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 №7

із дисципліни «*Технології розробки програмного забезпечення*»

Тема: «Патерни 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 «Mediator», «Facade», «Bridge», «Template method» та навчитися застосовувати їх в реалізації програмної системи.

Теоретичні відомості:

Шаблон «Mediator» (посередник) використовується для визначення взаємодії об'єктів за допомогою іншого об'єкта (замість зберігання посилань один на одного). Даний шаблон схожий на шаблон «команда», проте в даному випадку замість зберігання даних про конкретну дію, зберігаються дані про взаємодії між компонентами. Даний шаблон зручно застосовувати у випадках, коли безліч об'єктів взаємодіє між собою деяким структурованим чином, однак складним для розуміння. У такому випадку вся логіка взаємодії виноситься в окремий об'єкт. Кожен із взаємодіючих об'єктів зберігає посилання на об'єкт «медіатор». «Медіатор» нагадує диригента при управлінні оркестром. Диригент стежить за тим, щоб кожен інструмент грав в правильний час і в злагоді з іншими інструментами. Функції «медіатора» повністю це повторюють.

Переваги та недоліки:

+ Організація взаємодії між об’єктами лише через посередника спрощує розуміння та супроводження такого коду.

+ Додавання нових посередників без зміни існуючих компонентів дозволяє розширювати систему без зміни існуючого коду.

+ Зменшення залежностей між об’єктами підвищує гнучкість системи.

- Посередник, з часом, може перетворитися на дуже складний об’єкт, який робить все («God Object»).

Шаблон «Facade» (фасад) передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми. Оскільки підсистема може складатися з безлічі класів, а кількість її функцій – не більше десяти, то щоб уникнути створення «спагеті-коду» (коли все тісно пов'язано між собою) виділяють один загальний інтерфейс доступу, здатний правильним чином звертатися до внутрішніх деталей. Це також відволікає користувачів від змін в підсистемі (внутрішня реалізація може змінюватися, а наданої послуги немає), що також скоротить кількість змін в використовуваних фасад класах (без фасаду довелося б змінювати вихідні коди в безлічі точок).

Переваги та недоліки:

+ Інкапсуляція внутрішньої структури від клієнтського коду.

+ Спрощується інтерфейс для роботи з модулем закритим фасадом.

- Зниження гнучкості в налаштуванні та використанні програмного коду закритого фасадом.

Шаблон «Bridge» (міст) використовується для поділу інтерфейсу і його реалізації. Це необхідно у випадках, коли може існувати кілька різних абстракцій, над якими можна проводити дії різними способами.

Переваги та недоліки:

+ Дозволяє змінювати ієрархії абстракції та реалізації незалежно одна від одної.

+ Розділивши абстракцію від реалізації отримуємо більшу гнучкість та простіший супровід такого коду.

- Підвищена гнучкість при використанні патерну отримується за рахунок більшої складності, введення додаткових проміжних рівнів.

Шаблон «Template Method» (шаблонний метод) дозволяє реалізувати покроково алгоритм в абстрактному класі, але залишити специфіку реалізації підкласам. Можна привести в приклад формування вебсторінки: необхідно додати заголовки, вміст сторінки, файли, що додаються, і нижню частину сторінки. Код для додавання вмісту сторінки може бути абстрактним і реалізовуватися в різних класах – AspNetCompiler, HtmlCompiler, PhpCompiler і т.п. Додавання всіх інших елементів виконується за допомогою вихідного абстрактного класу з алгоритмом. Даний шаблон дещо нагадує шаблон «Фабричний метод», однак область його використання абсолютно інша – для покрокового визначення

конкретного алгоритму; більш того, даний шаблон не обов'язково створює нові об'єкти – лише визначає послідовність дій.

Переваги та недоліки:

+ Полегшує повторне використання коду.

- Ви жорстко обмежені скелетом існуючого алгоритму.

- Ви можете порушити принцип підстановки Барбари Лісков, змінюючи базову поведінку одного з кроків алгоритму через підклас.

- З ростом складності загального алгоритму шаблонний метод стає занадто складно підтримувати, особливо, коли є багато віртуальних методів для перевизначення в підкласах.

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:

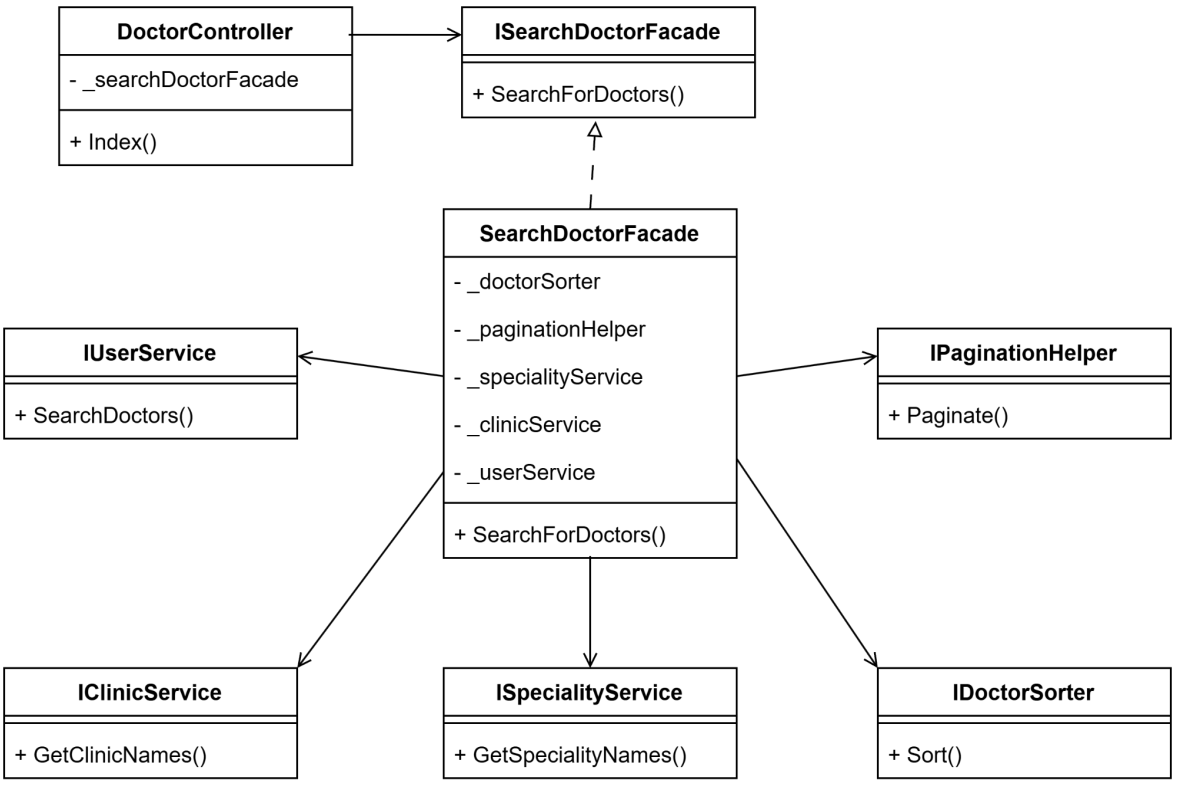


Рис.1 - Діаграма класів шаблону “Фасад”

На цій діаграмі:
DoctorController - клас-контролер, який обробляє вхідний запит та викликає клас фасад для отримання даних лікарів.
ISearchDoctorFacade - інтерфейс класа-фасада для отримання даних лікарів.
SearchDoctorFacade - клас-фасад для отримання даних лікарів. Містить в собі сукупність інтерфейсів класів для отримання даних лікарів, клінік та спеціальностей; сортує дані лікарів; робить пагінацію.
IUserService - інтерфейс для роботи з даними користувачів.
IClinicService - інтерфейс для роботи з даними лікарень.
ISpecialityService - інтерфейс для роботи з даними спеціальностей.
IDoctorSorter - інтерфейс для сортування даних лікарів.
IPaginationHelper - інтерфейс для пагінації даних лікарів.

Вихідний код класів:

```
using BookingClinic.Services;
using BookingClinic.Services.Appointment;
using BookingClinic.Services.Data.Appointment;
using BookingClinic.Services.Data.Doctor;
using BookingClinic.Services.Doctor;
using BookingClinic.Services.Doctor.Facade;
using BookingClinic.Services.Helpers.ReviewsHelper;
using FluentValidation.AspNetCore;
using FluentValidation.Results;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System.Text.Json;

namespace BookingClinic.Controllers
{
    [Route("doctors")]
    Ссылок: 1
    public class DoctorController : Controller
    {
        private readonly IDoctorService _doctorService;
        private readonly IAppointmentService _appointmentService;
        private readonly IReviewsHelper _reviewsHelper;
        private readonly ISearchDoctorFacade _searchDoctorFacade;

        Ссылок: 0
        public DoctorController(
            IDoctorService doctorService,
            IAppointmentService appointmentService,
            IReviewsHelper reviewsHelper,
            ISearchDoctorFacade searchDoctorFacade)
        {
            _doctorService = doctorService;
            _appointmentService = appointmentService;
            _reviewsHelper = reviewsHelper;
            _searchDoctorFacade = searchDoctorFacade;
        }

        [HttpGet]
        Ссылок: 0
        public IActionResult Index([FromQuery] SearchDoctorDto dto, [FromQuery] int page)
        {
            var res = _searchDoctorFacade.SearchForDoctors(dto, page);

            ViewData["Specialities"] = res.Specialities;
            ViewData["Clinics"] = res.Clinics;
            ViewData["Sortings"] = res.Sortings;
```

Рис.2 - Код класу DoctorController

```
    ViewData["Page"] = res.Page;
    ViewData["Pages"] = res.Pages;
    ViewData["Doctors"] = res.Doctors;
    ViewData["Errors"] = res.Errors;

    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);
    }

    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")]
```

Рис.3 - Код класу DoctorController

```
        [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");
        }

        [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.4 - Код класу DoctorController

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

namespace BookingClinic.Services.Doctor.Facade
{
    Ссылок: 4
    public interface ISearchDoctorFacade
    {
        Ссылок: 2
        SearchDoctorIndexResult SearchForDoctors(SearchDoctorDto dto, int page);
    }
}
```

Рис.5 - Код інтерфейсу ISearchDoctorFacade

```
using BookingClinic.Services.Clinic;
using BookingClinic.Services.Data.Doctor;
using BookingClinic.Services.Helpers.DoctorsSortingHelper.DoctorSorter;
using BookingClinic.Services.Helpers.DoctorsSortingHelper.DoctorSorterStrategies;
using BookingClinic.Services.Helpers.PaginationHelper;
using BookingClinic.Services.Options;
using BookingClinic.Services.Speciality;
using BookingClinic.Services.UserService;
using Microsoft.Extensions.Options;

namespace BookingClinic.Services.Doctor.Facade
{
    Ссылок: 2
    public class SearchDoctorFacade : ISearchDoctorFacade
    {
        private readonly IDoctorSorter _doctorSorter;
        private readonly IPaginationHelper<SearchDoctorResDto> _paginationHelper;
        private readonly ISpecialityService _specialityService;
        private readonly IClinicService _clinicService;
        private readonly IUserService _userService;
        private readonly Dictionary<string, IDoctorSorterStrategy> _docSortingStrategies;

        Ссылок: 0
        public SearchDoctorFacade(
            IDoctorSorter doctorSorter,
            IPaginationHelper<SearchDoctorResDto> paginationHelper,
            ISpecialityService specialityService,
            IClinicService clinicService,
            IUserService userService,
            IOptions<DoctorSortingOptions> docSortingStrategies)
        {
            _doctorSorter = doctorSorter;
            _paginationHelper = paginationHelper;
            _specialityService = specialityService;
            _clinicService = clinicService;
            _userService = userService;
            _docSortingStrategies = docSortingStrategies.Value.Strategies;
        }

        Ссылок: 2
        public SearchDoctorIndexResult SearchForDoctors(SearchDoctorDto dto, int page)
        {
            var doctorsRes = _userService.SearchDoctors(dto);
            var specialitiesRes = _specialityService.GetSpecialityNames();
            var clinicsRes = _clinicService.GetClinicNames();

            List<ServiceError> errors = new();
```

Рис.6 - Код класу SearchDoctorFacade

```
            List<ServiceError> errors = new();
            var doctors = doctorsRes.Result!;

            if (!doctorsRes.IsSuccess)
            {
                errors.AddRange(doctorsRes.Errors);
                doctors = Enumerable.Empty<SearchDoctorResDto>();
            }

            _doctorSorter.SetStrategy(dto.OrderBy);
            doctors = _doctorSorter.Sort(doctors);
            doctors = _paginationHelper.Paginate(doctors, page, 5, out var pages);

            if (!specialitiesRes.IsSuccess)
            {
                errors.AddRange(specialitiesRes.Errors);
            }

            if (!clinicsRes.IsSuccess)
            {
                errors.AddRange(clinicsRes.Errors);
            }

            var res = new SearchDoctorIndexResult()
            {
                IsSuccess = errors.Count == 0,
                Doctors = doctors,
                Clinics = clinicsRes.IsSuccess ? clinicsRes.Result! : Enumerable.Empty<string>(),
                Specialities = specialitiesRes.IsSuccess ? specialitiesRes.Result! : Enumerable.Empty<string>(),
                Errors = errors,
                Page = (!doctorsRes.IsSuccess || page == 0) ? 1 : page,
                Sortings = _docSortingStrategies.Keys.ToList(),
                Pages = pages,
            };

            return res;
        }
    }
}
```

Рис.7 - Код класу SearchDoctorFacade

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

namespace BookingClinic.Services.Helpers.DoctorsSortingHelper.DoctorSorter
{
    Ссылок: 4
    public interface IDoctorSorter
    {
        Ссылок: 2
        IEnumerable<SearchDoctorResDto> Sort(IEnumerable<SearchDoctorResDto> items);
        Ссылок: 2
        void SetStrategy(string? strategy);
    }
}
```

Рис.8 - Код інтерфейсу IDoctorSorter

```
namespace BookingClinic.Services.Helpers.PaginationHelper
{
    Ссылок: 8
    public interface IPaginationHelper<T>
    {
        Ссылок: 2
        int GetTotalPages(IEnumerable<T> entities, int pageSize);
        Ссылок: 4
        IEnumerable<T> Paginate(IEnumerable<T> entities, int page, int pageSize, out List<int> pages);
    }
}
```

Рис.9 - Код інтерфейсу IPaginationHelper

```
namespace BookingClinic.Services.Speciality
{
    Ссылок: 4
    public interface ISpecialityService
    {
        Ссылок: 2
        ServiceResult<IEnumerable<string>> GetSpecialityNames();
    }
}
```

Рис.10 - Код інтерфейсу ISpecialityService

```
namespace BookingClinic.Services.Clinic
{
    Ссылок: 4
    public interface IClinicService
    {
        Ссылок: 2
        ServiceResult<IEnumerable<string>> GetClinicNames();
    }
}
```

Рис.11 - Код інтерфейсу IClinicService

```
using BookingClinic.Services.Data.Doctor;
using BookingClinic.Services.Data.User;
using System.Security.Claims;

namespace BookingClinic.Services.UserService
{
    public interface IUserService
    {
        ServiceResult<IEnumerable<SearchDoctorResDto>> SearchDoctors(SearchDoctorDto dto);

        ServiceResult<ClaimsPrincipal> LoginUser(LoginUserDto dto);

        Task<ServiceResult<ClaimsPrincipal>> RegisterUser(RegisterUserDto dto);
        Ссылок: 2
        ServiceResult<UserPageDataDto> GetUserData(ClaimsPrincipal userPrincipal);
        Ссылок: 2
        Task<ServiceResult<object>> UpdateUserPhoto(IFormFile file, ClaimsPrincipal principal);
        Ссылок: 2
        Task<ServiceResult<UserPageDataDto>> UpdateUser(UserPageDataUpdateDto dto, ClaimsPrincipal principal);
    }
}
```

Рис.12 - Код інтерфейсу IUserService

Висновки:

У цій лабораторній роботі я ознайомився з новими шаблонами проектування. “Посередник” дає можливість винести логіку взаємодії між компонентами до класа-посередника, що зменшує між ними зв’язність;

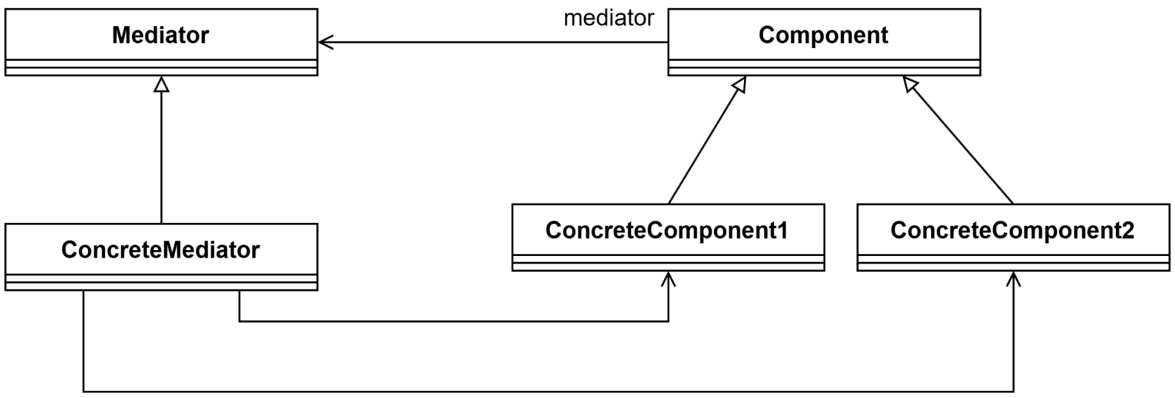
“Фасад” надає зручний та простий інтерфейс для взаємодії з класами якоїсь підсистеми, замість того щоб вручну налаштовувати між ними взаємодію; “Міст” розділяє абстракцію та реалізацію, щоб зробити ієрархії класів більш гнучкими та простими; “Шаблонний метод” визначає структуру алгоритму та дає своїм нащадкам перевизначати окремі кроки алгоритму. Я обрав та реалізував шаблон “Фасад” на прикладі сервісу для отримання даних лікарів, який використовує різні класи для отримання даних, їхнього сортування, пагінації та пошуку за запитом. Вивчене покращило моє розуміння патернів проектування та знадобиться в подальшій розробці.

Контрольні питання:

1. Яке призначення шаблону «Посередник»?

Цей шаблон вводить додаткового об’єкта-посередника, який визначає взаємодію між ними замість того щоб давати їм можливість працювати напряму. Схожий на шаблон “Команда”, але не зберігає дані про конкретну дію, а лише зберігає дані про взаємодію. Зручно застосовувати у випадках, коли безліч об'єктів взаємодіє між собою деяким структурованим чином, однак складним для розуміння.

2. Нарисуйте структуру шаблону «Посередник».



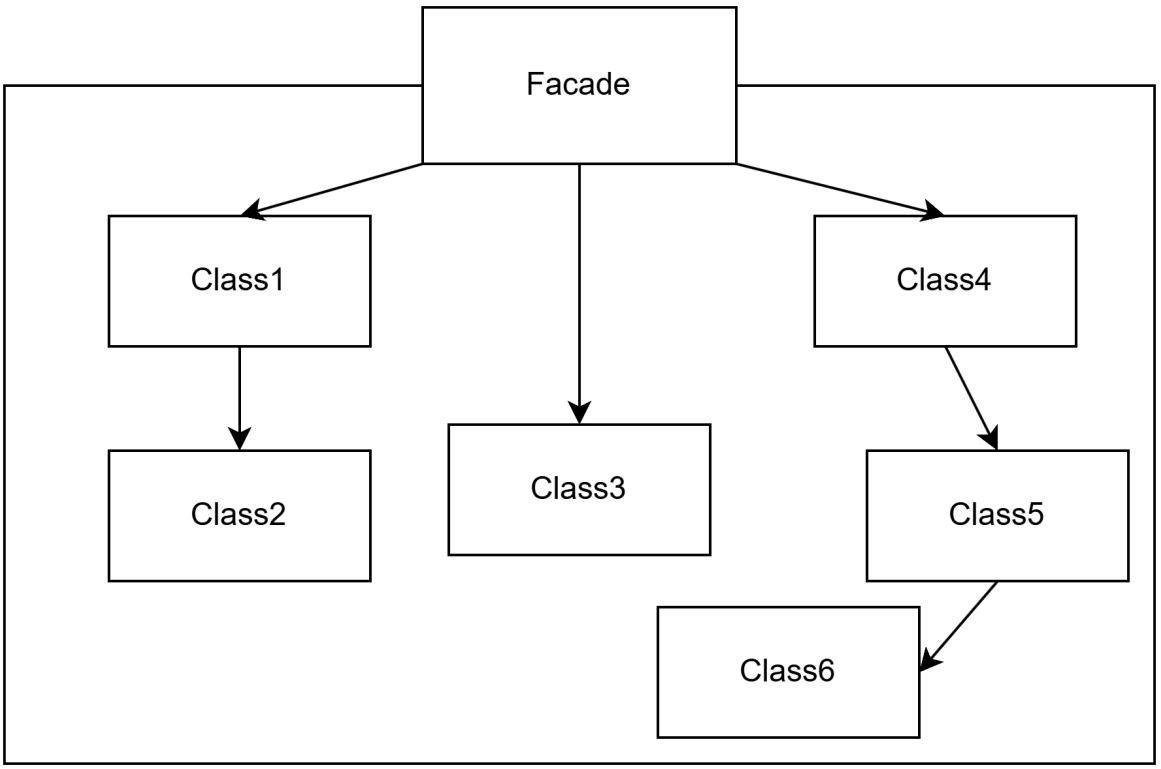
3. Які класи входять в шаблон «Посередник», та яка між ними взаємодія?

Component - Базовий клас якогось компонента, який буде звертатись до медіатора для делегації виклику. Містить посилання на медіатора для делегації йому взаємодії з іншими компонентами.
ConcreteComponent1,2 - Конкретні реалізації компонентів.
Mediator - Базовий клас медіатора. Компоненти будуть викликати його, щоб він обирав як робити між ними взаємодію.
ConcreteMediator - Конкретна реалізація медіатора.

4. Яке призначення шаблону «Фасад»?

Фасад передбачає створення єдиного уніфікованого способу доступу до підсистеми без розкриття внутрішніх деталей підсистеми. Якщо система складається з безлічі класів, то є сенс виділити загальний інтерфейс-фасад, який здатний правильним чином звертатись до внутрішніх деталей. Це також відволікає користувачів від змін у підсистемі, що також скоротить кількість змін в використовуваних фасад класах.

5. Нарисуйте структуру шаблону «Фасад».



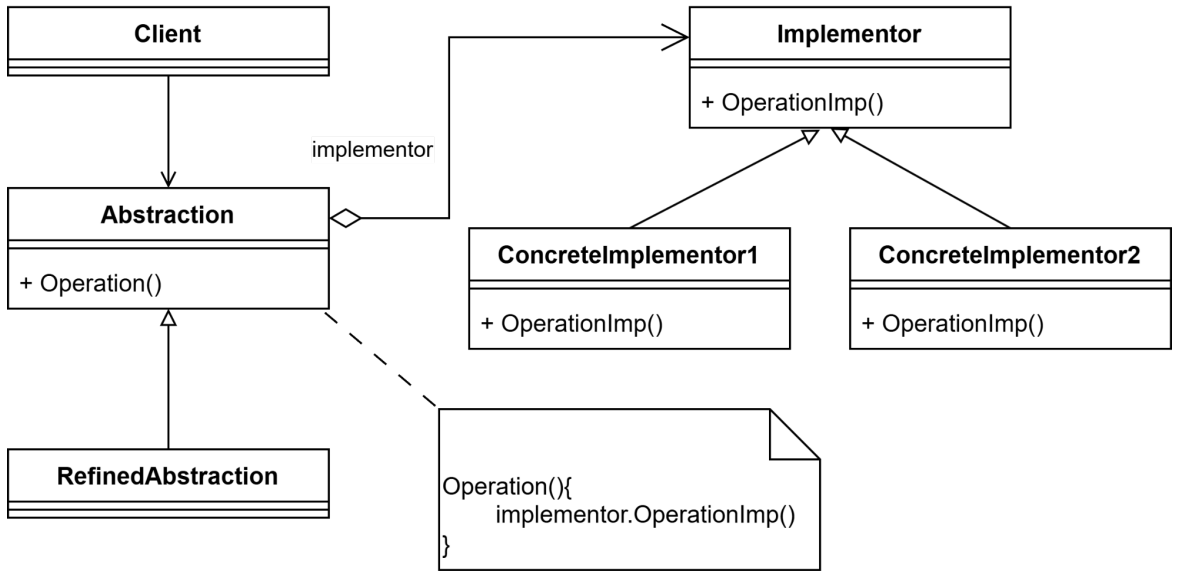
6. Які класи входять в шаблон «Фасад», та яка між ними взаємодія?

Facade - клас що надає інтерфейс до підсистеми, яка містить в собі безліч різних інших класів.
Class1,2,3,4,5,6 - класи підсистеми, що виконують різні функції. Замість того щоб працювати з кожним з них окремо виклики до них об’єднуються у фасаді.

7. Яке призначення шаблону «Міст»?

Міст застосовується для розділення інтерфейсу та реалізації. Такий підхід дозволяє спрощувати ієрархії класів і дозволяє абстракції та реалізації розвиватись незалежно один від одного.

8. Нарисуйте структуру шаблону «Міст».



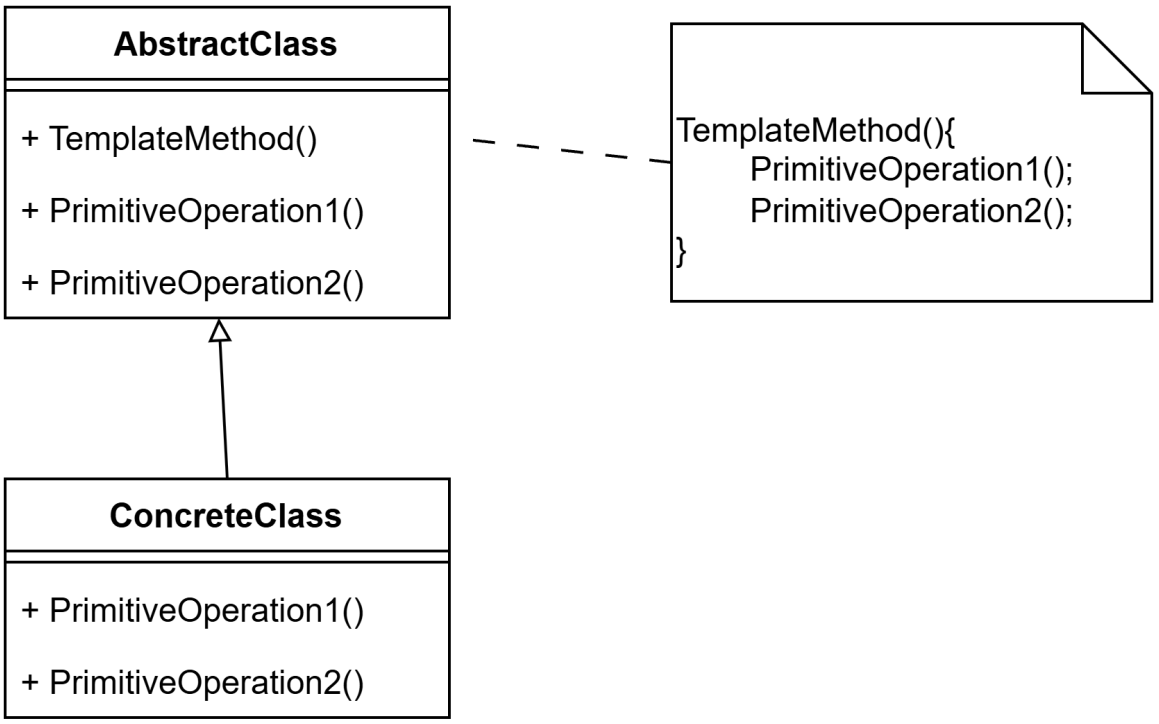
9. Які класи входять в шаблон «Міст», та яка між ними взаємодія?

Client - клієнтський клас, який використовує абстракцію.
Abstraction - високорівнева абстракція, яка використовується клієнтом. Делегує виконання конкретним реалізаціям.
RefinedAbstraction - певна реалізація абстракції.
Implementor - інтерфейс реалізації.
ConcreteImplementor1,2 - конкретні класи реалізації.

10.Яке призначення шаблону «Шаблоний метод»?

Шаблонний метод визначає структуру алгоритму та дозволяє класам-наслідникам перевизначати частини цього алгоритму. Наприклад шаблонний метод може визначати процес формування веб-сторінки. Конкретні класи-наслідники можуть перевизначати частини алгоритму створення, наприклад формування заголовків, вмісту сторінки, додаткових файлів тощо.

11.Нарисуйте структуру шаблону «Шаблоний метод».



12.Які класи входять в шаблон «Шаблоний метод», та яка між ними взаємодія?

AbstractClass - абстрактний клас шаблонного методу. Може мати реалізацію кроків алгоритму
ConcreteClass - наслідник абстрактного класу шаблонного методу, який визначає частини алгоритму.

13.Чим відрізняється шаблон «Шаблонний метод» від «Фабричного методу»?

Шаблонний метод визначає певний алгоритм, може визначтаи його кроки та дає класам-наслідникам можливість перевизначати кроки цього методу. Фабричний метод потрібен для створення об’єктів. Класи-наслідники фабричного методу створюють об’єкти свого типу.

14.Яку функціональність додає шаблон «Міст»?

Він дає змогу змінювати абстракцію та реалізацію незалежно. Тобто підвищується гнучкість та підтримуваність коду, завдяки тому що можна незалежно розширювати ці дві ієрархії, динамічно змінювати реалізацію. Також “Міст” зменшує кількість класів, якщо порівнювати з наслідуванням, оскільки без мосту потрібно було робити окремий клас для комбінацій абстракції та реалізації.