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 №6

ШАБЛОНИ «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator»

Варіант 10

Виконала
студентка групи ІА – 13:
Луценко Юлія Сергіївна

Перевірив:
Мягкий Михайло

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

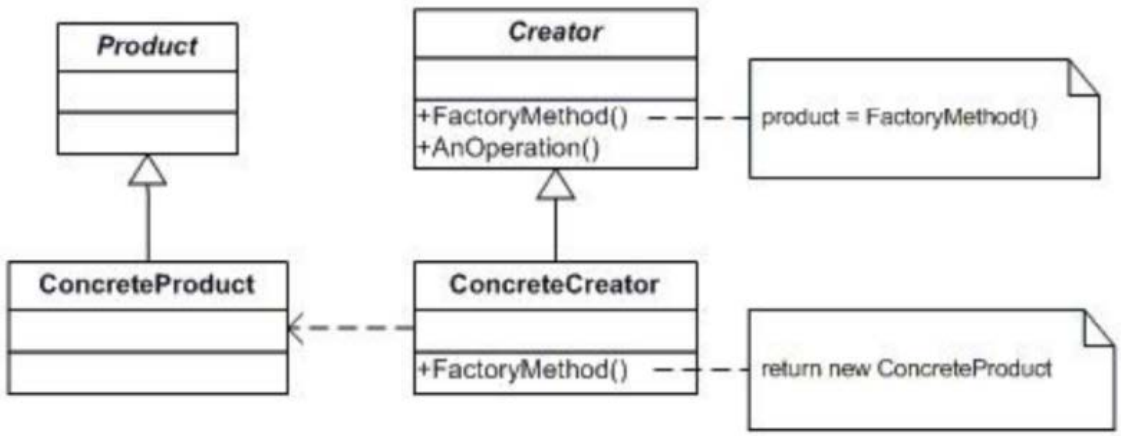
Хід роботи

Паттерн Фабричний метод(Віртуальний конструктор, Factory Method)

Фабричний метод - це породжувальний патерн проектування, який визначає загальний інтерфейс для створення об'єктів у суперкласі, дозволяючи підкласам змінювати тип створюваних об'єктів.

Шаблон «Factory Method»

Структура:



Реалізація фабричного методу

```python
from src.GitVersionControl import GitVersionControl
from src.MercurialVersionControl import MercurialVersionControl
from src.SVNVersionControl import SVNVersionControl


2 usages  new *
class VersionControlFactory:
    1 usage  new *
    @staticmethod
    def create_version_control_system(connection, vcs_type):
        if vcs_type == "Git":
            return GitVersionControl(connection)
        elif vcs_type == "Mercurial":
            return MercurialVersionControl(connection)
        elif vcs_type == "SVN":
            return SVNVersionControl(connection)
        else:
            raise ValueError(f"Unsupported VCS type: {vcs_type}")
```

Використання патерну у main.py

```python
if choice in ["1", "2", "3"]:
    vcs_type = "Git" if choice == "1" else "Mercurial" if choice == "2" else "SVN"
    repo_name = input(f"Enter the path to your {vcs_type} repository: ")
    repo_name = convert_to_absolute_path(repo_name)
    version_control = VersionControlFactory.create_version_control_system(connection, vcs_type)
    process_vcs_commands(version_control, vcs_type, repo_name)
elif choice == "5":
    print("Exiting...")
    break
else:
    print("Invalid choice. Please enter a valid option.")
```

Спільний інтерфейс для версій контролю

```python
from abc import ABC, abstractmethod
from datetime import date


6 usages  yuliiaaaaa +1
class IVersionControlSystem(ABC):
    5 usages (5 dynamic)  Yulia +1
    @abstractmethod
    def commit(self, path, file_name, message):
        pass

    1 usage (1 dynamic)  yuliiaaaaa
    @abstractmethod
    def watch_history(self, repo_name):
        pass

    1 usage (1 dynamic)  yuliiaaaaa
    @abstractmethod
    def initialize_repository(self, repo_directory,vcs_type):
        pass
```

Реалізація MercurialVersionCntrol

```python
class MercurialVersionControl(IVersionControlSystem):
    yuliiaaaaa
    def __init__(self, connection):
        self.connection = connection
        self.ui = ui.ui()

    5 usages (5 dynamic)  yuliiaaaaa
    def commit(self, repo_name, file_name, repo_id, message, commit_date):
        with self.repo.lock():
            # Add the file to the commit
            self.repo.add(file_name)

            # Commit the changes
            try:
                commit_user = f"{repo_id} <{repo_id}@example.com>"
                commit_date_str = commit_date.strftime('%Y-%m-%d %H:%M:%S %z')
                self.repo.commit(message, user=commit_user, date=commit_date_str)

                print(f"Committed changes for {file_name} with message: {message}")
            except hg.error.Abort as e:
                print(f"Error committing changes: {e}")

    1 usage (1 dynamic)  yuliiaaaaa
    def watch_history(self, repo_name):
        try:
            with self.repo.lock():
                # Display the commit history
                for rev in reversed(list(self.repo)):
```

Реалізація GitVersionCntrol

```python
class GitVersionControl(IVersionControlSystem):
    yuliiaaaaa +1
    def __init__(self, connection):
        self.connection = connection
        self.repo = None

    1 usage (1 dynamic)  yuliiaaaaa +1
    def initialize_repository(self, repo_directory, vcs_type):
        git_dir = os.path.join(repo_directory, ".git")
        repo_name = os.path.basename(repo_directory)
        if os.path.exists(git_dir) and os.path.isdir(git_dir):
            print(f"Git repository already exists in: {repo_directory}")
            self.repo = Repo(repo_directory)
        else:
            try:
                self.repo = Repo.init(repo_directory)
                print(f"Git repository initialized successfully in: {repo_directory}")
                repository = Repository(self.connection)
                repository.create(RepositoryDTO( id: None, repo_name, vcs_type, repo_directory))
            except Exception as e:
                print(f"Error initializing Git repository: {e}")

    5 usages (5 dynamic)  yuliiaaaaa
    def commit(self, repo_path, file_name, message):
        try:
            repo = git.Repo(repo_path, search_parent_directories=True)
            if not repo.bare:
                index = repo.index
                if index.diff(None):
```

Реалізація SVNVersionCntrol

```
class SVNVersionControl(IVersionControlSystem):
    yuliiaaaaa
    def __init__(self, connection):
        self.connection = connection

    5 usages (5 dynamic)  yuliiaaaaa
    def commit(self, repo_url, message):
        try:
            client = pysvn.Client()
            client.checkin(repo_url, message)
            print(f"Committed changes with message: {message}")
        except pysvn.ClientError as svn_error:
            print(f"Error committing changes: {svn_error}")
        except Exception as e:
            print(f"Error committing changes: {e}")

    1 usage (1 dynamic)  yuliiaaaaa
    def watch_history(self, repo_url):
        try:
            client = pysvn.Client()
            log_entries = client.log(repo_url)
            for entry in log_entries:
                print(f"Revision {entry.revision.number}: {entry.message}")
        except pysvn.ClientError as svn_error:
            print(f"Error while watching commit history: {svn_error}")
        except Exception as e:
            print(f"Error while watching commit history: {e}")
```

У цьому коді фабричний метод реалізований у вигляді статичного методу create_version_control_system у класі VersionControlFactory. Цей метод приймає параметр vcs_type і залежно від цього типу повертає відповідний об'єкт системи контролю версій (MercurialVersionControl, SVNVersionControl або GitVersionControl).

При цьому код користувача не залежить від конкретної реалізації кожного класу системи контролю версій. Замість цього він використовує спільний інтерфейс IVersionControlSystem для взаємодії з усіма видами систем контролю версій.

Це дає нам гнучкість та можливість легко додавати нові типи систем контролю версій у майбутньому, не змінюючи великої частини вже існуючого коду, оскільки код Зпрацює з фабрикою та інтерфейсом, а не з конкретними реалізаціями.

Цей підхід дозволяє створювати об'єкти систем контролю версій без прив'язки коду до конкретних класів цих систем.

+ Позбавляє клас від прив’язки до конкретних класів продуктів.

+ Виділяє код виробництва продуктів в одне місце, спрощуючи підтримку коду.

+ Спрощує додавання нових продуктів до програми.

- Може призвести до створення великих паралельних ієрархій класів.

Висновок: я реалізувала факторі метод,де метод приймає параметр vcs_type і залежно від цього типу повертає відповідний об'єкт системи контролю версій.Це позбавляє клас від прив’язки до конкретних класів продуктів.