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 №5

Технології розроблення програмного забезпечення

«Шаблони «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»»

Виконала
студентка групи ІА-14:

Логінова І.С.

Перевірив:

Мягкий М.Ю.

Київ 2023

Тема: Шаблони «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE».

Завдання:

1. Ознайомитися з короткими теоретичними відомостями.

2. Реалізувати частину функціоналу робочої програми у вигляді класів та 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 Адаптер (Wrapper, Обгортка, Adapter)

Адаптер - це структурний патерн проєктування, який дає змогу об'єктам із несумісними інтерфейсами працювати разом.

```python
class VCSAdapter(VCSInterface):
    def __init__(self, client_socket, database, vcs_type):
        self.database = database
        self.vcs_type = vcs_type
        self.client_socket = client_socket

        if self.vcs_type == "git":
            self.vcs_impl = GIT(client_socket, database)
        elif self.vcs_type == "mercurial":
            self.vcs_impl = Mercurial(client_socket, database)
        elif self.vcs_type == "svn":
            self.vcs_impl = SVN(client_socket, database)
        else:
            raise ValueError(f"Unsupported VCS Type: {vcs_type}")
```

```python
    def __getattribute__(self, item):
        return object.__getattribute__(self, item)

    1 usage (1 dynamic)
    def commit(self, repo_path, comment):
        self.vcs_impl.commit(repo_path, comment)

    1 usage (1 dynamic)
    def update(self, repo_path):
        self.vcs_impl.update(repo_path)

    1 usage (1 dynamic)
    def push(self, repo_path):
        self.vcs_impl.push(repo_path)

    1 usage (1 dynamic)
    def init_repo(self, repository_name, repo_path):
        self.vcs_impl.init_repo(repository_name, repo_path)

    1 usage (1 dynamic)
    def log(self, repo_path):
        self.vcs_impl.log(repo_path)

    2 usages (2 dynamic)
    def status(self, repo_path):
        self.vcs_impl.status(repo_path)

    1 usage (1 dynamic)
    def add(self, repo_path, files):
        self.vcs_impl.add(repo_path, files)

    1 usage (1 dynamic)
    def add_all(self, repo_path):
        self.vcs_impl.add_all(repo_path)
```

Адаптер використовується для того, щоб забезпечити єдинообразний інтерфейс для взаємодії з різними системами контролю версій (Git, Mercurial, SVN). Є різні класи, які реалізують VCSInterface, але кожен з них представляє різний тип VCS. Адаптер дозволяє цим різним класам співпрацювати через єдиний інтерфейс.

Переваги:

1) Адаптер дозволяє всім різним типам VCS (Git, Mercurial, SVN) виглядати як один і той самий інтерфейс VCSInterface. Це означає, що незалежно від того, яка конкретна VCS використовується, програма може спілкуватися з нею за допомогою тих самих методів.
2) Якщо потрібно буде додати новий тип VCS, не потрібно змінювати всю основну логіку програми. Просто потрібно створити новий клас, який реалізує VCSInterface, і адаптувати його за допомогою адаптера.

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 Adapter при реалізації програми.