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»»

Виконала
студентка групи ІА-14:

Логінова І.С.

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 і їх взаємодій для досягнення конкретних функціональних можливостей.

3. Застосування одного з розглянутих шаблонів при реалізації програми.

Варіант:

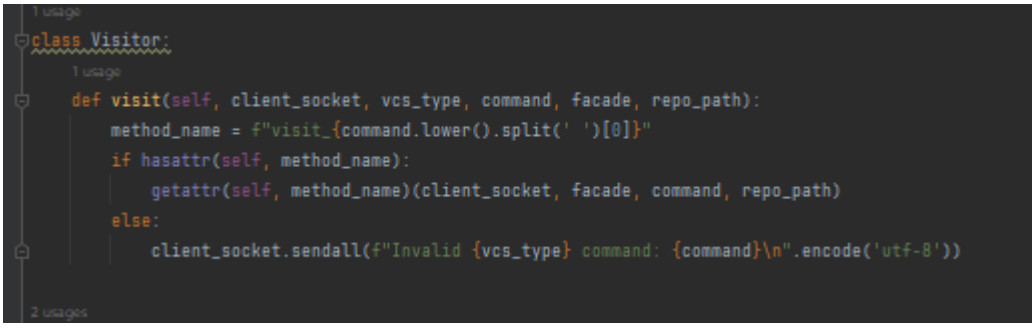
10. VCS all-in-one (iterator, adapter, factory method, facade, visitor, p2p)

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 Відвідувач (Visitor)

Відвідувач — це поведінковий патерн проектування, що дає змогу додавати до програми нові операції, не змінюючи класи об’єктів, над якими ці операції можуть виконуватися.



```
1 usage
class Visitor:
    1 usage
    def visit(self, client_socket, vcs_type, command, facade, repo_path):
        method_name = f"visit_{command.lower().split(' ')[0]}"
        if hasattr(self, method_name):
            getattr(self, method_name)(client_socket, facade, command, repo_path)
        else:
            client_socket.sendall(f"Invalid {vcs_type} command: {command}\n".encode('utf-8'))

2 usages
```

```python
class Executor(Visitor):

    def visit_backup(self, client_socket, facade, command, repo_path):
        if not os.path.exists(backup_directory):
            os.makedirs(backup_directory)
        repo_folder_name = os.path.basename(repo_path)
        backup_path = os.path.join(backup_directory, f"{repo_folder_name}_backup")

        try:
            shutil.copytree(repo_path, backup_path)
            client_socket.sendall(b"Backup completed successfully.")
        except Exception as e:
            client_socket.sendall(f"Error during backup: {e}".encode('utf-8'))

    def visit_backup_load(self,client_socket, facade, command, repo_path):
        try:
            _, backup_name, local_path = command.split(maxsplit=2)
            backup_path = os.path.join(backup_directory, backup_name)

            if os.path.exists(backup_path):
                if not os.path.exists(local_path):
                    os.makedirs(local_path)
                for item in os.listdir(backup_path):
                    source = os.path.join(backup_path, item)
                    destination = os.path.join(local_path, item)
                    if os.path.isdir(source):
                        shutil.copytree(source, destination)
                    else:
                        shutil.copy2(source, destination)

                client_socket.sendall(b"Backup loaded successfully.")
            else:
                client_socket.sendall(f"Backup '{backup_name}' not found on the server.\n".encode('utf-8'))
        except Exception as e:
            client_socket.sendall(f"Error loading backup: {str(e)}\n".encode('utf-8'))

    def visit_commit(self, client_socket,facade,command,repo_path):
        try:
            _, comment = command.split(maxsplit=1)
            facade.commit_changes(repo_path, comment)
        except Exception as e:
            client_socket.sendall(f"Error executing 'commit' command: {str(e)}\n".encode('utf-8'))

    def visit_update(self, client_socket,facade,command,repo_path):
        try:
            facade.update_repository(repo_path)
        except Exception as e:
            client_socket.sendall(f"Error executing 'update' command: {str(e)}\n".encode('utf-8'))

    def visit_push(self, client_socket,facade,command,repo_path):
        try:
            facade.push_changes(repo_path)
        except Exception as e:
            client_socket.sendall(f"Error executing 'push' command: {str(e)}\n".encode('utf-8'))

    def visit_help(self, client_socket,facade,command,repo_path):
        help(client_socket)
```

```python
    def visit_init(self, client_socket, facade, command, repo_path):
        try:
            _, repository_name = command.split(maxsplit=1)
            facade.initialize_repository(repository_name, repo_path)
        except Exception as e:
            client_socket.sendall(f"Error executing 'init' command: {str(e)}\n".encode('utf-8'))

    def visit_log(self, client_socket, facade, command, repo_path):
        try:
            facade.view_commit_history(repo_path)
        except Exception as e:
            client_socket.sendall(f"Error executing 'log' command: {str(e)}\n".encode('utf-8'))

    def visit_status(self, client_socket, facade, command, repo_path):
        try:
            facade.view_repository_status(repo_path)
        except Exception as e:
            client_socket.sendall(f"Error executing 'status' command: {str(e)}\n".encode('utf-8'))

    def visit_add(self, client_socket, facade, command, repo_path):
        try:
            _, files_input = command.split(maxsplit=1)
            files = [file.strip() for file in files_input.split(',')]
            facade.add_files(repo_path, files)
        except Exception as e:
            client_socket.sendall(f"Error executing 'add' command: {str(e)}\n".encode('utf-8'))
```

```
def visit_add_all(self, client_socket, facade, command, repo_path):
    try:
        facade.add_all_changes(repo_path)
    except Exception as e:
        client_socket.sendall(f"Error executing 'add_all' command: {str(e)}\n".encode('utf-8'))

def visit_patch(self, client_socket, facade, command, repo_path):
    try:
        _, patch_file_path = command.split(maxsplit=1)
        facade.apply_patch(repo_path, patch_file_path)
    except Exception as e:
        client_socket.sendall(f"Error executing 'patch' command: {str(e)}\n".encode('utf-8'))

def visit_branch(self, client_socket, facade, command, repo_path):
    try:
        _, branch_name = command.split(maxsplit=1)
        facade.create_branch(repo_path, branch_name)
    except Exception as e:
        client_socket.sendall(f"Error executing 'branch' command: {str(e)}\n".encode('utf-8'))

def visit_merge(self, client_socket, facade, command, repo_path):
    try:
        _, branch_name = command.split(maxsplit=1)
        facade.merge_branch(repo_path, branch_name)
    except Exception as e:
        client_socket.sendall(f"Error executing 'merge' command: {str(e)}\n".encode('utf-8'))

def visit_tag(self, client_socket, facade, command, repo_path):
    try:
        _, tag_name = command.split(maxsplit=1)
        facade.create_tag(repo_path, tag_name)
    except Exception as e:
        client_socket.sendall(f"Error executing 'tag' command: {str(e)}\n".encode('utf-8'))

def visit_list(self, client_socket, facade, command, repo_path):
    try:
        facade.list(repo_path)
    except Exception as e:
        client_socket.sendall(f"Error executing 'list' command: {str(e)}\n".encode('utf-8'))
```

Паттерн Visitor дозволяє визначити нові операції над об'єктами, не змінюючи їхнього класу. У нашому випадку, операції визначаються методами в класі Executor, який є підкласом Visitor.

Основна ідея полягає в тому, що кожна команда (наприклад, backup, commit, update, і т.д.) має відповідний метод у класі Executor, який обробляє цю команду. При цьому використовується метод visit у класі Visitor, який визиває потрібний метод згідно з переданою командою.

Це дозволяє динамічно визначати, який метод потрібно викликати залежно від команди. Паттерн Visitor особливо корисний, коли є багато різних операцій, які можуть бути виконані над об'єктами, і потрібно дотримуватися принципу open/closed principle - можливість додавати нові операції без зміни існуючого коду.

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 Visitor при реалізації програми.