Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота № 4

Технології розроблення програмного забезпечення

«Вступ до паттернів проектування.»

Тема: IRC-Клієнт

Виконала:
студентка групи ІА-32
Красоха В.О.

Перевірив:
Мягкий М.Ю.

Київ 2025

Тема: Вступ до паттернів проектування.

Мета: Вивчити структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчитися застосовувати їх в реалізації програмної системи.

Хід роботи

Шаблон проєктування Singleton (Одинак)

Це породжувальний шаблон, який гарантує, що в системі існує лише один екземпляр певного класу, та надає глобальну точку доступу до нього. Основна ідея Singleton полягає в обмеженні створення об’єктів: конструктор класу роблять закритим або захищеним, а створення й отримання єдиного екземпляра відбувається через спеціальний статичний метод. Singleton часто використовується для керування спільними ресурсами, такими як підключення до бази даних, конфігураційні об’єкти, менеджери логування або кешу, де створення кількох екземплярів може призвести до помилок, перевитрати ресурсів або неконсистентності даних. Перевагами шаблону є контроль над кількістю екземплярів класу, зручний доступ до об’єкта з будь-якої частини програми та централізоване керування станом. Водночас Singleton має й недоліки: він створює приховану глобальну залежність, ускладнює модульне тестування, може порушувати принцип єдиної відповідальності та за надмірного використання призводить до сильної зв’язаності компонентів. У багатопотокових застосунках важливо забезпечити потокобезпечну реалізацію Singleton, інакше можливе створення кількох екземплярів одночасно. Загалом Singleton доцільно застосовувати лише тоді, коли наявність єдиного екземпляра є обґрунтованою вимогою архітектури системи.

Шаблон проєктування Iterator (Ітератор)

Це поведінковий шаблон, який надає спосіб послідовного доступу до елементів колекції без розкриття її внутрішньої структури. Основна ідея шаблону полягає у винесенні логіки обходу колекції в окремий об’єкт-ітератор, що дозволяє працювати з різними типами колекцій уніфікованим способом. Iterator визначає стандартний інтерфейс для доступу до елементів, зазвичай через операції на

кшталт переходу до наступного елемента та перевірки наявності наступного елемента. Колекція при цьому не зобов’язана знати, яким саме чином відбувається обхід її елементів. Шаблон Iterator зменшує зв’язаність між клієнтським кодом і структурами даних, спрощує підтримку та розширення програми, а також дозволяє реалізовувати декілька способів обходу однієї й тієї ж колекції. Він широко використовується в мовах програмування та стандартних бібліотеках, наприклад, для обходу списків, масивів, дерев або інших складних структур даних, і є типовим прикладом застосування принципу єдиної відповідальності, оскільки відокремлює зберігання даних від логіки їх перегляду.

Шаблон проєктування Proxy (Замісник)

Це структурний шаблон, який передбачає використання спеціального об’єкта-посередника для контролю доступу до іншого об’єкта. Proxy надає такий самий інтерфейс, як і реальний об’єкт, але перехоплює всі звернення до нього, дозволяючи виконувати додаткові дії до або після виклику основної логіки. Основна ідея шаблону полягає в тому, щоб відокремити клієнта від безпосередньої роботи з «важким», віддаленим або чутливим об’єктом і тим самим підвищити гнучкість, безпеку та керованість системи. Proxy часто використовується для лінивого створення об’єктів, коли реальний об’єкт ініціалізується лише в момент першого звернення до нього, для контролю прав доступу, для кешування результатів викликів, а також для роботи з віддаленими об’єктами в розподілених системах. У структурі шаблону зазвичай виділяють інтерфейс або абстрактний клас, який визначає спільний набір операцій, реальний об’єкт, що реалізує основну бізнес-логіку, та сам Proxy, який зберігає посилання на реальний об’єкт і керує доступом до нього. Перевагою шаблону Proxy є можливість додавати нову поведінку без зміни коду реального об’єкта, а недоліком — збільшення кількості класів і ускладнення структури системи.

Шаблон проєктування State (Стан)

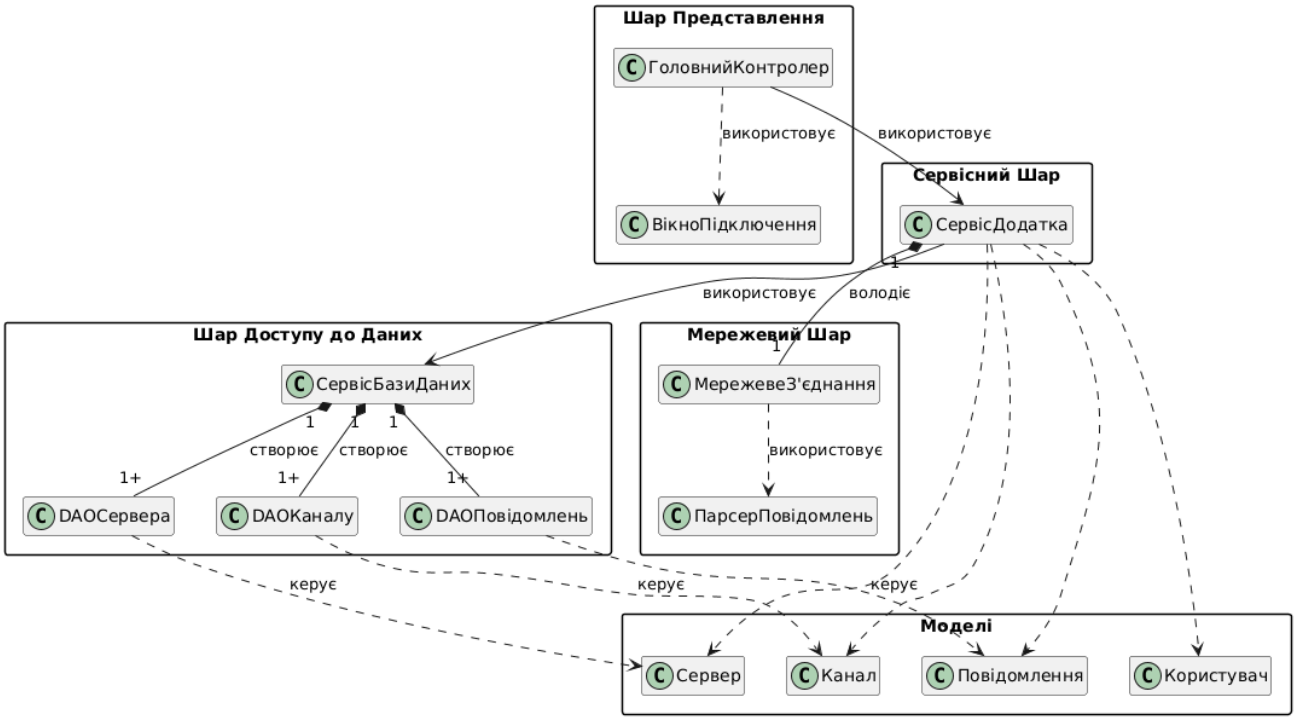
Цей шаблон належить до поведінкових шаблонів і використовується для організації поведінки об’єкта залежно від його внутрішнього стану, так що при зміні стану змінюється і поведінка об’єкта без використання великої кількості умовних операторів. Основна ідея шаблону полягає в тому, що кожен можливий стан об’єкта інкапсулюється в окремий клас, а сам об’єкт-контекст зберігає посилання на поточний стан і делегує йому виконання відповідних операцій. У структурі шаблону виділяють інтерфейс або абстрактний клас State, який оголошує методи, що відповідають поведінці об’єкта в різних станах, конкретні класи ConcreteState, які реалізують ці методи по-різному залежно від стану, та клас Context, який містить посилання на поточний стан і надає клієнтам єдиний інтерфейс для роботи з об’єктом. При зміні стану Context просто замінює об’єкт стану, не змінюючи власний код, що забезпечує дотримання принципу відкритості/закритості. Шаблон State доцільно застосовувати у випадках, коли поведінка об’єкта істотно залежить від його стану і змінюється під час виконання програми, а також коли умовні конструкції типу if або switch ускладнюють код і погіршують його підтримуваність. Перевагами шаблону є зменшення складності умовної логіки, чітке розділення відповідальностей між станами та полегшення розширення системи новими станами, тоді як серед недоліків зазвичай відзначають збільшення кількості класів і складнішу структуру проєкту.

Шаблон Strategy (Стратегія)

Це поведінковий шаблон проєктування, який визначає сімейство алгоритмів, інкапсулює кожен із них у окремому класі та робить їх взаємозамінними, завдяки чому алгоритм можна змінювати незалежно від клієнта, який його використовує. Основна ідея Strategy полягає у винесенні змінної поведінки з класу-контексту в окремі об’єкти-стратегії, що реалізують спільний інтерфейс, при цьому контекст зберігає посилання на об’єкт стратегії і делегує йому виконання відповідного алгоритму. Це дозволяє уникнути великої кількості умовних операторів (if/else або switch), спрощує розширення системи новими

алгоритмами та відповідає принципу відкритості/закритості (Open/Closed Principle), оскільки нові стратегії можна додавати без зміни існуючого коду контексту. Типова структура шаблону включає інтерфейс Strategy, конкретні реалізації стратегій (ConcreteStrategy), які містять різні алгоритми, та клас Context, що використовує обрану стратегію для виконання операцій. Перевагами шаблону є підвищення гнучкості та повторного використання коду, спрощення тестування алгоритмів і можливість динамічної зміни поведінки об’єкта під час виконання програми, однак серед недоліків можна виділити збільшення кількості класів у системі та необхідність, щоб клієнт або контекст коректно обирав потрібну стратегію. Шаблон Strategy широко застосовується, наприклад, для вибору способу сортування, алгоритму оплати, способу стискання даних або маршрутизації, коли одна й та сама задача може мати кілька альтернативних реалізацій.

Діаграма класів:



Обраний шаблон: Singleton (Одинак). Мета: запобігти створенню декількох з'єднань з одним і тим самим файлом бази даних, що могло б призвести до помилок запису або блокування файлу (database is locked).

Реалізація у проєкті

У IRC-клієнті цей патерн реалізовано в класі DatabaseService.

Структура класів:

1. DatabaseService: Клас-Singleton, що тримає єдине підключення Connection.
2. MessageDao: Клас, який отримує з'єднання від Singleton для виконання запитів.
3. Main (або IRCService): Клієнтський клас, який звертається до Singleton при запуску програми.

Опис взаємодії:

1. При запуску програми викликається DatabaseService.getInstance().
2. Система перевіряє, чи було вже створено об'єкт. Якщо ні — створює з'єднання з файлом ircclient.db.
3. Усі інші класи (наприклад, MessageDao) використовують той самий об'єкт з'єднання. Це гарантує стабільну роботу з SQLite.

Доданий код:

ChannelDao.java:

```
package org.ircclient.dao;

import org.ircclient.model.Channel;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class ChannelDao {
    private final Connection connection;

    public ChannelDao(Connection connection) {
        this.connection = connection;
    }

    public void insert(Channel channel) throws SQLException {
        String sql = "INSERT INTO channels (server_id, name, joined_at) VALUES (?, ?, ?)";
        try (PreparedStatement stmt = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
            stmt.setInt(1, channel.getServerId());
            stmt.setString(2, channel.getName());
            stmt.setTimestamp(3, Timestamp.valueOf(channel.getJoinedAt()));
            stmt.executeUpdate();

            try (ResultSet rs = stmt.getGeneratedKeys()) {
                if (rs.next()) {
                    channel.setId(rs.getInt(1));
                }
            }
        }
    }

    public List<Channel> findByServerId(Integer serverId) throws SQLException {
        List<Channel> channels = new ArrayList<>();
        String sql = "SELECT id, server_id, name, joined_at FROM channels WHERE server_id = ?";
        try (PreparedStatement stmt = connection.prepareStatement(sql)) {
            stmt.setInt(1, serverId);
            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    Channel channel = new Channel();
                    channel.setId(rs.getInt("id"));
                    channel.setServerId(rs.getInt("server_id"));
                    channel.setName(rs.getString("name"));
                    Timestamp timestamp = rs.getTimestamp("joined_at");
                    if (timestamp != null) {
                        channel.setJoinedAt(timestamp.toLocalDateTime());
                    }
                    channels.add(channel);
                }
            }
        }
        return channels;
    }

    public Channel findByServerIdAndName(Integer serverId, String name) throws SQLException {
        String sql = "SELECT id, server_id, name, joined_at FROM channels WHERE server_id = ? AND name = ?";
        try (PreparedStatement stmt = connection.prepareStatement(sql)) {
            stmt.setInt(1, serverId);
            stmt.setString(2, name);
```

```
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    Channel channel = new Channel();
                    channel.setId(rs.getInt("id"));
                    channel.setServerId(rs.getInt("server_id"));
                    channel.setName(rs.getString("name"));
                    Timestamp timestamp = rs.getTimestamp("joined_at");
                    if (timestamp != null) {
                        channel.setJoinedAt(timestamp.toLocalDateTime());
                    }
                    return channel;
                }
            }
        }
        return null;
    }

    public void delete(Integer id) throws SQLException {
        String sql = "DELETE FROM channels WHERE id = ?";
        try (PreparedStatement stmt = connection.prepareStatement(sql)) {
            stmt.setInt(1, id);
            stmt.executeUpdate();
        }
    }

    public void deleteByServerId(Integer serverId) throws SQLException {
        String sql = "DELETE FROM channels WHERE server_id = ?";
        try (PreparedStatement stmt = connection.prepareStatement(sql)) {
            stmt.setInt(1, serverId);
            stmt.executeUpdate();
        }
    }
}
```

MessageDao.java:

```
package org.ircclient.dao;
import org.ircclient.model.Message;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class MessageDao {
    private final Connection connection;

    public MessageDao(Connection connection) {
        this.connection = connection;
    }

    public void insert(Message message) throws SQLException {
        String sql = "INSERT INTO messages (channel_id, sender, content, timestamp, is_private) VALUES
(?, ?, ?, ?, ?)";
        try (PreparedStatement stmt = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
            stmt.setObject(1, message.getChannelId(), Types.INTEGER);
            stmt.setString(2, message.getSender());
            stmt.setString(3, message.getContent());
            stmt.setTimestamp(4, Timestamp.valueOf(message.getTimestamp()));
            stmt.setBoolean(5, message.getIsPrivate());
            stmt.executeUpdate();

            try (ResultSet rs = stmt.getGeneratedKeys()) {
```

```
            if (rs.next()) {
                message.setId(rs.getInt(1));
            }
        }
    }
}

public List<Message> findByChannelId(Integer channelId) throws SQLException {
    List<Message> messages = new ArrayList<>();
    String sql = "SELECT id, channel_id, sender, content, timestamp, is_private FROM messages WHERE channel_id
= ? ORDER BY timestamp";
    try (PreparedStatement stmt = connection.prepareStatement(sql)) {
        stmt.setInt(1, channelId);
        try (ResultSet rs = stmt.executeQuery()) {
            while (rs.next()) {
                Message message = new Message();
                message.setId(rs.getInt("id"));
                message.setChannelId(rs.getInt("channel_id"));
                message.setSender(rs.getString("sender"));
                message.setContent(rs.getString("content"));
                Timestamp timestamp = rs.getTimestamp("timestamp");
                if (timestamp != null) {
                    message.setTimestamp(timestamp.toLocalDateTime());
                }
                message.setIsPrivate(rs.getBoolean("is_private"));
                messages.add(message);
            }
        }
    }
    return messages;
}

public void deleteByChannelId(Integer channelId) throws SQLException {
    String sql = "DELETE FROM messages WHERE channel_id = ?";
    try (PreparedStatement stmt = connection.prepareStatement(sql)) {
        stmt.setInt(1, channelId);
        stmt.executeUpdate();
    }
}
}
```

ServerDao.java:

```
package org.ircclient.dao;

import org.ircclient.model.Server;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class ServerDao {
    private final Connection connection;

    public ServerDao(Connection connection) {
        this.connection = connection;
    }

    public void insert(Server server) throws SQLException {
        String sql = "INSERT INTO servers (name, host, port, nickname) VALUES (?, ?, ?, ?)";
        try (PreparedStatement stmt = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {
```

```
            stmt.setString(1, server.getName());
            stmt.setString(2, server.getHost());
            stmt.setInt(3, server.getPort());
            stmt.setString(4, server.getNickname());
            stmt.executeUpdate();

            try (ResultSet rs = stmt.getGeneratedKeys()) {
                if (rs.next()) {
                    server.setId(rs.getInt(1));
                }
            }
        }
    }

    public List<Server> findAll() throws SQLException {
        List<Server> servers = new ArrayList<>();
        String sql = "SELECT id, name, host, port, nickname FROM servers";
        try (Statement stmt = connection.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {
            while (rs.next()) {
                Server server = new Server();
                server.setId(rs.getInt("id"));
                server.setName(rs.getString("name"));
                server.setHost(rs.getString("host"));
                server.setPort(rs.getInt("port"));
                server.setNickname(rs.getString("nickname"));
                servers.add(server);
            }
        }
        return servers;
    }

    public Server findById(Integer id) throws SQLException {
        String sql = "SELECT id, name, host, port, nickname FROM servers WHERE id = ?";
        try (PreparedStatement stmt = connection.prepareStatement(sql)) {
            stmt.setInt(1, id);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    Server server = new Server();
                    server.setId(rs.getInt("id"));
                    server.setName(rs.getString("name"));
                    server.setHost(rs.getString("host"));
                    server.setPort(rs.getInt("port"));
                    server.setNickname(rs.getString("nickname"));
                    return server;
                }
            }
        }
        return null;
    }

    public void delete(Integer id) throws SQLException {
        String sql = "DELETE FROM servers WHERE id = ?";
        try (PreparedStatement stmt = connection.prepareStatement(sql)) {
            stmt.setInt(1, id);
            stmt.executeUpdate();
        }
    }
}
```

Контрольні питання:

1. Що таке шаблон проєктування?

Шаблон проєктування — це типове, перевірене часом рішення повторюваної проблеми проєктування в програмуванні, яке описує структуру класів та об’єктів і спосіб їх взаємодії.

2. Навіщо використовувати шаблони проєктування?

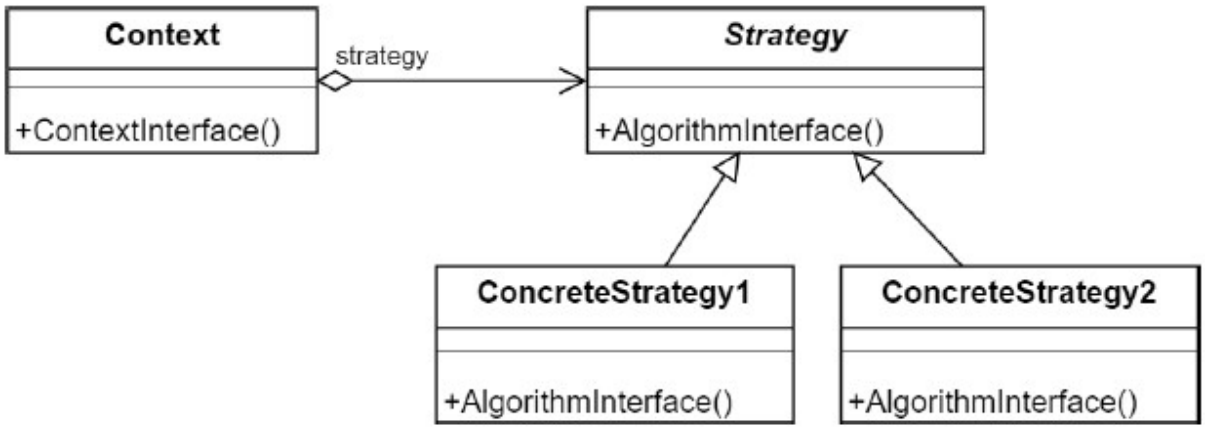
Шаблони проєктування використовують для:

- підвищення гнучкості та розширюваності коду;
- зменшення складності та кількості помилок;
- повторного використання перевірених архітектурних рішень;
- покращення читабельності та підтримуваності програм.

3. Яке призначення шаблону «Стратегія»?

Шаблон Strategy (Стратегія) призначений для інкапсуляції набору алгоритмів і забезпечення можливості їх взаємозамінності, що дозволяє змінювати алгоритм без зміни класу, який його використовує.

4. Нарисуйте структуру шаблону «Стратегія».



5. Які класи входять в шаблон «Стратегія», та яка між ними взаємодія?

У шаблон «Стратегія» входять:

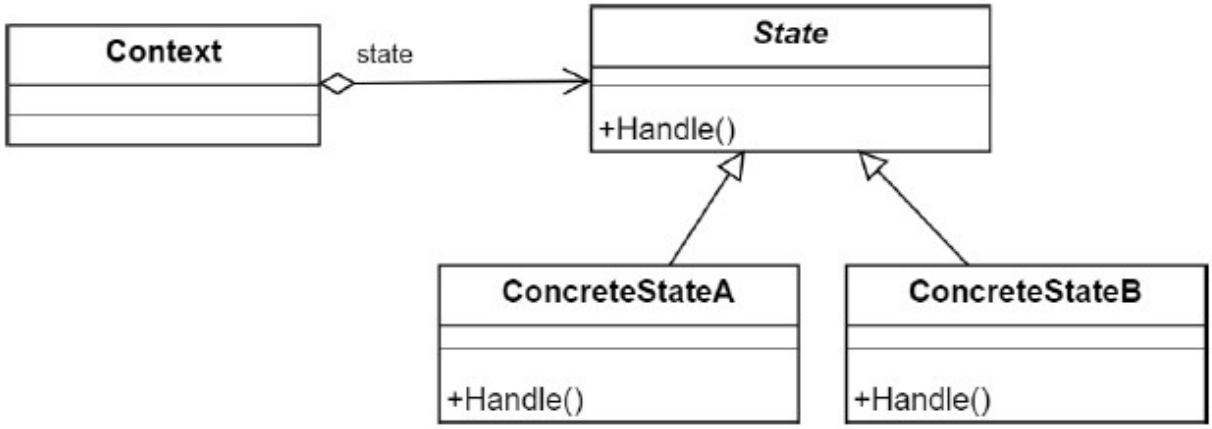
- інтерфейс Strategy;
- конкретні реалізації ConcreteStrategy;

- клас Context, який використовує об’єкт стратегії. Взаємодія полягає в тому, що Context делегує виконання алгоритму обраній стратегії.

6. Яке призначення шаблону «Стан»?

Шаблон State (Стан) призначений для зміни поведінки об’єкта залежно від його внутрішнього стану, при цьому зміна стану виглядає як зміна класу об’єкта.

7. Нарисуйте структуру шаблону «Стан».



8. Які класи входять в шаблон «Стан», та яка між ними взаємодія?

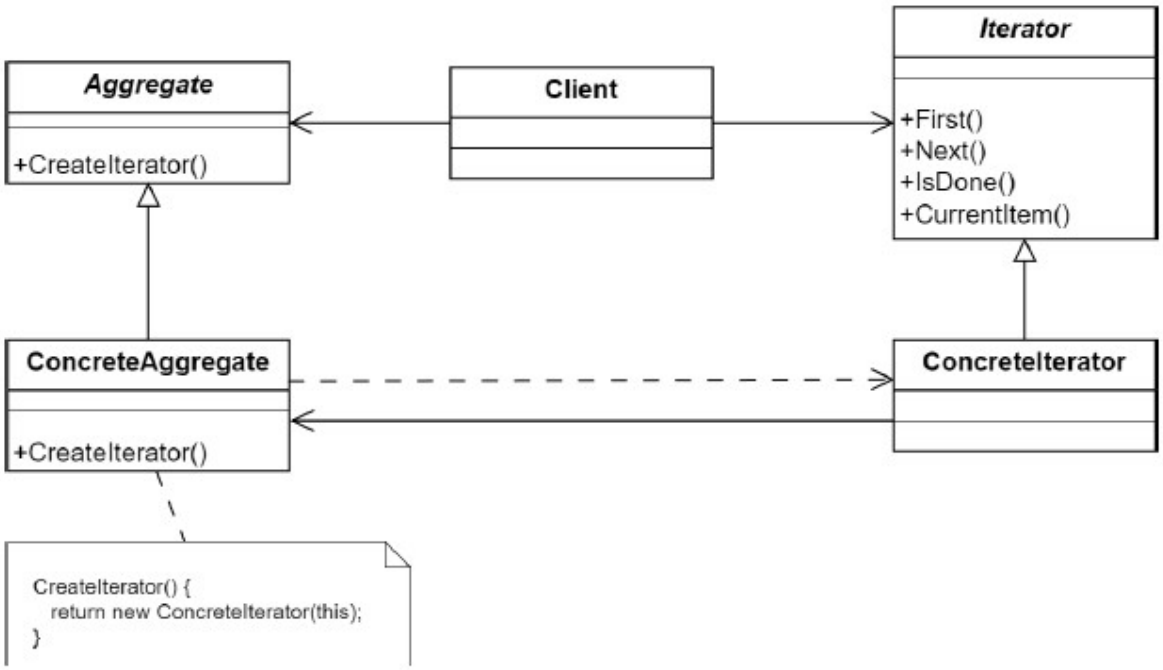
До шаблону «Стан» входять:

- інтерфейс State;
- конкретні стани ConcreteState;
- клас Context, що зберігає поточний стан. Context делегує поведінку поточному стану, а об’єкти станів можуть змінювати стан контексту.

9. Яке призначення шаблону «Ітератор»?

Шаблон Iterator (Ітератор) призначений для послідовного доступу до елементів колекції без розкриття її внутрішньої структури.

10. Нарисуйте структуру шаблону «Ітератор».



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

- інтерфейс Iterator;
- конкретний ітератор ConcreteIterator;
- інтерфейс колекції Aggregate;
- конкретна колекція ConcreteAggregate. ConcreteAggregate створює ітератор, а ConcreteIterator керує обходом елементів колекції.

12. В чому полягає ідея шаблона «Одинак»?
Шаблон Singleton (Одинак) гарантує, що клас має лише один екземпляр, і надає глобальну точку доступу до нього.

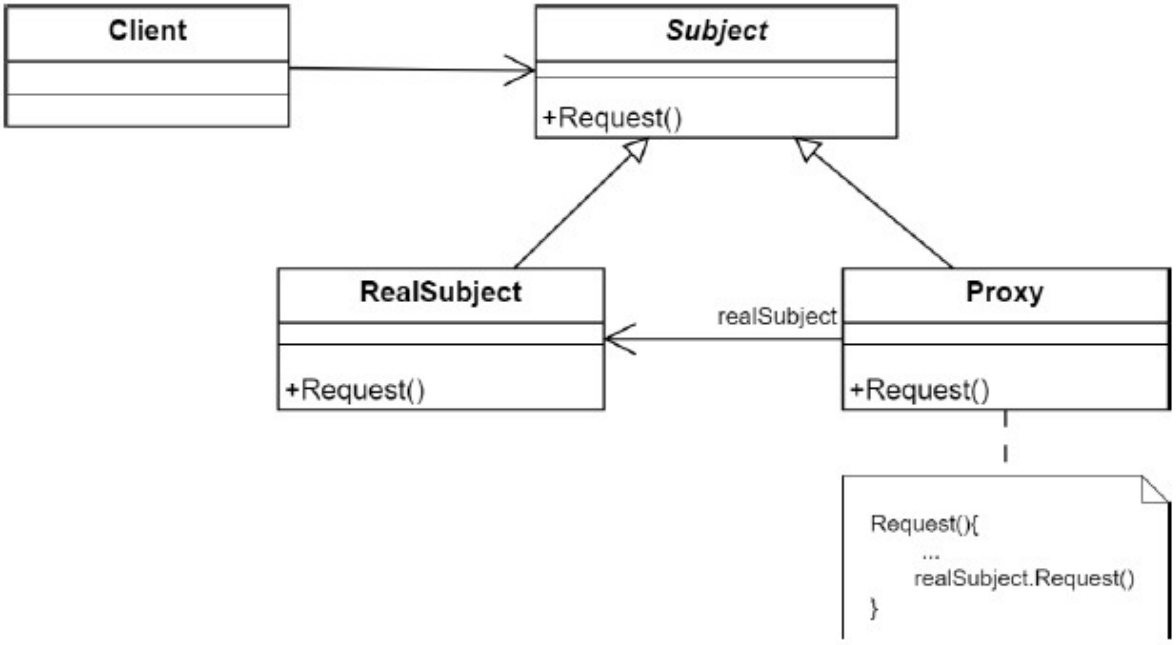
13. Чому шаблон «Одинак» вважають «анти-шаблоном»?
Singleton вважають анти-шаблоном, оскільки він:

- приховує глобальний стан;

- ускладнює тестування;
- порушує принцип єдиної відповідальності;
- створює сильну зв’язаність між компонентами.

14. Яке призначення шаблону «Проксі»?
Шаблон Proxy (Проксі) призначений для контролю доступу до іншого об’єкта, додаючи додаткову логіку без зміни самого об’єкта.

15. Нарисуйте структуру шаблону «Проксі».



16. Які класи входять в шаблон «Проксі», та яка між ними взаємодія?
До шаблону «Проксі» входять:

- інтерфейс Subject;
- реальний об’єкт RealSubject;
- клас Proxy.
  Proxy реалізує той самий інтерфейс, що й RealSubject, і керує доступом до нього.

Висновок: Під час лабораторної роботи я вивчила структуру шаблонів «Singleton», «Iterator», «Proxy», «State», «Strategy» та навчилася застосовувати їх в реалізації програмної системи.