МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Лабораторна робота №5

з курсу

СПЕЦІАЛІЗОВАНІ МОВИ ПРОГРАМУВАННЯ

Виконав студент

групи ІТ-21сп

**Резніченко Р.Д.**

Прийняв

**Щербак С.С.**

Львів - 2023

**Мета:** створення додатка для малювання 3D-фігур у ASCII-арті на основі об’єктно-орієнтованого підходу та мови Python.

**Хід роботи**

**Завдання 1: Проектування класів.**

Розробіть структуру класів для вашого генератора 3D ASCII-арту. Визначте основні компоненти, атрибути та методи, необхідні для програми.

**Завдання 2: Введення користувача.**

Створіть методи у межах класу для введення користувача та вказання 3D-фігури, яку вони хочуть намалювати, та її параметрів (наприклад, розмір, кольори).

**Завдання 3: Представлення фігури.**

Визначте структури даних у межах класу для представлення 3D-фігури. Це може включати використання списків, матриць або інших структур даних для зберігання форми фігури та її властивостей.

**Завдання 4: Проектування з 3D в 2D.**

Реалізуйте метод, який перетворює 3D-представлення фігури у 2D-представлення, придатне для ASCII-арту.

**Завдання 5: Відображення ASCII-арту.**

Напишіть метод у межах класу для відображення 2D-представлення 3D-фігури як ASCII-арту. Це може включати відображення кольорів і форми за допомогою символів ASCII.

**Завдання 6: Інтерфейс, зрозумілий для користувача.**

Створіть зручний для користувача командний рядок або графічний інтерфейс користувача (GUI) за допомогою об'єктно-орієнтованих принципів, щоб дозволити користувачам спілкуватися з програмою.

**Завдання 7: Маніпуляція фігурою.**

Реалізуйте методи для маніпулювання 3D-фігурою, такі масштабування або зміщення, щоб надавати користувачам контроль над її виглядом.

**Завдання 8: Варіанти кольорів.**

Дозвольте користувачам вибирати варіанти кольорів для їхніх 3D ASCII-арт-фігур. Реалізуйте методи для призначення кольорів різним частинам фігури.

**Завдання 9: Збереження та експорт.**

Додайте функціональність для зберігання згенерованого 3D ASCII-арту у текстовий файл.

**Завдання 10: Розширені функції.**

Розгляньте можливість додавання розширених функцій, таких як тінь, освітлення та ефекти перспективи, для підвищення реалізму 3D ASCII-арту.

Код виконаних завдань представлено нижче.

Клас Cube:

from colorama import init, Fore  
import re  
  
class Cube:  
 def \_\_init\_\_(self, size=1, color='WHITE', symbol='#', remove\_shades=False):  
 self.size = size  
 self.color = color  
 self.symbol = symbol  
 self.remove\_shades = remove\_shades  
 self.cube\_3D = None  
 self.cube\_2D = None  
  
 def remove\_color\_codes(self, string):  
 # This regular expression matches all color codes  
 ansi\_escape = re.compile(r'\x1B\[[0-?]\*[ -/]\*[@-~]')  
 return ansi\_escape.sub('', string)  
  
 def draw\_3D(self):  
 # Initialize colorama  
 init()  
  
 # Define colors  
 colors = {  
 'RED': Fore.RED,  
 'GREEN': Fore.GREEN,  
 'YELLOW': Fore.YELLOW,  
 'BLUE': Fore.BLUE,  
 'MAGENTA': Fore.MAGENTA,  
 'CYAN': Fore.CYAN,  
 'WHITE': Fore.WHITE  
 }  
  
 # Define matrix for cube representation  
 self.cube\_3D = [  
 ["\*" \* (3 if self.size == 1 else 3) + colors[self.color] + "X" + "=" \* self.size \* 10 + "X" + Fore.RESET],  
 ["\*" \* (1 if self.size == 1 else 1) + colors[self.color] + "/" + self.symbol \* self.size \* 11 + "/" + "|"],  
 ["X" + "=" \* self.size \* 11 + "X" + self.symbol \* self.size + "|"],  
 ["|" + self.symbol \* self.size \* 11 + "|" + self.symbol \* self.size + "X"],  
 ["|" + self.symbol \* self.size \* 11 + "|" + self.symbol \* self.size + "/"],  
 ["|" + self.symbol \* self.size \* 11 + "|" + "/"],  
 ["X" + "=" \* self.size \* 11 + "X" + Fore.RESET]  
 ]  
  
 # Cube shades  
 if self.remove\_shades:  
 self.cube\_3D[0][0] = self.cube\_3D[0][0].replace("\*", " ", 3 if self.size == 1 else 3)  
 self.cube\_3D[1][0] = self.cube\_3D[1][0].replace("\*", " ", 1)  
  
 # Add spaces in place of removed "\*"  
 if self.size == 2 and self.remove\_shades:  
 self.cube\_3D[0][0] = " " + self.cube\_3D[0][0]  
 self.cube\_3D[1][0] = " " + self.cube\_3D[1][0]  
  
 # Print the cube  
 for row in self.cube\_3D:  
 print(''.join(row))  
  
 def draw\_2D(self):  
 # Initialize colorama  
 init()  
  
 # Define colors  
 colors = {  
 'RED': Fore.RED,  
 'GREEN': Fore.GREEN,  
 'YELLOW': Fore.YELLOW,  
 'BLUE': Fore.BLUE,  
 'MAGENTA': Fore.MAGENTA,  
 'CYAN': Fore.CYAN,  
 'WHITE': Fore.WHITE  
 }  
  
 # Define matrix for 2D representation  
 self.cube\_2D = [  
 [colors[self.color] + "X" + "=" \* self.size \* 11 + "X"],  
 ["|" + self.symbol \* self.size \* 11 + "|"],  
 ["|" + self.symbol \* self.size \* 11 + "|"],  
 ["|" + self.symbol \* self.size \* 11 + "|"],  
 ["X" + "=" \* self.size \* 11 + "X" + Fore.RESET]  
 ]  
  
 # Print the 2D representation  
 for row in self.cube\_2D:  
 print(''.join(row))  
  
 def saveCube\_3D(self, filename):  
 # Check if cube\_3D is not None  
 if self.cube\_3D is None:  
 print("Error: Draw the 3D cube first.")  
 return  
  
 # Remove color codes from cube\_3D  
 cube\_3D\_no\_color = [''.join([self.remove\_color\_codes(cell) for cell in row]) for row in self.cube\_3D]  
  
 # Save the cube to a file  
 with open(filename, 'w') as f:  
 for row in cube\_3D\_no\_color:  
 f.write(row + '\n')

Клас Pyramid:

from colorama import init, Fore  
import re  
  
class Pyramid:  
 def \_\_init\_\_(self, size=1, color='WHITE', symbol='#', remove\_shadesPyramid=False):  
 self.size = size  
 self.color = color  
 self.symbol = symbol  
 self.remove\_shadesPyramid = remove\_shadesPyramid  
 self.pyramid\_3D = None  
 self.pyramid\_2D = None  
  
 def remove\_color\_codes(self, string):  
 # This regular expression matches all color codes  
 ansi\_escape = re.compile(r'\x1B\[[0-?]\*[ -/]\*[@-~]')  
 return ansi\_escape.sub('', string)  
  
 def remove\_shades(self):  
 # Pyramid shades  
 if self.pyramid\_3D is not None:  
 self.pyramid\_3D = [[cell.replace("=", "") for cell in row] for row in self.pyramid\_3D]  
 if self.pyramid\_2D is not None:  
 self.pyramid\_2D = [[cell.replace("=", "") for cell in row] for row in self.pyramid\_2D]  
  
 def draw\_3D(self):  
 # Initialize colorama  
 init()  
  
 # Define colors  
 colors = {  
 'RED': Fore.RED,  
 'GREEN': Fore.GREEN,  
 'YELLOW': Fore.YELLOW,  
 'BLUE': Fore.BLUE,  
 'MAGENTA': Fore.MAGENTA,  
 'CYAN': Fore.CYAN,  
 'WHITE': Fore.WHITE  
 }  
  
 # Define matrix for pyramid representation  
 self.pyramid\_3D = [  
 [colors[self.color] + " " \* self.size \* 11 + "/\\" + "\*" \* self.size + "."],  
 [" " \* self.size \* 10 + "/" + self.symbol \* self.size \* 2 + "\\" + "\*" \* self.size \* 2 + "."],  
 [" " \* self.size \* 9 + "/" + self.symbol \* self.size \* 4 + "\\" + "\*" \* self.size \* 3 + "."],  
 [" " \* self.size \* 8 + "/" + self.symbol \* self.size \* 6 + "\\" + "\*" \* self.size \* 4 + "."],  
 [" " \* self.size \* 7 + "/" + self.symbol \* self.size \* 8 + "\\" + "\*" \* self.size \* 4 + "|"],  
 [" " \* self.size \* 6 + "/" + self.symbol \* self.size \* 10 + "\\" + "\*" \* self.size \* 3 + "|"],  
 [" " \* self.size \* 5 + "/" + self.symbol \* self.size \* 12 + "\\" + "\*" \* self.size \* 2 + "|"],  
 [" " \* self.size \* 4 + "/" + self.symbol \* self.size \* 14 + "\\" + "\*" \* self.size + "|"],  
 [" " \* self.size \* 3 + "/" + "\_" \* self.size \* 16 + "\\" + "|" + Fore.RESET],  
 [" " \* self.size \* 2 + "=" \* self.size \* 15],  
 [" " \* self.size \* 2 + "=" \* self.size \* 12],  
 [" " \* self.size \* 2 + "=" \* self.size \* 9],  
 [" " \* self.size \* 2 + "=" \* self.size \* 6],  
 [" " \* self.size \* 2 + "=" \* self.size \* 3]  
 ]  
  
 # If remove\_shadesPyramid is True, remove the shades  
 if self.remove\_shadesPyramid:  
 self.remove\_shades()  
  
 # Print the pyramid  
 for row in self.pyramid\_3D:  
 print(''.join(row))  
  
 def draw\_2D(self):  
 # Initialize colorama  
 init()  
  
 # Define colors  
 colors = {  
 'RED': Fore.RED,  
 'GREEN': Fore.GREEN,  
 'YELLOW': Fore.YELLOW,  
 'BLUE': Fore.BLUE,  
 'MAGENTA': Fore.MAGENTA,  
 'CYAN': Fore.CYAN,  
 'WHITE': Fore.WHITE  
 }  
  
 # Define matrix for 2D representation  
 self.pyramid\_2D = [  
 [colors[self.color] + " " \* self.size \* 11 + "/\\"],  
 [" " \* self.size \* 10 + "/" + self.symbol \* self.size \* 2 + "\\"],  
 [" " \* self.size \* 9 + "/" + self.symbol \* self.size \* 4 + "\\"],  
 [" " \* self.size \* 8 + "/" + self.symbol \* self.size \* 6 + "\\"],  
 [" " \* self.size \* 7 + "/" + self.symbol \* self.size \* 8 + "\\"],  
 [" " \* self.size \* 6 + "/" + self.symbol \* self.size \* 10 + "\\"],  
 [" " \* self.size \* 5 + "/" + self.symbol \* self.size \* 12 + "\\"],  
 [" " \* self.size \* 4 + "/" + self.symbol \* self.size \* 14 + "\\"],  
 [" " \* self.size \* 3 + "/" + "\_" \* self.size \* 16 + "\\" + Fore.RESET]  
 ]  
  
 # If remove\_shadesPyramid is True, remove the shades  
 if self.remove\_shadesPyramid:  
 self.remove\_shades()  
  
 # Print the 2D representation  
 for row in self.pyramid\_2D:  
 print(''.join(row))  
  
 def set\_size(self, size):  
 if size in [1, 2]:  
 self.size = size  
 else:  
 print("Invalid size. Please choose 1 or 2.")  
  
 def savePyramid\_3D(self, filename):  
 # Check if pyramid\_3D is not None  
 if self.pyramid\_3D is None:  
 print("Error: Draw the 3D pyramid first.")  
 return  
  
 # Remove color codes from pyramid\_3D  
 pyramid\_3D\_no\_color = [''.join([self.remove\_color\_codes(cell) for cell in row]) for row in self.pyramid\_3D]  
  
 # Save the pyramid to a file  
 with open(filename, 'w') as f:  
 for row in pyramid\_3D\_no\_color:  
 f.write(row + '\n')

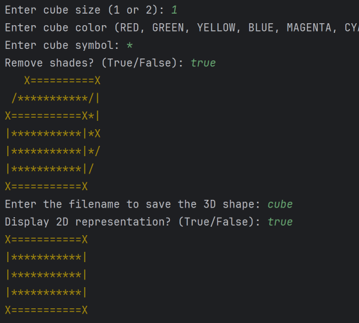
Клас ShapeInterface:

# Importing Cube and Pyramid classes from modules Lb5.Cube and Lb5.Pyramid  
from Lb5.Cube import Cube  
from Lb5.Pyramid import Pyramid  
  
# Interface class for creating, drawing, and manipulating 3D shapes  
class ShapeInterface:  
 def \_\_init\_\_(self):  
 self.shape = None  
  
 # Method to create a cube based on user input  
 def create\_cube(self):  
 size = int(input("Enter cube size (1 or 2): "))  
 color = input("Enter cube color (RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE): ")  
 symbol = input("Enter cube symbol: ")  
 remove\_shades = input("Remove shades? (True/False): ").lower() == 'true'  
 self.shape = Cube(size, color, symbol, remove\_shades)  
  
 # Method to create a pyramid based on user input  
 def create\_pyramid(self):  
 size = int(input("Enter pyramid size (1 or 2): "))  
 color = input("Enter pyramid color (RED, GREEN, YELLOW, BLUE, MAGENTA, CYAN, WHITE): ")  
 symbol = input("Enter pyramid symbol: ")  
 remove\_shades\_pyramid = input("Remove shades in pyramid? (True/False): ").lower() == 'true'  
 self.shape = Pyramid(size, color, symbol, remove\_shades\_pyramid)  
  
 # Method to draw the 3D representation of the shape  
 def draw\_3D(self):  
 if self.shape is not None:  
 self.shape.draw\_3D()  
 else:  
 print("Error: No shape created. Use create\_cube() or create\_pyramid() first.")  
  
 # Method to draw the 2D representation of the shape  
 def draw\_2D(self):  
 if self.shape is not None:  
 self.shape.draw\_2D()  
 else:  
 print("Error: No shape created. Use create\_cube() or create\_pyramid() first.")  
  
 # Method to set a new size for the shape  
 def set\_size(self):  
 if self.shape is not None:  
 size = int(input("Enter new size (1 or 2): "))  
 self.shape.set\_size(size)  
 else:  
 print("Error: No shape created. Use create\_cube() or create\_pyramid() first.")  
  
 # Method to save the 3D shape to a file  
 def save\_shape\_3D(self):  
 if self.shape is not None:  
 filename = input("Enter the filename to save the 3D shape: ")  
 if isinstance(self.shape, Cube):  
 self.shape.saveCube\_3D(filename + ".txt")  
 elif isinstance(self.shape, Pyramid):  
 self.shape.savePyramid\_3D(filename + ".txt")  
 else:  
 print("Error: No shape created. Use create\_cube() or create\_pyramid() first.")  
  
 # Method to display the 2D representation of the shape based on user input  
 def display\_2D\_option(self):  
 if self.shape is not None:  
 option = input("Display 2D representation? (True/False): ").lower() == 'true'  
 if option:  
 self.shape.draw\_2D()  
 else:  
 print("Error: No shape created. Use create\_cube() or create\_pyramid() first.")  
  
# Example usage of the ShapeInterface  
interface = ShapeInterface()  
  
# User input to choose the type of shape (cube or pyramid)  
shape\_type = input("Enter shape type (cube-c/pyramid-p): ").lower()  
if shape\_type == 'c':  
 interface.create\_cube()  
elif shape\_type == 'p':  
 interface.create\_pyramid()  
else:  
 print("Invalid shape type. Choose 'cube' or 'pyramid'.")  
  
# Displaying the 3D representation, saving to a file, and displaying the 2D representation  
interface.draw\_3D()  
interface.save\_shape\_3D()  
interface.display\_2D\_option()

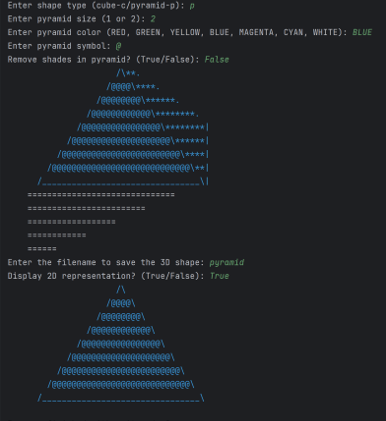
Клас Runner:

# Importing the ShapeInterface class from the Lb5.Interface module  
from Lb5.Interface import ShapeInterface  
  
# Class for running the program and interacting with the user  
class Runner:  
 def \_\_init\_\_(self):  
 # Initializing an instance of the ShapeInterface  
 self.interface = ShapeInterface()  
  
 # Main entry point of the program  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Creating an instance of the Runner class and running the program  
 runner = Runner()  
 runner.run()

На рисунку 1 та 2 зображено результат виконання програми.



*Рис.1 Вивід куба*

**

*Рис.2 Вивід піраміди*

**Висновок:** під час виконання лабораторної роботи я навчився створювати високорівневий об'єктно-орієнтований генератор 3D ASCII-арту, який дозволить користувачам проектувати, відображати та маніпулювати 3D-фігурами в ASCII-арті.