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»

Виконав:
студент групи ІМ-43
Олексійчук Станіслав Юрійович
номер у списку групи: 20

Перевірив:
Порєв В. М.

Київ 2025

Варіант завдання та основні вимоги

Варіант 20:

1. Для усіх варіантів завдань необхідно дотримуватися вимог та положень, викладених вище у порядку виконання роботи та методичних рекомендаціях.

2. Номер варіанту завдання дорівнює номеру зі списку студентів у журналі.

Студенти з парним номером (2, 4, 6, . . .) програмують динамічний обєкт класу MyEditor, забезпечивши коректне його створення та знищення.

3. Усі кольори та стилі (за винятком "гумового" сліду) геометричних форм – як у попередній лабораторній роботі №3. "Гумовий" слід при вводі усіх фігур малювати пунктирною лінією.

4. Окрім чотирьох типів фігур, які були у попередніх лаб. No2 та 3, запрограмувати ще введення та відображення двох нових фігур – лінія з кружечками та каркас куба.

Кольори ліній та заповнення цих нових фігур студент визначає на свій розсуд. Для об’єктів типів лінії з кружечками та каркасу кубу відповідні класи запрограмувати саме множинним успадкуванням. У цій лабораторній роботі не дозволяється замінювати множинне спадкування, наприклад, композицією. У першу чергу це стосується метода Show для нових фігур – для відображення ліній треба використовувати виклики метода Show з класу LineShape, для відображення кружечків – виклики метода Show з класу EllipseShape, а для відображення прямокутників – виклики метода Show з класу RectShape.

5. Для усіх шести типів форм зробити кнопки Toolbar з підказками (tooltips)

6. У звіті повинна бути схема успадкування класів – діаграма класів.

Текст програми

Лабораторна робота виконана мовою програмування Python з використанням бібліотеки Tkinter. Це завдання складається з головного файлу та 4 модулів:

1) main.py

```
import tkinter as tk
from tkinter import messagebox
from my_editor import MyEditor
from shape import PointShape, LineShape, RectShape, EllipseShape, LineSegmentShape,
CubeShape
from toolbar import Toolbar

class Main:
    def __init__(self):
        self._main_window = None
        self._canvas = None
        self._editor_manager = None
        self._current_mode = None
        self._menu_bar = None
        self._toolbar = None

    def run(self):
        self._create_window()
        self._create_canvas()
        self._create_editor()
        self._create_menu()
        self._create_toolbar()
        self._bind_events()
        self._set_default_mode()
        self._main_window.mainloop()

    def _create_window(self):
        self._main_window = tk.Tk()
        self._main_window.geometry("600x400+400+150")
        self._main_window.title("Lab3")
        self._main_window.resizable(False, False)

    def _create_canvas(self):
        self._canvas = tk.Canvas(self._main_window, width=600, height=400, bg="white")
        self._canvas.pack()

    def _create_editor(self):
        self._editor_manager = MyEditor(self._canvas)

    def _create_menu(self):
        self._current_mode = tk.StringVar(value="Point")
        self._menu_bar = tk.Menu(self._main_window)
```

```
        self._create_file_menu()
        self._create_objects_menu()
        self._create_help_menu()

        self._main_window.config(menu=self._menu_bar)

    def _create_file_menu(self):
        file_menu = tk.Menu(self._menu_bar, tearoff=0)
        file_menu.add_command(label="Exit", command=self._main_window.quit)
        self._menu_bar.add_cascade(label="File", menu=file_menu)

    def _create_toolbar(self):
        self._toolbar = Toolbar(self._main_window, self._canvas, self,
self._current_mode)
        self._toolbar.create_toolbar()

    def _create_objects_menu(self):
        objects_menu = tk.Menu(self._menu_bar, tearoff=0)

        objects_menu.add_radiobutton(
            label="Крапка",
            command=self.switch_to_point_mode,
            variable=self._current_mode,
            value="Point"
        )
        objects_menu.add_radiobutton(
            label="Лінія",
            command=self.switch_to_line_mode,
            variable=self._current_mode,
            value="Line"
        )
        objects_menu.add_radiobutton(
            label="Прямокутник",
            command=self.switch_to_rect_mode,
            variable=self._current_mode,
            value="Rectangle"
        )
        objects_menu.add_radiobutton(
            label="Еліпс",
            command=self.switch_to_ellipse_mode,
            variable=self._current_mode,
            value="Ellipse"
        )
        objects_menu.add_radiobutton(
            label="Відрізок",
            command=self.switch_to_line_segment_mode,
            variable=self._current_mode,
            value="Line Segment"
        )
```

```
        objects_menu.add_radiobutton(
            label="Куб",
            command=self.switch_to_cube_mode,
            variable=self._current_mode,
            value="Cube"
        )

        self._menu_bar.add_cascade(label="Objects", menu=objects_menu)

    def _create_help_menu(self):
        help_menu = tk.Menu(self._menu_bar, tearoff=0)
        help_menu.add_command(label="About", command=self._show_lab_information)
        self._menu_bar.add_cascade(label="Help", menu=help_menu)

    def switch_to_point_mode(self):
        self._editor_manager.start(PointShape())
        self._current_mode.set("Point")
        self._main_window.title(f"Lab4 - {self._current_mode.get()} Mode")
        self._toolbar.update_button_states()

    def switch_to_line_mode(self):
        self._editor_manager.start(LineShape())
        self._current_mode.set("Line")
        self._main_window.title(f"Lab4 - {self._current_mode.get()} Mode")
        self._toolbar.update_button_states()

    def switch_to_rect_mode(self):
        self._editor_manager.start(RectShape())
        self._current_mode.set("Rectangle")
        self._main_window.title(f"Lab4 - {self._current_mode.get()} Mode")
        self._toolbar.update_button_states()

    def switch_to_ellipse_mode(self):
        self._editor_manager.start(EllipseShape())
        self._current_mode.set("Ellipse")
        self._main_window.title(f"Lab4 - {self._current_mode.get()} Mode")
        self._toolbar.update_button_states()

    def switch_to_line_segment_mode(self):
        self._editor_manager.start(LineSegmentShape())
        self._current_mode.set("Line Segment")
        self._main_window.title(f"Lab4 - {self._current_mode.get()} Mode")
        self._toolbar.update_button_states()

    def switch_to_cube_mode(self):
        self._editor_manager.start(CubeShape())
        self._current_mode.set("Cube")
        self._main_window.title(f"Lab4 - {self._current_mode.get()} Mode")
        self._toolbar.update_button_states()
```

```
    def _show_lab_information(self):
        messagebox.showinfo("Info", "Lab 4 is working fine\n(c) Copyright 2025")

    def _bind_events(self):
        self._canvas.bind("<Button-1>", self._on_click)
        self._canvas.bind("<B1-Motion>", self._on_drag)
        self._canvas.bind("<ButtonRelease-1>", self._on_drop)

    def _on_click(self, event):
        self._editor_manager.on_click(event.x, event.y)

    def _on_drag(self, event):
        self._editor_manager.on_drag(event.x, event.y)

    def _on_drop(self, event):
        self._editor_manager.on_drop(event.x, event.y)

    def _set_default_mode(self):
        self.switch_to_point_mode()

if __name__ == "__main__":
    app = Main()
    app.run()
```

2) shape.py

```
from abc import ABC, abstractmethod

class Shape(ABC):
    def __init__(self):
        self.x1 = 0
        self.y1 = 0
        self.x2 = 0
        self.y2 = 0

    def set(self, x1, y1, x2, y2):
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

    @abstractmethod
    def show(self, canvas):
        pass

    @abstractmethod
    def show_preview(self, canvas):
        pass


class PointShape(Shape):
    def show(self, canvas):
```

```
        canvas.create_oval(self.x1-2, self.y1-2,
                           self.x1+2, self.y1+2,
                           fill="black", outline="black")

    def show_preview(self, canvas):
        return canvas.create_oval(
            self.x1-3, self.y1-3, self.x1+3, self.y1+3, outline="red", dash=(4, 2))

class LineShape(Shape):
    def show(self, canvas, fill_color="black", width=2):
        canvas.create_line(self.x1, self.y1,
                           self.x2, self.y2,
                           fill=fill_color, width=width)

    def show_preview(self, canvas, fill_color="red", width=2, dash=(4, 2)):
        return canvas.create_line(
            self.x1, self.y1, self.x2, self.y2,
            fill=fill_color, width=width, dash=dash)

class RectShape(Shape):
    def __init__(self):
        super().__init__()
        self._fill_color = "pink"

    def show(self, canvas, outline_color="black", width=2, fill_color=None):
        fill = fill_color if fill_color is not None else self._fill_color
        canvas.create_rectangle(self.x1, self.y1, self.x2, self.y2,
                                outline=outline_color, fill=fill, width=width)

    def show_preview(self, canvas, outline_color="red", width=2, dash=(4, 2)):
        return canvas.create_rectangle(self.x1, self.y1, self.x2, self.y2,
                                       outline=outline_color, width=width, dash=dash)

class EllipseShape(Shape):
    def __init__(self):
        super().__init__()
        self._fill_color = "white"

    def show(self, canvas, outline_color="black", width=2, fill_color=None):
        center_x = self.x1
        center_y = self.y1
        current_x = self.x2
        current_y = self.y2

        x1 = 2 * center_x - current_x
        y1 = 2 * center_y - current_y
        x2 = current_x
        y2 = current_y

        fill = fill_color if fill_color is not None else self._fill_color
```

```
        canvas.create_oval(x1, y1,
                           x2, y2,
                           outline=outline_color, fill=fill, width=width)

    def show_preview(self, canvas, outline_color="red", width=2, dash=(4, 2)):
        center_x = self.x1
        center_y = self.y1
        current_x = self.x2
        current_y = self.y2

        x1 = 2 * center_x - current_x
        y1 = 2 * center_y - current_y
        x2 = current_x
        y2 = current_y

        return canvas.create_oval(x1, y1, x2, y2,
                                  outline=outline_color, width=width, dash=dash)


class LineSegmentShape(LineShape, EllipseShape):
    def __init__(self):
        LineShape.__init__(self)
        EllipseShape.__init__(self)
        self._line_color = "black"
        self._radius = 3

    def _calculate_ellipses(self):
        ellipse1_coords = (self.x1 - self._radius, self.y1 - self._radius, self.x1 +
self._radius, self.y1 + self._radius)
        ellipse2_coords = (self.x2 - self._radius, self.y2 - self._radius, self.x2 +
self._radius, self.y2 + self._radius)
        return ellipse1_coords, ellipse2_coords

    def show(self, canvas):
        original_coords = (self.x1, self.y1, self.x2, self.y2)

        LineShape.show(self, canvas, fill_color=self._line_color)

        ellipse1_coords, ellipse2_coords = self._calculate_ellipses()

        temp_ellipse = EllipseShape()

        x1_o, y1_o, x2_o, y2_o = ellipse1_coords
        center_x_1 = (x1_o + x2_o) / 2
        center_y_1 = (y1_o + y2_o) / 2
        temp_ellipse.set(center_x_1, center_y_1, x2_o, y2_o)
        temp_ellipse.show(canvas, outline_color=self._line_color, fill_color="white")

        x1_o, y1_o, x2_o, y2_o = ellipse2_coords
```

```
        center_x_2 = (x1_o + x2_o) / 2
        center_y_2 = (y1_o + y2_o) / 2
        temp_ellipse.set(center_x_2, center_y_2, x2_o, y2_o)
        temp_ellipse.show(canvas, outline_color=self._line_color, fill_color="white")

        self.set(*original_coords)

    def show_preview(self, canvas):
        preview_ids = []
        original_coords = (self.x1, self.y1, self.x2, self.y2)

        line_id = LineShape.show_preview(self, canvas, fill_color="red")
        preview_ids.append(line_id)

        ellipse1_coords, ellipse2_coords = self._calculate_ellipses()

        temp_ellipse = EllipseShape()

        x1_o, y1_o, x2_o, y2_o = ellipse1_coords
        center_x_1 = (x1_o + x2_o) / 2
        center_y_1 = (y1_o + y2_o) / 2
        temp_ellipse.set(center_x_1, center_y_1, x2_o, y2_o)
        preview_ids.append(temp_ellipse.show_preview(canvas, outline_color="red"))

        x1_o, y1_o, x2_o, y2_o = ellipse2_coords
        center_x_2 = (x1_o + x2_o) / 2
        center_y_2 = (y1_o + y2_o) / 2
        temp_ellipse.set(center_x_2, center_y_2, x2_o, y2_o)
        preview_ids.append(temp_ellipse.show_preview(canvas, outline_color="red"))

        self.set(*original_coords)

        return preview_ids

class CubeShape(LineShape, RectShape):
    def __init__(self):
        LineShape.__init__(self)
        RectShape.__init__(self)
        self._cube_color = "black"
        self._depth_ratio = 0.3

    def _calculate_cube_points(self):
        x1_start, y1_start = min(self.x1, self.x2), min(self.y1, self.y2)
        x2_end, y2_end = max(self.x1, self.x2), max(self.y1, self.y2)

        width = x2_end - x1_start
        height = y2_end - y1_start
        depth = int(min(width, height) * self._depth_ratio)

        front_face_coords = (x1_start, y1_start, x2_end, y2_end)
```

```
        x1_back = x1_start + depth
        y1_back = y1_start - depth
        x2_back = x2_end + depth
        y2_back = y2_end - depth

        edges = [
            (x1_back, y1_back, x2_back, y1_back),
            (x2_back, y1_back, x2_back, y2_back),
            (x2_back, y2_back, x1_back, y2_back),
            (x1_back, y2_back, x1_back, y1_back),

            (x1_start, y1_start, x1_back, y1_back),
            (x2_end, y1_start, x2_back, y1_back),
            (x2_end, y2_end, x2_back, y2_back),
            (x1_start, y2_end, x1_back, y2_back)

        ]
        return front_face_coords, edges

    def show(self, canvas):
        front_face_coords, edges = self._calculate_cube_points()
        original_coords = (self.x1, self.y1, self.x2, self.y2)

        for edge in edges:
            self.set(edge[0], edge[1], edge[2], edge[3])
            LineShape.show(self, canvas, fill_color=self._cube_color)

        temp_rect = RectShape()
        temp_rect.set(*front_face_coords)
        temp_rect.show(canvas, outline_color=self._cube_color, fill_color="")

        self.set(*original_coords)

    def show_preview(self, canvas):
        preview_ids = []
        front_face_coords, edges = self._calculate_cube_points()
        original_coords = (self.x1, self.y1, self.x2, self.y2)

        for edge in edges:
            self.set(edge[0], edge[1], edge[2], edge[3])
            preview_ids.append(LineShape.show_preview(self, canvas, fill_color="red"))

        temp_rect = RectShape()
        temp_rect.set(*front_face_coords)
        preview_ids.append(temp_rect.show_preview(canvas, outline_color="red"))

        self.set(*original_coords)

        return preview_ids
```

3) my_editor.py

```
class MyEditor:
    def __init__(self, canvas):
        self._canvas = canvas
        self._shapes = []
        self._current_shape = None
        self._preview_id = None
        self._start_x = 0
        self._start_y = 0

    def start(self, shape):
        self._current_shape = shape

    def on_click(self, x, y):
        self._start_x = x
        self._start_y = y
        if self._current_shape:
            self._current_shape.set(x, y, x, y)
            self._draw_preview()

    def on_drag(self, x, y):
        if self._current_shape:
            self._current_shape.set(self._start_x, self._start_y, x, y)
            self._update_preview()

    def on_drop(self, x, y):
        if self._current_shape:
            self._current_shape.set(self._start_x, self._start_y, x, y)
            self._shapes.append(self._current_shape)
            self._clear_preview()
            self._redraw()
            self._current_shape = type(self._current_shape)()

    def _draw_preview(self):
        if self._current_shape and not self._preview_id:
            preview_result = self._current_shape.show_preview(self._canvas)
            if isinstance(preview_result, list):
                self._preview_id = preview_result
            else:
                self._preview_id = [preview_result]

    def _update_preview(self):
        if self._current_shape:
            self._clear_preview()
            self._draw_preview()

    def _clear_preview(self):
        if self._preview_id:
            for preview_id in self._preview_id:
                self._canvas.delete(preview_id)
```

```
            self._preview_id = None

    def _redraw(self):
        self._canvas.delete("all")
        for shape in self._shapes:
            shape.show(self._canvas)
```

4) bitmap_factory.py

```
from PIL import Image, ImageDraw, ImageTk

class BitmapFactory:
    def __init__(self):
        self._images = {}

    def create_toolbar_bitmaps(self, size=28):
        center = size // 2

        self._images["Крапка"] = self._create_point_bitmap(size, center)
        self._images["Лінія"] = self._create_line_bitmap(size, center)
        self._images["Прямокутник"] = self._create_rectangle_bitmap(size, center)
        self._images["Еліпс"] = self._create_ellipse_bitmap(size, center)
        self._images["Відрізок"] = self._create_line_segment_bitmap(size, center)
        self._images["Куб"] = self._create_cube_bitmap(size, center)

        return self._images

    def _create_point_bitmap(self, size, center):
        point_img = Image.new("RGB", (size, size), "white")
        point_draw = ImageDraw.Draw(point_img)

        point_draw.ellipse([center-4, center-4, center+4, center+4],
                           fill="black", outline="black", width=1)
        return ImageTk.PhotoImage(point_img)

    def _create_line_bitmap(self, size, center):
        line_img = Image.new("RGB", (size, size), "white")
        line_draw = ImageDraw.Draw(line_img)

        line_draw.line([center-7, center+7, center+7, center-7],
                       fill="black", width=2)
        return ImageTk.PhotoImage(line_img)

    def _create_rectangle_bitmap(self, size, center):
        rectangle_img = Image.new("RGB", (size, size), "white")
        rectangle_draw = ImageDraw.Draw(rectangle_img)

        rectangle_draw.rectangle([center-7, center-4, center+7, center+6],
                       fill="black", outline="black", width=1)
        return ImageTk.PhotoImage(rectangle_img)

    def _create_ellipse_bitmap(self, size, center):
```

```
        ellipse_img = Image.new("RGB", (size, size), "white")
        ellipse_draw = ImageDraw.Draw(ellipse_img)

        ellipse_draw.ellipse([center-7, center-4, center+7, center+4],
                        fill="black", outline="black", width=1)
        return ImageTk.PhotoImage(ellipse_img)

    def _create_line_segment_bitmap(self, size, center):
        line_segment_img = Image.new("RGB", (size, size), "white")
        line_segment_draw = ImageDraw.Draw(line_segment_img)

        line_segment_draw.ellipse([center-9, center+3, center-3, center+9],
fill="white", outline="black", width=2)
        line_segment_draw.line([center-4, center+4, center+6, center-6],
                        fill="black", width=2)
        line_segment_draw.ellipse([center+3, center-9, center+9, center-3],
fill="white", outline="black", width=2)
        return ImageTk.PhotoImage(line_segment_img)

    def _create_cube_bitmap(self, size, center):
        cube_img = Image.new("RGB", (size, size), "white")
        cube_draw = ImageDraw.Draw(cube_img)

        cube_draw.rectangle([center-6, center-4, center+2, center+4], outline="black",
width=1)
        cube_draw.rectangle([center-2, center-6, center+6, center+2], outline="black",
width=1)
        cube_draw.line([center-6, center-4, center-2, center-6], fill="black",
width=1)
        cube_draw.line([center+2, center-4, center+6, center-6], fill="black",
width=1)
        cube_draw.line([center-6, center+4, center-2, center+2], fill="black",
width=1)
        cube_draw.line([center+2, center+4, center+6, center+2], fill="black",
width=1)

        return ImageTk.PhotoImage(cube_img)

    def get_image(self, name):
        return self._images.get(name)

    def get_all_images(self):
        return self._images
```

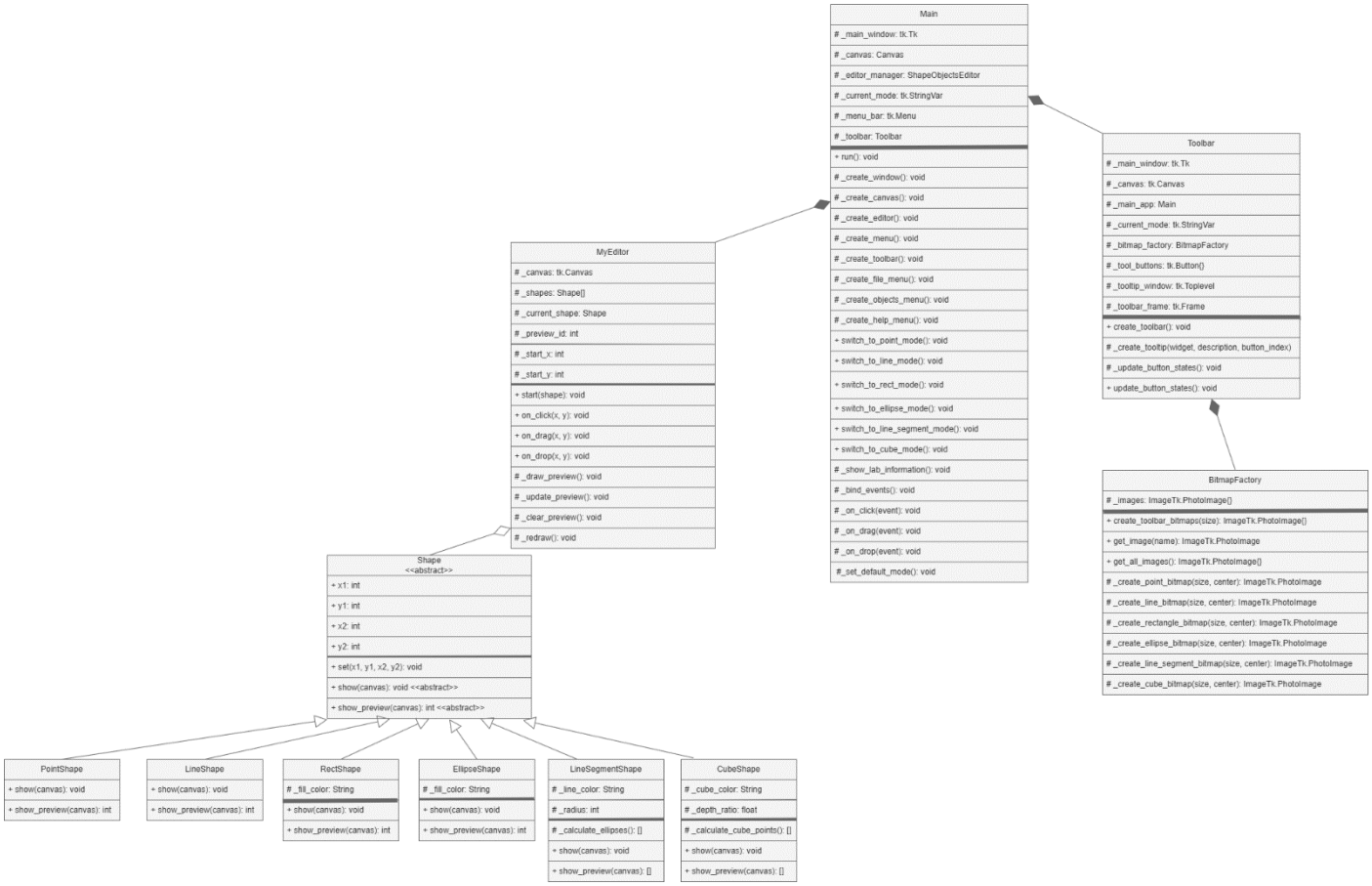
5) toolbar.py

```
import tkinter as tk
from bitmap_factory import BitmapFactory

class Toolbar:
    def __init__(self, main_window, canvas, main_app, current_mode):
        self._main_window = main_window
```

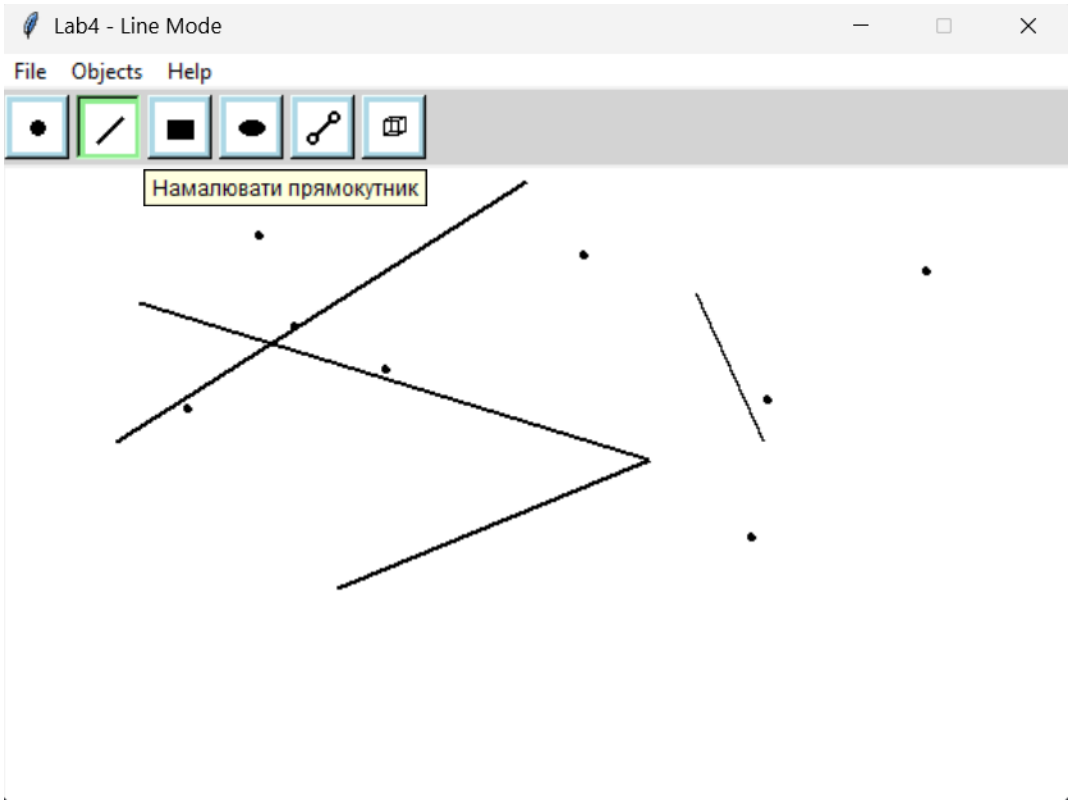
```
        self._canvas = canvas
        self._main_app = main_app
        self._current_mode = current_mode
        self._bitmap_factory = BitmapFactory()
        self._tool_buttons = {}
        self._tooltip_window = None
        self._toolbar_frame = None

    def create_toolbar(self):
        self._toolbar_frame = tk.Frame(self._main_window, bg="lightgray", height=42)
        self._toolbar_frame.pack(fill="x", side="top", before=self._canvas)
        self._toolbar_frame.pack_propagate(False)

        images = self._bitmap_factory.create_toolbar_bitmaps()

        tools = [
            ("Крапка", self._main_app.switch_to_point_mode, "Намалювати крапку"),
            ("Лінія", self._main_app.switch_to_line_mode, "Намалювати лінію"),
            ("Прямокутник", self._main_app.switch_to_rect_mode, "Намалювати
прямокутник"),
            ("Еліпс", self._main_app.switch_to_ellipse_mode, "Намалювати еліпс"),
            ("Відрізок", self._main_app.switch_to_line_segment_mode, "Намалювати
відрізок"),
            ("Куб", self._main_app.switch_to_cube_mode, "Намалювати куб")
        ]

        for i, (name, command, tooltip) in enumerate(tools):
            button = tk.Button(
                self._toolbar_frame,
                image=images[name],
                command=command,
                relief="raised",
                bg="lightblue",
                width=30,
                height=30
            )

            button.pack(side="left", padx=2, pady=2)
            self._tool_buttons[name] = button

            self._create_tooltip(button, tooltip, i)

    def _create_tooltip(self, widget, description, button_index):
        def show_tooltip(event):
            if self._tooltip_window:
                self._tooltip_window.destroy()

            self._tooltip_window = tk.Toplevel(self._main_window)
            self._tooltip_window.wm_overrideredirect(True)
```

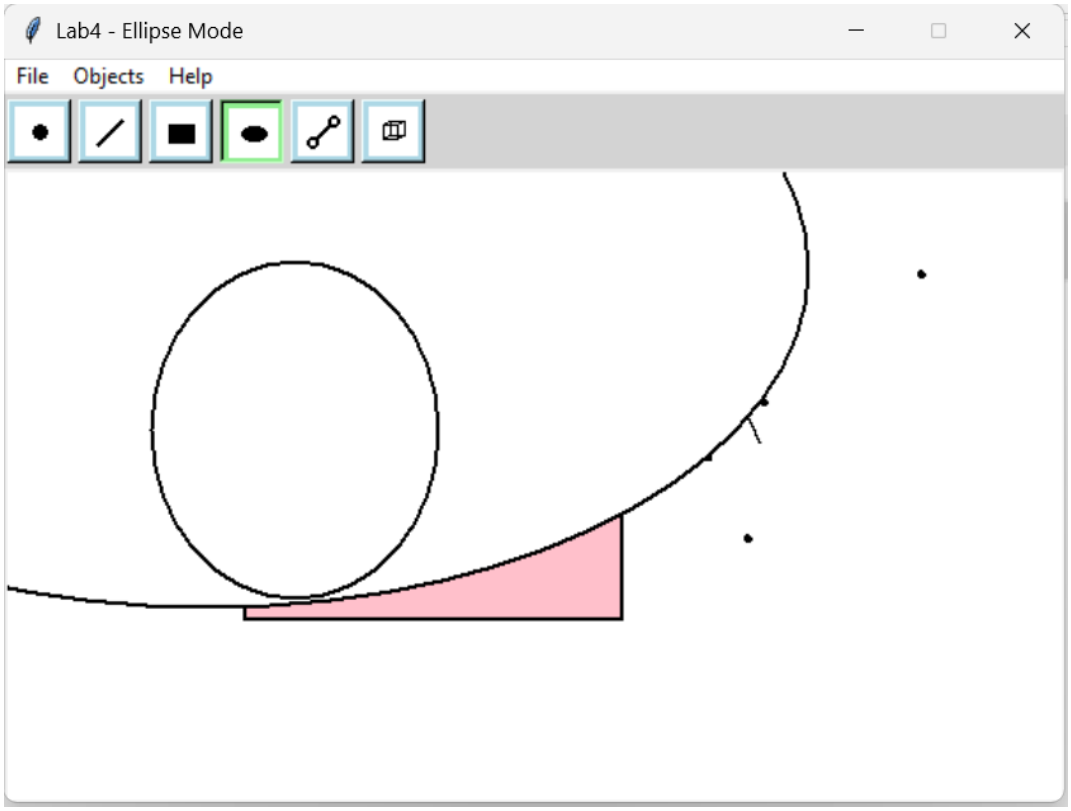
```
            base_x = self._main_window.winfo_rootx() + 10
            base_y = self._main_window.winfo_rooty() + 45

            screen_width = self._main_window.winfo_screenwidth()
            tooltip_x = min(base_x + (button_index * 35), screen_width - 200)
            tooltip_y = base_y

            self._tooltip_window.wm_geometry(f"+{tooltip_x}+{tooltip_y}")

            label = tk.Label(self._tooltip_window, text=description,
                             bg="lightyellow",
                             relief="solid",
                             borderwidth=1,
                             font=("Arial", 9),
                             padx=4, pady=2)
            label.pack()

        def hide_tooltip(event):
            if self._tooltip_window:
                self._tooltip_window.destroy()
                self._tooltip_window = None

        widget.bind("<Enter>", show_tooltip)
        widget.bind("<Leave>", hide_tooltip)
        widget.bind("<ButtonPress>", hide_tooltip)

    def _update_button_states(self):
        mode_mapping = {
            "Point": "Крапка",
            "Line": "Лінія",
            "Rectangle": "Прямокутник",
            "Ellipse": "Еліпс",
            "Line Segment": "Відрізок",
            "Cube": "Куб"
        }

        current_button_name = mode_mapping.get(self._current_mode.get())

        for name, button in self._tool_buttons.items():
            if name == current_button_name:
                button.config(relief="sunken", bg="lightgreen")
            else:
                button.config(relief="raised", bg="lightblue")

    def update_button_states(self):
        self._update_button_states()
```

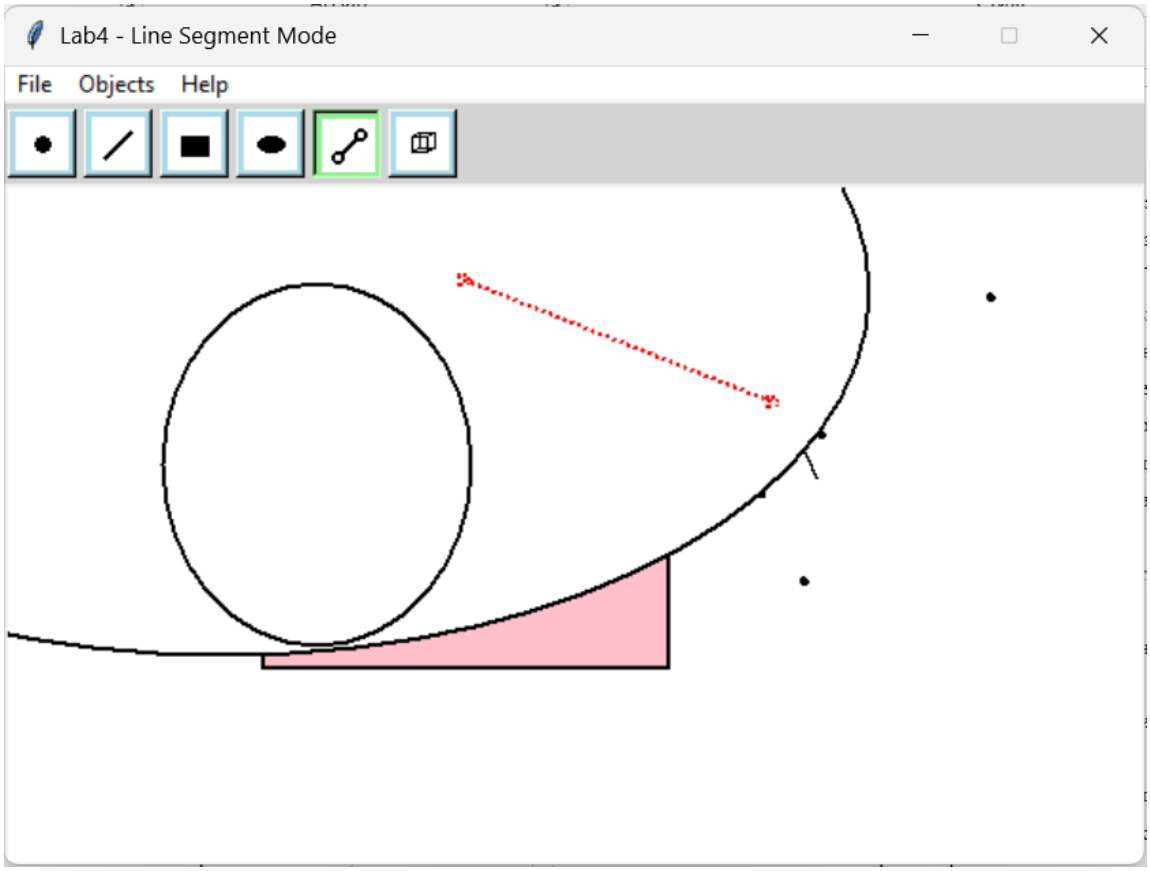
Ієрархія класів

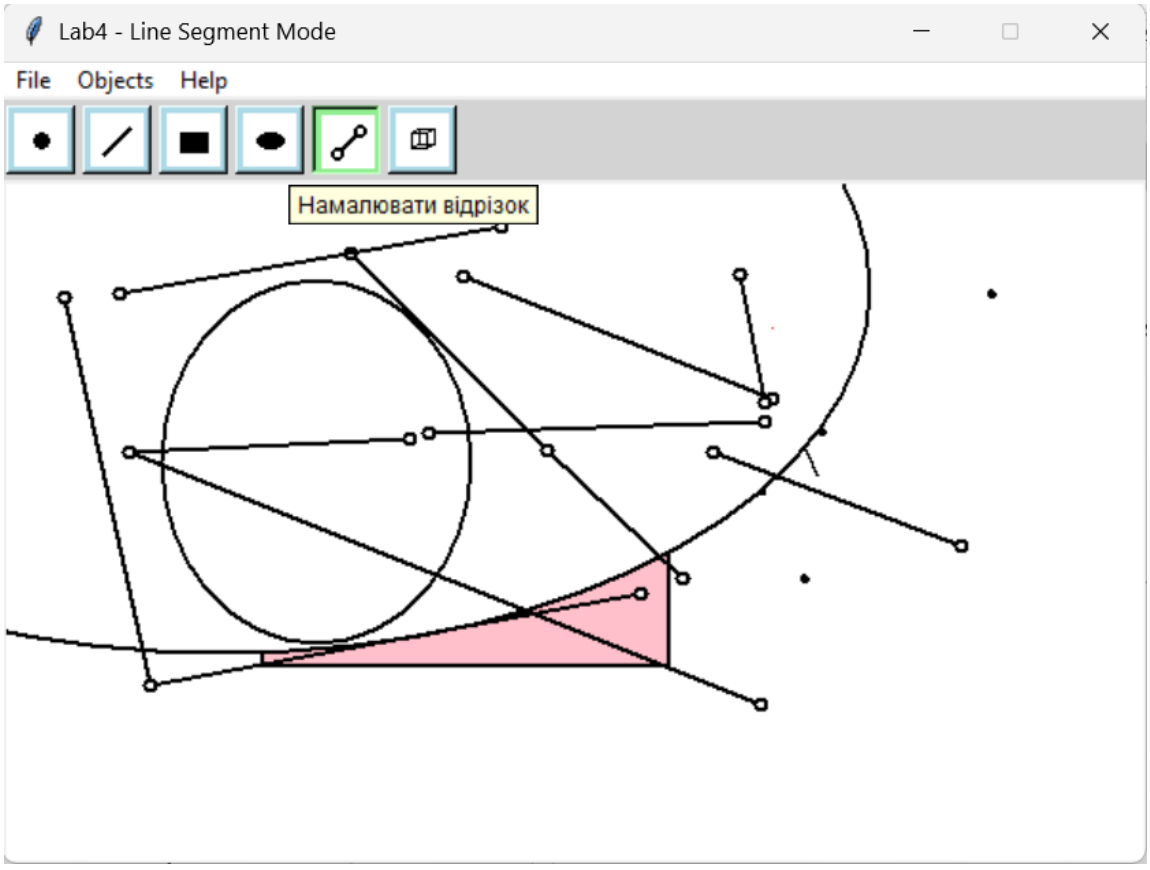


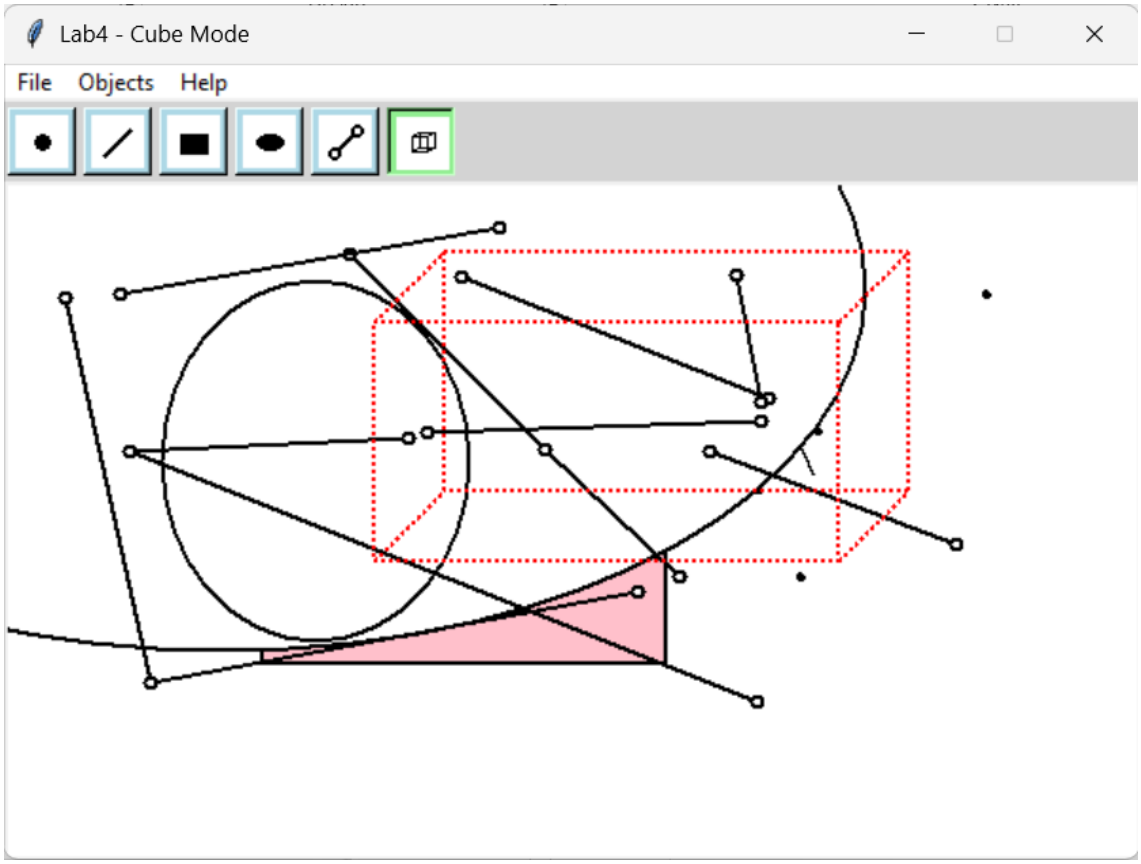
Результати тестування програми

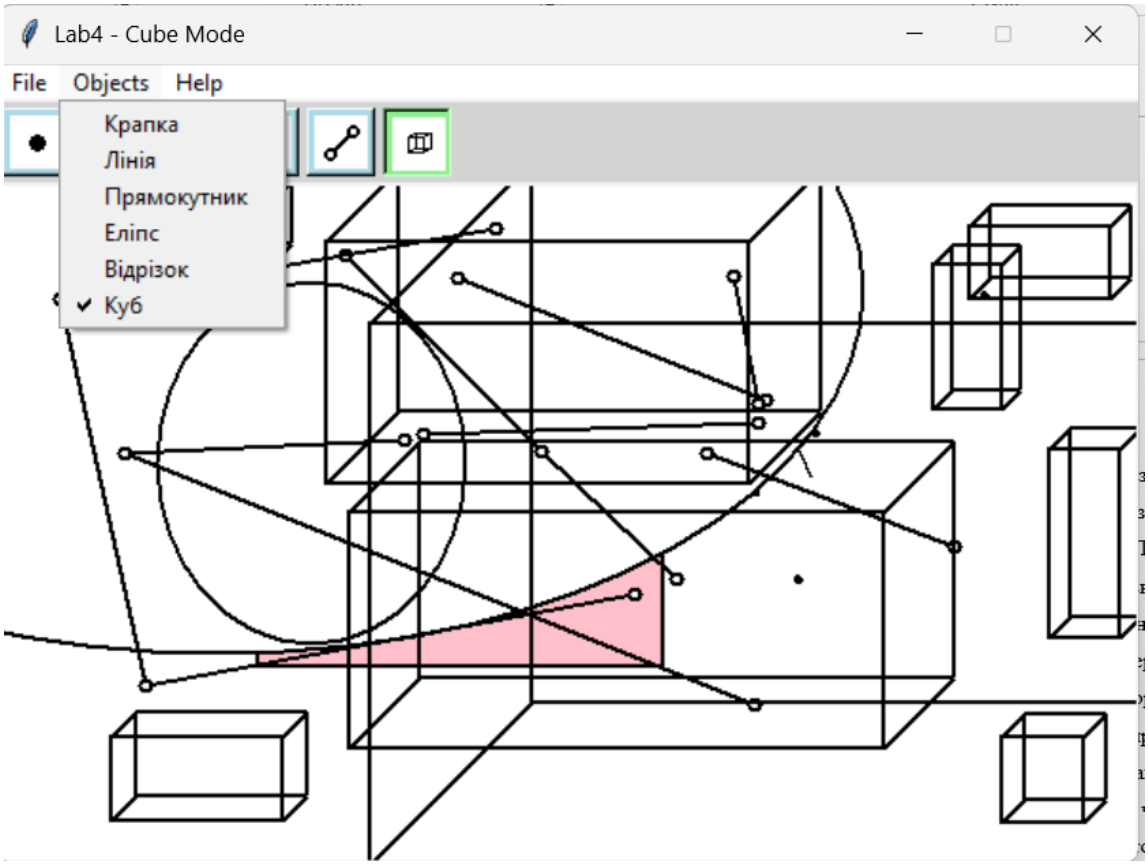





Lab4 - Line Segment Mode
File
Objects
Help


Lab4 - Line Segment Mode
File
Objects
Help
Намалювати відрізок


Lab4 - Cube Mode
File
Objects
Help


Lab4 - Cube Mode
File
Objects
Help
Крапка
Лінія
Прямокутник
Еліпс
Відрізок
Куб

Висновки

У ході виконання лабораторної роботи було реалізовано графічний редактор з панеллю меню, а саме кнопками File (кнопка Exit закриває програму), Objects (радіокнопки для вибору типу об'єкта: Крапка, Лінія, Прямокутник, Еліпс, Відрізок, Куб) та Help (кнопка About показує інформацію про роботу та авторські права).

Реалізовано інтерактивне полотно для малювання, на якому при виборі відповідного режиму через меню Objects або попередньо створений Toolbar, синхронізований з кнопкою меню Objects, можна створювати графічні примітиви: точку (при кліку мишею), лінію (перетягуванням миші), прямокутник (перетягуванням з виділенням області), еліпс (перетягуванням з виділенням області), відрізок з кружечками на кінцях (перетягуванням миші) та каркас куба (перетягуванням з виділенням області). Під час малювання реалізовано "гумовий слід" – червоний пунктирний контур, що відображає поточну позицію створюваного об'єкта.

Реалізовано підказки у вигляді тексту при наведенні курсором миші на одну з кнопок Toolbar. Для нових типів фігур (відрізок з кружечками та каркас куба) застосовано множинне наслідування класів, що дозволило поєднати функціональність базових класів ліній, прямокутників та еліпсів.

У результаті виконання роботи було набуто практичні навички роботи з мовою програмування Python, а саме бібліотекою Tkinter для створення графічного інтерфейсу, PIL для створення бітмапів, використаних у Toolbar, відпрацьовано принципи об'єктно-орієнтованого програмування на практиці через створення ієрархії класів для редакторів та графічних фігур, реалізовано принципи ін'єкції залежностей та Open/Closed Principle для забезпечення розширюваності системи, застосовано множинне наслідування для створення складних графічних об'єктів, реалізовано модульну архітектуру програми з чітким розподілом відповідальностей між компонентами та закріплено знання з проектування масштабованих та гнучких програмних рішень.