

Министерство образования Республики Беларусь  
Учреждение образования  
«Брестский государственный технический  
университет»  
Кафедра «ЭВМ и системы»

Лабораторная работа №5  
Дисциплина «ПИС»

Выполнил:  
студент группы:  
ПО-3  
Будяков В.В.  
Проверил:  
Лаврущик А.И.

Брест 2021 г.

**Цель работы:** познакомиться с тактическими шаблонами предметно-ориентированного проектирования.

**Постановка задачи:** реализуйте не менее двух агрегатов, сущностей, объектов значений домена Вашей 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.

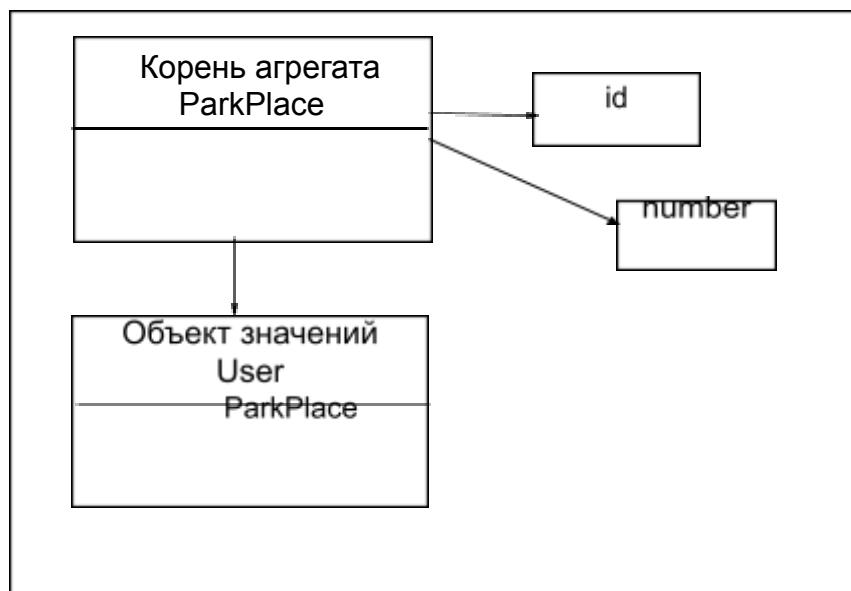
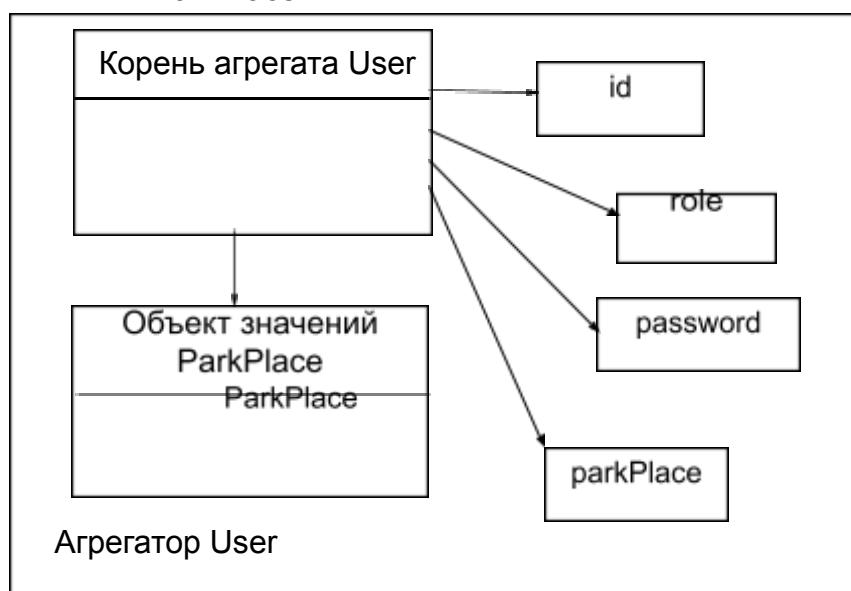
Понятие агрегата — это кластер доменных объектов, с которыми можно обращаться как с единым целым.

#### Ход работы:

Для связи моих объектов с объектами базы данных я использовал ORM framework. Для удобства, уже на уровне Entity я получал из базы данных агрегацию объектов, т. е. Entity ParkPlace содержит пользователя, который забронировал данное парковочное место, так же и Entity User содержит список парковочных мест, которые данный пользователь забронировал.

#### Агрегаторы:

- User
- ParkPlace



## Arperator ParkPlace

### Entiry User:

```
<?php
```

```
namespace App\Entity;

use Doctrine\Common\Collections\ArrayCollection;
use Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;
use Symfony\Component\Security\Core\User\UserInterface;

/

 * @ORM\Entity(repositoryClass="App\Repository\UserRepository")
 */
class User implements UserInterface
{
    /

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

    /

    * @ORM\Column(type="string", length=180, unique=true)
    */
    private $username;

    /

    * @ORM\Column(type="json")
    */
    private $roles = [];

    /

    * @var string The hashed password
    * @ORM\Column(type="string")
    */
    private $password;

    /

    * @ORM\OneToMany(targetEntity="App\Entity\ParkPlace", mappedBy="user")
    */
    private $parkPlaces;

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

    /

    * @return Collection|ParkPlace[]
    */
}
```

```

public function getParkPlaces(): Collection
{
    return $this->parkPlaces;
}

public function addParkPlace(ParkPlace $sparkPlace): self
{
    if (!$this->parkPlaces->contains($sparkPlace)) {
        $this->parkPlaces[] = $sparkPlace;
        $sparkPlace->setUser($this);
    }
    return $this;
}

public function removeParkPlace(ParkPlace $sparkPlace): self
{
    if ($this->parkPlaces->contains($sparkPlace)) {
        $this->parkPlaces->removeElement($sparkPlace);
        // set the owning side to null (unless already changed)
        if ($sparkPlace->getUser() === $this) {
            $sparkPlace->setUser(null);
        }
    }
    return $this;
}

```

### Entiry ParkPlace:

```

<?php
namespace App\Entity;
use App\event\TimeEventInterface;
use DateTimeImmutable;
use Doctrine\ORM\Mapping as ORM;

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

    /
    * @ORM\Column(type="integer")
    */
    private $number;

    /
    * @ORM\ManyToOne(targetEntity="App\Entity\User", inversedBy="parkPlaces")
    */

```

```

*/
private $user;

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

public function getNumber(): ?int
{
    return $this->number;
}

public function setNumber(int $number): self
{
    $this->number = $number;

    return $this;
}

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

public function setUser(?User $user): self
{
    $this->user = $user;
    $this->updatedAt = new DateTimeImmutable();
    return $this;
}

```

**Репозитории:**

- ParkPlaceRepository
- UserRepository

<?php

```

namespace App\Repository;

use App\Entity\ParkPlace;
use App\Entity\User;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Common\Persistence\ManagerRegistry;
use Doctrine\ORM\ORMException;
use Symfony\Component\Security\Core\Authentication\Token\Storage\TokenStorageInterface;
use Symfony\Component\Security\Core\Exception\UnsupportedUserException;

/**
 * @method ParkPlace|null find($id, $lockMode = null, $lockVersion = null)
 * @method ParkPlace|null findOneBy(array $criteria, array $orderBy = null)
 * @method ParkPlace[]    findAll()

```

```

 * @method ParkPlace[]  findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class ParkPlaceRepository extends ServiceEntityRepository
{
    private $tokenStorage;

    public function __construct(ManagerRegistry $registry, TokenStorageInterface
$tokenStorageInject)
    {
        parent::__construct($registry, ParkPlace::class);
        $this->tokenStorage = $tokenStorageInject;
    }

    /**
     * Used to upgrade (rehash) the user's password automatically over time.
     * @param ParkPlace $sparkPlace
     * @return bool
     */
    public function setUser(ParkPlace $sparkPlace): bool
    {
        $user = $this->tokenStorage->getToken()->getUser();
        if (!$user instanceof User) {
            throw new UnsupportedUserException(sprintf('Instances of "%s" are not supported.', \get_class($user)));
        }
        if ($sparkPlace->getUser() === null) {
            $sparkPlace->setUser($user);
            try {
                $this->em->flush();
            } catch (ORMException $e) {
                return false;
            }
            return true;
        } else {
            return false;
        }
    }

    /**
     * Used to upgrade (rehash) the user's password automatically over time.
     * @param ParkPlace $sparkPlace
     * @return bool
     */
    public function deleteUser(ParkPlace $sparkPlace): bool
    {
        $user = $this->tokenStorage->getToken()->getUser();
        if (!$user instanceof User) {
            throw new UnsupportedUserException(sprintf('Instances of "%s" are not supported.', \get_class($user)));
        }
        if ($sparkPlace->getUser() == $user) {

```

```

        $SparkPlace->setUser(null);
        try {
            $this->_em->flush();
        } catch (ORMException $e) {
            return false;
        }
        return true;
    } else {
        return false;
    }
}

<?php

namespace App\Repository;

use App\Entity\User;
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
use Doctrine\Common\Persistence\ManagerRegistry;
use Doctrine\ORM\OptimisticLockException;
use Doctrine\ORM\ORMException;
use Symfony\Component\Security\Core\Exception\UnsupportedUserException;
use Symfony\Component\Security\Core\User>PasswordUpgraderInterface;
use Symfony\Component\Security\Core\User\UserInterface;

/**
 * @method User|null find($id, $lockMode = null, $lockVersion = null)
 * @method User|null findOneBy(array $criteria, array $orderBy = null)
 * @method User[]   findAll()
 * @method User[]   findBy(array $criteria, array $orderBy = null, $limit = null, $offset = null)
 */
class UserRepository extends ServiceEntityRepository implements PasswordUpgraderInterface
{
    public function __construct(ManagerRegistry $registry)
    {
        parent::__construct($registry, User::class);
    }

    /**
     * Used to upgrade (rehash) the user's password automatically over time.
     */
    public function upgradePassword(UserInterface $user, string $newEncodedPassword): void
    {
        if (!$user instanceof User) {
            throw new UnsupportedUserException(sprintf('Instances of "%s" are not supported.',
                \get_class($user)));
        }

        $user->setPassword($newEncodedPassword);
        $this->_em->persist($user);
        $this->_em->flush();
    }
}

```

```

public function addUser(User $user): bool
{
    try {
        $this->_em->persist($user);
        $this->_em->flush();
        return true;
    } catch (OptimisticLockException|ORMException $e) {
        return false;
    }
}

```

### **Перейдем к доменным событиям**

Доменное событие — это класс с именем на основе страдательного причастия прошедшего времени. Он содержит свойства, которые выразительно передают это событие. В данной системе парковочные места можно бронировать и освобождать, и было бы удобно знать когда парковочное место в последний раз меняло статус.

Создадим интерфейс

```

interface TimeEventInterface
{
    public function createdAt();
    public function updatedAt();
}

```

Напишем реализацию данного интерфейса

```

class ParkPlace implements TimeEventInterface
{
    private $id;
    private $number;
    private $user;
    private $createdAt;
    private $updatedAt;

    public function __construct()
    {
        $this->createdAt = new DateTimeImmutable();
    }

    public function setUser(?User $user): self
    {
        $this->user = $user;
        $this->updatedAt = new DateTimeImmutable();
        return $this;
    }

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

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

```

}

Насчет **автогенерации идентификаторов сущностей** - они генерируются автоматически фреймворком на уровне репозитория

**Выводы:** познакомился с тактическими шаблонами предметно ориентированного проектирования.