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» на тему
“Розробка багатовіконного інтерфейсу користувача для графічного редактора об’єктів”

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. Номер варіанту завдання дорівнює номеру зі списку студентів у журналі - 18 варіант

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. Оголошення класів для усіх типів об'єктів робити у окремих заголовочних файлах *.h, а визначення функцій членів – у окремих файлах *.cpp. Таким чином, програмний код для усіх наявних типів об'єктів розподіляється по множині окремих модулів.
8. Ієрархія класів та побудова модулів повинні бути зручними для можливостей додавання нових типів об'єктів без переписування коду вже існуючих модулів.
9. У звіті повинна бути схема успадкування класів – діаграма класів.
11. Б**онуси-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.

При виконанні бонусів 1 та 2 забороняється робити для цього нові залежності модуля **my_table** від інших .cpp файлів. Тоді як надіслати повідомлення (наприклад, про виділення користувачем якогось рядка таблиці) від вікна таблиці клієнту цього вікна (наприклад, коду головного файлу .cpp)? Підказки можна знайти у матеріалі лекції стосовно технології **Callback,** а також патернів Observer, Listener.

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.

1. **Вихідний текст головного файлу (Lab5.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";

interface Lab5Props {
  shapes: Shape[];
  setShapes: (shape: any) => void;
}

export const Lab5 = ({ shapes, setShapes }: Lab5Props) => {
  const canvasRef = useRef<HTMLCanvasElement | null>(null);
  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: Shape[]) => [...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";
import { WebviewWindow } from "@tauri-apps/api/window";
import { Ellipse, Dot, Cube, Line, LineWithCircles, Rectangle } from "@/app/modules/MyEditor";

const openTable = () => {
  new WebviewWindow("table", {
    url: "/?window=table",
    title: "table",
    width: 800,
    height: 600,
    resizable: true,
  });
};

const closeTable = async () => {
  const targetWindow = WebviewWindow.getByLabel("table");
  if (targetWindow) {
    await targetWindow.close();
  }
};

type MenuItem = Required<MenuProps>["items"][number];
export const items: MenuItem[] = [
  {
    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: "Таблиця",
    key: "table",
    children: [
      {
        label: "Відкрити",
        key: "open_table",
        onClick: () => openTable(),
      },
      { label: "Закрити", key: "close_table", onClick: () => closeTable() },
    ],
  },
];

export const getDrawnObject = (shape: any) => {
  if (shape.radiusX && !shape.radius) return "Еліпс";
  if (shape.width && !shape.depth) return "Прямокутник";
  if (shape.radius) return "Лінія з кружечками";
  if (shape.depth) return "Куб";
  if (shape.startX && !shape.radius) return "Лінія";
  return "Крапка";
};

export const formatToClass = (shapes: any[]) => {
  return shapes.map((shape) => {
    const objectName = getDrawnObject(shape);
    if (objectName === "Еліпс") {
      return new Ellipse(
        shape.x,
        shape.y,
        shape.radiusX,
```

```
        shape.radiusY,
        shape.color
      );
    } else if (objectName === "Крапка") {
      return new Dot(shape.x, shape.y, shape.color);
    } else if (objectName === "Куб") {
      return new Cube(shape.x, shape.y, shape.width, shape.height, shape.color);
    } else if (objectName === "Лінія") {
      return new Line(
        shape.startX,
        shape.startY,
        shape.endX,
        shape.endY,
        shape.color
      );
    } else if (objectName === "Лінія з кружечками") {
      return new LineWithCircles(
        shape.startX,
        shape.startY,
        shape.endX,
        shape.endY,
        shape.color
      );
    } else if (objectName === "Прямокутник") {
      return new Rectangle(
        shape.x,
        shape.y,
        shape.width,
        shape.height,
        shape.color
      );
    }
  });
};
```

3. Модуль Toolbar.tsx

```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. Модуль класів MyEditor.ts

```
export abstract class Shape {
  protected fillColor: string;
  protected strokeColor: string;

  constructor(
    fillColor: string = "transparent",
    strokeColor: string = "black"
  ) {
    this.fillColor = fillColor;
    this.strokeColor = strokeColor;
  }

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

  highlight(strokeColor: string): void {
    this.strokeColor = strokeColor;
  }
}

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

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

  draw(ctx: CanvasRenderingContext2D): void {
    ctx.fillStyle = this.fillColor;
    ctx.fillRect(this.x, this.y, this.width, this.height);
    ctx.setLineDash([]);
    ctx.strokeStyle = this.strokeColor;
    ctx.strokeRect(this.x, this.y, this.width, this.height);
  }

  drawPreview(ctx: CanvasRenderingContext2D): void {
    ctx.setLineDash([5, 5]);
    ctx.strokeStyle = this.strokeColor;
    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,
    fillColor: string = "grey",
    strokeColor: string = "black"
  ) {
    super(fillColor, strokeColor);
    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.fillColor;
    ctx.fill();
    ctx.setLineDash([]);
    ctx.strokeStyle = this.strokeColor;
    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 = this.strokeColor;
    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,
    fillColor: string = "black",
    strokeColor: string = "black"
  ) {
    super(startX, startY, endX, endY, fillColor, strokeColor);
    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.strokeColor
    );
    this.drawCircle(ctx, this.endX, this.endY, this.radius, this.strokeColor);
  }

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

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
    ) {
      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,
    fillColor: string = "transparent",
    strokeColor: string = "black"
  ) {
    super(x, y, width, height, fillColor, strokeColor);
    this.depth = depth;
  }

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

    ctx.strokeStyle = this.strokeColor;
```

```
    ctx.strokeRect(backX, backY, this.width, this.height);

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

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

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

5. Модуль класу таблиці TableManager.ts

```
import { listen } from "@tauri-apps/api/event";
import { WebviewWindow } from "@tauri-apps/api/window";
import { getDrawnObject } from "../components/lab5/constants";
import { writeFile } from "@tauri-apps/api/fs";
import { desktopDir } from "@tauri-apps/api/path";

export interface ShapesForTable {
  key: number;
  object: string;
  x: number | null;
  y: number | null;
  startX: number | null;
  startY: number | null;
  endX: number | null;
  endY: number | null;
}

export class TableManager {
  private static instance: TableManager;
  private receivedShapes: ShapesForTable[] = [];
  private originalShapes: any[] = [];
  private subscribers: Set<() => void> = new Set();

  private constructor() {}

  public static getInstance(): TableManager {
    if (!TableManager.instance) {
      TableManager.instance = new TableManager();
    }
    return TableManager.instance;
  }

  public subscribe(callback: () => void): void {
    this.subscribers.add(callback);
  }

  public unsubscribe(callback: () => void): void {
    this.subscribers.delete(callback);
  }

  private notifySubscribers(): void {
    this.subscribers.forEach((callback) => callback());
  }

  public getReceivedShapes(): ShapesForTable[] {
    return this.receivedShapes;
  }

  public setReceivedShapes(shapes: ShapesForTable[]): void {
    this.receivedShapes = shapes;
    this.notifySubscribers();
  }

  public getOriginalShapes(): any[] {
    return this.originalShapes;
  }

  public setOriginalShapes(shapes: any[]): void {
    this.originalShapes = shapes;
    this.notifySubscribers();
  }

  public deleteShape(key: number): void {
    this.receivedShapes = this.receivedShapes.filter(
      (shape) => shape.key !== key
    );
```

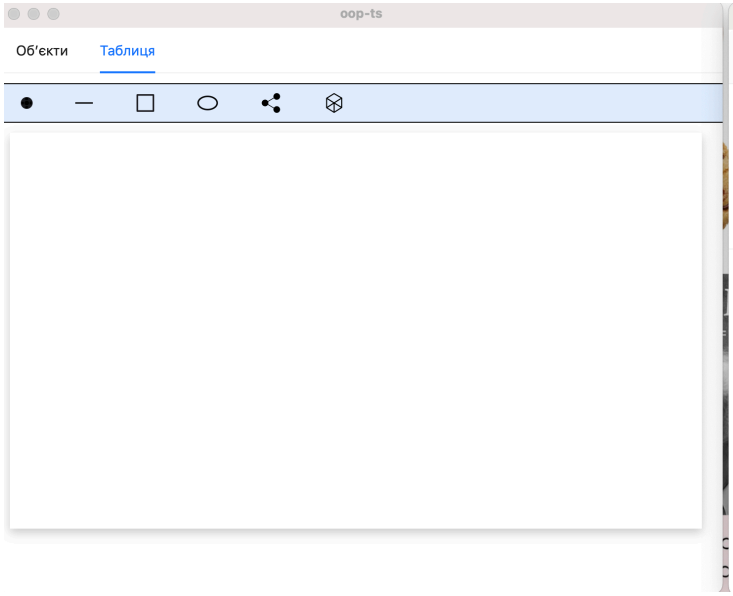
```
    this.originalShapes = this.originalShapes.filter(
      (_, index) => index !== key
    );

    this.notifySubscribers();

    const mainWindow = WebviewWindow.getByLabel("main");
    if (mainWindow) {
      mainWindow.emit("update-shapes", this.originalShapes);
    }
  }

  public highlightShapes(selectedRowKeys: React.Key[]): void {
    const mainWindow = WebviewWindow.getByLabel("main");
    if (mainWindow) {
      mainWindow.emit("highlight-shapes", selectedRowKeys);
    }
  }

  public async writeShapesToFile(shapes: ShapesForTable[]): Promise<void> {
    try {
      const desktopPath = await desktopDir();
      const filePath = `${desktopPath}/shapes.txt`;

      const fileContent = shapes
        .map(
          (shape) =>
            `${shape.object} з координатами: ${JSON.stringify({
              x: shape.x || null,
              y: shape.y || null,
              startX: shape.startX || null,
              startY: shape.startY || null,
              endX: shape.endX || null,
              endY: shape.endY || null,
            })}`
        )
        .join("\n");

      await writeFile({
        path: filePath,
        contents: fileContent,
      });

      console.log(`File written successfully to ${filePath}`);
    } catch (err) {
      console.error("Error writing file:", err);
    }
  }

  public listenForShapes(): void {
    listen<any[]>("send-shapes", async (event) => {
      const shapes = event.payload.map((shape, index) => ({
        key: index,
        object: getDrawnObject(shape),
        x: shape.x || null,
        y: shape.y || null,
        startX: shape.startX || null,
        startY: shape.startY || null,
        endX: shape.endX || null,
        endY: shape.endY || null,
      }));

      this.setReceivedShapes(shapes);
      this.setOriginalShapes(event.payload);

      await this.writeShapesToFile(shapes);
    });
```

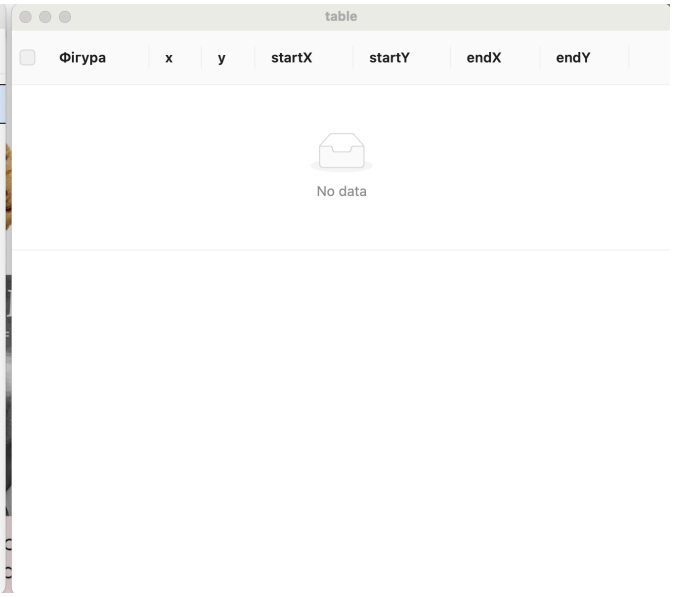
```
  }
}
```

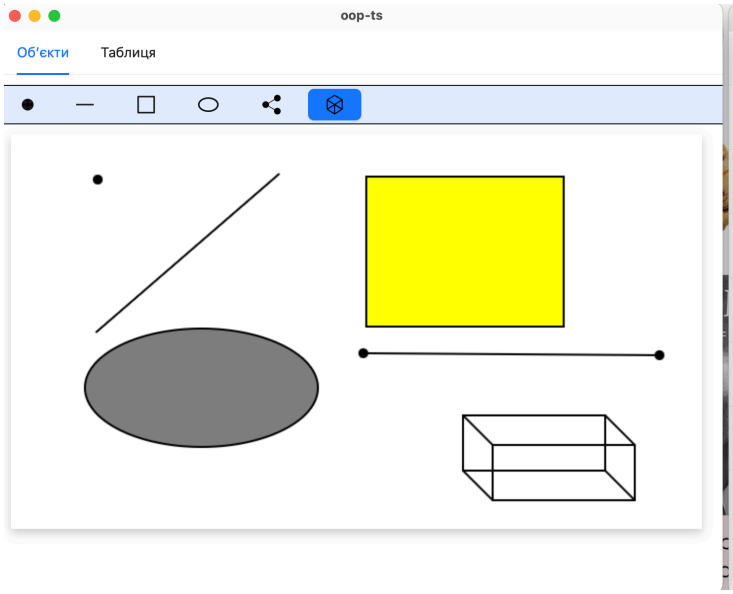
6. Модуль компоненту вікна таблиці Table.tsx

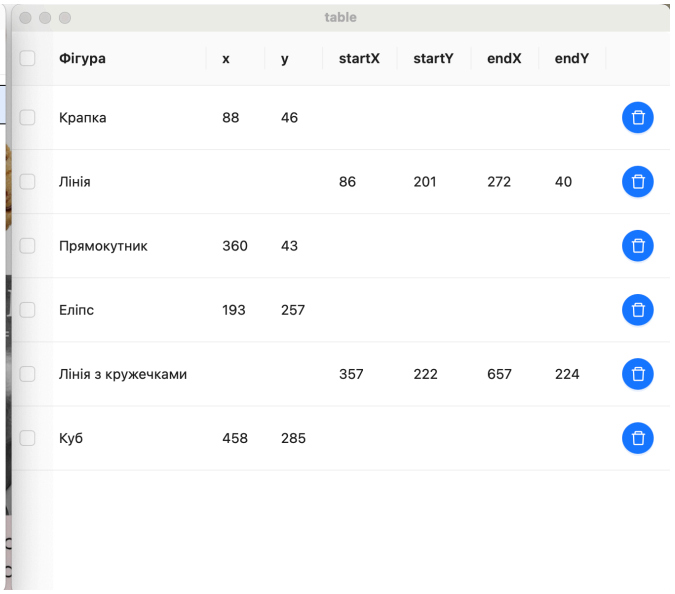
```
import { Table as AntTable, Button } from "antd";
import { useEffect, useState } from "react";
import { DeleteOutlined } from "@ant-design/icons";
import { ShapesForTable, TableManager } from "@/app/modules/TableManager";

export const Table = () => {
  const [tableManager] = useState(() => TableManager.getInstance());
  const [receivedShapes, setReceivedShapes] = useState<ShapesForTable[]>([]);

  useEffect(() => {
    const updateShapes = () => {
      setReceivedShapes(tableManager.getReceivedShapes());
    };

    tableManager.subscribe(updateShapes);
    tableManager.listenForShapes();

    return () => {
      tableManager.unsubscribe(updateShapes);
    };
  }, [tableManager]);

  const columns = [
    { title: "Фігура", dataIndex: "object", key: "object" },
    { title: "x", dataIndex: "x", key: "x" },
    { title: "y", dataIndex: "y", key: "y" },
    { title: "startX", dataIndex: "startX", key: "startX" },
    { title: "startY", dataIndex: "startY", key: "startY" },
    { title: "endX", dataIndex: "endX", key: "endX" },
    { title: "endY", dataIndex: "endY", key: "endY" },
    {
      title: "",
      dataIndex: "delete",
      key: "delete",
      render: (_: any, record: ShapesForTable) => (
        <Button
          type="primary"
          shape="circle"
          icon={<DeleteOutlined />}
          onClick={() => tableManager.deleteShape(record.key)}
        />
      ),
    },
  ];

  const handleSelect = (selectedRowKeys: React.Key[]) => {
    tableManager.highlightShapes(selectedRowKeys);
  };

  return (
    <AntTable
      rowSelection={{
        type: "checkbox",
        onChange: handleSelect,
      }}
      dataSource={receivedShapes}
      columns={columns}
      pagination={false}
    />
  );
};
```

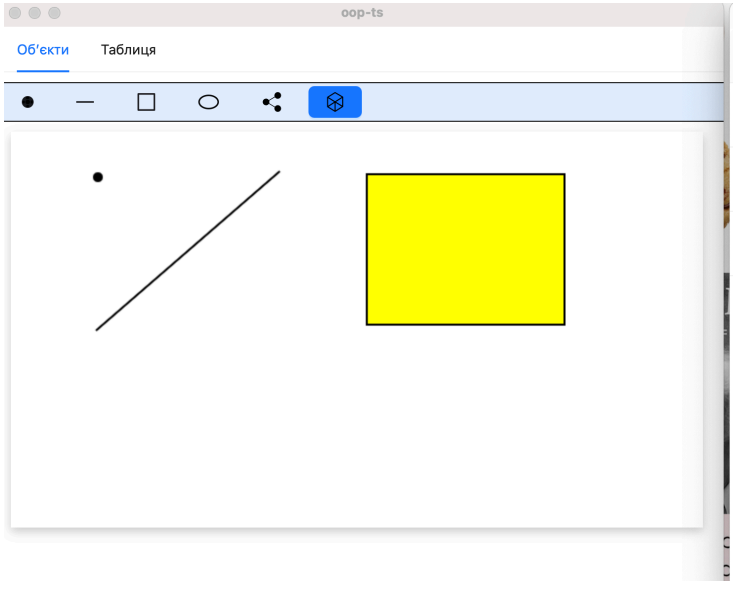
Скріншоти виконання програми:

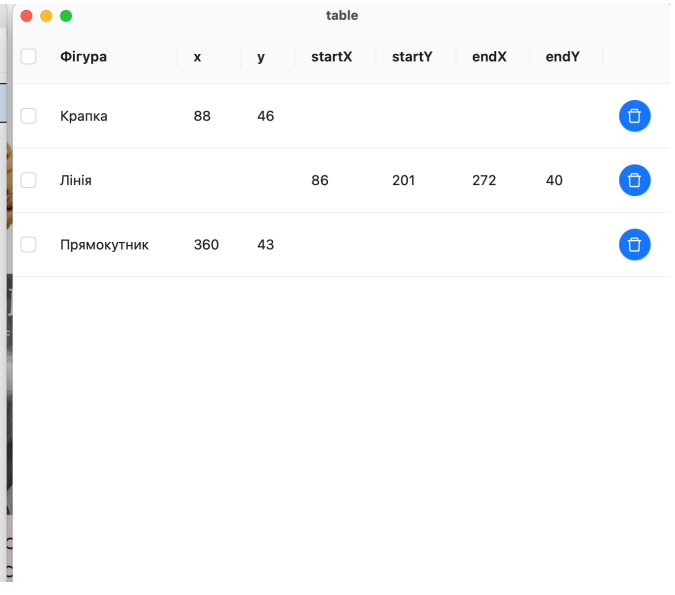


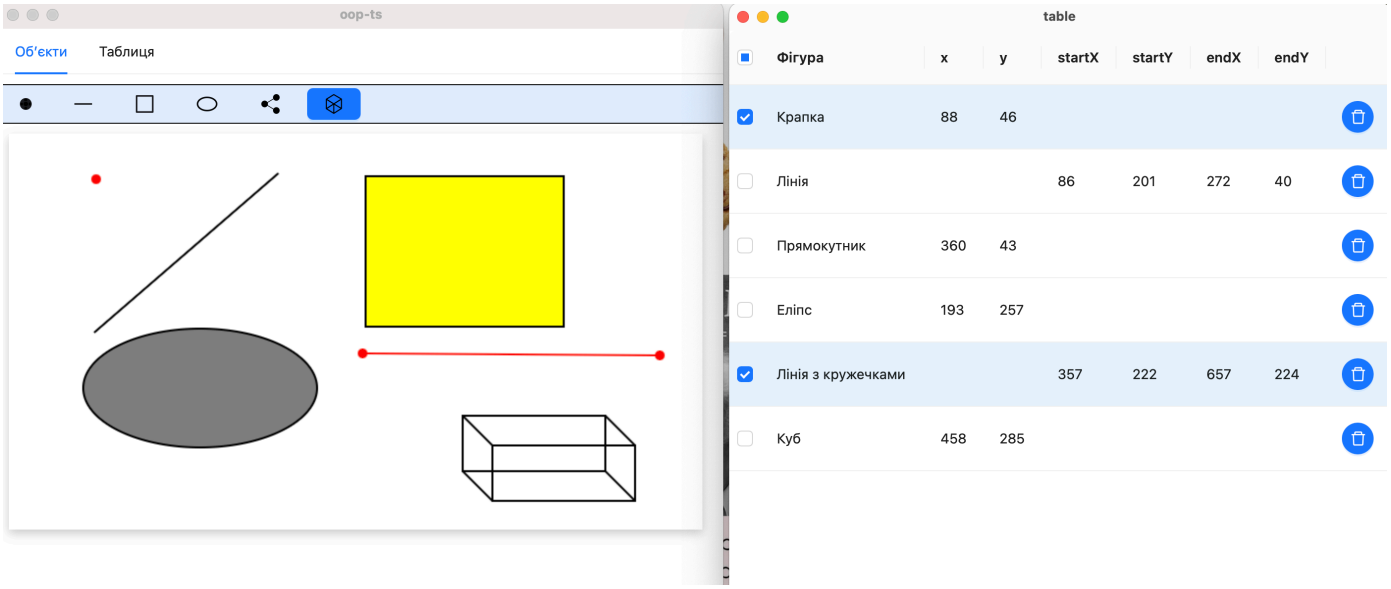


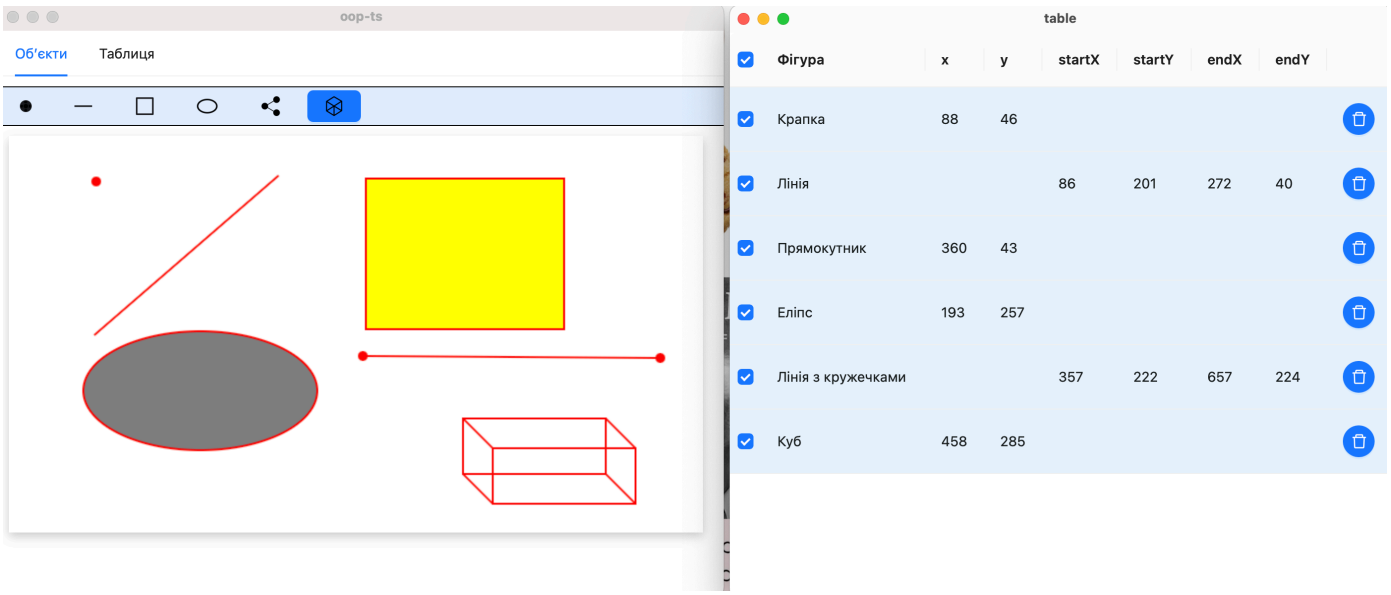


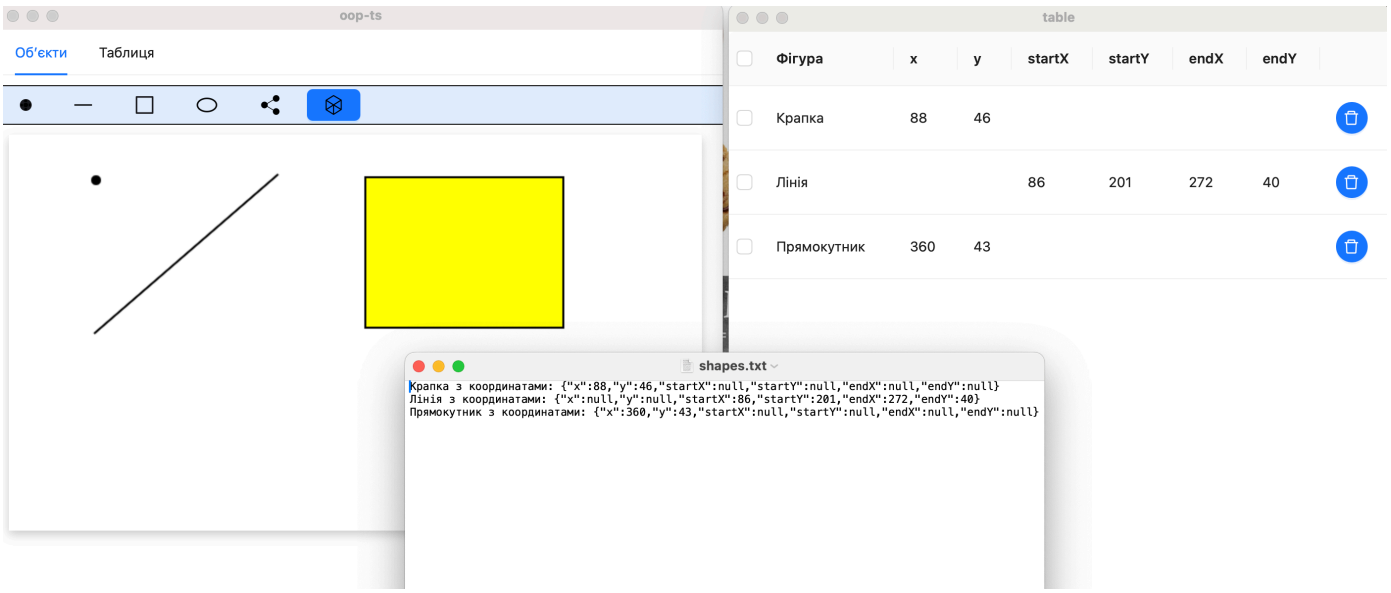







oop-ts
Об'єкти
Таблиця
table
Фігура x y startX startY endX endY
Крапка 88 46
Лінія 86 201 272 40
Прямокутник 360 43
Еліпс 193 257
Лінія з кружечками 357 222 657 224
Куб 458 285


oop-ts
Об'єкти
Таблиця
table
Фігура x y startX startY endX endY
Крапка 88 46
Лінія 86 201 272 40
Прямокутник 360 43
Еліпс 193 257
Лінія з кружечками 357 222 657 224
Куб 458 285


oop-ts
Об'єкти
Таблиця
table
Фігура x y startX startY endX endY
Крапка 88 46
Лінія 86 201 272 40
Прямокутник 360 43
shapes.txt
Крапка з координатами: {"x":88,"y":46,"startX":null,"startY":null,"endX":null,"endY":null}
Лінія з координатами: {"x":null,"y":null,"startX":86,"startY":201,"endX":272,"endY":40}
Прямокутник з координатами: {"x":360,"y":43,"startX":null,"startY":null,"endX":null,"endY":null}

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 додавши в меню пункти для відкриття і закриття нового вікна таблиці для взаємодії з головним.

Це стало можливим з використанням функцій апі фреймворку Tauri, який я використовую для виконання лабораторних робіт. Усі методи для взаємодії з отриманими даними структуровані в класі патерну сінглтон. Цей модуль є абсолютно незалежним і доступним звідусіль.

Загалом, використовувати глобальний об’єкт такого формату є доцільним досить рідко, так як зміна його станів впливатиме на усі місця, де він був застосований. Ба більше наслідування чи створення декількох екземплярів є неможливим. Відповідно застосувати цю технологію для малювання фігур у класах модулю MyEditor є неможливим.