Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №4

з дисципліни <<Технології розробки програмного забезпечення>>

Тема <<Шаблони «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»>>

Виконав:
студент ІА-23
Содолиснький Вадим

Перевірив:
Мягкий М. Ю.

Київ 2024

Тема: Шаблони «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY».

Хід роботи

Варіант №13

Office communicator

Мережевий комунікатор для офісу повинен нагадувати функціонал програми Skype з можливостями голосового / відео / конференц-зв'язку, відправки текстових повідомлень і файлів (можливо, оффлайн), веденням організованого списку груп / контактів.

Короткий опис патернів

Патерн Strategy

Патерн Strategy належить до поведінкових патернів проєктування. Він дозволяє визначити сімейство алгоритмів, інкапсулювати кожен з них і зробити їх взаємозамінними. Це забезпечує можливість змінювати поведінку об'єкта під час виконання програми без зміни його коду.

Патерн Singleton

SINGLETON є патерном проектування, який забезпечує створення лише одного екземпляра класу і надає глобальну точку доступу до нього.

Патерн Iterator

ITERATOR є патерном, що дозволяє по черзі переглядати елементи колекції без необхідності розкривати її внутрішню структуру. Це досягається завдяки створенню окремого об’єкта, який ітеративно обробляє елементи. Цей патерн корисний для роботи зі структурами даних, такими як списки, дерева або графи, де зручний доступ до елементів є критичним.

Патерн Proxy

PROXY є патерном, який створює об'єкт-посередник для управління доступом до іншого об'єкта. Він використовується для роботи з віддаленими об'єктами, наприклад, у випадку викликів віддалених процедур, або для лінивої ініціалізації, коли реальний об'єкт створюється тільки тоді, коли це дійсно необхідно.

Патерн State

STATE дозволяє об'єкту змінювати свою поведінку залежно від свого стану. Поведінка представлена як окремі класи, і об'єкт делегує їм виконання операцій.

Застосування патерну Strategy

Ключові ідеї:

Виділення поведінки в окремі класи (стратегії), які реалізують спільний інтерфейс. Контекст (основний клас) не змінює свою структуру, але може змінювати поведінку, вибираючи потрібну стратегію.

Переваги

Заміна алгоритмів на льоту. Легко змінювати поведінку без модифікації контексту.

Алгоритми ізольовані в окремих класах, що спрощує їх тестування та повторне використання.

Проблема

Оскільки способи спілкування в моєму проектуванні поділяються на контакти та групи, то при надсиланні повідомлення потрібно реалізовувати різні способи перевірки користувача та повідомлення.

Рішення

Для цього я використав патерн Strategy. В мене є інтерфейс IChecker від якого унаслідується два класи GroupChecker і ContactChecker. Цей інтерфейс реалізує метод перевірки користувача та повідомлення.

```
3 references
public interface IChecker
{
    5 references
    public Task<bool> CheckPermissionUser(int userId, int chatId);
}
```

Рис. 1 – Інтерфейс

```
public class GroupChecker : IChecker
{
    private readonly OfficeDbContext _dbContext;

    public GroupChecker(OfficeDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    public async Task<bool> CheckPermissionUser(int userId, int chatId)
    {
        Group? group = await _dbContext.Groups.Include(g => g.Users).FirstOrDefaultAsync(g => g.ChatId == chatId);
        if (group == null) return false;
        return group.Users.Any(u => u.Id == userId);
    }
}
```

Рис. 2 – Клас GroupChecker

GroupChecker, перевіряє доступ користувача через групи. У цьому класі використовується об'єкт OfficeDbContext для роботи з базою даних. Метод CheckPermissionUser() асинхронно перевіряє, чи є користувач членом групи, яка пов'язана з певним чатом. Для цього здійснюється запит до таблиці Groups, включаючи пов’язаних користувачів. Якщо групу з вказаним chatId не знайдено, повертається значення false. У протилежному випадку перевіряється, чи входить користувач у список членів групи. Якщо користувач знайдений, повертається true, інакше — false.

```
public class ContactChecker : IChecker
{
    private readonly OfficeDbContext _dbContext;

    public ContactChecker(OfficeDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    public async Task<bool> CheckPermissionUser(int userId, int chatId)
    {
        Contact? contact = await _dbContext.Contacts.FirstOrDefaultAsync(c => c.ChatId == chatId);
        if (contact == null || (contact.UserId != userId && contact.AssociatedUserId != userId)) return false;
        return true;
    }
}
```

Рис. 3 – Клас ContactChecker

ContactChecker, перевіряє доступ до чату через контакти. Як і у випадку з GroupChecker, тут використовується OfficeDbContext для роботи з базою даних. Метод CheckPermissionUser() асинхронно виконує пошук запису про контакт у таблиці Contacts за вказаним chatId. Якщо контакт не знайдено або userId не відповідає дозволеним користувачам (головний користувач або асоційованому

користувачу), доступ забороняється, і метод повертає false. Якщо ж перевірка пройшла успішно, повертається true.

```
2 references
public async Task<Message?> AddAsync(MessageDto messageDto, IChecker checker)
{
    if (await checker.CheckPermissionUser(messageDto.UserId, messageDto.ChatId)) return null;

    Message message = _mapper.Map<Message>(messageDto);
    await _dbContext.AddAsync(message);
    await _dbContext.SaveChangesAsync();
    return message;
}
```

```
Message? message;

if(messageDto.CommunicationType.Equals(typeof(Group))) message = await _messageRepository.AddAsync(messageDto, _groupChecker);
else if(messageDto.CommunicationType.Equals(typeof(Contact))) message = await _messageRepository.AddAsync(messageDto, _contactChecker);
else return BadRequest("Invalid communication type");
```

Рис. 4 – Використання патерну в методі AddAsync()

Метод AddAsync() використовується для додавання нового повідомлення до бази даних. Метод приймає об'єкт messageDto, що містить дані повідомлення, та об'єкт checker, який є реалізацією інтерфейсу IChecker. В залежності від того чи повідомлення надіслане в чат групи чи контакту викликається той чи інший метод перевірки.

Висновок: шаблони «SINGLETON», «ITERATOR», «PROXY», «STATE» і «STRATEGY» є інструментами для створення гнучкої та масштабованої архітектури програм. SINGLETON забезпечує унікальність об'єкта, ITERATOR спрощує доступ до елементів колекцій, PROXY контролює доступ до об'єктів, STATE змінює поведінку об'єкта залежно від стану, а STRATEGY дозволяє динамічно змінювати алгоритми. Я використав патерн Strategy для зміни логіки перевірки дотупу користувача до чати в залежності від групи та контакту.