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»на тему
“Вдосконалення структури коду 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. 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.
3. Окрім чотирьох типів фігур, які були у попередніх лаб. №2 та 3, запрограмувати ще введення та відображення двох нових фігур – лінія з кружечками та каркас куба.
4. 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).

1. Вихідний текст головного файлу (Lab4.tsx):

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

export const Lab4: 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;
    case "dumbbell":
      if (lastPosition) {
        newShape = new LineWithCircles(lastPosition.x, lastPosition.y, x, y);
      }
      break;
    case "cube":
      if (lastPosition) {
        newShape = new Cube(
          lastPosition.x,
          lastPosition.y,
          x - lastPosition.x,
          y - lastPosition.y
        );
      }
      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.drawPreview(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 (
    <>
      <Menu
        onClick={onClick}
        selectedKeys={[currentTab]}
        mode="horizontal"
        items={items}
      />
      <Toolbar
        chosenItem={currentTab}
        setChosenItem={(key: string) => setCurrentTab(key)}
      />
      <canvas
        ref={canvasRef}
        style={{ margin: "10px", boxShadow: "0 4px 10px rgba(0, 0, 0, 0.2)" }}
        width={700}
        height={400}
      />
    </>
  );
};
```

2. Файл з елементами меню constants.ts

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

type MenuItem = Required<MenuProps>["items"][number];
export const items: MenuItem[] = [
  {
    label: "Файл",
    key: "file",
    children: [
      { label: "Створити", key: "create" },
      { label: "Відкрити", key: "open" },
      { label: "Зберегти", key: "save" },
      { type: "divider" },
      { label: "Друк", key: "print" },
      { type: "divider" },
      { label: "Вихід", key: "exit" },
    ],
  },
  {
    label: "Об’єкти",
    key: "objects",
    children: [
      { label: "Крапка", key: "dot" },
      { label: "Лінія", key: "line" },
      { label: "Прямокутник", key: "rectangle" },
      { label: "Еліпс", key: "ellipse" },
      { label: "Лінія з кружечками", key: "dumbbell" },
      { label: "Куб", key: "cube" },
    ],
  },
  {
    label: "Довідка",
    key: "note",
  },
];
```

3. Модуль Toolbar.tsx

```
import Icon, { BorderOutlined, LineOutlined } from "@ant-design/icons";
import { Button, Tooltip } from "antd";
import { DotIcon } from "../../assets/icons/DotIcon";
import { EllipseIcon } from "../../assets/icons/EllipseIcon";
import { CubeIcon } from "@/app/assets/icons/CubeIcon";
import { ShareIcon } from "@/app/assets/icons/ShareIcon";

interface ToolbarProps {
  chosenItem: string;
  setChosenItem: (key: string) => void;
}

export const Toolbar = ({ chosenItem, setChosenItem }: ToolbarProps) => {
  const toolbarItems = [
    {
      key: "dot",
      icon: <Icon component={DotIcon} style={{ width: 12, height: 12 }} />,
      tooltipTitle: "Крапка",
    },
    {
      key: "line",
      icon: <LineOutlined style={{ fontSize: 20 }} />,
      tooltipTitle: "Лінія",
    },
    {
      key: "rectangle",
      icon: <BorderOutlined style={{ fontSize: 20 }} />,
```

```
      tooltipTitle: "Прямокутник",
    },
    {
      key: "ellipse",
      icon: <EllipseIcon style={{ width: 22, height: 20 }} />,
      tooltipTitle: "Еліпс",
    },
    {
      key: "dumbbell",
      icon: <ShareIcon style={{ width: 22, height: 20 }} />,
      tooltipTitle: "Лінія з кружечками",
    },
    {
      key: "cube",
      icon: <CubeIcon style={{ width: 22, height: 20 }} />,
      tooltipTitle: "Куб",
    },
  ];
  return (
    <div
      style={{
        width: "100%",
        height: "40px",
        marginTop: 10,
        backgroundColor: "#e0edff",
        borderColor: "black",
        borderTopWidth: 1,
        borderBottomWidth: 1,
        display: "flex",
        flexDirection: "row",
        gap: 10,
        alignItems: "center",
        justifyContent: "flex-start",
        padding: 5,
      }}
    >
      {toolbarItems.map((item) => (
        <Tooltip key={item.key} placement="bottom" title={item.tooltipTitle}>
          <Button
            type={chosenItem === item.key ? "primary" : "text"}
            onClick={() => {
              setChosenItem(item.key);
            }}
          >
            {item.icon}
          </Button>
        </Tooltip>
      ))}
    </div>
  );
};
```

4. Новий модуль класів MyEditor.ts з оновленою реалізацією гумового сліду пунктирною лінією та двома варіантами реалізації множинного наслідування у Typescript

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

  abstract draw(ctx: CanvasRenderingContext2D): void;
  abstract drawPreview(ctx: CanvasRenderingContext2D): void;
}

export class Dot extends Shape {
  protected x: number;
  protected 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();
  }

  drawPreview(ctx: CanvasRenderingContext2D): void {
    this.draw(ctx);
  }
}

export class Line extends Shape {
  protected startX: number;
  protected startY: number;
  protected endX: number;
  protected 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.setLineDash([]);
    ctx.strokeStyle = this.color;
    ctx.lineWidth = 2;
    ctx.stroke();
```

```
  }

  drawPreview(ctx: CanvasRenderingContext2D): void {
    ctx.beginPath();
    ctx.moveTo(this.startX, this.startY);
    ctx.lineTo(this.endX, this.endY);
    ctx.setLineDash([5, 5]);
    ctx.strokeStyle = "black";
    ctx.lineWidth = 2;
    ctx.stroke();
  }
}

export class Rectangle extends Shape {
  protected x: number;
  protected y: number;
  protected width: number;
  protected 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.setLineDash([]);
    ctx.strokeStyle = "black";
    ctx.strokeRect(this.x, this.y, this.width, this.height);
  }

  drawPreview(ctx: CanvasRenderingContext2D): void {
    ctx.setLineDash([5, 5]);
    ctx.strokeStyle = "black";
    ctx.strokeRect(this.x, this.y, this.width, this.height);
  }
}

export class Ellipse extends Shape {
  protected x: number;
  protected y: number;
  protected radiusX: number;
  protected 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.setLineDash([]);
    ctx.strokeStyle = "black";
    ctx.stroke();
  }

  drawPreview(ctx: CanvasRenderingContext2D): void {
    ctx.beginPath();
    ctx.ellipse(this.x, this.y, this.radiusX, this.radiusY, 0, 0, Math.PI * 2);
    ctx.setLineDash([5, 5]);
    ctx.strokeStyle = "black";
    ctx.stroke();
  }
}

interface Circle {
  drawCircle(
    ctx: CanvasRenderingContext2D,
    x: number,
    y: number,
    radius: number,
    color: string
  ): void;
}

export class LineWithCircles extends Line implements Circle {
  private radius: number;

  constructor(
    startX: number,
    startY: number,
    endX: number,
    endY: number,
    radius: number = 5,
    color: string = "blue"
  ) {
    super(startX, startY, endX, endY, color);
    this.radius = radius;
  }

  drawCircle(
    ctx: CanvasRenderingContext2D,
    x: number,
    y: number,
    radius: number,
    color: string
  ): void {
    ctx.beginPath();
    ctx.arc(x, y, radius, 0, Math.PI * 2);
    ctx.fillStyle = color;
    ctx.fill();
  }

  draw(ctx: CanvasRenderingContext2D): void {
    super.draw(ctx);
    this.drawCircle(ctx, this.startX, this.startY, this.radius, this.color);
    this.drawCircle(ctx, this.endX, this.endY, this.radius, this.color);
  }

  drawPreview(ctx: CanvasRenderingContext2D): void {
    super.drawPreview(ctx);
    this.drawCircle(ctx, this.startX, this.startY, this.radius, "black");
```

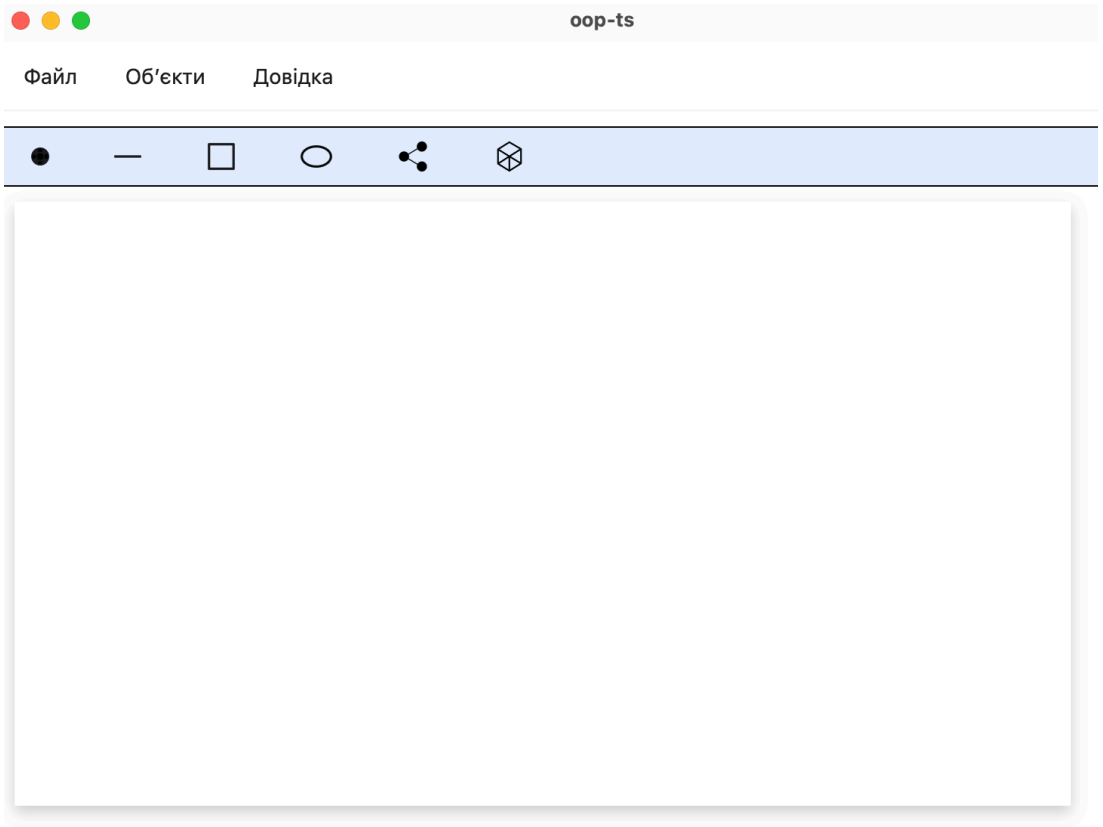
```
    this.drawCircle(ctx, this.endX, this.endY, this.radius, "black");
  }
}

type Constructor<T = object> = new (...args: any[]) => T;

function LineMixin<TBase extends Constructor>(Base: TBase) {
  return class extends Base {
    drawLine(
      ctx: CanvasRenderingContext2D,
      startX: number,
      startY: number,
      endX: number,
      endY: number,
      color: string = "black"
    ) {
      ctx.beginPath();
      ctx.moveTo(startX, startY);
      ctx.lineTo(endX, endY);
      ctx.strokeStyle = color;
      ctx.stroke();
    }
  };
}

export class Cube extends LineMixin(Rectangle) {
  private depth: number;

  constructor(
    x: number,
    y: number,
    width: number,
    height: number,
    depth: number = 30,
    color: string = "transparent"
  ) {
    super(x, y, width, height, color);
    this.depth = depth;
  }

  drawSides(ctx: CanvasRenderingContext2D): void {
    const backX = this.x + this.depth;
    const backY = this.y + this.depth;

    ctx.strokeStyle = "black";
    ctx.strokeRect(backX, backY, this.width, this.height);

    this.drawLine(ctx, this.x, this.y, backX, backY);
    this.drawLine(ctx, this.x + this.width, this.y, backX + this.width, backY);
    this.drawLine(
      ctx,
      this.x,
      this.y + this.height,
      backX,
      backY + this.height
    );
    this.drawLine(
      ctx,
      this.x + this.width,
      this.y + this.height,
      backX + this.width,
      backY + this.height
    );
  }

  draw(ctx: CanvasRenderingContext2D): void {
    super.draw(ctx);
```

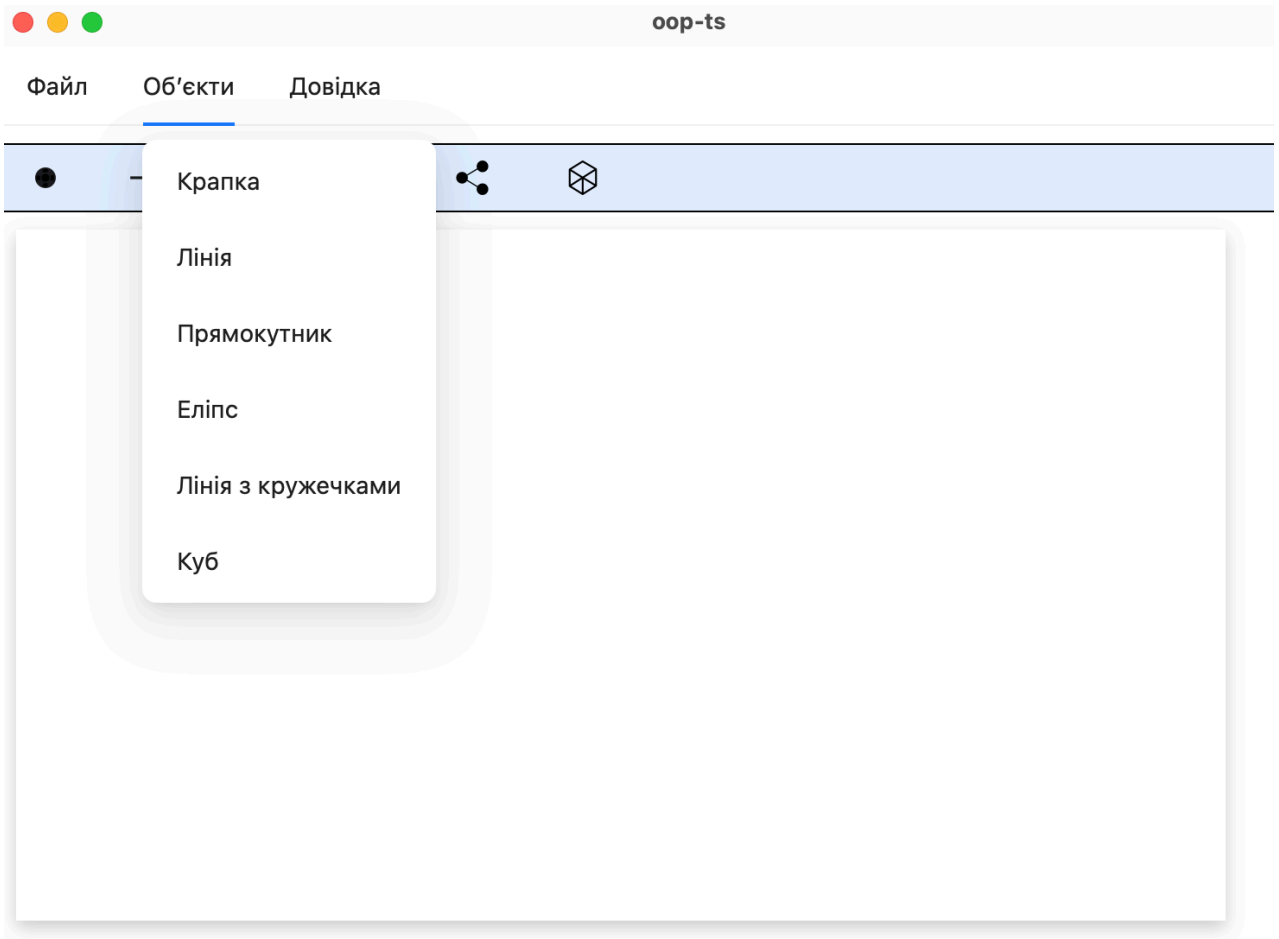
```
    this.drawSides(ctx);
  }

  drawPreview(ctx: CanvasRenderingContext2D): void {
    super.drawPreview(ctx);
    this.drawSides(ctx);
  }
}
```

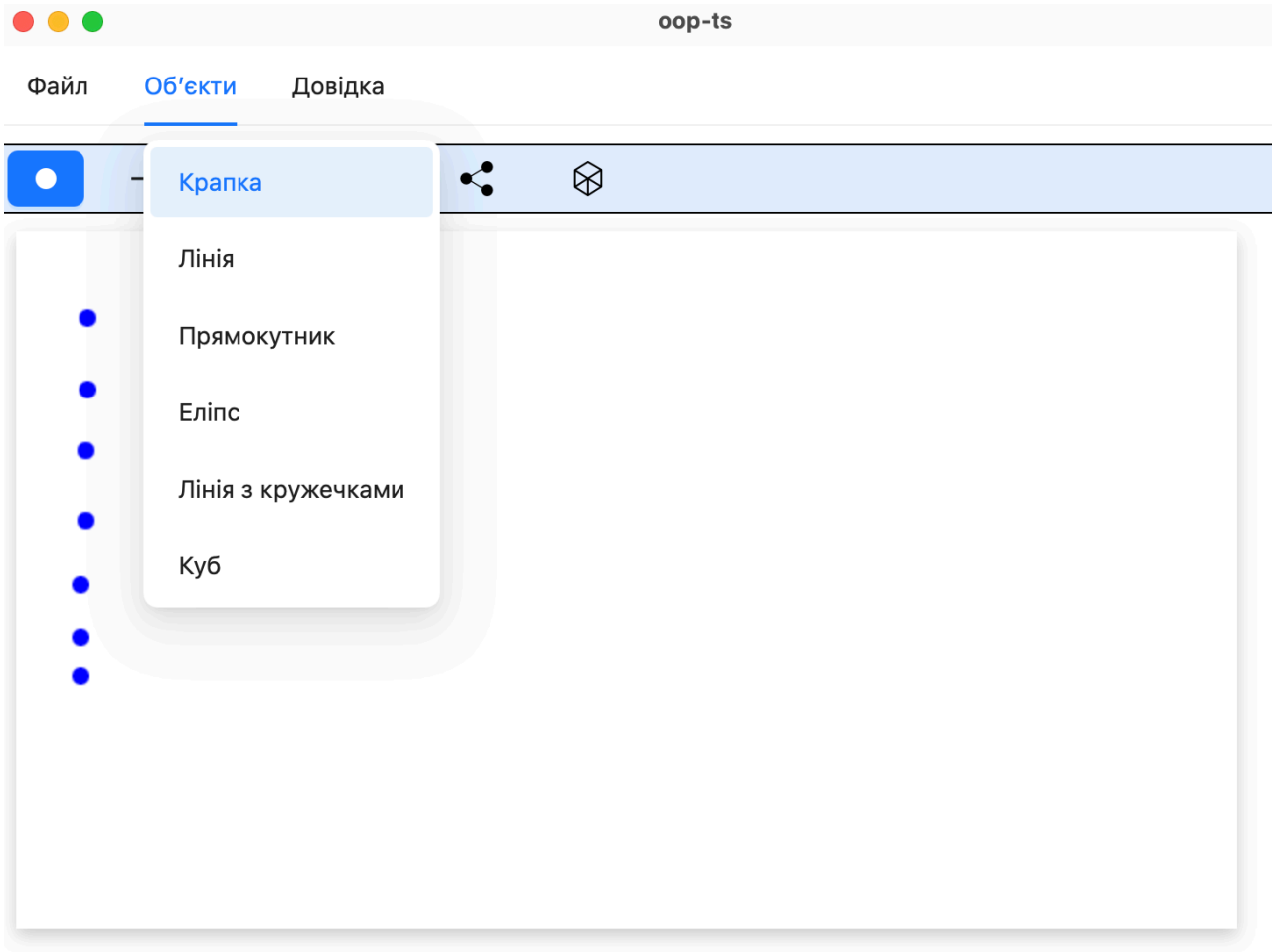
5. Модулі компонентів з кастомними іконками для куба та лінії з кружечками на кінцях

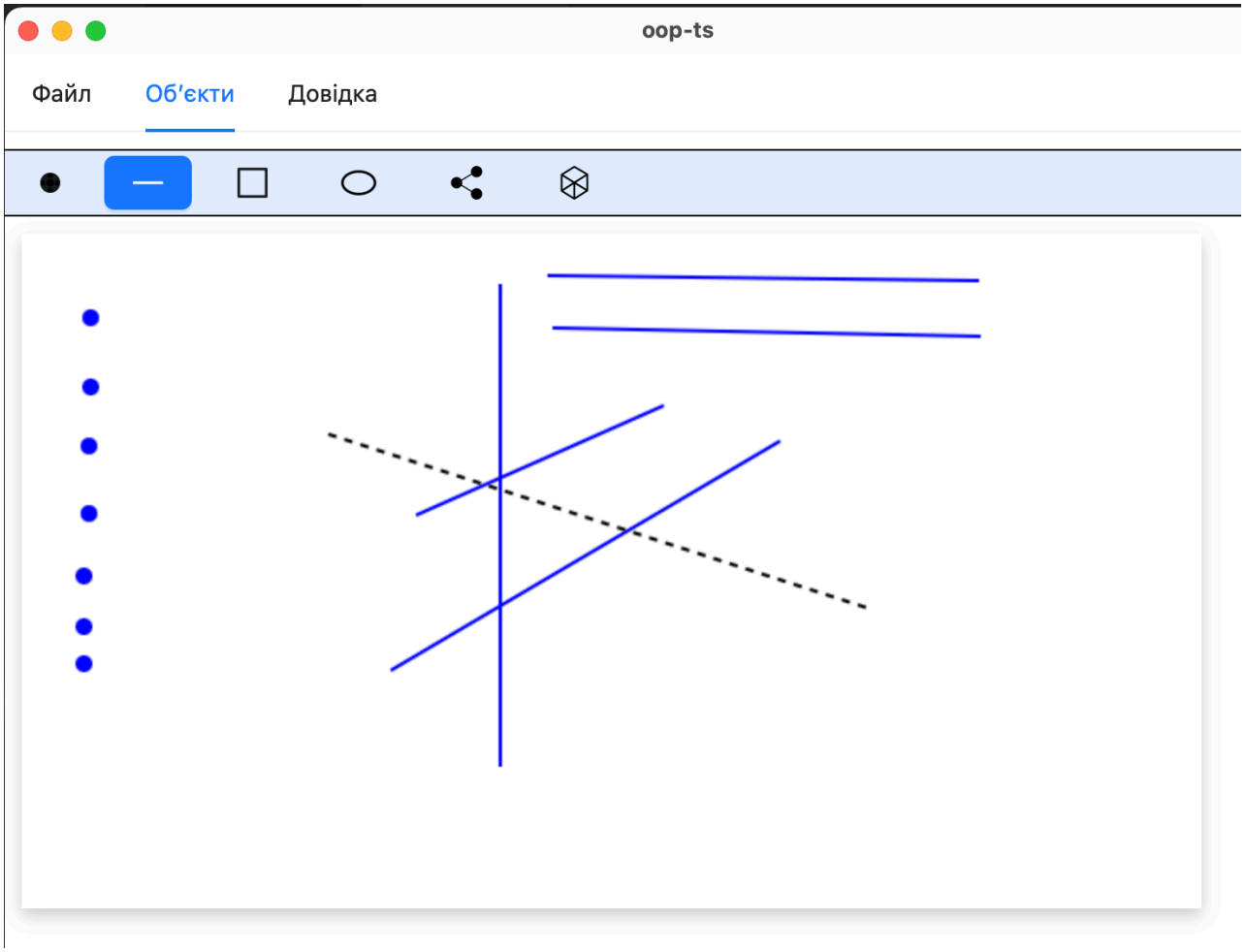
```
export const CubeIcon = (props: React.SVGProps<SVGSVGElement>) => (
  <svg
    fill="#000000"
    height="800px"
    width="800px"
    version="1.1"
    xmlns="http://www.w3.org/2000/svg"
    viewBox="0 0 226.153 226.153"
    enable-background="new 0 0 226.153 226.153"
    {...props}
  >
    <g>
      <path
d="m202.103,51.666c-0.057-0.031-0.117-0.052-0.174-0.081-0.11-0.074-0.218-0.15-0.333-0.218l-86.542-50.548c-1
.923-1.123-4.31-1.089-6.2,0.089l-84.485,52.648c-0.094,0.058-0.179,0.124-0.269,0.187-0.022,0.012-0.045,0.019-
0.067,0.031-1.906,1.058-3.088,3.066-3.088,5.246v108.122c0,2.072 1.069,3.997
2.827,5.092l85.082,53.012c0.971,0.605 2.072,0.908 3.173,0.908 1.114,0 2.227-0.31
3.206-0.928l87.18-55.11c1.739-1.1 2.794-3.014
2.794-5.072v-108.122c-2.84217e-14-2.187-1.189-4.2-3.104-5.256zm-101.264,60.464l-67.894,43.976v-86.279l67.8
94,42.303zm17.189,10.572l69.794,42.44-69.794,44.12v-86.56zm5.385-10.769l69.794-44.12v86.56l-69.794-42.44z
m63.503-55.242l.741,.433-75.653,47.824-73.713-45.929 .824-.514 73.012-45.498
74.789,43.684zm-148.782,110.352l67.894-43.977v86.279l-67.894-42.302z" />
    </g>
  </svg>
);

export const ShareIcon = (props: React.SVGProps<SVGSVGElement>) => (
  <svg
    fill="#000000"
    height="800px"
    width="800px"
    version="1.1"
    id="Capa_1"
    xmlns="http://www.w3.org/2000/svg"
    viewBox="0 0 59 59"
    {...props}
  >
    <path
      d="M47,39c-2.671,0-5.182,1.04-7.071,2.929c-0.524,0.524-0.975,1.1-1.365,1.709l-17.28-10.489

C21.741,32.005,22,30.761,22,29.456c0-1.305-0.259-2.549-0.715-3.693l17.284-10.409C40.345,18.142,43.456,20,47,20
  c5.514,0,10-4.486,10-10S52.514,0,47,0S37,4.486,37,10c0,1.256,0.243,2.454,0.667,3.562L20.361,23.985

c-1.788-2.724-4.866-4.529-8.361-4.529c-5.514,0-10,4.486-10,10s4.486,10,10,10c3.495,0,6.572-1.805,8.36-4.529L
37.664,45.43

C37.234,46.556,37,47.759,37,49c0,2.671,1.04,5.183,2.929,7.071C41.818,57.96,44.329,59,47,59s5.182-1.04,7.07
1-2.929
  C55.96,54.183,57,51.671,57,49s-1.04-5.183-2.929-7.071C52.182,40.04,49.671,39,47,39z"
    />
  </svg>
);
```

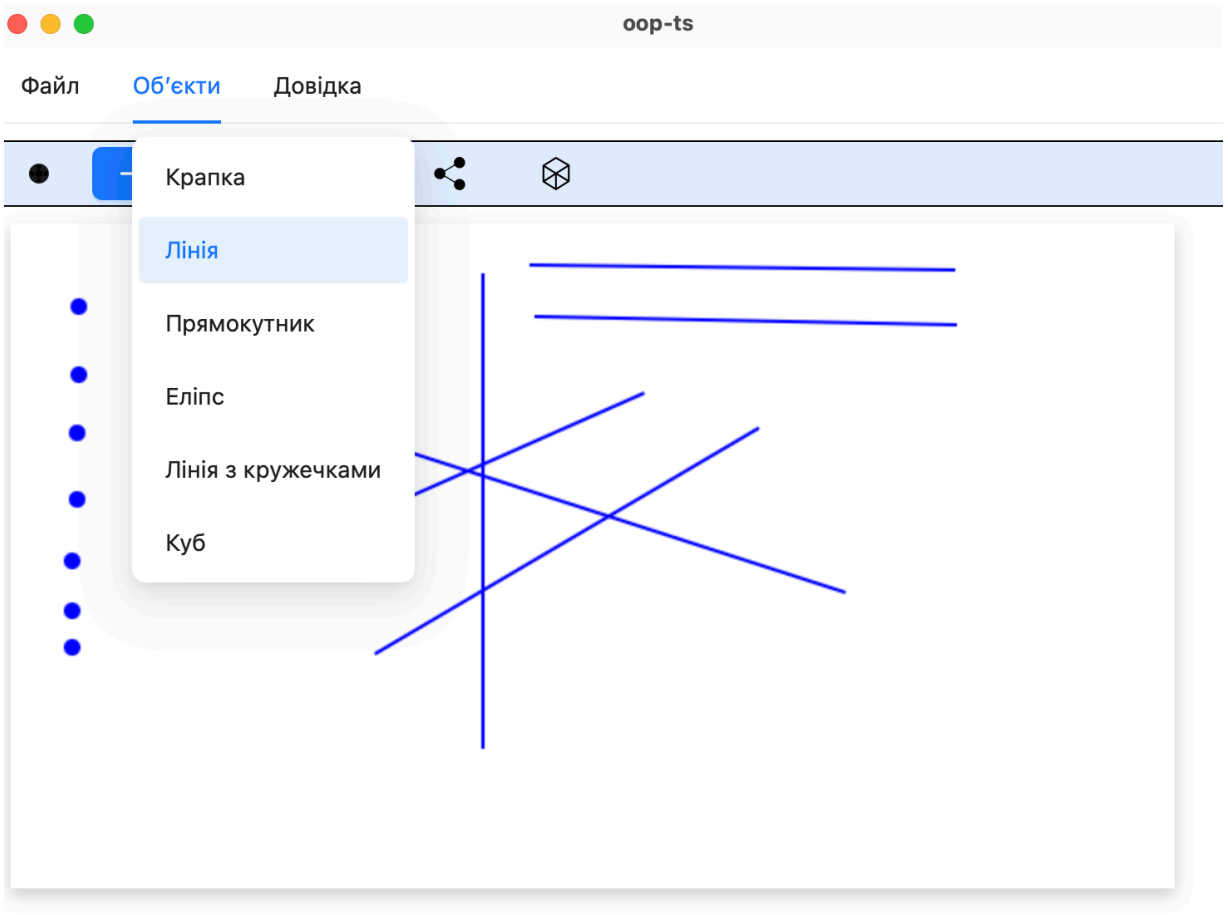
Скріншоти виконання програми:

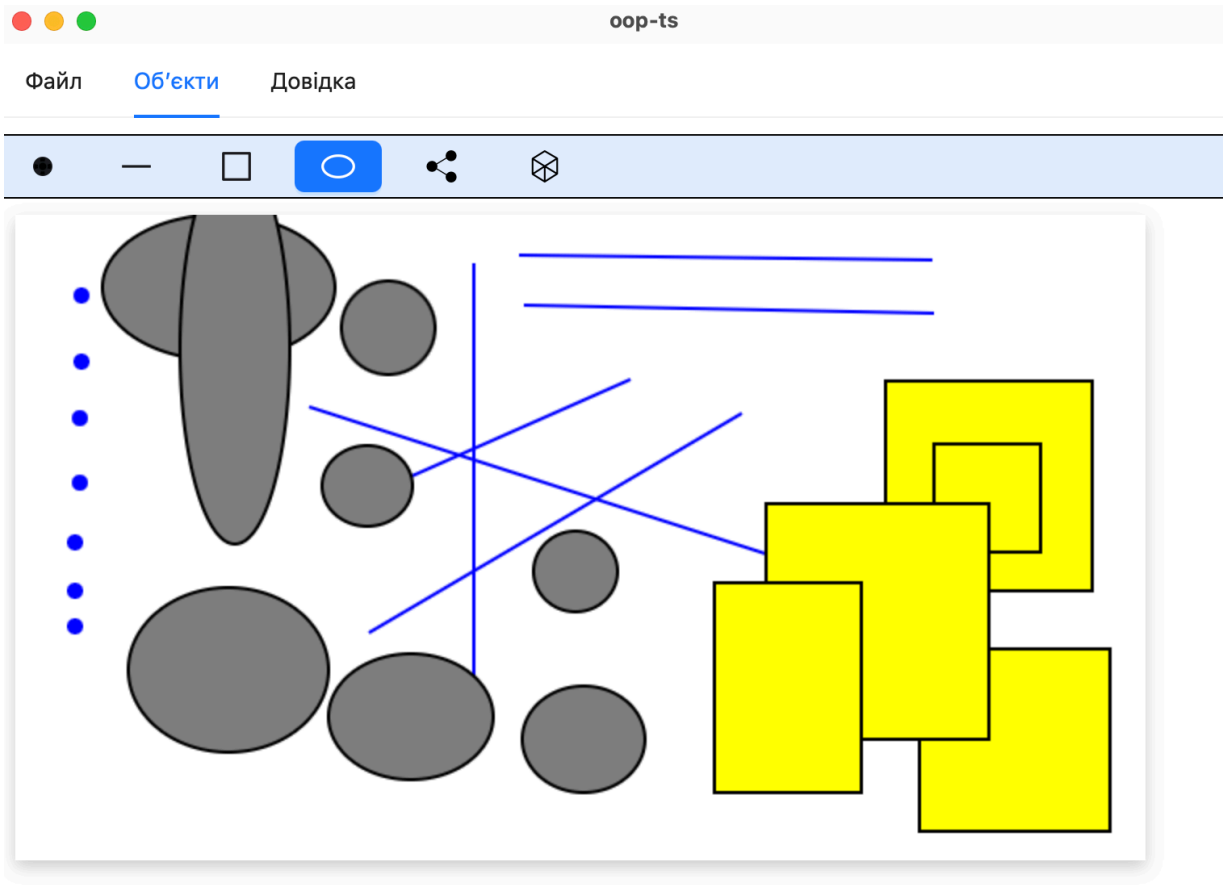


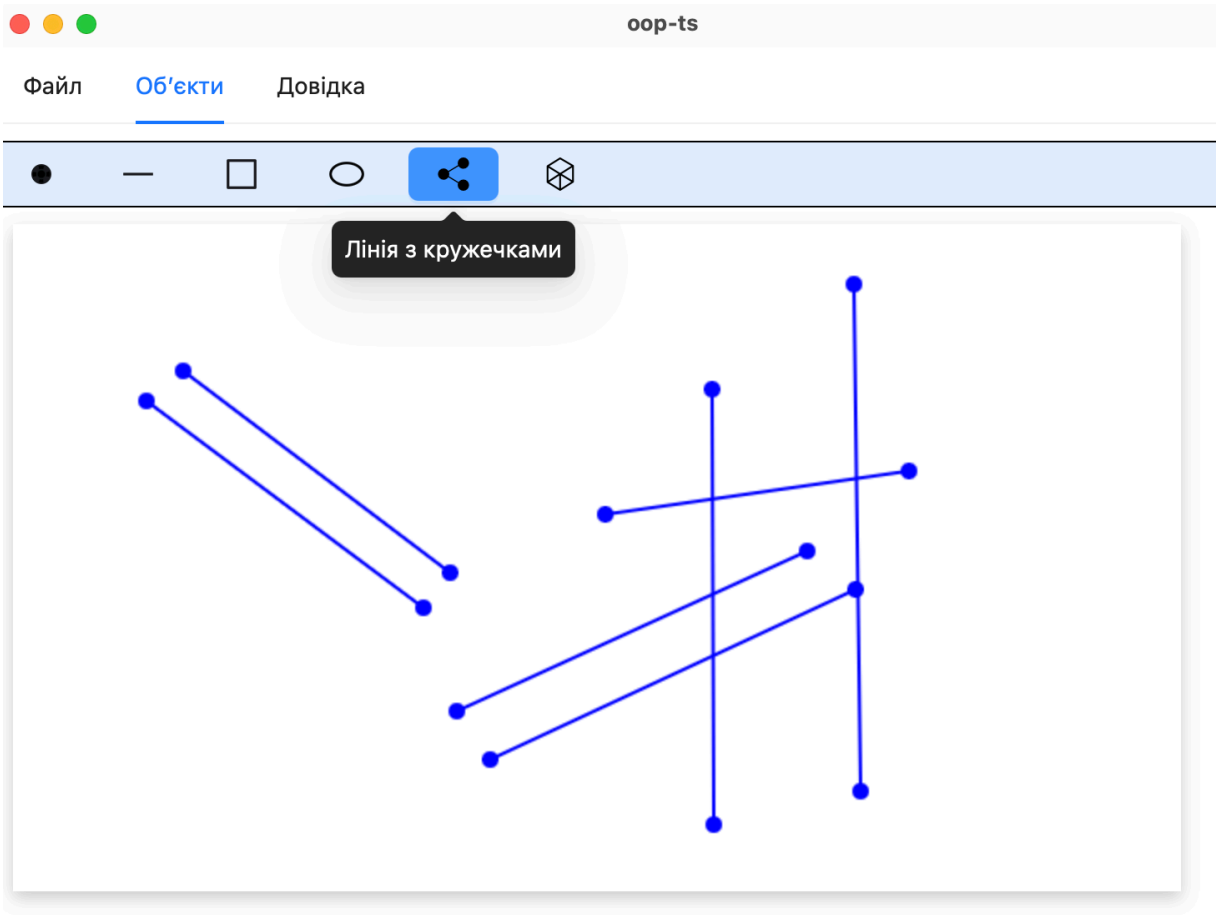


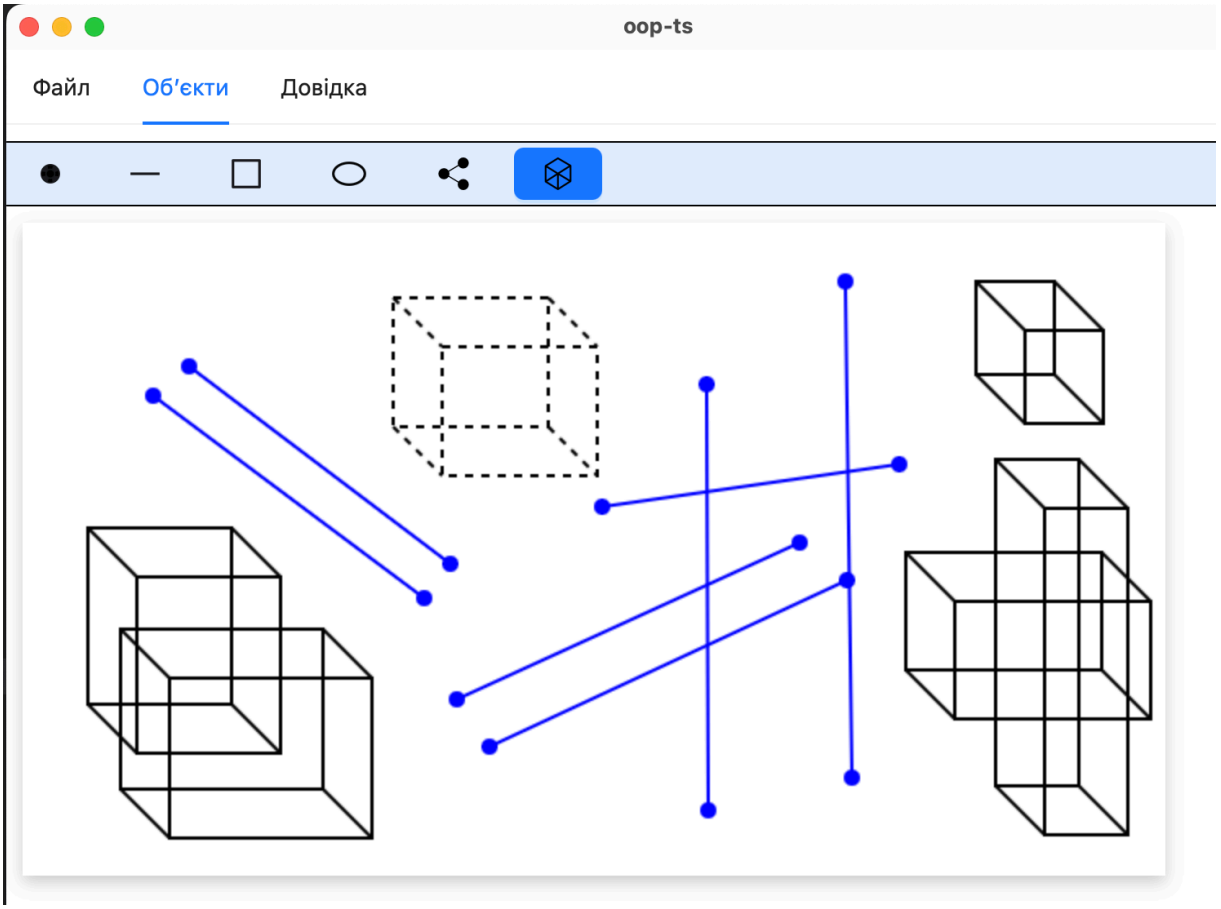

oop-ts
Файл
Об'єкти
Довідка
Крапка
Лінія
Прямокутник
Еліпс
Лінія з кружечками
Куб


oop-ts
Файл
Об'єкти
Довідка


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


oop-ts
Файл
Об'єкти
Довідка


oop-ts
Файл
Об'єкти
Довідка
Лінія з кружечками


oop-ts
Файл
Об'єкти
Довідка

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. Вдосконалила вже наявний функціонал з минулої лабораторної роботи змінивши реалізацію гумового сліду для усіх типів фігур та додавши два нових класи для малювання лінії з кружечками на кінцях і куба.

Так як у Typescript не існує класичної реалізації множинного успадкування, виконати завдання можна було трьома шляхами:

1) Композицією
2) Міксін-функціями
3) Інтерфейсним наслідуванням

Так як у постановці завдання було сказано не використовувати композицію, я спробувала скористатись двома іншими. Відповідно малювання лінії з кружечками на кінцях відбувається за рахунок одиничного спадкування й імплементації додаткового інтерфейсу для малювання кружечків, а малювання куба - міксін функції, в яку передається один із класів для розширення додатковим методом.

Множинне наслідування може бути потужним інструментом, але часто воно ускладнює структуру програми, створює труднощі з підтримкою коду та приводить до непередбачуваних проблем. Більш безпечним способом стане написання додаткових методів через інтерфейси та використання їх у потрібних місцях (навіть повторно).