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»

Виконав:

Студент групи ІП-04

Пащенко Дмитро Олексійович
номер у списку групи: 19

Перевірив:

Порєв Віктор Миколайович

Київ 2021

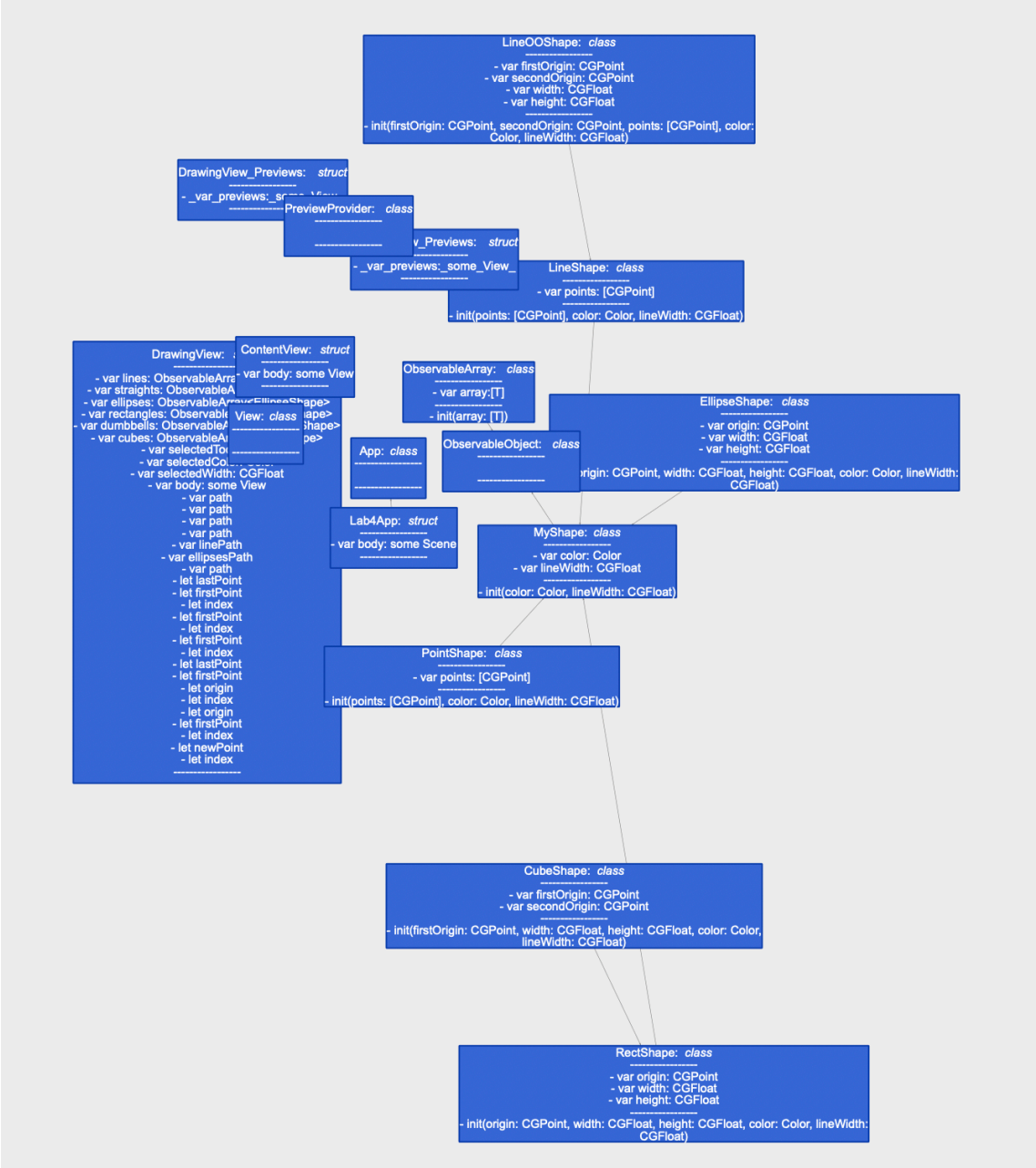
Мета:

Мета роботи – отримати вміння та навички проектування класів, виконавши модернізацію коду графічного редактора в об’єктно-орієнтованому стилі для забезпечення зручного додавання нових типів об’єктів.

Завдання:

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab4.
2. Написати вихідний текст програми згідно варіанту завдання.
3. Скомпілювати вихідний текст і отримати виконуваний файл програми.
4. Перевірити роботу програми. Налагодити програму.
5. Проаналізувати та прокоментувати результати та вихідний текст програми.
6. Оформити звіт.

Діаграма класів:



Код програми

Tools.swift

```
import Foundation
import SwiftUI
import Combine

class ObservableArray<T>: ObservableObject {
    @Published var array:[T] = []

    init(array: [T]) {
        self.array = array
    }
}

class MyShape: ObservableObject {
    var color: Color
    var lineWidth: CGFloat

    init(color: Color, lineWidth: CGFloat) {
        self.color = color
        self.lineWidth = lineWidth
    }
}

class PointShape: MyShape {
    var points: [CGPoint]

    init(points: [CGPoint], color: Color, lineWidth: CGFloat) {
        self.points = points
        super.init(color: color, lineWidth: lineWidth)
    }
}

class LineShape: MyShape {
    var points: [CGPoint]

    init(points: [CGPoint], color: Color, lineWidth: CGFloat) {
        self.points = points
        super.init(color: color, lineWidth: lineWidth)
    }
}

class EllipseShape: MyShape {
    var origin: CGPoint
    var width: CGFloat
    var height: CGFloat

    init(origin: CGPoint, width: CGFloat, height: CGFloat, color: Color,
lineWidth: CGFloat) {
        self.origin = origin
        self.width = width
        self.height = height
        super.init(color: color, lineWidth: lineWidth)
    }
}

class RectShape: MyShape {
    var origin: CGPoint
    var width: CGFloat
    var height: CGFloat

    init(origin: CGPoint, width: CGFloat, height: CGFloat, color: Color,
lineWidth: CGFloat) {
        self.origin = origin
```

```
        self.width = width
        self.height = height
        super.init(color: color, lineWidth: lineWidth)
    }
}

class LineOOShape: LineShape {
    var firstOrigin: CGPoint
    var secondOrigin: CGPoint
    var width: CGFloat = 30
    var height: CGFloat = 30

    init(firstOrigin: CGPoint, secondOrigin: CGPoint, points: [CGPoint], color:
Color, lineWidth: CGFloat) {
        self.firstOrigin = firstOrigin
        self.secondOrigin = secondOrigin
        super.init(points: points, color: color, lineWidth: lineWidth)
    }
}

class CubeShape: RectShape {
    var firstOrigin: CGPoint
    var secondOrigin: CGPoint

    init(firstOrigin: CGPoint, width: CGFloat, height: CGFloat, color: Color,
lineWidth: CGFloat) {
        self.firstOrigin = firstOrigin
        self.secondOrigin = CGPoint(x: firstOrigin.x + 40, y: firstOrigin.y -
40)
        super.init(origin: firstOrigin, width: width, height: height, color:
color, lineWidth: lineWidth)
    }

}
```

DrawingView.swift

```
import SwiftUI

struct DrawingView: View {
    @ObservedObject var lines: ObservableArray<PointShape> =
ObservableArray(array: [])
    @ObservedObject var straights: ObservableArray<LineShape> =
ObservableArray(array: [])
    @ObservedObject var ellipses: ObservableArray<EllipseShape> =
ObservableArray(array: [])
    @ObservedObject var rectangles: ObservableArray<RectShape> =
ObservableArray(array: [])
    @ObservedObject var dumbbells: ObservableArray<LineOOShape> =
ObservableArray(array: [])
    @ObservedObject var cubes: ObservableArray<CubeShape> =
ObservableArray(array: [])

    @State private var selectedTool: String = ""
    @State private var selectedColor: Color = .red
    @State private var selectedWidth: CGFloat = 5

    var body: some View {
        VStack {
            VStack {
                HStack {
                    Text("Выберите инструмент")
                    Spacer()
                    Picker(selection: $selectedTool, label: Text("Выберите
инструмент")) {
                        Text("Кривая").tag("Line")
```

```
                    Text("Прямая").tag("Straight")
                    Text("Эллипс").tag("Ellipse")
                    Text("Квадрат").tag("Rectangle")
                    Text("Гантеля").tag("Dumbbell")
                    Text("Куб").tag("Cube")
                }
            }

            HStack {
                Text("Выберите цвет")
                Spacer()
                ColorPicker("Выберите цвет", selection: $selectedColor)
                    .labelsHidden()
            }

            HStack {
                Text("Выберите толщину")
                Spacer()
                Slider(value: $selectedWidth, in: 1...10) {
                }
                .frame(maxWidth: 190)
            }

            HStack {
                Button("Очистить", action: {
                    lines.array = []
                    straights.array = []
                    ellipses.array = []
                    rectangles.array = []
                    dumbbells.array = []
                    cubes.array = []
                })
                    .foregroundColor(.red)
                Spacer()
            }
        }
        .padding()

        Divider()

        Canvas { context, size in
            for line in lines.array {
                var path = Path()
                path.addLines(line.points)

                context.stroke(path,
                               with: .color(line.color),
                               lineWidth: line.lineWidth)
            }

            for straight in straights.array {
                var path = Path()
                path.addLines(straight.points)

                context.stroke(path,
                               with: .color(straight.color),
                               lineWidth: straight.lineWidth)
            }

            for ellipse in ellipses.array {
                var path = Path()
                path.addEllipse(in: CGRect(origin: ellipse.origin,
                                           size: CGSize(width:
ellipse.width,
                                                        height:
ellipse.height)))
```

```
                    context.stroke(
                        path,
                        with: .color(ellipse.color),
                        lineWidth: ellipse.lineWidth)
                }

                for rectangle in rectangles.array {
                    var path = Path()
                    path.addRect(CGRect(origin: rectangle.origin,
                                        size: CGSize(width: rectangle.width,
                                                     height: rectangle.height)))

                    context.stroke(
                        path,
                        with: .color(rectangle.color),
                        lineWidth: rectangle.lineWidth)
                }

                for dumbbell in dumbbells.array {
                    var linePath = Path()
                    var ellipsesPath = Path()
                    linePath.addLines(dumbbell.points)
                    ellipsesPath.addEllipse(in: CGRect(origin:
dumbbell.firstOrigin,
                                                       size: CGSize(width:
dumbbell.width,
                                                                    height:
dumbbell.height)))
                    ellipsesPath.addEllipse(in: CGRect(origin:
dumbbell.secondOrigin,
                                                       size: CGSize(width:
dumbbell.width,
                                                                    height:
dumbbell.height)))

                    context.stroke(linePath,
                                   with: .color(dumbbell.color),
                                   lineWidth: dumbbell.lineWidth)
                    context.fill(ellipsesPath,
                                 with: .color(dumbbell.color))
//                    context.stroke(ellipsesPath,
//                                   with: .color(dumbbell.color),
//                                   lineWidth: dumbbell.lineWidth)

                }

                for cube in cubes.array {
                    var path = Path()
                    path.addRect(CGRect(origin: cube.origin,
                                        size: CGSize(width: cube.width,
                                                     height: cube.height)))

                    path.addRect(CGRect(origin: cube.secondOrigin,
                                        size: CGSize(width: cube.width,
                                                     height: cube.height)))

                    path.addLines([cube.firstOrigin, cube.secondOrigin])

                    path.addLines([CGPoint(x: cube.firstOrigin.x + cube.width,
                                           y: cube.firstOrigin.y),
                                   CGPoint(x: cube.secondOrigin.x + cube.width,
                                           y: cube.secondOrigin.y)])

                    path.addLines([CGPoint(x: cube.firstOrigin.x,
                                           y: cube.firstOrigin.y + cube.height),
```

```
                                    CGPoint(x: cube.secondOrigin.x,
                                            y: cube.secondOrigin.y +
cube.height)])

                    path.addLines([CGPoint(x: cube.firstOrigin.x + cube.width,
                                            y: cube.firstOrigin.y + cube.height),
                                    CGPoint(x: cube.secondOrigin.x + cube.width,
                                            y: cube.secondOrigin.y +
cube.height)])

                    context.stroke(path,
                                   with: .color(cube.color),
                                   lineWidth: cube.lineWidth)
                }

            }
            .gesture(DragGesture(minimumDistance: 0,
coordinateSpace: .local).onChanged({ value in
                switch selectedTool {
                case "Straight":
                    let lastPoint = value.location
                    if value.translation.width + value.translation.height == 0 {
                        let firstPoint = value.location
                        straights.array.append(LineShape(points: [firstPoint],
                                                         color: selectedColor,
                                                         lineWidth: selectedWidth))
                    } else {
                        let index = straights.array.count - 1

                        if straights.array[index].points.count == 2 {
                            straights.objectWillChange.send()
                            straights.array[index].points[1] = lastPoint
                        } else {
                            straights.array[index].points.append(lastPoint)
                        }
                    }

                case "Ellipse":
                    if value.translation.width + value.translation.height == 0 {
                        let firstPoint = value.startLocation
                            ellipses.array.append(EllipseShape(origin:
firstPoint,
                                                               width: 0,
                                                               height: 0,
                                                               color:
selectedColor,
                                                               lineWidth:
selectedWidth))
                    } else {
                        let index = ellipses.array.count - 1
                        ellipses.objectWillChange.send()
                        ellipses.array[index].width = value.translation.width
                        ellipses.array[index].height = value.translation.height
                    }

                case "Rectangle":
                    if value.translation.width + value.translation.height == 0 {
                        let firstPoint = value.startLocation
                            rectangles.array.append(RectShape(origin:
firstPoint,
                                                      width: 0,
                                                      height: 0,
                                                      color: selectedColor,
                                                      lineWidth: selectedWidth))
                    } else {
                        let index = rectangles.array.count - 1
```

```
                        rectangles.objectWillChange.send()
                        rectangles.array[index].width = value.translation.width
                        rectangles.array[index].height =
value.translation.height
                    }

                case "Dumbbell":
                    let lastPoint = value.location
                    if value.translation.width + value.translation.height == 0 {
                        let firstPoint = value.location
                        let origin = CGPoint(x: value.location.x - 15,
                                             y: value.location.y - 15)
                        dumbbells.array.append(LineOOShape(firstOrigin: origin,
                                                           secondOrigin: origin,
                                                           points: [firstPoint],
                                                           color: selectedColor,
                                                           lineWidth:
selectedWidth))
                    } else {
                        let index = dumbbells.array.count - 1

                        if dumbbells.array[index].points.count == 2 {
                            dumbbells.objectWillChange.send()
                            dumbbells.array[index].points[1] = lastPoint
                            let origin = CGPoint(x: value.location.x - 15,
                                                 y: value.location.y - 15)
                            dumbbells.array[index].secondOrigin = origin
                        } else {
                            dumbbells.array[index].points.append(lastPoint)
                        }
                    }

                case "Cube":
                    if value.translation.width + value.translation.height == 0 {
                        let firstPoint = value.startLocation
                        cubes.array.append(CubeShape(firstOrigin: firstPoint,
                                                     width: 0,
                                                     height: 0,
                                                     color: selectedColor,
                                                     lineWidth: selectedWidth))
                    } else {
                        let index = cubes.array.count - 1
                        cubes.objectWillChange.send()
                        cubes.array[index].width = value.translation.width
                        cubes.array[index].height = value.translation.height

                    }


                default:
                    let newPoint = value.location
                    if value.translation.width + value.translation.height == 0 {
                        lines.array.append(PointShape(points: [newPoint],
                                                      color: selectedColor,
                                                      lineWidth: selectedWidth))
                    } else {
                        let index = lines.array.count - 1
                        lines.objectWillChange.send()
                        lines.array[index].points.append(newPoint)
                    }
                }
            }))
            Divider()
        }
    }
}
```

```
struct DrawingView_Previews: PreviewProvider {
    static var previews: some View {
        DrawingView()
    }
}
```

Висновок

В даній лабораторній роботі ми отримали вміння та навички проектування класів, виконавши модернізацію коду графічного редактора в об’єктно-орієнтованому стилі для забезпечення зручного додавання нових типів об’єктів.

Наприклад, додали класи гантелі (пряма з кругами на краях) та куба (два з’єднанні між собою квадрати).