Министерство образования Республики БеларусьУчреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Отчет
По лабораторной работе №5
по дисциплине «ПрИС»

Выполнила
студентка 3 курса
группы ПО-3
Гаврилкович Е.В.
Проверил
Лаврущик А.И.

Брест 2021

Лабораторная работа №5

Доменная модель

ВАРИАНТ 5

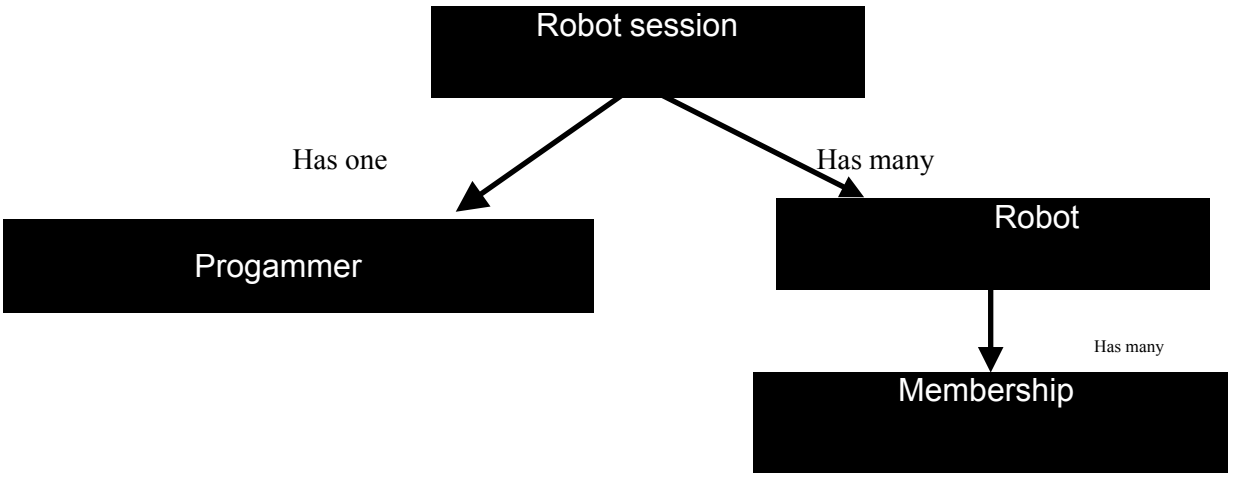
Цель работы. Познакомиться с тактическими шаблонами предметно-ориентированного проектирования.

Задание для выполнения. 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 включатьв 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. Проверку работы агрегатов организовать путём тестирования (ЛР4).

Предметная область. Промышленная робототехника.

Ход работы

Добавляем еще одну модель Membership и привязываем ее к покупкам.
Получаем следующую структуру моделей с агрегатами:



Программная реализация

Programmer.php

```
<?php

namespace App\Entity;

use App\Repository\ProgrammerRepository;
use
Doctrine\Common\Collections\ArrayCollection;use
Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=ProgrammerRepository::class)
 */
class Programmer
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
    * @ORM\Column(type="string", length=100)
     */

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

    /**
     * @ORM\ManyToMany(targetEntity=RobotSession::class, mappedBy="Programmer")
     */
    private $robotSession;

    // /**
    //  * @ORM\ManyToOne(targetEntity=Membership::class, inversedBy="member")
    //  */
    // private $membership;

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

    public function setNumber(int $number): self
    {

        $this->number = $number;

        return $this;
    }

    /**

     * @return Collection|RobotSession[]
     */
    public function getRobotSessions(): Collection
    {
        return $this->robotSession;
    }

    public function addRobotSession(RobotSession $robotSession): self
    {
        if (!$this->robotSession->contains($robotSession)) {
            $this->robotSession[] = $robotSession;
            $robotSession->addProgrammer($this);
        }

        return $this;
    }

    public function removeRobotSession(RobotSession $robotSession):
self
    {
        if ($this->robotSession->removeElement($robotSession)) {
            $robotSession->removeProgrammer($this);
        }

        return $this;
    }
}
```

Robot.php

```
<?php

namespace App\Entity;

use App\Repository\RobotRepository;
use
Doctrine\Common\Collections\ArrayCollection;use
Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=RobotRepository::class)
 */
class Robot
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;
```

```
/**
 * @ORM\Column(type="string", length=100)
 */
private $name;

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

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

/**
 * @ORM\OneToMany(targetEntity=RobotSession::class, mappedBy="Robot")
 */
private $robotSession;

public function__construct()
{
    $this->robotSession = 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 getCost(): ?int
{
    return $this->cost;
}

public function setCost(int $cost): self
{
    $this->cost = $cost;

    return $this;
}

public function getVersion(): ?int
{
    return $this->version;
}

public function setVersion(int $version): self
{

    $this->version = $version;

    return $this;
}

/**

 * @return Collection|RobotSession[]
```

```
     */
    public function getRobotSession(): Collection
    {
        return $this->robotSession;
    }

    public function addrobotSession(RobotSession $robotSession): self
    {
        if (!$this->robotSession->contains($robotSession)) {
            $this->robotSession[] = $robotSession;
            $robotSession->setRobot($this);
        }

        return $this;
    }

    public function removeRobotSession(RobotSession $robotSession):
self
    {
        if ($this->robotSession->removeElement($robotSession)) {
            // set the owning side to null (unless already
            changed)if ($robotSession->getRobot() === $this) {
                $robotSession->setRobot(null);
            }
        }

        return $this;
    }

    public function getMembership(): ?Membership
    {
        return $this->membership;
    }

    public function setMembership(?Membership $membership): self
    {
        $this->membership = $membership;

        return $this;
    }
}
```

Membership.php

```
<?php

namespace App\Entity;

use App\Repository\MembershipRepository;
use
Doctrine\Common\Collections\ArrayCollection;use
Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=MembershipRepository::class)
 */
class Membership
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="integer")
     */
```

```
private $duration;

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

/**
 * @ORM\OneToMany(targetEntity=Robot::class, mappedBy="membership")
 */
private $member;

public function___construct()
{
    $this->member = new ArrayCollection();
}

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

public function getDuration(): ?int
{
    return $this->duration;
}
public function setDuration(int $duration): self
{
    $this->duration = $duration;

    return $this;
}

public function getCost(): ?int
{
    return $this->cost;
}

public function setCost(int $cost): self
{

    $this->cost =

    $cost;return $this;
}

/**

 * @return Collection|Robot[]
 */
public function getMember(): Collection
{
    return $this->member;
}

public function addMember(Robot $member): self
{
    if (!$this->member->contains($member)) {
        $this->member[] = $member;
        $member->setMembership($this);
    }

    return $this;
}

public function removeMember(Robot $member): self
{
    if ($this->member->removeElement($member)) {
        // set the owning side to null (unless already
        changed)if ($member->getMembership() === $this) {
```

```
                $member->setMembership(null);
            }
        }

        return $this;
    }
}
```

RobotSession.php

```
<?php

namespace App\Entity;

use App\Repository\RobotSessionRepository; use
Doctrine\Common\Collections\ArrayCollection;use
Doctrine\Common\Collections\Collection;
use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass=RobotSessionRepository::class)
 */
class RobotSession
{
    /**
     * @ORM\Id
     * @ORM\GeneratedValue
     * @ORM\Column(type="integer")
     */
    private $id;

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

    /**
     * @ORM\ManyToOne(targetEntity=Cient::class, inversedBy="RobotSessions")
     */
    private $programmer;

    /**
     * @ORM\ManyToMany(targetEntity=Robot::class,
 inversedBy="RobotSessions")
     */
    private $robot;

    public function__construct()
    {
        $this->robot = new ArrayCollection();
    }

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

    public function getDate(): ?\DateTimeInterface
    {
        return $this->date;
    }

    public function setDate(\DateTimeInterface $date): self
    {
        $this->date = $date;

        return $this;
    }
```

```
    public function getRobot(): ?Robot
    {
        return $this->programmer;
    }

    public function setRobot(?Robot $robot): self
    {

        $this->robot = $robot;return

        $this;
    }

    /**

     * @return Collection|Robot[]
     */
    public function getRobot(): Collection
    {
        return $this->robot;
    }
    public function addProgrammer(Programmer $programmer): self
    {
        if (!$this->programmer->contains($programmer)) {
            $this->programmer[] = $programmer;
        }

        return $this;
    }

    public function removeProgrammer(Programmer $programmer): self
    {
        $this->programmer->removeElement($programmer);

        return $this;
    }
}
```

Тестирование агрегатных сущностей

RobotSessionTest.php

```
<?php namespace App\Tests;

use PHPUnit\Framework\TestCase;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;

class RobotSessionTest extends KernelTestCase
{
    private $entityManager;

    protected function setUp(): void
    {
        $kernel = self::bootKernel();
        $this->entityManager = $kernel->getContainer()->get('doctrine')-
>getManager();
    }

    public function testInsert()
    {
        $repository = $this->entityManager-
  >getRepository(\App\Entity\RobotSession::class);
          $robot_repository = $this->entityManager-
  >getRepository(\App\Entity\Robot::class);
       $programmer_repository = $this->entityManager-
>getRepository(\App\Entity\Programmer::cla
                    ss);
              $new_robot = $robot_repository->addRobot("test robot", 10,
```

```
10);
        $new_programmer = $programmer_repository->addProgrammer("test programmer",
        11111);
        $original_count = $repository->getRobotSessionCount();
        $repository->addRobotSession(new \DateTime(), $new_robot,

$new_programmer);
        $new_count = $repository->getRobotSessionCount();
        $this->assertEquals($original_count + 1, $new_count);
    }

    public function testFindAll()
    {
        $repository = $this->entityManager-
  >getRepository(\App\Entity\RobotSession::class);
        $our_count = $repository->getRobotSessionCount();
        $get_all_count = count($repository->findAll());
        $this->assertEquals($our_count, $get_all_count);
    }
}
```

MembershipTest.php

```
<?php namespace App\Tests;

use PHPUnit\Framework\TestCase;
use Symfony\Bundle\FrameworkBundle\Test\KernelTestCase;

class MembershipTest extends KernelTestCase
{
    private $entityManager;

    protected function setUp(): void
    {
        $kernel = self::bootKernel();

        $this->entityManager = $kernel->getContainer()
            ->get('doctrine')
            ->getManager();
    }

    public function testInsertAndAssignMember()
    {
        $repository = $this->entityManager-
>getRepository(\App\Entity\Membership::class);
        $robot_repository = $this->entityManager-
>getRepository(\App\Entity\Robot::class);

        $new_robot = $robot_repository->addRobot("iPhone", 1000, 10);

        $membership = $repository->addMembership(100, 60);

        $this->assertEquals(count($membership->getMember()), 0);

        $membership->addMember($new_robot);

        $this->assertEquals(count($membership->getMember()), 1);
    }

    public function testFindAll()
    {
        $repository = $this->entityManager-
>getRepository(\App\Entity\Membership::class);

        $our_count = $repository->getMembershipCount();
        $get_all_count = count($repository->findAll());
        $this->assertEquals($our_count, $get_all_count);
    }
```

```
}
```

```
Testing tests
........ 8 / 8 (166%)

Time: 164 ms, Memory: 15.00 МБ

IK (8 tests, 8 assertions)
```

Вывод

В данной лабораторной работе я познакомилась с тактическими шаблонами предметно-ориентированного проектирования.