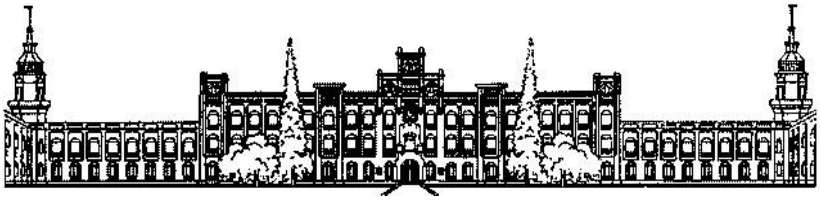


Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформатики та програмної інженерії

Лабораторна робота №4

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Вступ до паттернів проектування»

Виконав:

студент групи ІА-32

Нагорний Максим

Перевірив:

Мягкий М. Ю.

Київ 2025

Зміст

Вступ

Тема: Вступ до паттернів проектування.

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

Завдання

- Ознайомитись з короткими теоретичними відомостями.

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

- Реалізувати один з розглянутих шаблонів за обраною темою.

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

- 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.

14. **Архіватор** (strategy, adapter, factory method, facade, visitor, p2p)

Архіватор повинен являти собою візуальний додаток з можливістю створення і редагування архівів різного типу (.tar.gz, .zip, .rar, .ace) – додавання/ видалення файлів / папок, редагування метаданих (по можливості), перевірка checksum архівів, тестування архівів на наявність пошкоджень, розбиття архівів на частини.

Теоретичні відомості

Шаблон «Singleton»

Singleton - це шаблон, який гарантує, що клас матиме лише один екземпляр, і надає до нього глобальну точку доступу. Його ключова мета — контролювати створення об'єктів, що є корисним для ресурсів, які мають бути унікальними в системі, як-от конфігураційний менеджер або з'єднання з базою даних. Структурно це реалізується через приватний конструктор, щоб унеможливити створення екземплярів ззовні, приватне статичне поле для зберігання єдиного екземпляра та публічний статичний метод, який повертає цей екземпляр, створюючи його лише за першого виклику.

Шаблон «Iterator»

Iterator - це поведінковий шаблон, що надає стандартизований спосіб послідовного доступу до елементів колекції, не розкриваючи її внутрішньої реалізації. Це дозволяє уніфікувати обхід різних структур даних, чи то масивів, чи то списків, чи дерев. Клієнтський код працює з інтерфейсом ітератора, що зазвичай має методи, як-от hasNext() для перевірки наявності наступного елемента та next() для його отримання. Таким чином, логіка обходу інкапсулюється в окремому об'єкті-ітераторі, а не захаращує клієнтський код.

Шаблон «Proxy»

Proxy - це шаблон, який надає об'єкт-замісник для контролю доступу до іншого, реального об'єкта. Проксі має той самий інтерфейс, що й реальний об'єкт, що дозволяє клієнту не помічати різниці. Його використовують для різних цілей: лінивої ініціалізації (створення "важкого" об'єкта лише за потреби), контролю доступу (перевірки прав), логування викликів або кешування результатів операцій. Проксі виступає посередником, додаючи додаткову логіку до або після звернення до реального об'єкта.

Шаблон «State»

State - це шаблон, який дозволяє об'єкту змінювати свою поведінку залежно від внутрішнього стану. Зовні це виглядає так, ніби об'єкт змінив свій клас. Замість громіздких умовних конструкцій (if/switch) логіка кожного стану інкапсулюється в окремому класі. Контекстний об'єкт зберігає посилання на поточний об'єкт-стан і делегує йому виконання роботи. Кожен клас-стан реалізує спільний інтерфейс і може самостійно керувати переходом до іншого стану, змінюючи об'єкт стану в контексті. Це робить систему гнучкою та легкою для розширення новими станами.

Шаблон «Strategy»

Strategy - це шаблон, який визначає сімейство алгоритмів, інкапсулює кожен з них і робить їх взаємозамінними. Це дозволяє клієнту (контексту) вибирати потрібний алгоритм під час виконання програми. Структура складається з контексту, який працює зі стратегією через загальний інтерфейс, та набору конкретних стратегій, кожна з яких реалізує один з алгоритмів. Наприклад, у навігаторі можна динамічно перемикатися між стратегіями побудови маршруту для автомобіля, пішохода чи громадського транспорту, не змінюючи основний код навігатора. Це сприяє дотриманню принципу відкритості/закритості, оскільки нові алгоритми можна додавати, не модифікуючи існуючий код.

Хід виконання

Для розробки системи архіватора було обрано шаблон проектування «Стратегія». Вибір цього шаблону обумовлений необхідністю реалізувати архівування різних типів файлів із можливістю гнучкої зміни алгоритмів без модифікації основного коду системи.

Суть шаблону «Стратегія» полягає в тому, що поведінка об’єкта може змінюватися під час виконання програми шляхом підстановки різних алгоритмів. У нашому випадку, кожен алгоритм відповідає за певний спосіб архівування (ZIP, TAR, TAR.GZ тощо). Завдяки цьому підходу ми можемо досягати однієї мети — архівування файлів — різними методами, не змінюючи структуру основного класу архіватора.

Для реалізації цього шаблону ми виконали такі дії:

1. Створено загальний інтерфейс StrategyArchive, цей інтерфейс визначає метод для архівування.
2. Створено клас ZipStrategyArchive, що імплементується від StrategyArchive та перевизначає метод compress та реалізовує логіку архівування типом Zip.
3. Створено клас TarGzStrategyArchive, що імплементується від StrategyArchive та перевизначає метод compress та реалізовує логіку архівування типом Tar.
4. Створено клас RarStrategyArchive, що імплементується від StrategyArchive та перевизначає метод compress та реалізовує логіку архівування типом Rar з допомогою програми WinRar.

Переваги обраного шаблону:

- Можливість роботи з кожним типом архівування окремо не впливаючи, на інші типи архівування;

- Зменшення залежностей між класами, підтримка IoC, тепер модулі верхнього рівня не залежать від модулів нижнього рівня;

На рис. 1 наведена модифікована діаграма класів із представленням використання шаблону в реалізації системи

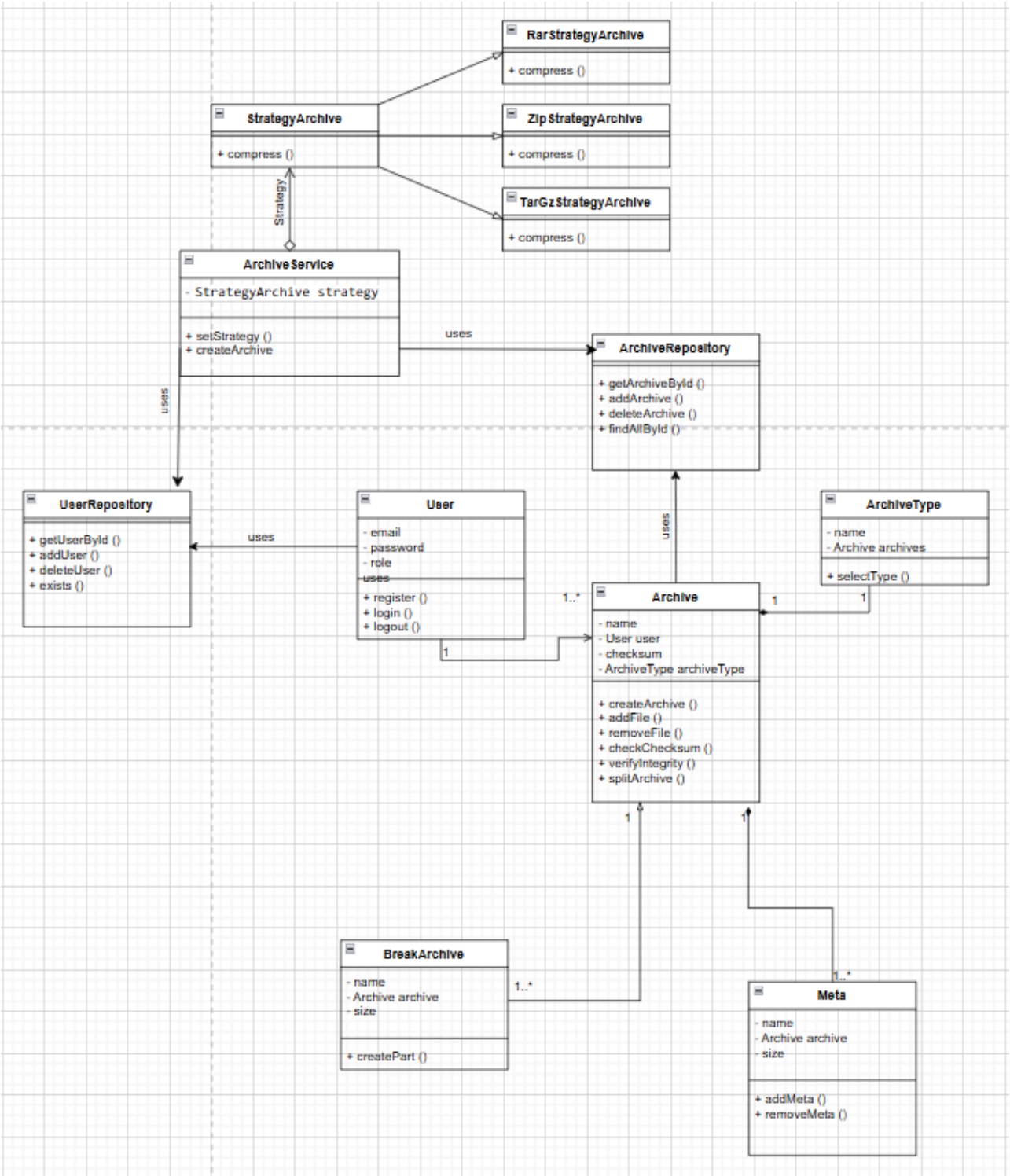


Рис. 1. Діаграма класів

Код програми

```
package com.example.demo.strategy;
import java.io.File;
import java.io.IOException;
import java.util.List;
public interface StrategyArchive {
    void compress(List<File> files, String outputPath) throws IOException;
}
package com.example.demo.strategy;
import java.io.*;
import java.nio.charset.StandardCharsets;
import java.util.*;
public class ZipStrategyArchive implements StrategyArchive {
    private static final int LOCAL_FILE_HEADER_SIGNATURE = 0x04034b50;
    private static final int CENTRAL_DIR_SIGNATURE = 0x02014b50;
    private static final int END_OF_CENTRAL_DIR_SIGNATURE = 0x06054b50;
    @Override
    public void compress(List<File> files, String outputPath) throws IOException {
        try (DataOutputStream dos = new DataOutputStream(new FileOutputStream(outputPath))) {
            List<CentralDirEntry> centralDir = new ArrayList<>();
            for (File file : files) {
                byte[] fileData = readFile(file);
                byte[] nameBytes = file.getName().getBytes(StandardCharsets.UTF_8);
                long localHeaderOffset = dos.size();
                dos.writeInt(Integer.reverseBytes(LOCAL_FILE_HEADER_SIGNATURE));
                dos.writeShort(0);
                dos.writeShort(0);
                dos.writeShort(0);
                dos.writeShort(0);
                dos.writeShort(0);
                dos.writeInt(0);
                dos.writeInt(Integer.reverseBytes(fileData.length));
                dos.writeInt(Integer.reverseBytes(fileData.length));
                dos.writeShort(Short.reverseBytes((short) nameBytes.length));
```

```
            dos.writeShort(0);
            dos.write(nameBytes);
            dos.write(fileData);
            centralDir.add(new CentralDirEntry(file.getName(), (int) fileData.length, (int)
localHeaderOffset));
        }
        long centralDirOffset = dos.size();
        for (CentralDirEntry entry : centralDir) {
            byte[] nameBytes = entry.name.getBytes(StandardCharsets.UTF_8);
            dos.writeInt(Integer.reverseBytes(CENTRAL_DIR_SIGNATURE));
            dos.writeShort(0);
            dos.writeShort(0);
            dos.writeShort(0);
            dos.writeShort(0);
            dos.writeShort(0);
            dos.writeShort(0);
            dos.writeInt(0);
            dos.writeInt(Integer.reverseBytes(entry.size));
            dos.writeInt(Integer.reverseBytes(entry.size));
            dos.writeShort(Short.reverseBytes((short) nameBytes.length));
            dos.writeShort(0);
            dos.writeShort(0);
            dos.writeShort(0);
            dos.writeShort(0);
            dos.writeInt(0);
            dos.writeInt(Integer.reverseBytes(entry.offset));
            dos.write(nameBytes);
        }
        long centralDirSize = dos.size() - centralDirOffset;
        dos.writeInt(Integer.reverseBytes(END_OF_CENTRAL_DIR_SIGNATURE));
        dos.writeShort(0);
        dos.writeShort(0);
        dos.writeShort(Short.reverseBytes((short) centralDir.size()));
```

```
            dos.writeShort(Short.reverseBytes((short) centralDir.size()));
            dos.writeInt(Integer.reverseBytes((int) centralDirSize));
            dos.writeInt(Integer.reverseBytes((int) centralDirOffset));
            dos.writeShort(0);
        }
    }
    private byte[] readFile(File file) throws IOException {
        try (FileInputStream fis = new FileInputStream(file)) {
            return fis.readAllBytes();
        }
    }
    private static class CentralDirEntry {
        String name;
        int size;
        int offset;
        CentralDirEntry(String name, int size, int offset) {
            this.name = name;
            this.size = size;
            this.offset = offset;
        }
    }
}
package com.example.demo.strategy;
import java.io.*;
import java.util.List;
public class TarGzStrategyArchive implements StrategyArchive {
    private static final int BLOCK_SIZE = 512;
    @Override
    public void compress(List<File> files, String outputPath) throws IOException {
        try (FileOutputStream fos = new FileOutputStream(outputPath);
             BufferedOutputStream bos = new BufferedOutputStream(fos)) {

            for (File file : files) {
```

```
            addFileToTar(bos, file);
        }
        bos.write(new byte[BLOCK_SIZE]);
        bos.write(new byte[BLOCK_SIZE]);
    }
}

private void addFileToTar(OutputStream out, File file) throws IOException {
    byte[] header = createHeader(file);
    out.write(header);
    try (FileInputStream fis = new FileInputStream(file)) {
        byte[] buffer = new byte[8192];
        int count;
        while ((count = fis.read(buffer)) != -1) {
            out.write(buffer, 0, count);
        }
    }
    long fileSize = file.length();
    int padding = (int) (BLOCK_SIZE - (fileSize % BLOCK_SIZE));
    if (padding < BLOCK_SIZE) {
        out.write(new byte[padding]);
    }
}
private byte[] createHeader(File file) {
    byte[] header = new byte[BLOCK_SIZE];
    writeString(file.getName(), header, 0, 100);
    writeOctal(0755L, header, 100, 8);
    writeOctal(0L, header, 108, 8);
    writeOctal(0L, header, 116, 8);
    writeOctal(file.length(), header, 124, 12);
    writeOctal(file.lastModified() / 1000, header, 136, 12);
    header[156] = '0';
    writeString("        ", header, 148, 8);
```

```
        long checksum = 0;
        for (byte b : header) {
            checksum += b & 0xFF;
        }
        writeOctal(checksum, header, 148, 8);

        return header;
    }
    private void writeString(String str, byte[] buffer, int offset, int length) {
        byte[] bytes = str.getBytes();
        System.arraycopy(bytes, 0, buffer, offset, Math.min(bytes.length, length));
    }
    private void writeOctal(long value, byte[] buffer, int offset, int length) {
        String octal = String.format("%0" + (length - 1) + "o", value);
        writeString(octal + "\0", buffer, offset, length);
    }
}
package com.example.demo.strategy;
import java.io.File;
import java.io.IOException;
import java.util.List;
public class RarStrategyArchive implements StrategyArchive {
    private final String rarPath = "C:\\Program Files\\WinRAR\\WinRAR.exe";
    @Override
    public void compress(List<File> files, String outputPath) throws IOException {
        StringBuilder command = new StringBuilder("\"" + rarPath + "\" a -ep \"" + outputPath + "\"");
        for (File file : files) {
            command.append(" \"").append(file.getAbsolutePath()).append("\"");
        }
        Process process = Runtime.getRuntime().exec(command.toString());
        try {
            int exitCode = process.waitFor();
            if (exitCode != 0) {
```

```
            throw new IOException("RAR архівація завершилась з помилкою: " + exitCode);
          }
        } catch (InterruptedException e) {
          throw new IOException("RAR архівація перервана", e);
        }
    }
}
package com.example.demo.service;

import org.springframework.stereotype.Service;
import com.example.demo.strategy.StrategyArchive;
import java.io.File;
import java.io.IOException;
import java.util.List;
@Service
public class ArchiveService {
    private StrategyArchive strategy;
    public void setStrategy(StrategyArchive strategy) {
        this.strategy = strategy;
    }
    public void createArchive(List<File> files, String outputPath) throws IOException {
        strategy.compress(files, outputPath);
    }
}
```

Висновок

Під час виконання лабораторної роботи було досліджено поняття шаблону проектування. Почалася робота з вибору певного шаблону проектування. Після цього було реалізовано шаблон «стратегія» згідно теми «архіватор», під час реалізації було розроблено загальний інтерфейс ArchiveStrategy, який визначає метод compress для архівування та розроблено

класи – наслідники ZipArchiveStrategy, TarGzArchiveStrategy, RarArchiveStrategy для реалізації логіки архівування різних типів та останнім етапом було модифікації діаграми класів з представленням використання шаблону «стратегія».

Питання до лабораторної роботи

1. Що таке шаблон проєктування?

Шаблон проєктування — це перевірене, універсальне та відтворюване розв'язання типової проблеми, що часто виникає під час проєктування програмного забезпечення. Це не готовий код, а скоріше концепція або ідея, яку можна адаптувати для вирішення конкретного завдання у вашому проєкті.

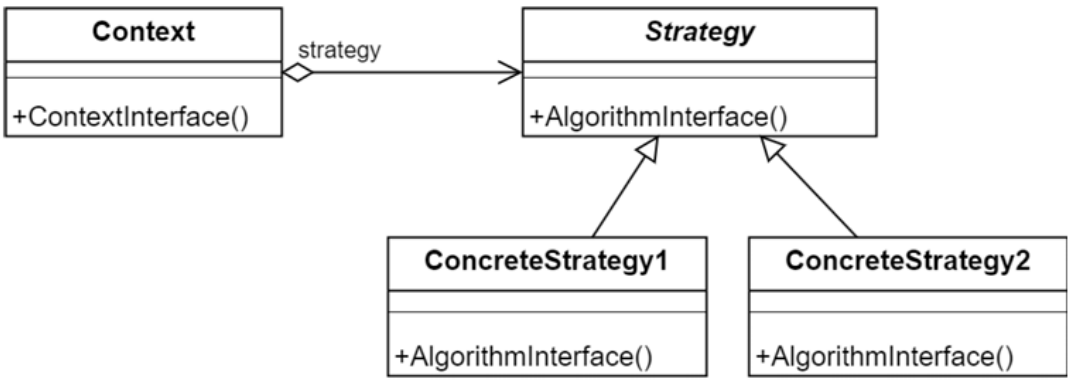
2. Навіщо використовувати шаблони проєктування?

- Вони пропонують надійні способи розв'язання проблем, які вже були випробувані та довели свою ефективність.
- Допомагають створювати гнучкий, розширюваний та легко підтримуваний код, що відповідає принципам SOLID.

3. Яке призначення шаблону «Стратегія»?

Призначення шаблону «Стратегія»— інкапсулювати сімейство взаємозамінних алгоритмів і дозволити клієнту вибирати потрібний алгоритм під час виконання програми. Цей шаблон дозволяє змінювати поведінку об'єкта без зміни його класу.

4. Нарисуйте структуру шаблону «Стратегія».



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

- Context – це клас, який використовує один із алгоритмів. Він містить посилання на об'єкт-стратегію і делегує йому виконання певної роботи, не вникаючи в деталі її реалізації.

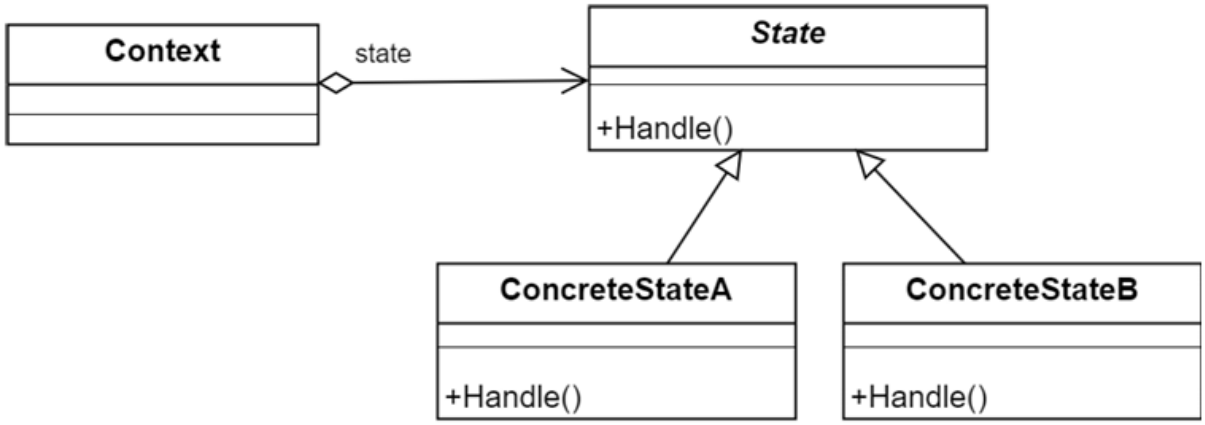
- Strategy – це інтерфейс, який визначає загальний метод для всіх конкретних стратегій. Контекст працює з об'єктами через цей інтерфейс.

- ConcreteStrategy – це класи, що реалізують інтерфейс Strategy. Кожен клас представляє один конкретний алгоритм із сімейства.

6. Яке призначення шаблону «Стан»?

Призначення шаблону «Стан» — дозволити об'єкту змінювати свою поведінку залежно від його внутрішнього стану. Зовні це виглядає так, ніби об'єкт змінив свій клас. Шаблон інкапсулює поведінку, пов'язану з певним станом, в окремі класи.

7. Нарисуйте структуру шаблону «Стан».



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

- Context – це клас поведінка якого змінюється. Він зберігає посилання на поточний об'єкт-стан.

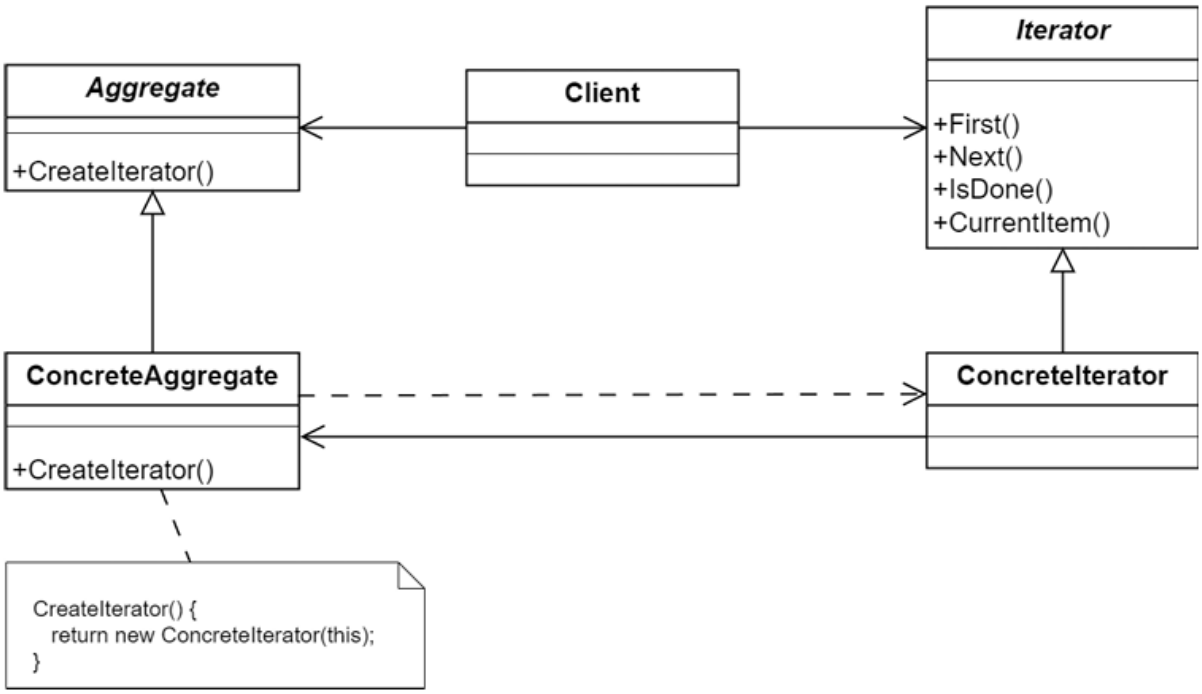
- State – це нтерфейс або абстрактний клас, що визначає методи, які повинні реалізувати всі конкретні стани.

- ConcreteState - це класи, що реалізують інтерфейс State. Кожен клас відповідає за поведінку Контексту в певному стані. Вони також можуть відповідати за перехід Контексту в інший стан.

9. Яке призначення шаблону «Ітератор»?

Призначення шаблону «Ітератор» — надати стандартизований спосіб послідовного доступу до елементів складеного об'єкта (колекції), не розкриваючи його внутрішньої структури (наприклад, масив, список, дерево).

10.Нарисуйте структуру шаблону «Ітератор».



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

- Iterator - це інтерфейс, що визначає методи для обходу колекції

- ConcreteIterator – це клас, що реалізує інтерфейс Iterator для конкретної колекції. Він відстежує поточну позицію обходу.

- Aggregate - це інтерфейс, який описує колекцію та визначає метод для створення Ітератора

- ConcreteAggregate - це клас, що реалізує інтерфейс Aggregate і повертає екземпляр ConcreteIterator, який може обходити його елементи.

12.В чому полягає ідея шаблона «Одинак»?

Ідея шаблону «Одинак» полягає в тому, щоб гарантувати існування тільки одного екземпляра певного класу в усій програмі та надати до нього єдину глобальну точку доступу. Це досягається шляхом приховування конструктора та надання статичного методу, який повертає єдиний екземпляр.

13.Чому шаблон «Одинак» вважають «анти-шаблоном»?

- Порушення принципу єдиної відповідальності - клас-одинак відповідає не тільки за свою бізнес-логіку, але й за контроль над кількістю своїх екземплярів.

- Він вводить глобальний стан у програму, що ускладнює відстеження залежностей та тестування коду.

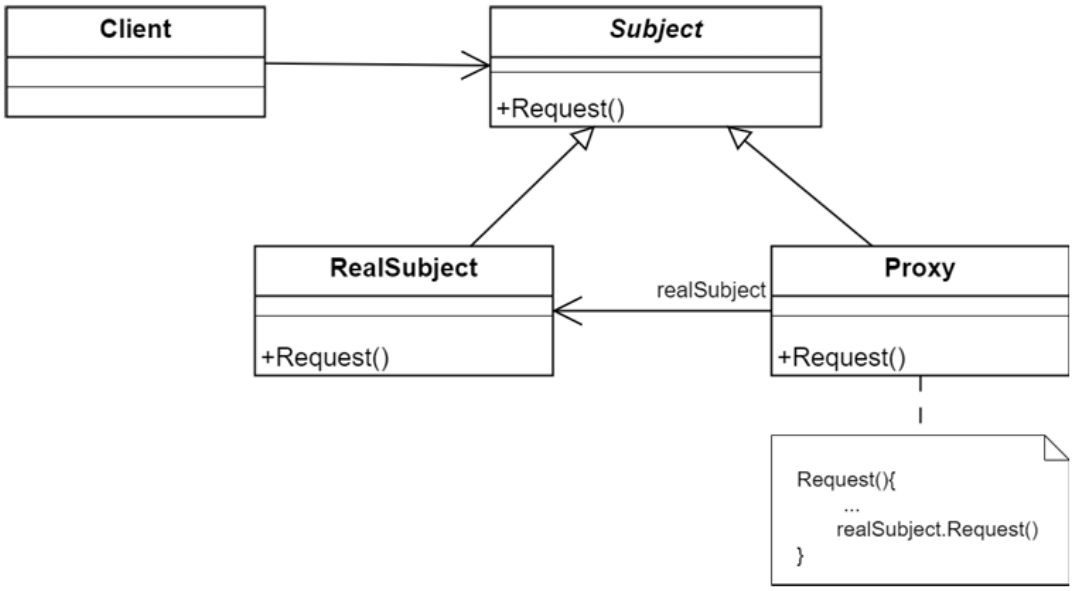
- Класи, що залежать від одинака, важко тестувати в ізоляції, оскільки неможливо легко замінити одинака на об’єкт заглушку

- Залежності від одинака приховані всередині коду, а не передаються через конструктор, що робить архітектуру менш прозорою.

14. Яке призначення шаблону «Проксі»?

Призначення шаблону «Проксі»— надати об'єкт-заступник, який контролює доступ до іншого, реального об'єкта. Проксі має той самий інтерфейс, що й реальний об'єкт, що дозволяє клієнту працювати з ним напряму, не помічаючи підміни.

15. Нарисуйте структуру шаблону «Проксі».



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

- Subject – це інтерфейс, який визначає загальні методи для реального об'єкта та його заступника
- RealSubject це - клас, що містить основну бізнес-логіку. Це "важкий" об'єкт, доступ до якого контролює Проксі.
- Proxy це - клас, що реалізує той самий інтерфейс Subject. Він містить посилання на RealSubject і може виконувати додаткові дії до або після виклику методів реального об'єкта.