



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 розроблення програмного забезпечення*»
Тема: «Патерни проектування»

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

Перевірив:
Мякий 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 «Mediator», «Facade», «Bridge», «Template method» 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.

Шаблон «Mediator» (Посередник) використовується для централізації взаємодії між об'єктами через окремий об'єкт-посередник. Замість прямого зв'язку об'єкти взаємодіють через посередника, що спрощує структуру системи, зменшує зв'язність між компонентами і полегшує внесення змін. Кожен об'єкт зберігає посилання на медіатор і через нього відпрацьовує логіку взаємодії. Основна перевага – зрозуміліша організація комунікацій та легке розширення системи, недолік – медіатор може стати «God Object» при надмірній складності.

Шаблон «Facade» (Фасад) створює єдиний спрощений інтерфейс доступу до складної підсистеми. Він приховує внутрішню структуру підсистеми та дозволяє змінювати її реалізацію без впливу на клієнтський код. Фасад полегшує використання підсистеми, зменшує «спагеті код», але знижує гнучкість прямого доступу до внутрішніх компонентів.

Шаблон «Bridge» (Міст) розділяє абстракцію і реалізацію, дозволяючи змінювати їх незалежно. Абстракція делегує виконання методів об'єкту реалізації. Це зменшує кількість перехресних підкласів при комбінації різних абстракцій і реалізацій, підвищує гнучкість та полегшує супровід. Недолік – введення додаткових проміжних рівнів підвищує складність структури.

Шаблон «Template Method» (Шаблонний метод) визначає загальний алгоритм у базовому класі, залишаючи частини реалізації підкласам. Це дозволяє повторно використовувати код і спрощує підтримку, оскільки загальна логіка залишається в базовому класі, а специфічні кроки реалізуються у підкласах. Недоліки – обмеженість скелета алгоритму, можливість порушення принципу підстановки Лісков і ускладнення підтримки при великій кількості віртуальних методів.

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

В даній частині було реалізовано логіку з додавання файлів та інтеграція з GoogleDrive. Було реалізовано відповідні сервіси, контролери та інтерфейси для виконання певного функціоналу.

```

1  using Google.Apis.Auth.OAuth2;
2  using Google.Apis.Drive.v3;
3  using Google.Apis.Services;
4  using Google.Apis.Util.Store;
5
6  namespace OnlineDiaryApp.Services
7  {
8      2 references
9      public class GoogleDriveService
10     {
11         private readonly string[] Scopes = { DriveService.Scope.DriveFile, DriveService.Scope.DriveReadOnly };
12         private readonly string ApplicationName = "OnlineDiaryApp";
13         private readonly DriveService _driveService;
14
15         0 references
16         public GoogleDriveService()
17         {
18             using var stream = new FileStream("wwwroot/credentials/credentials.json", FileMode.Open, FileAccess.Read);
19             var credPath = "wwwroot/credentials/token.json";
20
21             var credential = GoogleWebAuthorizationBroker.AuthorizeAsync(
22                 GoogleClientSecrets.FromStream(stream).Secrets,
23                 Scopes,
24                 "user",
25                 CancellationToken.None,
26                 new FileDataStore(credPath, true)).Result;
27
28             _driveService = new DriveService(new BaseClientService.Initializer()
29             {
30                 HttpClientInitializer = credential,
31                 ApplicationName = ApplicationName,
32             });
33
34             0 references
35             public async Task<string> UploadFileAsync(IFormFile file)
36             {
37                 var fileMetadata = new Google.Apis.Drive.v3.Data.File()
38                 {
39                     Name = file.FileName
40                 };
41
42                 using var stream = file.OpenReadStream();
43                 var request = _driveService.Files.Create(fileMetadata, stream, file.ContentType);
44                 request.Fields = "id, webViewLink, webContentLink";
45
46                 await request.UploadAsync();
47
48                 var uploadedFile = request.ResponseBody;
49
50                 return uploadedFile.WebViewLink;
51             }
52
53             0 references
54             public async Task<MemoryStream> DownloadFileAsync(string fileId)
55             {
56                 var request = _driveService.Files.Get(fileId);
57                 var stream = new MemoryStream();
58                 await request.DownloadAsync(stream);
59                 stream.Position = 0;
60                 return stream;
61             }
62
63             0 references
64             public async Task DeleteFileAsync(string fileId)
65             {
66                 await _driveService.Files.Delete(fileId).ExecuteAsync();
67             }
68         }
69     }

```

Код 1 – GoogleDriveService (Services)

GoogleDriveService – це сервіс, який відповідає за роботу з Google Drive у додатку. Він дозволяє: завантажувати файли з Google Drive на локальний комп'ютер, видаляти або оновлювати файли з хмарі, керувати файлами користувача без необхідності знати деталі роботи Google API. Тобто основне призначення сервісу – спростити інтеграцію додатку з Google Drive і надати простий спосіб зберігання та отримання файлів у хмарі.

```
FileService.cs*  GoogleDriveService.cs*  EmailService.cs  NotificationFacade.cs*
OnlineDiaryApp
1  using OnlineDiaryApp.Data;
2
3  namespace OnlineDiaryApp.Services
4  {
5      4 references
6      public class FileService
7      {
8          private readonly AppDbContext _context;
9
10         0 references
11         public FileService(AppDbContext context)
12         {
13             _context = context;
14         }
15
16         2 references
17         public async Task AddLinkFileAsync(int noteId, string fileName, string fileUrl)
18         {
19             var noteFile = new NoteFile
20             {
21                 NoteId = noteId,
22                 FileName = fileName,
23                 FilePath = fileUrl,
24                 MimeType = "link/google-drive"
25             };
26             _context.NoteFiles.Add(noteFile);
27             await _context.SaveChangesAsync();
28         }
29
30         1 reference
31         public async Task<IEnumerable<NoteFile>> GetFilesByNoteIdAsync(int noteId)
32         {
33             return _context.NoteFiles.Where(f => f.NoteId == noteId).ToListAsync();
34         }
35
36         1 reference
37         public async Task DeleteFileAsync(int fileId)
38         {
39             var file = await _context.NoteFiles.FindAsync(fileId);
40             if (file == null) return;
41             _context.NoteFiles.Remove(file);
42             await _context.SaveChangesAsync();
43         }
44     }
45 }
```

Код 2 – FileService (Services)

FileService – це сервіс для роботи з файлами, прив'язаними до нотаток у додатку. Основне призначення: додавання файлів або посилань на файли: метод `AddLinkFileAsync` дозволяє зберігати інформацію про файл (ім'я, шлях/URL, тип) у базі даних для конкретної нотатки, отримання файлів: метод `GetFilesByNoteIdAsync` повертає список усіх файлів, пов'язаних із заданою нотаткою, видалення файлів: метод `DeleteFileAsync` дозволяє видалити файл з бази даних. Сервіс спрощує роботу з файлами всередині додатку, абстрагуючи деталі роботи з базою даних і забезпечуючи єдиний інтерфейс для взаємодії з файлами.

NoteController.cs ×

FileService.cs*

GoogleDriveService.cs*

EmailService.cs

Notifica

OnlineDiaryApp

```

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         private readonly FileService _fileService;
14
15         0 references
16         public NoteController(
17             NoteService noteService,
18             ReminderService reminderService,
19             TagService tagService,
20             FileService fileService)
21         {
22             _noteService = noteService;
23             _reminderService = reminderService;
24             _tagService = tagService;
25             _fileService = fileService;
26         }
27
28         // GET: /Note/
29         0 references
30         public async Task<IActionResult> Index(string? sortBy, string? tag)
31         {
32             var userId = GetUserId();
33             if (userId == null) return RedirectToAction("Login", "User");
34
35             ISortStrategy? strategy = null;
36             if (!string.IsNullOrEmpty(sortBy))
37             {
38                 strategy = sortBy.ToLower() switch
39                 {
40                     "date" => new SortByDateStrategy(),
41                     "tag" when !string.IsNullOrEmpty(tag) => new SortByTagStrategy(tag),
42                     "title" => new SortByTitleStrategy(),
43                     _ => null
44                 };
45             }
46
47             var notes = await _noteService.GetAllNotesByUserAsync(userId.Value, strategy);
48             ViewBag.SortBy = sortBy;
49             ViewBag.SelectedTag = tag;
50             ViewBag.Tags = await _tagService.GetAllTagsAsync(userId.Value);
51
52             return View(notes);
53         }
54
55         [HttpGet]
56         0 references
57         public async Task<IActionResult> Create()
58         {
59             var userIdString = HttpContext.Session.GetString("UserId");
60             if (!int.TryParse(userIdString, out int userId))
61                 return RedirectToAction("Login", "User");
62
63             ViewBag.Tags = await _tagService.GetAllTagsAsync(userId);
64             return View();
65         }
66
67         [HttpPost]
68         0 references
69         public async Task<IActionResult> Create(
70             string title,
71             string content,
72             List<int>? tagIds,
73             DateTime? reminderDate,
74             List<string>? GoogleDriveLinks)
75         {
76             var userId = int.Parse(HttpContext.Session.GetString("UserId") ?? "0");
77
78             var note = await _noteService.CreateNoteAsync(title, content, userId, tagIds ?? new List<int>());
79
80             if (GoogleDriveLinks != null)
81             {
82                 foreach (var link in GoogleDriveLinks)
83                 {
84                     await _fileService.AddLinkFileAsync(note.Id, "Google Drive file", link);
85                 }
86             }
87
88             await _noteService.UpdateNoteAsync(note, tagIds ?? new List<int>(), reminderDate);
89
90             return RedirectToAction("Index");
91         }
92     }

```

Код 3.1 – NoteController (Controllers)

```

89
90 // GET: /Note/Edit/5
91 0 references
92 public async Task<IActionResult> Edit(int id)
93 {
94     var note = await _noteService.GetNoteByIdAsync(id);
95     if (note == null) return NotFound();
96
97     var userId = GetUserId();
98     if (userId == null) return RedirectToAction("Login", "User");
99
100     ViewBag.Tags = await _tagService.GetAllTagsAsync(userId.Value);
101     ViewBag.Reminder = await _reminderService.GetReminderByNoteIdAsync(id);
102
103     return View(note);
104 }
105
106 // POST: /Note/Edit/5
107 [HttpPost]
108 0 references
109 public async Task<IActionResult> Edit(
110     int id,
111     string title,
112     string content,
113     List<int>? tagIds,
114     DateTime? remindAt,
115     List<string>? GoogleDriveLinks,
116     List<int>? DeletedFileIds)
117 {
118     var note = await _noteService.GetNoteByIdAsync(id);
119     if (note == null) return NotFound();
120
121     note.Title = title;
122     note.Content = content;
123
124     // Видалення файлів
125     if (DeletedFileIds != null)
126     {
127         foreach (var fileId in DeletedFileIds)
128             await _fileService.DeleteFileAsync(fileId);
129     }
130
131     // Додавання нових файлів
132     if (GoogleDriveLinks != null && GoogleDriveLinks.Any())
133     {
134         foreach (var link in GoogleDriveLinks)
135             await _fileService.AddLinkFileAsync(note.Id, "Google Drive file", link);
136     }
137
138     await _noteService.UpdateNoteAsync(note, tagIds ?? new List<int>(), remindAt);
139
140     return RedirectToAction("Index");
141 }
142
143 // GET: /Note/Delete/5
144 0 references
145 public async Task<IActionResult> Delete(int id)
146 {
147     var reminder = await _reminderService.GetReminderByNoteIdAsync(id);
148     if (reminder != null)
149         await _reminderService.DeleteReminderAsync(reminder.Id);
150
151     await _noteService.DeleteNoteAsync(id);
152     return RedirectToAction("Index");
153 }
154
155 // GET: /Note/Details/5
156 0 references
157 public async Task<IActionResult> Details(int id)
158 {
159     var note = await _noteService.GetNoteByIdAsync(id);
160     if (note == null) return NotFound();
161
162     ViewBag.Reminder = await _reminderService.GetReminderByNoteIdAsync(id);
163
164     // Підвантажуємо файли для відображення
165     note.Files = (await _fileService.GetFilesByNoteIdAsync(note.Id)).ToList();
166
167     return View(note);
168 }
169
170 // GET: /Note/Search
171 0 references
172 public async Task<IActionResult> Search(string keyword)
173 {
174     var userId = GetUserId();
175     if (userId == null) return View(new List<Note>());
176
177     var notes = await _noteService.SearchByTitleAsync(keyword);
178     notes = notes.Where(n => n.UserId == userId.Value);
179     return View("Index", notes);
180 }
181
182 // Допоміжний метод для отримання UserId
183 3 references
184 private int? GetUserId()
185 {
186     var userIdString = HttpContext.Session.GetString("UserId");
187     if (!int.TryParse(userIdString, out int userId)) return null;
188     return userId;
189 }

```

Код 3.2 – NoteController (Controllers)

NoteController — це контролер, який відповідає за керування логікою роботи з нотатками у додатку. Його основне призначення — обробляти HTTP-запити, пов’язані зі створенням, переглядом, редагуванням та видаленням нотаток, а також координувати взаємодію між моделлю, представленнями (Views) та сервісами. Отже, NoteController виступає посередником між користувачем і бізнес-логікою, забезпечуючи правильну обробку дій у системі та передачу даних до відповідних представлень.

3. Реалізувати один з розглянутих шаблонів за обраною темою.

Для даного проєкту було обрано патерн Facade. Шаблон реалізований таким чином:

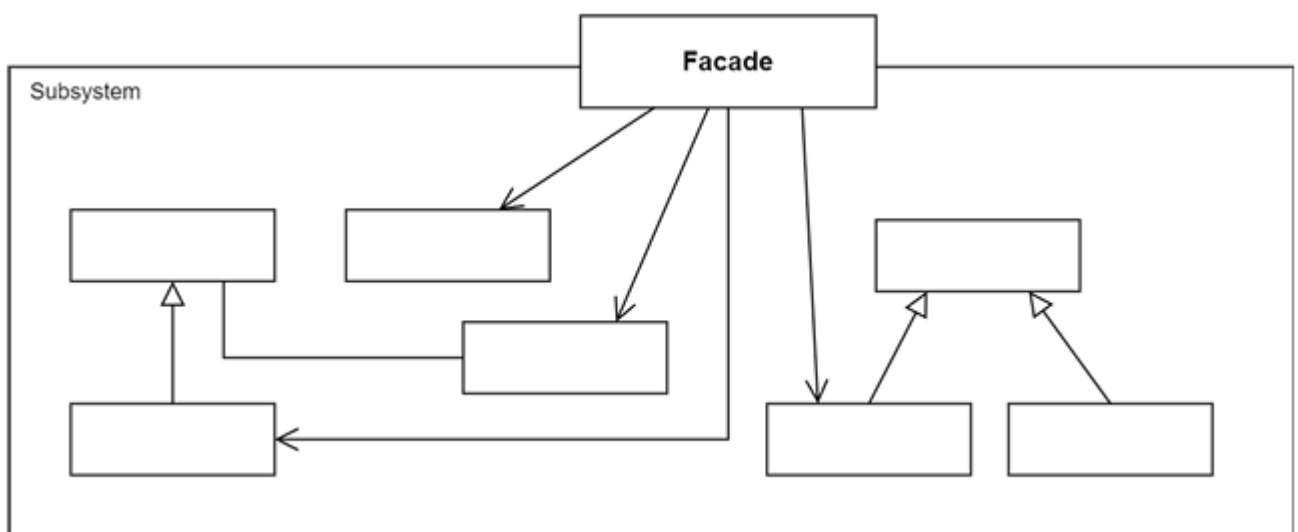


Рис.1 – Структура патерну Фасад

Facade: NotificationFacade — надає спрощений інтерфейс для надсилання повідомлень користувачам, інкапсулюючи роботу з підсистемою, яка відповідає за конкретні типи повідомлень (наразі — EmailService). Завдяки цьому клієнтам (спостерігачам) не потрібно знати, як саме відправляються повідомлення. Також він надає методи: SendEmailNotificationAsync(to, subject, message) — простий метод для відправки email, SendNotificationAsync(to, subject, message, type) —

гнучкий метод, який дозволяє додати інші типи повідомлень (SMS, Push) у майбутньому.

Subsystem: EmailService — це низькорівнева підсистема, яка безпосередньо відповідає за відправлення електронних листів через SMTP за допомогою бібліотек MailKit і MimeKit. Вона знає про технічні деталі: адресу SMTP-сервера, порт, облікові дані та налаштування безпеки. У патерні Фасад цей клас виконує роль конкретного виконавця — він не взаємодіє з клієнтом напряду, а працює лише через NotificationFacade, який координує його використання. Фасад приховує всі деталі конфігурації, тому клієнт може просто сказати: “відправ листа”, не знаючи, як саме це реалізовано.

Client: EmailObserver — це реалізація спостерігача, яка реагує на події нагадувань, надсилаючи користувачу листа. Він використовує інтерфейс IEmailSender, який, у свою чергу, реалізується через EmailService. Таким чином, EmailObserver — це "місток" між бізнес-логікою нагадувань та поштовою підсистемою. Фасад координує цю взаємодію, але не втручається у внутрішню логіку спостерігача.

Результат

Завдяки застосуванню патерну Фасад система стала: простішою у використанні — клієнту достатньо одного виклику, замість кількох складних кроків; гнучкою та розширюваною — можна змінювати реалізацію будь-якої підсистеми (наприклад, перейти з SMTP на API) без зміни коду клієнтів; з низьким зв'язуванням — фасад ізолює клієнтів від деталей реалізації підсистем; більш надійною — координує виклики у правильному порядку, зменшуючи ризик помилок.

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

NotificationFacade.cs* EmailObserver.cs ReminderService.cs ReminderSubject.cs ReminderBackgroundService.cs IEmailSender.cs IRemind

OnlineDiaryApp OnlineDiaryApp.Services.NotificationFacade

```

1  using OnlineDiaryApp.Interfaces;
2
3  namespace OnlineDiaryApp.Services
4  {
5      4 references
6      public class NotificationFacade : INotificationFacade
7      {
8          private readonly EmailService _emailService;
9
10         0 references
11         public NotificationFacade(EmailService emailService)
12         {
13             _emailService = emailService;
14         }
15
16         2 references
17         public async Task SendEmailNotificationAsync(string to, string subject, string message)
18         {
19             await _emailService.SendEmailAsync(to, subject, message);
20         }
21
22         1 reference
23         public async Task SendNotificationAsync(string to, string subject, string message, string type = "email")
24         {
25             switch (type.ToLower())
26             {
27                 case "email":
28                     await _emailService.SendEmailAsync(to, subject, message);
29                     break;
30                 default:
31                     throw new NotSupportedException($"Тип '{type}' не підтримується у NotificationFacade.");
32             }
33         }
34     }
35 }

```

Код 4 – NotificationFacade (Services)

EmailService.cs* NotificationFacade.cs* EmailObserver.cs ReminderService.cs ReminderSubject.cs ReminderBackgroundService.cs

OnlineDiaryApp OnlineDiaryApp.Services.EmailService

```

1  using MailKit.Net.Smtp;
2  using MimeKit;
3
4  namespace OnlineDiaryApp.Services
5  {
6      5 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         3 references
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 SmtplibClient();
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     }
30 }

```

Код 5 – EmailService (Services)

```

1  using OnlineDiaryApp.Interfaces;
2  using OnlineDiaryApp.Models;
3  using OnlineDiaryApp.Services;
4
5  namespace OnlineDiaryApp.Observers
6  {
7      public class EmailObserver : IReminderObserver
8      {
9          private readonly NotificationFacade _notificationFacade;
10
11         public EmailObserver(NotificationFacade notificationFacade)
12         {
13             _notificationFacade = notificationFacade;
14         }
15
16         public async Task OnReminderChangedAsync(Reminder reminder, string action)
17         {
18             if (action == "created" || action == "updated" || action == "time_reached")
19             {
20                 var user = reminder.User;
21                 if (user != null && !string.IsNullOrEmpty(user.Email))
22                 {
23                     string subject = $"Нагадування: {reminder.Note?.Title ?? "Без назви"}";
24                     string body = $"Ваше нагадування ({action}).\n\n{reminder.Note?.Content ?? "Без тексту"}";
25
26                     await _notificationFacade.SendEmailNotificationAsync(user.Email, subject, body);
27                 }
28             }
29         }
30     }
31 }

```

Код 6 – EmailObserver (Observers)

- 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: У ході лабораторної роботи було реалізовано патерн Фасад для системи надсилення сповіщень у додатку онлайн-щоденника. Основна мета патерну полягала у спрощенні взаємодії клієнтів із підсистемою відправки повідомлень, що дозволяє працювати лише з одним об'єктом NotificationFacade замість безпосередньої роботи з EmailService та іншими сервісами. Це забезпечує відокремлення реалізації від використання: клієнти не залежать від деталей роботи з SMTP чи інших каналів надсилення, а у разі змін у сервісі або додавання нових типів сповіщень код клієнтів залишатиметься незмінним. Застосування фасаду дозволяє легко додавати нові канали сповіщень, підвищує тестованість і надійність системи, а також сприяє слабкому зв'язуванню між ReminderService та

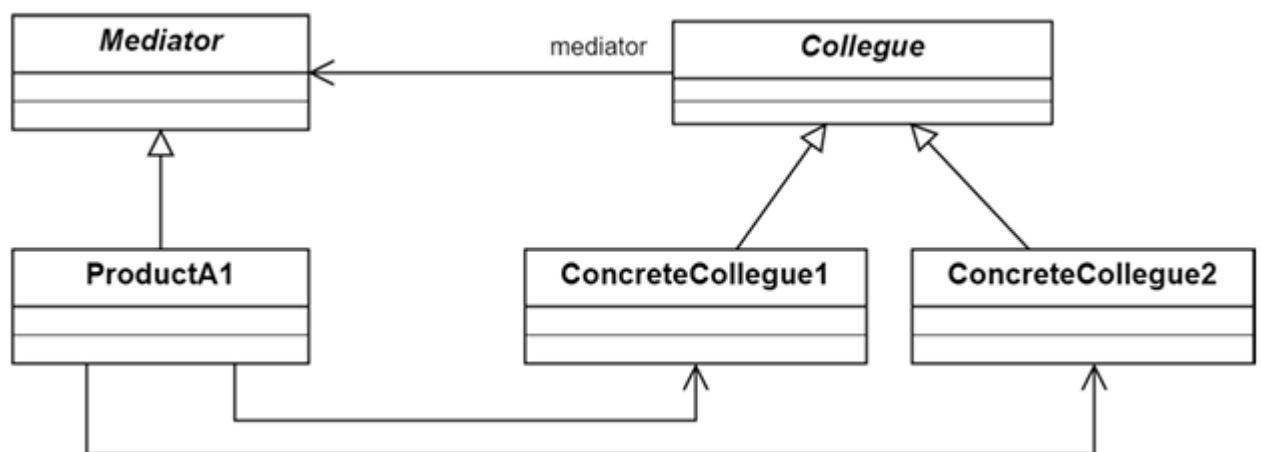
механізмом надсилання повідомлень. У результаті було досягнуто гнучкої і зрозумілої архітектури, що дозволяє централізовано керувати сповіщеннями та забезпечує розширюваність системи без необхідності модифікації існуючих класів.

Відповіді на контрольні питання:

1. Призначення шаблону «Посередник»

Шаблон «Посередник» (Mediator) призначений для централізації взаємодії між об'єктами. Замість того щоб об'єкти безпосередньо посилали повідомлення один одному, вони взаємодіють через посередника, що зменшує зв'язування між компонентами та спрощує підтримку і розширення системи.

2. Структура шаблону «Посередник»



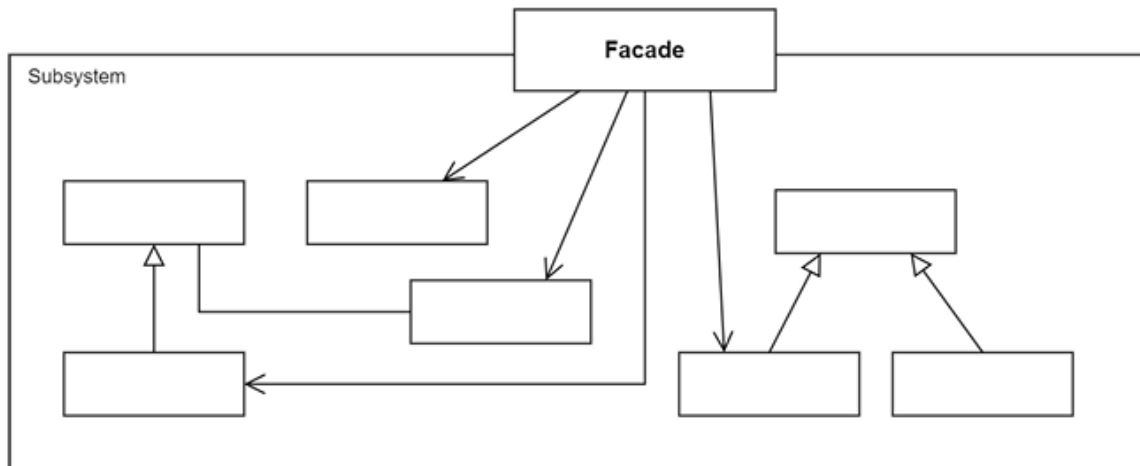
3. Класи та взаємодія шаблону «Посередник»

- Mediator (Інтерфейс Посередника) – описує методи для взаємодії колег.
- ConcreteMediator (Конкретний Посередник) – реалізує логіку взаємодії між колегами.
- Colleage (Колега) – об'єкти, які посилають повідомлення через посередника.
- Взаємодія: колеги не звертаються один до одного напряму, а передають дані через посередника, який координує комунікацію.

4. Призначення шаблону «Фасад»

Шаблон «Фасад» (Facade) спрощує роботу з комплексною системою або підсистемою, надаючи єдиний інтерфейс для взаємодії. Клієнти працюють через фасад, не залежачи від складної внутрішньої структури системи.

5. Структура шаблону «Фасад»



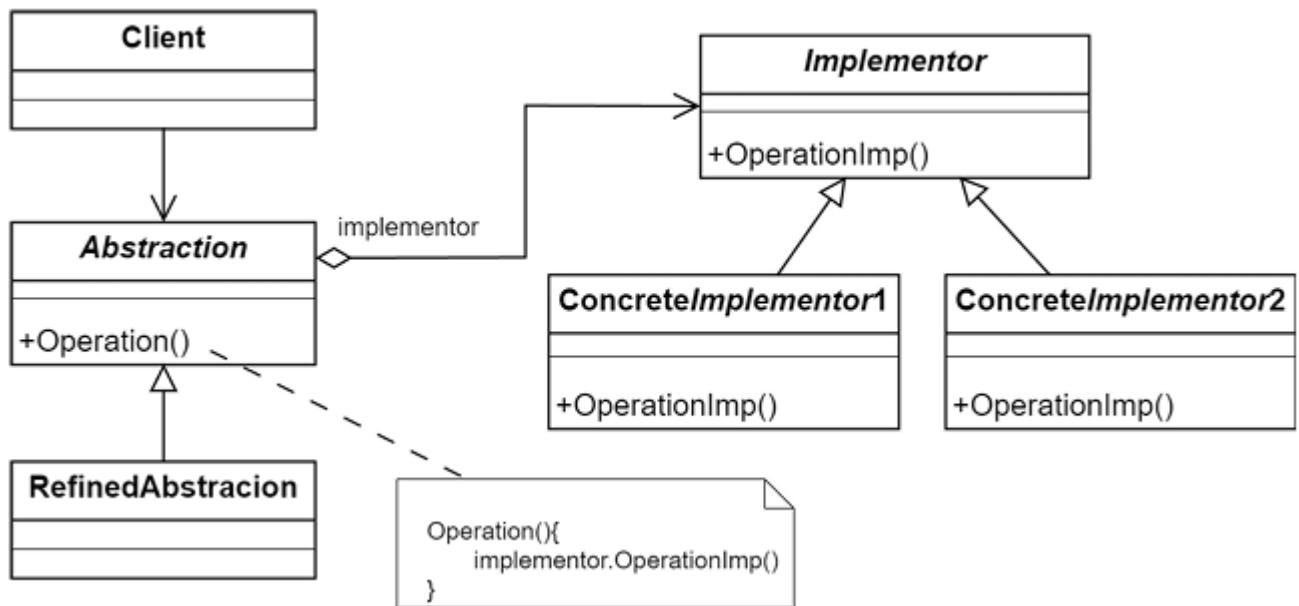
6. Класи та взаємодія шаблону «Фасад»

- Facade (Фасад) – надає спрощений інтерфейс для клієнтів.
- Subsystem (Підсистема) – клас або група класів, що реалізують бізнес-логіку.
- Взаємодія: клієнт викликає методи фасаду → фасад делегує запити підсистемам.

7. Призначення шаблону «Міст»

Шаблон «Міст» (Bridge) розділяє абстракцію і реалізацію, щоб вони могли змінюватися незалежно. Це дозволяє підключати різні реалізації до одного інтерфейсу без зміни клієнтського коду.

8. Структура шаблону «Міст»



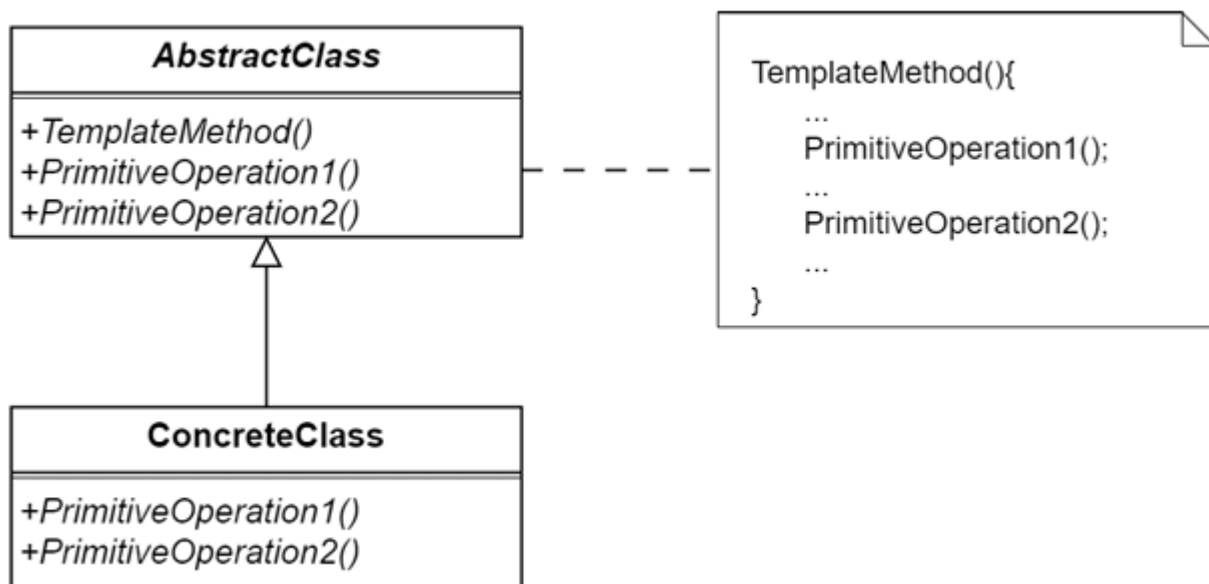
9. Класи та взаємодія шаблону «Міст»

- Abstraction (Абстракція) – визначає високорівневий інтерфейс для клієнта.
- RefinedAbstraction (Уточнена абстракція) – розширює абстракцію.
- Implementor (Інтерфейс реалізації) – описує низькорівневі методи.
- ConcreteImplementor (Конкретна реалізація) – реалізує методи Implementor.
- Взаємодія: абстракція делегує виконання методів конкретним реалізаціям, не знаючи деталей.

10. Призначення шаблону «Шаблонний метод»

Шаблон «Шаблонний метод» (Template Method) визначає скелет алгоритму в методі, залишаючи реалізацію деяких кроків підкласам. Це дозволяє уникнути дублювання коду та забезпечує гнучкість.

11. Структура шаблону «Шаблонний метод»



12. Класи та взаємодія шаблону «Шаблонний метод»

- AbstractClass (Абстрактний клас) – містить шаблонний метод і деякі реалізації кроків алгоритму.
- ConcreteClass (Конкретний клас) – реалізує конкретні кроки алгоритму.
- Взаємодія: клієнт викликає шаблонний метод → він визначає послідовність кроків → підкласи реалізують конкретні кроки.

13. Відмінність «Шаблонного методу» від «Фабричного методу»

- «Шаблонний метод» визначає алгоритм з деякими змінними кроками в підкласах.
- «Фабричний метод» створює об'єкти через інтерфейс або абстрактний клас, делегуючи створення підкласам. Головне у фабричному методі – інкапсуляція створення об'єктів, а не послідовності кроків.

14. Функціональність, яку додає шаблон «Міст»

Шаблон «Міст» дозволяє змінювати абстракцію і реалізацію незалежно, підключати нові реалізації без зміни клієнтського коду, зменшує зв'язування між класами та підвищує гнучкість і розширюваність системи.