Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційні систем та технологій

Лабораторна робота №4

із дисципліни «*Технології розробки програмного забезпечення*»

Тема: «Вступ до паттернів проектування»

Виконав:
Студент групи ІА-31
Клим’юк В.Л.

Перевірив:
Мягкий М.Ю.

Київ-2025

Тема: Система запису на прийом до лікаря (strategy, adapter, observer, facade, visitor, client-server).
Система дозволяє пацієнтам шукати та записуватись на прийоми до лікарів, де їм виноситься висновок за результатами прийому. Також наявні функції відміни прийому та додавання відгуків про лікарів.

Репозиторій: https://github.com/StaticReadonly/kpi-trpz-labs-klim

Мета:
Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

Теоретичні відомості:

Будь-який патерн проєктування, використовуваний при розробці інформаційних систем, являє собою формалізований опис, який часто зустрічається в завданнях проєктування, вдале рішення даної задачі, а також рекомендації по застосуванню цього рішення в різних ситуаціях. Крім того, патерн проєктування обов'язково має загальновживане найменування. Правильно сформульований патерн проєктування дозволяє, відшукавши одного разу вдале рішення, користуватися ним знову і знову.

Відповідне використання патернів проєктування дає розробнику ряд незаперечних переваг. Модель системи, побудована в межах патернів проєктування, фактично є структурованим виокремленням тих елементів і зв'язків, які значимі при вирішенні поставленого завдання. Крім цього, модель, побудована з використанням патернів проєктування, більш проста і наочна у вивченні, ніж стандартна модель. Проте, не дивлячись на простоту і наочність, вона дозволяє глибоко і всебічно опрацювати архітектуру розроблюваної системи з використанням спеціальної мови. Застосування патернів проєктування підвищує стійкість системи до зміни вимог та спрощує неминуче подальше доопрацювання системи. Крім того, важко переоцінити роль використання патернів при інтеграції інформаційних систем організації. Також слід зазначити, що сукупність патернів проєктування, по суті, являє собою єдиний словник проєктування, який, будучи уніфікованим засобом, незамінний для спілкування розробників один одним. Таким чином шаблони представляють собою, підтверджені роками розробок в різних компаніях і на різних проєктах,

«ескізи» архітектурних рішень, які зручно застосовувати у відповідних обставинах.

«Singleton» (Одинак) являє собою клас в термінах ООП, який може мати не більше одного об'єкта (звідси і назва «одинак»). Насправді, кількість об'єктів можна задати (тобто не можна створити більш n об'єктів даного класу). Даний об'єкт найчастіше зберігається як статичне поле в самому класі.

«Iterator» (Ітератор) являє собою шаблон реалізації об'єкта доступу до набору (колекції, агрегату) елементів без розкриття внутрішніх механізмів реалізації. Ітератор виносить функціональність перебору колекції елементів з самої колекції, таким чином досягається розподіл обов'язків: колекція відповідає за зберігання даних, ітератор – за прохід по колекції.

«Proxy» (Проксі) – об'єкти є об'єктами-заглушками або двійниками/замінниками для об'єктів конкретного типу. Зазвичай, проксі об'єкти вносять додатковий функціонал або спрощують взаємодію з реальними об'єктами. Проксі об'єкти використовувалися в більш ранніх версіях інтернет-браузерів, наприклад, для відображення картинки: поки картинка завантажується, користувачеві відображається «заглушка» картинки.

«State» (Стан) дозволяє змінювати логіку роботи об'єктів у випадку зміни їх внутрішнього стану. Наприклад, відсоток нарахованих на картковий рахунок грошей залежить від стану картки: Visa Electron, Classic, Platinum і т.д. Або обсяг послуг, які надані хостинг компанією, змінюється в залежності від обраного тарифного плану (стану членства – бронзовий, срібний або золотий клієнт).

«Strategy» (Стратегія) дозволяє змінювати деякий алгоритм поведінки об'єкта іншим алгоритмом, що досягає ту ж мету іншим способом. Прикладом можуть служити алгоритми сортування: кожен алгоритм має власну реалізацію і визначений в окремому класі; вони можуть бути взаємозамінними в об'єкті, який їх використовує.

Завдання:

- Ознайомитись з короткими теоретичними відомостями.
- Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

- Реалізувати один з розглянутих шаблонів за обраною темою.
- Реалізувати не менше 3-х класів відповідно до обраної теми.
- Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє використання шаблону в реалізації системи, навести фрагменти коду по реалізації цього шаблону.

Хід роботи:

Діаграма класів:

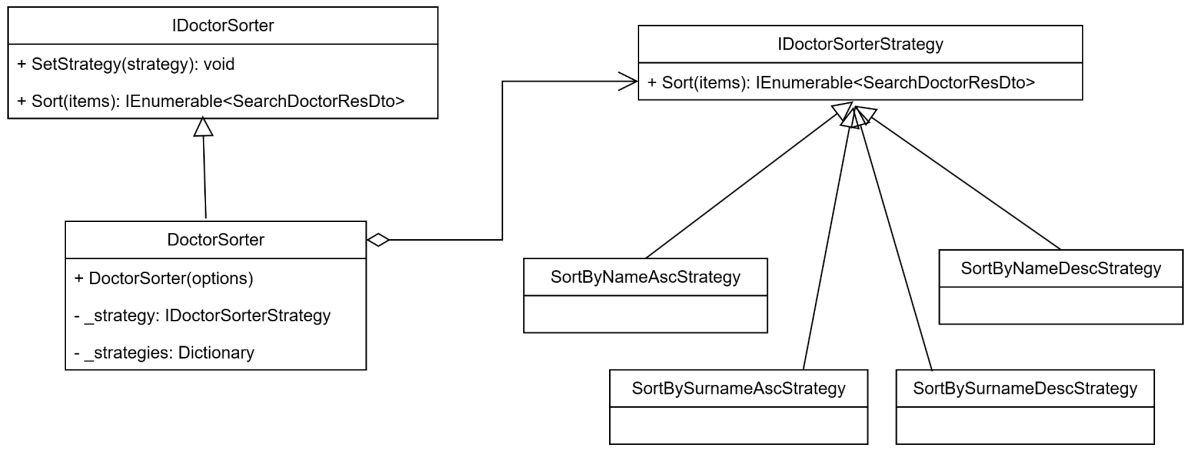


Рис.1 - Діаграма класів шаблону “Strategy”

На діаграмі:

- IDoctorSorter - Інтерфейс для класів, що сортують дані про лікарів.
- DoctorSorter - Конкретний екземпляр класу для сортування лікарів. Потрібна стратегія встановлюється через SetStrategy(strategy). Запуск стратегії сортування відбувається через Sort(items).
- IDoctorSorterStrategy - Інтерфейс для класів, що реалізують конкретні стратегії сортування даних лікарів. Містить метод Sort(items) для сортування.
- SortByNameAscStrategy, SortByNameDescStrategy, SortBySurnameAscStrategy, SortBySurnameDescStrategy - конкретні класи-реалізації стратегій для сортування даних лікарів.

```
using BookingClinic.Services.Data.Doctor;

namespace BookingClinic.Services.Helpers.DoctorsSortingHelper.DoctorSorter
{

    public interface IDoctorSorter
    {

        IEnumerable<SearchDoctorResDto> Sort(IEnumerable<SearchDoctorResDto> items);

        void SetStrategy(string? strategy);
    }
}
```

Рис.2 - Код IDoctorSorter

```
namespace BookingClinic.Services.Helpers.DoctorsSortingHelper.DoctorSorter
{
    Ссылок: 2
    public class DoctorSorter : IDoctorSorter
    {
        private readonly Dictionary<string, IDoctorSorterStrategy> _strategiesDict;

        private IDoctorSorterStrategy _strategy;

        Ссылок: 0
        public DoctorSorter(IOptions<DoctorSortingOptions> options)
        {
            _strategiesDict = options.Value.Strategies;
            _strategy = new NoStrategy();
        }

        Ссылок: 2
        public void SetStrategy(string? strategy)
        {
            if (!string.IsNullOrWhiteSpace(strategy) && _strategiesDict.ContainsKey(strategy))
            {
                _strategy = _strategiesDict[strategy];
            }
            else
            {
                _strategy = new NoStrategy();
            }
        }
```

Рис.3 - Код DoctorSorter

```
        Ссылок: 2
        public IEnumerable<SearchDoctorResDto> Sort(IEnumerable<SearchDoctorResDto> items)
        {
            return _strategy.Sort(items);
        }
    }
}
```

Рис.4 - Код DoctorSorter

```
namespace BookingClinic.Services.Helpers.DoctorsSortingHelper.DoctorSorterStrategies
{
    Ссылок: 10
    public interface IDoctorSorterStrategy
    {
        Ссылок: 6
        IEnumerable<SearchDoctorResDto> Sort(IEnumerable<SearchDoctorResDto> items);
    }
}
```

Рис.5 - Код IDoctorSorterStrategy

```
namespace BookingClinic.Services.Helpers.DoctorsSortingHelper.DoctorSorterStrategies
{
    public class NoStrategy : IDoctorSorterStrategy
    {
        public IEnumerable<SearchDoctorResDto> Sort(IEnumerable<SearchDoctorResDto> items) => items;
    }
}
```

Рис.6 - Код NoStrategy

```
namespace BookingClinic.Services.Helpers.DoctorsSortingHelper.DoctorSorterStrategies
{
    public class SortByNameAscStrategy : IDoctorSorterStrategy
    {
        Ссылок: 2
        public IEnumerable<SearchDoctorResDto> Sort(IEnumerable<SearchDoctorResDto> items) =>
            items.OrderBy(x => x.Name);
    }
}
```

Рис.7 - Код SortByNameAscStrategy

```
namespace BookingClinic.Services.Helpers.DoctorsSortingHelper.DoctorSorterStrategies
{
    public class SortByNameDescStrategy : IDoctorSorterStrategy
    {
        public IEnumerable<SearchDoctorResDto> Sort(IEnumerable<SearchDoctorResDto> items) =>
            items.OrderByDescending(x => x.Name);
    }
}
```

Рис.8 - Код SortByNameDescStrategy

```
namespace BookingClinic.Services.Helpers.DoctorsSortingHelper.DoctorSorterStrategies
{
    public class SortBySurnameAscStrategy : IDoctorSorterStrategy
    {
        public IEnumerable<SearchDoctorResDto> Sort(IEnumerable<SearchDoctorResDto> items) =>
            items.OrderBy(x => x.Surname);
    }
}
```

Рис.9 - Код SortBySurnameAscStrategy

```
namespace BookingClinic.Services.Helpers.DoctorsSortingHelper.DoctorSorterStrategies
{
    public class SortBySurnameDescStrategy : IDoctorSorterStrategy
    {
        public IEnumerable<SearchDoctorResDto> Sort(IEnumerable<SearchDoctorResDto> items) =>
            items.OrderByDescending(x => x.Surname);
    }
}
```

Рис.10 - Код SortBySurnameDescStrategy

```
namespace BookingClinic.Services.Options
{
    public class DoctorSortingOptions
    {
        public Dictionary<string, IDoctorSorterStrategy> Strategies { get; set; }
    }
}
```

Рис.11 - Код DoctorSortingOptions

```
services.Configure<DoctorSortingOptions>(cfg =>
{
    cfg.Strategies = new Dictionary<string, IDoctorSorterStrategy>();
    var strats = cfg.Strategies;

    strats["Name Asc"] = new SortByNameAscStrategy();
    strats["Name Desc"] = new SortByNameDescStrategy();
    strats["Surname Asc"] = new SortBySurnameAscStrategy();
    strats["Surname Desc"] = new SortBySurnameDescStrategy();
});
```

Рис.12 - Код налаштування DoctorSortingOptions

```
namespace BookingClinic.Controllers
{
    [Route("doctors")]
    Ссылок: 1
    public class DoctorController : Controller
    {
        private readonly IUserService _userService;
        private readonly ISpecialityService _specialityService;
        private readonly IClinicService _clinicService;
        private readonly IDoctorService _doctorService;
        private readonly IAppointmentService _appointmentService;
        private readonly IPaginationHelper<SearchDoctorResDto> _paginationHelper;
        private readonly IReviewsHelper _reviewsHelper;
        private readonly IDoctorSorter _doctorSorter;
        private readonly Dictionary<string, IDoctorSorterStrategy> _docSortingStrategies;

        Ссылок: 0
        public DoctorController(
            IUserService userService,
            IPaginationHelper<SearchDoctorResDto> paginationHelper,
            ISpecialityService specialityService,
            IClinicService clinicService,
            IDoctorService doctorService,
            IAppointmentService appointmentService,
            IReviewsHelper reviewsHelper,
            IDoctorSorter doctorSorter,
            IOptions<DoctorSortingOptions> docSortingStrategies)...

        [HttpGet]
        Ссылок: 0
        public IActionResult Index([FromQuery] SearchDoctorDto dto, [FromQuery] int page)
```

Рис.13 - Код DoctorController

```
        [HttpGet]
        Ссылок: 0
        public IActionResult Index([FromQuery] SearchDoctorDto dto, [FromQuery] int page)
        {
            var res = _userService.SearchDoctors(dto);
            var specialities = _specialityService.GetSpecialityNames();
            var clinics = _clinicService.GetClinicNames();

            if (specialities.IsSuccess)
            {
                ViewData["Specialities"] = specialities.Result;
            }

            if (clinics.IsSuccess)
            {
                ViewData["Clinics"] = clinics.Result;
            }

            ViewData["Sortings"] = _docSortingStrategies.Keys.ToList();
            ViewData["Page"] = page;

            if (res.IsSuccess)
            {
                _doctorSorter.SetStrategy(dto.OrderBy);

                var orderDoctors = _doctorSorter.Sort(res.Result!);

                var doctors = _paginationHelper.Paginate(orderDoctors, page, 5, out var pages);

                ViewData["Pages"] = pages;
```

Рис.14 - Код DoctorController

```
            ViewData["Pages"] = pages;
            ViewData["Doctors"] = doctors;

            return View(dto);
        }
        else
        {
            ViewData["Page"] = 1;
            ViewData["Pages"] = new List<int> { 1 };
            ViewData["Errors"] = res.Errors;
            ViewData["Doctors"] = new List<SearchDoctorResDto>() { };

            return View(dto);
        }
    }

    [HttpGet("{id:guid}")]
    Ссылок: 0
    public IActionResult Profile([FromRoute] Guid id)
    {
        if (TempData["Errors"] != null)
        {
            ViewData["Errors"] = JsonSerializer.Deserialize<List<ServiceError>>(TempData["Errors"].ToString());
        }

        if (TempData["ReviewErrors"] != null)
        {
            List<ValidationFailure> failures =
                JsonSerializer.Deserialize<List<ValidationFailure>>(TempData["ReviewErrors"].ToString());

            var valRes = new ValidationResult(failures);
            valRes.AddToModelState(ModelState);
        }
```

Рис.15 - Код DoctorController

```
        var res = _doctorService.GetDoctorData(id);

        if (res.IsSuccess)
        {
            if (User.IsInRole("Patient") || User.IsInRole("Admin"))
            {
                res.Result!.CanWriteReview = _reviewsHelper.CanUserWriteReview(id, User);
            }

            return View(res.Result);
        }
        else
        {
            ViewData["Errors"] = res.Errors;
            return View();
        }
    }

    [HttpPost]
    [Authorize("AuthUser")]
    Ссылок: 0
    public async Task<IActionResult> MakeAppointment(
        [FromForm] MakeAppointmentDto dto)
    {
        var res = await _appointmentService.CreateAppointment(dto, User);

        if (!res.IsSuccess)
        {
            TempData["Errors"] = JsonSerializer.Serialize(res.Errors);
            return RedirectToAction("Profile", new { id = dto.DoctorId });
        }
        return RedirectToAction("Index", "User");
    }
```

Рис.16 - Код DoctorController

```
        [HttpPost("docApp")]
        [Authorize("Doctors")]
        Ссылок: 0
        public async Task<IActionResult> MakeAppointmentDoctor(
            [FromForm] MakeAppointmentDocDto dto)
        {
            var res = await _appointmentService.CreateAppointmentDoctor(dto, User);

            if (res.IsSuccess)
            {
                return RedirectToAction("FinishedAppointment", "Appointment", new { dto.PatientId });
            }
            else
            {
                TempData["Errors"] = JsonSerializer.Serialize(res.Errors);
                return RedirectToAction("Index", "Appointment");
            }
        }
    }
}
```

Рис.17 - Код DoctorController

```
@using BookingClinic.Services.Data.Doctor
@model SearchDoctorDto;
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@{
    ViewData["Title"] = "Doctors";
    var data = (IEnumerable<SearchDoctorResDto>)ViewData["Doctors"];
    var pages = (IEnumerable<int>)ViewData["Pages"];
    var currentPage = (int)ViewData["Page"];
    var specialities = (IEnumerable<string>)ViewData["Specialities"];
    var clinics = (IEnumerable<string>)ViewData["Clinics"];
    var queryString = Context.Request.QueryString.Value;
    var sortings = (List<string>)ViewData["Sortings"];

    if (!string.IsNullOrEmpty(queryString))
    {
        var vals = queryString.Split('&');
        var filterVals = vals.Where(val => !val.Contains("page="));
        queryString = string.Join('&', filterVals);
    }
}

<form class="row g-2 align-items-center mb-4" method="get">
    <div class="col-md-4">
        <input type="text" class="form-control" asp-for="Query" placeholder="Name and surname" />
    </div>
    <div class="col-md-2">
        <select class="form-select" asp-for="Speciality">
            <option value="">Speciality</option>
            @foreach (var s in specialities)
            {
                <option value="@s">@s</option>
            }
        </select>
    </div>
    <div class="col-md-2">
        <select class="form-select" asp-for="Clinic">
            <option value="">Clinic</option>
            @foreach (var c in clinics)
            {
                <option value="@c">@c</option>
```

Рис.18 - Код Index.cshtml

```
                <option value="@c">@c</option>
            }
        </select>
    </div>
    <div class="col-md-2">
        <select class="form-select" asp-for="OrderBy">
            <option value="">Sort by</option>
            @foreach (var s in sortings)
            {
                <option value="@s">@s</option>
            }
        </select>
    </div>
    <div class="col-md-2">
        <button type="submit" class="btn btn-primary w-100">
            <img src="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/icons/search.svg" alt="Search" width="20" height="20" />
        </button>
    </div>
</form>
<div class="row row-cols-1 gy-4">
    @foreach (var doc in data)
    {
        <div class="col">
            <div class="card p-3 flex-row align-items-center">
                <div class="me-4">
                    <img src="@(doc.ProfilePicture == null ? Url.Content("~/profiles/default/defaultUser.png") : Url.Content("~/profiles/default/" +
                        alt="Doctor"
                        class="rounded-circle"
                        style="width: 96px; height: 96px; object-fit: cover;" />
                </div>
                <div class="flex-grow-1">
                    <div><strong>Name:</strong> @doc.Name</div>
                    <div><strong>Surname:</strong> @doc.Surname</div>
                    <div><strong>Clinic:</strong> @doc.Clinic</div>
                    <div><strong>Speciality:</strong> @doc.Speciality</div>
                    <a class="btn btn-outline-primary mt-2" href="~/doctors/@doc.Id">Go to doctor</a>
                </div>
            </div>
        </div>
    }
```

Рис.19 - Код Index.cshtml

```
    }
</div>
<nav class="mt-4">
    <ul class="pagination justify-content-center">
        @foreach (var p in pages)
        {
            var href = string.Empty;

            if (string.IsNullOrEmpty(queryString))
            {
                href = $"./doctors?page={p}";
            }
            else
            {
                href = $"./doctors{queryString}&page={p}";
            }
            @* if (!string.IsNullOrWhiteSpace(Model.OrderBy))
            {
                href += $"&ordering={Model.OrderBy}";
            } *@
            <li class="page-item @(p == currentPage ? "active" : "")">
                <a class="page-link" href="@href">@p</a>
            </li>
        }
    </ul>
</nav>
```

Рис.20 - Код Index.cshtml

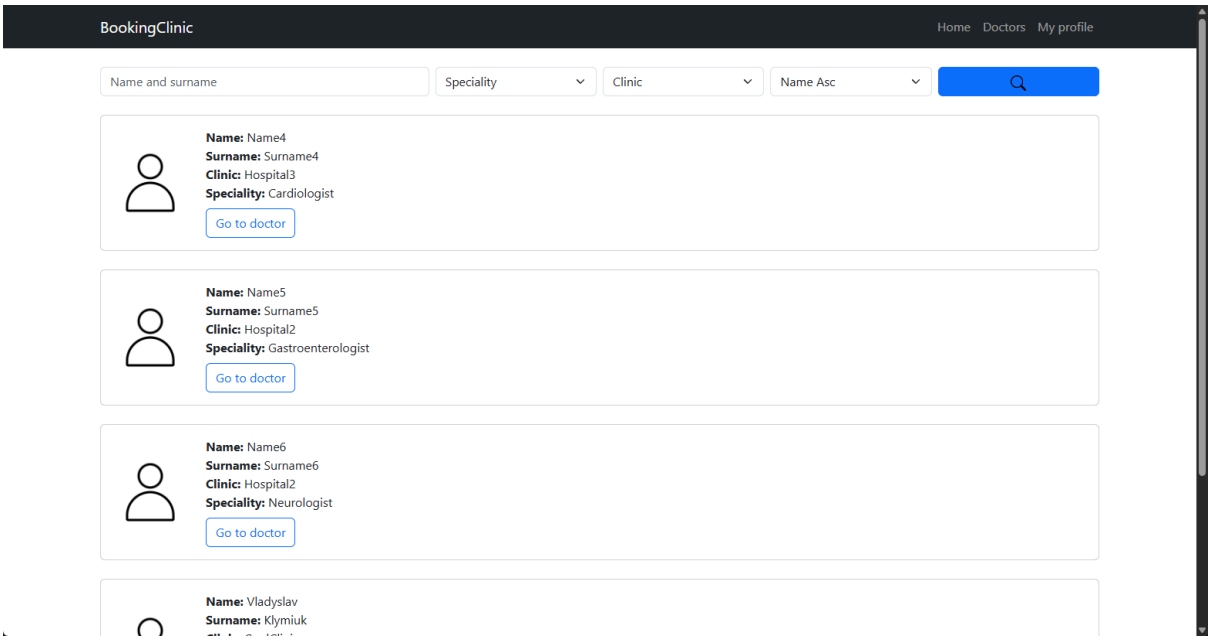


Рис.21 - Сортування за іменем (зростання)

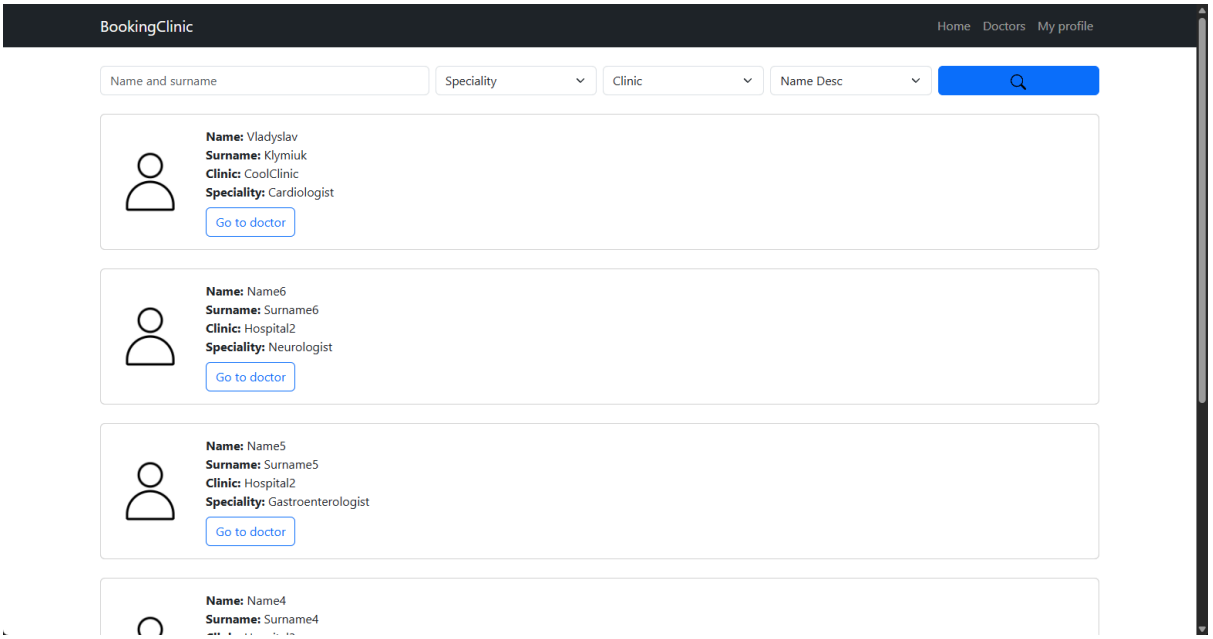


Рис.22 - Сортування за іменем (спадання)

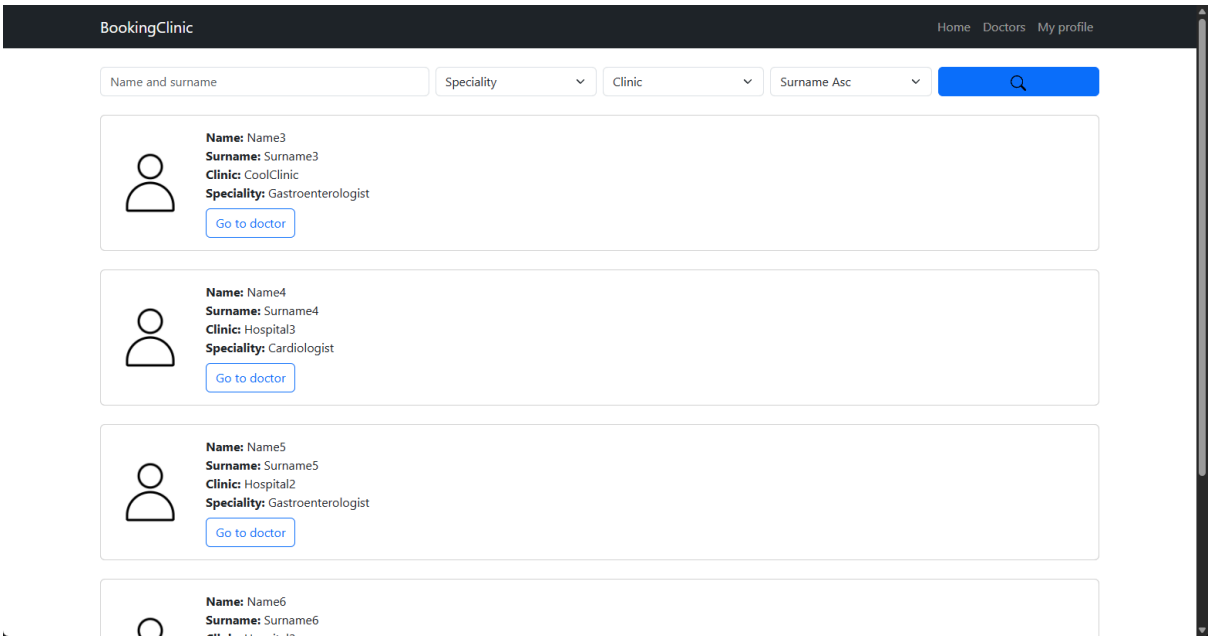


Рис.23 - Сортування за фамілією (зростання)

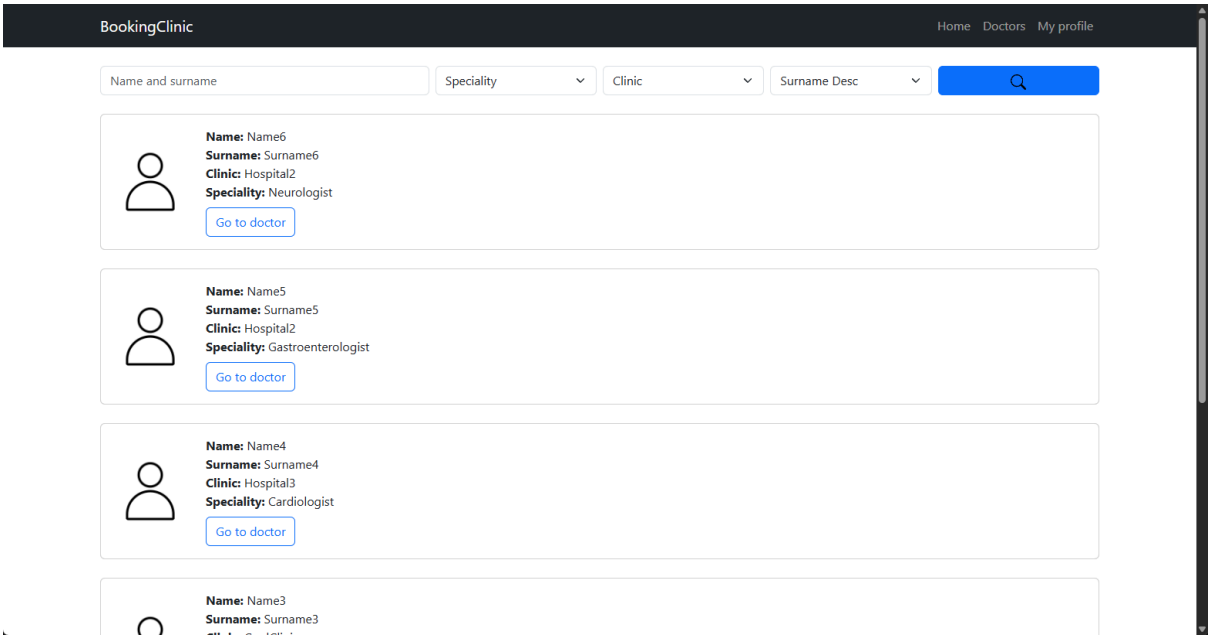


Рис.24 - Сортування за фамілією (спадання)

Контрольні питання:

1. Що таке шаблон проєктування?

Шаблони представляють собою підтверджені роками розробки в різних компаніях і на різних проєктах, «ескізи» архітектурних рішень, які зручно застосовувати у відповідних обставинах.

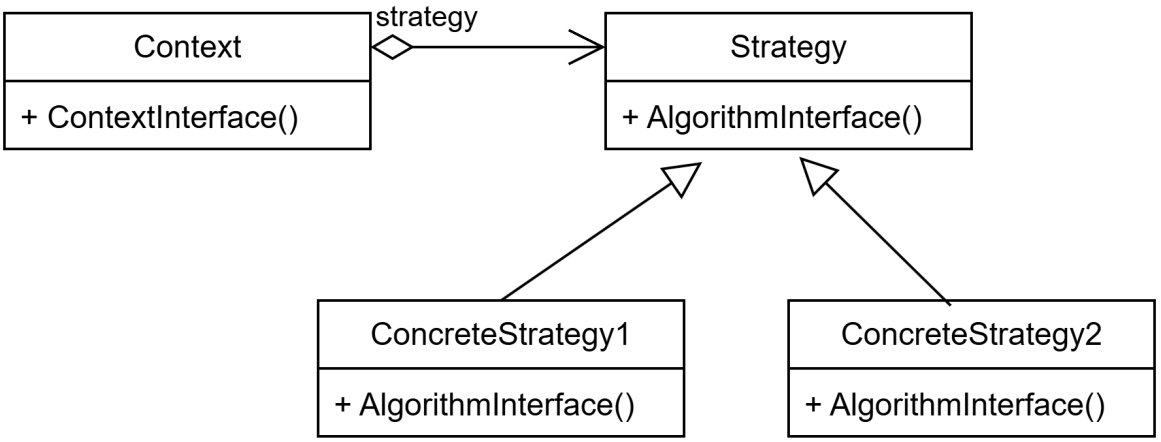
2. Навіщо використовувати шаблони проєктування?

Застосування патернів проєктування підвищує стійкість системи до зміни вимог та спрощує подальше доопрацювання системи.

3. Яке призначення шаблону «Стратегія»?

Щоб реалізувати можливість виконати якусь дію за заданим алгоритмом.

4. Нарисуйте структуру шаблону «Стратегія».



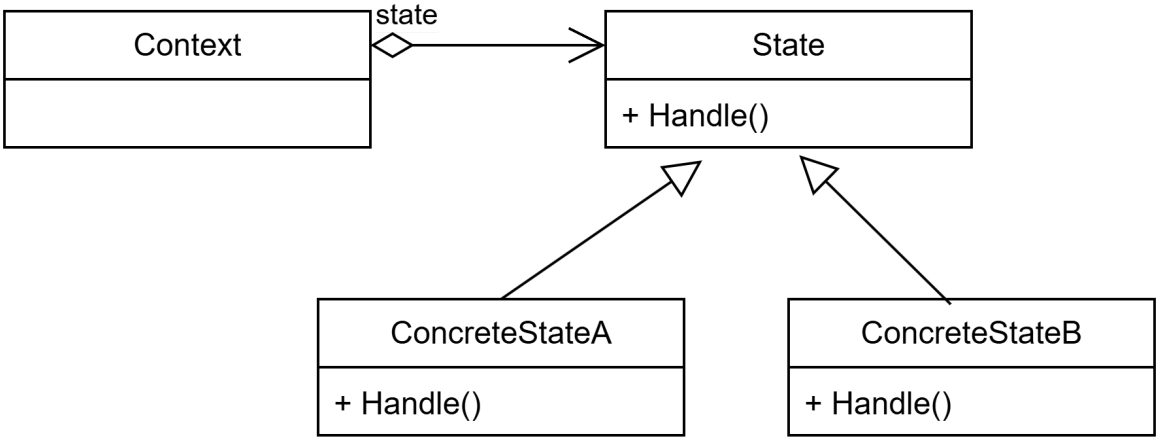
5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

Context - містить в собі базовий клас стратегії. Викликає AlgorithmInterface() класу Strategy.
Strategy - базовий клас стратегії.
ConcreteStrategy1,2 - конкретні реалізації Strategy зі своїми алгоритмами роботи.

6. Яке призначення шаблону «Стан»?

Дозволяє змінювати логіку роботи об'єктів у випадку зміни їх внутрішнього стану.

7. Нарисуйте структуру шаблону «Стан».



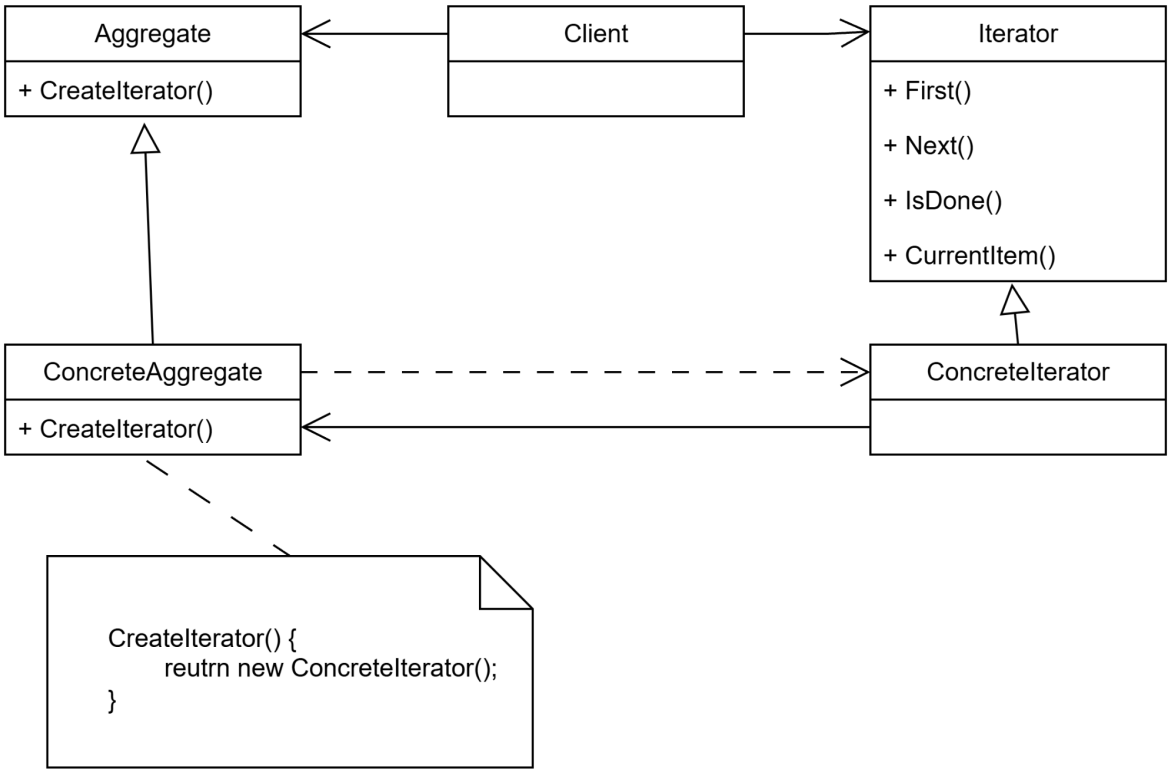
8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

Context - містить клас State та викликає його метод Handle().
State - базовий клас стану.
ConcreteStateA,B - конкретні реалізації State, які описують різні алгоритми роботи для кожного зі станів.

9. Яке призначення шаблону «Ітератор»?

Ітератор виносить функціональність перебору колекції елементів із самої колекції.

10. Нарисуйте структуру шаблону «Ітератор».



11. Які класи входять в шаблон «Ітератор», та яка між ними взаємодія?

Client - містить клас Aggregate і Iterator для обходу колекції. Викликає CreateIterator() щоб отримати ітератор для колекції. У ітератора викликає First(), Next(), IsDone(), CurrentItem() для ітерації по колекції.

Aggregate - базовий клас колекції.
ConcreteAggregate - конкретна реалізація Aggregate, яка створює свій конкретний ітератор.
Iterator - базовий клас ітератора.
ConcreteIterator - конкретна реалізація Iterator для певної колекції.

12. В чому полягає ідея шаблона «Одинак»?

Являє собою клас, який може мати не більше одного об'єкта.

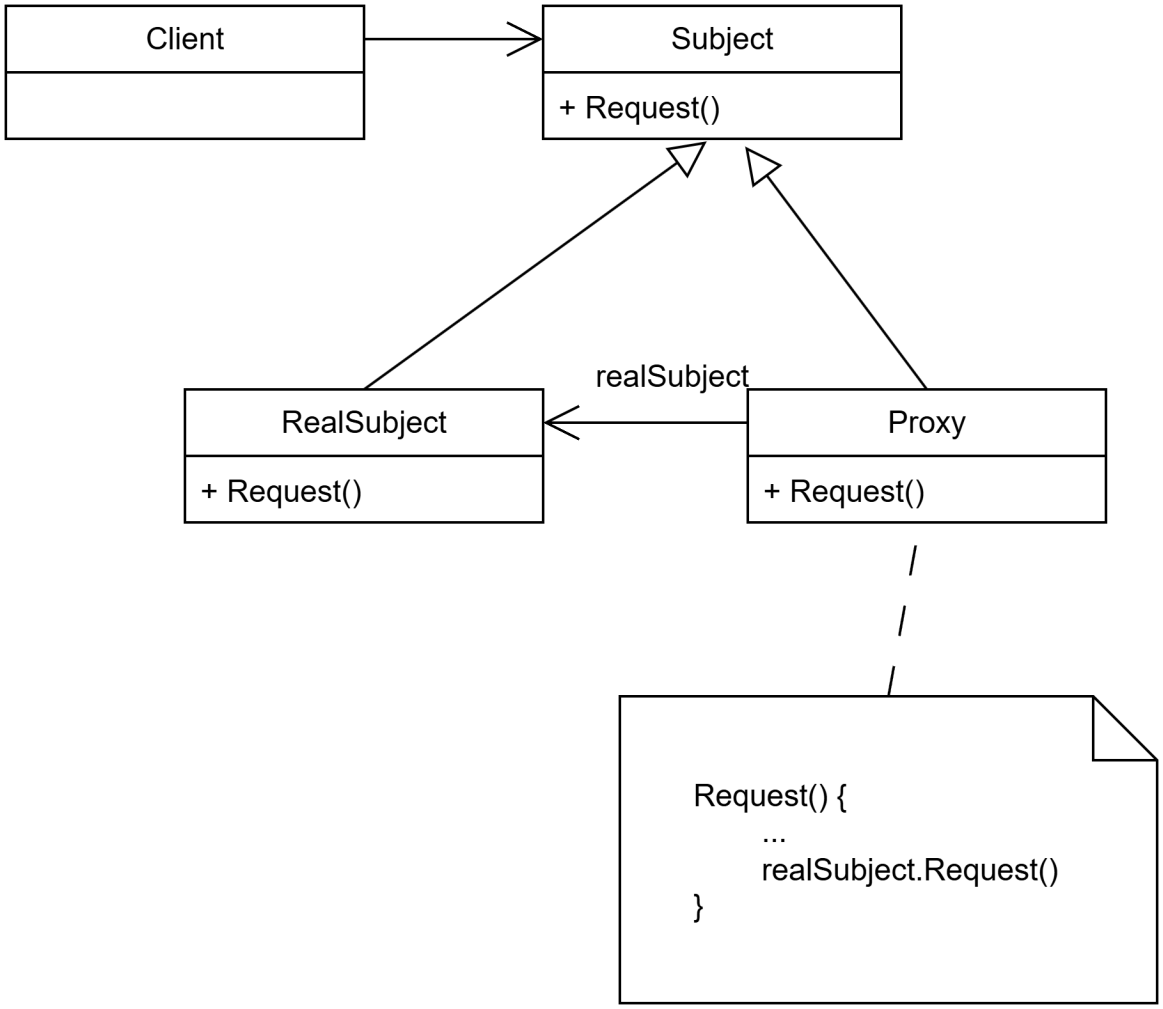
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?

Бо “одинак” представляє собою глобальний об’єкт, стан якого складно відслідковувати і коректно підтримувати. Також глобальні об'єкти важко тестуються і вносять складність в програмний код.

14. Яке призначення шаблону «Проксі»?

Проксі об'єкти вносять додатковий функціонал або спрощують взаємодію з реальними об'єктами. Такі об'єкти є об'єктами-заглушками або двійниками/замінниками для об'єктів конкретного типу.

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?

Client - клієнтський клас, що викликає метод суб'єкта Request().
Subject - базовий клас суб'єкта.
RealSubject - реальний клас-суб'єкт.
Proxy - суб'єкт, що містить у собі RealSubject і контролює його роботу.

Висновки:

У цій лабораторній роботі я ознайомився з поняттям шаблонів проектування та розглянув такі шаблони як: "Одинак", "Ітератор", "Стратегія", "Проксі", "Стан". Вивчене дало змогу зрозуміти способи застосування цих шаблонів. Також для проекту було реалізовано поведінку шаблону "Стратегія" на прикладі використання різних стратегій для сортування лікарів при пошуку.