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

# Технологія розроблення програмного забезпечення

«ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»

Варіант 8

Виконав
студент групи ІА-13
Крутиус Владислав Віталійович

Київ 2023р.

**Мета:** Дослідити шаблони «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY».

**Завдання.**
1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми.

## 1. Шаблон «SINGLETON»

Шаблон «SINGLETON» є одним із базових шаблонів проектування в програмній інженері. Його основна ідея полягає в тому, щоб гарантувати, що у системі існує лише один екземпляр певного класу, незважаючи на те, скільки разів він буде інстанційований.
Цей шаблон особливо корисний в ситуаціях, коли необхідно мати єдиний об'єкт, який керує загальними ресурсами чи конфігурацією, або коли треба уникнути надмірного використання пам'яті чи ресурсів на надто багато копій одного і того ж об'єкта.

Основні характеристики та особливості шаблону "SINGLETON":

- Приватний конструктор (Private Constructor): Клас має приватний конструктор, щоб заборонити зовнішнім частинам створювати нові екземпляри.
- Статичний метод для отримання інстанції (Static Method for Instance Retrieval): Клас надає статичний метод, який повертає єдиний екземпляр класу.
- Змінна для збереження єдиного екземпляру (Variable for Storing the Singleton): У класі є статична змінна, яка зберігає єдиний екземпляр.
- Ліниве створення (Lazy Initialization): Єдиний екземпляр може бути створений лише тоді, коли це потрібно, тобто під час першого виклику статичного методу для отримання інстанції.
- Потокобезпека (Thread Safety): Потокобезпечність може бути реалізована, щоб уникнути конфліктів, коли кілька потоків намагаються створити інстанцію.
- Глобальний доступ (Global Access): Єдиний екземпляр класу стає глобально доступним для всіх частин програми.

Шаблон "SINGLETON" використовується в багатьох сценаріях розробки, коли потрібно гарантувати, що певний клас має лише одну інстанцію, і ця інстанція є загальною для всієї програми.

## 2. Шаблон «ITERATOR»

Шаблон "ITERATOR" є одним із п'яти базових шаблонів проектування, які входять до групи "Поведінкові шаблони". Він вирішує завдання надання уніфікованого способу доступу до елементів складної структури даних без розголошення її внутрішньої організації.

- Основна ідея шаблону "ITERATOR" полягає в тому, щоб виділити логіку ітерації в окремий клас (ітератор), щоб можна було працювати з колекцією об'єктів без необхідності знати, як саме ця колекція організована.

Основні характеристики та особливості шаблону "ITERATOR":

- Інтерфейс ітератора (Iterator Interface): Визначає методи для навігації по колекції, такі як отримання наступного елементу, перевірка кінця колекції тощо.
- Конкретний ітератор (Concrete Iterator): Реалізує інтерфейс ітератора та управляє поточним положенням в колекції.
- Колекція (Collection): Визначає інтерфейс для створення ітератора.
- Конкретна колекція (Concrete Collection): Реалізує інтерфейс колекції, надаючи конкретну реалізацію структури даних.
- Можливість ітерування в кількох напрямках (Bidirectional Iteration): Деякі ітератори можуть дозволяти рухатися не лише вперед, а й назад по колекції.

Підтримка вилучення елементів (Optional Removal Support): Деякі ітератори можуть дозволяти вилучати елементи з колекції.

Шаблон "ITERATOR" дозволяє відокремити спосіб ітерації від способу зберігання даних, що робить код більш гнучким та легше зрозумілим. Він широко використовується при роботі зі складними даними, такими як списки, дерева, бази даних тощо.

## 3. Шаблон «PROXY»

Шаблон "PROXY" є одним з базових шаблонів проектування в програмній інженері. Основна мета цього шаблону - надати замісник (проксі) для іншого об'єкта з метою контролю, обгортання чи заборони деяких операцій.

Основні характеристики та особливості шаблону "PROXY":

- Замісник (Proxy): Клас, який містить схожий інтерфейс з оригінальним об'єктом, але може додавати додаткову логіку до методів.
- Оригінальний об'єкт (Real Subject): Клас, який реалізує основну функціональність, до якої може бути доданий проксі.
- Інтерфейс (Subject Interface): Визначає загальний інтерфейс для замісника та оригінального об'єкта, що дозволяє використовувати їх взаємозамінно.
- Керування доступом (Access Control): Проксі може контролювати доступ до методів оригінального об'єкта, встановлюючи власні обмеження.
- Ліниве завантаження (Lazy Loading): Проксі може відкладати завантаження ресурсів до того моменту, коли вони дійсно потрібні.
- Кешування (Caching): Проксі може зберігати результати попередніх викликів та повертати їх, щоб уникнути повторних обчислень.
- Вилучення (Protection): Проксі може забороняти виклики до певних методів або контролювати доступ до конфіденційних даних.
- Ведення обліку (Logging): Проксі може вести облік викликів до оригінального об'єкта для статистики чи відлагодження.

Шаблон "PROXY" дозволяє створювати обгортки для різних об'єктів з різними цілями, такими як контроль доступу, ліниве завантаження даних, кешування тощо. Він особливо корисний в ситуаціях, коли потрібно додатково керувати викликами до об'єкта без прямого втручання в його реалізацію.

## 4. Шаблон «STATE»

Шаблон «STATE» (Стан) є одним із поведінкових шаблонів проектування, який дозволяє об'єкту змінювати свою поведінку в залежності від свого поточного стану.

Основна ідея шаблону «STATE» - розділити поведінку об'єкта на окремі стани, кожен з яких визначає свою унікальну поведінку. При цьому перехід між станами відбувається динамічно під час виконання програми.

Основні характеристики та особливості шаблону «STATE»:

- Контекст (Context): Клас, що містить посилання на поточний стан та викликає його методи. Він також може змінювати свій поточний стан.
- Стани (States): Класи, що визначають конкретну поведінку для кожного стану. Кожен стан має методи, які визначають його особливу реакцію на вхідні події.
- Перехід між станами (State Transitions): Контекст може змінювати свій поточний стан, спричиняючи перехід до іншого стану.
- Управління зовнішніми подіями (Handling External Events): Контекст може обробляти зовнішні події та викликати методи відповідного стану.

Розділення внутрішнього стану та поведінки (Separation of Internal State and Behavior): Поведінка пов'язана з конкретним станом, що дозволяє додавати нові стани без модифікації контексту.

Уникнення умовних операторів (Avoidance of Conditional Statements): Шаблон дозволяє уникнути великої кількості умовних операторів, що можуть стати заплутаними у великих програмах.

Шаблон «STATE» дозволяє створювати програми зі складною логікою, що залежить від стану об'єкта. Він особливо корисний у випадках, коли потрібно керувати поведінкою об'єкта в залежності від його поточного стану, та уникнути великої кількості умовних операторів.

## 5. Шаблон «STRATEGY»

Шаблон «STRATEGY» (Стратегія) є одним з поведінкових шаблонів проектування, який дозволяє об'єкту вибирати або змінювати алгоритм виконання певної задачі в режимі виконання.

Основна ідея шаблону «STRATEGY» - виділити алгоритм в окремий клас-стратегію, щоб його можна було змінювати незалежно від контексту, який використовує цей алгоритм.

Основні характеристики та особливості шаблону «STRATEGY»:

- Контекст (Context): Клас, що містить посилання на об'єкт стратегії. Він використовує цю стратегію для виконання певної задачі.
- Інтерфейс стратегії (Strategy Interface): Визначає загальний інтерфейс для всіх конкретних стратегій, що реалізують алгоритм.
- Конкретні стратегії (Concrete Strategies): Класи, що реалізують конкретні алгоритми. Кожна стратегія може виконувати задачу по-своєму.

Зміна стратегії (Changing Strategies): Контекст може динамічно змінювати свою стратегію під час виконання програми.

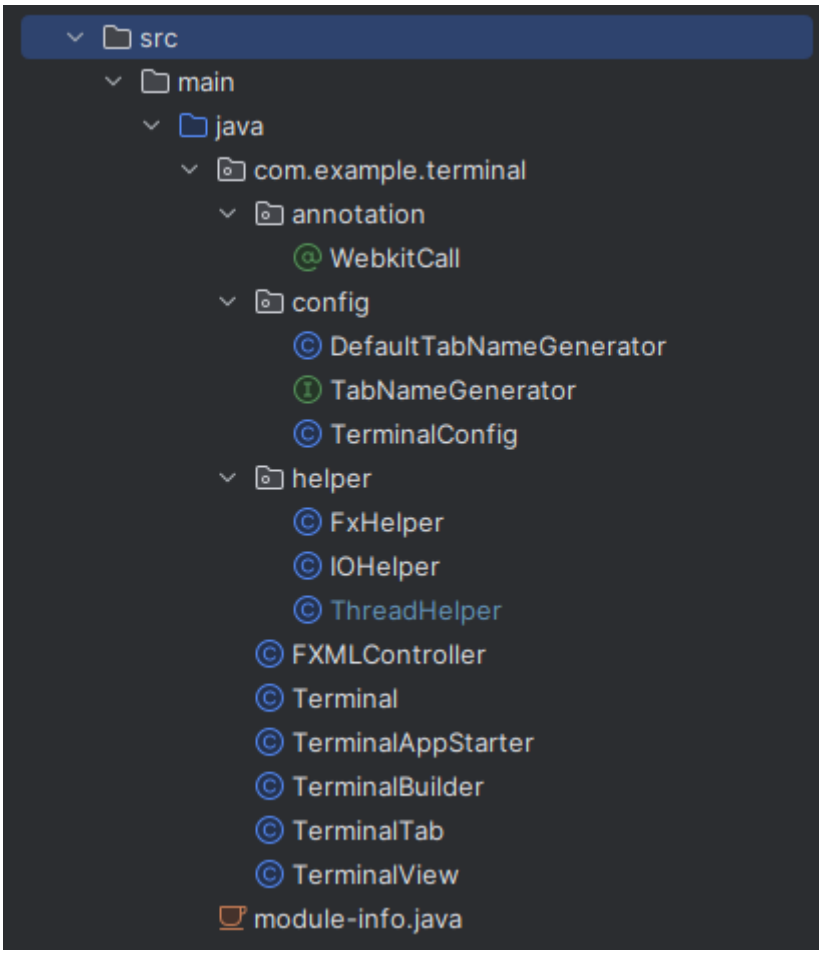
Уникнення умовних операторів (Avoidance of Conditional Statements): Шаблон дозволяє уникнути великої кількості умовних операторів для вибору алгоритму.

Розширення можливостей (Adding New Strategies): Нові стратегії можуть бути додані без зміни контексту, що робить систему більш гнучкою.

Підтримка замінності (Support for Substitution): Різні стратегії можуть бути використані в одному контексті, забезпечуючи можливість заміщення.

Шаблон «STRATEGY» дозволяє вибирати та змінювати алгоритми в залежності від потреб програми. Він особливо корисний в ситуаціях, коли потрібно забезпечити можливість вибору різних алгоритмів для вирішення одного завдання.

**Структура класів**




**Використання патерну State та Strategy:**

Розглянемо патерн на класі TerminalTab

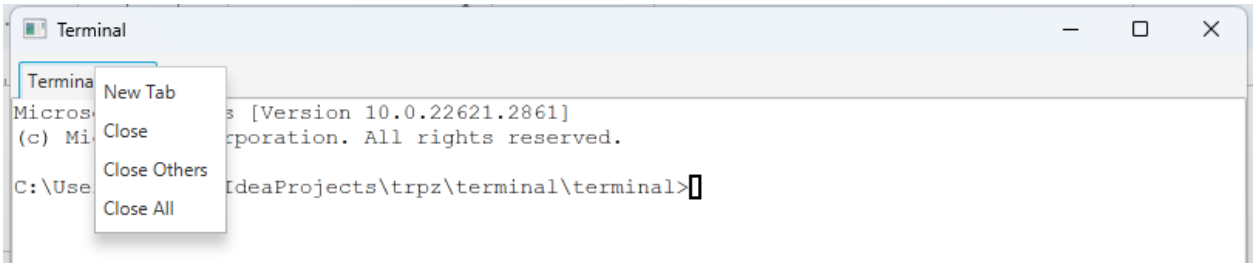
```
// imports
public class TerminalTab extends Tab {


   private final Terminal terminal;
   private final TabNameGenerator tabNameGenerator;
   private static final String NEW_TAB_KEY = "T";


   public TerminalTab(TerminalConfig terminalConfig,
TabNameGenerator tabNameGenerator, Path terminalPath) {
       this(new Terminal(terminalConfig, terminalPath),
tabNameGenerator);
   }
```

```
   public TerminalTab(Terminal terminal, TabNameGenerator
tabNameGenerator) {
       this.terminal = terminal;
       this.tabNameGenerator = tabNameGenerator;

       this.terminal.addEventFilter(KeyEvent.KEY_PRESSED, event -> {

           if (event.isShortcutDown() &&
NEW_TAB_KEY.equalsIgnoreCase(event.getText())) {
               newTerminal();
           }
       });

       this.setOnCloseRequest(event -> {
           event.consume();
           closeTerminal();
       });

       final String tabName = getTabNameGenerator().next();
       setText(tabName);

       final ContextMenu contextMenu = new ContextMenu();
       final MenuItem newTab = new MenuItem("New Tab");
       final MenuItem closeTab = new MenuItem("Close");
       final MenuItem closeOthers = new MenuItem("Close Others");
       final MenuItem closeAll = new MenuItem("Close All");

       newTab.setOnAction(this::newTerminal);
       closeTab.setOnAction(this::closeTerminal);
       closeAll.setOnAction(this::closeAllTerminal);
       closeOthers.setOnAction(this::closeOtherTerminals);

       contextMenu.getItems().addAll(newTab, closeTab, closeOthers,
closeAll);
```

```
        this.setContextMenu(contextMenu);


        setContent(terminal);
    }


    private void closeOtherTerminals(ActionEvent actionEvent) {
        final ObservableList<Tab> tabs =
FXCollections.observableArrayList(this.getTabPane().getTabs());
        for (final Tab tab : tabs) {
            if (tab instanceof TerminalTab) {
                if (tab != this) {
                    ((TerminalTab) tab).closeTerminal();
                }
            }
        }
    }


    private void closeAllTerminal(ActionEvent actionEvent) {
        final ObservableList<Tab> tabs =
FXCollections.observableArrayList(this.getTabPane().getTabs());
        for (final Tab tab : tabs) {
            if (tab instanceof TerminalTab) {
                ((TerminalTab) tab).closeTerminal();
            }
        }
    }


    public void newTerminal(ActionEvent... actionEvent) {
        final TerminalTab terminalTab = new
TerminalTab(getTerminalConfig(), getTabNameGenerator(),
getTerminalPath());
        getTabPane().getTabs().add(terminalTab);
        getTabPane().getSelectionModel().select(terminalTab);
    }
```

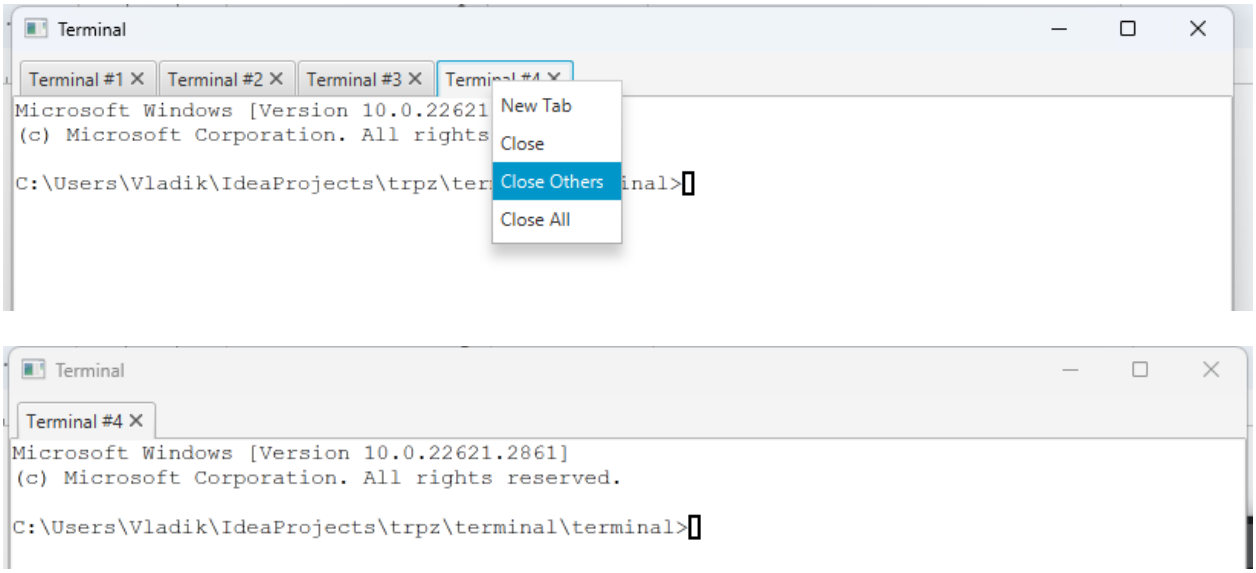
```
    public void closeTerminal(ActionEvent... actionEvent) {
        ThreadHelper.runActionLater(() -> {
            final ObservableList<Tab> tabs =
this.getTabPane().getTabs();
            if (tabs.size() == 1) {
                newTerminal(actionEvent);
            }
            tabs.remove(this);

            destroy();
        });
    }

    public void destroy() {
        ThreadHelper.start(() -> {
            while (Objects.isNull(getProcess())) {
                ThreadHelper.sleep(250);
            }
            getProcess().destroy();
            IOHelper.close(getInputReader(), getErrorReader(),
getOutputWriter());
        });
    }
  // getters and setters
}
```

Відкривши застосунок терміналу, ми можемо побачити що при натисканні правою кнопкою миші по вкладці терміналу ми отримаємо наступні опції



Кожен вибір є окремою стратегією, із відповідною логікою обробки

Також створивши декілька нових вікон терміналу, в залежності стану(яка саме вкладка зараз відкрита), ми маємо опції закрити цю вкладку, закрити всі окрім відкритої, або закрити всі.




**закриття всіх вікон окрім поточного відкритого**

**Висновок:**
Лабораторна робота була присвячена дослідженню та реалізації декількох важливих шаблонів проектування: «SINGLETON», «ITERATOR», «PROXY», «STATE», та «STRATEGY». Кожен із них має важливе призначення та може бути використаний для вирішення різноманітних завдань у розробці програмного забезпечення.

Шаблон проектування "SINGLETON": Цей шаблон дозволяє гарантувати, що у системі існує лише один екземпляр певного класу. Це особливо корисно, коли потрібно мати єдиний об'єкт, який керує загальними ресурсами, наприклад, конфігурацією, пулом з'єднань до бази даних тощо.

Шаблон проектування "ITERATOR": Цей шаблон дозволяє послідовно переглядати елементи складної структури даних без розголошення її внутрішньої реалізації. Це дозволяє працювати зі списками, колекціями та іншими структурами даних без необхідності знати, як вони внутрішньо організовані.

Шаблон проектування "PROXY": Цей шаблон дозволяє створювати об'єкти-замісники, які можуть виступати в ролі замінника реальних об'єктів. Це особливо корисно, коли потрібно контролювати доступ до об'єкта, відстрілювати віддалені виклики, кешувати дані тощо.

Шаблон проектування "STATE": Цей шаблон дозволяє змінювати поведінку об'єкта в залежності від його поточного стану. Він рекомендується, коли об'єкт

може змінювати свій стан у відповідь до подій, але не хоче змінювати свій інтерфейс.

Шаблон проектування "STRATEGY": Цей шаблон дозволяє вибирати алгоритм виконання певної задачі в режимі виконання. Він надає можливість обмінювати алгоритми без внесення змін у код клієнта.