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 №2

з дисципліни
«Об’єктно орієнтоване програмування»на тему “Розробка графічного редактора об’єктів на C++”

Виконала:
Студентка групи ІМ-33
Пилипчук Вероніка Олексіївна
Номер у списку групи: 18

Перевірив:
Порєв В.М.

Київ 2024

Варіант завдання:

1. Для усіх варіантів завдань необхідно дотримуватися вимог та положень, викладених вище у порядку виконання роботи та методичних рекомендаціях.
2. У звіті повинна бути схема успадкування класів – діаграма класів
3. Для вибору типу об’єкта в графічному редакторі Lab2 повинно бути меню "Об’єкти" з чотирма підпунктами. Меню "Об’єкти" повинно бути праворуч меню "Файл" та ліворуч меню "Довідка". Підпункти меню "Об’єкти" містять назви українською мовою геометричних форм – так, як наведено вище у порядку виконання роботи та методичних рекомендаціях. Геометричні форми згідно варіанту завдання.
4. динамічний масив Shape**pcshape
5. гумовий слід при вводі об’єктів: суцільна лінія синього кольору
6. прямокутник: по двом протилежним кутам + чорний контур з жовтим заповненням
7. еліпс: від центру до одного з кутів охоплюючого прямокутника + чорний контур з сірим заповненням
8. позначка поточного об’єкту, що вводиться: в меню

Вихідний текст головного файлу (Lab2.tsx):

```
import React, { useEffect, useRef, useState } from "react";
import { Menu, MenuProps } from "antd";
import { Dot, Ellipse, Line, Rectangle, Shape } from "@/app/modules/Shape";
import { items } from "./constants";

export const Lab2: React.FC = () => {
  const canvasRef = useRef<HTMLCanvasElement | null>(null);
  const [shapes, setShapes] = useState<Shape[]>([]);
  const [currentTab, setCurrentTab] = useState("");
  const [isDrawing, setIsDrawing] = useState(false);
  const [lastPosition, setLastPosition] = useState<{
    x: number;
    y: number;
  } | null>(null);
  const [previewShape, setPreviewShape] = useState<Shape | null>(null);

  const onClick: MenuProps["onClick"] = (e) => {
    setCurrentTab(e.key);
  };

  const drawShape = (event: MouseEvent, preview = false) => {
    if (!canvasRef.current) return;

    const rect = canvasRef.current.getBoundingClientRect();
    const x = event.clientX - rect.left;
    const y = event.clientY - rect.top;

    let newShape: Shape | null = null;

    switch (currentTab) {
      case "dot":
        newShape = new Dot(x, y);
        break;
      case "line":
        if (lastPosition) {
```

```
        newShape = new Line(lastPosition.x, lastPosition.y, x, y);
      }
      break;
    case "rectangle":
      if (lastPosition) {
        const width = x - lastPosition.x;
        const height = y - lastPosition.y;
        newShape = new Rectangle(
          lastPosition.x,
          lastPosition.y,
          width,
          height
        );
      }
      break;
    case "ellipse":
      if (lastPosition) {
        const radiusX = Math.abs(x - lastPosition.x);
        const radiusY = Math.abs(y - lastPosition.y);
        newShape = new Ellipse(
          lastPosition.x,
          lastPosition.y,
          radiusX,
          radiusY
        );
      }
      break;
  }

  if (preview && newShape) {
    setPreviewShape(newShape);
  } else if (newShape) {
    setShapes((prev) => [...prev, newShape]);
  }
};

useEffect(() => {
  const canvas = canvasRef.current;
  if (canvas) {
    const ctx = canvas.getContext("2d");
    if (ctx) {
      ctx.clearRect(0, 0, canvas.width, canvas.height);

      shapes.forEach((shape) => shape.draw(ctx));

      if (previewShape) {
        previewShape.draw(ctx);
      }
    }
  }
}, [shapes, previewShape]);

const startDrawing = (event: MouseEvent) => {
  const rect = canvasRef.current!.getBoundingClientRect();
  const x = event.clientX - rect.left;
  const y = event.clientY - rect.top;
  setLastPosition({ x, y });
  setIsDrawing(true);
};

const stopDrawing = () => {
  setIsDrawing(false);
  setLastPosition(null);
  setPreviewShape(null);
```

```
  };

  useEffect(() => {
    const canvas = canvasRef.current;

    const mouseDownHandler = (event: MouseEvent) => {
      startDrawing(event);
    };

    const mouseMoveHandler = (event: MouseEvent) => {
      if (isDrawing) {
        drawShape(event, true);
      }
    };

    const mouseUpHandler = (event: MouseEvent) => {
      if (isDrawing) {
        drawShape(event);
        stopDrawing();
      }
    };

    if (canvas) {
      canvas.addEventListener("mousedown", mouseDownHandler);
      canvas.addEventListener("mousemove", mouseMoveHandler);
      canvas.addEventListener("mouseup", mouseUpHandler);
      canvas.addEventListener("mouseleave", stopDrawing);

      return () => {
        canvas.removeEventListener("mousedown", mouseDownHandler);
        canvas.removeEventListener("mousemove", mouseMoveHandler);
        canvas.removeEventListener("mouseup", mouseUpHandler);
        canvas.removeEventListener("mouseleave", stopDrawing);
      };
    }
  }, [isDrawing]);

  return (
    <div>
      <div>
        <Menu
          onClick={onClick}
          selectedKeys={[currentTab]}
          mode="horizontal"
          items={items}
        />
      </div>
      <canvas
        ref={canvasRef}
        style={{ margin: "10px", boxShadow: "0 4px 10px rgba(0, 0, 0, 0.2)" }}
        width={700}
        height={400}
      />
    </div>
  );
};
```

Вихідні тексти модулів:

1. Файл з елементами меню constants.ts

```
import { MenuProps } from "antd";

type MenuItem = Required<MenuProps>["items"][number];
export const items: MenuItem[] = [
  {
    label: "Файл",
    key: "file",
  },
  {
    label: "Об’єкти",
    key: "objects",
    children: [
      { label: "Крапка", key: "dot" },
      { label: "Лінія", key: "line" },
      { label: "Прямокутник", key: "rectangle" },
      { label: "Еліпс", key: "ellipse" },
    ],
  },
  {
    label: "Довідка",
    key: "note",
  },
];
```

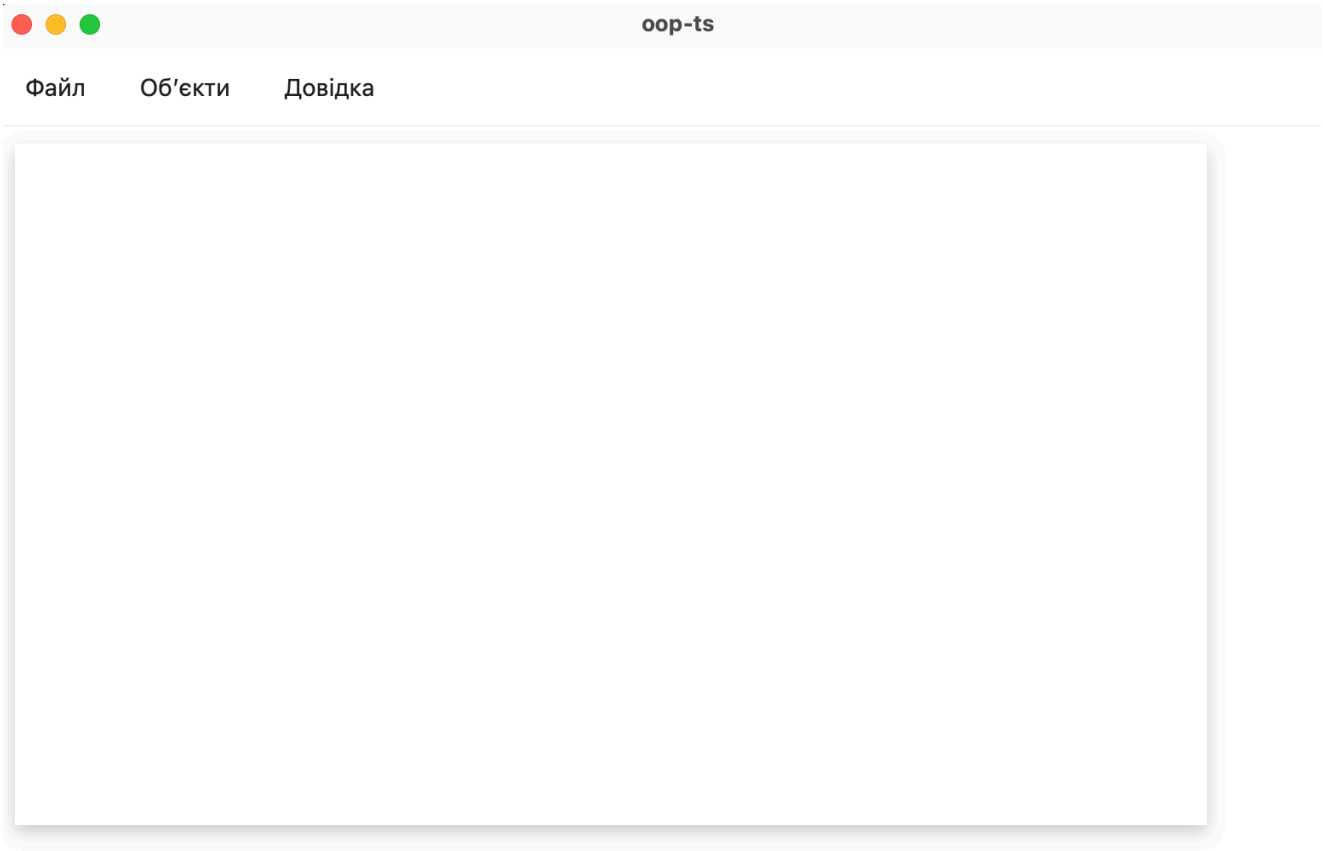
2. Модуль класу Shape та наслідуваними від нього класами

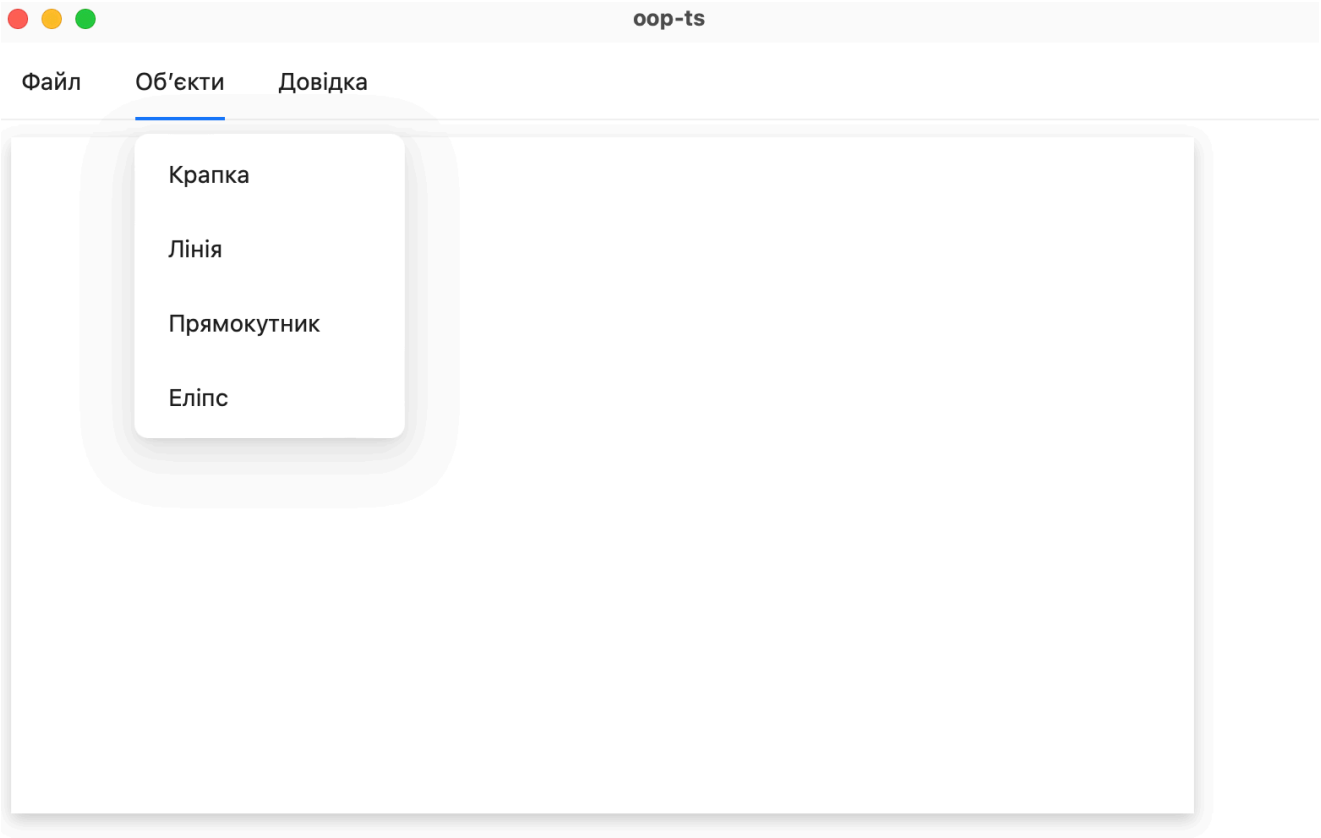
```
export abstract class Shape {
  protected color: string;
  constructor(color: string) {
    this.color = color;
  }

  abstract draw(ctx: CanvasRenderingContext2D): void;
}

export class Dot extends Shape {
  private x: number;
  private y: number;

  constructor(x: number, y: number, color: string = "blue") {
    super(color);
    this.x = x;
    this.y = y;
  }

  draw(ctx: CanvasRenderingContext2D): void {
    ctx.beginPath();
    ctx.arc(this.x, this.y, 5, 0, Math.PI * 2);
    ctx.fillStyle = this.color;
    ctx.fill();
  }
}

export class Line extends Shape {
  private startX: number;
  private startY: number;
```

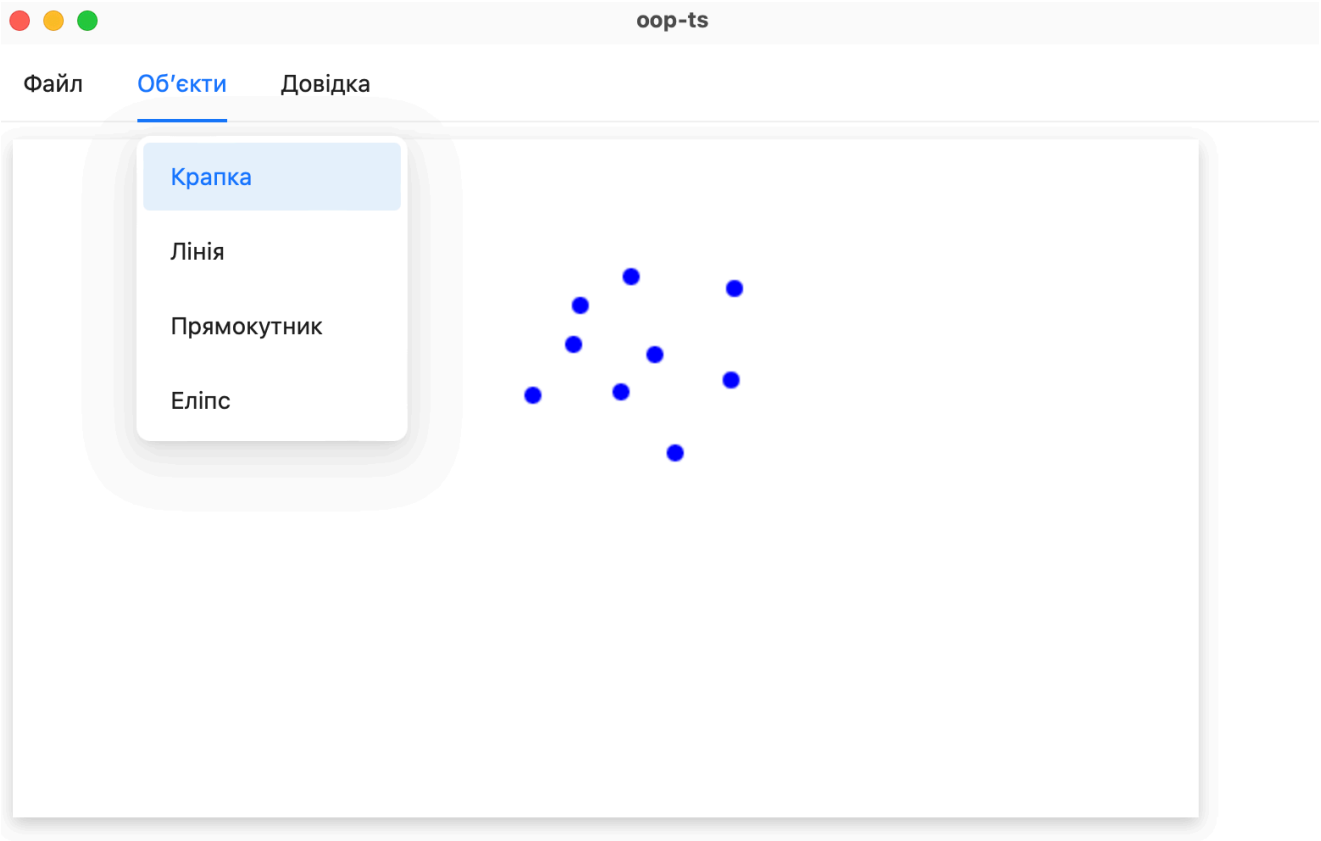
```
  private endX: number;
  private endY: number;

  constructor(
    startX: number,
    startY: number,
    endX: number,
    endY: number,
    color: string = "blue"
  ) {
    super(color);
    this.startX = startX;
    this.startY = startY;
    this.endX = endX;
    this.endY = endY;
  }

  draw(ctx: CanvasRenderingContext2D): void {
    ctx.beginPath();
    ctx.moveTo(this.startX, this.startY);
    ctx.lineTo(this.endX, this.endY);
    ctx.strokeStyle = this.color;
    ctx.lineWidth = 2;
    ctx.stroke();
  }
}

export class Rectangle extends Shape {
  private x: number;
  private y: number;
  private width: number;
  private height: number;

  constructor(
    x: number,
    y: number,
    width: number,
    height: number,
    color: string = "yellow"
  ) {
    super(color);
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
  }

  draw(ctx: CanvasRenderingContext2D): void {
    ctx.fillStyle = this.color;
    ctx.fillRect(this.x, this.y, this.width, this.height);
    ctx.strokeStyle = "black";
    ctx.strokeRect(this.x, this.y, this.width, this.height);
  }
}

export class Ellipse extends Shape {
  private x: number;
  private y: number;
  private radiusX: number;
  private radiusY: number;

  constructor(
    x: number,
    y: number,
```

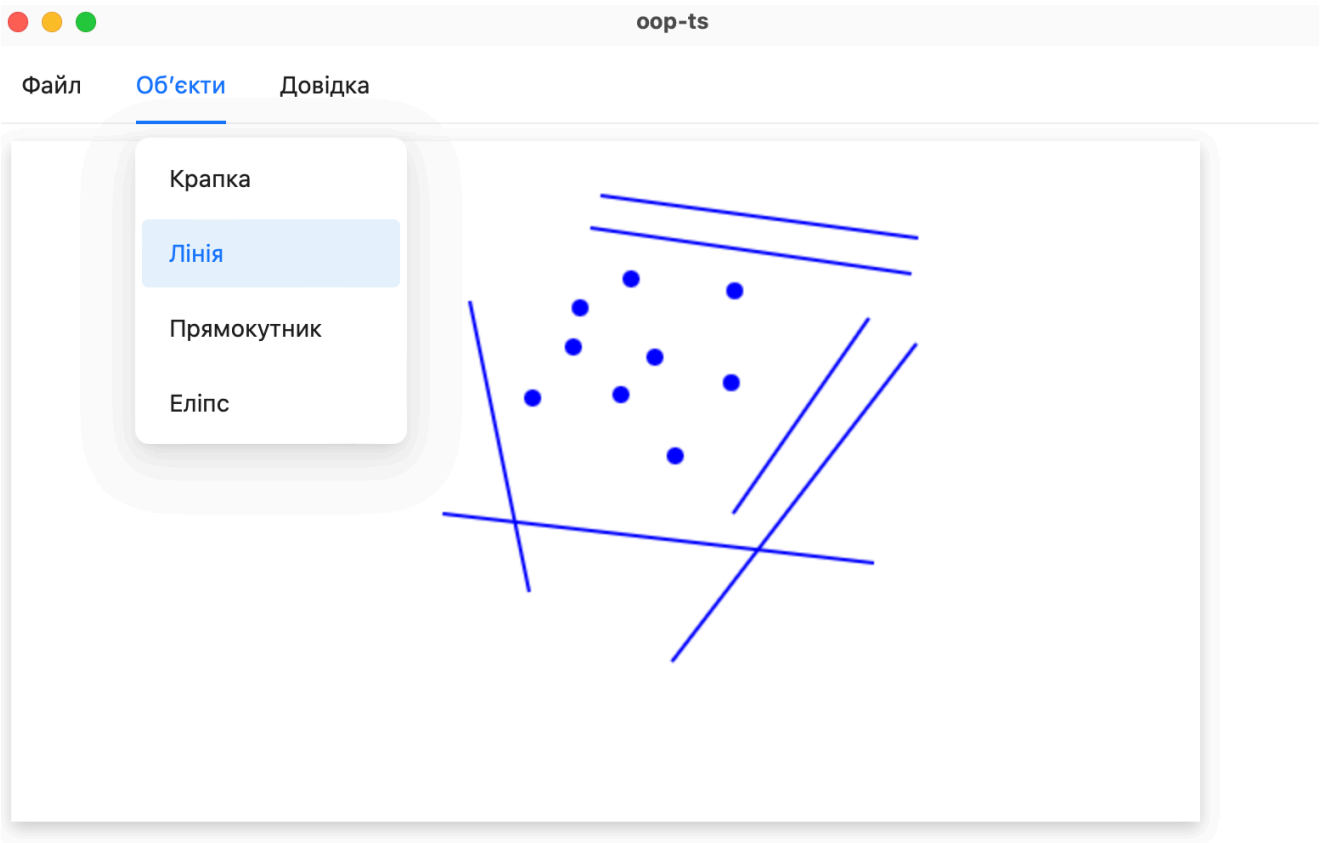
```
    radiusX: number,
    radiusY: number,
    color: string = "grey"
  ) {
    super(color);
    this.x = x;
    this.y = y;
    this.radiusX = radiusX;
    this.radiusY = radiusY;
  }

  draw(ctx: CanvasRenderingContext2D): void {
    ctx.beginPath();
    ctx.ellipse(this.x, this.y, this.radiusX, this.radiusY, 0, 0, Math.PI * 2);
    ctx.fillStyle = this.color;
    ctx.fill();
    ctx.strokeStyle = "black";
    ctx.stroke();
  }
}
```

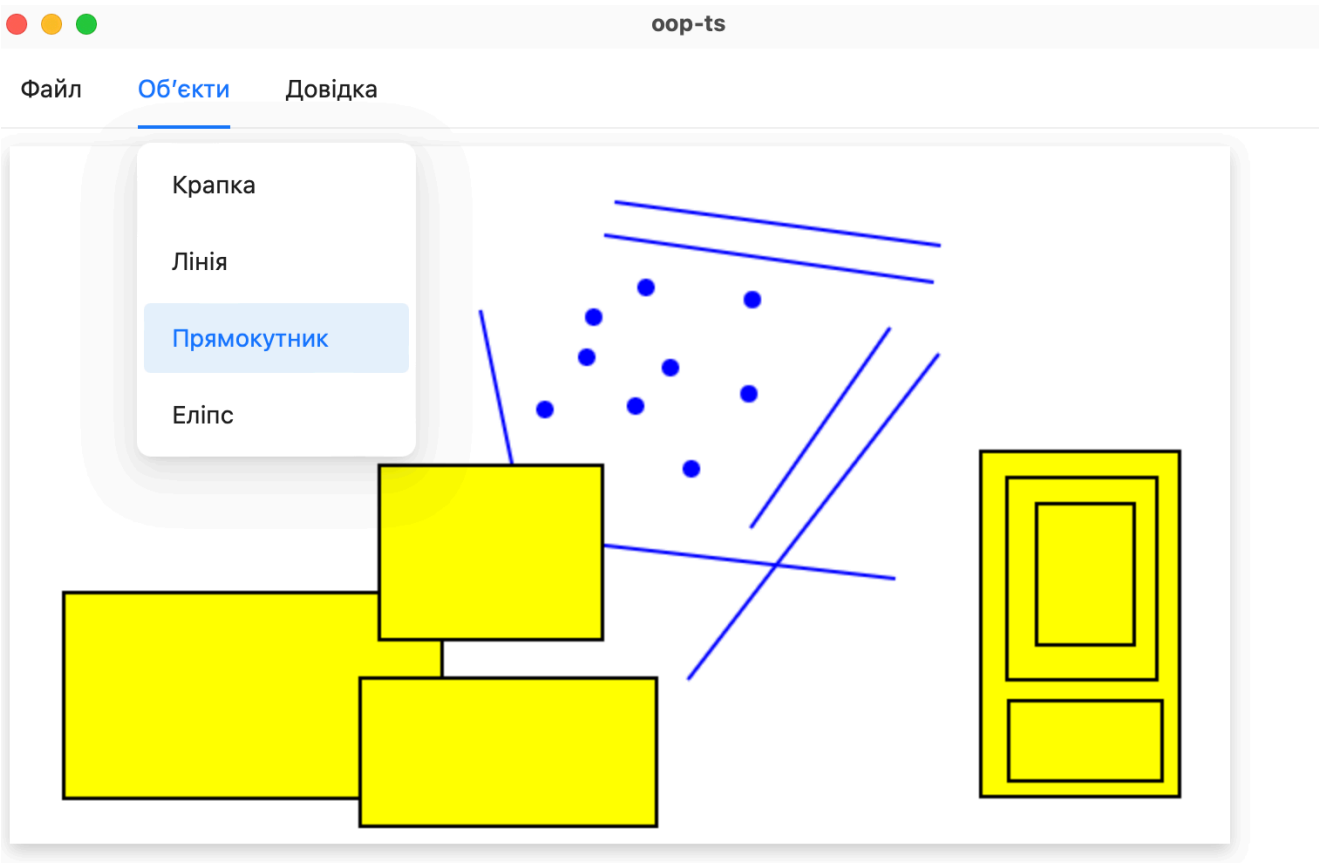
Скріншоти роботи виконаної програми:

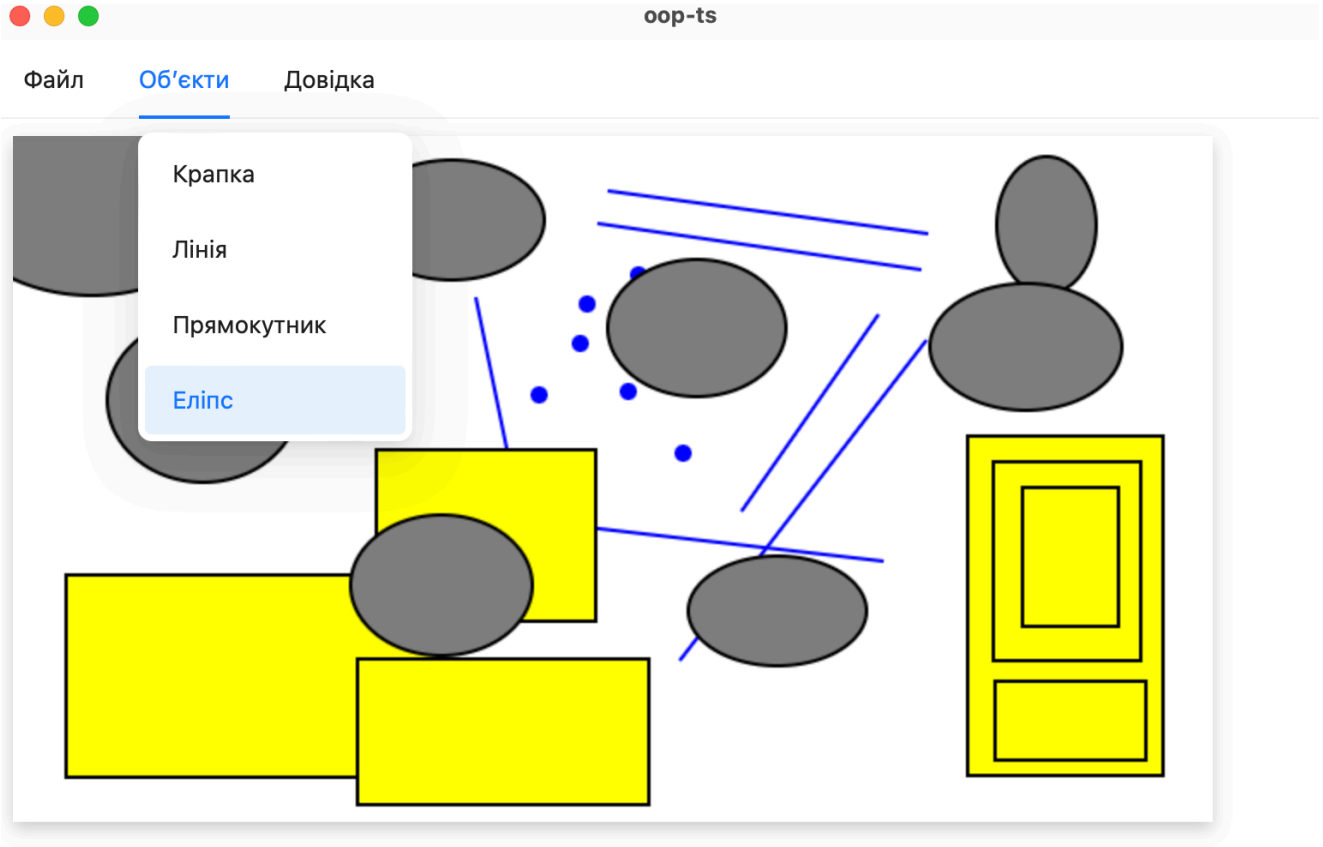





oop-ts
Файл
Об'єкти
Довідка
Крапка
Лінія
Прямокутник
Еліпс


oop-ts
Файл
Об'єкти
Довідка
Крапка
Лінія
Прямокутник
Еліпс


oop-ts
Файл
Об'єкти
Довідка
Крапка
Лінія
Прямокутник
Еліпс


oop-ts
Файл
Об'єкти
Довідка
Крапка
Лінія
Прямокутник
Еліпс

Висновки:
Моя лабораторна робота виконана із використанням бібліотек для створення користувацьких інтерфейсів на Typescript із використанням об’єктно-орієнтованого підходу.
Спробувала виконати поставлену задачу згідно зі своїм варіантом. Створила абстрактний клас Shape, який послужив своєрідним “каркасом” для наслідування. Наслідування й поліморфізм - два основних принципи об’єктно-орієнтованого програмування, при цьому одне є неможливим без іншого. Наслідування дає змогу використовувати вже написаний код без повторення з можливістю змінювати методи під потреби, а поліморфізм - викликати ці методи, змінюючи тільки назву самого класу, роблячи код більш гнучким.