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 №8

з дисципліни «Технології розроблення програмного забезпечення»

Виконала:
студентка групи ІА-23
Проценко. В. І.

Перевірив:
Мягкий М. Ю

Київ 2024

Зміст

Тема: Шаблони «COMPOSITE», «FLYWEIGHT», «INTERPRETER», «VISITOR».

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. Короткі теоретичні відомості

Шаблони роботи з БД при розробці корпоративних додатків:

- Шаблон «Active record» (Активний запис)
  Один об'єкт управляє і даними, і поведінкою. Більшість цих даних постійні і їх треба зберігати в БД. Цей патерн використовує найбільш очевидний підхід - зберігання логіки доступу до даних в об'єкті сутності. Об'єкт є «обгорткою» одного рядка з БД або подання, включає в себе доступ до БД і логіку поводження з даними. Шаблон використовується зокрема в в простих додатка. Зі збільшенням кількості різноманітних запитів зростає складність класу даних і як правило логіка запитів виноситься в окремий об'єкт.
- Шаблон «Table Data Gateway» (Шлюз до даних)
  В даному випадку окремий клас для кожного класу даних займається взаємодією з базою даних. Він зберігає в собі логіку всіх запитів. Такий підхід більш гнучкий і тестований, і дозволяє утримувати окремо дані і окремо взаємодію з базою даних. Однак для кожного шлюзу до даних код взаємодії як правило повторюється (з'єднання з базою, формування команди джерела даних і т.п.), тому їх часто абстрагують в окремий базовий клас для шлюзів. Ще такий шаблон часто називають репозиторій.
- Шаблон «Data mapping» (Відображення даних)
  Вкрай часто виникають проблеми перетворення об'єкта даних в рядок реляційного джерела. Така ж проблема може виникати при необхідності передачі об'єкта по мережі. Для її вирішення прийнято створювати окремі об'єкти (або методи усередині класу даних) для перетворення об'єктів даних в дані, що приймаються джерелом виду. Ця процедура носить назву відображення (mapping). Під час відображення виправляються всі невідповідності типів даних між об'єктом даних і джерелом даних.

Типовий маппер - з об'єкта даних в таблицю реляційного джерела даних. Він несе знання про назви колонок в таблиці і відповідні властивості об'єкта даних, які відповідають цим колонкам.

Розглянемо декілька шаблонів:

1. Шаблон «Composite».
   Шаблон використовується для складання об'єктів в деревоподібну структуру для подання ієрархій типу «частина цілого». Даний шаблон дозволяє уніфіковано обробляти як поодинокі об'єкти, так і об'єкти з вкладеністю. Простим прикладом може служити складання компонентів всередині звичайної форми. Форма може містити дочірні елементи (поля для введення тексту, цифр, написи, малюнки тощо); дочірні елементи можуть в свою чергу містити інші елементи. Наприклад, при виконанні операції розтягування форми необхідно, щоб вся ієрархія розтягнулася відповідним чином. В такому випадку форма розглядається як композитний об'єкт і операція розтягування застосовується до всіх дочірніх елементів рекурсивно. Даний шаблон зручно використовувати при необхідності подання та обробки ієрархій об'єктів.
   + Спрощує архітектуру клієнта при роботі зі складним деревом компонентів.
   + Полегшує додавання нових видів компонентів.
   - Створює занадто загальний дизайн класів.

2. Шаблон «Flyweight».
   Шаблон використовується для зменшення кількості об'єктів в додатку шляхом поділу цих об'єктів між ділянками додатку. Flyweight являє собою поділюваний об'єкт.
   Дуже важливою є концепція «внутрішнього» і «зовнішнього» станів. Внутрішній стан відображає дані, характерні саме поділюваному об'єкту (наприклад, код букви); зовнішній стан несе інформацію про його застосування в додатку (наприклад, рядок і стовпчик). Внутрішній стан зберігається в самому поділюваному об'єкті, зовнішній - в об'єктах додатку (контексту використання поділюваного об'єкта). Даний шаблон дуже добре застосовувати у випадках, коли використовується безліч однакових об'єктів (наприклад, графічних примітивів).
   + Заощаджує оперативну пам’ять.
   - Витрачає процесорний час на пошук/обчислення контексту.
   - Ускладнює код програми внаслідок введення безлічі додаткових класів.

3. Шаблон «Interpreter».

   Даний шаблон використовується для подання граматики і інтерпретатора для вибраної мови (наприклад, скриптової). Граматика мови представлена термінальними і нетермінальними символами, кожен з яких інтерпретується в контексті використання. Клієнт передає контекст і сформовану пропозицію в використовувану мову в термінах абстрактного синтаксичного дерева (деревоподібна структура, яка однозначно визначає ієрархію виклику підвиразів), кожен вираз інтерпретується окремо з використанням контексту. У разі наявності дочірніх виразів, батьківський вираз інтерпретує спочатку дочірні (рекурсивно), а потім обчислює результат власної операції. Шаблон зручно використовувати в разі невеликої граматики (інакше розростеться кількість використовуваних класів) і відносно простого контексту (без взаємозалежностей і т.п.). Даний шаблон визначає базовий каркас інтерпретатора, який за допомогою рекурсії повертає результат обчислення пропозиції на основі результатів окремих елементів. При використанні даного шаблону дуже легко реалізовується і розширюється граматика, а також додаються нові способи інтерпретації виразів.

   + Граматику стає легко розширювати та змінювати, реалізації класів, що описують вузли абстрактного синтаксичного дерева схожі (легко кодуються).

   + Можна легко змінювати спосіб обчислення виразів.

   - Сопровождение грамматики с большим числом правил затруднительно.

4. Шаблон «Visitor».

   Шаблон відвідувач дозволяє вказувати операції над елементами без зміни структури конкретних елементів. Таким чином вкрай зручно додавати нові операції, проте дуже важко додавати нові елементи в ієрархію (необхідно додавати відповідні методи для обробки їх відвідувань в кожного відвідувача). Даний шаблон дозволяє групувати однотипні операції, що застосовуються над різнотипними об'єктами.

   + Спрощує додавання операцій, працюючих зі складними структурами об’єктів.

   + Об’єднує споріднені операції в одному класі.

   + Відвідувач може накопичувати стан при обході структури елементів.

   - Патерн невиправданий, якщо ієрархія елементів часто змінюється.

   - Може призвести до порушення інкапсуляції елементів.

2. Хід роботи

Пригадаємо обрану індивідуальну тему для виконання лабораторних робіт.

..17 System activity monitor (iterator, command, abstract factory, bridge, visitor, SOA)

Монітор активності системи повинен зберігати і запам'ятовувати статистику використовуваних компонентів системи, включаючи навантаження на процесор, обсяг займаної оперативної пам'яті, натискання клавіш на клавіатурі, дії миші (переміщення, натискання), відкриття вікон і зміна вікон; будувати звіти про використання комп'ютера за різними критеріями (% часу перебування у веб-браузері, середнє навантаження на процесор по годинах, середній час роботи комп'ютера по днях і т.д.); правильно поводитися з «простоюванням» системи – відсутністю користувача.

2.1. Реалізація класів відповідно до обраної теми.

Реалізуємо функціонал створення різноманітних типів звітів у класі Report та переносимо логіку підрахунку часу використання комп’ютера до окремого монітору ComputerUsage.

Оновлюємо клас основного монітору ActivityMonitor для покращення вигляду користувацького інтерфейсу та у загальному структури даного класу.

Описану вище частину функціональності можна переглянути у спільному репозиторії, у папці system-activity-monitor за посиланням: https://github.com/protsenkoveronika/TRPZ/tree/new-test-branch/system-activity-monitor

2.2. Застосування одного з розглянутих шаблонів.

У системі моніторингу активності комп’ютера шаблон "Visitor" використовується для створення звітів різного формату, що сприяє відокремленню логіки отримання та генерації даних від їх представлення. Це дозволяє легко додавати нові формати звітів без змін у основному коді генерації. У коді це реалізовано через абстракційні класи, як базовий ReportVisitor, а також конкретні реалізації, такі як TextReportVisitor та JSONReportVisitor, які визначають, як саме форматувати та представляти різні типи звітів.

```python
class ReportVisitor(ABC):
    @abstractmethod
    def visit_daily_report(self, report, date, report_type):
        pass

    @abstractmethod
    def visit_periodic_report(self, report, start_date, end_date, report_type):
        pass


class JSONReportVisitor(ReportVisitor):
    def visit_daily_report(self, report, date, report_type):
        data = report.generate_daily_report(date, report_type)
        if not data:
            return None
        return json.dumps({"date": date, "data": data}, indent=4)

    def visit_periodic_report(self, report, start_date, end_date, report_type):
        data = report.generate_periodic_report(start_date, end_date, report_type)
        if not data:
            return None
        return json.dumps({"start_date": start_date, "end_date": end_date, "data": data}, indent=4)

class TextReportVisitor(ReportVisitor):
    def visit_daily_report(self, report, date, report_type):
        data = report.generate_daily_report(date, report_type)
        if not data:
            return None
        return f"Daily Report ({date}):\n{self.format_data(data)}"

    def visit_periodic_report(self, report, start_date, end_date, report_type):
        data = report.generate_periodic_report(start_date, end_date, report_type)
        if not data:
            return None
        return f"Periodic Report ({start_date} - {end_date}):\n{self.format_data(data)}"

    def format_data(self, data):
        return "\n".join(f"{key}: {value}" for key, value in data.items()) if data else "No data available."
```

Рисунок 1.1 - Абстрактний клас ReportVisitor та його реалізації JSONReportVisitor і TextReportVisitor

TextReportVisitor — конкретна реалізація ReportVisitor, яка форматує отримані дані для звіту у вигляді тексту. JSONReportVisitor — інша конкретна реалізація ReportVisitor, що форматує звіт у форматі JSON. Вони розширюють базовий шаблон і надають конкретну реалізацію методів.

Направимо реалізацію команд із попередньої частини роботи для відповідності інтеграції патерну Command, який ми використовували для керування створенням звіту, із Visitor.

```python
class GenerateDailyReportCommand(Command):
    def __init__(self, report, date, report_type, visitor):
        self.report = report
        self.date = date
        self.report_type = report_type
        self.visitor = visitor

    def execute(self):
        return self.visitor.visit_daily_report(self.report, self.date, self.report_type)

class GeneratePeriodicReportCommand(Command):
    def __init__(self, report, start_date, end_date, report_type, visitor):
        self.report = report
        self.start_date = start_date
        self.end_date = end_date
        self.report_type = report_type
        self.visitor = visitor

    def execute(self):
        return self.visitor.visit_periodic_report(self.report, self.start_date, self.end_date, self.report_type)
```

Рисунок 1.2 - Оновлена версія класів GenerateDailyReportCommand та GeneratePeriodicReportCommand, що підтримують патерн Відвідувач

Таким чином, використання патерну "Відвідувач" дозволяє додавати нові операції для об'єктів, не змінюючи їх код. Це дозволяє підтримувати відкритість для розширень та закритість для змін, забезпечуючи гнучкість та чистоту коду.

3. Висновок

У ході даної лабораторної роботи було реалізовано частину функціональності системи моніторингу активності, зокрема завершено роботу над функціональністю створення та висвітлення звітів у різних форматах. Ми ознайомились із кількома шаблонами які використовують у програмуванні, у тому числі і шаблоном "Visitor", який було використано при реалізації даної частини проекту.