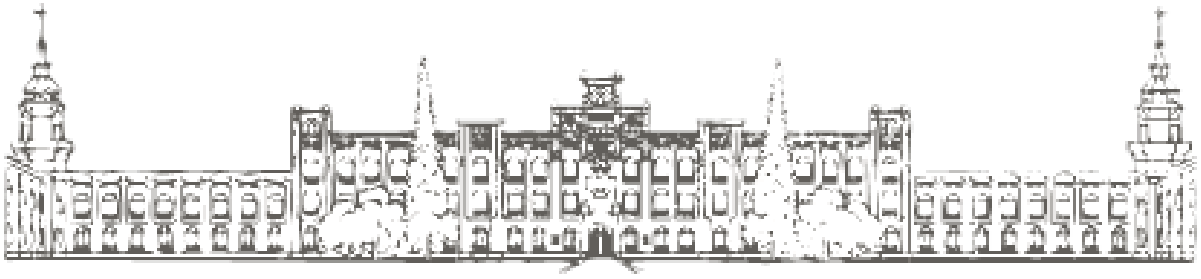


МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

Звіт лабораторної роботи №4

з курсу

«Технології розроблення програмного забезпечення»

Виконавець:
Нікітченко Наталя Олегівна
студентка групи ІА-33
залікова книжка № ІА-3318

«07» 11 2025 р.

Перевірив: **Мягкий М. Ю.**

Київ – 2025

4. ЛАБОРАТОРНА РОБОТА № 4

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.

Теоретичні відомості:

Поняття шаблону проєктування

Будь-який патерн проєктування, використовуваний при розробці інформаційних систем, являє собою формалізований опис, який часто зустрічається в завданнях проєктування, вдале рішення даної задачі, а також рекомендації по застосуванню цього рішення в різних ситуаціях [5]. Крім того, патерн проєктування обов’язково має загальновживане найменування. Правильно сформульований патерн проєктування дозволяє, відшукавши одного разу вдале рішення, користуватися ним знову і знову. Варто підкреслити, що важливим початковим етапом при роботі з патернами є адекватне моделювання розглянутої предметної області. Це є

необхідним як для отримання належним чином формалізованої постановки задачі, так і для вибору відповідних патернів проєктування.

Відповідне використання патернів проєктування дає розробнику ряд незаперечних переваг. Наведемо деякі з них. Модель системи, побудована в межах патернів проєктування, фактично є структурованим виокремленням тих елементів і зв’язків, які значимі при вирішенні поставленого завдання. Крім цього, модель, побудована з використанням патернів проєктування, більш проста і наочна у вивченні, ніж стандартна модель. Проте, не дивлячись на простоту і наочність, вона дозволяє глибоко і всебічно опрацювати архітектуру розроблюваної системи з використанням спеціальної мови.

Застосування патернів проєктування підвищує стійкість системи до зміни вимог та спрощує неминуче подальше доопрацювання системи. Крім того, важко переоцінити роль використання патернів при інтеграції інформаційних систем організації. Також слід зазначити, що сукупність патернів проєктування, по суті, являє собою єдиний словник проєктування, який, будучи уніфікованим засобом, незамінний для спілкування розробників один одним.

Таким чином шаблони представляють собою, підтверджені роками розробок в різних компаніях і на різних проєктах, «ескізи» архітектурних рішень, які зручно застосовувати у відповідних обставинах.

Хід роботи:

Реалізація частини функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

```java
package com.example.graphiceditor.repository;

import com.example.graphiceditor.model.User;
import com.example.graphiceditor.persistence.JpaUtil;
import jakarta.persistence.EntityManager;
import java.util.Optional;

public class UserRepository extends JpaBaseRepository<User, Long> {  5 usages  new *
    public UserRepository() { super(User.class); }  2 usages  new *
    public Optional<User> findByUsername(String username) {  1 usage  new *
        EntityManager em = JpaUtil.emf().createEntityManager();
        try {
            return em.createQuery( s: "select u from User u where u.username=:name", User.class)
                    .setParameter( s: "name", username)
                    .getResultStream().findFirst();
        } finally { em.close(); }
    }
}
```

```java
package com.example.graphiceditor.repository;

import com.example.graphiceditor.model.Project;
import com.example.graphiceditor.persistence.JpaUtil;
import java.util.List;

public class ProjectRepository extends JpaBaseRepository<Project, Long> {  5 usages  new *
    public ProjectRepository() {  2 usages  new *
        super(Project.class);
    }

    public List<Project> findByUserId(Long userId) {  1 usage  new *
        var em = JpaUtil.emf().createEntityManager();
        try {
            return em.createQuery( s: """
                select p from Project p
                where p.user.id = :uid
                order by p.lastModified desc
                """, Project.class).setParameter( s: "uid", userId).getResultList();
        } finally {em.close();}
    }
}
```

```java
package com.example.graphiceditor.repository;

import com.example.graphiceditor.persistence.JpaUtil;
import jakarta.persistence.EntityManager;
import java.util.List;

public abstract class JpaBaseRepository<T, ID> {  3 usages  3 inheritors  new *
    private final Class<T> type;  4 usages
    protected JpaBaseRepository(Class<T> type) { this.type = type; }  3 usages  new *

    public T save(T entity) {  5 usages  new *
        EntityManager em = JpaUtil.emf().createEntityManager();
        try {
            em.getTransaction().begin();
            T merged = em.merge(entity);
            em.getTransaction().commit();
            return merged;
        } finally { em.close(); }
    }

    public T find(ID id) {  no usages  new *
        EntityManager em = JpaUtil.emf().createEntityManager();
        try { return em.find(type, id); }
        finally { em.close(); }
    }

    public List<T> findAll() {  no usages  new *
        EntityManager em = JpaUtil.emf().createEntityManager();
        try {
            return em.createQuery( s: "select e from " + type.getSimpleName() + " e", type).getResultList();
        } finally { em.close(); }
    }
```

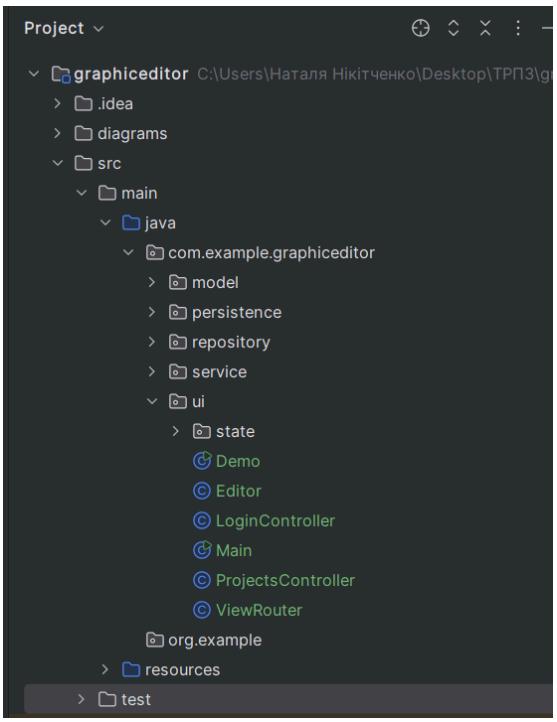
```java
package com.example.graphiceditor.service;

import com.example.graphiceditor.model.User;
import com.example.graphiceditor.repository.UserRepository;

public class AuthService {  3 usages  new *
    private final UserRepository users = new UserRepository();  2 usages

    public User loginOrCreate(String username) {  no usages  new *
        return users.findByUsername(username).orElseGet(() -> {
            var u = new User();
            u.setUsername(username);
            u.setAuthorized(true);
            return users.save(u);
        });
    }
}
```

```java
package com.example.graphiceditor.ui;

import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

public class ViewRouter {  4 usages  new *

    private static Stage primaryStage;  7 usages

    public static void setPrimaryStage(Stage stage) {  1 usage  new *
        primaryStage = stage;
    }

    public static void show(String fxmlPath, String title) {  1 usage  new *
        try {
            FXMLLoader loader = new FXMLLoader(ViewRouter.class.getResource(fxmlPath));
            Parent root = loader.load();

            primaryStage.setTitle(title);
            primaryStage.setScene(new Scene(root));
            primaryStage.show();
        } catch (Exception ex) {
            ex.printStackTrace();
            throw new RuntimeException("Cannot load FXML: " + fxmlPath, ex);
        }
    }
```

```java
    public static <T> T showAndGetController(String fxmlPath, String title) {  no usages  new *
        try {
            FXMLLoader loader = new FXMLLoader(ViewRouter.class.getResource(fxmlPath));
            Parent root = loader.load();

            primaryStage.setTitle(title);
            primaryStage.setScene(new Scene(root));
            primaryStage.show();

            return loader.getController();
        } catch (Exception ex) {
            ex.printStackTrace();
            throw new RuntimeException("Cannot load FXML: " + fxmlPath, ex);
        }
    }
}
```

Реалізація одного з розглянутих шаблонів, шаблону “State”.

Структура проекту з цим шаблоном:



Реалізація програмна:

Контекст — Editor:

```java
package com.example.graphiceditor.ui;

import com.example.graphiceditor.ui.state.ToolState;
import com.example.graphiceditor.ui.state.IdleState;

import java.util.ArrayList;
import java.util.List;

public class Editor {  14 usages  new *

    private ToolState state = new IdleState();  4 usages
    private int[][] selection;  6 usages
    private java.util.List<int[]> path = new java.util.ArrayList<>();  7 usages

    public void setState(ToolState newState) {  3 usages  new *
        System.out.println("[Editor] Switch -> " + newState.getName());
        this.state = newState;
    }

    public void mouseDown(int x, int y) { state.onMouseDown(x, y,  ctx: this); }  2 usages  new *
    public void mouseMove(int x, int y) { state.onMouseMove(x, y,  ctx: this); }  3 usages  new *
    public void mouseUp(int x, int y)   { state.onMouseUp(x, y,  ctx: this); }  3 usages  new *


    public void beginPath(int x, int y) {  no usages  new *
        path.clear();
        path.add(new int[]{x, y});
        System.out.println("[Canvas] begin path at (" + x + "," + y + ")");
    }
```

```
30
31      public void addPoint(int x, int y) {  no usages  new *
32          path.add(new int[]{x, y});
33          if (path.size() % 5 == 0)
34              System.out.println("[Canvas] path len=" + path.size());
35      }
36
37      public void stroke() {  no usages  new *
38          System.out.println("[Canvas] stroke path, points=" + path.size());
39          path.clear();
40      }
41
42      public void setSelection(int x1, int y1, int x2, int y2) {  no usages  new *
43          this.selection = new int[][]{{x1, y1}, {x2, y2}};
44          System.out.println("[Canvas] selection: (" + x1 + "," + y1 + ") → (" + x2 + "," + y2 + ")");
45      }
46
47      public void moveSelection(int dx, int dy) {  1 usage  new *
48          if (selection == null) {
49              System.out.println("[Canvas] no selection to move");
50              return;
51          }
52          selection[0][0] += dx;
53          selection[1][0] += dx;
54          selection[0][1] += dy;
55          selection[1][1] += dy;
56
57          System.out.println("[Canvas] moved selection dx=" + dx + ", dy=" + dy);
58      }
59  }
```

IdleState — неактивний інструмент:

```java
package com.example.graphiceditor.ui.state;

import com.example.graphiceditor.ui.Editor;

public class IdleState implements ToolState {  2 usages  new *

    @Override  no usages  new *
    public void onMouseDown(int x, int y, Editor ctx) {
        System.out.println("[Idle] Click ignored");
    }

    @Override  no usages  new *
    public void onMouseMove(int x, int y, Editor ctx) {}

    @Override  no usages  new *
    public void onMouseUp(int x, int y, Editor ctx) {}

    @Override  new *
    public String getName() {
        return "Idle";
    }
}
```

PenState — інструмент “Перо”:

```java
package com.example.graphiceditor.ui.state;

import com.example.graphiceditor.ui.Editor;

public class PenState implements ToolState {  1 usage  new *

    @Override  no usages  new *
    public void onMouseDown(int x, int y, Editor ctx) {
        System.out.println("[Pen] Start drawing");
        ctx.beginPath(x, y);
    }

    @Override  no usages  new *
    public void onMouseMove(int x, int y, Editor ctx) {
        ctx.addPoint(x, y);
    }

    @Override  no usages  new *
    public void onMouseUp(int x, int y, Editor ctx) {
        ctx.addPoint(x, y);
        ctx.stroke();
        System.out.println("[Pen] Finish drawing");
        ctx.setState(new IdleState());
    }

    @Override  new *
    public String getName() {
        return "Pen";
    }
}
```

SelectState — інструмент “Виділення”:

```java
package com.example.graphiceditor.ui.state;

import com.example.graphiceditor.ui.Editor;

public class SelectState implements ToolState {  1 usage  new *

    private int anchorX, anchorY;  3 usages
    private boolean dragging = false;  3 usages

    @Override  no usages  new *
    public void onMouseDown(int x, int y, Editor ctx) {
        anchorX = x;
        anchorY = y;
        dragging = true;
        System.out.println("[Select] Anchor at (" + x + "," + y + ")");
    }

    @Override  no usages  new *
    public void onMouseMove(int x, int y, Editor ctx) {
        if (dragging) {
            int x1 = Math.min(anchorX, x);
            int y1 = Math.min(anchorY, y);
            int x2 = Math.max(anchorX, x);
            int y2 = Math.max(anchorY, y);
            ctx.setSelection(x1, y1, x2, y2);
        }
    }
```

```java
    @Override  no usages  new *
    public void onMouseUp(int x, int y, Editor ctx) {
        dragging = false;
        System.out.println("[Select] Selection fixed");
        ctx.setState(new MoveState()); // автоматичний перехід
    }

    @Override  new *
    public String getName() {
        return "Select";
    }
}
```

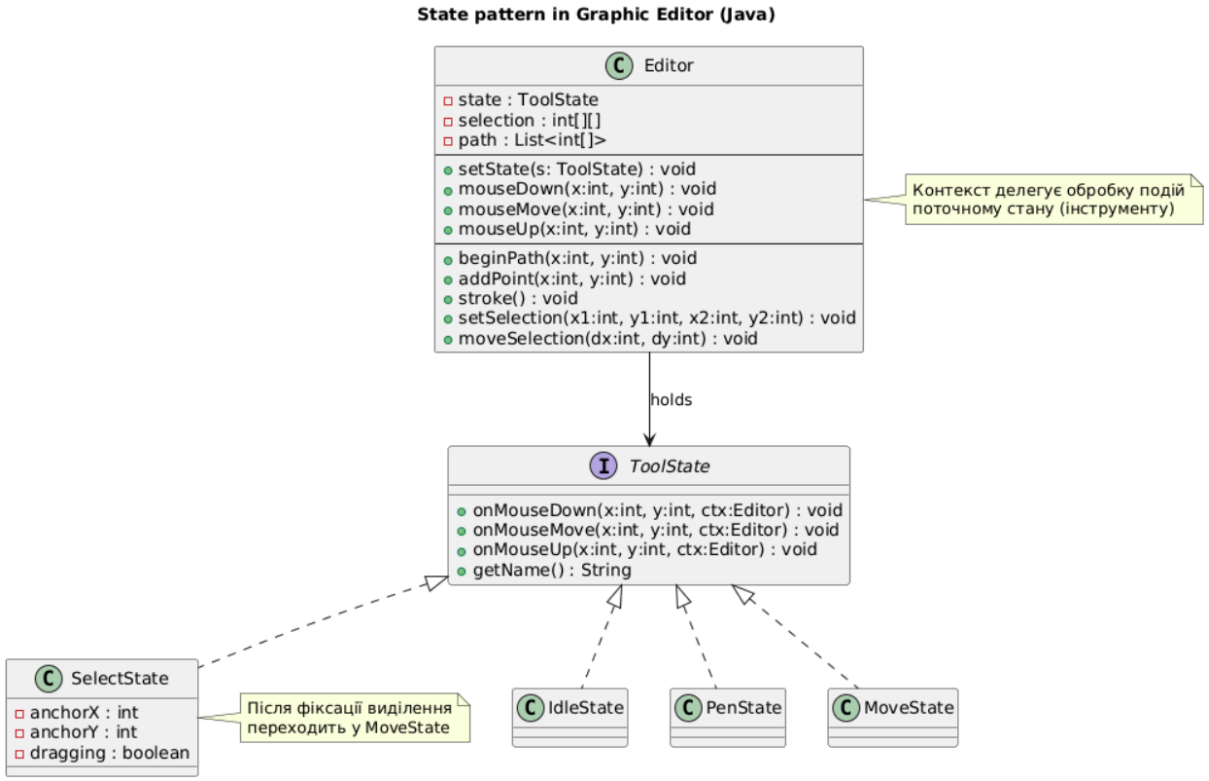
MoveState — інструмент переміщення:

```java
package com.example.graphiceditor.ui.state;

import com.example.graphiceditor.ui.Editor;

public class MoveState implements ToolState {  1 usage  new *
    @Override public void onMouseDown(int x, int y, Editor ctx) {
        System.out.println("[Move] Drag to move selection");
    }
    @Override public void onMouseMove(int x, int y, Editor ctx) {
        ctx.moveSelection( dx: 1,  dy: 0); // демо-рух
    }
    @Override public void onMouseUp(int x, int y, Editor ctx) {
        System.out.println("[Move] Move done");
        ctx.setState(new IdleState());
    }
    @Override public String getName() { return "Move"; }  new *
}
```

Демо (main):

```java
package com.example.graphiceditor.ui;

import com.example.graphiceditor.ui.state.*;

public class Demo {  new *
    public static void main(String[] args) {  new *

        Editor ed = new Editor();

        // Pen tool
        ed.setState(new PenState());
        ed.mouseDown( x: 10,  y: 10);
        for (int x = 11; x <= 30; x++)
            ed.mouseMove(x,  y: 10 + (x % 3));
        ed.mouseUp( x: 30,  y: 12);

        // Select tool
        ed.setState(new SelectState());
        ed.mouseDown( x: 5,  y: 5);
        ed.mouseMove( x: 20,  y: 15);
        ed.mouseUp( x: 20,  y: 15);

        // Move tool automatically activated
        for (int i = 0; i < 5; i++)
            ed.mouseMove( x: 0,  y: 0);
        ed.mouseUp( x: 0,  y: 0);
    }
}
```

Діаграма класів, яка представляє використання шаблону в реалізації системи:



ToolState інтерфейс

```
1   package com.example.graphiceditor.ui.state;
2
3   import com.example.graphiceditor.ui.Editor;
4
5   public interface ToolState {  4 usages  4 implementati
6
7       void onMouseDown(int x, int y, Editor ctx);
8
9       void onMouseMove(int x, int y, Editor ctx);
10
11      void onMouseUp(int x, int y, Editor ctx);  no
12
13      String getName();  4 implementations  new *
14  }
```

Висновки:

У ході роботи було реалізовано фрагмент програмної системи «Графічний редактор» із використанням шаблону проектування **State**. Даний патерн дозволив організувати змінювану поведінку інтерфейсу залежно від активного інструмента (стану), що є характерним для графічних редакторів.

Було створено інтерфейс **ToolState** та чотири конкретні стани: **IdleState**, **PenState**, **SelectState** і **MoveState**, а також клас **Editor**, який виконує роль контексту та делегує обробку подій відповідному стану. Завдяки цьому вдалося розподілити поведінку між окремими класами, уникнути надмірних умовних конструкцій та забезпечити гнучкість розширення системи новими інструментами.

Діаграма класів підтверджує коректну реалізацію шаблону State: кожен інструмент інкапсулює власну логіку обробки подій, а переходи між станами відбуваються динамічно в процесі роботи. Таким чином, поставлену задачу виконано, а використання патерну дозволило підвищити структурованість, зрозумілість і масштабованість застосунку.

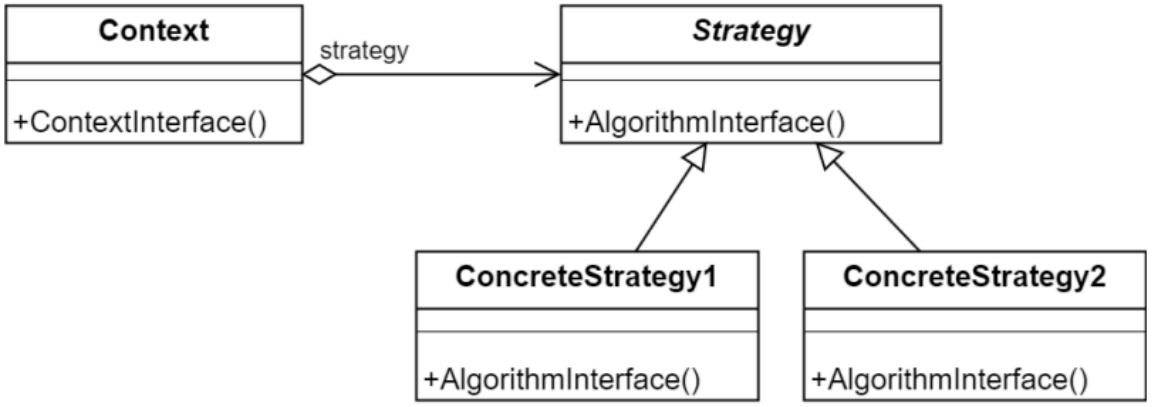
Контрольні питання:

1.Шаблон проєктування — це перевірене повторюване рішення типових проблем проєктування ПЗ на рівні взаємодії класів/об’єктів (а не готовий код).

2.Використовують шаблони, щоб підвищити зрозумілість і підтримуваність коду, зменшити зв’язність, покращити гнучкість/розширюваність і повторно застосовувати найкращі практики.

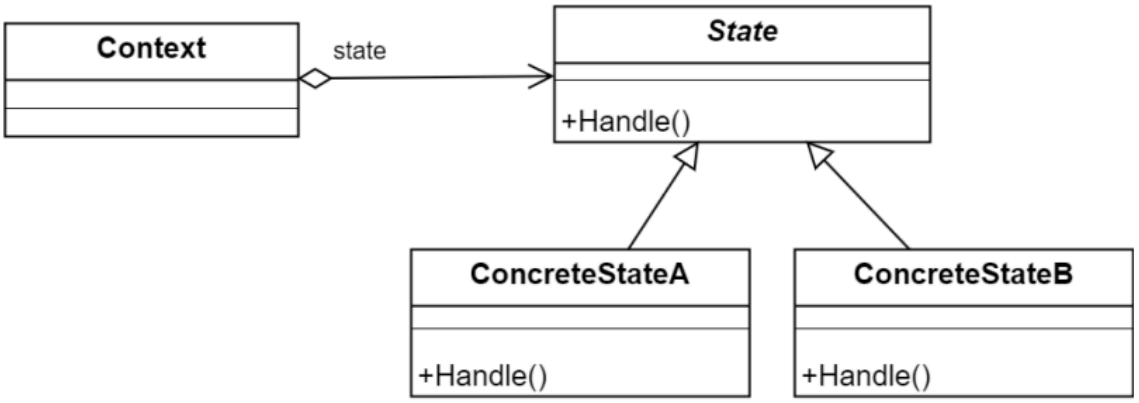
3.«Стратегія» (Strategy) інкапсулює взаємозамінні алгоритми за спільним інтерфейсом і дозволяє перемикати їх під час виконання без зміни коду клієнта.

4.Структура «Стратегії» (спрощено, ASCII-UML):

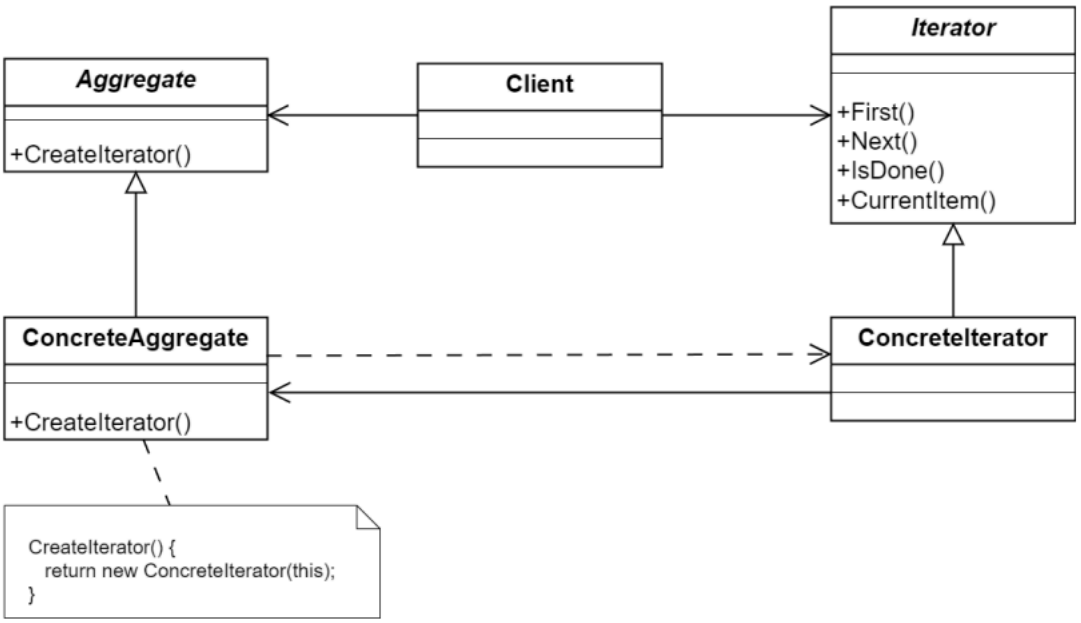


5. Класи та взаємодія «Стратегії»: Context містить посилання на Strategy; Strategy — інтерфейс алгоритму; ConcreteStrategyA/B — реалізації. Context делегує виконання алгоритму вибраній стратегії.
6. «Стан» (State) дозволяє об’єкту змінювати свою поведінку при зміні внутрішнього стану: замість розгалужень if/switch — делегування на об’єкти станів.

7. Структура «Стану»:



8. Класи та взаємодія «Стану»: Context тримає поточний State; State — інтерфейс дій; ConcreteStateA/B реалізують дії й можуть переводити Context у інший стан (виклик setState).
9. «Ітератор» (Iterator) надає стандартизований спосіб покрокового доступу до елементів колекції без розкриття її внутрішньої структури.
10. Структура «Ітератора»:



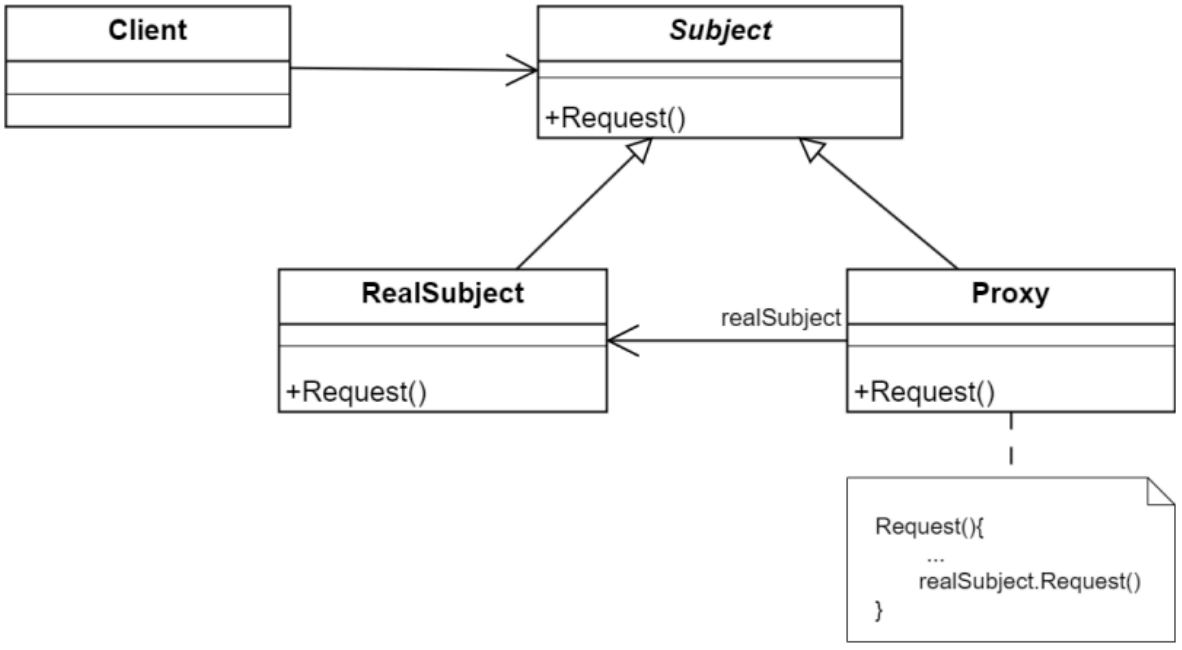
11. Класи та взаємодія «Ітератора»: Aggregate/ConcreteAggregate — колекція; Iterator/ConcreteIterator — інтерфейс і реалізація перебору (hasNext(), next()); клієнт отримує ітератор у Aggregate і рухається елементами через Iterator.

12. Ідея «Одинака» (Singleton): гарантувати, що клас має рівно один екземпляр і надати глобальну точку доступу до нього.

13. Чому «Одинак» часто вважають анти-шаблоном: він ускладнює тестування (глобальний стан), приховано підвищує зв’язність, порушує принцип єдиного обов’язку та часто замінюється DI/контейнерами, конфігурацією або фабриками.

14. «Проксі» (Proxy) надає сурогат/замінник об’єкта для контролю доступу до нього: ліниве створення, кешування, безпека, віддалений доступ, логування тощо.

15. Структура «Проксі»:



16. Класи та взаємодія «Проксі»: Subject — інтерфейс сервісу; RealSubject — реальна реалізація; Proxy реалізує Subject, тримає посилання на RealSubject і додає додаткову поведінку (перевірки, кеш, мережа), перш ніж делегувати виклик реальному об’єкту.