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

Технології розроблення програмного забезпечення

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

Варіант 30

Виконав
студент групи ІА-13
Якін С. О.

Перевірив:
Мягкий М.Ю.

Київ 2023

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

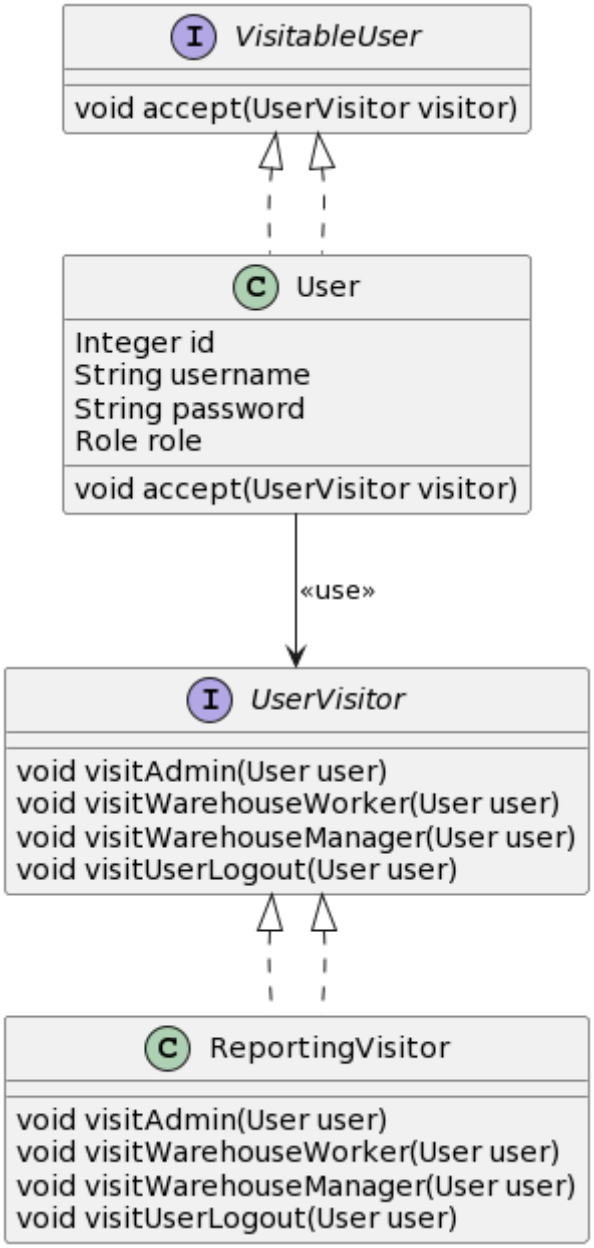
Завдання:

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.

Хід роботи:

Тема: Система складського обліку виробництва

Згідно мого варіанту, було виконано шаблон проектування «Visitor». Цей шаблон дозволяє додавати нову логіку в клас, при цьому не змінюючи код самого класу. Код додається у імплементацію візітора, де вже відбувається вся логіка обробки, при цьому в основний клас додається метод accept(), який дозволяє передати основний клас у візітор для подальших дій.



Структура класів шаблону «Відвідувач» та класів, що його використовують

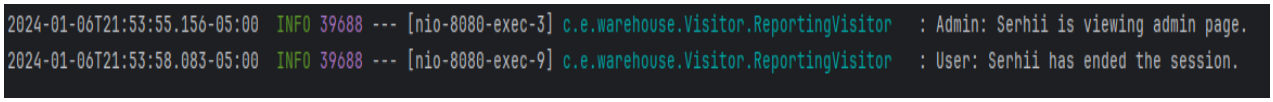
Діаграма відображає взаємодію між компонентами в рамках шаблону проектування "Відвідувач" (Visitor) у контексті системи. Ця діаграма є ключовою для розуміння того, як можна ефективно реалізувати розширення функціональності об'єктів без зміни їх внутрішньої структури.

На діаграмі представлено чотири основні компоненти: інтерфейс `UserVisitor`, клас `ReportingVisitor`, інтерфейс `VisitableUser` та клас `User`. Інтерфейс `UserVisitor` визначає набір методів для відвідування різних типів користувачів, таких як адміністратори, робітники складу та керівники. Ці методи включають `visitAdmin`, `visitWarehouseWorker`, `visitWarehouseManager` та

`visitUserLogout`, кожен з яких призначений для взаємодії з конкретним типом користувача.

Клас `ReportingVisitor` реалізує інтерфейс `UserVisitor`, надаючи конкретну реалізацію для кожного з методів відвідування. Це означає, що `ReportingVisitor` може виконувати специфічні дії, залежно від типу користувача, якого він відвідує. Наприклад, при відвідуванні адміністратора, `ReportingVisitor` може логувати інформацію про діяльність адміністратора.

Інтерфейс `VisitableUser` визначає один метод `accept`, який приймає об'єкт `UserVisitor`. Цей метод дозволяє об'єктам, які реалізують `VisitableUser`, приймати різні види відвідувачів і взаємодіяти з ними. Клас `User`, який реалізує `VisitableUser`, використовує метод `accept` для делегування виконання операцій відповідному відвідувачу, залежно від своєї ролі в системі. Загалом, ця діаграма демонструє, як шаблон "Відвідувач" може бути використаний для реалізації гнучкої системи взаємодії між різними типами користувачів та операціями, які потрібно виконати. Це дозволяє легко додавати нові види відвідувачів та операцій без необхідності зміни існуючих класів користувачів, що сприяє гнучкості та масштабованості системи.



Результат використання шаблону проектування «Відвідувач»

Як ми бачимо на рисунку, ми за допомогою LoggerInfo отримуємо повідомлення у системі про авторизацію певного користувача та його кінець сесії.

Код:

VisitableUser:

```
public interface VisitableUser {
    1 usage   1 implementation   serrb
    void accept(UserVisitor visitor);
}
```

Class User (accept method):

```
    @Override
    public void accept(UserVisitor visitor) {
        switch (this.role) {
            case ADMIN:
                visitor.visitAdmin( user: this);
                break;
            case WORKER:
                visitor.visitWarehouseWorker( user: this);
                break;
            case BOSS:
                visitor.visitWarehouseManager( user: this);
                break;
            default:
                throw new RuntimeException("Unknown role: " + this.role);
        }
    }
}
```

UserVisitor:

```
public interface UserVisitor {
    1 usage   1 implementation   serrb
    void visitAdmin(User user);
    1 usage   1 implementation   serrb
    void visitWarehouseWorker(User user);
    1 usage   1 implementation   serrb
    void visitWarehouseManager(User user);
    1 usage   1 implementation   serrb
    void visitUserLogout(User user);
}
```

ReportingVisitor:

```java
public class ReportingVisitor implements UserVisitor {

    4 usages
    private static final Logger logger = LoggerFactory.getLogger(ReportingVisitor.class);

    1 usage   serrb *
    @Override
    public void visitAdmin(User user) {

        logger.info("User: " + user.getUsername() + " is viewing admin page.");
    }

    1 usage   serrb
    @Override
    public void visitWarehouseWorker(User user) {
        logger.info("User: " + user.getUsername() + " is managing warehouse operations.");
    }

    1 usage   serrb
    @Override
    public void visitWarehouseManager(User user) {
        logger.info("User: " + user.getUsername() + " is working in warehouse system.");
    }

    1 usage   serrb *
    @Override
    public void visitUserLogout(User user) {

        logger.info("User: " + user.getUsername() + " has ended the session.");
    }
}
```

На рисунку показано використання саме логера для викидання повідомлення у нашу консоль.

Висновок: на цій лабораторній роботі я познайомився з такими паттернами, як «Composite», «Flyweight», «Interpreter», «Visitor», а також розібрався у принципі їх роботи та сфері використання. На практиці спроектував та реалізував паттерн «Visitor», згідно зі своїм варіантом, у своєму проекті.