Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №5

По дисциплине «ПИС»

за 5-й семестр

Выполнил:
студент 2 курса
группы ПО-3 (1)
Афанасьев В.В.

Проверил:
Лаврущик А.И.

Брест, 2021

Цель работы: познакомиться с тактическими шаблонами предметно ориентированного проектирования.

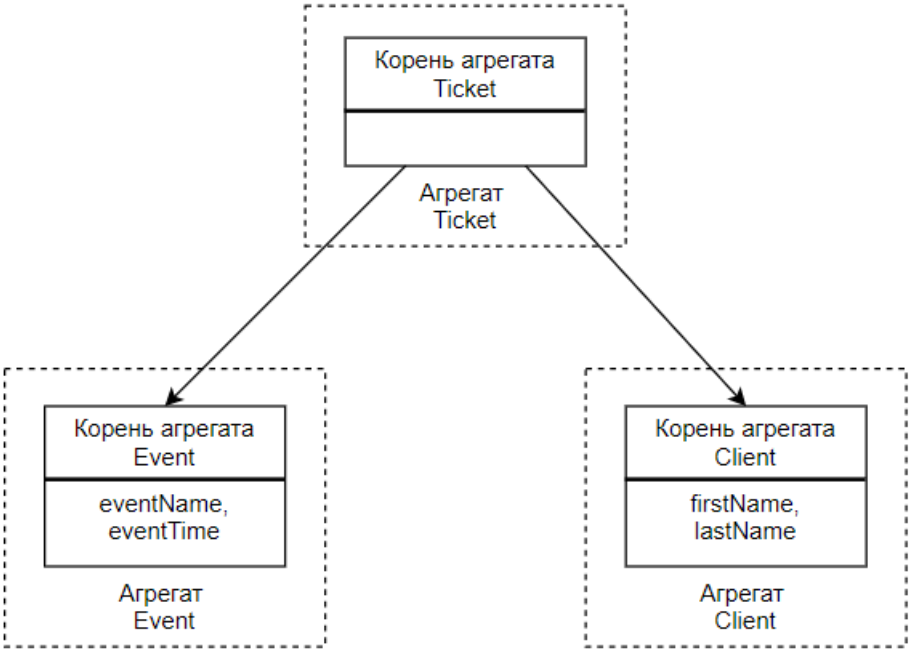
Вариант: 2

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. Проверку работы агрегатов организовать путём тестирования (ЛР №4).

Предметная область: продажа билетов на мероприятия.

Ход работы:



Исходный код:

Client.php

```
<?php

class Client
{
    private int $id;

    private string $firstName;
    private string $lastName;

    private array $events;

    /**
     * @return string
     */
    public function getFirstName(): string
    {
        return $this->firstName;
    }
```

```
    /**
     * @param string $firstName
     */
    public function setFirstName(string $firstName): void
    {
        $this->firstName = $firstName;
    }

    /**
     * @return string
     */
    public function getLastName(): string
    {
        return $this->lastName;
    }

    /**
     * @param string $lastName
     */
    public function setLastName(string $lastName): void
    {
        $this->lastName = $lastName;
    }

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

    /**
     * @param int $id
     */
    public function setId(int $id): void
    {
        $this->id = $id;
    }

    /**
     * @return array
     */
    public function getEvents(): array
    {
        return $this->events;
    }

    /**
     * @param array $events
     */
    public function setEvents(array $events): void
    {
        $this->events = $events;
    }
}
```

Event.php

```
<?php

class Event
{
    private int $id;

    private string $eventName;
    private string $eventTime;

    private array $clients;

    /**
     * @return int
     */
```

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

    /**
     * @param int $id
     */
    public function setId(int $id): void
    {
        $this->id = $id;
    }

    /**
     * @return string
     */
    public function getEventName(): string
    {
        return $this->eventName;
    }

    /**
     * @param string $eventName
     */
    public function setEventName(string $eventName): void
    {
        $this->eventName = $eventName;
    }

    /**
     * @return string
     */
    public function getEventTime(): string
    {
        return $this->eventTime;
    }

    /**
     * @param string $eventTime
     */
    public function setEventTime(string $eventTime): void
    {
        $this->eventTime = $eventTime;
    }

    /**
     * @param array $clients
     */
    public function setClients(array $clients): void
    {
        $this->clients = $clients;
    }

    /**
     * @return array
     */
    public function getClients(): array
    {
        return $this->clients;
    }
}
```

Ticket.php

```
<?php

class Ticket
{
    private int $id;

    private int $eventId;
    private Event $event;

    private int $clientId;
```

```
    private Client $client;

    private int $cost;

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

    /**
     * @param int $id
     */
    public function setId(int $id): void
    {
        $this->id = $id;
    }

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

    /**
     * @param int $cost
     */
    public function setCost(int $cost): void
    {
        $this->cost = $cost;
    }

    /**
     * @return int
     */
    public function getEventId(): int
    {
        return $this->eventId;
    }

    /**
     * @param int $eventId
     */
    public function setEventId(int $eventId): void
    {
        $this->eventId = $eventId;
    }

    /**
     * @return int
     */
    public function getClientId(): int
    {
        return $this->clientId;
    }

    /**
     * @param int $clientId
     */
    public function setClientId(int $clientId): void
    {
        $this->clientId = $clientId;
    }

    /**
     * @return Event
     */
    public function getEvent(): Event
    {
        return $this->event;
```

```
    }

    /**
     * @param Event $event
     */
    public function setEvent(Event $event): void
    {
        $this->event = $event;
    }

    /**
     * @return Client
     */
    public function getClient(): Client
    {
        return $this->client;
    }

    /**
     * @param Client $client
     */
    public function setClient(Client $client): void
    {
        $this->client = $client;
    }
}
```

ClientRepository.php

```
<?php
include("Interfaces/IClientRepository.php");

class ClientRepository implements IClientRepository
{
    private PDO $connection;

    public function __construct(PDO $connection)
    {
        $this->connection = $connection;
    }

    public function getAll(): array
    {
        $query = 'SELECT * FROM "Client"';
        $stmt = $this->connection->prepare($query);
        $stmt->execute();
        return $stmt->fetchAll();
    }

    public function create(Client $entity)
    {
        $query = 'INSERT INTO "Client"
                    ("FirstName", "LastName")
                VALUES
                    (?, ?)';

        $stmt = $this->connection->prepare($query);
        $stmt->execute([$entity->getFirstName(), $entity->getLastName()]);
        return $stmt->fetch();
    }

    public function getByIdOrNull($id): Client
    {
        // TODO: Implement getByIdOrNull() method.
    }

    public function update(Client $entity)
    {
        // TODO: Implement update() method.
    }

    public function delete($id)
    {
        // TODO: Implement delete() method.
```

```
    }
}
```

EventRepository.php

```
<?php
include("Interfaces/IEventRepository.php");

class EventRepository implements IEventRepository
{
    private PDO $connection;

    public function __construct(PDO $connection)
    {
        $this->connection = $connection;
    }

    public function getAll(): array
    {
        $query = 'SELECT * FROM "Event"';
        $stmt = $this->connection->prepare($query);
        $stmt->execute();
        return $stmt->fetchAll();
    }

    public function create(Event $entity)
    {
        $query = 'INSERT INTO "Event"
                        ("EventName", "EventTime")
                VALUES
                        (?, ?)';

        $stmt = $this->connection->prepare($query);
        $stmt->execute([$entity->getEventName(), $entity->getEventTime()]);

        return $stmt->fetch();
    }

    public function getByIdOrNull($id): Event
    {
        // TODO: Implement getByIdOrNull() method.
    }

    public function update(Event $entity)
    {
        // TODO: Implement update() method.
    }

    public function delete($id)
    {
        // TODO: Implement delete() method.
    }
}
```

TicketRepository.php

```
<?php
include("Interfaces/ITicketRepository.php");

class TicketRepository implements ITicketRepository
{
    private PDO $connection;

    public function __construct(PDO $connection)
    {
        $this->connection = $connection;
    }

    public function getAll(): array
    {
        $query = 'SELECT * FROM "Ticket"';
        $stmt = $this->connection->prepare($query);
        $stmt->execute();
        return $stmt->fetchAll();
```

```
    }

    public function create(Ticket $entity)
    {
        $query = 'INSERT INTO "Ticket"
                    ("ClientId", "EventId", "Cost")
                VALUES
                    (?, ?, ?)';

        $stmt = $this->connection->prepare($query);
        $stmt->execute([$entity->getClientId(), $entity->getEventId(), $entity->getCost()]);
        return $stmt->fetch();
    }

    public function getByIdOrNull($id): Ticket
    {
        // TODO: Implement getByIdOrNull() method.
    }

    public function update(Ticket $entity)
    {
        // TODO: Implement update() method.
    }

    public function delete($id)
    {
        // TODO: Implement delete() method.
    }
}
```

Выводы: в ходе выполнения лабораторной работы были ознакомлены с тактическими шаблонами предметно ориентированного проектирования.