Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра автоматики та управління в технічних системах

Лабораторна робота №5

Технології розроблення програмного забезпечення

«Шаблони «Adapter», «Builder», «Command», «Chain of Responsibility», «Prototype»»

Варіант 2

Виконала
студентка групи ІА–24:
Кійко А. О.

Перевірив
Мягкий М. Ю.

Київ 2024

ЗМІСТ

Тема: HTTP-сервер (state, builder, factory method, mediator, composite, p2p). Сервер повинен мати можливість розпізнавати вхідні запити і формувати коректні відповіді (згідно протоколу HTTP), надавати сторінки chtml (html сторінки з додаванням найпростіших C# конструкцій на розсуд студента), вести статистику вхідних запитів, обробку запитів у багатопотоковому/подієвому режимах.

Короткі теоретичні відомості

Шаблони проектування — це стандартизовані рішення для типових задач, які часто виникають під час розробки програмного забезпечення. Вони є своєрідними схемами, які допомагають ефективно вирішувати проблеми, роблячи код гнучкішим, розширюваним та зрозумілим.

Антипатерни, навпаки, демонструють погані рішення та практики, які, хоча й можуть працювати в певних умовах, зазвичай ведуть до зниження продуктивності й якості програмного забезпечення. Розуміння антипатернів є важливим аспектом професійного зростання, адже дозволяє розробникам уникати типових помилок і виявляти слабкі місця в системах.

У цій лабораторній роботі розглядаються наступні шаблони:

Шаблон Adapter використовується для забезпечення сумісності між несумісними класами. Його головна ідея полягає в створенні проміжного класу, який перетворює інтерфейс одного класу на інтерфейс, зрозумілий іншому. Adapter дозволяє взаємодіяти об'єктам, які зазвичай не можуть працювати разом, без змін у їх коді.

Шаблон Builder забезпечує створення складних об'єктів покроково. Він дозволяє ізолювати процес створення об'єкта від його представлення, що робить код більш гнучким і полегшує підтримку. Використання Builder

актуальне, коли об'єкт має багато параметрів або може існувати в різних конфігураціях.

Command — це шаблон, який інкапсулює дію або запит у вигляді об'єкта. Це дозволяє передавати дії як параметри, створювати черги команд, підтримувати скасування та повторне виконання операцій. Основна перевага Command у тому, що він сприяє зниженню зв'язності між відправником і виконавцем дії.

Chain of Responsibility організовує обробку запитів через ланцюг обробників. Кожен обробник приймає рішення, чи варто обробити запит, або ж передати його наступному обробнику в ланцюзі. Chain of Responsibility дозволяє динамічно змінювати послідовність обробки запитів і додає гнучкості в реалізацію логіки.

Шаблон Prototype використовується для створення нових об'єктів шляхом копіювання вже існуючих. Він корисний, коли створення об'єкта є дорогим процесом, а копіювання забезпечує значне зниження витрат. Prototype дозволяє зберігати стан об'єкта, що особливо важливо у випадках складних конфігурацій.

Хід роботи

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми.
4. Скласти звіт про виконану роботу.

Реалізація шаблону Builder

Реалізація шаблону “Builder” у моєму проєкті полягає у побудові відповіді на запит клієнта до Http-сервера. Формування (побудова) відповіді складається з наступних частин (кроків):

- додавання версії HTTP-протоколу;
- додавання коду Http-статусу виконання запиту;
- додавання стандартних заголовків (тип даних, тип кодування символів, довжина змісту відповіді тощо);
- додавання змісту відповіді.

З цією метою мною було створено інтерфейс IBuilder, який визначає методи, що забезпечують реалізацію всіх можливих кроків для формування Http-відповіді. Оскільки відповідь, що надається клієнту, представляє собою строку, всі методи інтерфейсу повертають тип string.

```
6   internal interface IBuilder
7   {
8       /// <summary>
9       /// Версія протоколу
10      /// </summary>
11      /// <returns></returns>
            6 references
12      public string BuildVersion();
13
14      /// <summary>
15      /// Статус виконання запиту
16      /// </summary>
17      /// <returns></returns>
            6 references
18      public string BuildStatus();
19
20      /// <summary>
21      /// Заголовки відповіді
22      /// </summary>
23      /// <returns></returns>
            6 references
24      public string BuildHeaders();
25
26      /// <summary>
27      /// Зміст (тіло) відповіді
28      /// </summary>
29      /// <returns></returns>
            6 references
30      public string BuildContentBody();
31
32  }
```

Рис. 1 - Інтерфейс IBuilder

Для реалізації інтерфейса IBuilder було створено наступні класи, що реалізують кроки формування Http-відповіді у залежності від типу запиту:

- BuilderPage - Builder-клас побудови відповіді на запит Web-сторінки;
- BuilderStat - Builder-клас побудови відповіді на запит статистики;
- BuilderInvalid - Builder-клас побудови відповіді на не валідний запит.

```
using HttpServApp.Models;
using System.Text;

namespace HttpServApp.Builder
{
  /// <summary>
  /// Builder-клас побудови відповіді на запит Web-сторінки
  /// </summary>
  internal class BuilderPage : IBuilder
  {
    HttpRequestPage httpRequestPage;
    public BuilderPage(HttpRequest httpRequest)
    {
      httpRequestPage = httpRequest as HttpRequestPage ?? throw new
ArgumentNullException(nameof(httpRequestPage));
    }

    public string BuildVersion() => "HTTP / " + (httpRequestPage.Version ??
"1.1");
    public string BuildStatus()
    {
      // Якщо сторінка, що запитується, не знайдена в репозиторії => STATUS =
NOT_FOUND
      // Якщо знайдена => STATUS = OK
      httpRequestPage.Status = !File.Exists(httpRequestPage.Path) ?
StatusEnum.NOT_FOUND : StatusEnum.OK;

      return $"{(int)httpRequestPage.Status} {httpRequestPage.Status}";
    }

    public string BuildHeaders() =>
        $"\nContent-Type:{httpRequestPage.ContentTypeRequest ??
"text/plain"};charset=UTF-8;" +
        $"\nConnection: close\n";

    public virtual string BuildContentBody()
    {
```

```
      // Якщо сторінка, що запитується, не знайдена в репозиторії => виводимо
повідомлення про це у відповідь
      if (!File.Exists(httpRequestPage.Path))
      {
        httpRequestPage.Message = $"Файл сторінки '{httpRequestPage.Path}' не
знайдений";
        httpRequestPage.Response = new HttpResponse(
            DateTime.Now, Encoding.UTF8.GetByteCount(httpRequestPage.Message));

        Console.WriteLine(httpRequestPage.Message);
        return $"Content-Length:{httpRequestPage.Response?.ContentLength ??
0}\n\n{httpRequestPage.Message}";
      }
      // Якщо сторінка, що запитується, знайдена в репозиторії => повертаємо
зміст сторінки
      else
      {
        // Зчитування змісту Web-сторінки
        string htmlResponse = string.Empty;
        using (StreamReader reader = new StreamReader(httpRequestPage.Path))
        {
          htmlResponse = reader.ReadToEnd();
        }
        httpRequestPage.Response = new HttpResponse(
            DateTime.Now, Encoding.UTF8.GetByteCount(htmlResponse));

        Console.WriteLine($"Сторінка {httpRequestPage.Path} сформована");
        return $"Content-Length:{httpRequestPage.Response?.ContentLength ??
0}\n\n{htmlResponse}";
      }
    }
  }
}
```

Рис. 2 - Клас BuilderPage

```
using HttpServApp.Models;
using System.Text;

namespace HttpServApp.Builder
{
  /// <summary>
  /// Builder-клас побудови відповіді на запит статистики
  /// </summary>
  internal class BuilderStat : IBuilder
  {
    private readonly HttpRequestStat httpRequestStat;
    private readonly List<HttpRequest>? periodRequests;
```

```
    public BuilderStat(HttpRequest httpRequest)
    {
      try
      {
        httpRequestStat = httpRequest as HttpRequestStat ?? throw new
ArgumentNullException(nameof(httpRequestStat));
        periodRequests =
httpRequestStat.Repository.GetRequestsByPeriod(httpRequestStat.DateBeg,
httpRequestStat.DateEnd);
        httpRequestStat.CntRows = periodRequests.Count;
      }
      catch (Exception exc)
      {
        httpRequestStat.Message = $"Відповідь на запит статистики не сформована:
{exc.Message}";
      }
    }

    public string BuildVersion() => "HTTP / " + (httpRequestStat.Version ??
"1.1");
    public string BuildStatus()
    {
      // Якщо periodRequests не визначено (сталась помилка при виборі даних), то
STATUS = BAD_SERVER
      if (periodRequests == null)
        httpRequestStat.Status = StatusEnum.BAD_SERVER;
      else
        // Якщо записи (запити) в репозиторії БД за обраний період не знайдено
=> STATUS = NOT_FOUND
        // Якщо знайдені => STATUS = OK
        httpRequestStat.Status =
           periodRequests.Count == 0
           ? StatusEnum.NOT_FOUND
           : StatusEnum.OK;

      return $"{(int)httpRequestStat.Status} {httpRequestStat.Status}";
    }

    public string BuildHeaders() =>
        $"\nContent-Type:{httpRequestStat.ContentTypeRequest ??
"text/plain"};charset=UTF-8;" +
        $"\nConnection: close\n";

    public virtual string BuildContentBody()
    {
      if (periodRequests == null)
      {
        httpRequestStat.Response = new HttpResponse(
```

```
            DateTime.Now, Encoding.UTF8.GetByteCount(httpRequestStat.Message ??
"Помилка при виборі даних статистики"));
        Console.WriteLine(httpRequestStat.Message);

        return $"Content-Length:{httpRequestStat.Response?.ContentLength ??
0}\n\n" +
            $"{httpRequestStat.Message ?? "Помилка при виборі даних
статистики"}";
      }
      else
      {
        string header = $"Дані статистики за період " +
        $"з {httpRequestStat.DateBeg:dd.MM.yyyy HH:mm:ss} по
{httpRequestStat.DateEnd:dd.MM.yyyy HH:mm:ss}";
        Console.WriteLine($"{header}{(periodRequests.Count > 0 ? "вибрані" :
"відсутні")}");

        string tableResponse = "";
        int indexRow = 1;

        httpRequestStat.Repository.Requests.ForEach(req =>
        {
          tableResponse +=
                      $"\t\t\t\t<tr class=\"{GetClassRow(req.TypeRequest)}\">\n"
+
                          $"\t\t\t\t\t<th scope=\"row\" >{indexRow++}</th>\n" +
                          $"\t\t\t\t\t<td
scope=\"col\">{req.DateTimeRequest:dd.MM.yyyy HH:mm:ss}</td>\n" +
                          $"\t\t\t\t\t<td
scope=\"col\">{req.TypeRequest}</td>\n" +
                          $"\t\t\t\t\t<td scope=\"col\">{req.IpAddress}</td>\n"
+
                          $"\t\t\t\t\t<td scope=\"col\">{req.Status}</td>\n" +
                          $"\t\t\t\t\t<td
scope=\"col\">{(req.Response?.StatusSend == 1 ? "Так" : "Ні")}</td>\n" +
                          $"\t\t\t\t\t<td
scope=\"col\">{req.ContentTypeRequest}</td>\n" +
                          $"\t\t\t\t\t<td scope=\"col\">{req.Method}</td>\n" +
                          $"\t\t\t\t\t<td scope=\"col\">{req.Version}</td>\n" +
                          $"\t\t\t\t\t<td scope=\"col\">" +
                              $"{(req.TypeRequest == TypeRequestEnum.СТОРІНКА ?
((HttpRequestPage)req).Path : string.Empty)}" +
                          $"</td>\n" +
                          $"\t\t\t\t\t<td scope=\"col\">" +
                              $"{(req.TypeRequest == TypeRequestEnum.СТАТИСТИКА
                              ?
((HttpRequestStat)req).DateBeg.ToString("dd.MM.yyyy HH:mm:ss") + "-" +
```

```
((HttpRequestStat)req).DateEnd.ToString("dd.MM.yyyy HH:mm:ss") + ": " +
                                    ((HttpRequestStat)req).CntRows + " запит(ів)"
                                : string.Empty)}" +
                            $"</td>\n" +
                            $"\t\t\t\t\t<td scope=\"col\">{req.Message}</td>\n" +
                        "\t\t\t\t</tr>\n";
        });

        string bodyResponse = TemplateBodyResponse(header, tableResponse);
        // Формуємо об'єкт Response
        httpRequestStat.Response = new HttpResponse(
            DateTime.Now, Encoding.UTF8.GetByteCount(bodyResponse));

        return $"Content-Length:{httpRequestStat.Response?.ContentLength ??
0}\n\n{bodyResponse}";
      }

    }
```

Рис. 3 - Клас BuilderStat

```
using HttpServApp.Models;
using System.Text;

namespace HttpServApp.Builder
{
  /// <summary>
  /// Builder-клас побудови відповіді на не валідний запит
  /// </summary>
  internal class BuilderInvalid : IBuilder
  {
    HttpRequestInvalid httpRequestInvalid;
    public BuilderInvalid(HttpRequest httpRequest)
    {
      httpRequestInvalid = httpRequest as HttpRequestInvalid ?? throw new
ArgumentNullException(nameof(httpRequestInvalid));
    }
    public string BuildVersion() => "HTTP / " + (httpRequestInvalid.Version ??
"1.1");

    public virtual string BuildStatus() => $"{(int)httpRequestInvalid.Status}
{httpRequestInvalid.Status}";

    public string BuildHeaders() =>
        $"\nContent-Type:{httpRequestInvalid.ContentTypeRequest ??
"text/plain"};charset=UTF-8;" +
        $"\nConnection: close\n";
```

```
    public string BuildContentBody()
    {
      httpRequestInvalid.Response = new HttpResponse(
          DateTime.Now, Encoding.UTF8.GetByteCount(httpRequestInvalid.Message ??
string.Empty));

      return $"Content-Length:{httpRequestInvalid.Response?.ContentLength ??
0}\n\n" +
          $"{httpRequestInvalid.Message ?? string.Empty}";
    }

  }
}
```

Рис. 4 - Клас BuilderInvalid

Класом-розпорядником (director), що "керує побудовою" відповіді, є відповідний клас стану запиту після валідації. Саме він створює об’єкт “будівельника” відповідного типу та послідовно викликає методи для формування строки Http-відповіді у визначеному порядку.

```
10      public void ProcessingHandler(HttpRequest httpRequest, Socket socket)
11      {
12          // Будуємо відповідь за допомогою методів інтерфейсу IBuilder
13          IBuilder builder = new BuilderPage(httpRequest);
14          string htmlResponse =
15              builder.BuildVersion() +
16              builder.BuildStatus() +
17              builder.BuildHeaders() +
18              builder.BuildContentBody();
19
20          // Відсилаємо відповідь клієнту
21          httpRequest.SendResponse(socket, htmlResponse);
22          Console.WriteLine($"HttpRequest state: ValidatePageState");
23
24          // Перехід у новий стан: після відправки відповіді клієнту
25          httpRequest.TransitionTo(new SendedState(), socket);
26      }
```

Рис. 5 - Приклад створення об’єкту класу BuilderPage і послідовного виклику методів формування відповіді

Висновок: у ході виконання даної лабораторної роботи я реалізувала шаблон Builder, який дозволяє поетапно створювати складний об'єкт за допомогою чітко визначеної послідовності дій (кроків). Керування “будівництвом” здійснюється об'єктом-розпорядником (director), якому потрібно знати тільки тип створюваного об'єкта. Я переконалась, що використання цього паттерну дає гнучкіший контроль над процесом створення відповіді на запит клієнта до Http-серверу, адже дозволяє відокремити побудований об'єкт від його представлення.

Посилання на репозиторій: AnnaKiikoIA24/TRPZ_labs_Kiiko_AO_IA24 (github.com)