МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»
Кафедра інформаційних систем та мереж

ЗВІТ

про виконання лабораторної роботи № 5
«Розробка ASCII ART генератора для візуалізації 3D-фігур»
з дисципліни "Спеціалізовані мови програмування”

Виконала:
ст. гр. ІТ-32,
Троцько О. М.
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.

План 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 програми:

```
# shape.py
"""
Shape Module

A module that defines the Shape abstract base class for shapes.
"""
from abc import ABC, abstractmethod

class Shape(ABC):
    """
    Abstract base class for shapes.
    """

    @abstractmethod
    def generate_2D(self):
        """
        Abstract method to generate a 2D representation of the shape.
        """

    @abstractmethod
    def generate_3D(self):
        """
        Abstract method to generate a 3D representation of the shape.
        """

    @abstractmethod
```

```
    def set_settings(self):
        """
        Abstract method to set the settings of the shape.
        """
```

 # cube.py

```
"""
Cube Module

This module defines the Cube class, representing a cube shape.
The Cube class is a subclass of the Shape and PrintArt classes.
It provides methods for generating 2D and 3D representations of a cube,
initializing a 3D array, setting coordinates in the array, and more.
"""
from shared.settings import get_lab_settings
from classes.lab5.art.art_settings import ArtSettings
from classes.lab5.art.print_art import PrintArt
from classes.lab5.shapes.shape import Shape

settings = get_lab_settings("lab5")
DEFAULT_SHAPE_SETTINGS = settings["default_shape_settings"]
DEFAULT_COLOR = DEFAULT_SHAPE_SETTINGS["color"]
DEFAULT_SIZE = DEFAULT_SHAPE_SETTINGS["size"]
DEFAULT_JUSTIFY = DEFAULT_SHAPE_SETTINGS["justify"]


class Cube(Shape, PrintArt):
    """
    Represents a cube shape.

    Attributes:
    - size: The size of the cube
    - justify: The justification of the cube
    - color: The color of the cube
    """

             def    __init__(self,    size=DEFAULT_SIZE,    justify=DEFAULT_JUSTIFY,
color=DEFAULT_COLOR):
        """
        Initializes a new instance of the Cube class.
```

```
    Parameters:
    - size: The size of the cube (default: DEFAULT_SIZE)
    - justify: The justification of the cube (default: DEFAULT_JUSTIFY)
    - color: The color of the cube (default: DEFAULT_COLOR)
    """
    self._settings = ArtSettings(size, justify, color)
    self._art_2D = self.generate_2D()
    self._art_3D = self.generate_3D()
    PrintArt.__init__(self, self)


def set_settings(self, cube_settings):
    """
    Sets the settings of the cube.

    Parameters:
    - settings: The settings object to be set
    """
    self._settings = cube_settings
    self._art_2D = self.generate_2D()
    self._art_3D = self.generate_3D()


def get_settings(self):
    """
    Returns the settings of the cube.

    Returns:
    - The settings of the cube
    """
    return self._settings

def generate_2D(self):
    """
    Generates the 2D representation of the cube.

    Returns:
    - The 2D representation of the cube as a string
    """
    size = self._settings.get_size()
```

```
    array_3d = self.__set_coordinates(self.__init_3d_array())
    layers = self.__get_layers(array_3d)
    art_len = size - 1

    art = ""
    art += layers[art_len] + "\n"
    for i in range(art_len):
        art += layers[-(size+1)].replace("—", " ") + "\n"

    art += layers[-1] + "\n"

    return art

def generate_3D(self):
    """
    Generates the 3D representation of the cube.

    Returns:
    - The 3D representation of the cube as a string
    """
    array_3d = self.__set_coordinates(self.__init_3d_array())
    layers = self.__get_layers(array_3d)

    size = self._settings.get_size()
    num_of_layers = len(layers)
    art_len = size - 1

    index_1 = 1
    index_2 = size
    # print(layers)

    art = layers[0] + "\n"

    for i in range(art_len):
        art += self.__unite_str(layers[index_1 + i], layers[index_2]) + "\n"

    index_2 = index_2 + index_2 - 1
    index_1 = num_of_layers - size

    for i in range(art_len):
```

```
            art += self.__unite_str(layers[index_1 + i], layers[index_2]) + "\n"

        art += layers[num_of_layers-1]

        return art


    def __init_3d_array(self):
        """
        Initializes a 3D array for the cube.

        Returns:
        - The initialized 3D array
        """
        size = self._settings.get_size()
          array_3d = [[[0 for col in range(size)]for row in range(size)] for x in
range(size)]
        for i in range(size):
            for j in range(size):
                for k in range(size):
                    array_3d[i][j][k] = '0'
        return array_3d


    def __set_coordinates(self, array_3d):
        """
        Sets the coordinates of the cube in the 3D array.

        Parameters:
        - array_3d: The 3D array to set the coordinates in

        Returns:
        - The updated 3D array with coordinates set
        """
        N = self._settings.get_size() - 1
        array_3d[0][0][0] = "A"
        array_3d[N][0][0] = "B"
        array_3d[N][0][N] = "C"
        array_3d[0][0][N] = "D"
        array_3d[0][N][0] = "E"
```

```
        array_3d[N][N][0] = "F"
        array_3d[N][N][N] = "G"
        array_3d[0][N][N] = "H"

        return array_3d

    def __get_layers(self, array_3d):
        """
        Gets the layers of the cube from the 3D array.

        Parameters:
        - array_3d: The 3D array representing the cube

        Returns:
        - The layers of the cube as a list of strings
        """
        N = self._settings.get_size() - 1
        layers = []
        for y in range(N, -1, -1):
            for z in range(N, -1, -1):
                layer = " "*(z)
                for x in range(N+1):
                    if array_3d[x][y][z]  == '0':
                        if (y != 0 or y != N) and (x == 0 or x == N) and (z == 0 or
z == N):
                            layer += "|"
                        elif  x == 0 or x == N:
                            layer += "/"
                        elif z == 0 or z == N:
                            layer += " — "
                        else:
                            layer += "   "
                    else:
                        layer += array_3d[x][y][z]
                layers.append(layer)
            # print(layer, end=" ")
            # print()
        return layers
```

```
    def __unite_str(self, str1, str2):
        """
         Unites two strings by replacing spaces in the first string with non-space
characters from the second string.

        Parameters:
        - str1: The first string
        - str2: The second string

        Returns:
        - The united string
        """
        result = ""
        if len(str1) < len(str2):
            str1 += " " * (len(str2) - len(str1))
        else:
            str2 += " " * (len(str1) - len(str2))

        for i in range(len(str1)):
            if str1[i] != " " :
                result += str1[i]
            elif str2[i] != " " and str2[i] != "—":
                result += str2[i]
            else:
                result += " "
        return result

    def get_2D(self):
        """
        Returns the 2D representation of the cube.

        Returns:
        - The 2D representation of the cube as a string
        """
        return self._art_2D

    def get_3D(self):
        """
        Returns the 3D representation of the cube.
```

```
        Returns:
        - The 3D representation of the cube as a string
        """
        return self._art_3D
```

 # pyramid.py

```
"""
Pyramid Module

This module defines the Pyramid class, representing a pyramid shape.
The Pyramid class is a subclass of the Shape and PrintArt classes.
It provides methods for generating 2D and 3D representations of a pyramid,
initializing a 3D array, setting coordinates in the array, and more.
"""
from classes.lab5.art.art_settings import ArtSettings
from classes.lab5.art.print_art import PrintArt
from shared.settings import get_lab_settings
from .shape import Shape

settings = get_lab_settings("lab5")
DEFAULT_SHAPE_SETTINGS = settings["default_shape_settings"]
DEFAULT_SIZE = DEFAULT_SHAPE_SETTINGS["size"]
DEFAULT_JUSTIFY = DEFAULT_SHAPE_SETTINGS["justify"]
DEFAULT_COLOR = DEFAULT_SHAPE_SETTINGS["color"]

main = "|"
back = "-"

class Pyramid(Shape, PrintArt):
    """
    Represents a pyramid shape.

    Attributes:
        _settings (ArtSettings): The settings of the pyramid.
        _art_2D (str): The 2D representation of the pyramid.
        _art_3D (str): The 3D representation of the pyramid.
    """

              def    __init__(self,    size=DEFAULT_SIZE,    justify=DEFAULT_JUSTIFY,
color=DEFAULT_COLOR):
```

```
    """
    Initializes a Pyramid object.

    Args:
        size (int): The size of the pyramid.
        justify (str): The justification of the pyramid.
        color (str): The color of the pyramid.
    """
    self._settings = ArtSettings(size, justify, color)
    self._art_2D = self.generate_2D()
    self._art_3D = self.generate_3D()
    PrintArt.__init__(self, self)

def set_settings(self, pyramid_settings):
    """
    Sets the settings of the pyramid.

    Args:
        settings (ArtSettings): The settings to be set.
    """
    self._settings = pyramid_settings
    self._art_2D = self.generate_2D()
    self._art_3D = self.generate_3D()

def get_settings(self):
    """
    Returns the settings of the pyramid.

    Returns:
        ArtSettings: The settings of the pyramid.
    """
    return self._settings

def generate_2D(self):
    """
    Generates the 2D representation of the pyramid.

    Returns:
        str: The 2D representation of the pyramid.
    """
```

```
    art = ""
    counter = 1
    size = self._settings.get_size()

    for i in range(1, size+1):
        space = " " * (size - i)
        art += space + main * counter + "\n"
        counter += 2

    return art

def generate_3D(self):
    """
    Generates the 3D representation of the pyramid.

    Returns:
        str: The 3D representation of the pyramid.
    """
    art = ""
    counter = 1
    size = self._settings.get_size()
    stop = size - round(size/5)
    step = round(stop/(size - stop+2))
    num = step

    for i in range(1, size+1):
        space = " " * (size - i)

        if i >= stop:
            art += space + main * counter + back * (i-num) + "\n"
            num += step+1
        else:
            art += space + main * counter + back * i + "\n"

        counter += 2

    return art

def get_2D(self):
    """
```

```
        Returns the 2D representation of the pyramid.

        Returns:
            str: The 2D representation of the pyramid.
        """
        return self._art_2D

    def get_3D(self):
        """
        Returns the 3D representation of the pyramid.

        Returns:
            str: The 3D representation of the pyramid.
        """
        return self._art_3D
```

 # art_settings.py

```
"""
Art Settings Module

A module that defines the ArtSettings class representing the settings for creating
art.
"""
from UI.menu import Menu
from UI.menu_item import Item
from  classes.lab4.console_reader.data_from_console  import  get_size_from_console,
get_color_from_console, get_justify_from_console

class ArtSettings:
    """
    A class that represents the settings for creating art.

    Attributes:
        __size (int): The size of the art.
        __justify (str): The justification of the art.
        __color (str): The color of the art.
    """
    def __init__(self, size=5, justify="left", color="white"):
        """
        Initializes an instance of the ArtSettings class.
```

```
        Args:
            size (int): The size of the art.
            justify (str): The justification of the art.
            color (str): The color of the art.

        Returns:
            None
        """
        self.__size = size
        self.__justify = justify
        self.__color = color

    def __str__(self):
                return  f"Size:  {self.__size}\nColor:  {self.__color}\nJustify:
{self.__justify}"

    def settings_menu(self):
        """
        Displays the settings menu.
        """
        settings_menu = Menu("\nSettings Menu")
        settings_menu.add_item(Item('1', 'Change Size', self.change_size))
        settings_menu.add_item(Item('2', 'Change Color', self.change_color))
        settings_menu.add_item(Item('3', 'Change Justify', self.change_justify))
        settings_menu.add_item(Item('4', 'View Settings', self.print_settings))
        settings_menu.add_item(Item('0', 'Back'))

        settings_menu.run()

    def change_size(self):
        """
        Changes the size of the art.
        """
        self.__size = get_size_from_console()

    def change_color(self):
        """
        Changes the color of the art.
        """
```

```
        self.__color = get_color_from_console()

    def change_justify(self):
        """
        Changes the justification of the art.
        """
        self.__justify = get_justify_from_console()

    def set_size(self, size):
        """
        Sets the size of the art.

        Parameters:
            size (int): The size of the art.
        """
        self.__size = size

    def set_color(self, color):
        """
        Sets the color of the art.

        Parameters:
            color (str): The color of the art.
        """
        self.__color = color

    def set_justify(self, justify):
        """
        Sets the justification of the art.

        Parameters:
            justify (str): The justification of the art.
        """
        self.__justify = justify

    def get_size(self):
        """
        Returns the size of the art.

        Returns:
```

```
            int: The size of the art.
        """
        return self.__size

    def get_color(self):
        """
        Returns the color of the art.

        Returns:
            str: The color of the art.
        """
        return self.__color

    def get_justify(self):
        """
        Returns the justification of the art.

        Returns:
            str: The justification of the art.
        """
        return self.__justify

    def get_settings_obj(self):
        """
        Returns the ArtSettings object.

        Returns:
            ArtSettings: The ArtSettings object.
        """
        return self

    def print_settings(self):
        """
        Prints the current settings of the art.
        """
        print(str(self))

# print_art.py
"""
Print Art Module
```

```
A module that defines the PrintArt class for printing 2D and 3D art.
"""
from termcolor import colored
from classes.lab4.console_reader.data_from_console import get_console_width

class PrintArt:
    """
    A class that represents the printing of art.

    Attributes:
    - _shape: The Shape object to be printed.
    """

    def __init__(self, Shape):
        """
        Initialize the PrintArt object.

        Parameters:
        - Shape: The Shape object to be printed.
        """
        self._shape = Shape

    def get_shape(self):
        """
        Get the Shape object associated with the PrintArt object.

        Returns:
        - The Shape object.
        """
        return self._shape

    def set_shape(self, shape):
        """
        Set the Shape object associated with the PrintArt object.

        Parameters:
        - shape: The new Shape object.
        """
        self._shape = shape
```

```
    def print_art_2D(self):
        """
        Print the 2D art of the Shape object.
        """
        art = self.__justify_art(self._shape.get_2D())
        print(colored("\nArt 2D\n"))
        print(colored(art, self._shape.get_settings().get_color()))

    def print_art_3D(self):
        """
        Print the 3D art of the Shape object.
        """
        art = self.__justify_art(self._shape.get_3D())
        print(colored("\nArt 3D\n"))
        print(colored(art, self._shape.get_settings().get_color()))

    def __justify_art(self, art):
        """
        Justify the art by adding padding based on the justification settings.

        Parameters:
        - art: The art to be justified.

        Returns:
        - The justified art.
        """
        padding = self.__get_padding(art)
        art_lines = art.split('\n')
        aligned_lines = [" " * padding + line for line in art_lines]
        art = '\n'.join(aligned_lines)
        return art

    def __get_padding(self, art):
        """
            Calculate the padding based on the console width and justification
settings.

        Parameters:
        - art: The art to be justified.
```

```
        Returns:
        - The padding value.
        """
        console_width = get_console_width()
        art_len = len(art)//self._shape.get_settings().get_size()
        justify = self._shape.get_settings().get_justify()

        if justify == "center":
            return (console_width - art_len) // 2
        if justify == "right":
            return console_width - art_len
        return 0
```

 # art_menu.py

```
"""
Art Menu Module

A module that defines the ArtMenu class for creating and manipulating shapes.
"""
from UI.menu import Menu
from UI.menu_item import Item
from shared.file_handler import FileHandler
from shared.settings import get_lab_settings
from classes.lab5.shapes.cube import Cube
from classes.lab5.shapes.pyramid import Pyramid

settings = get_lab_settings("lab5")
ART_2D_PATH = settings["art_2D_path"]
ART_3D_PATH = settings["art_3D_path"]

class ArtMenu:
    """
    A class representing an art menu for creating and manipulating shapes.

    Attributes:
    - __shape: The currently selected shape.
    """
    def __init__(self):
        self.__shape = None
```

```
def menu(self):
    """
    Displays the main menu for the art program.
    Allows the user to choose a shape or exit the program.
    """
    art_menu = Menu("\nArt Menu")
    art_menu.add_item(Item('1', 'Choose shape', self.set_shape))
    art_menu.add_item(Item('0', 'Exit'))

    art_menu.run()

def set_shape(self):
    """
    Sets the shape based on user input.
    """
    shape = input("Choose shape [ cube, pyramid ]: ")
    if shape not in ['cube', 'pyramid']:
        print("Invalid shape")
        return

    if shape == 'cube':
        self.__shape = Cube()

    if shape == 'pyramid':
        self.__shape = Pyramid()

    self.sub_menu()

def get_shape(self):
    """
    Returns the currently selected shape.
    """
    return self.__shape

def sub_menu(self):
    """
    Displays a sub-menu for viewing and manipulating the shape.
    """
    sub_menu = Menu("\nArt Menu")
```

```
        sub_menu.add_item(Item('1', 'View 2D', self.view_art_2D))
        sub_menu.add_item(Item('2', 'View 3D', self.view_art_3D))
        sub_menu.add_item(Item('3', 'Change Settings', self.change_settings))
        sub_menu.add_item(Item('4', 'Save Art', self.save_menu))
        sub_menu.add_item(Item('0', 'Back'))

        sub_menu.run()

    def view_art_2D(self):
        """
        Prints the 2D representation of the shape.
        """
        self.__shape.print_art_2D()

    def view_art_3D(self):
        """
        Prints the 3D representation of the shape.
        """
        self.__shape.print_art_3D()

    def change_settings(self):
        """
        Displays a menu for changing the settings of the shape.
        """
        self.__shape.get_settings().settings_menu()
        self.__shape.set_settings(self.__shape.get_settings().get_settings_obj())

    def save_menu(self):
        """
        Displays a menu for saving and viewing the saved art.
        """
        save_menu = Menu("\nSave Menu")
        save_menu.add_item(Item('1', 'Save 2D to File', self._save_to_file_2D))
        save_menu.add_item(Item('2', 'Save 3D to File', self._save_to_file_3D))
        save_menu.add_item(Item('3', 'View Saved 2D', self._view_saved_2D))
        save_menu.add_item(Item('4', 'View Saved 3D', self._view_saved_3D))
        save_menu.add_item(Item('0', 'Back'))

        save_menu.run()
```

```
    def _save_to_file_2D(self):
        """
        Saves the 2D representation of the shape to a file.
        """
        saved_file = FileHandler(ART_2D_PATH)
        saved_file.write_to_file(self.__shape.get_2D())

    def _save_to_file_3D(self):
        """
        Saves the 3D representation of the shape to a file.
        """
        saved_file = FileHandler(ART_3D_PATH)
        saved_file.write_to_file(self.__shape.get_3D())

    def _view_saved_2D(self):
        """
        Reads and displays the saved 2D art from a file.
        """
        saved_file = FileHandler(ART_2D_PATH)
        saved_file.read_from_file()

    def _view_saved_3D(self):
        """
        Reads and displays the saved 3D art from a file.
        """
        saved_file = FileHandler(ART_3D_PATH)
        saved_file.read_from_file()
```

 # runner.py

```
"""
Module: run_art_menu

Module provides a simple script to run the Art Menu for Lab 5.
"""
from classes.lab5.art.art_menu import ArtMenu

def run():
    art_menu = ArtMenu()
    art_menu.menu()
```

GitHub Repository: https://github.com/trolchiha/SPL-labs.git

Висновок: під час виконання лабораторної роботи навчилася створювати високорівневий об'єктно-орієнтований генератор 3D ASCII-арту, який дозволяє користувачам проектувати, відображати та маніпулювати 3D-фігурами в ASCII-арті. Цей проект надав глибоке розуміння об'єктно-орієнтованого програмування і алгоритмів графіки, сприятиме творчому підходу до створення ASCII-арту.