++rr

**МIНIСТЕРСТВО ОСВIТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**

**ІМЕНІ ІГОРЯ СІКОРСЬКОГО”**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп’ютерних систем**

**Лабораторна робота № 1**

**з дисципліни “Об’єктно орієнтоване програмування”**

**тема “Реалізація структурних шаблонів проектування”**

|  |  |  |
| --- | --- | --- |
| **Виконав**  **студент II курсу**  **групи КП-83**  **Коваль Андрій Олекснадрович**  **(прізвище, ім’я, по батькові)** |  | **Перевірив**  **“\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.**  **викладач**  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **(прізвище, ім’я, по батькові)** |

**Київ 2020**

**Мета роботи**

Ознайомлення з основними характеристиками шаблонів «Декоратор», «Компонувальник» та «Міст», запам’ятовування поширених ситуацій, коли використання цих шаблонів є доцільним, набуття вмінь та навичок реалізації шаблонів під час створення програмного коду.

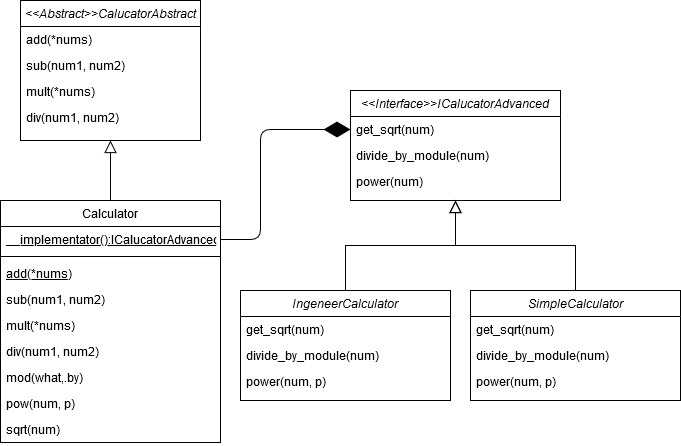
**Постановка завдання**

Варіант 8

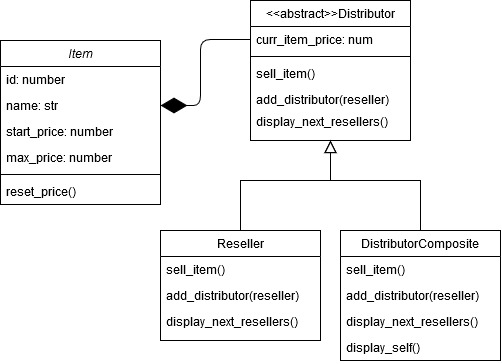
1. За допомогою використання шаблона Bridge створіть програму-калькулятор, яка працює в 2 режимах: звичайного калькулятора (забезпечує підтримку 4 основних математичних операцій: додавання, віднімання, множення, ділення) та інженерного калькулятора (забезпечує підтримку 4 основних математичних операцій, а також видобування квадратного кореня, обчислення залишку від ділення, піднесення числа до заданого ступеня).

2. За допомогою шаблона проектування реалізувати механізм продажу косметики через мережу дистриб’юторів (мережевий маркетинг). Кожна людина може бути або кінцевим продавцем косметики, або менеджером певної кількості підлеглих продавців, яким він збуває товар. Кожний менеджер продає косметику на певний відсоток дорожче, ніж він її отримав. Але перевищити кінцеву роздрібну вартість продукту, встановлену компанією, дистриб’ютор не має права.

**Діаграма класів**



Завдання 1

****

Завдання 2

**Фрагменти коду програми**

Завдання 1

|  |
| --- |
| **Абстрактрый калькулятор** |
| import functools  class CalculatorAbstract:  @staticmethod  def add(\*nums):  print('Adding nums', nums)  \_sum = sum(nums)  return \_sum  @staticmethod  def sub(num1, num2):  print('Substracting numbers', num1, num2)  return num1 - num2  @staticmethod  def mult(\*nums):  print('Multiplyinig numbers', nums)  return functools.reduce(lambda acc, curr: acc \* curr, nums)  @staticmethod  def div(num1, num2):  print('Dividing number', num1, 'by', num2)  return num1 / num2 |

|  |
| --- |
| **Продвинутий калькулятор** |
| from abc import ABCMeta, abstractmethod  class ICalculatorAdvanced():  \_\_metaclass\_\_= ABCMeta  # @staticmethod  @abstractmethod  def get\_sqrt(num):  pass    @staticmethod  @abstractmethod  def divide\_by\_module(what, by):  pass  @staticmethod  @abstractmethod  def power(num, p):  pass |
| **Два імплементатори** |
| from math import sqrt as math\_sqrt  from advanced\_calc\_interface import ICalculatorAdvanced    class IngeneerCalculator(ICalculatorAdvanced):  @staticmethod  def get\_sqrt(num: float) -> float:  print('Getting square root of', num)  return math\_sqrt(num)    @staticmethod  def divide\_by\_module(what: float, by: float) -> int:  print('Dividing', what, 'by module of', by)  return what % by  @staticmethod  def power(num: float, p: float):  print('Raising', num, 'to the power of', p)  return num \*\* p  class SimpleCalculator(ICalculatorAdvanced):  @staticmethod  def get\_sqrt(num: float):  print('Unable to get sqrt in a simple calculator')    @staticmethod  def divide\_by\_module(what: float, by: float) -> int:  print('Unable to divide by module in a simple calculator')  @staticmethod  def power(num: float, p: float) -> float:  print('Unable to get a power of a number in a simple calculator') |

|  |
| --- |
| **Головний міст** |
| class Calculator(CalculatorAbstract):  def \_\_init\_\_(self, implementator: ICalculatorAdvanced):  self.\_\_implementator = implementator    def result\_decorator(func):  def wrapper(self, \*args, \*\*kwargs):  result = func(self, \*args, \*\*kwargs)  self.\_\_display\_result(result)  return result  return wrapper  @staticmethod  def \_\_display\_result(result) -> None:  print('The result is', result)  @result\_decorator  def add(self, \*nums: float) -> float:  result = super().add(\*nums)  return result  @result\_decorator  def sub(self, num1: float, num2: float) -> float:  result = super().sub(num1, num2)  # self.\_\_display\_result(result)  return result  @result\_decorator  def mult(self, \*nums: float) -> float:  result = super().mult(\*nums)  # self.\_\_display\_result(result)  return result  @result\_decorator  def div(self, num1: float, num2: float) -> float:  result = super().div(num1, num2)  # self.\_\_display\_result(result)  return result    @result\_decorator  def mod(self, what: float, by: float) -> float:  result = self.\_\_implementator.divide\_by\_module(what, by)  # self.\_\_display\_result(result)  return result    @result\_decorator  def pow(self, num: float, p: float) -> float:  result = self.\_\_implementator.power(num, p)  # self.\_\_display\_result(result)  return result  @result\_decorator  def sqrt(self, num: float) -> float:  result = self.\_\_implementator.get\_sqrt(num)  # self.\_\_display\_result(result)  return result |

Завдання 2

|  |
| --- |
| **Абстрактний клас дистрибютора** |
| from item import Item  class Distributor:  @property  def curr\_item\_price(self):  if (self.item != None):  return self.item.current\_price \* (1 + (self.comission\_persentage / 100))  return 0  def \_\_init\_\_(self, name, comission\_persentage, item: Item = None):  self.child\_distributors = []  self.name = name  self.item = item  self.comission\_persentage = comission\_persentage    def sell\_item(self):  pass    def add\_distibutor(self, reseller):  pass  def remove(self, reseller):  pass  def display\_next\_resellers(self):  pass |

|  |
| --- |
| **Кінцевий продавець (листок компонувальника)** |
| class Reseller(Distributor):  def sell\_item(self, item):  if self.item == None:  self.item = item  new\_price = self.curr\_item\_price  if (new\_price > self.item.max\_price):  print('Unable with price more than was set by distributor')  print('Current price', new\_price)  print('Max price', self.item.max\_price)  else:  print('The product', self.item.name, 'has been sold for', new\_price)  def add\_distibutor(self, reseller):  print('Final reseller can not have partners')  def remove(self, reseller):  print("Final reseller doesn't have partners")  def display\_next\_resellers(self, item, level = 1):  print('\n') |

|  |
| --- |
| **Дистрибутор (компонувальник)** |
| class DistributorComposite(Distributor):  def sell\_item(self, item):  self.item = item  self.item.current\_price = self.curr\_item\_price  for d in self.child\_distributors:  print(self.name, 'resells the product to', d.name)  item.price = self.curr\_item\_price  d.sell\_item(item)  self.item.reset\_price()    def display\_self(self, level = 1):  print('-'.ljust(level \* 2, '-'), self.name, self.item.current\_price)  def add\_distibutor(self, reseller):  self.child\_distributors.append(reseller)  def remove(self, reseller):  self.child\_distributors.remove(reseller)  def display\_next\_resellers(self, item, level = 1):  self.item = item  self.item.current\_price = self.curr\_item\_price  self.display\_self(level)  for d in self.child\_distributors:  d.display\_next\_resellers(item, level + 1)  self.item.reset\_price() |

|  |
| --- |
| **Одиниця товару** |
| class Item:  id = 0  def \_\_init\_\_(self, name: str, start\_price: float, max\_price: float):  Item.id += 1  self.id = Item.id  self.name = name  self.start\_price = start\_price  self.current\_price = start\_price  self.max\_price = max\_price  def reset\_price(self):  self.current\_price = self.start\_price |

**Висновки**

Я ознайомився з основними характеристиками шаблонів «Декоратор», «Компонувальник» та «Міст», запам’ятав поширені ситуацій, коли використання цих шаблонів є доцільним та набув вмінь та навичок реалізації шаблонів під час створення програмного коду.