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 ІМ-21
Сірик Максим Олександрович
номер у списку групи: 22

Перевірив:
Порєв Віктор Миколайович

Київ 2023

Зміст

1 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.

2 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.

2.1 Варіант:

Варіанти завдань та основні вимоги
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.

$$\text{Номер варіанту} = 22 \tag{1}$$

Студенти з непарним номером (1, 3, 5, . . .) програмують глобальний статичний об'єкт класу MyEditor.
Студенти з парним номером (2, 4, 6, . . .) програмують динамічний обєкт класу MyEditor, забезпечивши коректне його створення та знищення.
3. Усі кольори та стилі (за винятком "гумового"сліду) геометричних форм – як у попередній лабораторній роботі No3. "Гумовий"слід при вводі усіх фігур малювати пунктирною лінією.
4. Окрім чотирьох типів фігур, які були у попередніх лаб. No2 та 3, запрограмувати ще введення та відображення двох нових фігур – лінія з кружечками та каркас куба. Кольори ліній та заповнення цих нових фігур студент визначає на свій розсуд.
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 шости типів форм зробити кнопки Toolbar з підказками (tooltips)
6. У звіті повинна бути схема успадкування класів – діаграма класів.
Потрібно побудувати діаграму класів засобами Visual Studio C++.

3 Текст програми:

3.1 Module: com.github.erotourtes.drawing.editor

Лістинг 1: DmProcessor.kt

```
package com.github.erotourtes.drawing.editor
```

```
import com.github.erotourtes.utils.Dimension

fun interface DmProcessor {
    fun process(dm: Dimension): Dimension
}
```

Лістинг 2: Editor.kt

```
package com.github.erotourtes.drawing.editor

import com.github.erotourtes.drawing.shape.*
import javafx.scene.paint.Color
import com.github.erotourtes.utils.*
import javafx.scene.canvas.GraphicsContext
import javafx.scene.input.MouseEvent

abstract class Editor(protected val shapes: ShapesList, protected val gc: GraphicsContext)
    protected val dm = Dimension()
    protected open val processor: DmProcessor = DmProcessor { it }
    protected abstract val shape: Shape

    open fun listenToEvents() {
        val c = gc.canvas
        c.setOnMousePressed(::onMousePressed)
        c.setOnMouseDragged(::onMouseDragged)
        c.setOnMouseReleased(::onMouseReleased)
    }

    protected open fun onMousePressed(e: MouseEvent) {
        redraw()
        dm.setStart(e.x, e.y)
    }

    protected open fun onMouseDragged(e: MouseEvent) {
        redraw()

        dm.setEnd(e.x, e.y)
        previewLine()
    }

    protected open fun onMouseReleased(e: MouseEvent) {
        if (e.isDragDetect) return // returns if mouse was not dragged
        shape.setDm(processor.process(dm))
        shapes.add(shape.copy())
        redraw()
    }

    private fun drawAll() {
        for (shape in shapes) shape.drawWithProperties()
    }

    private fun clear() = gc.clearRect(0.0, 0.0, gc.canvas.width, gc.canvas.height)

    protected fun redraw() {
```

```
            clear()
            drawAll()
        }

        protected open fun previewLine() {
            gc.drawOnce {
                setPreviewProperties()
                shape.setDm(processor.process(dm))
                shape.draw()
            }
        }

        protected fun setPreviewProperties() {
            gc.setLineDashes(5.0)
            gc.stroke = Color.BLACK
            gc.fill = Color.TRANSPARENT
        }
}
```

Лістинг 3: Editors.kt

```
package com.github.erotourtes.drawing.editor

import com.github.erotourtes.drawing.shape.*
import com.github.erotourtes.utils.Dimension
import javafx.scene.canvas.GraphicsContext
import javafx.scene.input.MouseEvent

class PointEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Point(gc)

    override fun onMouseDragged(e: MouseEvent) {}

    override fun onMousePressed(e: MouseEvent) {
        dm.setEnd(e.x, e.y)
        super.onMousePressed(e)
        shape.setDm(dm)
    }

    override fun onMouseReleased(e: MouseEvent) {
        shapes.add(shape.copy())
        redraw()
    }

    override fun previewLine() {}
}

class LineEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Line(gc)
}

class RectEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Rect(gc)
    override val processor = DmProcessor { Dimension.toCorner(it) }
}
```

```
class EllipseEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Ellipse(gc)
    override val processor: DmProcessor = DmProcessor { Dimension.toCorner(it) }
}

class EmptyEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = object : Shape(gc) {
        override fun draw() {}
    }

    override fun onMouseDragged(e: MouseEvent) {}
    override fun onMousePressed(e: MouseEvent) {}
    override fun onMouseReleased(e: MouseEvent) {}
    override fun previewLine() {}
}

class DumbbellEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Dumbbell(gc)
}

class CubeEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Cube(gc)
}
class CubeExEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = CubeEx(gc)
}
```

Лістинг 4: ShapesList.kt

```
package com.github.erotourtes.drawing.editor

import com.github.erotourtes.drawing.shape.Shape

class ShapesList(n: Int) : Iterable<Shape> {
    private val shapeArr = Array<Shape?>(n) { null }
    private var shapeIndex = 0

    val size: Int
        get() = shapeIndex

    fun add(sh: Shape) {
        if (shapeIndex == shapeArr.size) throw IllegalArgumentException("History is overflo
        shapeArr[shapeIndex++] = sh
    }

    override fun iterator(): Iterator<Shape> = ShapeIterator()

    inner class ShapeIterator : Iterator<Shape> {
        private var curIndex = 0
        override fun hasNext(): Boolean = curIndex < size
        override fun next(): Shape = if (hasNext()) shapeArr[curIndex++]!! else throw Illeg
    }
    override fun toString(): String = "ShapesList(index=$shapeIndex)"
}
```

3.2 Module: 1.0

Лістинг 5: MANIFEST.MF

```
Manifest-Version: 1.0
Main-Class: com.github.erotourtes.app.MyApp
```

3.3 Module: com.github.erotourtes.view

Лістинг 6: MainController.kt

```
package com.github.erotourtes.view

import com.github.erotourtes.drawing.EditorHandler
import com.github.erotourtes.drawing.editor.*
import com.github.erotourtes.utils.ShapeInfo
import de.jensd.fx.glyphs.fontawesome.FontAwesomeIcon
import javafx.scene.canvas.Canvas
import tornadofx.*

class MainController : Controller() {
    val shapesInfo = listOf(
        ShapeInfo("Dot", "Dot", PointEditor::class.java, FontAwesomeIcon.DOT_CIRCLE_ALT),
        ShapeInfo("Line", "Line", LineEditor::class.java, FontAwesomeIcon.MINUS),
        ShapeInfo("Rectangle", "Rectangle", RectEditor::class.java, FontAwesomeIcon.SQUARE
        ShapeInfo("Ellipse", "Ellipse", EllipseEditor::class.java, FontAwesomeIcon.CIRCLE_A
        ShapeInfo("Dumbbell", "Dumbbell", DumbbellEditor::class.java, FontAwesomeIcon.MARS
        ShapeInfo("Cube", "Cube", CubeEditor::class.java, FontAwesomeIcon.CUBE),
        ShapeInfo("Cube", "CubeEx", CubeExEditor::class.java, FontAwesomeIcon.CUBES),
    )

    val editorHandler: EditorHandler

    init {
        val scope = super.scope as ScopeInfo
        editorHandler = EditorHandler(scope.canvas)
        editorHandler.useEditor(EmptyEditor::class.java)
    }

    data class ScopeInfo(val canvas: Canvas) : Scope()
}
```

Лістинг 7: MainView.kt

```
package com.github.erotourtes.view

import com.github.erotourtes.drawing.CanvasPane
import javafx.scene.canvas.Canvas
import tornadofx.*

class MainView : View("Lab3") {
    private val canvas = Canvas()
    private val ctrl: MainController by inject(MainController.ScopeInfo(canvas))

    override val root = borderpane {
        top = MenuBar.create(ctrl.editorHandler, ctrl.shapesInfo)
```

```
        center = borderpane {
            top = ToolBar.create(ctrl.editorHandler, ctrl.shapesInfo)
            center = CanvasPane(canvas)
        }
    }
}
```

Лістинг 8: MenuBar.kt

```
package com.github.erotourtes.view

import com.github.erotourtes.drawing.EditorHandler
import com.github.erotourtes.utils.PopupView
import com.github.erotourtes.utils.ShapeInfo
import com.github.erotourtes.utils.g
import com.github.erotourtes.utils.n
import javafx.scene.control.*
import javafx.scene.control.MenuBar
import javafx.stage.StageStyle
import tornadofx.*

class MenuBar(vararg menu: Menu) : MenuBar() {
    init {
        menu("File") {
            val invoke: MenuItem.() -> Unit = {
                action { find<PopupView>(PopupView.ScopeInfo(text)).openModal(StageStyle.U
            }

            item("New...") { invoke() }
            item("Open...") { invoke() }
            item("Save as...") { invoke() }
            separator()
            item("Print") { invoke() }
            separator()
            item("Exit") { invoke() }
        }

        menus.addAll(menu)

        menu("Help") {
            item(
                """
                    0) Ж = $g
                    1) Статичний масив (Ж mod 3 != 0) обсягом $g + 100 = $n.
                    2) "Гумовий" слід при вводі об'єктів - пунктирна лінія чорного кольору
                    3) Прямокутник:
                        Увід прямокутника:
                            - від центру до одного з кутів для (Ж mod 2 = 1) g % 2 = ${g %
                        Відображення прямокутника:
                            - чорний контур прямокутника без заповнення для (Ж mod 5 = 3 a6
                        Кольори заповнення прямокутника:
                            - сірий для (Ж mod 6 = 5) g % 6 = ${g % 6}
                    4) Еліпс:
                        Ввід еліпсу:
                            - по двом протилежним кутам охоплюючого прямокутника для варіан
```

```
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Відображення␣еліпсу:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣чорний␣контур␣з␣кольоровим␣заповненням␣для␣(Ж␣mod␣5␣=␣3␣або␣4
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Кольори␣заповнення␣еліпсу:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣помаранчевий␣для␣(Ж␣mod␣6␣=␣5)␣g␣%␣6␣=␣${g␣%␣6}
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣5)␣Позначка␣поточного␣типу␣об'єкту,␣що␣вводиться
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣в␣заголовку␣вікна␣для␣(Ж␣mod␣2␣=␣1)␣g␣%␣2␣=␣${g␣%␣2}
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣""".trimIndent()
            )
        }
    }

    companion object {
        fun create(editorHandler: EditorHandler, list: List<ShapeInfo>): MenuBar {
            val group = ToggleGroup()

            editorHandler.listenToChanges { _, _, newValue ->
                group.toggles.forEach {
                    val userData = it.userData as ShapeInfo
                    it.isSelected = userData.kotlinClass == newValue.javaClass
                }
            }

            val objectsUI = list.map {
                RadioMenuItem(it.name).apply {
                    action { editorHandler.useEditor(it.kotlinClass) }
                    toggleGroup = group
                    isSelected = false
                    userData = it
                }
            }

            val menu = Menu("Objects").apply { items.addAll(objectsUI) }

            return com.github.erotourtes.view.MenuBar(menu)
        }
    }
}
```

Лістинг 9: ToolBar.kt

```
package com.github.erotourtes.view

import com.github.erotourtes.drawing.EditorHandler
import com.github.erotourtes.drawing.editor.EmptyEditor
import com.github.erotourtes.styles.ToolbarStyles
import com.github.erotourtes.utils.ShapeInfo
import de.jensd.fx.glyphs.fontawesome.FontAwesomeIconView
import javafx.scene.Node
import javafx.scene.control.ToggleButton
import javafx.scene.control.ToggleGroup
import javafx.scene.layout.HBox
import tornadofx.*

class ToolBar(items: List<Node>) : HBox() {
    init {
```

```
        addClass(ToolbarStyles.toolbar)

        items.forEach { this += it }
    }

    companion object {
        fun create(editorHandler: EditorHandler, list: List<ShapeInfo>): ToolBar {
            val group = ToggleGroup()

            editorHandler.listenToChanges { _, _, newValue ->
                group.toggles.forEach {
                    val userData = it.userData as ShapeInfo
                    it.isSelected = userData.kotlinClass == newValue.javaClass
                }
            }

            val shapesUI = list.map {
                ToggleButton().apply {
                    tooltip(it.tooltip)
                    addClass(ToolbarStyles.iconButton)
                    add(FontAwesomeIconView(it.icon).apply { addClass(ToolbarStyles.icon) }
                    toggleGroup = group
                    isSelected = false
                    userData = it
                    action {
                        editorHandler.useEditor(
                            if (this.isSelected) it.kotlinClass else EmptyEditor::class.jav
                        )
                    }
                }
            }

            return ToolBar(shapesUI)
        }
    }
}
```

3.4 Module: com.github.erotourtes.app

Лістинг 10: MyApp.kt

```
package com.github.erotourtes.app

import com.github.erotourtes.styles.ToolbarStyles
import com.github.erotourtes.view.MainView
import tornadofx.App

class MyApp: App(MainView::class, ToolbarStyles::class)
```

3.5 Module: com.github.erotourtes.drawing

Лістинг 11: CanvasPane.kt

```
package com.github.erotourtes.drawing

import javafx.scene.canvas.Canvas
```

```
import javafx.scene.layout.Pane

class CanvasPane(canvas: Canvas) : Pane() {
    init {
        children.add(canvas)

        canvas.widthProperty().bind(this.widthProperty())
        canvas.heightProperty().bind(this.heightProperty())
    }
}
```

Лістинг 12: EditorHandler.kt

```
package com.github.erotourtes.drawing

import com.github.erotourtes.drawing.editor.Editor
import com.github.erotourtes.drawing.editor.ShapesList
import com.github.erotourtes.utils.n
import javafx.beans.property.SimpleObjectProperty
import javafx.beans.value.ChangeListener
import javafx.scene.canvas.Canvas
import javafx.scene.canvas.GraphicsContext

class EditorHandler(private val canvas: Canvas) {
    private val shapes = ShapesList(n)
    private var map: MutableMap<Class<out Editor>, Editor> = mutableMapOf()
    private val curEditor = SimpleObjectProperty<Editor>()

    fun useEditor(editorClass: Class<out Editor>) {
        val editor = getOrCreateEditor(editorClass)
        editor.listenToEvents()
        curEditor.set(editor)
    }

    fun listenToChanges(subscriber: ChangeListener<Editor>) = curEditor.addListener(subscri

    private fun getOrCreateEditor(editorClass: Class<out Editor>): Editor = map.getOrPut(e
        editorClass.getConstructor(ShapesList::class.java, GraphicsContext::class.java)
            .newInstance(shapes, canvas.graphicsContext2D)
    }
}
```

3.6 Module: com.github.erotourtes.drawing.shape

Лістинг 13: Shape.kt

```
package com.github.erotourtes.drawing.shape

import com.github.erotourtes.utils.Dimension
import com.github.erotourtes.utils.drawOnce
import javafx.scene.canvas.GraphicsContext
import javafx.scene.paint.Color
import java.lang.RuntimeException

abstract class Shape(val gc: GraphicsContext) {
    protected val dm = Dimension()
```

```
    var colorFill: Color = Color.BLACK
    var colorStroke: Color = Color.BLACK

    abstract fun draw()

    open fun drawWithProperties() {
        gc.drawOnce {
            setProperties()
            draw()
        }
    }

    open fun setDm(curDm: Dimension) = curDm.copyTo(dm)

    private fun setProperties() {
        with(gc) {
            fill = colorFill
            stroke = colorStroke
        }
    }

    fun copy(): Shape {
        try {
            val shape = this::class.java.getConstructor(GraphicsContext::class.java).newIn
            shape.dm.copyFrom(dm)
            shape.colorFill = colorFill
            shape.colorStroke = colorStroke
            return shape
        } catch (e: Exception) {
            throw RuntimeException("Can't copy a shape")
        }
    }
}
```

Лістинг 14: Shapes.kt

```
package com.github.erotourtes.drawing.shape

import com.github.erotourtes.utils.*
import javafx.scene.canvas.GraphicsContext
import javafx.scene.paint.Color
import kotlin.math.abs

class Point(gc: GraphicsContext) : Shape(gc) {
    override fun draw() {
        val radius = 12.0
        gc.apply {
            val (x, y) = dm.getBoundaries().first
            fillOval(x, y, radius, radius)
        }
    }
}

class Line(gc: GraphicsContext) : Shape(gc) {
```

```
    override fun draw() {
        gc.apply { strokeLine(dm) }
    }
}

class Rect(gc: GraphicsContext) : Shape(gc) {
    init {
        colorFill = Color.TRANSPARENT
        colorStroke = Color.BLACK
    }

    override fun draw() {
        gc.apply {
            fillRect(dm)
            strokeRect(dm)
        }
    }
}

class Ellipse(gc: GraphicsContext) : Shape(gc) {
    init {
        colorFill = Color.ORANGE
        colorStroke = Color.BLACK
    }

    override fun draw() {
        gc.apply {
            fillOval(dm)
            strokeOval(dm)
        }
    }
}

class Dumbbell(gc: GraphicsContext) : Shape(gc) {
    override fun draw() {
        gc.apply {
            val radius = 24.0
            val (start, end) = dm.getRaw()

            fillOval(start.x - radius / 2, start.y - radius / 2, radius, radius)
            fillOval(end.x - radius / 2, end.y - radius / 2, radius, radius)

            strokeOval(start.x - radius / 2, start.y - radius / 2, radius, radius)
            strokeOval(end.x - radius / 2, end.y - radius / 2, radius, radius)

            strokeLine(dm)
        }
    }
}

class CubeEx(gc: GraphicsContext) : Shape(gc) {
    // failed math but got a nice effect
    override fun draw() {
        gc.apply {
```

```
            val (s, e) = dm.getRaw()
            val w = e.x - s.x
            val h = e.y - s.y

            val depthFactor = 0.5
            val size = abs(w.coerceAtLeast(h))
            val depthX = w * depthFactor
            val depthY = h * depthFactor

            val bgX = s.x + depthX
            val bgY = s.y - depthY

            strokeRect(s.x, s.y, size, size)
            strokeRect(bgX, bgY, size, size)

            strokeLine(s.x, s.y, bgX, bgY)
            strokeLine(s.x, s.y + size, bgX, bgY + size)
            strokeLine(s.x + size, s.y, bgX + size, bgY)
            strokeLine(s.x + size, s.y + size, bgX + size, bgY + size)
        }
    }
}

class Cube(gc: GraphicsContext) : Shape(gc) {
    override fun draw() {
        gc.apply {
            var (s, e) = dm.getRaw()
            val w = e.x - s.x
            val h = e.y - s.y

            val depthFactor = 0.5
            val sizeX = abs(w)
            val sizeY = abs(h)
            val depthX = w * depthFactor
            val depthY = h * depthFactor

            s = dm.getBoundaries().first

            val bgX = s.x + depthX
            val bgY = s.y - depthY

            strokeRect(s.x, s.y, sizeX, sizeY)
            strokeRect(bgX, bgY, sizeX, sizeY)

            strokeLine(s.x, s.y, bgX, bgY)
            strokeLine(s.x + sizeX, s.y, bgX + sizeX, bgY)
            strokeLine(s.x, s.y + sizeY, bgX, bgY + sizeY)
            strokeLine(s.x + sizeX, s.y + sizeY, bgX + sizeX, bgY + sizeY)
        }
    }
}
```

3.7 Module: com.github.erotourtes.styles

Лістинг 15: ToolbarStyles.kt

```
package com.github.erotourtes.styles

import tornadofx.*
import javafx.scene.paint.Color

class ToolbarStyles : Stylesheet() {

    companion object {
        val toolbar by cssclass()
        val iconButton by cssclass()
        val icon by cssclass()
        val dark = c("#555")
        val light = Color.LIGHTSTEELBLUE!!
    }

    init {
        val toolbarHeight = 40.px
        toolbar {
            padding = box(5.px)
            spacing = 5.px
            minHeight = toolbarHeight
            maxHeight = toolbarHeight
            alignment = javafx.geometry.Pos.CENTER_LEFT
            borderWidth += box(0.px, 0.px, 1.px, 0.px)
            borderColor += box(dark)
        }

        iconButton {
            backgroundColor += Color.TRANSPARENT

            and(selected) {
                backgroundColor += dark
                icon { fill = light }
            }

            minHeight = toolbarHeight / 1.5
            maxHeight = toolbarHeight / 1.5
        }

        icon { fill = dark }
    }
}
```

3.8 Module: com.github.erotourtes.utils

Лістинг 16: Dimension.kt

```
package com.github.erotourtes.utils

import kotlin.math.abs

class Dimension {
    private var x1: Double = 0.0
    private var y1: Double = 0.0
```

```
private var x2: Double = 0.0
private var y2: Double = 0.0

val width: Double
    get() = abs(x2 - x1)
val height: Double
    get() = abs(y2 - y1)

fun setStart(x: Double, y: Double): Dimension {
    x1 = x
    y1 = y
    return this
}

fun setEnd(x: Double, y: Double): Dimension {
    x2 = x
    y2 = y
    return this
}

fun copyTo(dst: Dimension) {
    dst.x1 = x1
    dst.y1 = y1
    dst.x2 = x2
    dst.y2 = y2
}

fun copyFrom(src: Dimension) = src.copyTo(this)

fun getBoundaries(): Pair<Point, Point> {
    return Pair(
        Point(x1.coerceAtMost(x2), y1.coerceAtMost(y2)),
        Point(x1.coerceAtLeast(x2), y1.coerceAtLeast(y2))
    )
}

fun getRaw(): Pair<Point, Point> {
    return Pair(
        Point(x1, y1),
        Point(x2, y2)
    )
}

data class Point(val x: Double, val y: Double)

override fun toString(): String = "Dimension(x1=$x1, y1=$y1, x2=$x2, y2=$y2)"

companion object {
    fun toCorner(dm: Dimension): Dimension {
        val (c, end) = dm.getRaw()

        // it is not width, it is half of the width; can be negative
        val w = end.x - c.x
        val h = end.y - c.y
```

```
            val sX = c.x - w
            val sY = c.y - h

            return Dimension().setStart(end.x, end.y).setEnd(sX, sY)
        }
    }
}
```

Лістинг 17: ExtensionFunctions.kt

```
package com.github.erotourtes.utils

import javafx.scene.canvas.GraphicsContext


fun GraphicsContext.fillRect(dm: Dimension) {
    val (s, e) = dm.getBoundaries()
    fillRect(s.x, s.y, e.x - s.x, e.y - s.y)
}

fun GraphicsContext.strokeRect(dm: Dimension) {
    val (s, e) = dm.getBoundaries()
    strokeRect(s.x, s.y, e.x - s.x, e.y - s.y)
}

fun GraphicsContext.fillOval(dm: Dimension) {
    val (s, e) = dm.getBoundaries()
    fillOval(s.x, s.y, e.x - s.x, e.y - s.y)
}

fun GraphicsContext.strokeOval(dm: Dimension) {
    val (s, e) = dm.getBoundaries()
    strokeOval(s.x, s.y, e.x - s.x, e.y - s.y)
}

fun GraphicsContext.strokeLine(dm: Dimension) {
    val (s, e) = dm.getRaw()
    strokeLine(s.x, s.y, e.x, e.y)
}

inline fun GraphicsContext.drawOnce(lambda: GraphicsContext.() -> Unit) {
    save()
    lambda(this)
    restore()
}
```

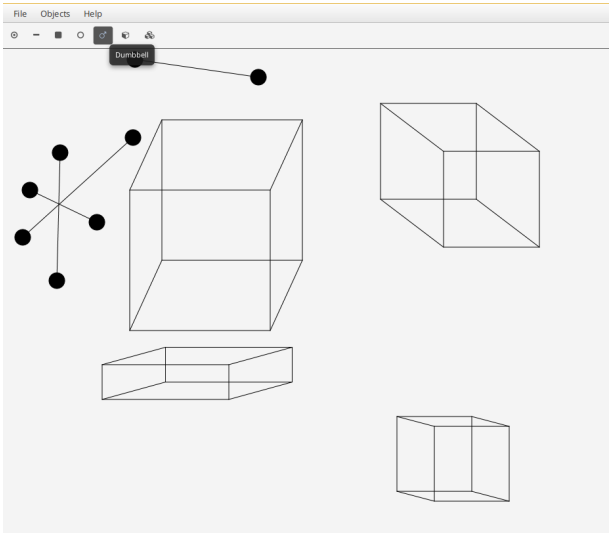
Лістинг 18: Utils.kt

```
package com.github.erotourtes.utils

import com.github.erotourtes.drawing.editor.Editor
import de.jensd.fx.glyphs.fontawesome.FontAwesomeIcon
import tornadofx.*
```

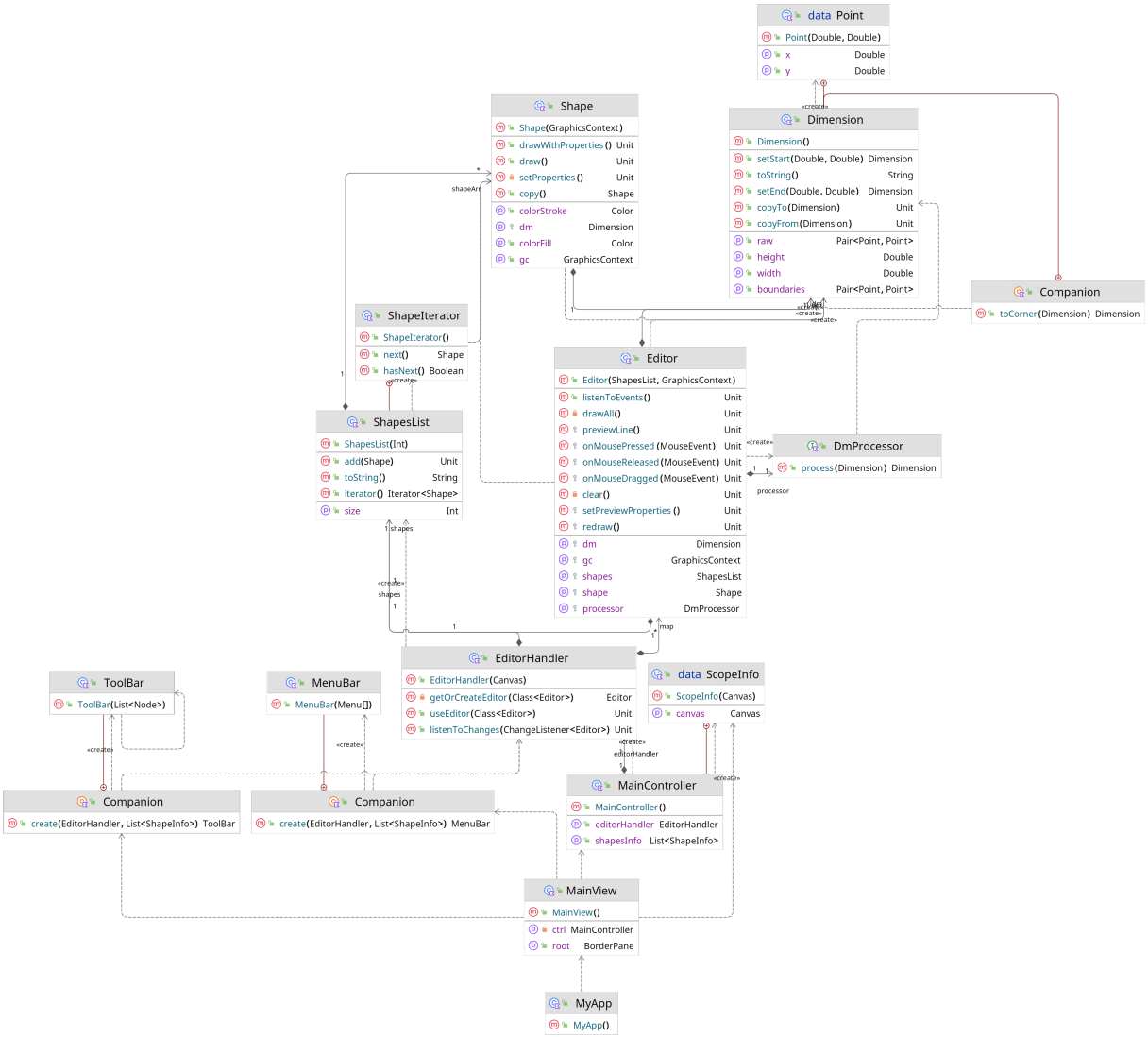
```
data class ShapeInfo(
    val name: String,
    val tooltip: String,
    val kotlinClass : Class<out Editor>,
    // [icons](https://fontawesome.com/v4/icons/)
    var icon: FontAwesomeIcon? = null,
)

const val g = 22 + 1
const val n = 100 + g

class PopupView : Fragment("Mode selection check") {
    private val action = super.scope as ScopeInfo

    override val root = vbox {
        style { prefWidth = 250.px }
        label("You clicked on ${action.name}")
        button("Close").action { close() }
    }

    data class ScopeInfo(val name: String) : Scope()
}
```

4 Ілюстрації:

4.1 Images



4.2 UML



5 Висновки:

Отже, я отримав вміння та навички проектування класів, виконавши модернізацію коду графічного редактора в об'єктно- орієнтованому стилі для забезпечення зручного додавання нових типів об'єктів.