),
        ]));

        return $entity;
    }

    public static function createAndPublish(PostAuthor $author, PostTitle $title, PostContent $content)
    {
        $entity = static::create($author, $title, $content);
        $entity->publish();

        return $entity;
    }

    public function publish(DateTimeImmutable $publishedAt = null)
    {
        $this->publishedAt = ($publishedAt ?: new DateTimeImmutable());
        $this->updatedAt   = new DateTimeImmutable();

        $this->raise(new Events\PostPublished([
            'author' => $this->author, 'title' => $this->title, 'published_at' => $this->publishedAt(),
        ]));
    }

    public function removeFromPublication()
    {
        $this->publishedAt = null;
        $this->updatedAt   = new DateTimeImmutable();

        $this->raise(new Events\PostRemovedFromPublishedList([
            'author' => $this->author, 'title' => $this->title, 'removed_at' => new DateTimeImmutable(),
        ]));
    }

    public function changeTitle(PostTitle $title)
    {
        $this->title     = $title;
        $this->updatedAt = new DateTimeImmutable();

        $this->raise(new Events\PostTitleChanged([
            'author' => $this->author, 'title' => $this->title, 'updated_at' => new DateTimeImmutable(),
        ]));
    }

    public function replaceContentWith(PostContent $content)
    {
        $this->content   = $content;
        $this->updatedAt = new DateTimeImmutable();

        $this->raise(new Events\PostContentChanged([
            'author' => $this->author, 'title' => $this->title, 'updated_at' => new DateTimeImmutable(),
        ]));
    }

    public function comments(): Collection
    {
        return new ArrayCollection($this->comments->toArray());
    }

    public function leaveComment(Commenter $commenter, string $comment)
    {
        $this->comments->add(new Comment($this, $commenter, $comment));
        $this->updatedAt = new DateTimeImmutable();

        $this->raise(new Events\CommentLeftOnPost([
            'title'      => $this->title,
            'commenter'  => $commenter,
            'comment'    => $comment,
            'created_at' => new DateTimeImmutable(),
        ]));
    }
}
```

CLEAN, RICH, DOMAIN OBJECTS

(https://github.com/dave-redfern/better-entities/) *The End Result*



MORE…

- Domain Driven Design
  - Eric Evans (blue book)
  - Vernon Vaughn (red book)
- Event Sourcing
  - Greg Young
- Doctrine Project
- Example code available
  - https://github.com/dave-redfern/better-entities