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»

Виконав:

Студент групи ІП-04

Пащенко Дмитро Олексійович
номер у списку групи: 19

Перевірив:

Порєв Віктор Миколайович

Київ 2021

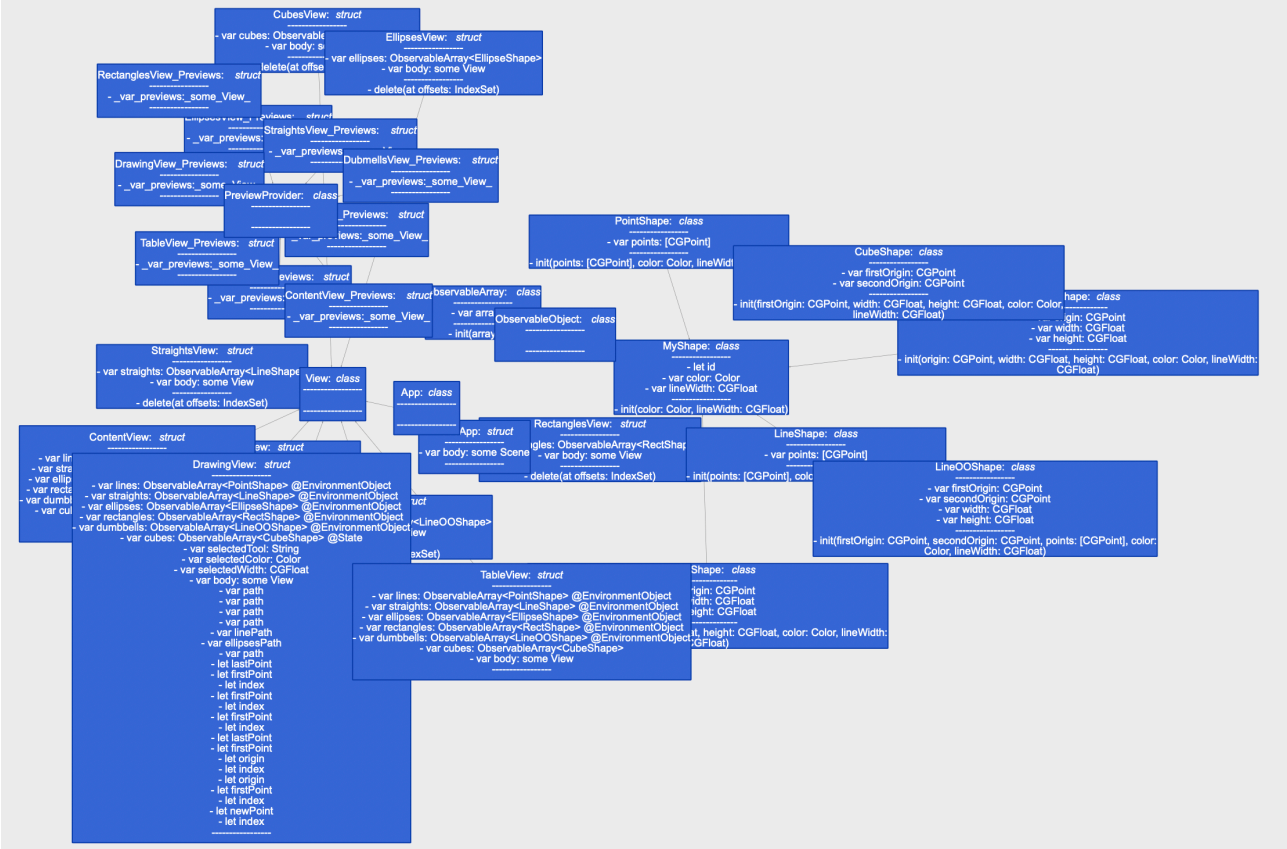
Мета:

Мета роботи – отримати вміння та навички програмувати багатовіконний інтерфейс програми на C++ в об’єктно-орієнтованому стилі.

Завдання:

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab5.
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:



Висновок

В даній лабораторній роботі ми отримали вміння та навички програмувати багатовіконний інтерфейс програми.

Зроблено два вікна — Полотно та Таблиця елементів.

Є можливість переглядати властивості елементів — їхні координати, колір, товщину лінії.

Можливо видалити елемент із таблиці — тоді елемент автоматично буде видалено із полотна.

Реалізований патерн програмування Singleton (один об’єкт на всю програму).

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, Identifiable {
    let id = UUID()
    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

```
import SwiftUI

struct DrawingView: View {
    @EnvironmentObject var lines: ObservableArray<PointShape>
    @EnvironmentObject var straights: ObservableArray<LineShape>
    @EnvironmentObject var ellipses: ObservableArray<EllipseShape>
    @EnvironmentObject var rectangles: ObservableArray<RectShape>
    @EnvironmentObject var dumbbells: ObservableArray<LineOOShape>
    @EnvironmentObject var cubes: ObservableArray<CubeShape>

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

    var body: some View {
        VStack {
            VStack (alignment: .leading) {
                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)
                }

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

                    }


                case 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()
    }
}
```

TableView

```
import SwiftUI

struct TableView: View {
    @EnvironmentObject var lines: ObservableArray<PointShape>
    @EnvironmentObject var straights: ObservableArray<LineShape>
    @EnvironmentObject var ellipses: ObservableArray<EllipseShape>
    @EnvironmentObject var rectangles: ObservableArray<RectShape>
    @EnvironmentObject var dumbbells: ObservableArray<LineOOShape>
    @EnvironmentObject var cubes: ObservableArray<CubeShape>

    var body: some View {
        NavigationView {
            List {
                NavigationLink(destination: LinesView()) {
                    HStack {
                        Image(systemName: "alternatingcurrent")
                            .scaledToFit()
                            .frame(width: 25)
                        VStack (alignment: .leading) {
                            Text("Кривые")
                            Text("Количество: \(lines.array.count)")
                                .font(.subheadline)
                                .foregroundColor(.secondary)
                        }
                    }
                }

                NavigationLink(destination: StraightsView()) {
                    HStack {
                        Image(systemName: "line.diagonal")
                            .scaledToFit()
                            .frame(width: 25)
                        VStack (alignment: .leading) {
                            Text("Прямые")
                            Text("Количество: \(straights.array.count)")
                                .font(.subheadline)
                                .foregroundColor(.secondary)
                        }
                    }
                }

                NavigationLink(destination: EllipsesView()) {
                    HStack {
                        Image(systemName: "circle")
                            .scaledToFit()
                            .frame(width: 25)
                        VStack (alignment: .leading) {
                            Text("Эллипсы")
                            Text("Количество: \(ellipses.array.count)")
                                .font(.subheadline)
                                .foregroundColor(.secondary)
                        }
                    }
                }

                NavigationLink(destination: RectanglesView()) {
                    HStack {
                        Image(systemName: "rectangle")
                            .scaledToFit()
                            .frame(width: 25)
                        VStack (alignment: .leading) {
                            Text("Квадраты")
                            Text("Количество: \(rectangles.array.count)")
                                .font(.subheadline)
```

```
                                .foregroundColor(.secondary)
                        }
                    }
                }

                NavigationLink(destination: DumbellsView()) {
                    HStack {
                        Image(systemName: "eyeglasses")
                            .scaledToFit()
                            .frame(width: 25)
                        VStack (alignment: .leading) {
                            Text("Гантели")
                            Text("Количество: \(dumbbells.array.count)")
                                .font(.subheadline)
                                .foregroundColor(.secondary)
                        }
                    }
                }

                NavigationLink(destination: CubesView()) {
                    HStack {
                        Image(systemName: "cube")
                            .scaledToFit()
                            .frame(width: 25)
                        VStack (alignment: .leading) {
                            Text("Кубы")
                            Text("Количество: \(cubes.array.count)")
                                .font(.subheadline)
                                .foregroundColor(.secondary)
                        }
                    }
                }
            }
            .navigationTitle("Таблица")
        }
    }
}

struct TableView_Previews: PreviewProvider {
    static var previews: some View {
        TableView()
    }
}
```

LinesView

```
import SwiftUI

struct LinesView: View {
    @EnvironmentObject var lines: ObservableArray<PointShape>

    var body: some View {
        List {
            ForEach(lines.array, id: \.id) { item in
                VStack (alignment: .leading) {
                    Text("(\(String(format: "%.2f", item.points[0].x)), \
(String(format: "%.2f", item.points[0].y))) → (\(String(format: "%.2f",
item.points[item.points.count - 1].x)), \(String(format: "%.2f",
item.points[item.points.count - 1].y)))")
                    HStack {
                        Text("Цвет:")
                            .font(.subheadline)
                            .foregroundColor(.secondary)
                        Rectangle()
                            .frame(width: 15, height: 15)
                            .foregroundColor(item.color)
```

```
                    }
                    Text("Толщина: \(String(format: "%.2f", item.lineWidth))")
                        .font(.subheadline)
                        .foregroundColor(.secondary)
                }
            }
            .onDelete(perform: delete)
        }
        .navigationTitle("Кривые")

    }

    func delete(at offsets: IndexSet) {
        lines.array.remove(atOffsets: offsets)
    }
}

struct LinesView_Previews: PreviewProvider {
    static var previews: some View {
        LinesView()
    }
}
```

StraightsView

```
import SwiftUI

struct StraightsView: View {
    @EnvironmentObject var straights: ObservableArray<LineShape>

    var body: some View {
        List {
            ForEach(straights.array, id: \.id) { item in
                VStack (alignment: .leading) {
                    Text("(\(String(format: "%.2f", item.points[0].x)), \
(String(format: "%.2f", item.points[0].y))) → (\(String(format: "%.2f",
item.points[item.points.count - 1].x)), \(String(format: "%.2f",
item.points[item.points.count - 1].y)))")
                    HStack {
                        Text("Цвет:")
                            .font(.subheadline)
                            .foregroundColor(.secondary)
                        Rectangle()
                            .frame(width: 15, height: 15)
                            .foregroundColor(item.color)
                    }
                    Text("Толщина: \(String(format: "%.2f", item.lineWidth))")
                        .font(.subheadline)
                        .foregroundColor(.secondary)
                }
            }
            .onDelete(perform: delete)
        }
        .navigationTitle("Прямые")
    }

    func delete(at offsets: IndexSet) {
        straights.array.remove(atOffsets: offsets)
    }
}

struct StraightsView_Previews: PreviewProvider {
    static var previews: some View {
        StraightsView()
    }
}
```

EllipsesView

```
import SwiftUI

struct EllipsesView: View {
    @EnvironmentObject var ellipses: ObservableArray<EllipseShape>

    var body: some View {
        List {
            ForEach(ellipses.array, id: \.id) { item in
                VStack (alignment: .leading) {
                    Text("(\(String(format: "%.2f", item.origin.x)), \
(String(format: "%.2f", item.origin.y))) → (\(String(format: "%.2f",
item.origin.x + item.width)), \(String(format: "%.2f", item.origin.y +
item.height))")
                    HStack {
                        Text("Цвет:")
                            .font(.subheadline)
                            .foregroundColor(.secondary)
                        Rectangle()
                            .frame(width: 15, height: 15)
                            .foregroundColor(item.color)
                    }
                    Text("Толщина: \(String(format: "%.2f", item.lineWidth))")
                        .font(.subheadline)
                        .foregroundColor(.secondary)
                }
            }
            .onDelete(perform: delete)
        }
        .navigationTitle("Эллипсы")
    }

    func delete(at offsets: IndexSet) {
        ellipses.array.remove(atOffsets: offsets)
    }
}

struct EllipsesView_Previews: PreviewProvider {
    static var previews: some View {
        EllipsesView()
    }
}
```

RectanglesView

```
import SwiftUI

struct RectanglesView: View {
    @EnvironmentObject var rectangles: ObservableArray<RectShape>

    var body: some View {
        List {
            ForEach(rectangles.array, id: \.id) { item in
                VStack (alignment: .leading) {
                    Text("(\(String(format: "%.2f", item.origin.x)), \
(String(format: "%.2f", item.origin.y))) → (\(String(format: "%.2f",
item.origin.x + item.width)), \(String(format: "%.2f", item.origin.y +
item.height))")
                    HStack {
                        Text("Цвет:")
                            .font(.subheadline)
                            .foregroundColor(.secondary)
                        Rectangle()
                            .frame(width: 15, height: 15)
```

```
                            .foregroundColor(item.color)
                    }
                    Text("Толщина: \(String(format: "%.2f", item.lineWidth))")
                        .font(.subheadline)
                        .foregroundColor(.secondary)
                }
            }
            .onDelete(perform: delete)
        }
        .navigationTitle("Квадраты")
    }

    func delete(at offsets: IndexSet) {
        rectangles.array.remove(atOffsets: offsets)
    }
}

struct RectanglesView_Previews: PreviewProvider {
    static var previews: some View {
        RectanglesView()
    }
}
```

DumbellsView

```
import SwiftUI

struct DumbellsView: View {
    @EnvironmentObject var dumbbells: ObservableArray<LineOOShape>

    var body: some View {
        List {
            ForEach(dumbbells.array, id: \.id) { item in
                VStack (alignment: .leading) {
                    Text("(\(String(format: "%.2f", item.firstOrigin.x)), \
(String(format: "%.2f", item.firstOrigin.y))) → (\(String(format: "%.2f",
item.secondOrigin.x)), \(String(format: "%.2f", item.secondOrigin.y)))")
                    HStack {
                        Text("Цвет:")
                            .font(.subheadline)
                            .foregroundColor(.secondary)
                        Rectangle()
                            .frame(width: 15, height: 15)
                            .foregroundColor(item.color)
                    }
                    Text("Толщина: \(String(format: "%.2f", item.lineWidth))")
                        .font(.subheadline)
                        .foregroundColor(.secondary)
                }
            }
            .onDelete(perform: delete)
        }
        .navigationTitle("Гантели")
    }

    func delete(at offsets: IndexSet) {
        dumbbells.array.remove(atOffsets: offsets)
    }
}

struct DubmellsView_Previews: PreviewProvider {
    static var previews: some View {
        DumbellsView()
    }
}
```

CubesView

```
import SwiftUI

struct CubesView: View {
    @EnvironmentObject var cubes: ObservableArray<CubeShape>

    var body: some View {
        List {
            ForEach(cubes.array, id: \.id) { item in
                VStack (alignment: .leading) {
                    Text("(\(String(format: "%.2f", item.firstOrigin.x)), \
(String(format: "%.2f", item.firstOrigin.y))) → (\(String(format: "%.2f",
item.secondOrigin.x)), \(String(format: "%.2f", item.secondOrigin.y)))")
                    HStack {
                        Text("Цвет:")
                            .font(.subheadline)
                            .foregroundColor(.secondary)
                        Rectangle()
                            .frame(width: 15, height: 15)
                            .foregroundColor(item.color)
                    }
                    Text("Толщина: \(String(format: "%.2f", item.lineWidth))")
                        .font(.subheadline)
                        .foregroundColor(.secondary)
                }
            }
            .onDelete(perform: delete)
        }
        .navigationTitle("Кубы")
    }

    func delete(at offsets: IndexSet) {
        cubes.array.remove(atOffsets: offsets)
    }
}

struct CubesView_Previews: PreviewProvider {
    static var previews: some View {
        CubesView()
    }
}
```