

Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный университет»
Кафедра ИИТ»

Лабораторная работа №5
Тема: «Доменная модель»

Выполнила: студентка
группы ПО-3
Ковалева А.И.
Проверил:
Лаврущик А.И.

Брест 2021

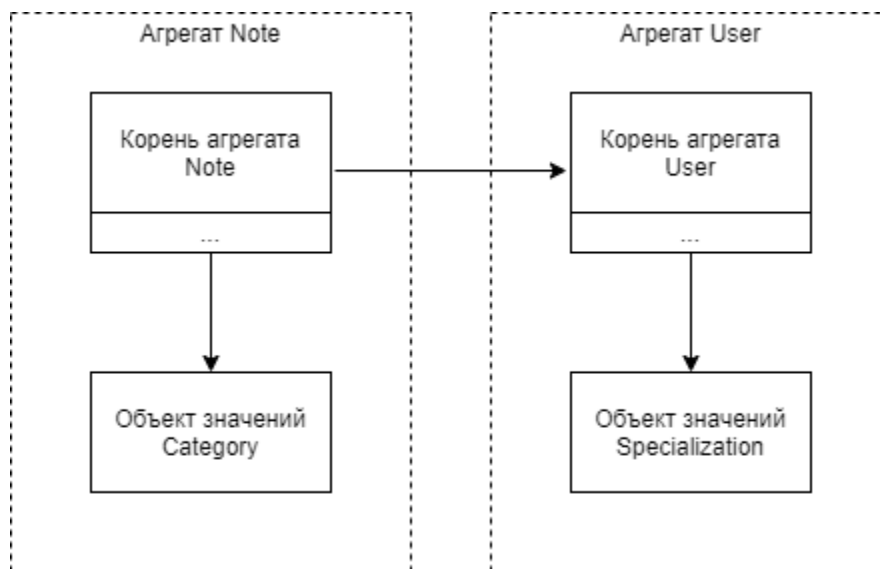
Вариант 12

Цель: Познакомиться с тактическими шаблонами предметно-ориентированного проектирования.

Предметная область: блог о фермерском хозяйстве.

Задание для выполнения: Реализуйте не менее двух агрегатов, сущностей, объектов значений домена имеющейся гексагональной архитектуры из предыдущей работы, а также репозитории и не менее двух доменных событий для агрегатов. Один агрегат должен включать в себя сущность-корень агрегата, список дочерних сущностей, каждая сущность содержать наряду с обычными свойствами ещё и объекты-значения. Идентификаторы сущностей должны генерироваться: для корней агрегатов – репозиторием, для внутренних сущностей – самим корнем агрегата.

Ход работы



Агрегаты:

- Note
- User

Объекты значений:

- Specialization
- Category

Репозитории:

- NoteRepository
- UserRepository
- SpecializationRepository
- NoteChangeEventRepository

Arperat Note:

```
<?php

namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\NoteRepository")
 */
class Note
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=64)
     */
    private $title;

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

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\User", inversedBy="notes")
     */
    private $user;

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\Category", inversedBy="notes")
     * @ORM\JoinColumn(nullable=false)
     */
    private $category;

    public function getId(): ?int
    {
        return $this->id;
    }
}
```

```
public function getTitle(): ?string
{
    return $this->title;
}

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

    return $this;
}

public function getContent(): ?string
{
    return $this->content;
}

public function setContent(string $content): self
{
    $this->content = $content;

    return $this;
}

public function getUser(): ?User
{
    return $this->user;
}

public function setUser(?User $user): self
{
    $this->user = $user;

    return $this;
}

public function getCategory(): ?Category
{
    return $this->category;
}

public function setCategory(?Category $category): self
{
    $this->category = $category;

    return $this;
}
```

```
}
```

Arperat User:

```
<?php

namespace App\Entity;

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

/**
 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 */
class User
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=32)
     */
    private $name;

    /**
     * @ORM\Column(type="string", length=32)
     */
    private $surname;

    /**
     * @ORM\OneToMany(targetEntity="App\Entity>Note", mappedBy="user")
     */
    private $notes;

    /**
     * @ORM\ManyToOne(targetEntity="App\Entity\Specialization", inversedBy="users")
     */
    private $specialization;

    public function __construct()
    {
        $this->notes = new ArrayCollection();
    }
}
```

```
public function getId(): ?int
{
    return $this->id;
}

public function getName(): ?string
{
    return $this->name;
}

public function setName(string $name): self
{
    $this->name = $name;

    return $this;
}

public function getSurname(): ?string
{
    return $this->surname;
}

public function setSurname(string $surname): self
{
    $this->surname = $surname;

    return $this;
}

/**
 * @return Collection|Note[]
 */
public function getNotes(): Collection
{
    return $this->notes;
}

public function addNote(Note $note): self
{
    if (!$this->notes->contains($note)) {
        $this->notes[] = $note;
        $note->setUser($this);
    }

    return $this;
}
```

```
public function removeNote(Note $note): self
{
    if ($this->notes->contains($note)) {
        $this->notes->removeElement($note);
        // set the owning side to null (unless already changed)
        if ($note->getUser() === $this) {
            $note->setUser(null);
        }
    }

    return $this;
}

public function __toString()
{
    return $this->name.' '.$this->surname;
}

public function getSpecialization(): ?Specialization
{
    return $this->specialization;
}

public function setSpecialization(?Specialization $specialization): self
{
    $this->specialization = $specialization;

    return $this;
}
}
```

Объект значений Specialization:

```
<?php

namespace App\Entity;

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

/**
 * @ORM\Entity(repositoryClass="App\Repository\SpecializationRepository")
 */
class Specialization
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=16)
     */
    private $title;

    /**
     * @ORM\OneToMany(targetEntity="App\Entity\User", mappedBy="specialization")
     */
    private $users;

    public function __construct()
    {
        $this->users = new ArrayCollection();
    }

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getTitle(): ?string
    {
        return $this->title;
    }
}
```

```

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

    return $this;
}

/**
 * @return Collection|User[]
 */
public function getUsers(): Collection
{
    return $this->users;
}

public function addUser(User $user): self
{
    if (!$this->users->contains($user)) {
        $this->users[] = $user;
        $user->setSpecialization($this);
    }

    return $this;
}

public function removeUser(User $user): self
{
    if ($this->users->contains($user)) {
        $this->users->removeElement($user);
        // set the owning side to null (unless already changed)
        if ($user->getSpecialization() === $this) {
            $user->setSpecialization(null);
        }
    }

    return $this;
}
}

```

Объект значений Category:

```
<?php

namespace App\Entity;

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

/**
 * @ORM\Entity(repositoryClass="App\Repository\CategoryRepository")
 */
class Category
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=32)
     */
    private $name;

    /**
     * @ORM\OneToMany(targetEntity="App\Entity>Note", mappedBy="category")
     */
    private $notes;

    public function __construct()
    {
        $this->notes = new ArrayCollection();
    }

    public function getId(): ?int
    {
        return $this->id;
    }

    public function getName(): ?string
    {
        return $this->name;
    }
}
```

```

public function setName(string $name): self
{
    $this->name = $name;

    return $this;
}

/**
 * @return Collection|Note[]
 */
public function getNotes(): Collection
{
    return $this->notes;
}

public function addNote(Note $note): self
{
    if (!$this->notes->contains($note)) {
        $this->notes[] = $note;
        $note->setCategory($this);
    }

    return $this;
}

public function removeNote(Note $note): self
{
    if ($this->notes->contains($note)) {
        $this->notes->removeElement($note);
        // set the owning side to null (unless already changed)
        if ($note->getCategory() === $this) {
            $note->setCategory(null);
        }
    }

    return $this;
}
}

```

Событие:

```
<?php
namespace App\Entity;
use Doctrine\ORM\Mapping as ORM;
/**
 * @ORM\Entity(repositoryClass="App\Repository>NoteChangeEventRepository")
 */
class NoteChangeEvent implements \TimeEventInterface
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
     * @ORM\Column(type="integer")
     */
    private $id;
    /**
     * @ORM\OneToOne(targetEntity="App\Entity>Note", cascade={"persist", "remove"})
     * @ORM\JoinColumn(nullable=false)
     */
    private $Note;
    /**
     * @ORM\Column(type="date")
     */
    private $created;
    /**
     * @ORM\Column(type="date")
     */
    private $updatedAt;
    public function __construct()
    {
        $this->created = new \DateTimeImmutable();
        $this->updatedAt = null;
    }
    public function editNote()
    {
        $this->updatedAt = new \DateTimeImmutable();
    }
    public function getId(): ?int
    {
        return $this->id;
    }
    public function getNote(): ?Note
    {
        return $this->Note;
    }
}
```

```
public function setNote(Note $Note): self
{
    $this->Note = $Note;
    return $this;
}
public function getCreated(): ?\DateTimeInterface
{
    return $this->created;
}
public function setCreated(\DateTimeInterface $created): self
{
    $this->created = $created;
    return $this;
}
public function getUpdatedAt(): ?\DateTimeInterface
{
    return $this->updatedAt;
}
public function setUpdatedAt(\DateTimeInterface $updatedAt): self
{
    $this->updatedAt = $updatedAt;
    return $this;
}
public function createdAt()
{
    return $this->created;
}
public function updatedAt()
{
    return $this->updatedAt;
}
}
```

Вывод: Я познакомилась с тактическими шаблонами предметно-ориентированного проектирования.