



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 розроблення програмного забезпечення»*
Тема: «Патерни проектування»

Виконала:
Студент групи ІА-31
Самелюк А.С.

Перевірив:
Мягкий М.Ю.

Тема: Патерни проектування.

Тема проєкту: Онлайн-щоденник (strategy, adapter, observer, facade, composite, client-server). Веб-застосунок дозволяє користувачу створювати, редагувати та організовувати особисті записи з можливістю пошуку, сортування та додавання тегів, а також отримання email-нагадувань.

Мета: Вивчити структуру шаблонів «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype» та навчитися застосовувати їх в реалізації програмної системи.

Посилання на репозиторій з проєктом та звітами:

https://github.com/Sameliuk/OnlineDiary_trpz/tree/main

https://github.com/Sameliuk/OnlineDiary_trpz/tree/reports

Хід роботи

1. Ознайомитись з короткими теоретичними відомостями.

Патерн «Adapter» (Адаптер)

Патерн призначений для узгодження інтерфейсів різних класів. Він дозволяє об'єктам з несумісними інтерфейсами працювати разом через спільний інтерфейс.

Приклад: аудіоплеєр, який підтримує різні формати, може використовувати адаптери для кожного формату, щоб працювати через єдиний інтерфейс IPlayer.

Переваги: спрощує інтеграцію, розділяє логіку та інтерфейси, легко додавати нові адаптери.

Недолік: збільшення кількості класів.

Патерн «Builder» (Будівельник)

Використовується для відокремлення процесу створення складного об'єкта від його представлення.

Приклад: створення відповіді web-сервера поетапно — заголовки, статус, тіло

тощо.

Переваги: гнучкість, незалежність від змін структури.

Недолік: клієнт прив'язаний до конкретного будівельника.

Патерн «Command» (Команда)

Перетворює виклик дії у самостійний об'єкт. Це дозволяє зберігати, відмінити, повторювати або комбінувати команди.

Приклад: у графічному додатку дії меню, кнопок і контекстного меню реалізуються через спільний інтерфейс команд.

Переваги: гнучкість, підтримка відміни, логування, розширюваність.

Патерн «Chain of Responsibility» (Ланцюжок відповідальності)

Дозволяє передавати запит по ланцюжку обробників, доки один з них не обробить його.

Приклад: формування контекстного меню у складному інтерфейсі — кожен елемент додає свої пункти й передає далі.

Переваги: зменшує залежність між об'єктами, легко змінювати ланцюг.

Недолік: запит може залишитися необробленим.

Патерн «Prototype» (Прототип)

Створює нові об'єкти шляхом клонування існуючого прототипу.

Приклад: у редакторі рівнів кнопки створюють об'єкти не через new, а копіюванням шаблону GameObject.

Переваги: швидке створення складних об'єктів, гнучкість, менше наслідування.

Недолік: складність реалізації глибокого копіювання.

2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

В даній частині було реалізовану логіку з нотатками та відредаговано логіку з нагадуваннями. Було реалізовано відповідні сервіси, контролери та інтерфейси для виконання певного функціоналу.

```

Details.cshtml  IEmailSender.cs  EmailService.cs  EmailServiceAdapter.cs  Remir
C# D:\KPI\OnlineDiaryApp\OnlineDiaryApp\OnlineDiaryApp.csproj
@model OnlineDiaryApp.Models.Note

<link rel="stylesheet" href="~/css/notes.css" />

<div class="note-details">
    <h2>@Model.Title</h2>
    <p class="note-date">Створено: @Model.CreatedAt.ToString("f")</p>

    <div class="note-content">
        <h4>Зміст</h4>
        <p>@Model.Content</p>
    </div>

    <div class="note-tags">
        <h4>Теги</h4>
        @if (Model.Tags.Any())
        {
            @foreach (var tag in Model.Tags)
            {
                <span class="note-tag">@tag.Name</span>
            }
        }
        else
        {
            <span class="note-tag-empty">Без тегів</span>
        }
    </div>

    @if (ViewBag.Reminder != null)
    {
        <div class="note-reminder">
            <h4>Нагадування</h4>
            <p>@(((DateTime)ViewBag.Reminder.RemindAt).ToString("f"))</p>
        </div>
    }

    <div class="note-actions">
        <a asp-action="Edit" asp-route-id="@Model.Id" class="btn-edit">Редагувати</a>
        <a asp-action="Delete" asp-route-id="@Model.Id" class="btn-delete">Видалити</a>
        <a asp-action="Index" class="btn-back">← Назад</a>
    </div>
</div>

```

Код 1 – Details (Views.Note)

```

NoteController.cs Details.cshhtml IEmailSender.cs EmailService.cs EmailServiceAdapter.cs
OnlineDiaryApp OnlineDiaryApp.Controllers.NoteController

1 using Microsoft.AspNetCore.Mvc;
2 using OnlineDiaryApp.Models;
3 using OnlineDiaryApp.Services;
4
5 namespace OnlineDiaryApp.Controllers
6 {
7     1 reference
8     public class NoteController : Controller
9     {
10         private readonly NoteService _noteService;
11         private readonly ReminderService _reminderService;
12         private readonly TagService _tagService;
13
14         0 references
15         public NoteController(NoteService noteService, ReminderService reminderService, TagService tagService)
16         {
17             _noteService = noteService;
18             _reminderService = reminderService;
19             _tagService = tagService;
20         }
21
22         0 references
23         public async Task<IActionResult> Index(string? sortBy, string? tag)
24         {
25             var userIdString = HttpContext.Session.GetString("UserId");
26             if (!int.TryParse(userIdString, out int userId))
27                 return RedirectToAction("Login", "User");
28
29             ISortStrategy? strategy = null;
30             if (!string.IsNullOrEmpty(sortBy))
31             {
32                 strategy = sortBy.ToLower() switch
33                 {
34                     "date" => new SortByDateStrategy(),
35                     "tag" when !string.IsNullOrEmpty(tag) => new SortByTagStrategy(tag),
36                     "title" => new SortByTitleStrategy(),
37                     _ => null
38                 };
39             }
40
41             var notes = await _noteService.GetAllNotesByUserAsync(userId, strategy);
42
43             ViewBag.SortBy = sortBy;
44             ViewBag.SelectedTag = tag;
45             ViewBag.Tags = await _tagService.GetAllTagsAsync(userId);
46
47             return View(notes);
48         }
49
50         0 references
51         public async Task<IActionResult> Create()
52         {
53             var userIdString = HttpContext.Session.GetString("UserId");
54             if (!int.TryParse(userIdString, out int userId))
55                 return RedirectToAction("Login", "User");
56
57             var tags = await _tagService.GetAllTagsAsync(userId);
58             ViewBag.Tags = tags ?? new List<Tag>();
59
60             return View();
61         }
62
63         [HttpPost]
64         0 references
65         public async Task<IActionResult> Create(string title, string content, List<int>? tagIds, DateTime? reminderDate)
66         {
67             var userIdString = HttpContext.Session.GetString("UserId");
68             if (!int.TryParse(userIdString, out int userId))
69                 return RedirectToAction("Index");
70
71             var note = await _noteService.CreateNoteAsync(title, content, userId, tagIds ?? new List<int>());
72
73             if (reminderDate.HasValue)
74             {
75                 await _reminderService.CreateReminderAsync(note.Id, reminderDate.Value, userId);
76             }
77
78             return RedirectToAction("Index");
79         }
80
81         0 references
82         public async Task<IActionResult> Edit(int id)
83         {
84             var note = await _noteService.GetNoteByIdAsync(id);
85             if (note == null)
86                 return NotFound();
87
88             var userIdString = HttpContext.Session.GetString("UserId");
89             if (!int.TryParse(userIdString, out int userId))
90                 return RedirectToAction("Login", "User");
91
92             ViewBag.Tags = await _tagService.GetAllTagsAsync(userId) ?? new List<Tag>();
93
94             var reminder = await _reminderService.GetReminderByNoteIdAsync(id);
95             ViewBag.Reminder = reminder;
96
97             return View(note);
98         }
99     }
100 }

```

Код 2.1 – NoteController (Controllers)

```

95 [HttpPost]
96 0 references
97 public async Task<IActionResult> Edit(int id, string title, string content, List<int>? tagIds, DateTime? reminderDate)
98 {
99     var note = await _noteService.GetNoteByIdAsync(id);
100     if (note == null)
101         return NotFound();
102
103     note.Title = title;
104     note.Content = content;
105
106     await _noteService.UpdateNoteAsync(note, tagIds ?? new List<int>());
107
108     var existingReminder = await _reminderService.GetReminderByNoteIdAsync(note.Id);
109     if (reminderDate.HasValue)
110     {
111         if (existingReminder != null)
112         {
113             await _reminderService.UpdateReminderAsync(existingReminder, reminderDate.Value);
114         }
115         else
116         {
117             await _reminderService.CreateReminderAsync(note.Id, reminderDate.Value, note.UserId);
118         }
119     }
120
121     return RedirectToAction("Index");
122 }
123
124 0 references
125 public async Task<IActionResult> Delete(int id)
126 {
127     await _noteService.DeleteNoteAsync(id);
128     return RedirectToAction("Index");
129 }
130
131 0 references
132 public async Task<IActionResult> Search(string keyword)
133 {
134     var userIdString = HttpContext.Session.GetString("UserId");
135     if (!int.TryParse(userIdString, out int userId))
136         return View(new List<Note>());
137
138     var notes = await _noteService.SearchByTitleAsync(keyword);
139     notes = notes.Where(n => n.UserId == userId);
140     return View("Index", notes);
141 }
142
143 0 references
144 public async Task<IActionResult> Details(int id)
145 {
146     var note = await _noteService.GetNoteByIdAsync(id);
147     if (note == null)
148         return NotFound();
149
150     var reminder = await _reminderService.GetReminderByNoteIdAsync(id);
151     ViewBag.Reminder = reminder;
152
153     return View(note);
154 }

```

Код 2.2 – NoteController (Controllers)

```
1 using OnlineDiaryApp.Services;
2
3 public class ReminderBackgroundService : BackgroundService
4 {
5     private readonly IServiceProvider _serviceProvider;
6
7     public ReminderBackgroundService(IServiceProvider serviceProvider)
8     {
9         _serviceProvider = serviceProvider;
10    }
11
12    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
13    {
14        while (!stoppingToken.IsCancellationRequested)
15        {
16            try
17            {
18                using var scope = _serviceProvider.CreateScope();
19                var reminderService = scope.ServiceProvider.GetRequiredService<ReminderService>();
20
21                var now = DateTime.UtcNow;
22
23                var reminders = (await reminderService.GetAllRemindersAsync())
24                    .Where(r => r.Status == "active" && r.RemindAt <= now)
25                    .ToList();
26
27                foreach (var reminder in reminders)
28                {
29                    await reminderService.SendReminderEmailAsync(reminder);
30                    await reminderService.UpdateReminderAsync(reminder, newStatus: "sent");
31                }
32            }
33            catch (Exception ex)
34            {
35                Console.WriteLine($"Reminder service error: {ex.Message}");
36            }
37
38            await Task.Delay(TimeSpan.FromMinutes(1), stoppingToken);
39        }
40    }
41 }
42
```

Код 3 – ReminderBackgroundService (Services)

3. Реалізувати один з розглянутих шаблонів за обраною темою.

Для даного проєкту було обрано патерн Adapter (Адаптер). Шаблон реалізований таким чином:

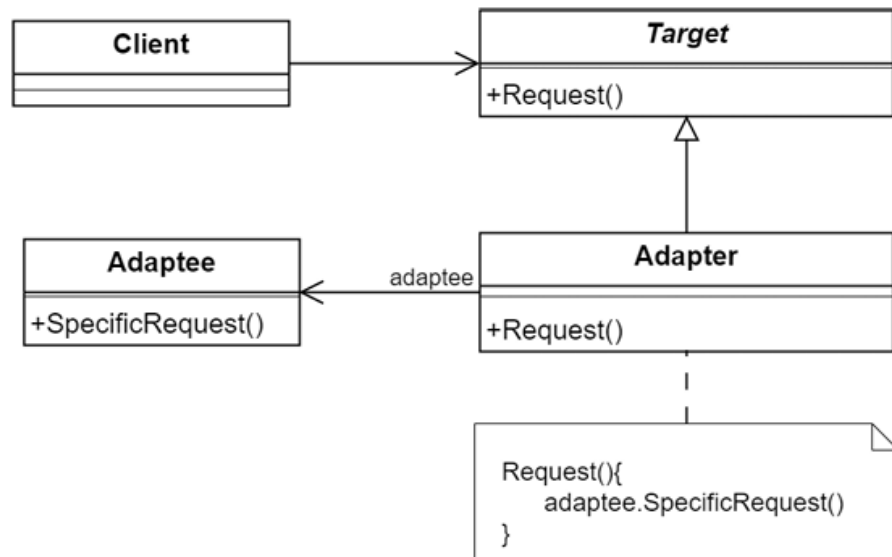


Рис.1 – Структура патерну Адаптер

Target: `ISender` – це інтерфейс, який описує метод `SendEmailAsync`. Він задає єдиний контракт для всіх способів відправлення електронних листів (SMTP, API, тестовий варіант тощо).


Client: `ReminderService` – це клієнт, який використовує `ISender`, не знаючи, як саме реалізовано відправку листів. Він просто викликає `SendEmailAsync`, а вже адаптер вирішує, який саме механізм відправлення використати.

Adaptee: `EmailService` – це клас-адаптований об’єкт, який уже вміє відправляти листи через SMTP. Однак його інтерфейс не відповідає тому, що очікує система (`ISender`).

Adapter: `EmailServiceAdapter` – це адаптер, який “обгортає” `EmailService` і перетворює його інтерфейс у формат `ISender`. Завдяки йому `EmailService` можна легко використовувати в коді, що працює з інтерфейсом `ISender`.

В результаті: можна легко змінювати або підключати різні способи відправлення електронних листів (через SMTP, API, інший сервіс) без зміни коду клієнта — `ReminderService`. Сервіс працює лише з інтерфейсом `ISender`, не знаючи, як саме реалізовано відправку. Це і є суть патерну Адаптер – узгодження несумісних інтерфейсів за допомогою проміжного класу, що перетворює виклики клієнта у форму, яку розуміє адаптований об’єкт (`EmailService`).

4. Реалізувати не менше 3-х класів відповідно до обраної теми.



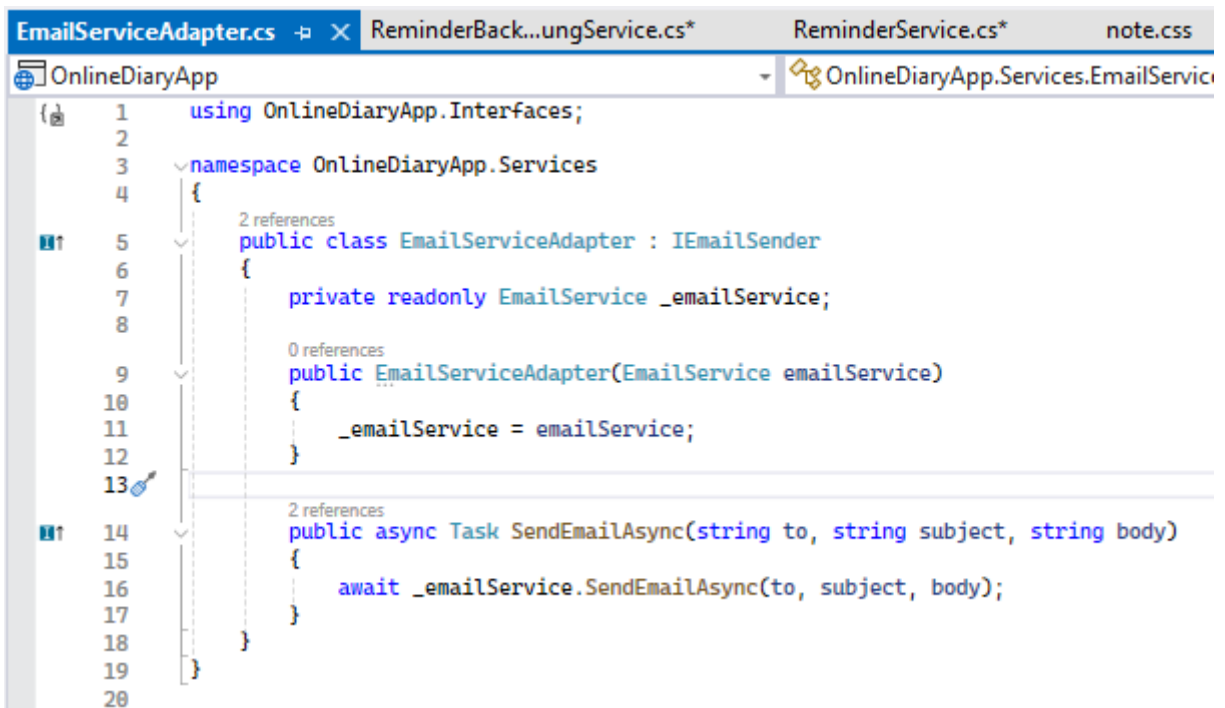
```

ReminderService.cs*  Index.cshtml  tag.css  Create.cshtml  note.c
OnlineDiaryApp  OnlineDiaryApp.Service

1  using OnlineDiaryApp.Models;
2  using OnlineDiaryApp.Interfaces;
3  using OnlineDiaryApp.Repositories.Interfaces;
4
5  namespace OnlineDiaryApp.Services
6  {
7      7 references
8      public class ReminderService
9      {
10         private readonly IRepository<Reminder> _reminderRepository;
11         private readonly IUserRepository _userRepository;
12         private readonly IEmailSender _emailSender;
13
14         0 references
15         public ReminderService(
16             IRepository<Reminder> reminderRepository,
17             IUserRepository userRepository,
18             IEmailSender emailSender)
19         {
20             _reminderRepository = reminderRepository;
21             _userRepository = userRepository;
22             _emailSender = emailSender;
23         }
24
25         3 references
26         public async Task CreateReminderAsync(int noteId, DateTime remindAt, int userId)
27         {
28             var utcRemindAt = DateTime.SpecifyKind(remindAt, DateTimeKind.Utc);
29
30             var reminder = new Reminder
31             {
32                 NoteId = noteId,
33                 UserId = userId,
34                 RemindAt = utcRemindAt,
35                 Status = "active"
36             };
37
38             await _reminderRepository.AddAsync(reminder);
39             await _reminderRepository.SaveChangesAsync();
40
41             await SendReminderEmailAsync(reminder);
42         }
43
44         2 references
45         public async Task SendReminderEmailAsync(Reminder reminder)
46         {
47             var user = reminder.User ?? await _userRepository.GetByIdAsync(reminder.UserId);
48             var note = reminder.Note;
49
50             if (user != null && note != null && !string.IsNullOrEmpty(user.Email))
51             {
52                 await _emailSender.SendEmailAsync(
53                     user.Email,
54                     $"Нагадування: {note.Title}",
55                     $"Нагадування по нотатці:\n\n{note.Content}");
56             }
57         }
58
59         1 reference
60         public async Task<IEnumerable<Reminder>> GetAllRemindersAsync()
61         {
62             return await _reminderRepository.GetAllAsync();
63         }
64
65         2 references
66         public async Task UpdateReminderAsync(Reminder reminder, DateTime? newRemindAt = null, string? newStatus = null)
67         {
68             if (newRemindAt.HasValue)
69                 reminder.RemindAt = DateTime.SpecifyKind(newRemindAt.Value, DateTimeKind.Utc);
70
71             if (!string.IsNullOrEmpty(newStatus))
72                 reminder.Status = newStatus;
73
74             await _reminderRepository.UpdateAsync(reminder);
75             await _reminderRepository.SaveChangesAsync();
76         }
77
78         0 references
79         public async Task DeleteReminderAsync(int reminderId)
80         {
81             var reminder = await _reminderRepository.GetByIdAsync(reminderId);
82             if (reminder != null)
83             {
84                 await _reminderRepository.DeleteAsync(reminder.Id);
85                 await _reminderRepository.SaveChangesAsync();
86             }
87         }
88
89         3 references
90         public async Task<Reminder?> GetReminderByNoteIdAsync(int noteId)
91         {
92             return await _reminderRepository.GetByNoteIdAsync(noteId);
93         }
94     }

```

Код 4 – ReminderService (Services)

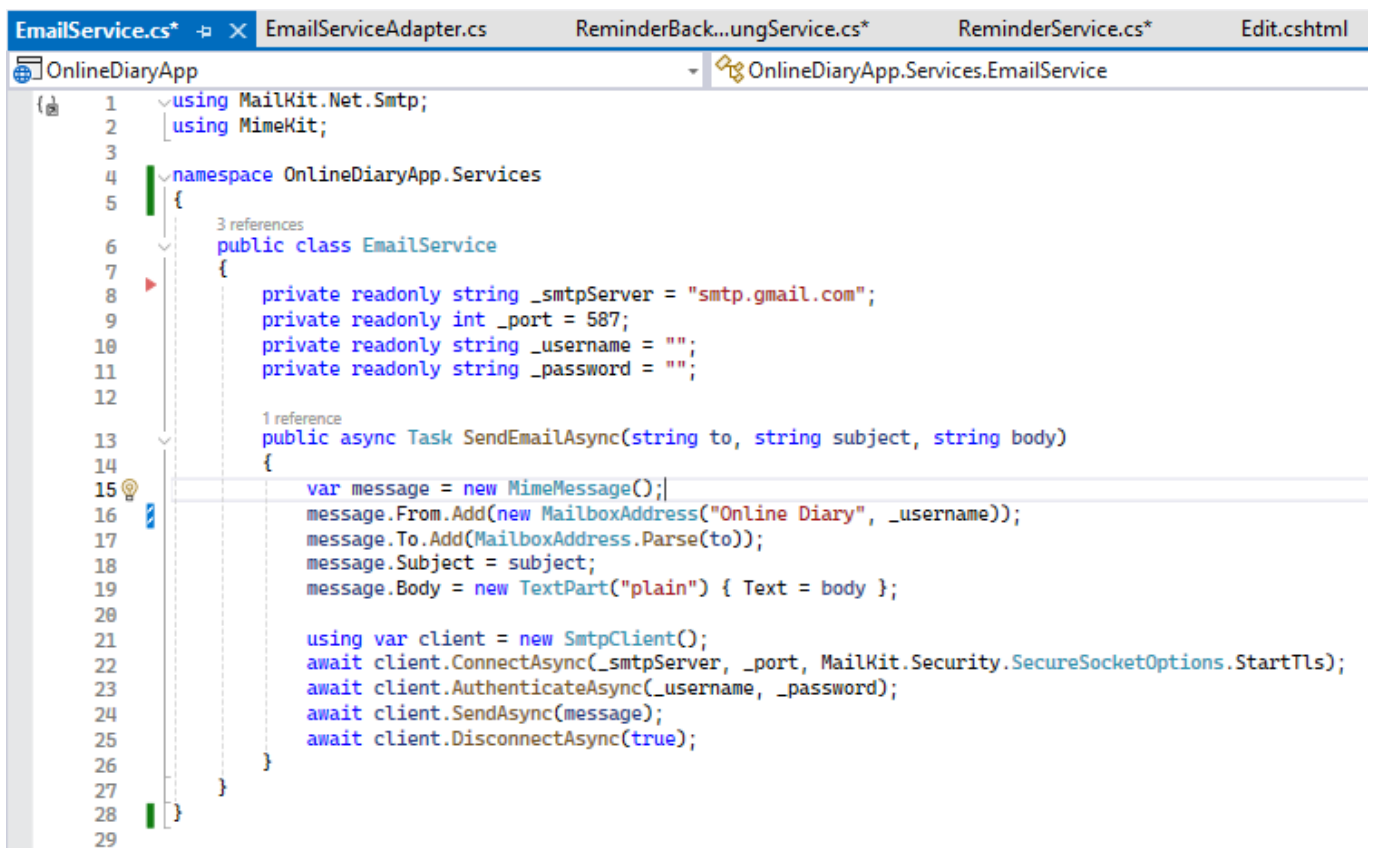


```

1  using OnlineDiaryApp.Interfaces;
2
3  namespace OnlineDiaryApp.Services
4  {
5      2 references
6      public class EmailServiceAdapter : IEmailSender
7      {
8          private readonly EmailService _emailService;
9
10         0 references
11         public EmailServiceAdapter(EmailService emailService)
12         {
13             _emailService = emailService;
14
15         2 references
16         public async Task SendEmailAsync(string to, string subject, string body)
17         {
18             await _emailService.SendEmailAsync(to, subject, body);
19         }
20     }

```

Код 5 – EmailServiceAdapter (Services)

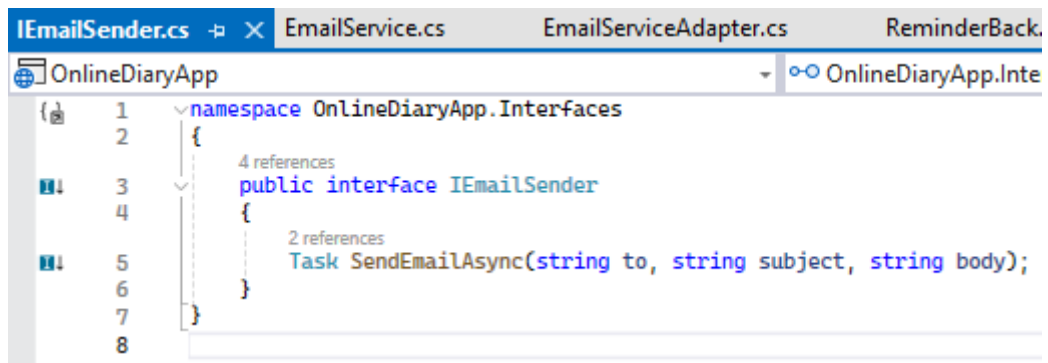


```

1  using MailKit.Net.Smtp;
2  using MimeKit;
3
4  namespace OnlineDiaryApp.Services
5  {
6      3 references
7      public class EmailService
8      {
9          private readonly string _smtpServer = "smtp.gmail.com";
10         private readonly int _port = 587;
11         private readonly string _username = "";
12         private readonly string _password = "";
13
14         1 reference
15         public async Task SendEmailAsync(string to, string subject, string body)
16         {
17             var message = new MimeMessage();
18             message.From.Add(new MailboxAddress("Online Diary", _username));
19             message.To.Add(MailboxAddress.Parse(to));
20             message.Subject = subject;
21             message.Body = new TextPart("plain") { Text = body };
22
23             using var client = new SmtpClient();
24             await client.ConnectAsync(_smtpServer, _port, MailKit.Security.SecureSocketOptions.StartTls);
25             await client.AuthenticateAsync(_username, _password);
26             await client.SendAsync(message);
27             await client.DisconnectAsync(true);
28         }
29     }

```

Код 6 – EmailService (Services)



Код 7 – IEmailSender (Interfaces)

5. 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.

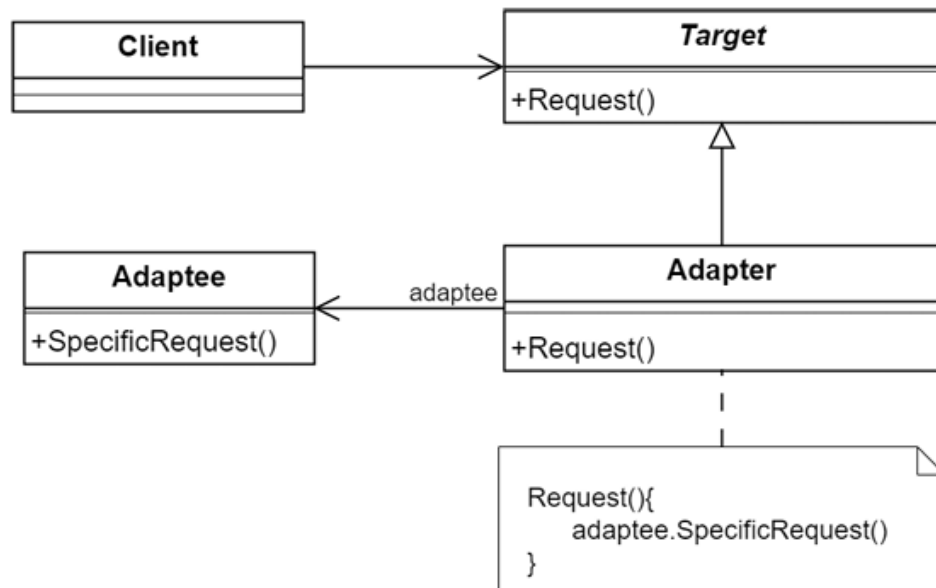
Висновки: Виконання лабораторної роботи дозволило ознайомитися з основними методами об'єктно-орієнтованого моделювання системи за допомогою UML. Було створено діаграму варіантів використання, яка наочно відображає взаємодію користувачів із системою та ключові функції, що вона надає. За допомогою діаграми класів було показано основні класи системи, їх атрибути та методи, а також зв'язки між класами, що дозволяє зрозуміти структурну організацію програми та логіку взаємодії об'єктів.

Відповіді на контрольні питання:

1. Яке призначення шаблону «Адаптер»?

Забезпечує сумісність між двома інтерфейсами, які спочатку не можуть працювати разом. Адаптер «обгортає» існуючий клас і перетворює його інтерфейс у той, який очікує клієнт.

2. Нарисуйте структуру шаблону «Адаптер».



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

- Client — використовує об'єкт через інтерфейс Target.
- Target — описує інтерфейс, який очікує клієнт.
- Adaptee — існуючий клас із несумісним інтерфейсом.
- Adapter — реалізує Target, але всередині викликає методи Adaptee.

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

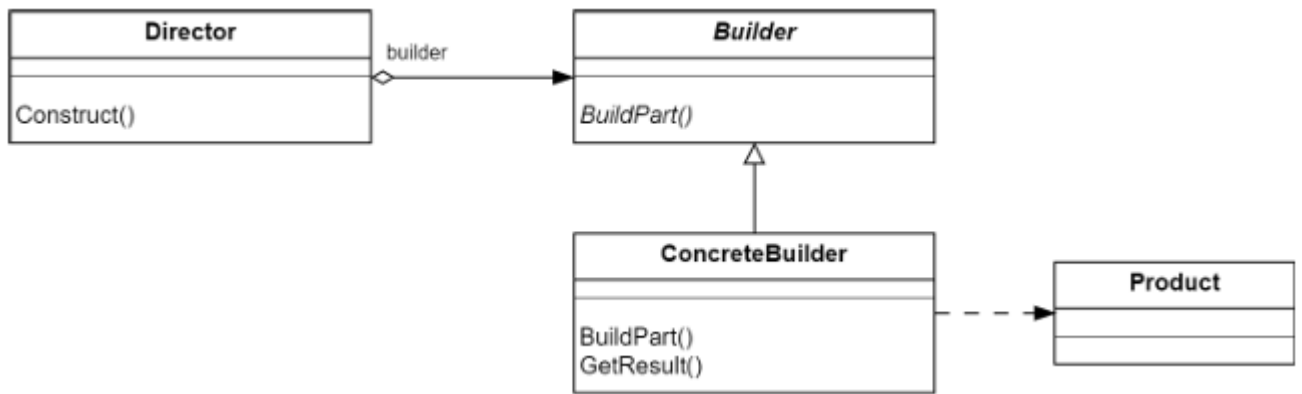
- Об'єктний адаптер — використовує композицію (всередині містить екземпляр Adaptee).
- Класовий адаптер — використовує спадкування (наслідує Adaptee і реалізує Target).

Об'єктний підхід гнучкіший, бо дозволяє адаптувати кілька класів одночасно.

5. Яке призначення шаблону «Будівельник»?

Відокремлює процес створення складного об'єкта від його представлення, дозволяючи будувати різні варіанти об'єкта з однієї і тієї ж послідовності кроків.

6. Нарисуйте структуру шаблону «Будівельник».



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

- Builder — інтерфейс, що описує кроки побудови об'єкта.
- ConcreteBuilder — конкретна реалізація, що створює частини продукту.
- Director — керує послідовністю викликів методів будівельника.
- Product — кінцевий складний об'єкт.

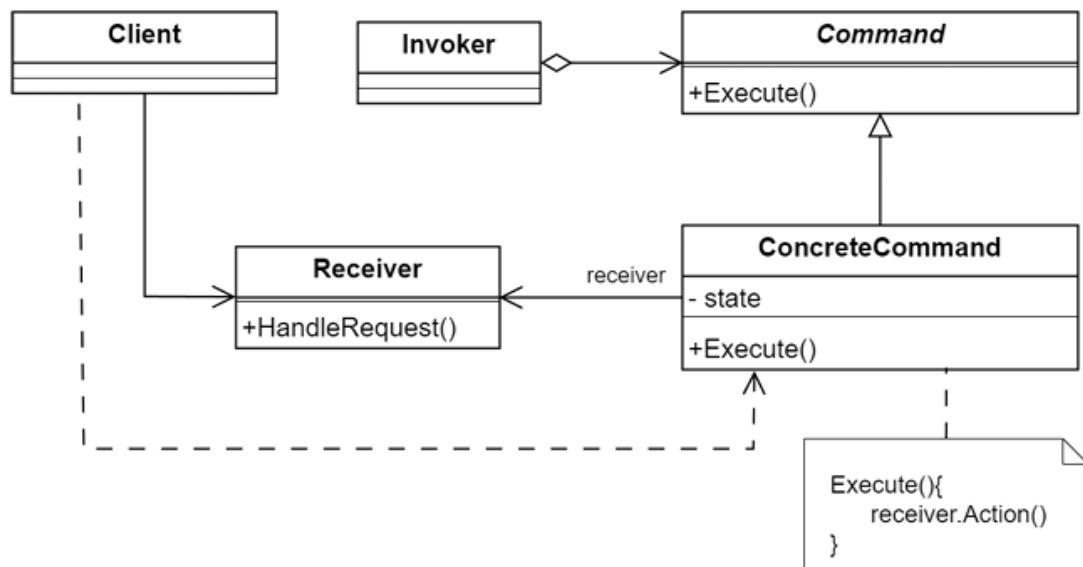
8. У яких випадках варто застосовувати шаблон «Будівельник»?

Коли об'єкт потрібно створювати поступово або в кілька кроків, і є багато варіантів конфігурації цього об'єкта.

9. Яке призначення шаблону «Команда»?

Інкапсулює запит (дію) як об'єкт, що дозволяє параметризувати об'єкти діями, ставити їх у чергу або скасовувати операції.

10. Нарисуйте структуру шаблону «Команда».



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

- Command — інтерфейс із методом Execute().
- ConcreteCommand — реалізує команду, викликаючи метод у Receiver.
- Receiver — виконує реальну роботу.
- Invoker — зберігає команду і викликає її виконання.
- Client — створює команду й призначає її виконавцю.

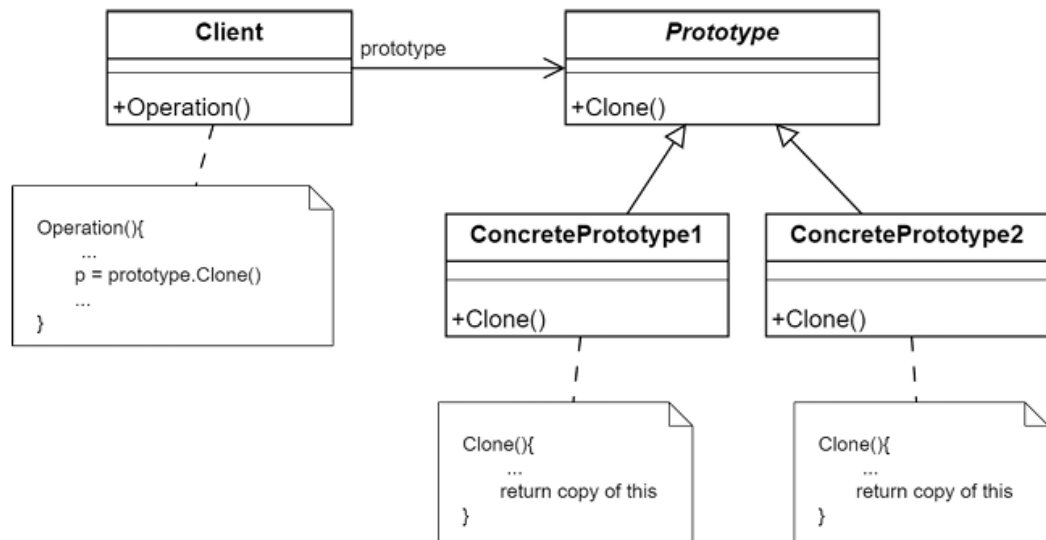
12. Розкажіть як працює шаблон «Команда».

Клієнт створює команду і задає отримувача. Коли Invoker викликає метод Execute(), команда звертається до Receiver, який виконує необхідну дію. Це дозволяє відділити відправника запиту від його виконання.

13. Яке призначення шаблону «Прототип»?

Створює нові об'єкти шляхом копіювання вже існуючих екземплярів (прототипів), замість створення через конструктор.

14. Нарисуйте структуру шаблону «Прототип».



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

- **Prototype** — оголошує метод `Clone()`.
- **ConcretePrototype** — реалізує клонування самого себе.
- **Client** — створює нові об'єкти через метод `Clone()`.

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

- Обробка запитів у системах підтримки (спочатку оператор, потім менеджер, потім директор).
- Фільтрація запитів у веб-додатках (middleware).
- Обробка подій у GUI (коли подія передається вгору по ієрархії елементів інтерфейсу).