МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА «ИИТ»

Лабораторная работа №5

По дисциплине «Проектирование интернет систем»

Выполнил: студент ФЭИС

группы ПО-2

Шелест П.Ю.

Проверила: Слинко Е.В.

Брест 2020

**Цель:** Познакомиться с тактическими шаблонами предметно-ориентированного проектирования

**Задание:**

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.

**Вариант**: *магазин электрооборудования*

Описание:

**1.Список сущностей**:

- EntityInterface – абстрактная сущность

- Client – клиент сайта

- Electrics – электрооборудование.

**2.Список объектов значений:**

- Adress – адрес клиента

- Provider – поставщик электрооборудования

**3.Список репозиториев:**

- ClientRepository – интерфейс репозитория клиентов

- ClientMySQLRepository – реализация для MySql репозитория клиента

- ProductRepository – интерфейс репозитория техники

- ProductMySQLRepository – реализация для MySql репозитория электрооборудования

- OrderRepository – интерфейс репозитория событий

- OrderMySQLRepository – реализация для MySql репозитория событий

**4.Список доменных событий:**

**-** OrderInterface – интерфейс события

- OrderDispatcher – диспетчер событий

- Authorization – авторизация клиента в системе

- ClientByProduct – покупка клиентом электрооборудования

**Агрегаторы:**

Корень агрегата Produсt

Корень агрегата Client

Объект значений Adress

Объект значений Provider

**Агрегат Product**

**Агрегат Client**

**Текст программы:**

    abstract class EntityInterface

    {

        protected $id;

        public function \_\_construct()

        {

            $this->id = uniqid();

        }

        public function getId(): string

        {

            return $this->id;

        }

    }

class Adress extends EntityInterface

{

    private $adress\_id;

    private $city;

    private $street;

    private $home;

    public function \_\_construct(string $city,string $adress\_id, string $street, string $home)

    {

       parent::\_\_construct();

       $this->adress\_id = $adress\_id;

       $this->city = $city;

       $this->street=$street;

       $this->home=$home;

    }

    public function getAdressId(): string

    {

       return $this->adress\_id;

    }

    public function getStreet(): string

    {

       return $this->street;

    }

    public function getCity(): string

    {

       return $this->city;

    }

    public function getHome(): string

    {

       return $this->home;

    }

    public function setAdressId(string $city\_id)

    {

       $this->city\_id = $city\_id;

    }

    public function setStreet(string $street)

    {

       $this->street = $street;

    }

    public function setHome(string $home)

    {

       $this->home = $home;

    }

}

class Client extends EntityInterface

{

    private $login;

    private $adress;

    private $user\_id;

    private $Order= [];

    public function \_\_construct(string $login, string $adress\_id,string $city, string $street, string $home, string $sub\_id)

    {

       parent::\_\_construct();

       $this->login = $login;

       $this->adress=new Adress($city,$adress\_id, $street, $home);

       $this->user\_id=$sub\_id;

    }

    public function getLogin(): string

    {

       return $this->login;

    }

    public function getAdressId(): string

    {

       return $this->adress->getAdressId();

    }

    public function getCity(): string

    {

       return $this->adress->getCity();

    }

    public function getStreet(): string

    {

       return $this->adress->getStreet();

    }

    public function getHome(): string

    {

        return $this->adress->getHome();

    }

    public function getUserId(): string

    {

       return $this->user\_id;

    }

    public function setLogin(string $login)

    {

        $this->login=$login;

    }

    public function setAdressId(string $adress\_id)

    {

        $this->adress->setAdressId($adress\_id);

    }

    public function setStreet(string $street)

    {

        $this->adress->setStreet($street);

    }

    public function setHome(string $home)

    {

        $this->adress->setHome($home);

    }

    public function setUserId(string $user\_id)

    {

        $this->user\_id = $user\_id;

    }

    public function authorize()

    {

        $this->recordOrder(new Authorization($this));

    }

    private function recordOrder(OrderInterface $Order)

    {

        $this->Order[] = $Order;

    }

    public function releaseOrder(): array

    {

        $temp=$this->Order;

        $this->Order= [];

        return $temp;

    }

}

class Provider extends EntityInterface

{

    private $provider\_id;

    private $provider\_name;

    public function \_\_construct(string $provider\_id,string $provider\_name)

    {

       parent::\_\_construct();

       $this->provider\_id = $provider\_id;

       $this->provider\_name = $provider\_name;

    }

    public function getProviderId(): string

    {

       return $this->provider\_id;

    }

    public function getProviderName(): string

    {

       return $this->provider\_name;

    }

    public function setProviderName(string $city\_name)

    {

       $this->provider\_name = $provider\_name;

    }

}

class Product extends EntityInterface

{

    private $name;

    private $price;

    private $providers = [];

    private $good\_id;

    private $Order = [];

    private $inform;

    private $image;

    public function \_\_construct(string $name, string $price, string $good\_id,string $inform,string $image)

    {

       parent::\_\_construct();

       $this->name = $name;

       $this->price = $price;

       $this->good\_id=$good\_id;

       $this->inform = $inform;

       $this->image = $image;

    }

    public function getName(): string

    {

       return $this->name;

    }

    public function getPrice(): string

    {

       return $this->price;

    }

    public function getGoodId(): string

    {

       return $this->good\_id;

    }

    public function getInform(): string

    {

       return $this->inform;

    }

    public function getImage(): string

    {

       return $this->image;

    }

    public function setName(string $name)

    {

        $this->name=$name;

    }

    public function setPrice(string $price)

    {

        $this->price = $price;

    }

    public function setGoodId(string $good\_id)

    {

        $this->good\_id = $good\_id;

    }

    public function setInform(string $inform)

    {

        $this->inform = $inform;

    }

    public function setImage(string $image)

    {

        $this->image = $image;

    }

    public function AddProvider(string $provider\_id,string $provider\_name)

    {

        for($i=0; $i<count($this->providers); $i++)

            if($this->providers[$i]->getProviderId()===$provider\_id)

                return;

        $prov=new Provider($provider\_id,$provider\_name);

        $this->providers[] = $prov;

    }

    public function removeProvider(string $provider\_id)

    {

        for($i=0; $i<count($this->providers); $i++)

            if($this->cities[$i]->getProviderId()===$provider\_id)

            {

                unset($providers[$i]);

                return;

            }

    }

    public function ClientBy(Client $client)

    {

        $this->recordOrder(new ClientByProduct($client, $this));

    }

    private function recordOrder(OrderInterface $Order)

    {

        $this->Order[] = $Order;

    }

    public function releaseOrder(): array

    {

        $temp=$this->Order;

        $this->Order= [];

        return $temp;

    }

}

interface ClientRepository

    {

        public function getClientByLP(string $login, string $pass);

    }

class ClientMySQLRepository implements ClientRepository

{

    private $mysqli;

    public function \_\_construct()

    {

        $this->mysqli = new mysqli("localhost", "root", "", "pris");

    }

    public function getClientByLP(string $login, string $pass)

    {

        $stmt = $this->mysqli->prepare("

        SELECT B.id as id,login,city,street,home,A.id as adr FROM pris.users

as B Left outer join pris.addresses as A ON A.id\_user = B.id

where login=?  AND  password = ?");

        mysqli\_stmt\_bind\_param($stmt, "ss", $login, $pass);

        mysqli\_stmt\_execute($stmt);

        $login;$sub\_id; $adress\_id; $street; $home;

        mysqli\_stmt\_bind\_result($stmt,$sub\_id,$login,$city, $street, $home,$adress\_id);

        mysqli\_stmt\_fetch($stmt);

        if($sub\_id!="")

        {

            return new Client($login, $adress\_id, $city,$street, $home, $sub\_id);

        }

        else

        {

            return null;

        }

    }

}

interface ProductRepository

{

    public function getProduct();

    public function addProduct(string $name, string $price,string $inform, string $image);

    public function removeProduct(Product $Product);

    public function ClientByProduct(Client $client, Product $Product);

    public function getClientProduct(Client $client);

    public function getProductById(string $id);

}

class ProductMySQLRepository implements ProductRepository

{

    private $mysqli;

    public function \_\_construct()

    {

        $this->mysqli = new mysqli("localhost", "root", "", "pris");

    }

    public function getProduct()

    {

        $Product = [];

        $res = $this->mysqli->query("SELECT \* FROM pris.goods;");

        if($res->num\_rows==0)

            return null;

        while ($row = $res->fetch\_assoc())

        {

            $Product = new Product ($row['name'],$row['price'],$row['id'],$row['inform'],$row['image\_link']);

            $subres=$this->mysqli->query("SELECT \* FROM pris.good\_provider

             INNER JOIN pris.providers as P ON P.provider\_id = id\_provider where id\_good ='".$row['id']."'");

            while ($subrow = $subres->fetch\_assoc())

                $Product->addProvider($subrow['provider\_id'],$subrow['provider\_name']);

            $Product[] = $Product;

        }

        return $Product;

    }

    public function getClientProduct(Client $client)

    {

        $id\_client = $client->getUserId();

        $Product = [];

        $res = $this->mysqli->query("SELECT \* FROM user\_good where `id\_user`='".$id\_client."'");

        if($res->num\_rows==0)

            return null;

            $res = $this->mysqli->query("SELECT \* FROM user\_good where `id\_user`='".$id\_client."'");

            while ($row1 = mysqli\_fetch\_array($res))

                {

                    $result =  $this->mysqli->query("SELECT \* FROM goods where `id`=$row1[id\_good]");

                    while($row = mysqli\_fetch\_array($result))

                    {

                        $Product = new Product($row['name'],$row['price'],$row['id'],$row['inform'],$row['image\_link']);

                        $subres=$this->mysqli->query("SELECT \* FROM pris.good\_provider

                        INNER JOIN pris.providers as P ON P.provider\_id = id\_provider where id\_good ='".$row['id']."'");

                        while ($subrow = $subres->fetch\_assoc())

                        $Product->addProvider($subrow['provider\_id'],$subrow['provider\_name']);

                        $Product[] = $Product;

                    }

                }

        return $Product;

    }

    public function addProduct(string $name, string $price,string $inform, string $image)

    {

        if(!is\_numeric($price))

            return 0;

        $stmt = $this->mysqli->prepare("INSERT INTO `pris`.`goods` (`name`, `price`, `inform`, `image\_link`) VALUES (?, ?, ?, ?);");

        mysqli\_stmt\_bind\_param($stmt, "sdss",$name, $price,$inform,$image);

        mysqli\_stmt\_execute($stmt);

        if(!$stmt)

            return 0;

        return 1;

    }

    public function removeProduct(Product $Product)

    {

        $id=$Product->getGoodId();

        $stmt = $this->mysqli->prepare("DELETE FROM `pris`.`goods` WHERE (`id` = ?)");

        mysqli\_stmt\_bind\_param($stmt, "s", $id);

        mysqli\_stmt\_execute($stmt);

        $stmt = $this->mysqli->prepare("DELETE FROM pris.good\_provider where (`id\_good` = ?)");

        mysqli\_stmt\_bind\_param($stmt, "s", $id);

        mysqli\_stmt\_execute($stmt);

        $stmt = $this->mysqli->prepare("DELETE FROM `pris`.`user\_good` WHERE (`id\_good` = ?)");

        mysqli\_stmt\_bind\_param($stmt, "s", $id);

        mysqli\_stmt\_execute($stmt);

    }

    public function ClientByProduct(Client $client, Product $Product)

    {

        $clientId=$client->getUserId();

        $ProductId=$Product->getGoodId();

        $stmt = $this->mysqli->prepare("INSERT INTO `pris`.`user\_good` (`id\_user`, `id\_good`) VALUES (?, ?)");

        mysqli\_stmt\_bind\_param($stmt, "ss", $clientId, $ProductId);

        mysqli\_stmt\_execute($stmt);

        if(!$stmt)

            return 0;

        return 1;

    }

    public function getProductById($id)

    {

       $Product;

        $res = $this->mysqli->query("SELECT \* FROM pris.goods;");

        if($res->num\_rows==0)

            return null;

        while ($row = $res->fetch\_assoc())

        {

            $Product = new Product ($row['name'],$row['price'],$row['id'],$row['inform'],$row['image\_link']);

            if($Product->getGoodId()==$id)

            return $Product;

        }

        $Product=0;

        return $Product;

    }

}

interface OrderRepository

    {

        public function writeOrder(string $time, string $message);

    }

class OrderRepositoryMySQL implements OrderRepository

{

    private $mysqli;

    public function \_\_construct()

    {

        $this->mysqli = new mysqli("localhost", "root", "", "pris");

    }

    public function writeOrder(string $time, string $message)

    {

        $stmt = $this->mysqli->prepare("INSERT INTO pris.Order (time, message) VALUES(?,?)");

        mysqli\_stmt\_bind\_param($stmt, "ss", $time, $message);

        mysqli\_stmt\_execute($stmt);

    }

}

interface OrderInterface

    {

        public function getOccurredOn();

        public function getMessage();

    }

class Authorization implements OrderInterface

    {

        private $client;

        private $occurredOn;

        public function \_\_construct(Client $client)

        {

            $this->client = $client;

            $this->occurredOn = new \DateTimeImmutable();

        }

        public function getOccurredOn()

        {

            return $this->occurredOn;

        }

        public function getMessage()

        {

            return "Клиент :".$this->client->getLogin()." авторизовался в системе";

        }

    }

class ClientByProduct implements OrderInterface

    {

        private $client;

        private $occurredOn;

        public function \_\_construct(Client $client)

        {

            $this->client = $client;

            $this->occurredOn = new \DateTimeImmutable();

        }

        public function getOccurredOn()

        {

            return $this->occurredOn;

        }

        public function getMessage()

        {

            return "Клиент :".$this->client->getLogin()." авторизовался в системе";

        }

    }

class OrderDispatcher

{

    public function dispatch(array $Order)

    {

        $OrderRepository=new OrderRepositoryMySQL();

        for($i=0; $i<count($Order); $i++)

        {

            $OrderRepository->writeOrder($Order[$i]->getOccurredOn()->format('Y-m-d H:i:s'), $Order[$i]->getMessage());

        }

    }

}

**Выводы:**

В результате выполнения работы я ознакомился с тактическими шаблонами предметно-ориентированного проектирования и написал программу, демонстрирующую проделанную работу.