Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій

Кафедра програмних систем і технологій

ЗВІТ

з лабораторної роботи № 5

Тема: **“ Структурні патерни проектування ”**

Дисципліна «Об’єктно-орієнтоване конструювання програм»

Підготував:
студент гр. ІПЗ-22(1)
Польнюк Руслан Дмитрович
Дата : _____

Перевірила:
доц. Зубик Л.В.

Тема: Структурні патерни проектування.
Мета роботи: ознайомитися з структурними паттернами проектування.
Умова: Розробити класи (згідно варіанту) з використанням структурного патерну проєктування. Реалізувати програму на C# з простим графічним інтерфейсом. У висновках обґрунтувати вибір патерну і пояснити, яким чином він дав змогу спростити виконання вашого варіанту завдання.

Завдання :

Розробити класи (згідно варіанту) з використанням структурного патерну проєктування. Реалізувати програму на C# з простим графічним інтерфейсом. У висновках обґрунтувати вибір патерну і пояснити, яким чином він дав змогу спростити виконання вашого варіанту завдання.

Програма обліку товарів на складі. Характеристики товарів: назва, вартість, дата надходження, вага, габарити (ШхГхВ). Методи прийому товару на склад та відвантаження зі складу. Методи отримання інформації про всі товари на складі. Пошук товарів за вибраною характеристикою. Підрахунок кількості товару з однаковою назвою, не враховуючи інші характеристики. Передбачити можливість додавання характеристик товарів (наприклад, температура зберігання).

Хід роботи:

Для реалізації даної лабораторної роботи з запропонованих паттернів я обрав паттерн “Фасад”.

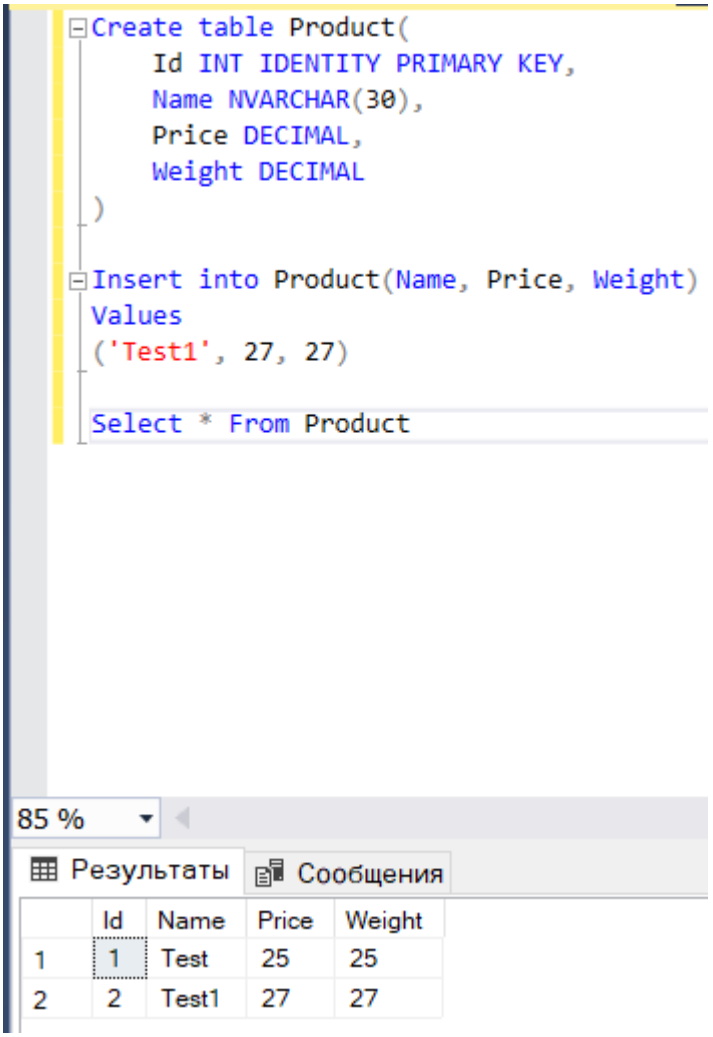
Фасад - це простий інтерфейс для роботи зі складною підсистемою, що містить багато класів. Фасад може мати урізаний інтерфейс, що не має 100% функціональності, якої можна досягти, використовуючи складну підсистему безпосередньо. Але він надає ті фічі, які потрібні клієнту, і приховує всі інші. Фасад корисний, якщо ви використовуєте якусь складну бібліотеку з безліччю рухомих частин, але потрібна лише частина її можливостей.

Застосованість паттерну:

- Коли вам потрібно уявити простий або урізаний інтерфейс до складної підсистеми.

- Коли ви бажаєте розкласти підсистему на окремі шари.

Для реалізації поставленого завдання згідно варіанту ми будемо використовувати Orm Dapper та паттерн Facade, першим кроком потрібно створити базу даних яку ми будемо використовувати в нашому застосунку. Для демонстрації паттерну спроектована не складна ієрархія бд з 1 таблицею.



Після виконання даного кроку можна приступити до реалізації програмного коду на мові C#. Для початку створимо бізнес-сутності, які будуть відповідати структурі бд.

```
Ссылок: 0
public class Product
{
    Ссылок: 0
    public int Id { get; set; }
    Ссылок: 0
    public string Name { get; set; }
    Ссылок: 0
    public decimal Price { get; set; }
    Ссылок: 0
    public decimal Weight { get; set; }
}
```

Далі можна налаштувати зв’язок з нашою базою даних за допомогою Dapper, та створити сервіси які будуть комунікувати з бд та видавати відповіді фронтенд застосунку на основі MVC(Razor).

```
Ссылок: 0
public class DatabaseConnection
{
    Ссылок: 0
    public static SqlConnection CreateConnection()
    {
        return new SqlConnection("Data Source=DESKTOP-ELR5B0P;Initial Catalog=Sklad;Integrated Security=True;");
    }
}
```

```
Ссылок: 0
public class SkladService
{
    Ссылок: 0
    public IEnumerable<Product> GetAllProducts()
    {
        using(var connection = DatabaseConnection.CreateConnection())
        {
            return connection.Query<Product>("SELECT * FROM Product");
        }
    }

    Ссылок: 0
    public void AddProduct(Product product)
    {
        using(var connection = DatabaseConnection.CreateConnection())
        {
            connection.Execute("INSERT INTO Product (Name, Price, Weight) Values(@Name, @Price, @Weight)");
        }
    }

    Ссылок: 0
    public IEnumerable<Product> GetByPrice(decimal price)
    {
        using(var connection = DatabaseConnection.CreateConnection())
        {
            var sql = $"SELECT * FROM Product WHERE Price = {price}";
            return connection.Query<Product>(sql);
        }
    }
}
```

Після створення сервісу потрібно створити клас, який буде створювати екземпляри сервісу та викликати необхідні методи.

Цей клас буде названо Facade, реалізація цього класу є дуже простою.

```
Ссылок: 1
public class Facade
{
    private readonly SkladService _service;

    Ссылок: 0
    public Facade(SkladService service)
    {
        _service = service;
    }

    Ссылок: 0
    public IEnumerable<Product> GetAllProducts()
    {
        return _service.GetAllProducts();
    }

    Ссылок: 0
    public void AddProduct(Product product)
    {
        _service.AddProduct(product);
    }

    Ссылок: 0
    public IEnumerable<Product> GetByPrice(decimal price)
    {
        return _service.GetByPrice(price);
    }
}
```

Після виконання попередніх кроків ми можеме приступи до реалізації користувацького інтерфейсу на основі Razor. Для початку зареєструємо наш клас Facade, та налаштуємо routing.

```
Ссылок: 0
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();
    services.AddSingleton<FacadePattern>();
}
```

Одним з останніх кроків буде створення контролів, які будуть проміжним слоєм комунікації між бд та користувацьким інтерфейсом

```
public class BaseController : Controller
{

    public IActionResult Index()
    {
        return View();
    }
}
```

```
Ссылок: 1
public class SkladController : Controller
{
    private readonly FacadePattern _facade;

    Ссылок: 0
    public SkladController(FacadePattern facade)
    {
        _facade = facade;
    }

    Ссылок: 0
    public IActionResult Index()
    {
        return View();
    }

    [HttpGet]
    Ссылок: 0
    public IActionResult GetAllProducts()
    {
        var result = _facade.GetAllProducts();
        return View(result);
    }

    [HttpGet]
    Ссылок: 0
    public IActionResult AddProduct(int Id)
    {
        ViewBag.Id = Id;
        return View();
    }
```

```
[HttpPost]
Ссылок: 0
public string AddProduct(Product product)
{
    _facade.AddProduct(product);
    return "Product," + product.Name + ",Added";
}

[HttpGet]
Ссылок: 0
public IActionResult GetByPrice(decimal price)
{
    var result = _facade.GetByPrice(price);
    return View(result);
}
```

Останній крок це створення користувацького інтерфейсу за допомогою Razor, я продемонструє на прикладі коду для організації, тому що для департаменту він буде ідентичний.

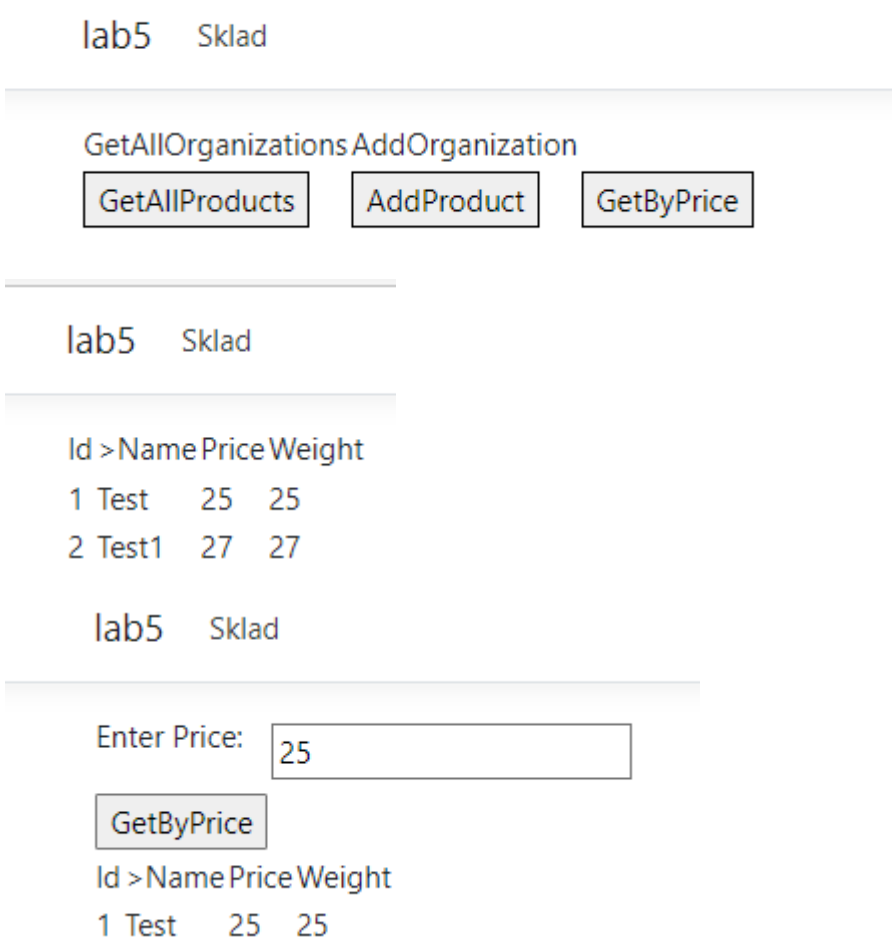
```
<body>
    <div>
    <table>
        <tr>
            <td>GetAllOrganizations</td>
            <td>AddOrganization</td>
        </tr>
        <tr>
            <td>
                <form action="@Url.Action("GetAllProducts", "Sklad")" method="get">
                 <input type="hidden"/>
                 <button type="submit">GetAllProducts</button>
                </form>
            </td>
            <td>
                  <form action="@Url.Action("AddProduct", "Sklad")" method="get">
                  <input type="hidden" name="Id"/>
                  <button type="submit">AddProduct</button>
                  </form>
             </td>
             <td>
                  <form action="@Url.Action("GetByPrice", "Sklad")" method="get">
                  <input type="hidden" name="Id"/>
                  <button type="submit">GetByPrice</button>
                  </form>
             </td>
        </tr>
    </table>
 </div>
 </body>
```

```
<body>
    <div>
    <table>
        <tr>
            <td>Id</td>
            <td>>Name</td>
            <td>Price</td>
            <td>Weight</td>
        </tr>
        @foreach (var value in Model)
        {
            <tr>
              <td>@value.Id</td>
              <td>@value.Name</td>
              <td>@value.Price</td>
              <td>@value.Weight</td>
            </tr>
        }
    </table>
    </div>
 </body>
```

```
<form method="post" action="">
    <input type="hidden" value="@ViewBag.Id" name="Id"/>
    <table>
        <tr>
            <td><p>Enter Name:</p></td>
            <td><input type="text" name="Name"/></td>
        </tr>
        <tr>
            <td><p>Enter Price:</p></td>
            <td><input type="text" name="Price"/></td>
        </tr>
        <tr>
            <td><p>Enter Weight:</p></td>
            <td><input type="text" name="Weight"/></td>
        </tr>
        <tr>
            <td><input type="submit" value="Add Product"/></td>
        </tr>
    </table>
</form>
```

```
<form method="get" action="">
    <input type="hidden" value="@ViewBag.Id" name="Id"/>
    <table>
        <tr>
            <td><p>Enter Price:</p></td>
            <td><input type="text" name="Price"/></td>
        </tr>
        <tr>
            <td><input type="submit" value="GetByPrice"/></td>
        </tr>
    </table>
</form>
<body>
    <div>
    <table>
        <tr>
            <td>Id</td>
            <td>>Name</td>
            <td>Price</td>
            <td>Weight</td>
        </tr>
        @foreach (var value in Model)
        {
            <tr>
              <td>@value.Id</td>
              <td>@value.Name</td>
              <td>@value.Price</td>
              <td>@value.Weight</td>
            </tr>
        }
    </table>
    </div>
</body>
```

Результат:



Висновок

Під час виконання цією лабораторної роботи я ознайомився з структурними патернами проектування, які входять в Gof, та допомагають створювати більш гнучкі та великі системи. Для реалізації завдання я обрав патерн “Фасад”, тому що вважаю він більш доцільно підходить під це завдання.