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 №5

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 непарним номером (1, 3, 5, . . .) програмують глобальний статичний об'єкт класу MyEditor у вигляді Singleton Меєрса.

Студенти з парним номером (2, 4, 6, . . .) програмують об’єкт класу MyEditor на основі класичної реалізації Singleton.

3. Усі кольори та стилі геометричних форм – як у попередньої лаб. роботі №4.

4. Запрограмувати вікно таблиці. Для його відкриття та закриття передбачити окремий пункт меню. Вікно таблиці повинно автоматично закриватися при виході з програми.

5. Вікно таблиці – немодальне вікно діалогу. Таблиця повинна бути запрограмована як клас у окремому модулі. Інтерфейс модуля у вигляді оголошення класу таблиці

6. Запрограмувати запис файлу множини об’єктів, що вводяться

7. Ієрархія класів та побудова модулів повинні бути зручними для можливостей додавання нових типів об'єктів без переписування коду вже існуючих модулів.

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

9. Бонуси-заохочення, які можуть суттєво підвищити оцінку лабораторної роботи. Оцінка підвищується за виконання кожного пункту, з наведених нижче:

1) Якщо у вікні таблиці буде передбачено, щоб користувач міг виділити курсором рядок таблиці і відповідний об’єкт буде якось виділятися на зображенні у головному вікні.
2) Якщо у вікні таблиці користувач може виділити курсором рядок таблиці і відповідний об’єкт буде вилучено з масиву об’єктів.
3) Якщо програма не тільки записує у файл опис множини об’єктів, а ще й здатна завантажити такий файл і відобразити відповідні об’єкти у головному вікні та вікні таблиці

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 my_table import MyTable
from table_view import TableView
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._current_mode = None
        self._menu_bar = None
        self._toolbar = None
        self._editor_manager = MyEditor.get_instance(canvas=None)
        MyTable.get_instance()

    def run(self):
        self._create_window()
        self._create_canvas()

        MyEditor.get_instance()._canvas = self._canvas

        editor = MyEditor.get_instance()

        def table_reload_callback():
            table_instance = TableView.get_instance(self._main_window)
            if table_instance:
                table_instance.reload_data()
                table_instance._toplevel.update_idletasks()
                table_instance._toplevel.lift()

        editor.register_gui_callback(table_reload_callback)

        MyTable.get_instance().register_listener(editor.handle_table_event)

        self._create_menu()
        self._create_toolbar()
        self._bind_events()
```

```
        self._set_default_mode()
        self._main_window.mainloop()

        TableView.destroy_window()

    def _create_window(self):
        self._main_window = tk.Tk()
        self._main_window.geometry("800x600+100+100")
        self._main_window.title("Lab5")
        self._main_window.bind('<Destroy>', self._on_exit)

    def _create_canvas(self):
        self._canvas = tk.Canvas(self._main_window, bg="white")
        self._canvas.pack(fill="both", expand=True)

    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_view_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="Save to File", command=self._save_data)
        file_menu.add_command(label="Load from File", command=self._load_data)
        file_menu.add_separator()
        file_menu.add_command(label="Exit", command=self._main_window.quit)
        self._menu_bar.add_cascade(label="File", menu=file_menu)

    def _create_view_menu(self):
        view_menu = tk.Menu(self._menu_bar, tearoff=0)
        view_menu.add_command(label="Show/Hide Table",
command=self._toggle_table_view)
        self._menu_bar.add_cascade(label="View", menu=view_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"Lab5 - {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"Lab5 - {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"Lab5 - {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"Lab5 - {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"Lab5 - {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"Lab5 - {self._current_mode.get()} Mode")
        self._toolbar.update_button_states()

    def _show_lab_information(self):
        messagebox.showinfo("Info", "Lab 5 is working fine\n(c) Copyright 2025")

    def _save_data(self):
        path = "lab5/Lab5_objects.txt"
        filename = path.split('/')[-1]
        try:
            self._editor_manager.save_to_file(path)
            messagebox.showinfo("Save to File", f"Object data successfully saved to
{filename}")
        except Exception as e:
            messagebox.showerror("Error", f"Failed to save data: {e}")

    def _load_data(self):
        path = "lab5/Lab5_objects.txt"
        filename = path.split('/')[-1]
        success = self._editor_manager.load_from_file(path)
        if success:
            messagebox.showinfo("Upload", f"Object are successfully loaded from
{filename}!")
        else:
             messagebox.showerror("Error", f"File {filename} is not found or it has
incorrect form.")


    def _toggle_table_view(self):
        if TableView._window_instance is None:
            TableView.get_instance(self._main_window)
        else:
            TableView.destroy_window()

    def _on_exit(self, event=None):
        TableView.destroy_window()

    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

    @abstractmethod
    def get_bounding_box(self):
        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))

    def get_bounding_box(self):
        radius = 5
        return (self.x1 - radius,
```

```
                self.y1 - radius,
                self.x1 + radius,
                self.y1 + radius)

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)

    def get_bounding_box(self):
        min_x = min(self.x1, self.x2)
        min_y = min(self.y1, self.y2)
        max_x = max(self.x1, self.x2)
        max_y = max(self.y1, self.y2)
        return min_x, min_y, max_x, max_y

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=2, dash=4)

    def get_bounding_box(self):
        min_x = min(self.x1, self.x2)
        min_y = min(self.y1, self.y2)
        max_x = max(self.x1, self.x2)
        max_y = max(self.y1, self.y2)
        return min_x, min_y, max_x, max_y

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=2, dash=(4, 2))

    def get_bounding_box(self):
        center_x = self.x1
        center_y = self.y1
        current_x = self.x2
        current_y = self.y2

        x1_bound = 2 * center_x - current_x
        y1_bound = 2 * center_y - current_y
        x2_bound = current_x
        y2_bound = current_y

        min_x = min(x1_bound, x2_bound)
        min_y = min(y1_bound, y2_bound)
        max_x = max(x1_bound, x2_bound)
        max_y = max(y1_bound, y2_bound)

        return min_x, min_y, max_x, max_y


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

    def get_bounding_box(self):

        min_x = min(self.x1, self.x2)
        min_y = min(self.y1, self.y2)
        max_x = max(self.x1, self.x2)
        max_y = max(self.y1, self.y2)

        padding = self._radius

        return (min_x - padding,
                min_y - padding,
                max_x + padding,
                max_y + padding)

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

    def get_bounding_box(self):
        front_face_coords, edges = self._calculate_cube_points()


        all_coords = [self.x1, self.y1, self.x2, self.y2]

        for edge in edges:
```

```
            all_coords.extend(edge)

        all_x = all_coords[::2]
        all_y = all_coords[1::2]

        min_x = min(all_x)
        min_y = min(all_y)
        max_x = max(all_x)
        max_y = max(all_y)

        return min_x, min_y, max_x, max_y

CLASS_MAP = {
    'PointShape': PointShape,
    'LineShape': LineShape,
    'RectShape': RectShape,
    'EllipseShape': EllipseShape,
    'LineSegmentShape': LineSegmentShape,
    'CubeShape': CubeShape,
}
```

3) my_table.py

```
class MyTable:
    _instance = None

    def __init__(self):
        if MyTable._instance is not None:
            raise Exception("This is Singleton! Use MyEditor.get_instance()")

        self._data = []
        self._view = None
        self._listeners = []
        MyTable._instance = self

    @staticmethod
    def get_instance():
        if MyTable._instance is None:
            MyTable()
        return MyTable._instance

    def set_view(self, view):
        self._view = view

    def get_data(self):
        return self._data

    def add_shape(self, shape):
        self._data.append(shape)
        new_index = len(self._data) - 1

        if self._view:
```

```
            self._view.insert_row(shape, new_index)

        return new_index

    def register_listener(self, listener_callback):
        self._listeners.append(listener_callback)

    def notify_listeners(self, event_type, index):
        for callback in self._listeners:
            callback(event_type, index)

EVENT_SELECT = "select"
EVENT_DELETE = "delete"
```

4) table_view.py

```
import tkinter as tk
from tkinter import ttk
from my_table import MyTable, EVENT_SELECT, EVENT_DELETE

class TableView:
    _window_instance = None

    @staticmethod
    def get_instance(root=None):
        if TableView._window_instance is None:
            if root is None:
                pass
            TableView._window_instance = TableView(root)
            MyTable.get_instance().set_view(TableView._window_instance)
        return TableView._window_instance

    @staticmethod
    def destroy_window():
        if TableView._window_instance:
            MyTable.get_instance().set_view(None)
            TableView._window_instance._toplevel.destroy()
            TableView._window_instance = None

    def __init__(self, root):
        self._toplevel = tk.Toplevel(root)
        self._toplevel.title("Objects table")
        self._toplevel.geometry("500x350")
        self._toplevel.protocol("WM_DELETE_WINDOW", self.destroy_window)

        self._tree = self._create_table_widget()
        self._create_controls()
        self._load_initial_data()

    def _create_table_widget(self):
        columns = ("Назва", "X1", "Y1", "X2", "Y2")
        tree = ttk.Treeview(self._toplevel, columns=columns, show="headings")
```

```
        for col in columns:
            tree.heading(col, text=col)
            width = 75 if col != "Name" else 120
            tree.column(col, anchor="center", width=width)

        vsb = ttk.Scrollbar(self._toplevel, orient="vertical", command=tree.yview)
        tree.configure(yscrollcommand=vsb.set)

        vsb.pack(side="right", fill="y")
        tree.pack(fill="both", expand=True)

        tree.bind("<<TreeviewSelect>>", self._on_select_item)

        return tree

    def _create_controls(self):
        frame = tk.Frame(self._toplevel)
        frame.pack(fill="x", pady=5, padx=5)

        delete_btn = tk.Button(frame, text="Delete object",
command=self._on_delete_item)
        delete_btn.pack(side="bottom", padx=5)

    def _on_select_item(self, event):
        selected_items = self._tree.selection()
        if selected_items:
            item_id = selected_items[0]
            index = int(self._tree.item(item_id, "text"))

            MyTable.get_instance().notify_listeners(EVENT_SELECT, index)
        else:
             MyTable.get_instance().notify_listeners(EVENT_SELECT, -1)

    def _on_delete_item(self):
        selected_items = self._tree.selection()
        if selected_items:
            item_id = selected_items[0]
            index = int(self._tree.item(item_id, "text"))

            MyTable.get_instance().notify_listeners(EVENT_DELETE, index)

    def _load_initial_data(self):
        self.reload_data()

    def reload_data(self):
        try:
            self._tree.selection_remove(self._tree.selection())
        except:
             pass
```

```
        self.clear_table()

        data = MyTable.get_instance().get_data()
        for index, shape in enumerate(data):
            self.insert_row(shape, index)

    def insert_row(self, shape, index):
        self._tree.insert("", "end", text=str(index),
                          values=(shape.__class__.__name__,
                                  shape.x1, shape.y1,
                                  shape.x2, shape.y2))

    def clear_table(self):
        for item in self._tree.get_children():
            self._tree.delete(item)
```

5) my_editor.py

```
from my_table import MyTable, EVENT_SELECT, EVENT_DELETE
from shape import CLASS_MAP

class MyEditor:
    _instance = None

    def __init__(self, canvas=None):
        if MyEditor._instance is not None:
            raise Exception("This is Singleton! Use MyEditor.get_instance()")

        self._canvas = canvas
        self._shapes = []
        self._current_shape = None
        self._preview_id = None
        self._start_x = 0
        self._start_y = 0
        self._highlight_id = None
        self._gui_reload_callback = None
        MyEditor._instance = self

    @staticmethod
    def get_instance(canvas=None):
        if MyEditor._instance is None:
            MyEditor(canvas)
        return MyEditor._instance

    def register_gui_callback(self, callback):
        self._gui_reload_callback = callback

    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:
            new_shape = type(self._current_shape)()
            new_shape.set(self._start_x, self._start_y, x, y)

            self._shapes.append(new_shape)

            MyTable.get_instance().add_shape(new_shape)

            self._clear_preview()
            self._redraw()
            self._current_shape = type(self._current_shape)()

    def _redraw(self):
        self._canvas.delete("all")
        self._highlight_id = None
        for shape in self._shapes:
            shape.show(self._canvas)

    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 _highlight_shape(self, index):
```

```
        if self._highlight_id:
            self._canvas.delete(self._highlight_id)
            self._highlight_id = None

        if 0 <= index < len(self._shapes):
            shape_to_highlight = self._shapes[index]

            min_x, min_y, max_x, max_y = shape_to_highlight.get_bounding_box()

            padding = 5
            self._highlight_id = self._canvas.create_rectangle(
                min_x - padding, min_y - padding,
                max_x + padding, max_y + padding,
                outline="red", width=2, dash=(4, 4)
            )

    def _delete_shape(self, index):
        if 0 <= index < len(self._shapes):

            self._shapes.pop(index)
            try:
                 MyTable.get_instance().get_data().pop(index)
            except IndexError:
                 pass

            self._redraw()

            if self._gui_reload_callback:
                self._gui_reload_callback()

            self._highlight_shape(-1)

    def handle_table_event(self, event_type, index):
        if event_type == EVENT_SELECT:
            self._highlight_shape(index)
        elif event_type == EVENT_DELETE:
            self._delete_shape(index)

    def save_to_file(self, path="lab5/Lab5_objects.txt"):
        with open(path, 'w') as f:
            for shape in self._shapes:
                class_name = type(shape).__name__
                line =
f"{class_name}\t{shape.x1}\t{shape.y1}\t{shape.x2}\t{shape.y2}\n"
                f.write(line)
        return True

    def load_from_file(self, path="lab5/Lab5_objects.txt"):
        self._shapes.clear()
        MyTable.get_instance().get_data().clear()
```

```
        try:
            with open(path, 'r') as f:
                for line in f:
                    line = line.strip()
                    if not line:
                        continue

                    parts = line.split('\t')
                    if len(parts) != 5:
                        continue

                    class_name = parts[0]
                    coords = [int(p) for p in parts[1:]]

                    if class_name in CLASS_MAP:
                        ShapeClass = CLASS_MAP[class_name]
                        new_shape = ShapeClass()
                        new_shape.set(*coords)

                        self._shapes.append(new_shape)
                        MyTable.get_instance().add_shape(new_shape)

            self._redraw()
            if self._gui_reload_callback:
                self._gui_reload_callback()

            return True

        except FileNotFoundError:
            return False
        except Exception:
            return False
```

6) 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["Елипс"] = 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
```

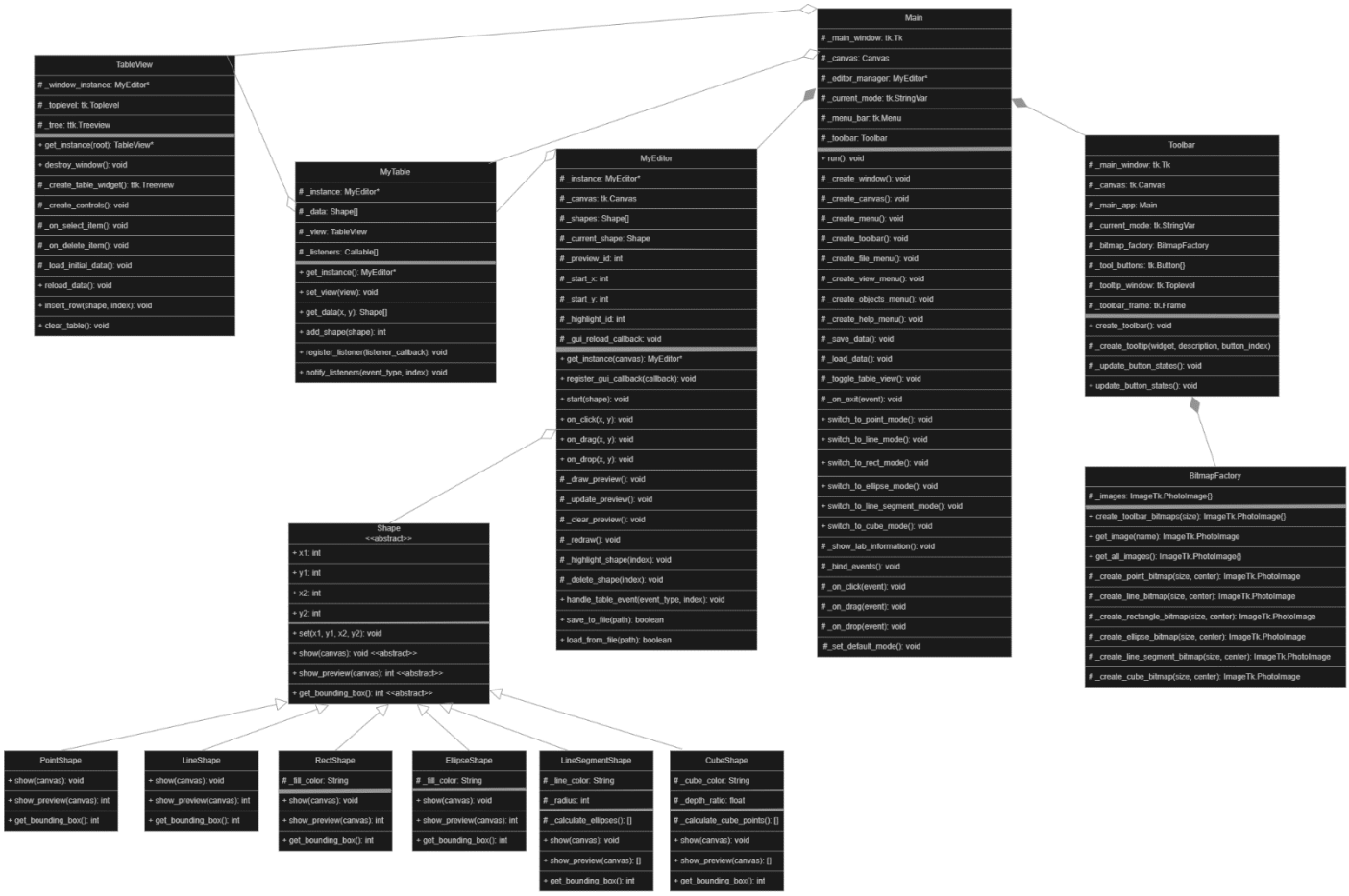
5) toolbar.py

```
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"),
```

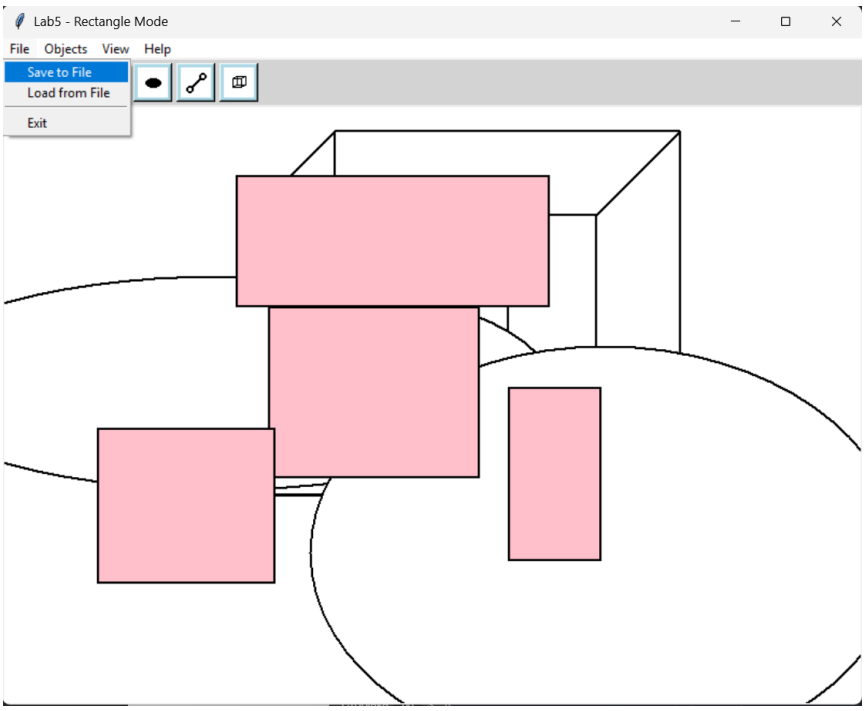
```
        ("Куб", self._main_app.switch_to_cube_mode, "Намалювати куб")
    ]

    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)
```

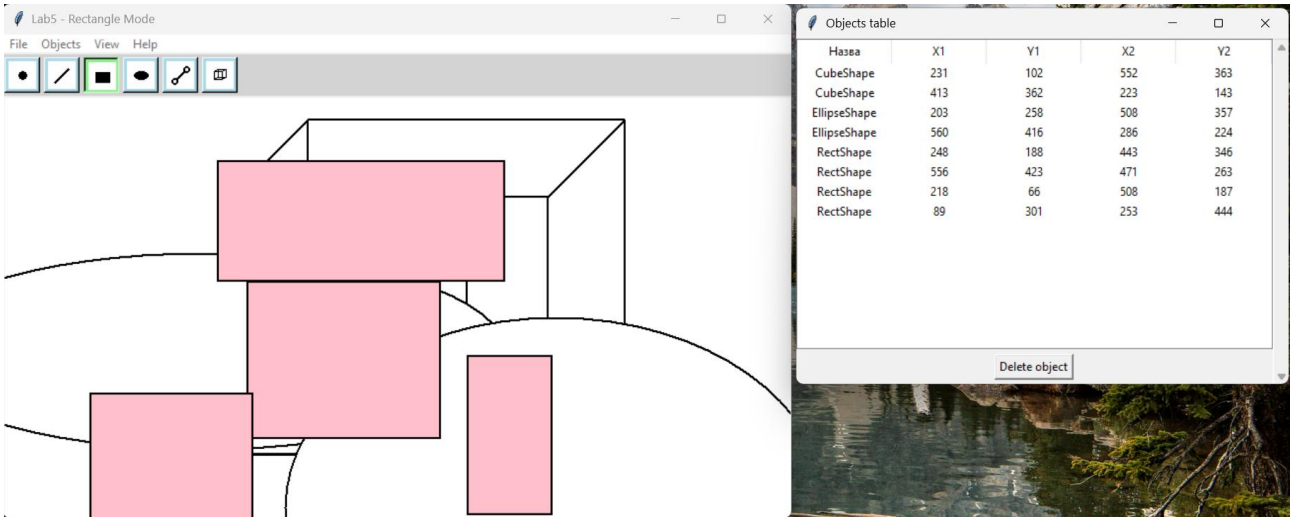
```
        widget.bind("<Leave>", hide_tooltip)
        widget.bind("<ButtonPress>", hide_tooltip)

    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()
```

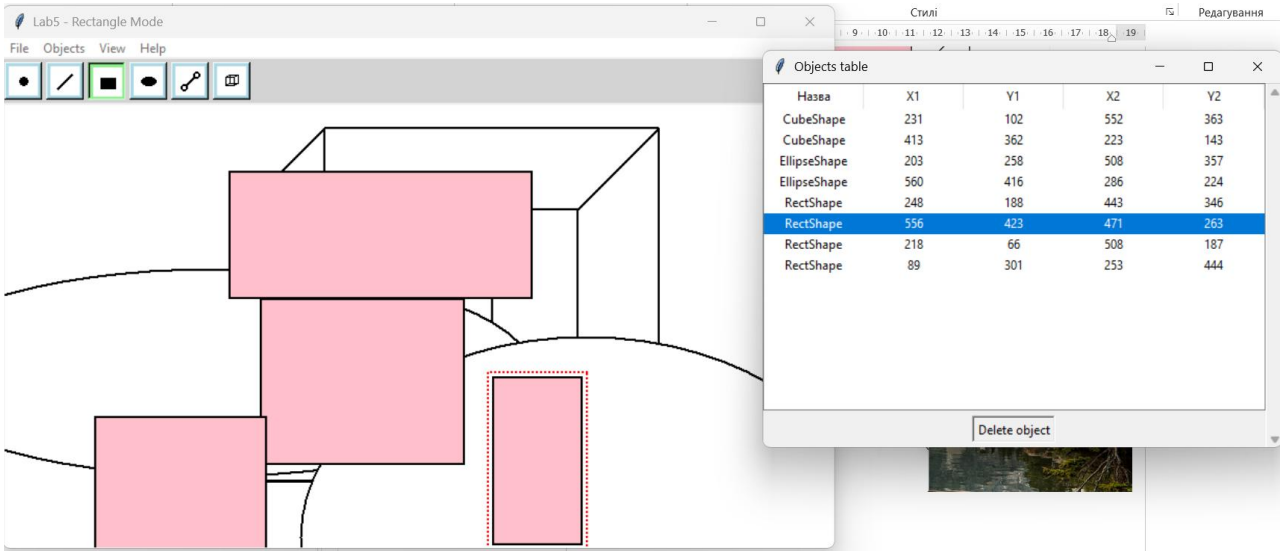
Ієрархія класів

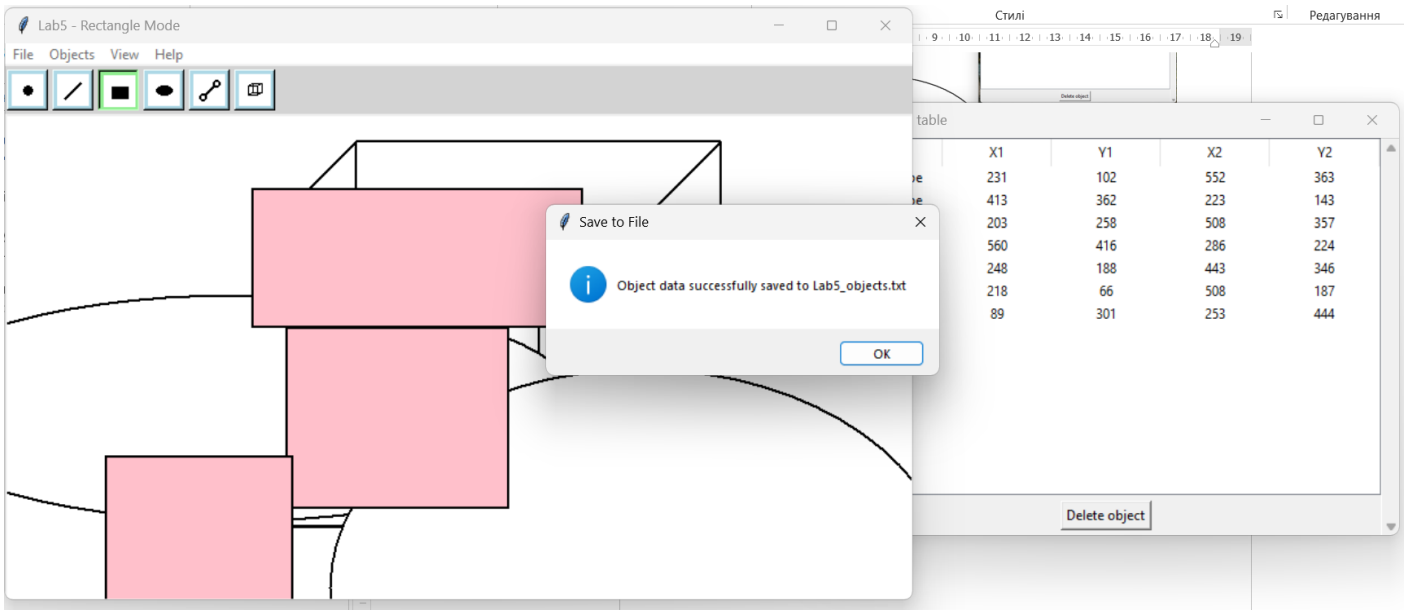


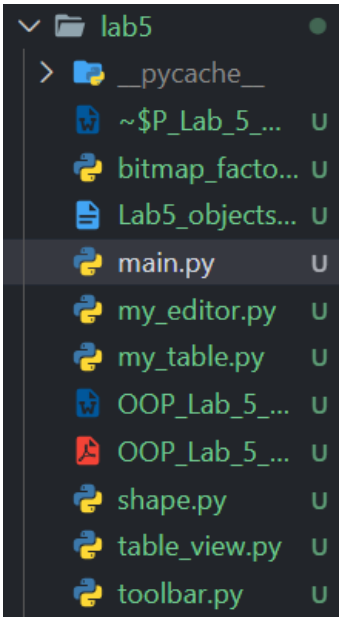
Результати тестування програми

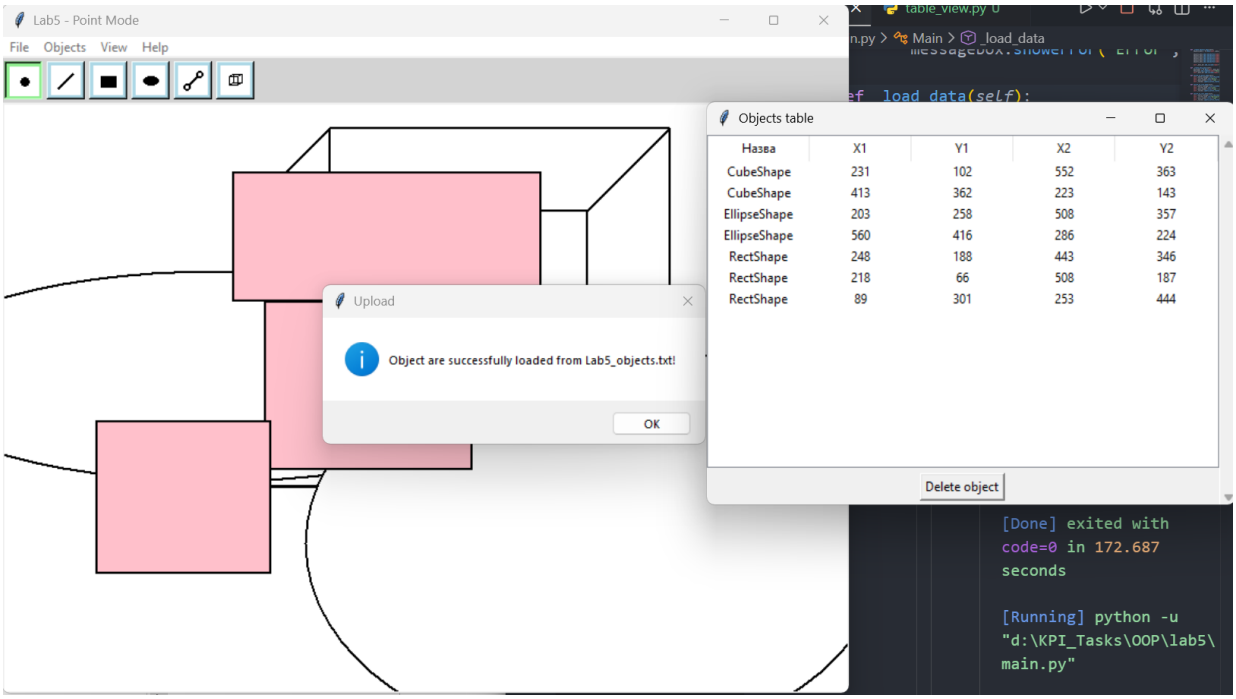







Lab5 - Rectangle Mode
File Objects View Help
Save to File
Object data successfully saved to Lab5_objects.txt
OK
table
X1 Y1 X2 Y2
231 102 552 363
413 362 223 143
203 258 508 357
560 416 286 224
248 188 443 346
218 66 508 187
89 301 253 444
Delete object


lab5
__pycache__
~$P_Lab_5_...
bitmap_facto...
Lab5_objects...
main.py
my_editor.py
my_table.py
OOP_Lab_5_...
OOP_Lab_5_...
shape.py
table_view.py
toolbar.py


Lab5 - Point Mode
File Objects View Help
Upload
Object are successfully loaded from Lab5_objects.txt!
OK
Objects table
Назва X1 Y1 X2 Y2
CubeShape 231 102 552 363
CubeShape 413 362 223 143
EllipseShape 203 258 508 357
EllipseShape 560 416 286 224
RectShape 248 188 443 346
RectShape 218 66 508 187
RectShape 89 301 253 444
Delete object
[Done] exited with code=0 in 172.687 seconds
[Running] python -u "d:\KPI_Tasks\OOP\lab5\main.py"

Висновки

У ході виконання лабораторної роботи було реалізовано графічний редактор із панеллю меню та інтерактивним полотном. Структура меню включала кнопки File (функції Save для збереження полотна, Load для відображення завантаженого полотна та Exit для закриття програми), Objects (радіокнопки для вибору типу об'єкта: Крапка, Лінія, Прямокутник, Еліпс, Відрізок, Куб) та Help (кнопка About показує інформацію про роботу).

Було 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), а також складні фігури: відрізок з кружечками на кінцях (перетягуванням миші) та каркас куба (перетягуванням з виділенням області). Під час малювання було реалізовано ефект "гумового сліду".

Для нових типів фігур (відрізок з кружечками та каркас куба) було застосовано множинне наслідування класів, що дозволило поєднати функціональність базових класів ліній, прямокутників та еліпсів. Головний об'єкт-редактор було реалізовано із застосуванням патерну Singleton.

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. Було відпрацьовано принципи об'єктно-орієнтованого програмування на практиці через створення ієрархії класів для редакторів та графічних фігур. Створено таблицю для відображення координат створених фігур, їхнього видалення та показу при завантаженні збереженого файлу. Реалізовано виділення та видалення фігур, користуючись таблицею.