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 №8

із дисципліни «**Технології розроблення програмного забезпечення**»

Тема **«Шаблони «Composite», «Flyweight», «Interpreter», «Visitor»»**

Виконав:

студент групи ІА–24

Яблонський Д.Б.

Перевірив:

Мягкий М.Ю.

Київ 2024

Зміст

Мета: метою виконання лабораторної роботи є вивчення та практичне застосування шаблонів проєктування, таких як «Composite», «Flyweight», «Interpreter» та «Visitor», для створення ефективних і гнучких програмних рішень. Ця лабораторна робота спрямована на розвиток навичок у використанні різних патернів для вирішення задач у розробці програмного забезпечення.

Теоретичні відомості

Шаблони «Composite», «Flyweight», «Interpreter» та «Visitor» належать до структурних і поведінкових патернів, які вирішують специфічні проблеми в організації коду. «Composite» та «Flyweight» зосереджуються на структурі об'єктів і оптимізації їх використання, тоді як «Interpreter» та «Visitor» забезпечують розширюваність та можливість виконання складних операцій у різних контекстах. Далі розглянемо кожен із них детальніше.

Composite використовується для роботи з ієрархіями об'єктів, де є відношення "частина-ціле". Завдяки цьому шаблону клієнтський код може однаково взаємодіяти як з окремими об'єктами, так і з групами об'єктів. Це досягається за рахунок створення спільного інтерфейсу для компонентів дерева, незалежно від того, чи це простий елемент, чи складна структура. Composite застосовується, наприклад, у файлових системах, де файли й папки мають однаковий набір операцій.

Flyweight зменшує споживання пам'яті та підвищує продуктивність шляхом повторного використання об'єктів. Основна ідея полягає в поділі стану об'єкта на внутрішній і зовнішній: внутрішній зберігається в спільному пулі, а зовнішній передається під час виконання операцій. Цей підхід особливо ефективний у випадках, коли в системі є велика кількість об'єктів з повторюваним станом, наприклад, символів у текстових редакторах чи графічних елементах у програмі.

Interpreter забезпечує можливість інтерпретації виразів або команд певної мови. Він визначає граматику мови у вигляді набору класів, де кожен клас відповідає за окремий елемент граматики. Вирази інтерпретуються шляхом побудови дерева виконання, яке дозволяє розуміти й обробляти вхідні

дані. Interpreter часто використовується в конфігураційних системах, SQL-запитах чи обробці математичних формул.

Visitor дозволяє відокремити логіку операцій від структури об'єктів, над якими ці операції виконуються. Це досягається через створення класів-відвідувачів, які "проходять" через об'єкти структури й виконують потрібні дії. Visitor корисний, коли необхідно додати нові операції до складних об'єктних структур, наприклад, для аналізу, модифікації чи рендерингу даних, зберігаючи при цьому стабільність основного коду.

Хід роботи

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.

Варіант 20

Тема: Mind-mapping software

Реалізація шаблону Composite

В застосунку Mind-mapping software важливим аспектом, яклму варто приділити багато уваги є відображення карти в інтерфейсі користувача. Коли користувач заходить на сторінку карти, для нього відображається карта у вигляді даних про неї, та вузлів, які належать цій карті. Тобто можна описати цей процес як відображення самої карти, а також вкладених у нех вузлів, де відображення карти провокуватиме відображення і всіх вкладених в неї вузлів. Умовно, карта має операцію “render”, що викликатиме таку саму операцію “render” для кожного вкладеного вузла. Таку поведінку компонентів описує шаблон, Composite, який ми і використаємо.

```tsx
const MindMap = async ({ params }: { params: Promise<{ mindMapId: string }> }) => {
  const mindMapId = Number((await params).mindMapId);
  const mindMap = mindMapMapper[mindMapId];

  return (
    <MindMapPage fullMindMap={mindMap}/>
  );
};

export default MindMap;  no usages  yablonya
```

Рис. 1 - Компонент MindMap

На рис. 1 зображено React компонент MindMap, який відповідає сторінці карти пам’яті, що бачить користувач. В цьому компоненті викликається відображення самої карти пам’яті шляхом виклику компонента MindMapPage

MindMapPage.tsx

```tsx
import React, { FC } from 'react';
import { FullMindMapType } from "@/types/FullMindMapType";
import NodesContainer from "@/components/common/nodes-container/NodesContainer";
import "./MindMapPage.scss";

interface MindMapPageProps {  Show usages  yablonya
  fullMindMap: FullMindMapType;
}

const MindMapPage: FC<MindMapPageProps> = ({ fullMindMap }) => {  Show usages  ya
  return (
    <div className="mind-map-page">
      <h2>{fullMindMap.mindMap.name}</h2>
      <NodesContainer nodes={fullMindMap.nodes}/>
    </div>
  );
};

export default MindMapPage;  Show usages  yablonya
```

Рис. 2 - Компонент MindMapPage

NodesContainer.tsx

```tsx
import React, { FC } from 'react';
import { NodeType } from "@/types/NodeType";
import "./NodeContainer.scss"
import NodeComponent from "@/components/common/node-component/NodeComponent";

interface NodesContainerProps {  Show usages  yablonya
  nodes: NodeType[];
}

const NodesContainer: FC<NodesContainerProps> = ({ nodes }) => {  Show usages
  return (
    <div className="node-container">
      {nodes && nodes.map((node) => (
        <NodeComponent node={node} key={node.id} />
      ))}
    </div>
  );
};

export default NodesContainer;  Show usages  yablonya
```

Рис. 3 - Компонент NodesContainer

NodeComponent.tsx

```tsx
import React, { FC } from 'react';
import { NodeType } from "@/types/NodeType";
import "./NodeComponent.scss"

interface NodeComponentProps {  Show usages  yablonya
  node: NodeType;
}

const NodeComponent: FC<NodeComponentProps> = ({ node }) => {
  return (
    <div
      className="node-component"
      style={{
        left: node.xPosition,
        top: node.yPosition,
      }}
    >
      {node.content}
    </div>
  );
};

export default NodeComponent;  Show usages  yablonya
```

Рис. 4 - Компонент NodeComponent

Бачимо компонент MindMapPage, де відображаються дані про карту, а також внутрішній компонент NodesContainer, що інкапсулює логіку відображення вузлів. В компоненті NodesContainer застосунок проходиться по вузлах карти, тим самим провокуючи їх відображення. Компонент NodeComponent вже безпосередньо займається логікою відображення вузла, задаючи йому потрібні координати на карті.

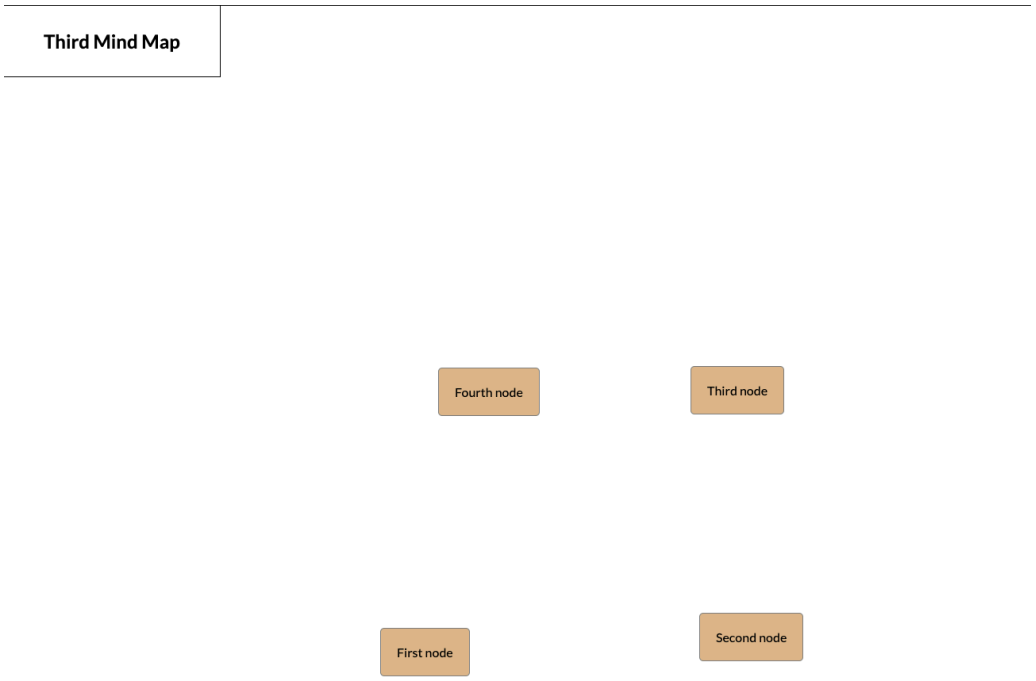

Рис. 4 - Результат відкриття сторінки MindMap для карти під номером 4

В результаті можна побачити, що вся деревоподібна структура карти пам’яті відображається коректно. А головним є те, що для відображення повноцінної карти не потрібно кожного разу вручну відображати дані про карту, перебирати вузли тощо. За допомогою шаблону Composite я додав логіку, при якій потрібно лише відобразити компонент MindMapPage, передавши в нього дані про карту. Завдяки тому що всі елементи, перелічені вище є React компонентами, у них за замовчуванням є метод “render”, що дозволяє зручно налаштовувати відображення даних, що мають деревоподібну структуру.

Весь оновлений код можна переглянути в даній директорії репозиторію web-частини проєкту:
https://github.com/yablonya/TRPZ_labs_WEB_yablonskyi_ia-24/tree/main/src

Висновок: У цій лабораторній роботі було реалізовано шаблон проєктування Composite для відображення структури mind map у frontend частині мого проекту Mind mapping software. Завдяки шаблону Composite вдалося створити гнучке та зручне рішення для роботи з деревоподібними структурами, що дозволяє однаково обробляти як окремі вузли, так і цілу карту, що є групою вузлів. Це спростило реалізацію ієрархічного компонента, зокрема забезпечило можливість динамічного додавання, видалення та редагування елементів mind map.

Реалізація Composite дозволила створити спільний інтерфейс для вузлів карти, який використовується як простими вузлами, так і їх контейнерами. Кожен елемент mind map представлено як окремий компонент, що має спільні методи для взаємодії. Це дало змогу досягти високої гнучкості та масштабованості, адже для розширення функціональності достатньо додати нові класи вузлів чи їх підтипів без зміни існуючої архітектури.

Застосування цього шаблону показало його практичну ефективність для побудови складних структур у реальних проєктах. Використовуючи Composite, я не лише спростив логіку відображення mind map, але й забезпечив зручність підтримки коду та додавання нових можливостей. Ця лабораторна робота продемонструвала, як правильно підібраний шаблон проєктування допомагає вирішувати конкретні завдання й покращувати якість програмного продукту.

ДОДАТКИ

Тестові дані, використані для налаштування інтерфейсу:


```
1: {
  mindMap: {
    id: 1,
    creatorId: 1,
    name: "First Mind Map",
    title: "My first mind map",
    creationDate: "2024-12-04T17:28:20.016+00:00"
  },
  nodes: [
    {
      id: 1,
      mindMapId: 1,
      type: "",
      content: "First node",
      priority: 0,
      category: "",
      creationDate: "2024-12-04T17:32:14.985+00:00",
      xPosition: 374,
      yPosition: 684
    },
    {
      id: 2,
      mindMapId: 1,
      type: "",
      content: "Second node",
      priority: 0,
      category: "",
      creationDate: "2024-12-04T17:32:14.985+00:00",
      xPosition: 876,
      yPosition: 765
    },
    {
      id: 3,
      mindMapId: 1,
      type: "",
      content: "Third node",
      priority: 0,
      category: "",
      creationDate: "2024-12-04T17:32:14.985+00:00",
      xPosition: 547,
      yPosition: 323
    }
  ],
  savedAt: "2024-12-04T17:28:20.016+00:00",
},
```

```
2: {
  mindMap: {
    id: 2,
    creatorId: 1,
    name: "Second Mind Map",
    title: "My second mind map",
    creationDate: "2024-12-04T17:28:20.016+00:00"
  },
  nodes: [
    {
      id: 4,
      mindMapId: 2,
      type: "",
      content: "First node",
      priority: 0,
      category: "",
      creationDate: "2024-12-04T17:32:14.985+00:00",
      xPosition: 678,
      yPosition: 234
    },
    {
      id: 5,
      mindMapId: 2,
      type: "",
      content: "Second node",
      priority: 0,
      category: "",
      creationDate: "2024-12-04T17:32:14.985+00:00",
      xPosition: 1023,
      yPosition: 234
    },
  ],
  savedAt: "2024-12-04T17:28:20.016+00:00",
},
```

```
3: {
  mindMap: {
    id: 3,
    creatorId: 1,
    name: "Third Mind Map",
    title: "My third mind map",
    creationDate: "2024-12-04T17:28:20.016+00:00"
  },
  nodes: [
    {
      id: 6,
      mindMapId: 3,
      type: "",
      content: "First node",
      priority: 0,
      category: "",
      creationDate: "2024-12-04T17:32:14.985+00:00",
      xPosition: 474,
      yPosition: 784
    },
    {
      id: 7,
      mindMapId: 3,
      type: "",
      content: "Second node",
      priority: 0,
      category: "",
      creationDate: "2024-12-04T17:32:14.985+00:00",
      xPosition: 876,
      yPosition: 765
    },
    {
      id: 8,
      mindMapId: 3,
      type: "",
      content: "Third node",
      priority: 0,
      category: "",
      creationDate: "2024-12-04T17:32:14.985+00:00",
      xPosition: 865,
      yPosition: 454
    },
```

```
    {
      id: 9,
      mindMapId: 3,
      type: "",
      content: "Fourth node",
      priority: 0,
      category: "",
      creationDate: "2024-12-04T17:32:14.985+00:00",
      xPosition: 547,
      yPosition: 456
    }
  ],
  savedAt: "2024-12-04T17:28:20.016+00:00",
},
4: {
  mindMap: {
    id: 4,
    creatorId: 1,
    name: "Fourth Mind Map",
    title: "My fourth mind map",
    creationDate: "2024-12-04T17:28:20.016+00:00"
  },
  nodes: [
    {
      id: 10,
      mindMapId: 4,
      type: "",
      content: "First node",
      priority: 0,
      category: "",
      creationDate: "2024-12-04T17:32:14.985+00:00",
      xPosition: 555,
      yPosition: 555
    }
  ],
  savedAt: "2024-12-04T17:28:20.016+00:00",
}
```