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

Варіант 10

Виконала
студентка групи ІА – 13:
Луценко Юлія Сергіївна

Перевірив:
М’який Михайло

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:

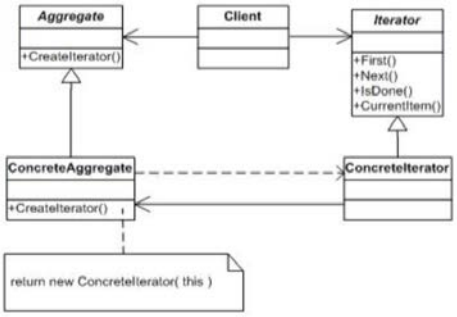
10. VCS all-in-one (iterator, adapter, factory method, facade, visitor, p2p)
Клієнт для всіх систем контролю версій повинен підтримувати основні команди і дії (commit, update, push, pull, fetch, list, log, patch, branch, merge, tag) для 3-х основних систем управління версіями (svn, git, mercurial), а також мати можливість вести реєстр репозиторіїв (і їх типів) і відображати дерева фіксації графічно

Хід роботи

Шаблон Ітератор

«Iterator» (Ітератор) являє собою шаблон реалізації об'єкта доступу до набору (колекції, агрегату) елементів без розкриття внутрішніх механізмів реалізації. Ітератор виносить функціональність перебору колекції елементів з самої колекції, таким чином досягається розподіл обов'язків: колекція відповідає за зберігання даних, ітератор - за прохід по колекції.

Шаблон «Iterator»



Ідея патерна Ітератор полягає в тому, щоб винести поведінку обходу колекції з самої колекції в окремий клас

```
from Iterator import RepositoryIteratorInterface

2 usages
class RepositoryIterator(RepositoryIteratorInterface):
    def __init__(self, repositories):
        self.repositories = repositories
        self.index = 0

    def first(self):
        self.index = 0

    1 usage
    def next(self):
        self.index += 1

    2 usages
    def isDone(self):
        return self.index >= len(self.repositories)

    1 usage
    def currentItem(self):
        if self.isDone():
            return None
        return self.repositories[self.index]
```

```
    def __iter__(self):
        return self

    def __next__(self):
        if self.isDone():
            raise StopIteration
        item = self.currentItem()
        self.next()
        return item
```

```python
def show_active_repositories():
    print(Fore.BLUE + "Active Repositories:")
    repo = Repository(connection)
    iterator = repo.iterate()

    for repository in iterator:
        print(f"Repository: №{repository.id} NAME:{repository.name} TYPE:{repository.vcs_type} URL:{repository.url}")
```

```python
from abc import ABC, abstractmethod

2 usages
class RepositoryIteratorInterface(ABC):
    @abstractmethod
    def first(self):
        pass

    @abstractmethod
    def next(self):
        pass

    @abstractmethod
    def isDone(self):
        pass

    @abstractmethod
    def currentItem(self):
        pass
```

```
Choose a Version Control System (VCS):
1. Git
2. Mercurial
3. SVN
4. Show Active Repositories
5. Exit
Enter the number of your choice: 4
Active Repositories:
Repository: №1 NAME:repo TYPE:Git URL:D:\ТПР3\repo
```

Висновок:у цій лабораторній було реалізовано патерн ітератор,що дало змогу нам проходитись по даним наших репозиторіїв.Але,на мою думку цей патерн тут не дуже доречний та потрібний,адже в нас не складна структура даних і можна обійтись звичайним циклом.