МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ

Факультет

Механіки, енергетики та ІТ

Кафедра

Інформаційних технологій

Практична робота №2

На тему: “Структурні шаблони проєктування”

Виконала студентка групи: IТ-31

Розвєзєва Христина Тарасівна

Перевірив: Татомир Андрій

Мета роботи - познайомитися з групою структурних шаблонів проєктування.

**Структурні шаблони проектування** спрощують дизайн системи, через організацію класів та об'єктів для формування великих структур. Вони спрямовані на створення ефективних взаємозв'язків між об'єктами, які підвищують масштабованість і підтримку коду. Основна ідея структурних шаблонів — це композиція класів та об'єктів для реалізації нової функціональності без зміни існуючих класів.

**Шаблони структурної групи:**

1. Adapter (Адаптер)
2. Bridge (Міст)
3. Composite (Компонувальник)
4. Decorator (Декоратор)
5. Facade (Фасад)
6. Flyweight (Примірник)
7. Proxy (Замісник)

**Опис:**

Шаблон Adapter використовується для того, щоб узгодити два несумісні інтерфейси без їх зміни. Він дозволяє класам із різними інтерфейсами співпрацювати, забезпечуючи можливість взаємодії старого коду з новим інтерфейсом. Це досягається через створення класу-адаптера, який "обгортає" один інтерфейс та представляє його як інший.

Приклад:

( Інтерфейс кавової машини)

class CoffeeMachine:

def brew\_coffee(self):

raise NotImplementedError(Метод, який потрібно реалізувати")

(Конкретна кавова машина)

class BasicCoffeeMachine(CoffeeMachine):

def brew\_coffee(self):

print("Готуємо каву...")

( Інтерфейс чайної машини)

class TeaMachine:

def brew\_tea(self):

raise NotImplementedError("Цей метод потрібно реалізувати")

(Конкретна чайна машина)

class BasicTeaMachine(TeaMachine):

def brew\_tea(self):

print("Готуємо чай...")

(Адаптер для чайної машини, щоб вона працювала як кавова)

class TeaMachineAdapter(CoffeeMachine):

def \_\_init\_\_(self, tea\_machine):

self.tea\_machine = tea\_machine

def brew\_coffee(self):

(Викликаєю метод для приготування чаю замість кави)

self.tea\_machine.brew\_tea()

(Тестування)

if \_\_name\_\_ == "\_\_main\_\_":

# Кавова машина

coffee\_machine = BasicCoffeeMachine()

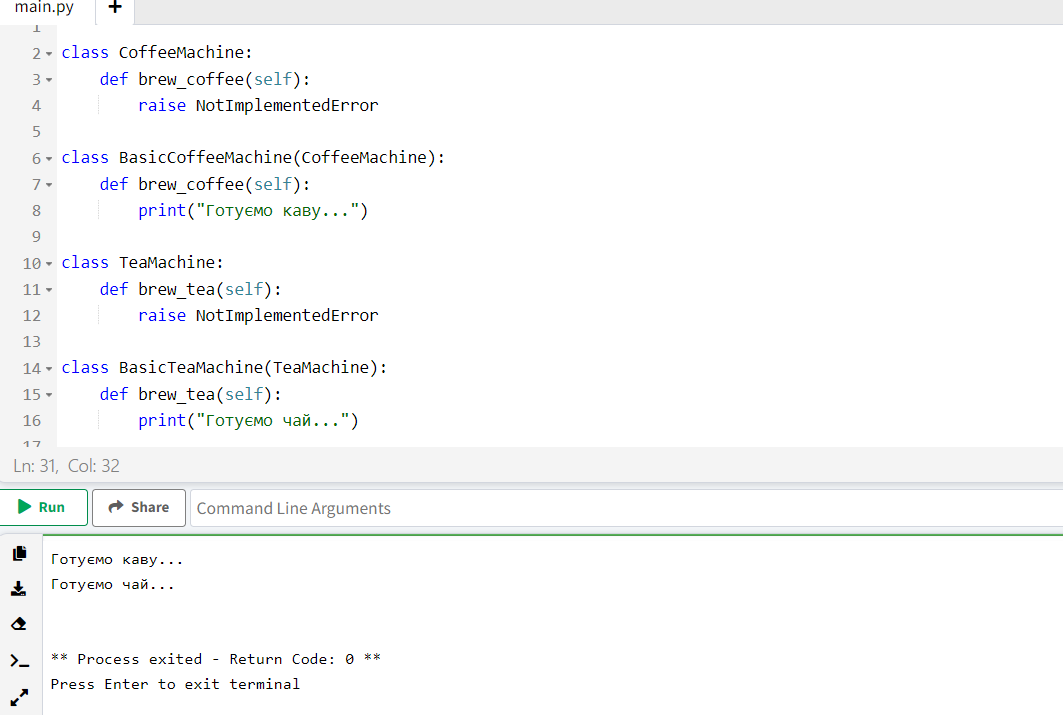
coffee\_machine.brew\_coffee() # Виводить: "Готуємо каву..."

(Чайна машина через адаптер)

tea\_machine = BasicTeaMachine()

tea\_adapter = TeaMachineAdapter(tea\_machine)

tea\_adapter.brew\_coffee() (Виводить: "Готуємо чай...")



UML-діаграма

BasicCoffeeMachine

+ brew\_coffee()

+ brew\_tea()

BasicTeaMachine

+ brew\_tea()

TeaMachine

+ brew\_coffee()

TeaMachineAdapter

+ brew\_coffee()

CoffeeMachine

**CoffeeMachine** — це клас для кавових машин із методом brew\_coffee().

**BasicCoffeeMachine** — клас, який реалізує кавову машину і містить метод brew\_coffee(). **TeaMachine** — це клас для чайних машин із методом brew\_tea().

**BasicTeaMachine —** клас, який реалізує чайну машину і містить метод brew\_tea().

**TeaMachineAdapter** — це адаптер, який імплементує інтерфейс **CoffeeMachine,** але обгортає **TeaMachine** і викликає метод brew\_tea() замість brew\_coffee().

Висновок: **Патерн Adapter** дозволяє узгоджувати інтерфейси двох несумісних класів, щоб вони могли працювати разом. У нашому прикладі адаптер дозволяє використовувати чайну машину через інтерфейс кавової машини, не змінюючи вихідний код класів. Це забезпечує гнучкість та розширюваність системи, зберігаючи принципи об'єктно-орієнтованого програмування.