**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»

Виконала:

студентка групи ІМ-43
Хубеджева Єлизавета Павлівна
номер у списку групи: 28

Перевірив:

Порєв В. М.

Київ 2025

**Завдання**

Так, як $Ж = Ж_{лаб} + 1 = 28 + 1 = 29$, то:

29 mod 3 = 2 - **статичний масив**

**N** = 129 - кількість елементів масиву

**Гумовий слід:**

29 mod 4 = 1 - суцільна лінія червоного кольору

**Прямокутник:**

29 mod 2 = 1 - від центру до одного з кутів

29 mod 5 = 4 - чорний контур прямокутника без заповнення

**Еліпс:**

29 mod 2 = 1 - по двом протилежним кутам охоплюючого прямокутника

29 mod 5 = 4 - чорний контур з кольоровим заповненням

29 mod 6 = 5 - помаранчевий колір заповнення

**Позначка поточного типу об’єкту, що вводиться:**

29 mod 2 = 1 - в заголовку вікна

## 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 editors import PointEditor, LineEditor, RectEditor, EllipseEditor
from toolbar import Toolbar

class DrawingApp:

    MAX_SHAPES = 129

    def __init__(self, root):
        self.root = root
        self.base_title = "Лабораторна робота 3"
        self.root.title(self.base_title)
        self.root.geometry("950x600")

        self.shapes = [None] * self.MAX_SHAPES
        self.shape_count = 0

        self.current_editor = None
        self.current_tool = None

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

        self.editors = {
            'point': PointEditor(self.canvas),
            'line': LineEditor(self.canvas),
            'rect': RectEditor(self.canvas),
            'ellipse': EllipseEditor(self.canvas)
        }

        self.select_tool('point')

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

```
        self.canvas.bind("<ButtonRelease-1>", self.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"
        }

        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):
        self.status_bar = tk.Label(
            self.root,
            text=f"Об'єктів: {self.shape_count}/{self.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.current_editor = self.editors[tool]
        self.toolbar.set_active_tool(tool)
        self.update_status()

    def on_mouse_down(self, event):
        if self.current_editor and self.shape_count < self.MAX_SHAPES:
            self.current_editor.on_mouse_down(event)

    def on_mouse_move(self, event):
        if self.current_editor:
            self.current_editor.on_mouse_move(event)

    def on_mouse_up(self, event):
        if self.current_editor and self.shape_count < self.MAX_SHAPES:
            shape = self.current_editor.on_mouse_up(event)
            if shape:
                self.add_shape(shape)

    def add_shape(self, shape):
        if self.shape_count < self.MAX_SHAPES:
            self.shapes[self.shape_count] = shape
            self.shape_count += 1
            shape.show(self.canvas)
            self.update_status()
```

```
    def clear_canvas(self):
        if self.shape_count > 0:
            result = messagebox.askyesno(
                "Підтвердження",
                "Ви впевнені, що хочете очистити полотно?"
            )
            if result:
                self.canvas.delete("all")
                self.shapes = [None] * self.MAX_SHAPES
                self.shape_count = 0
                self.update_status()

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

        tool_names = {
            'point': 'Точка',
            'line': 'Лінія',
            'rect': 'Прямокутник',
            'ellipse': 'Еліпс'
        }
        current_tool_name = tool_names.get(self.current_tool, "Невідомо")
        self.root.title(f"{self.base_title} - Вибрано:
{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\n"
            "Максимальна кількість об'єктів: 129"
        )
```

```
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")
        }

        subsample_factor = 24

        for tool, (img_file, tooltip_text) in buttons_data.items():
            try:
                original_img = tk.PhotoImage(file=img_file)
                resized_img = original_img.subsample(subsample_factor,
subsample_factor)
                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)
```

**editors.py:**

```
import tkinter as tk
from abc import ABC, abstractmethod
from shapes import Shape, PointShape, LineShape, RectShape, EllipseShape

class Editor(ABC):

    def __init__(self, canvas):
        self.canvas = canvas
        self.start_x = 0
        self.start_y = 0
        self.is_drawing = False

    @abstractmethod
    def on_mouse_down(self, event):
        pass

    @abstractmethod
    def on_mouse_move(self, event):
        pass

    @abstractmethod
    def on_mouse_up(self, event):
        pass

class ShapeEditor(Editor):
    def __init__(self, canvas: tk.Canvas):
        super().__init__(canvas)
        self.temp_id = None

    def on_mouse_down(self, event) -> None:
        self.start_x = event.x
        self.start_y = event.y
        self.is_drawing = True

    def on_mouse_move(self, event) -> None:
        if self.is_drawing:
            if self.temp_id:
                self.canvas.delete(self.temp_id)
```

```
            self.temp_id = self.create_preview(event)

    def on_mouse_up(self, event) -> Shape:
        if self.is_drawing:
            if self.temp_id:
                self.canvas.delete(self.temp_id)
                self.temp_id = None
            self.is_drawing = False
            return self.create_shape(event)
        return None

    def create_preview(self, event):
        return None

    @abstractmethod
    def create_shape(self, event) -> Shape:
        pass

class PointEditor(ShapeEditor):
    def create_shape(self, event) -> Shape:
        return PointShape(self.start_x, self.start_y, self.start_x,
self.start_y)

class LineEditor(ShapeEditor):
    def create_preview(self, event):
        return self.canvas.create_line(
            self.start_x, self.start_y, event.x, event.y,
            fill="red"
        )

    def create_shape(self, event) -> Shape:
        return LineShape(self.start_x, self.start_y, event.x, event.y)

class RectEditor(ShapeEditor):
    def create_preview(self, event):
        x1, y1, x2, y2 = self.calculate_bounds(event)
        return self.canvas.create_rectangle(
            x1, y1, x2, y2,
            outline="red"
        )
```

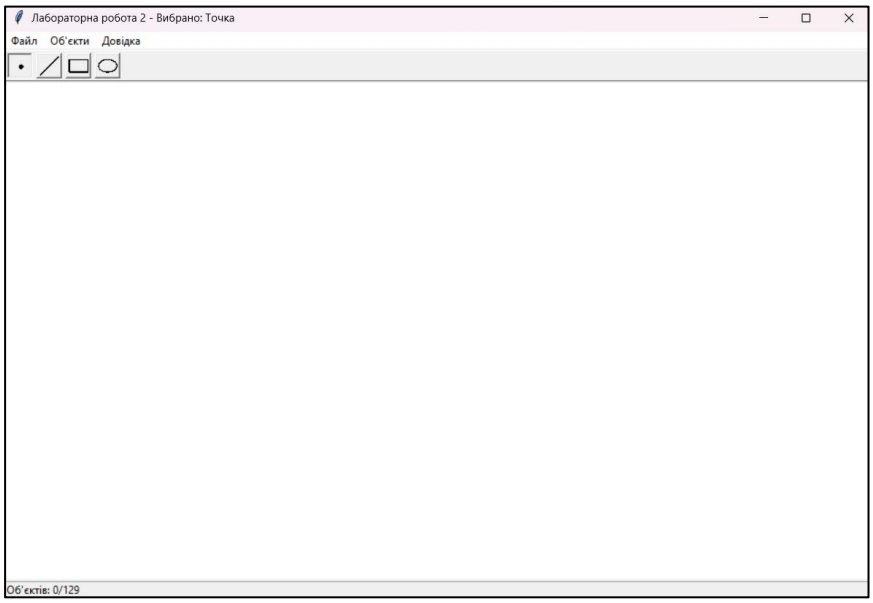
```
    def create_shape(self, event) -> Shape:
        x1, y1, x2, y2 = self.calculate_bounds(event)
        return RectShape(x1, y1, x2, y2)

    def calculate_bounds(self, event):
        dx = abs(event.x - self.start_x)
        dy = abs(event.y - self.start_y)

        x1 = self.start_x - dx
        y1 = self.start_y - dy
        x2 = self.start_x + dx
        y2 = self.start_y + dy

        return x1, y1, x2, y2

class EllipseEditor(ShapeEditor):
    def create_preview(self, event):
        return self.canvas.create_oval(
            self.start_x, self.start_y, event.x, event.y,
            outline="red"
        )

    def create_shape(self, event) -> Shape:
        return EllipseShape(self.start_x, self.start_y, event.x, event.y)
```

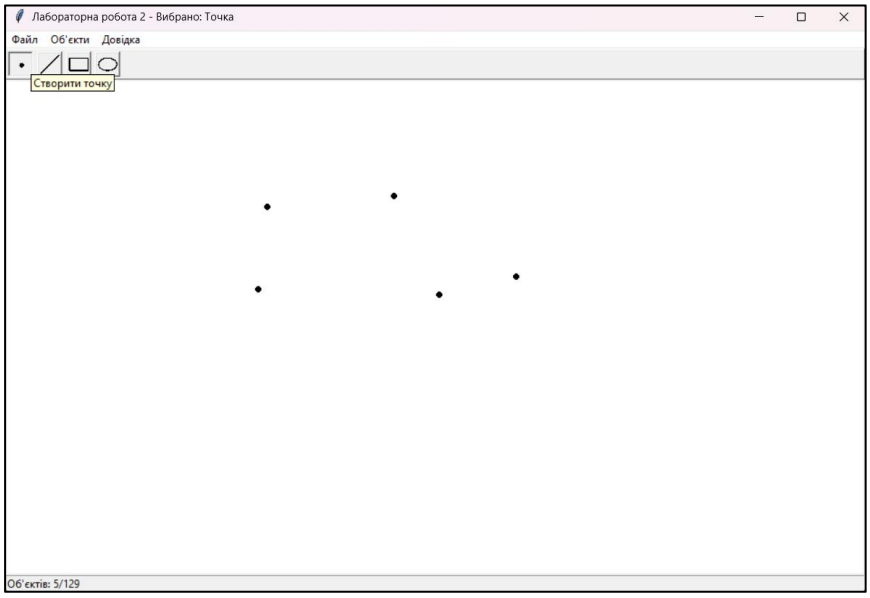
**shapes.py:**

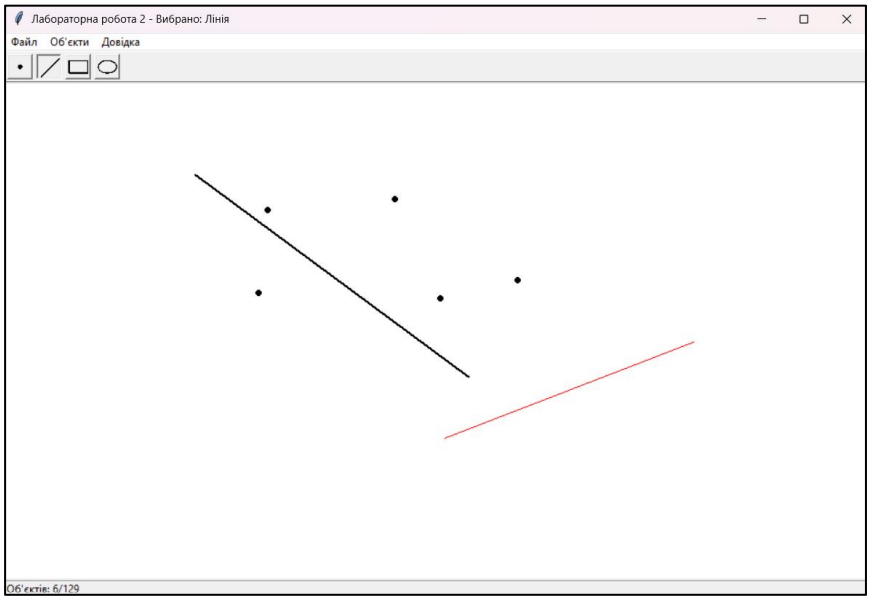
```
from abc import ABC, abstractmethod

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):
        pass

class PointShape(Shape):
    def show(self, canvas):
        radius = 3
        canvas.create_oval(
            self.x1 - radius, self.y1 - radius,
            self.x1 + radius, self.y1 + radius,
            fill=self.color, outline=self.color
        )

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

class RectShape(Shape):
    def show(self, canvas):
        canvas.create_rectangle(
            self.x1, self.y1, self.x2, self.y2,
            fill=None, outline=self.color, width=self.width
        )
```

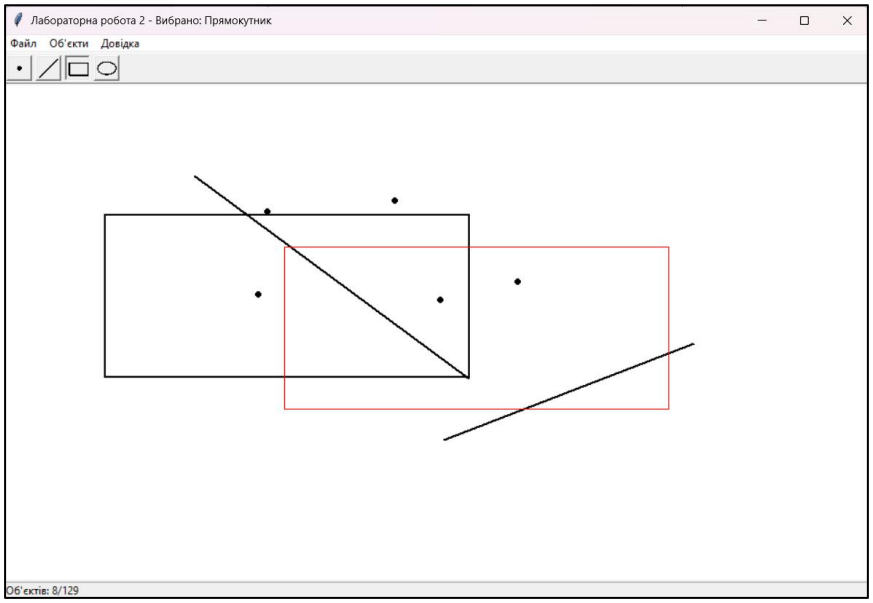
```
class EllipseShape(Shape):
    def show(self, canvas):
        canvas.create_oval(
            self.x1, self.y1, self.x2, self.y2,
            fill="orange", outline=self.color, width=self.width
        )
```

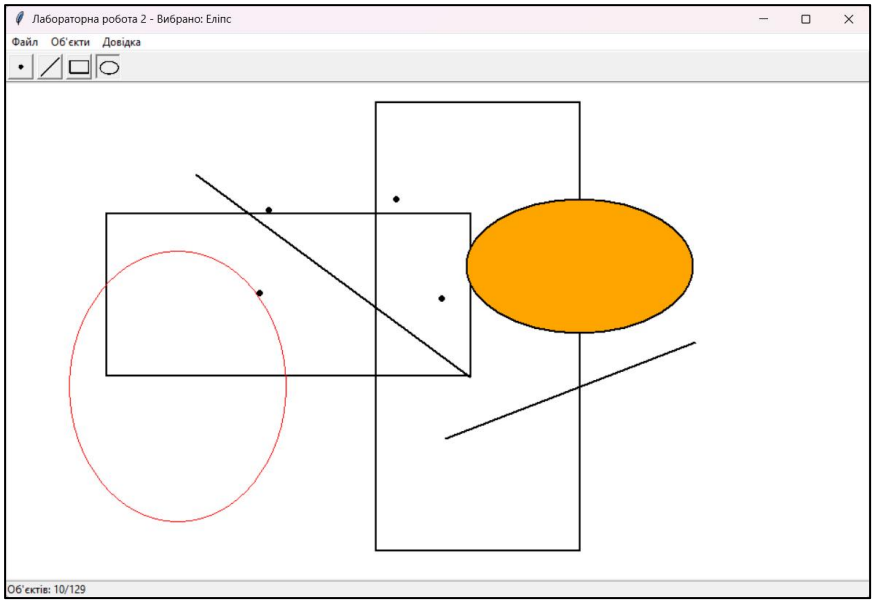
# Скріншоти:


Лабораторна робота 2 - Вибрано: Точка
Файл Об'єкти Довідка
Об'єктів: 0/129

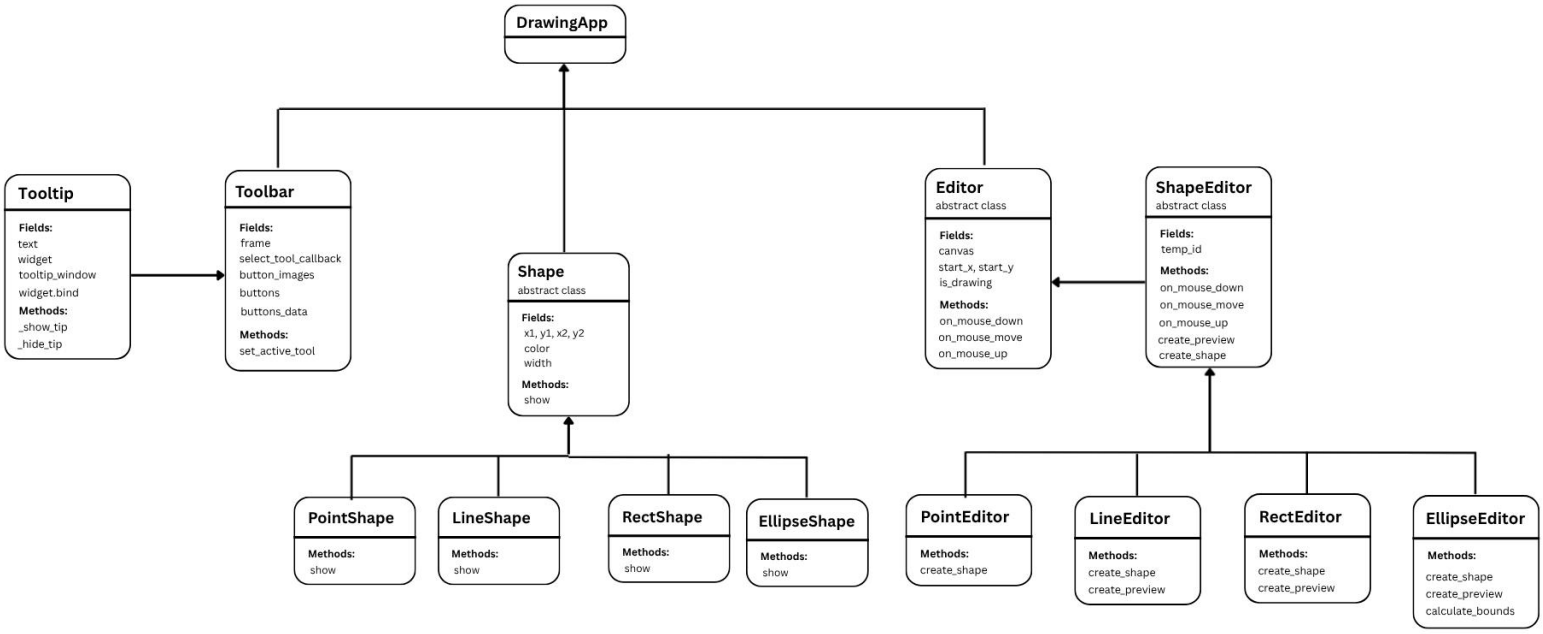

Лабораторна робота 2 - Вибрано: Точка
Файл Об'єкти Довідка
Створити точку
Об'єктів: 5/129


Лабораторна робота 2 - Вибрано: Лінія
Файл Об'єкти Довідка
Об'єктів: 6/129


Лабораторна робота 2 - Вибрано: Прямокутник
Файл
Об'єкти
Довідка
Об'єктів: 8/129


Лабораторна робота 2 - Вибрано: Еліпс
Файл
Об'єкти
Довідка
Об'єктів: 10/129

# 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.

Було додано власний Toolbar для вибору фігури для малювання: на вікно було додано 4 кнопки, в кожну з яких завантажується відповідне фігурі зображення.

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

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 Editor також є абстрактним і описує поведінку інструмента під час натискання, пересування та відпускання кнопки миші. Конкретні редактори (PointEditor, LineEditor, RectEditor, EllipseEditor) успадковують поведінку від загального ShapeEditor, реалізують попередній перегляд фігури під час малювання і створення фінальної фігури на полотні.

Клас DrawingApp відповідає за інтерфейс і взаємодію: створення меню, полотна (canvas), статус-бару, вибір інструментів, обробку подій миші та збереження списку об'єктів (з обмеженням 129 елементів). Також передбачено очищення полотна та інформаційне діалогове вікно «Про програму».