Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №4

з дисципліни
«Об’єктно-орієнтоване програмування»

Виконала:

студентка групи ІМ-43
Хубеджева Єлизавета Павлівна
номер у списку групи: 26

Перевірив:

Порєв В. М.

Київ 2025

Завдання

- Статичний масив
- **N** = 126 - кількість елементів масиву
- **Гумовий слід:** пунктирна лінія чорного кольору
- **Прямокутник:**

  Малювання від центру до одного з кутів

  Чорний контур прямокутника без заповнення
- **Еліпс:**

  Малювання по двом протилежним кутам охоплюючого прямокутника

  Чорний контур з кольоровим заповненням

  Помаранчевий колір заповнення
- **Лінія з кружечками та каркас куба**
- **Позначка поточного типу об’єкту, що вводиться** в заголовку вікна
- **Обєкт класу MyEditor:** динамічний

Текст головного файлу програми

main.py:

```
import tkinter as tk
from app import DrawingApp

def main():
    root = tk.Tk()
    app = DrawingApp(root)
    app.run()

if __name__ == "__main__":
    main()
```

Тексти модульних файлів програми

app.py:

```
import tkinter as tk
from tkinter import messagebox
from toolbar import Toolbar
from my_editor import MyEditor

class DrawingApp:

    def __init__(self, root):
        self.root = root
        self.base_title = "Лабораторна робота 4. Хубеджева Єлизавета"
        self.root.title(self.base_title)
        self.root.geometry("950x600")

        self.current_tool = 'point'

        self.create_menu()
        self.create_toolbar()
        self.create_canvas()

        self.my_editor = MyEditor(self.canvas, self.update_status)

        self.create_status_bar()

        self.select_tool('point')

        self.canvas.bind("<Button-1>", self.my_editor.on_mouse_down)
        self.canvas.bind("<B1-Motion>", self.my_editor.on_mouse_move)
        self.canvas.bind("<ButtonRelease-1>", self.my_editor.on_mouse_up)

    def create_menu(self):
        menubar = tk.Menu(self.root)
        self.root.config(menu=menubar)
        file_menu = tk.Menu(menubar, tearoff=0)

        menubar.add_cascade(label="Файл", menu=file_menu)
        file_menu.add_command(label="Очистити",
command=self.clear_canvas)
        file_menu.add_separator()
```

```
        file_menu.add_command(label="Вихід", command=self.root.quit)

        self.objects_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Об'єкти", menu=self.objects_menu)

        menu_items = {
            "Точка": "point",
            "Лінія": "line",
            "Прямокутник": "rect",
            "Еліпс": "ellipse",
            "Лінія з кружечками": "line_oo",
            "Куб": "cube"
        }

        for label, tool in menu_items.items():
            self.objects_menu.add_command(
                label=label,
                command=lambda t=tool: self.select_tool(t)
            )

        help_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="Довідка", menu=help_menu)
        help_menu.add_command(label="Про програму",
command=self.show_about)

    def create_toolbar(self):
        self.toolbar = Toolbar(self.root, self.select_tool)

    def create_canvas(self):
        self.canvas = tk.Canvas(
            self.root,
            bg="white",
            cursor="crosshair"
        )
        self.canvas.pack(fill=tk.BOTH, expand=True)

    def create_status_bar(self):
        shape_count = self.my_editor.get_shape_count()
        self.status_bar = tk.Label(
            self.root,
```

```
            text=f"Об'єктів: {shape_count}/{MyEditor.MAX_SHAPES}",
            bd=1,
            relief=tk.SUNKEN,
            anchor=tk.W
        )
        self.status_bar.pack(side=tk.BOTTOM, fill=tk.X)

    def select_tool(self, tool):
        self.current_tool = tool
        self.my_editor.select_tool(tool)
        self.toolbar.set_active_tool(tool)
        self.update_status()

    def clear_canvas(self):
        if self.my_editor.get_shape_count() > 0:
            if messagebox.askyesno("Підтвердження", "Ви впевнені, що
хочете очистити полотно?"):
                self.my_editor.clear()

    def update_status(self):
        shape_count = self.my_editor.get_shape_count()
        self.status_bar.config(
            text=f"Об'єктів: {shape_count}/{MyEditor.MAX_SHAPES}"
        )

        tool_names = {
            'point': 'Точка',
            'line': 'Лінія',
            'rect': 'Прямокутник',
            'ellipse': 'Еліпс',
            'line_oo': 'Лінія з кружечками',
            'cube': 'Куб'
        }
        current_tool_name = tool_names.get(self.current_tool, "Невідомо")
        self.root.title(f"{self.base_title} - [{current_tool_name}]")

    def show_about(self):
        messagebox.showinfo(
            "Про програму",
            "Програма для малювання геометричних фігур\n\n"
```

```
            "Автор: Хубеджева Єлизавета\n\n"
            "Підтримувані фігури:\n"
            "• Точка\n"
            "• Лінія\n"
            "• Прямокутник\n"
            "• Еліпс\n"
            "• Лінія з кружечками\n"
            "• Каркас кубу\n\n"
            "Максимальна кількість об'єктів: 126"
        )

    def run(self):
        self.root.mainloop()
```

toolbar.py:

```
import tkinter as tk

class Tooltip:
    def __init__(self, widget, text):
        self.widget = widget
        self.text = text
        self.tooltip_window = None
        self.widget.bind("<Enter>", self._show_tip)
        self.widget.bind("<Leave>", self._hide_tip)

    def _show_tip(self, event=None):
        x, y, _, _ = self.widget.bbox("insert")
        x += self.widget.winfo_rootx() + 25
        y += self.widget.winfo_rooty() + 25
        self.tooltip_window = tk.Toplevel(self.widget)
        self.tooltip_window.wm_overrideredirect(True)
        self.tooltip_window.wm_geometry(f"+{x}+{y}")
        label = tk.Label(
            self.tooltip_window, text=self.text, justify='left',
            background="#ffffe0", relief='solid', borderwidth=1,
        )
        label.pack(ipadx=1)

    def _hide_tip(self, event=None):
        if self.tooltip_window:
            self.tooltip_window.destroy()
        self.tooltip_window = None

class Toolbar:
    def __init__(self, root, select_tool_callback):
        self.frame = tk.Frame(root, bd=1, relief=tk.RAISED)
        self.select_tool_callback = select_tool_callback
        self.button_images = {}
        self.buttons = {}

        buttons_data = {
            'point': ("./icons/point.png", "Створити точку"),
            'line': ("./icons/line.png", "Намалювати лінію"),
```

```
            'rect': ("./icons/rect.png", "Намалювати прямокутник"),
            'ellipse': ("./icons/ellipse.png", "Намалювати еліпс"),
            'line_oo': ("./icons/line_oo.png", "Намалювати лінію з
кружечками"),
            'cube': ("./icons/cube.png", "Намалювати каркас кубу"),
        }

        img_scale = 22

        for tool, (img_file, tooltip_text) in buttons_data.items():
            try:
                original_img = tk.PhotoImage(file=img_file)
                resized_img = original_img.subsample(img_scale,
img_scale)
                self.button_images[tool] = resized_img

                button = tk.Button(
                    self.frame,
                    image=resized_img,
                    command=lambda t=tool: self.select_tool_callback(t),
                    relief=tk.RAISED
                )
                button.pack(side=tk.LEFT, padx=2, pady=2)
                Tooltip(button, tooltip_text)
                self.buttons[tool] = button
            except tk.TclError:
                print(f"Помилка: Файл зображення '{img_file}' не
знайдено.")

        self.frame.pack(side=tk.TOP, fill=tk.X)

    def set_active_tool(self, active_tool):
        for tool, button in self.buttons.items():
            if tool == active_tool:
                button.config(relief=tk.SUNKEN)
            else:
                button.config(relief=tk.RAISED)
```

my-editor.py:

```
import tkinter as tk
from tkinter import messagebox
from shapes import Shape, PointShape, LineShape, RectShape, EllipseShape, LineOOShape, CubeShape


class MyEditor:
    MAX_SHAPES = 126

    def __init__(self, canvas, status_update_callback):
        self.canvas = canvas
        self.status_update_callback = status_update_callback

        self.shapes = None
        self.shape_count = 0

        self.current_tool = 'point'
        self.is_drawing = False
        self.start_x = 0
        self.start_y = 0

        self.temp_shape = None

    def _ensure_array_exists(self):
        if self.shapes is None:
            self.shapes = [None] * self.MAX_SHAPES

    def __del__(self):
        del self.shapes

    def select_tool(self, tool):
        self.current_tool = tool

    def on_mouse_down(self, event):
        self._ensure_array_exists()

        if self.shape_count >= self.MAX_SHAPES:
            messagebox.showwarning("Увага", "Досягнуто максимальну кількість об'єктів (126)")
            return
```

```
        self.is_drawing = True
        self.start_x = event.x
        self.start_y = event.y

    def on_mouse_move(self, event):
        if not self.is_drawing:
            return

        self.canvas.delete("preview")
        self.temp_shape = self._create_shape_instance(event)

        if self.temp_shape:
            self.temp_shape.show(self.canvas, is_preview=True)

    def on_mouse_up(self, event):
        if not self.is_drawing:
            return

        self.is_drawing = False
        self.canvas.delete("preview")

        final_shape = self._create_shape_instance(event)

        if final_shape:
            self.add_shape(final_shape)

        self.temp_shape = None

    def _create_shape_instance(self, event):
        x1, y1 = self.start_x, self.start_y
        x2, y2 = event.x, event.y

        if self.current_tool == 'point':
            return PointShape(x1, y1, x1, y1)
        elif self.current_tool == 'line':
            return LineShape(x1, y1, x2, y2)
        elif self.current_tool == 'rect':
            return RectShape(x1, y1, x2, y2)
        elif self.current_tool == 'ellipse':
```

```
            return EllipseShape(x1, y1, x2, y2)
        elif self.current_tool == 'line_oo':
            return LineOOShape(x1, y1, x2, y2)
        elif self.current_tool == 'cube':
            return CubeShape(x1, y1, x2, y2)
        return None

    def add_shape(self, shape):
        self._ensure_array_exists()

        if self.shape_count < self.MAX_SHAPES:
            self.shapes[self.shape_count] = shape
            self.shape_count += 1
            shape.show(self.canvas)
            self.status_update_callback()

    def get_shape_count(self):
        if self.shapes is None:
            return 0
        return self.shape_count

    def clear(self):
        self.canvas.delete("shape")
        self.shapes = None
        self.shape_count = 0
        self.status_update_callback()
```

shapes.py:

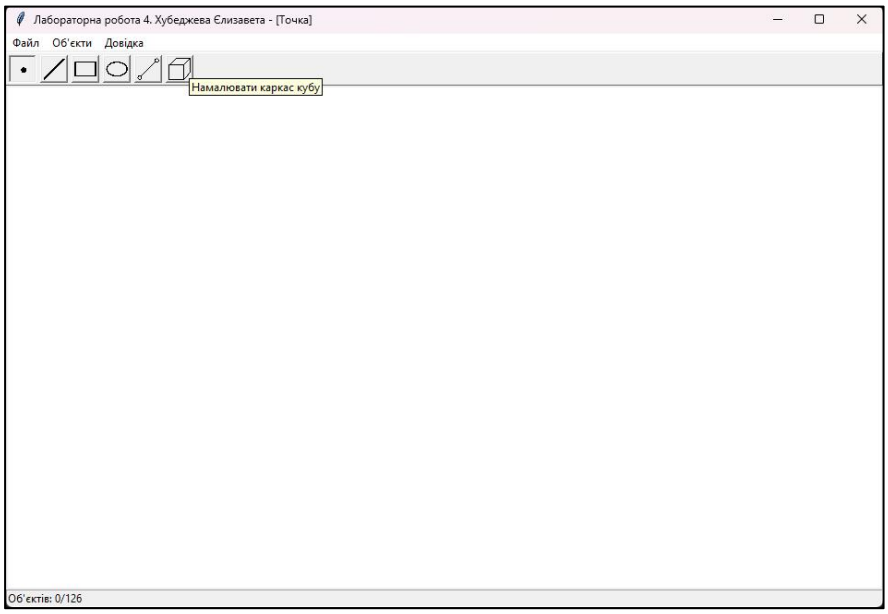
```
from abc import ABC, abstractmethod

class Shape(ABC):
    def __init__(self, x1, y1, x2, y2, color="black", width=2):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2
        self.color = color
        self.width = width

    @abstractmethod
    def show(self, canvas, is_preview=False):
        pass

    def _get_style(self, is_preview, default_style):
        if is_preview:
            return {"outline": "black", "fill": None, "dash": (4, 2)}
        return default_style

    def _get_tags(self, is_preview):
        return "preview" if is_preview else "shape"

class PointShape(Shape):
    def show(self, canvas, is_preview=False):
        radius = 3

        style = self._get_style(is_preview, {
            "fill": self.color,
            "outline": self.color
        })

        canvas.create_oval(
            self.x1 - radius, self.y1 - radius,
            self.x1 + radius, self.y1 + radius,
            **style,
            tags=self._get_tags(is_preview)
        )
```

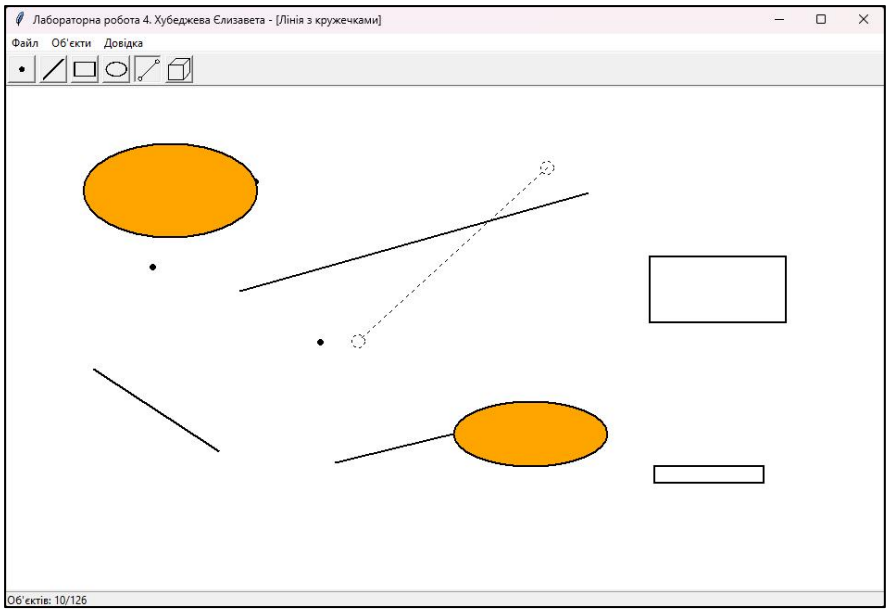
```
class LineShape(Shape):
    def show(self, canvas, is_preview=False):
        style = self._get_style(is_preview, {
            "fill": self.color,
            "width": self.width
        })

        if is_preview:
            if "outline" in style:
                style["fill"] = style.pop("outline")
            style["width"] = 1

        canvas.create_line(
            self.x1, self.y1, self.x2, self.y2,
            **style,
            tags=self._get_tags(is_preview)
        )

class RectShape(Shape):
    def show(self, canvas, is_preview=False):
        style = self._get_style(is_preview, {
            "fill": None,
            "outline": self.color,
            "width": self.width
        })

        canvas.create_rectangle(
            self.x1, self.y1, self.x2, self.y2,
            **style,
            tags=self._get_tags(is_preview)
        )

class EllipseShape(Shape):
    def show(self, canvas, is_preview=False):
        style = self._get_style(is_preview, {
            "fill": "orange",
            "outline": self.color,
            "width": self.width
        })
```

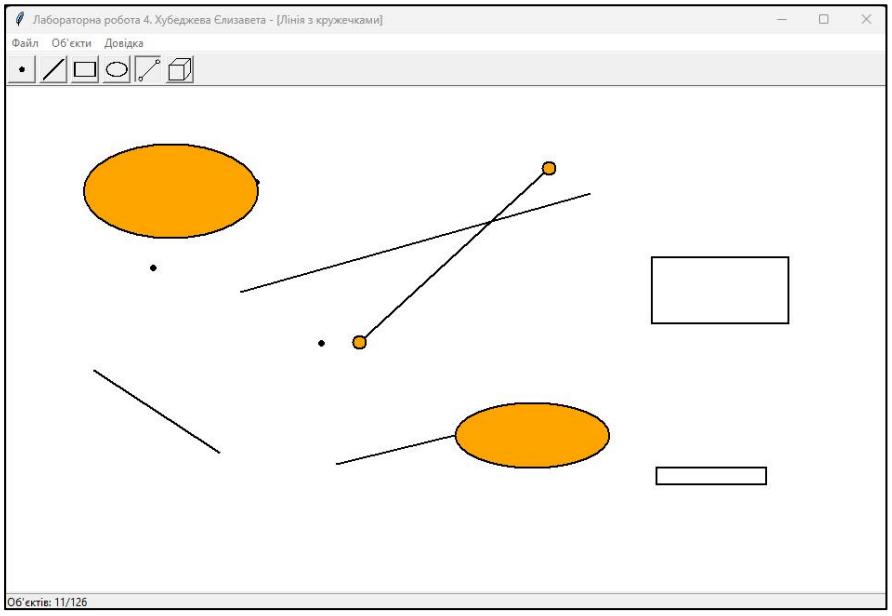
```
        canvas.create_oval(
            self.x1, self.y1, self.x2, self.y2,
            **style,
            tags=self._get_tags(is_preview)
        )

class LineOOShape(LineShape, EllipseShape):

    CIRCLE_RADIUS = 7

    def __init__(self, x1, y1, x2, y2):
        Shape.__init__(self, x1, y1, x2, y2)
        self.radius = self.CIRCLE_RADIUS

    def show(self, canvas, is_preview=False):
        LineShape.show(self, canvas, is_preview)
        orig_x1, orig_y1, orig_x2, orig_y2 = self.x1, self.y1, self.x2,
self.y2

        self.x1, self.y1 = orig_x1 - self.radius, orig_y1 - self.radius
        self.x2, self.y2 = orig_x1 + self.radius, orig_y1 + self.radius
        EllipseShape.show(self, canvas, is_preview)

        self.x1, self.y1 = orig_x2 - self.radius, orig_y2 - self.radius
        self.x2, self.y2 = orig_x2 + self.radius, orig_y2 + self.radius
        EllipseShape.show(self, canvas, is_preview)

        self.x1, self.y1, self.x2, self.y2 = orig_x1, orig_y1, orig_x2,
orig_y2

class CubeShape(RectShape, LineShape):

    PERSPECTIVE_RATIO = 0.4

    def __init__(self, x1, y1, x2, y2):
        Shape.__init__(self, x1, y1, x2, y2, color="blue")

    def show(self, canvas, is_preview=False):
        orig_x1, orig_y1 = self.x1, self.y1
        orig_x2, orig_y2 = self.x2, self.y2
```

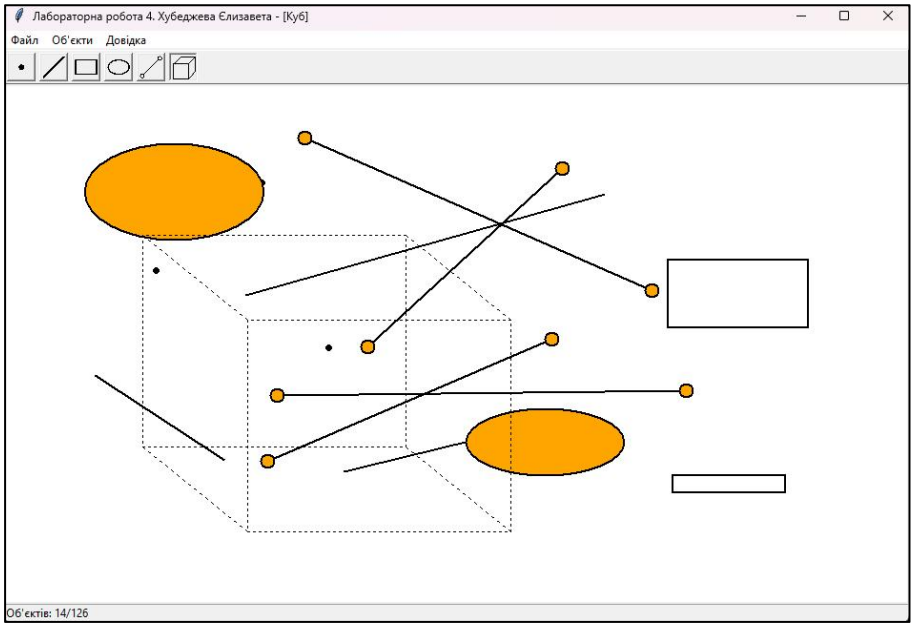
```
RectShape.show(self, canvas, is_preview)

dx = (self.x2 - self.x1) * self.PERSPECTIVE_RATIO
dy = (self.y2 - self.y1) * self.PERSPECTIVE_RATIO

self.x1, self.y1 = orig_x1 + dx, orig_y1 - dy
self.x2, self.y2 = orig_x2 + dx, orig_y2 - dy
RectShape.show(self, canvas, is_preview)

self.x1, self.y1 = orig_x1, orig_y1
self.x2, self.y2 = orig_x1 + dx, orig_y1 - dy
LineShape.show(self, canvas, is_preview)

self.x1, self.y1 = orig_x2, orig_y1
self.x2, self.y2 = orig_x2 + dx, orig_y1 - dy
LineShape.show(self, canvas, is_preview)

self.x1, self.y1 = orig_x1, orig_y2
self.x2, self.y2 = orig_x1 + dx, orig_y2 - dy
LineShape.show(self, canvas, is_preview)

self.x1, self.y1 = orig_x2, orig_y2
self.x2, self.y2 = orig_x2 + dx, orig_y2 - dy
LineShape.show(self, canvas, is_preview)

self.x1, self.y1 = orig_x1, orig_y1
self.x2, self.y2 = orig_x2, orig_y2
```

Скріншоти:


Лабораторна робота 4. Хубеджева Єлизавета - [Точка]
Файл Об'єкти Довідка
Намалювати каркас кубу
Об'єктів: 0/126


Лабораторна робота 4. Хубеджева Єлизавета - [Лінія з кружечками]
Файл Об'єкти Довідка
Об'єктів: 10/126


Лабораторна робота 4. Хубеджева Єлизавета - [Лінія з кружечками]
Файл Об'єкти Довідка
Об'єктів: 11/126


Лабораторна робота 4. Хубеджева Єлизавета - [Куб]
Файл
Об'єкти
Довідка
Об'єктів: 14/126

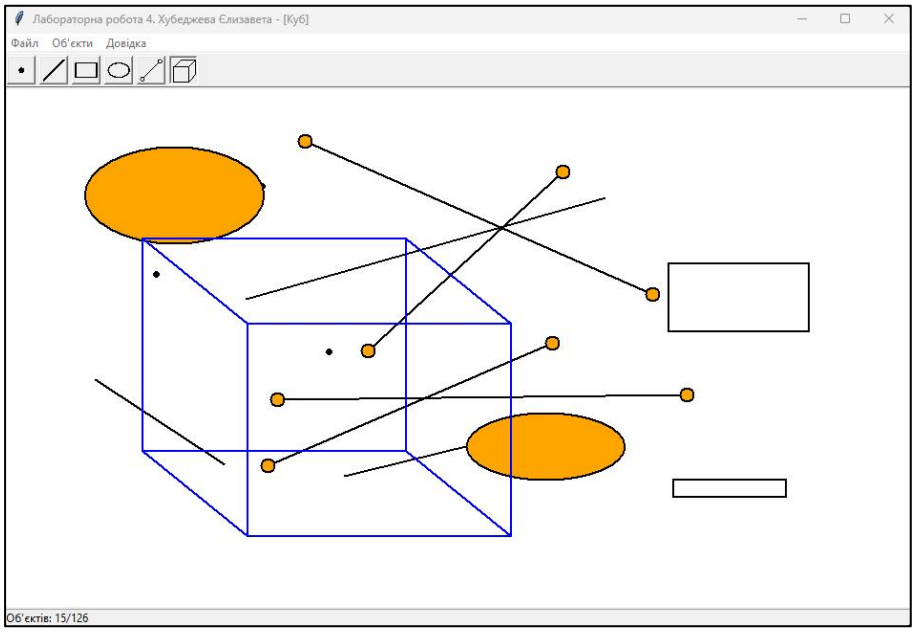
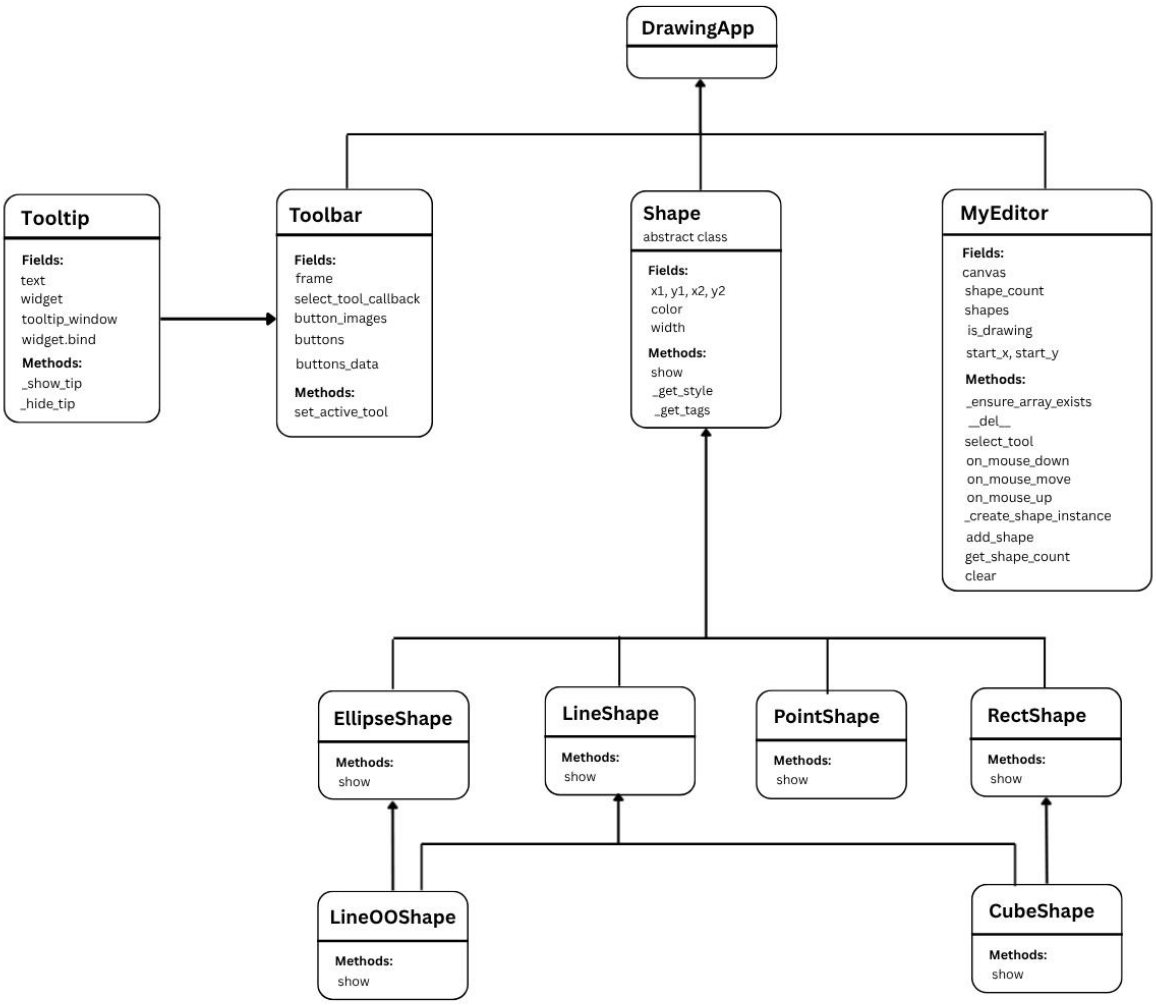

Лабораторна робота 4. Хубеджева Єлизавета - [Куб]
Файл
Об'єкти
Довідка
Об'єктів: 15/126

Схема успадкування класів:



Висновок

У ході виконання лабораторної продовжено розробку програми - графічний редактор для малювання простих геометричних фігур (точка, лінія, прямокутник, еліпс) на мові програмування Python з використанням бібліотеки Tkinter для побудови графічного інтерфейсу користувача.

Замість абстрактного класу Editor та залежних від нього класів ShapeEditor, PointEditor, LineEditor, RectEditor, EllipseEditor було запрограмовано єдиний динамічний клас MyEditor, що інкапсулює масив об'єктів типу Shape та містить інтерфейсні public функції-члени, які повинні реалізовувати основні функції графічного редактора.

Було додано дві нові фігури: LineOOShape (лінія з кружечками), що є похідним від класів LineShape й EllipseShape, та CubeShape (каркас куба), що є похідним від класів LineShape й RectShape. Тобто було використано множинне успадкування класів.

Також до Toolbar було додано ще дві кнопки для нових фігур.

У реалізації дотримано принципів об’єктно-орієнтованого програмування: інкапсуляції, абстракції, наслідування, поліморфізму та множинного успадкування.

Клас Shape визначено як абстрактний базовий клас із полями для координат, а також абстрактним методом show, який реалізують похідні класи PointShape, LineShape, RectShape та EllipseShape.

Клас DrawingApp відповідає за інтерфейс і взаємодію: створення меню, полотна (canvas), статус-бару. Також передбачено очищення полотна та інформаційне діалогове вікно «Про програму».