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 №5

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 «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» та навчитися застосовувати їх в реалізації програмної системи.

Теоретичні відомості:

Шаблон "Adapter" (Адаптер) використовується для адаптації інтерфейсу одного об'єкту до іншого. Наприклад, існує декілька бібліотек для роботи з принтерами, проте кожна має різний інтерфейс (хоча однакові можливості і призначення). Має сенс розробити уніфікований інтерфейс (сканування, асинхронне сканування, двостороннє сканування, потокове сканування і тому подібне), і реалізувати відповідні адаптери для приведення бібліотек до уніфікованого інтерфейсу. Це дозволить у програмі звертатися до загального інтерфейсу, а не приводити різні сценарії роботи залежно від способу реалізації бібліотеки. Адаптери також називаються "wrappers" (обгортками).

Переваги та недоліки:

+ Відокремлює інтерфейс або код перетворення даних від основної бізнес-логіки.

+ Можна добавляти нові адаптери не змінюючи код у класі Client.

- Умовним недоліком можна назвати збільшення кількості класів, але за рахунок використання паттерна Адаптер програмний код, як правило, стає легше читати.

Шаблон «Builder» (Будівельник) використовується для відділення процесу створення об'єкту від його представлення. Це доречно у випадках, коли об'єкт має складний процес створення (наприклад, Web-сторінка як елемент повної відповіді web- сервера) або коли об'єкт повинен мати

декілька різних форм створення (наприклад, при конвертації тексту з формату у формат).

Переваги та недоліки:

+ Дозволяє використовувати один і той самий код для створення різноманітних продуктів.

- Клієнт буде прив’язаний до конкретних класів будівельників, тому що в інтерфейсі будівельника може не бути методу отримання результату.

Шаблон "command" (команда) перетворить звичайний виклик методу в клас. Таким чином дії в системі стають повноправними об'єктами. Це зручно в наступних випадках:

• Коли потрібна розвинена система команд – відомо, що команди будуть добавлятися;

• Коли потрібна гнучка система команд – коли з'являється необхідність додавати командам можливість відміни, логування і інш.;

• Коли потрібна можливість складання ланцюжків команд або виклику команд в певний час.

Об'єкт команда сама по собі не виконує ніяких фактичних дій окрім перенаправлення запиту одержувачеві (тобто команди все ж виконуються одержувачем), однак ці об'єкти можуть зберігати дані для підтримки додаткових функцій відміни, логування й інші. Наприклад, команда вставки символу може запам'ятовувати символ, і при виклику відміни викликати відповідну функцію витирання символу. Можна також визначити параметр «застосовності» команди (наприклад, на картинці писати не можна) – і використати цей атрибут для засвічування піктограми в меню. Такий підхід до команд дозволяє побудувати дуже гнучку систему команд, що настроюється. У більшості додатків це буде зайвим (використовується спрощений варіант), проте життєво важливий в додатках з великою кількістю команд (редактори).

Переваги та недоліки:

+ Ініціатор виконання команди не знає деталей реалізації виконавця команди.

+ Підтримує операції скасування та повторення команд.

+ Послідовність команд можна логувати і при необхідності виконати цю послідовність ще раз.

+ Простота розширення за рахунок додавання нових команд без необхідності внесення змін в уже існуючий код (принцип відкритості-закритості).

Шаблон «Chain of responsibility» (Ланцюжок відповідальності) частково можна спостерігати в житті, коли підписання відповідного документу проходить від його складання у одного із співробітників компанії через менеджера і начальника до головного начальника, який ставить свій підпис. Це поведінковий патерн проектування, що дає змогу передавати запити послідовно ланцюжком обробників. Кожен наступний обробник вирішує, чи може він обробити запит сам і чи варто передавати запит далі ланцюжком.

Переваги та недоліки:

+ Зменшує залежність між клієнтом та обробниками: клієнт не знає хто обробить запит, а обробники не знають структуру ланцюжка.

+ Реалізує додаткову гнучкість в обробці запиту: легко вставити або вилучити з ланцюжка нові обробники.

- Запит може залишитися ніким не опрацьованим: запит не має вказаного обробника, тому може бути не опрацьованим.

Шаблон «Prototype» (Прототип) використовується для створення об'єктів за «шаблоном» (чи «кресленням», «ескізом») шляхом копіювання шаблонного об'єкту, який називається прототипом. Для цього визначається метод «клонувати» в об'єктах цього класу.

Цей шаблон зручно використати, коли заздалегідь відомо як виглядатиме кінцевий об'єкт (мінімізується кількість змін до об'єкту шляхом створення шаблону), а також для видалення необхідності створення об'єкту – створення відбувається за рахунок клонування, і самій програмі немає необхідності знати, як створювати об'єкт.

Також, це дозволяє маніпулювати об'єктами під час виконання програми шляхом настроювання відповідних прототипів; значно зменшується ієрархія наслідування (оскільки в іншому випадку це були б не прототипи, а вкладені класи, що наслідують).

Переваги та недоліки:

+ За рахунок клонування складних об’єктів замість їх створення, підвищується продуктивність.

+ Різні варіації об’єктів можна отримувати за рахунок клонування, а не розширення ієрархії класів.

+ Вища гнучкість, тому що клоновані об’єкти можна модифікувати незалежно, не впливаючи на об’єкт з якого була зроблена копія.

- Реалізація глибокого клонування досить проблематична, коли об’єкт що клонується містить складну внутрішню структуру та посилання на інші об’єкти.

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:

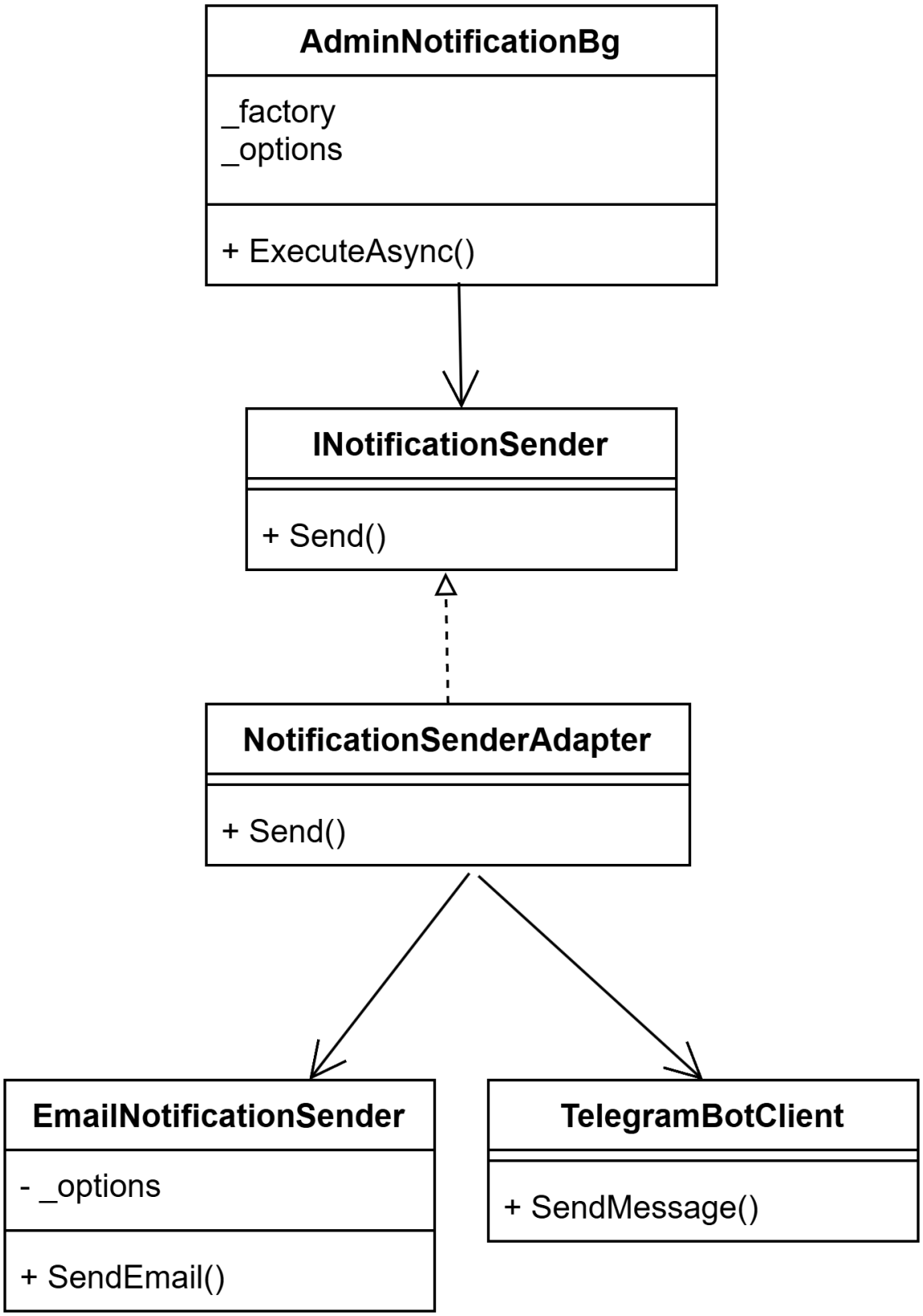


Рис.1 - Діаграма класів патерну “Адаптер”

На діаграмі зображено:

AdminNotificationBg - сервіс, що відправляє повідомлення адміністраторам по мірі їх надходження. Має метод ExecuteAsync() для виконання.

INotificationSender - інтерфейс для відправника повідомлень. Визначає єдиний метод Send() для відправки повідомлень.

NotificationSenderAdapter - адаптує інтерфейс сервісів для відправки повідомлень по пошті та через телеграм бота.

EmailNotificationSender - клас для відправки повідомлень за допомогою електронної пошти, застосовує smtp-клієнт зі своїми параметрами.

TelegramBotClient - клас для роботи з ботом у телеграмі, що має свій інтерфейс для відправки повідомлень.

Вихідний код класів:

```
using Microsoft.Extensions.Options;

namespace BookingClinic.Services.NotificationService.AdminNotificationService
{
    Ссылок: 4
    public class AdminNotificationBg : BackgroundService
    {
        private readonly INotificationSender _notificationSender;
        private readonly AdminNotificationsOptions _options;
        private readonly ILogger<AdminNotificationBg> _logger;
        private readonly IAdminAlertQueue _alertQueue;

        Ссылок: 0
        public AdminNotificationBg(
            INotificationSender notificationSender,
            IOptions<AdminNotificationsOptions> options,
            ILogger<AdminNotificationBg> logger,
            IAdminAlertQueue alertQueue)
        {
            _notificationSender = notificationSender;
            _options = options.Value;
            _logger = logger;
            _alertQueue = alertQueue;
        }

        Ссылок: 0
        protected override async Task ExecuteAsync(CancellationToken stoppingToken)
        {
            while (!stoppingToken.IsCancellationRequested)
            {
                await foreach (var alert in _alertQueue.ReadAllAsync(stoppingToken))
                {
                    try
                    {
                        var emails = _options.Emails;
                        var chats = _options.ChatIds;

                        if (emails != null)
                        {
                            foreach (var email in emails)
```

Рис.2 - Код AdminNotificationBg

```
                            {
                                await _notificationSender.Send(email, alert.Subject, alert.Message);
                            }
                        }

                        if (chats != null)
                        {
                            foreach (var chat in chats)
                            {
                                await _notificationSender.Send(chat.ToString(), alert.Subject, alert.Message);
                            }
                        }
                    }
                    catch (Exception ex)
                    {
                        _logger.LogError(ex, "Failed to send admin alert");
                    }
                }
            }
        }
    }
}
```

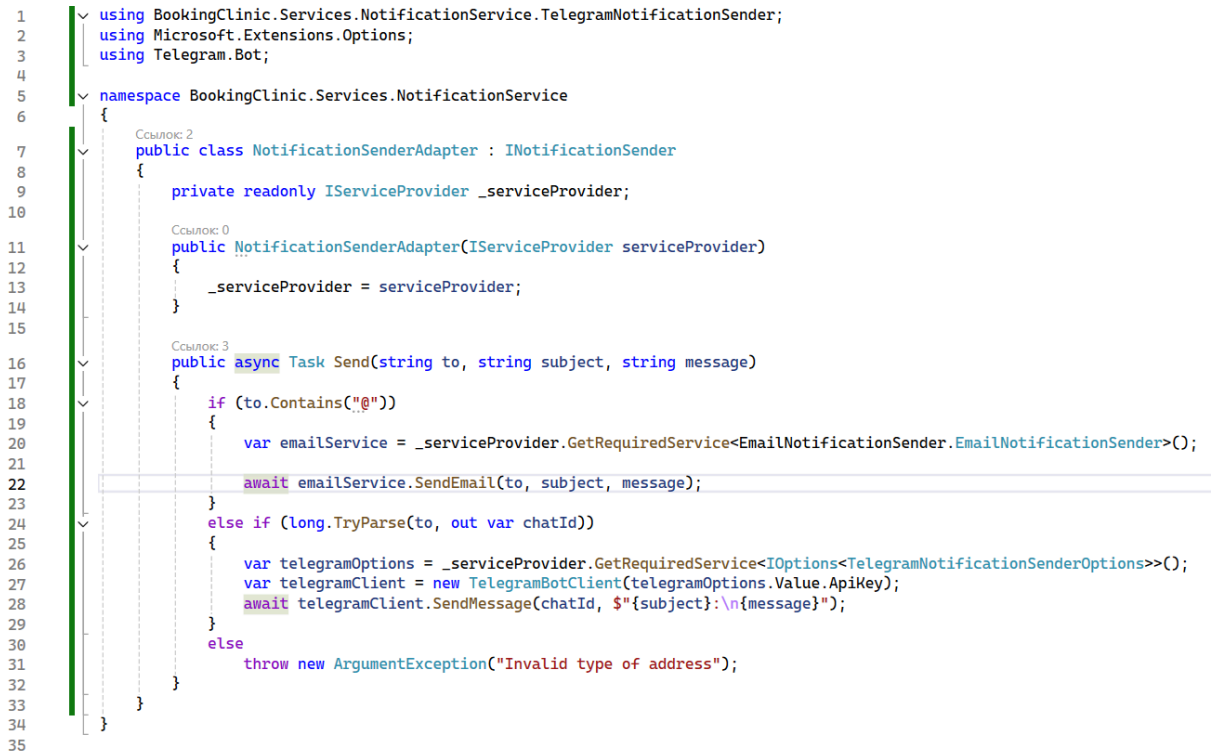
Рис.3 - Код AdminNotificationBg

```
namespace BookingClinic.Services.NotificationService
{
    Ссылок: 4
    public interface INotificationSender
    {
        Ссылок: 3
        Task Send(string to, string subject, string message);
    }
}
```

Рис.4 - Код INotificationSender

```
using MailKit.Net.Smtp;
using Microsoft.Extensions.Options;
using MimeKit;

namespace BookingClinic.Services.NotificationService.EmailNotificationSender
{
    Ссылок: 3
    public class EmailNotificationSender
    {
        private EmailNotificationSenderOptions _options;

        Ссылок: 0
        public EmailNotificationSender(IOptions<EmailNotificationSenderOptions> options)
        {
            _options = options.Value;
        }

        Ссылок: 1
        public async Task SendEmail(string email, string subject, string body)
        {
            var message = new MimeMessage();
            message.From.Add(new MailboxAddress("BookingClinic", _options.Username));
            message.To.Add(new MailboxAddress("", email));
            message.Subject = subject;
            message.Body = new TextPart("plain")
            {
                Text = body
            };

            using SmtpClient client = new SmtpClient();
            await client.ConnectAsync(_options.Server, _options.Port, MailKit.Security.SecureSocketOptions.StartTls);
            await client.AuthenticateAsync(_options.Username, _options.Password);
            await client.SendAsync(message);
            await client.DisconnectAsync(true);
        }
    }
}
```

Рис.5 - Код EmailNotificationSender



```
using BookingClinic.Services.NotificationService.TelegramNotificationSender;
using Microsoft.Extensions.Options;
using Telegram.Bot;

namespace BookingClinic.Services.NotificationService
{
    Ссылок: 2
    public class NotificationSenderAdapter : INotificationSender
    {
        private readonly IServiceProvider _serviceProvider;

        Ссылок: 0
        public NotificationSenderAdapter(IServiceProvider serviceProvider)
        {
            _serviceProvider = serviceProvider;
        }

        Ссылок: 3
        public async Task Send(string to, string subject, string message)
        {
            if (to.Contains("@"))
            {
                var emailService = _serviceProvider.GetRequiredService<EmailNotificationSender.EmailNotificationSender>();

                await emailService.SendEmail(to, subject, message);
            }
            else if (long.TryParse(to, out var chatId))
            {
                var telegramOptions = _serviceProvider.GetRequiredService<IOptions<TelegramNotificationSenderOptions>>();
                var telegramClient = new TelegramBotClient(telegramOptions.Value.ApiKey);
                await telegramClient.SendMessage(chatId, $"{subject}:\n{message}");
            }
            else
                throw new ArgumentException("Invalid type of address");
        }
    }
}

```

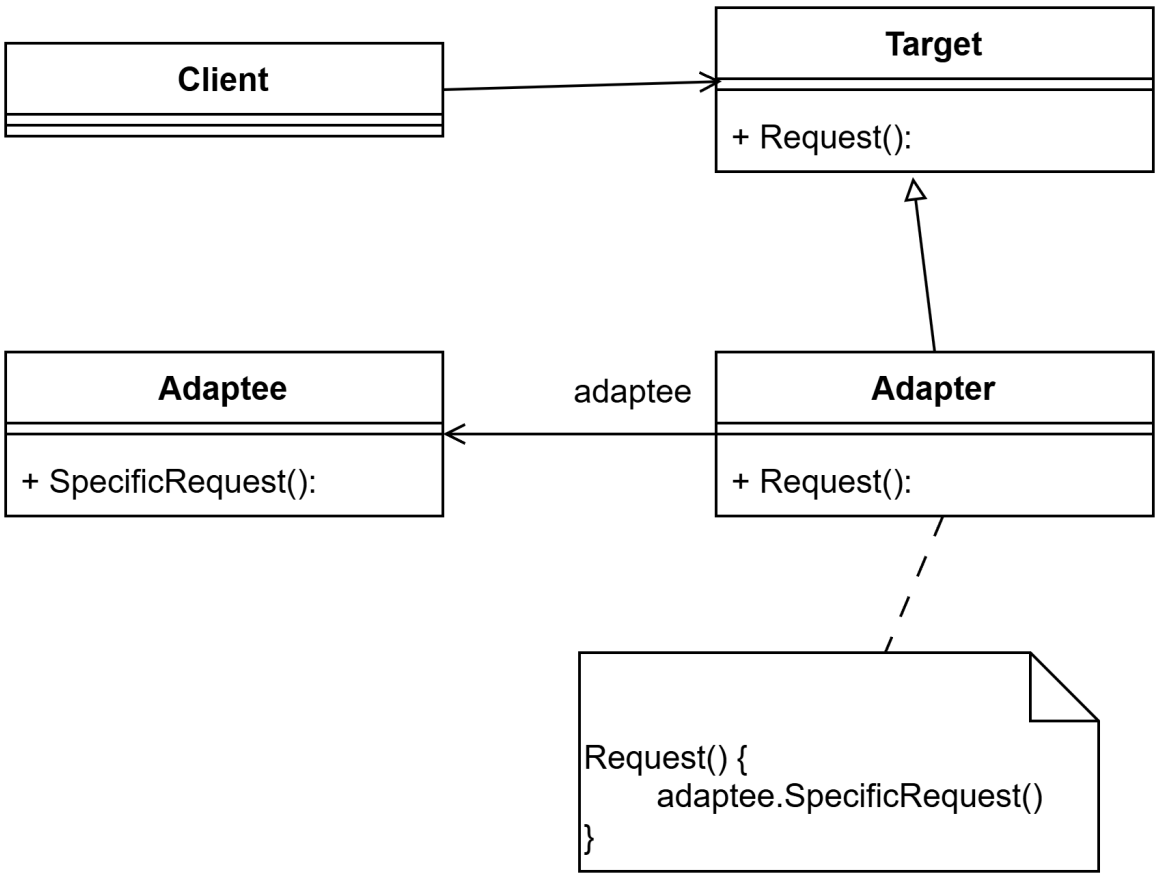
Рис.6 - Код NotificationSenderAdapter

Контрольні питання:

1. Яке призначення шаблону «Адаптер»?

Адаптер - це структурний шаблон проектування, який дозволяє разом працювати класам з несумісними інтерфейсами. Наприклад, є клас для відправки повідомлень, що отримує інтерфейс відправника повідомлень для відправки, а також є різні сервіси для відправки повідомлень і кожний з них має свій інтерфейс. У такому разі застосовуються адаптери, щоб адаптувати інтерфейси сервісів під той що потрібен для відправки повідомлення.

2. Нарисуйте структуру шаблону «Адаптер».



На цій діаграмі:
Client - Клієнтський код, який використовує клас Target з певним інтерфейсом.
Adaptee - клас з несумісним інтерфейсом, який потрібно адаптувати під Target.
Target - клас, інтерфейс якого використовує клієнт.
Adapter - клас, що слугує адаптером класу Adaptee для Target.

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

Адаптер обов’язково має певний клієнтський клас, який використовує якийсь певний інтерфейс. Має бути клас, що надає якийсь сервіс, але має несумісний для клієнтського коду інтерфейс. І також є клас-адаптер, який обгортає клас з сервісом і реалізує інтерфейс, який потрібен клієнтському коду, тобто клас-адаптер адаптує інтерфейс сервісу для клієнта.

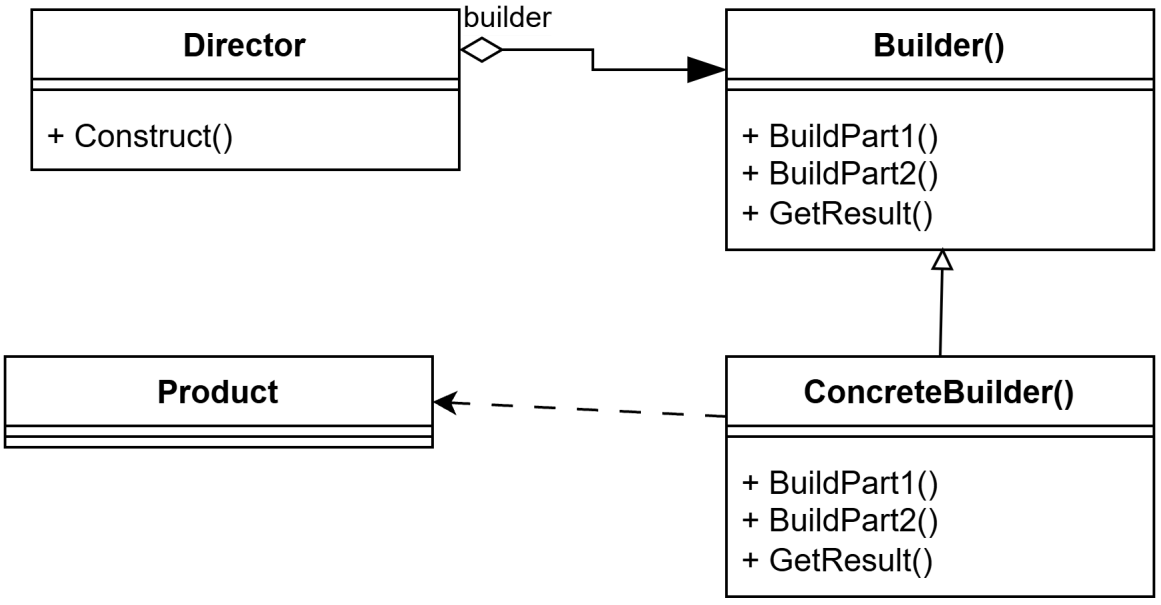
4. Яка різниця між реалізацією «Адаптера» на рівні об’єктів та на рівні класів?

Адаптер на рівні об'єктів обгортає об’єкт класу, який він адаптує. Тобто адаптер об’єкту тримає посилання на адаптований об’єкт і делегує виклики методів йому.
Адаптер на рівні класів використовує множинне наслідування. У такому разі клас, який треба адаптувати, додатково наслідує інтерфейс, який треба реалізувати.

5. Яке призначення шаблону «Будівельник»?

Будівельник це породжуючий паттерн, який дозволяє поетапно будувати складні об’єкти. В залежності від типу об’єкта, який потрібно створити, потрібно створювати окремий клас будівельника. Наприклад потрібно створити будинок, який складається з підлоги, стін, даху тощо. Для цього робиться будівельник з методами для окремої побудови цих складових, а потім реалізуються конкретні будівельники для різних типів будинків: кам’яного, дерев’яного тощо.
Також до будівника застосовується клас-директор, який задає порядок виклику методів будівельника.

6. Нарисуйте структуру шаблону «Будівельник».



На цій діаграмі:
Product - клас, який створюється будівником.
Builder - базовий інтерфейс будівника.
ConcreteBuilder - конкретний будівник, який створює продукт певного виду.
Director - клас, що викликає методи інтерфейса будівника для побудови продукту.

7. Які класи входять в шаблон «Будівельник», та яка між ними взаємодія?

В шаблон входить інтерфейс будівельника, який використовується класом-директором для побудови продуктів. Він визначає методи, для побудови складових продукту.
Клас-директор використовує будівельника для побудови продуктів. Вні визначає алгоритм побудови.
Конкретні класи-будівельники реалізують інтерфейс та визначають процес створення своїх певних продуктів.
Клас-продукт, який є результатом побудови будівельником.

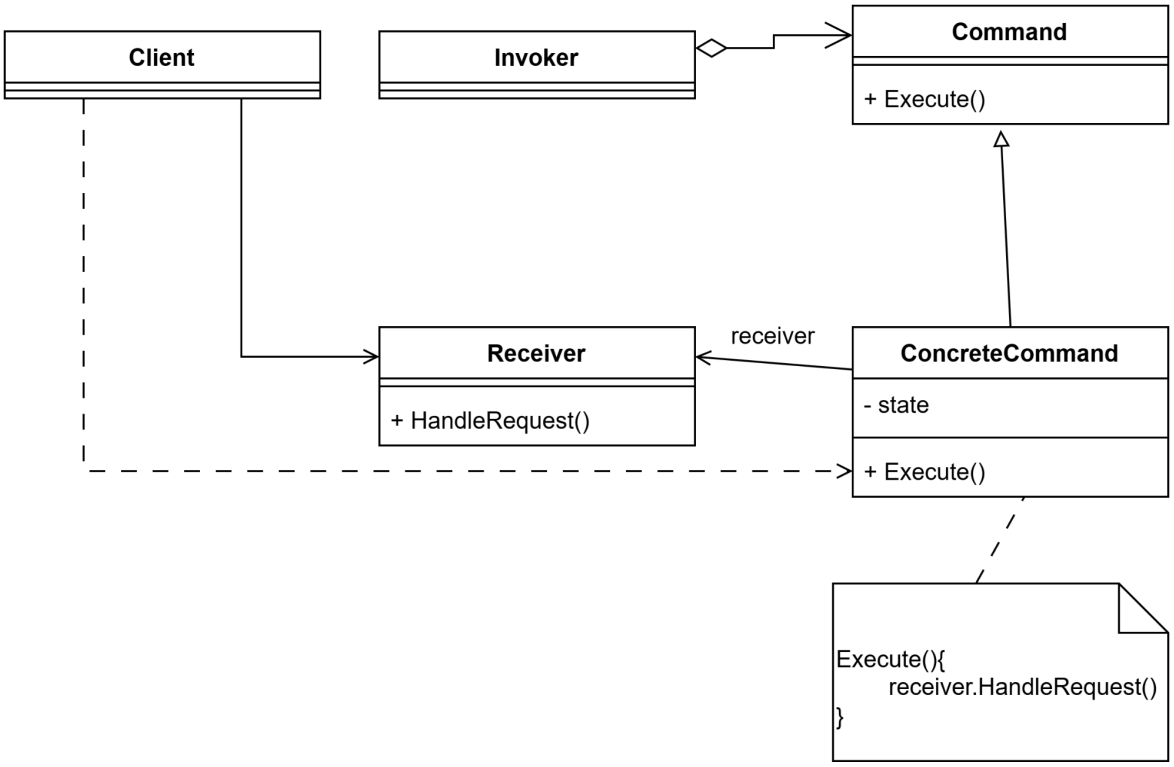
8. У яких випадках варто застосовувати шаблон «Будівельник»"?

Паттерн варто застосовувати, коли об’єкт має складний процес створення або має мати багато різних форм створення. Наприклад, для створення веб-запиту потрібно додати стандартні заголовки, код статусу, заголовки відповіді, тіло запиту тощо. У такому разі доречно абстрагувати кожний етап в окремий метод будівельника.

9. Яке призначення шаблону «Команда»?

Паттерн перетворює звичайні виклики методів у клас-команду. Тобто кожна дія в системі представляється своїм класом-командою. Це дозволяє побудувати більш розвинену систему команд; додає можливітсь логування, скасування та інших операцій над командами; дозволяє будувати ланцюжки команд або викликати їх в певний час.

10. Нарисуйте структуру шаблону «Команда».



На цій діаграмі:
Client - клієнтський код, який призначає певні команди своїм викликачам.
Receiver - клас, що обробляє команди.
Invoker - при якійсь дії запускає відповідну команду.
Command - описує базовий клас команди з одним методом запуску.
ConcreteCommand - конкретна команда, що містить якісь свої специфічні дані.

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

Клієнтський клас Client, який задає який Invoker викликає яку команду.
Базовий клас команди Command, що має метод Execute() для виконання команди.
Конкретні класи-реалізації команд ConcreteCommand, що можуть мати свої дані відповідно до команди і виконують роботу за допомогою Receiver.
Клас-викликач Invoker, який викликає команду коли потрібно.
Клас Receiver, який виконує реальну роботу команд.

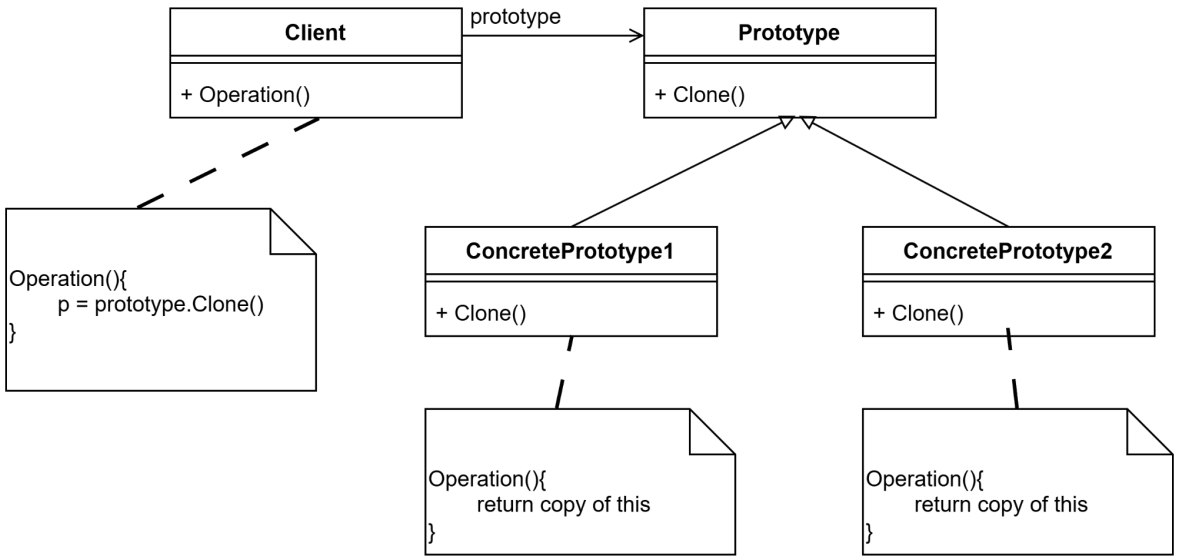
12. Розкажіть як працює шаблон «Команда».

Клієнт створює викликачів та задає їм команди. Викликачі викликають певні команди, які в свою чергу передають управління отримувачу. Наприклад може бути товстий клієнт з інтерфейсом, у якому є деякі дії, які додатково можуть повторюватись. Ми створюємо клас, який описує певну команду та при натисканні певних кнопок просто викликаємо її. Команда буде передавати виконання отримувачу, де буде відбуватись дія.

13. Яке призначення шаблону «Прототип»?

Прототип призначений для клонування об’єктів за допомогою спеціального методу, який реалізується в цих же об’єктах. Таким чином об'єкт отримує можливість створити копію самого себе включаючи приватні поля.

14. Нарисуйте структуру шаблону «Прототип».



На цій діаграмі:
Client - клієнтський код, який використовує клас-прототип.
Prototype - інтерфейс класу-прототипу, що містить метод Clone() для клонування.
ConcretePrototype1,2 - конкретні класи-прототипи, що вміють клонувати себе.

15. Які класи входять в шаблон «Прототип», та яка між ними взаємодія?

Паттерн містить клієнтський клас, якому потрібно клонувати об’єкт. Є інтерфейс, що визначає метод для клонування об’єкту, що його реалізує. Також наявні конкретні класи, що реалізують інтерфейс клонування та реалізують метод для клонування, в якому створюють копію себе. Клієнтський клас викликає метод клонування у класів для отримання їх копій.

16. Які можна привести приклади використання шаблону «Ланцюжок відповідальності»?

Шаблон застосовується коли ми передаємо кожному об’єкту по черзі керування і він може щось зробити.
Наприклад якщо у нас є контексте меню де можуть змінюватись кількість елементів ми можемо не робити загальний метод для виклику меню, а дати кожному компоненту метод для оновлення меню так щоб кожний додав у меню свою частину.
Також прикладом є реалізації обробки запитів в ASP.NET. При виникненні запиту він обробляється по черзі компонентами middleware, кожен з яких виконує якісь свої дії над запитом: авторизує користувача, обробляє заголовки, серіалізує параметри та тіло запиту тощо.

Висновки:

У цій лабораторній роботі я ознайомився з паттернами: “Адаптер”, “Будівельник”, “Команда”, “Ланцюг обов’язків”, “Прототип”. Обрав та реалізував патерн “Адаптер” для адаптації інтерфейсу сервісу відправки повідомлень у своєму проекті, щоб сповіщати адміністраторів про важливі помилки в роботі системи. Вивчене покращило моє розуміння патернів проектування та знадобиться в майбутньому.