

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

Содолиський Вадим

Перевірив:

Мягкий М. Ю.

Київ 2024

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

## Хід роботи

### Патерн Стратегія (Strategy)

Визначення Патерн Стратегія 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.

#### 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 проєктуванні поділяються на контакти та групи, то при надсиланні повідомлення потрібно реалізовувати різні способи перевірки користувача та повідомлення.

### Рішення

Для цього я використав патерн стратегія. В мене є інтерфейс IChecker від якого унаслідкується 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 – Интерфейс

```

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

```

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

```

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

**Висновок:** я використав патерн Strategy для зміни поведінки об'єкта без зміни його коду.