**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 №3**

з дисципліни  
«Об’єктно-орієнтоване програмування»

Виконав: Перевірив:

студент групи IM-43 Порєв В. М.

Олексійчук Станіслав Юрійович

номер у списку групи: 20

Київ 2025

***Варіант завдання та основні вимоги***

Варіант 20+1(21):

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

2. Усі методи-обробники повідомлень, зокрема, і метод OnNotify, повинні бути функціями-членами деякого класу (класів).

3. Для вибору типу об’єкту в графічному редакторі Lab3 повинно бути вікно Toolbar з кнопками відповідно типам об'єктів. Кнопки дублюють підпункти меню "Об’єкти". Кнопки мають бути з підказками (tooltips). Меню "Об’єкти" повинно бути праворуч меню "Файл" та ліворуч меню "Довідка".

Підпункти меню "Об’єкти" містять назви геометричних форм українською мовою. Геометричні форми згідно варіанту завдання.

4. Для вибору варіанту використовується значення Ж = Жлаб2 + 1, де

Жлаб2 – номер студента в журналі, який використовувався для попередньої лаб. роботи No2.

5. Масив вказівників для динамічних об’єктів типу Shape

- динамічний масив Shape \*\*pcshape;

Динамічний масив обирають студенти, у яких варіант (Ж mod 3 = 0).

6. "Гумовий" слід при вводі об’єктів

- суцільна лінія червоного кольору для (Ж mod 4 = 1)

7. Чотири геометричні форми (крапка, лінія, прямокутник, еліпс) можуть мати наступні різновиди вводу та відображення.

7.1. Прямокутник

Увід прямокутника:

- від центру до одного з кутів для (Ж mod 2 = 1)

Відображення прямокутника:

- чорний контур з кольоровим заповненням для (Ж mod 5 = 1 або 2)

Кольори заповнення прямокутника:

- рожевий для (Ж mod 6 = 3)

7.2. Еліпс

Ввід еліпсу:

- по двом протилежним кутам охоплюючого прямокутника для варіантів (Ж mod 2 = 1)

Відображення еліпсу:

- чорний контур з білим заповненням для (Ж mod 5 = 1)

8. Позначка поточного типу об’єкту, що вводиться

- в заголовку вікна для (Ж mod 2 = 1)

***Текст програми***

Лабораторна робота виконана мовою програмування Python з використанням бібліотеки Tkinter. Це завдання складається з головного файлу та трьох модулів:  
1) main.py

import tkinter as tk

from tkinter import messagebox

from shape\_objects\_editor import ShapeObjectsEditor

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\_editor()

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 = ShapeObjectsEditor(self.\_canvas, self.\_main\_window)

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"

)

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\_point\_editor()

self.\_current\_mode.set("Point")

self.\_main\_window.title(f"Lab3 - {self.\_current\_mode.get()} Mode")

self.\_toolbar.update\_button\_states()

def switch\_to\_line\_mode(self):

self.\_editor\_manager.start\_line\_editor()

self.\_current\_mode.set("Line")

self.\_main\_window.title(f"Lab3 - {self.\_current\_mode.get()} Mode")

self.\_toolbar.update\_button\_states()

def switch\_to\_rect\_mode(self):

self.\_editor\_manager.start\_rect\_editor()

self.\_current\_mode.set("Rectangle")

self.\_main\_window.title(f"Lab3 - {self.\_current\_mode.get()} Mode")

self.\_toolbar.update\_button\_states()

def switch\_to\_ellipse\_mode(self):

self.\_editor\_manager.start\_ellipse\_editor()

self.\_current\_mode.set("Ellipse")

self.\_main\_window.title(f"Lab3 - {self.\_current\_mode.get()} Mode")

self.\_toolbar.update\_button\_states()

def \_show\_lab\_information(self):

messagebox.showinfo("Info", "Lab 3 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\_editor(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

class PointShape(Shape):

def show(self, canvas):

canvas.create\_oval(self.x1-1, self.y1-1,

self.x1+1, self.y1+1,

fill="black", outline="black")

class LineShape(Shape):

def show(self, canvas):

canvas.create\_line(self.x1, self.y1,

self.x2, self.y2,

fill="black", width=1)

class RectShape(Shape):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.\_fill\_color = "pink"

def show(self, canvas):

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

canvas.create\_rectangle(x1, y1, x2, y2,

outline="black", fill=self.\_fill\_color, width=1)

class EllipseShape(Shape):

def \_\_init\_\_(self):

super().\_\_init\_\_()

self.\_fill\_color = "white"

def show(self, canvas):

canvas.create\_oval(self.x1, self.y1,

self.x2, self.y2,

outline="black", fill=self.\_fill\_color, width=1)

3) editor.py

from abc import ABC, abstractmethod

from shape import PointShape, LineShape, RectShape, EllipseShape

class Editor(ABC):

@abstractmethod

def on\_click(self, x, y): pass

@abstractmethod

def on\_drag(self, x, y): pass

@abstractmethod

def on\_drop(self, x, y): pass

class ShapeEditor(Editor):

def \_\_init\_\_(self, canvas):

self.\_canvas = canvas

self.\_current\_shape = None

self.\_preview\_id = None

self.\_start\_x = 0

self.\_start\_y = 0

def on\_click(self, x, y):

self.\_start\_x = x

self.\_start\_y = y

def on\_drag(self, x, y):

if self.\_current\_shape and self.\_preview\_id:

self.\_canvas.delete(self.\_preview\_id)

self.\_current\_shape.set(self.\_start\_x, self.\_start\_y, x, y)

self.\_preview\_id = self.\_draw\_preview()

def on\_drop(self, x, y):

if self.\_current\_shape:

self.\_current\_shape.set(self.\_start\_x, self.\_start\_y, x, y)

if self.\_preview\_id:

self.\_canvas.delete(self.\_preview\_id)

shape = self.\_current\_shape

self.\_current\_shape = None

self.\_preview\_id = None

return shape

return None

@abstractmethod

def \_draw\_preview(self):

pass

class PointEditor(ShapeEditor):

def on\_click(self, x, y):

super().on\_click(x, y)

self.\_current\_shape = PointShape()

self.\_current\_shape.set(x, y, x, y)

if self.\_preview\_id:

self.\_canvas.delete(self.\_preview\_id)

self.\_preview\_id = self.\_draw\_preview()

def \_draw\_preview(self):

return self.\_canvas.create\_oval(

self.\_start\_x-2, self.\_start\_y-2,

self.\_start\_x+2, self.\_start\_y+2,

outline="red", width=1

)

class LineEditor(ShapeEditor):

def on\_click(self, x, y):

super().on\_click(x, y)

self.\_current\_shape = LineShape()

self.\_current\_shape.set(x, y, x, y)

if self.\_preview\_id:

self.\_canvas.delete(self.\_preview\_id)

self.\_preview\_id = self.\_draw\_preview()

def \_draw\_preview(self):

current\_shape = self.\_current\_shape

return self.\_canvas.create\_line(

self.\_start\_x, self.\_start\_y,

current\_shape.x2, current\_shape.y2,

fill="red", width=1

)

class RectEditor(ShapeEditor):

def on\_click(self, x, y):

super().on\_click(x, y)

self.\_current\_shape = RectShape()

self.\_current\_shape.set(x, y, x, y)

if self.\_preview\_id:

self.\_canvas.delete(self.\_preview\_id)

self.\_preview\_id = self.\_draw\_preview()

def \_draw\_preview(self):

center\_x = self.\_start\_x

center\_y = self.\_start\_y

current\_shape = self.\_current\_shape

current\_x = current\_shape.x2

current\_y = current\_shape.y2

x1 = 2 \* center\_x - current\_x

y1 = 2 \* center\_y - current\_y

x2 = current\_x

y2 = current\_y

return self.\_canvas.create\_rectangle(

x1, y1, x2, y2,

outline="red", width=1

)

class EllipseEditor(ShapeEditor):

def on\_click(self, x, y):

super().on\_click(x, y)

self.\_current\_shape = EllipseShape()

self.\_current\_shape.set(x, y, x, y)

if self.\_preview\_id:

self.\_canvas.delete(self.\_preview\_id)

self.\_preview\_id = self.\_draw\_preview()

def \_draw\_preview(self):

current\_shape = self.\_current\_shape

return self.\_canvas.create\_oval(

self.\_start\_x, self.\_start\_y,

current\_shape.x2, current\_shape.y2,

outline="red", width=1

)

4) shape\_objects\_editor.py

from editor import PointEditor, LineEditor, RectEditor, EllipseEditor

class ShapeObjectsEditor:

def \_\_init\_\_(self, canvas, main\_window=None):

self.\_canvas = canvas

self.\_main = main\_window

self.\_shapes = []

self.\_current\_editor = None

def start\_point\_editor(self):

self.\_current\_editor = PointEditor(self.\_canvas)

def start\_line\_editor(self):

self.\_current\_editor = LineEditor(self.\_canvas)

def start\_rect\_editor(self):

self.\_current\_editor = RectEditor(self.\_canvas)

def start\_ellipse\_editor(self):

self.\_current\_editor = EllipseEditor(self.\_canvas)

def on\_click(self, x, y):

if self.\_current\_editor:

self.\_current\_editor.on\_click(x, y)

def on\_drag(self, x, y):

if self.\_current\_editor:

self.\_current\_editor.on\_drag(x, y)

def on\_drop(self, x, y):

if self.\_current\_editor:

shape = self.\_current\_editor.on\_drop(x, y)

if shape:

self.\_shapes.append(shape)

self.\_redraw()

return None

def \_redraw(self):

self.\_canvas.delete("all")

for shape in self.\_shapes:

if shape:

shape.show(self.\_canvas)

5) 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)

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 get\_image(self, name):

return self.\_images.get(name)

def get\_all\_images(self):

return self.\_images

4) 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")

]

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"

}

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")

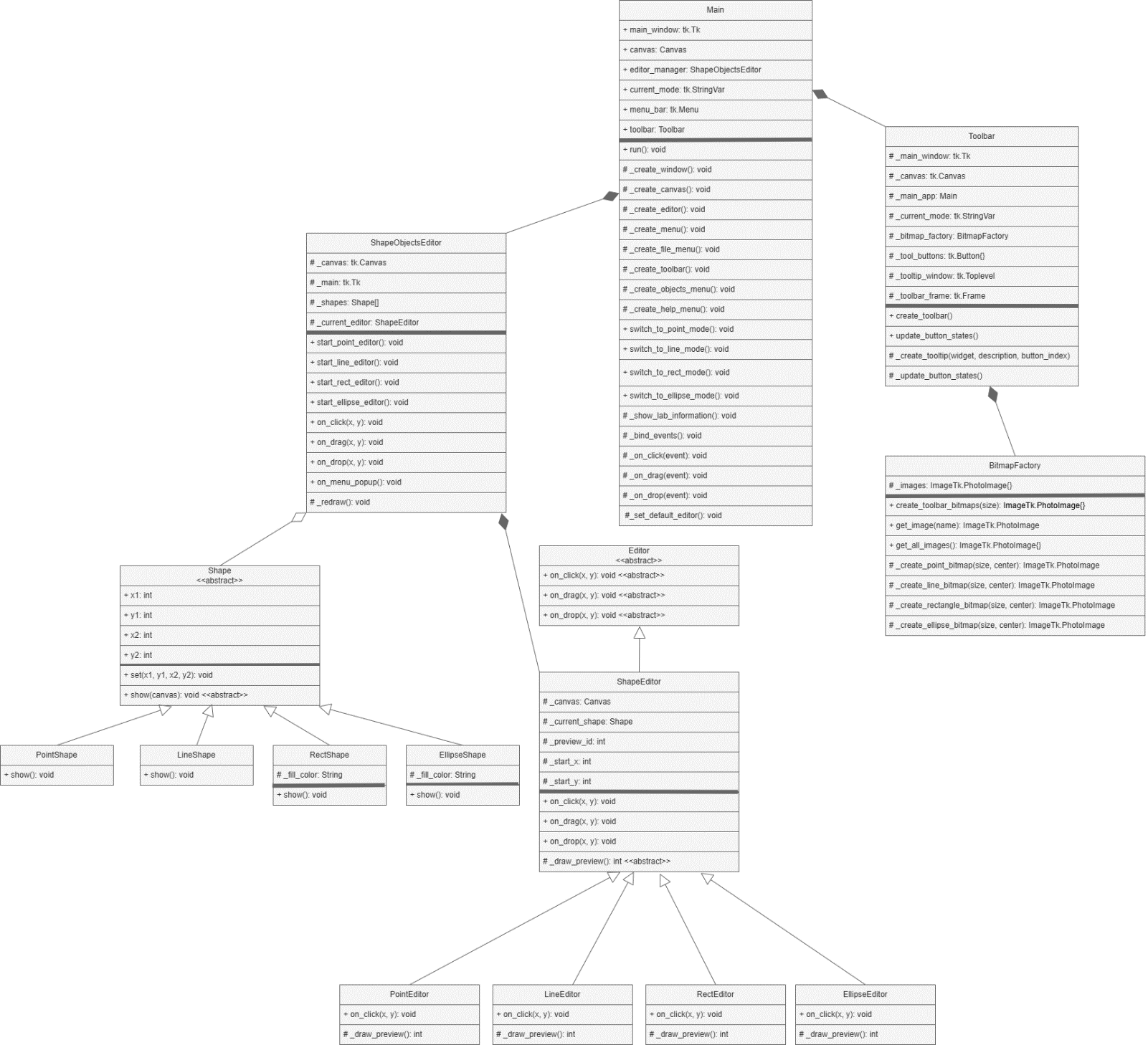
else:

button.config(relief="raised", bg="lightblue")

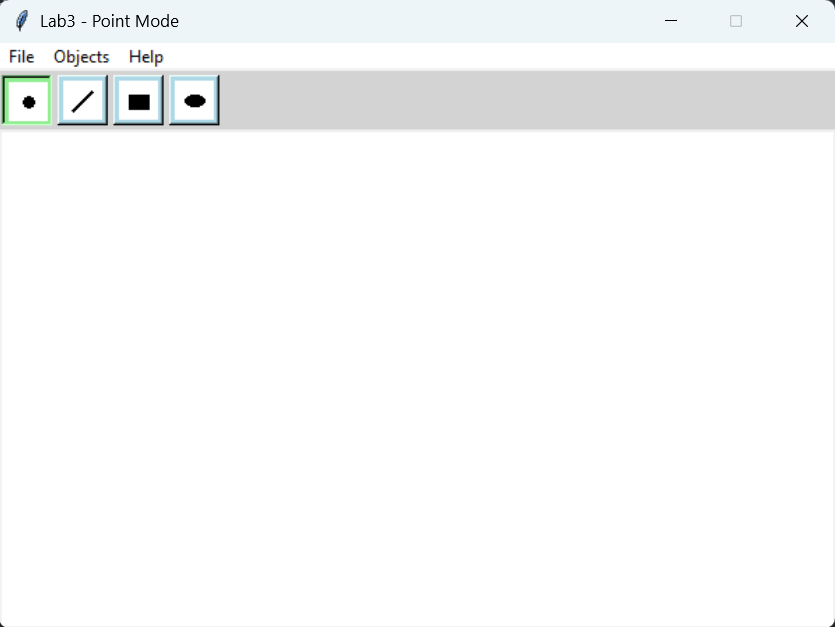
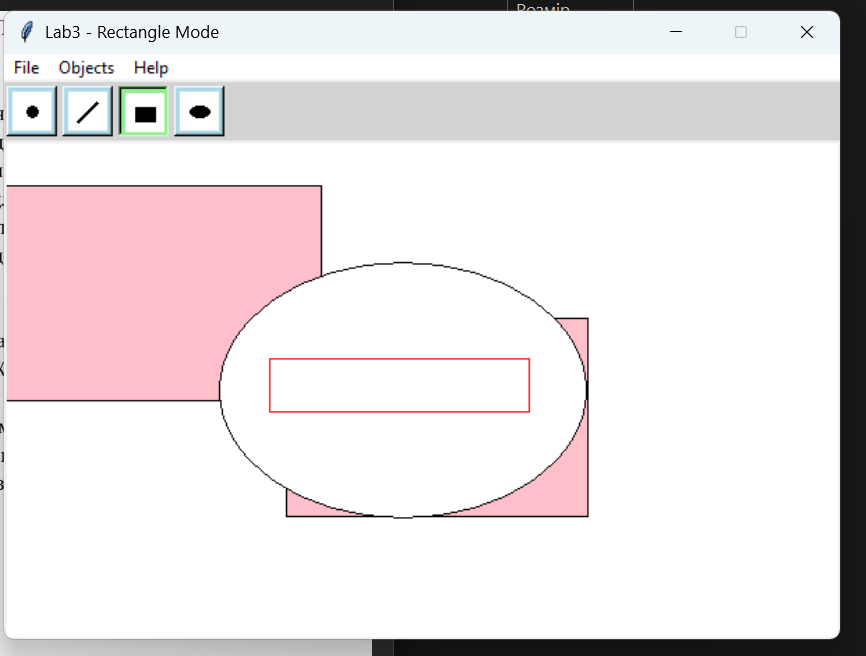
def update\_button\_states(self):

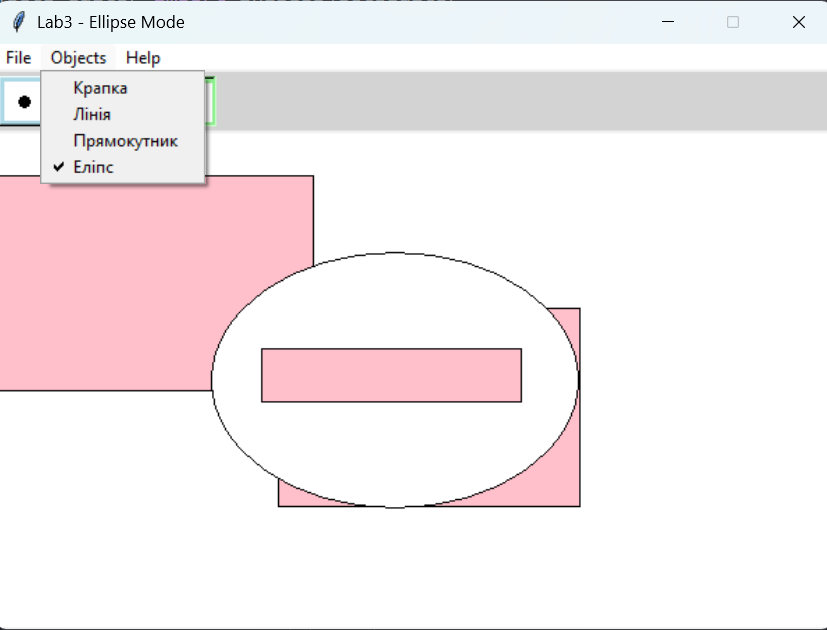
self.\_update\_button\_states()

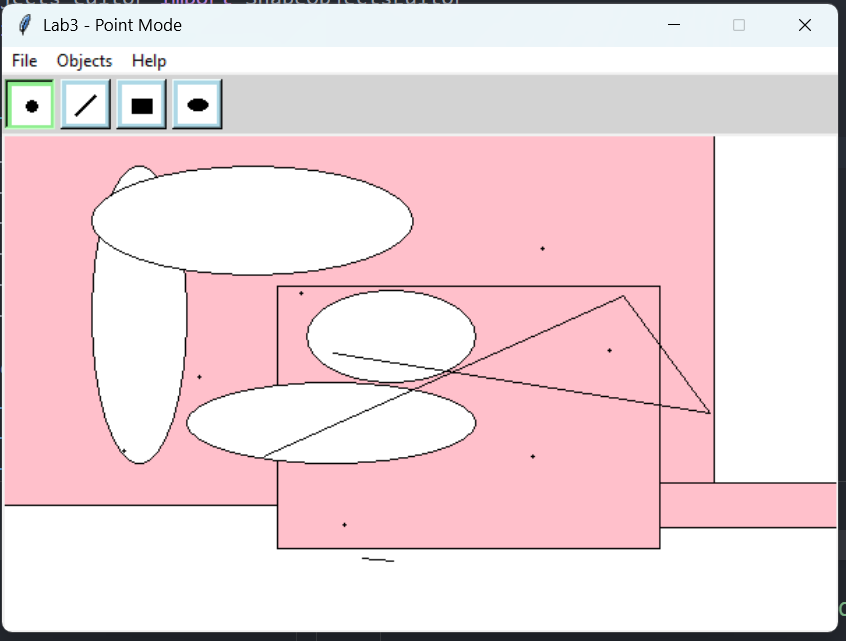
***Ієрархія класів***

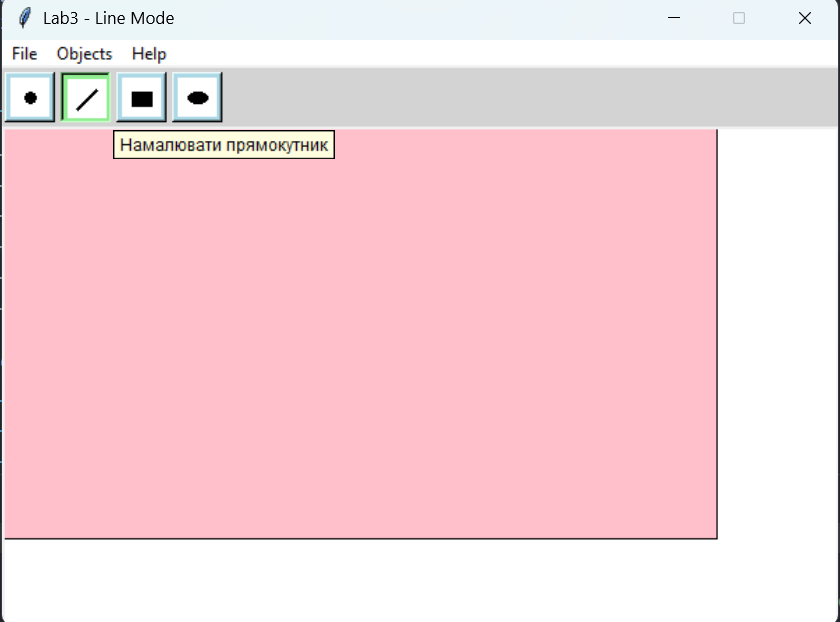
**

***Результати тестування програми***

\







***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) 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 від центру з виділенням області) та еліпс (перетягування від протилежних кутів описаного прямокутника з виділенням області). Під час малювання реалізовано "гумовий слід" – червоний контур, що відображає поточну позицію створюваного об'єкта.

Реалізовано підказки у вигляді тексту при наведенні курсором миші на одну з кнопок.

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 модульну архітектуру програми з чітким розподілом відповідальностей між компонентами та закріплено знання з проектування масштабованих та гнучких програмних рішень.