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

Технології розроблення програмного забезпечення

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

Виконала
студентка групи ІА-14:

Логінова І.С.

Перевірив:

Мягкий М.Ю.

Київ 2023

Тема: 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

Варіант:

10. VCS all-in-one (iterator, adapter, factory method, facade, visitor, p2p)

Клієнт для всіх систем контролю версій повинен підтримувати основні команди і дії (commit, update, push, pull, fetch, list, log, patch, branch, merge, tag) для 3-х основних систем управління версіями (svn, git, mercurial), а також мати можливість вести реєстр репозиторіїв (і їх типів) і відображати дерева фіксації графічно

Хід роботи

Паттерн Iterator

Ітератор — це поведінковий патерн проектування, що дає змогу послідовно обходити елементи складових об’єктів, не розкриваючи їхньої внутрішньої організації.

```python
class CommandIterator:
    def __init__(self, client_socket):
        self.client_socket = client_socket
        self.commands = []
        self.index = 0

    1 usage
    def add_command(self, command):
        self.commands.append(command)

    def __iter__(self):
        return self

    def __next__(self):
        if self.index < len(self.commands):
            command = self.commands[self.index]
            self.index += 1
            return command
        else:
            raise StopIteration
```

```python
def process_vcs_commands(client_socket, facade, vcs_type, repo_path, db):
    client_socket.sendall(f"[{vcs_type} [{repo_path}]]".encode('utf-8'))
    command_iterator = CommandIterator(client_socket)
    command_executor = Executor()

    while True:
        command = client_socket.recv(1024).decode('utf-8').strip()
        if command.lower() == "back":
            handle_peer(client_socket, db)
        else:
            command_iterator.add_command(command)
            for cmd in command_iterator:
                command_executor.visit(client_socket, vcs_type, cmd, facade, repo_path)
```

Паттерн Iterator використовується для ітерації через команди, які вводить користувач.

Клас “CommandIterator” реалізує інтерфейс ітератора. У конструкторі ініціалізується список куди будуть додаватися команди та зміну “index”, яка вказує на поточний індекс команди для ітерації.

Дандер метод “__iter__” повертає об’єкт(сам себе), оскільки він вже є ітератором.
Дандер метод “__next__” повертає наступну команду за кожним викликом та збільшує індекс. Якщо всі команди вже оброблено, ви викидає виключення.

У функції “process_vcs_commands” створюється єкземпляр ітератора для обробки команд в циклі.

Команда поступає від клієнта та додається до ітератора за допомогою commnd_iterator.add_command(command).

Потім ітеруємося по ітератору за допомогою циклу та викликаємо метод для обробки кожної команди.

Висновок: при виконанні цієї лабораторної роботи я реалізувала частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей із застосуванням паттерну Iterator при реалізації програми.