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”

Виконав:
Студент групи ІМ-22
Кушнір Микола Миколайович
номер у списку групи: 13

Перевірив:
Порєв В.М.

Київ 2023

Мета

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 Desktop Application 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.

Умови завдання за варіантом (Ж = 13):

- Глобальний статичний об’єкт класу ***MyEditor*** у вигляді **Singleton Меєрса** *(13 % 2 ≠ 0)*

Вихідні тексти файлів програми

Lab5.kt

```
package com.oop.lab5

import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity

import com.oop.lab5.my_editor.MyEditor
import com.oop.lab5.my_table.MyTable
import com.oop.lab5.file_manager.FileManager
import com.oop.lab5.main_toolbar.MainToolbar
import com.oop.lab5.objects_toolbar.ObjectsToolbar
import com.oop.lab5.paint_view.PaintView
import com.oop.lab5.shape.Shape

class Lab5 : AppCompatActivity() {
    private lateinit var editor: MyEditor
    private lateinit var table: MyTable
    private lateinit var fileManager: FileManager
    private lateinit var mainToolbar: MainToolbar
    private lateinit var objectsToolbar: ObjectsToolbar

override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main_activity)

        // Стартові налаштування MyTable
        table = MyTable()
        table.setOnHideTableListener { hideTable() }
        table.setOnSelectRowListener { rowIndex ->
```

```
editor.selectShape(rowIndex) }
        table.setOnCancelRowsListener { rowsIndices ->
editor.cancelShapes(rowsIndices) }
        table.setOnDeleteRowsListener { rowsIndices ->
editor.deleteShapes(rowsIndices) }
        supportFragmentManager
            .beginTransaction()
            .add(R.id.table_container, table)
            .hide(table)
            .commit()

        // Стартові налаштування MyEditor
        editor = MyEditor.getInstance()
        editor.onCreate(this)
        val paintView = findViewById<PaintView>(R.id.paint_view)
        paintView.handler = editor
        editor.paintUtils = paintView
        editor.setOnNewShapeListener { shape ->
            table.addRow(editor.serializeShape(shape))
        }
        editor.setOnUndoListener { table.onUndo() }
        editor.setOnClearAllListener { table.onClearAll() }

        // Стартові налаштування MainToolbar
        mainToolbar = findViewById(R.id.main_toolbar)
        mainToolbar.onCreate(editor)
        mainToolbar.setFileListeners(
            { fileManager.files(supportFragmentManager) },
            { fileManager.save() },
            { fileManager.saveAs(supportFragmentManager) }
        )
        mainToolbar.setTableListener {
            if (!table.isDisplayed) showTable()
            else hideTable()
        }
        mainToolbar.setObjListeners(::selectObj, ::cancelObj)

        // Стартові налаштування ObjectsToolbar
        objectsToolbar = findViewById(R.id.objects_toolbar)
        objectsToolbar.onCreate(editor)
        objectsToolbar.setObjListeners(::selectObj, ::cancelObj)

        // Стартові налаштування FileManager
        fileManager = FileManager(this)
        fileManager.onCreate { fileName ->
            mainToolbar.setFileName(fileName)
        }
        fileManager.setOnFileListeners(
            { newFileName ->
                mainToolbar.setFileName(newFileName)
                editor.serializeDrawing()
            },
            { fileName, serializedDrawing ->
                mainToolbar.setFileName(fileName)
                editor.deserializeDrawing(serializedDrawing)
            },
```

```
                { editor.serializeDrawing() },
                { _, newFileName ->
                    if (newFileName != null) {
                        mainToolbar.setFileName(newFileName)
                        if (!editor.isDrawingEmpty()) editor.clearAll()
                    }
                }
            )
        }

        override fun onDestroy() {
            super.onDestroy()
            if (table.isDisplayed) hideTable()
        }

        private fun showTable() {
            table.isDisplayed = true
            supportFragmentManager
                .beginTransaction()
                .show(table)
                .commit()
            mainToolbar.onShowTable()
        }

        private fun hideTable() {
            table.isDisplayed = false
            supportFragmentManager
                .beginTransaction()
                .hide(table)
                .commit()
            mainToolbar.onHideTable()
        }

        private fun selectObj(shape: Shape) {
            mainToolbar.onSelectObj(shape)
            objectsToolbar.onObjSelect(shape)
            editor.start(shape)
        }

        private fun cancelObj() {
            mainToolbar.onCancelObj()
            objectsToolbar.onObjCancel()
            editor.close()
        }
    }
```

PaintUtils.kt

```
package com.oop.lab5.paint_view

import android.graphics.Canvas
```

```
interface PaintUtils {
    val drawnShapesCanvas: Canvas
    val rubberTraceCanvas: Canvas

    fun repaint()
    fun clearCanvas(canvas: Canvas)
}
```

PaintView.kt

```
package com.oop.lab5.paint_view

import android.content.Context
import android.graphics.Bitmap
import android.graphics.Canvas
import android.graphics.Color
import android.graphics.PorterDuff
import android.util.AttributeSet
import android.view.MotionEvent
import android.view.View

import com.oop.lab5.my_editor.PaintMessagesHandler

class PaintView(context: Context, attrs: AttributeSet?):
    View(context, attrs),
    PaintUtils {
    lateinit var handler: PaintMessagesHandler

    override lateinit var drawnShapesCanvas: Canvas
    override lateinit var rubberTraceCanvas: Canvas

    private lateinit var drawnShapesBitmap: Bitmap
    private lateinit var rubberTraceBitmap: Bitmap

    override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {
        super.onSizeChanged(w, h, oldw, oldh)
        drawnShapesBitmap = Bitmap.createBitmap(w, h,
Bitmap.Config.ARGB_8888)
        drawnShapesCanvas = Canvas(drawnShapesBitmap)
        rubberTraceBitmap = Bitmap.createBitmap(w, h,
Bitmap.Config.ARGB_8888)
        rubberTraceCanvas = Canvas(rubberTraceBitmap)
    }

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)
        if (!handler.isRubberTraceModeOn) {
            handler.onPaint()
            canvas.drawBitmap(drawnShapesBitmap, 0F, 0F, null)
        } else {
            canvas.drawBitmap(drawnShapesBitmap, 0F, 0F, null)
            canvas.drawBitmap(rubberTraceBitmap, 0F, 0F, null)
        }
```

```
    }

    override fun onTouchEvent(event: MotionEvent): Boolean {
        super.onTouchEvent(event)
        val x = event.x
        val y = event.y
        when (event.action) {
            MotionEvent.ACTION_DOWN -> handler.onFingerTouch(x, y)
            MotionEvent.ACTION_MOVE -> handler.onFingerMove(x, y)
            MotionEvent.ACTION_UP -> handler.onFingerRelease()
        }
        return true
    }

    override fun repaint() {
        invalidate()
    }

    override fun clearCanvas(canvas: Canvas) {
        canvas.drawColor(Color.TRANSPARENT, PorterDuff.Mode.MULTIPLY)
    }
}
```

PaintMessagesHandler.kt

```
package com.oop.lab5.shape_editor

interface PaintMessagesHandler {
    var isRubberTraceModeOn: Boolean

    fun onFingerTouch(x: Float, y: Float)
    fun onFingerMove(x: Float, y:Float)
    fun onFingerRelease()
    fun onPaint()
}
```

MyEditor.kt

```
package com.oop.lab5.my_editor

import android.content.Context
import java.lang.StringBuilder

import com.oop.lab5.paint_view.PaintUtils
import com.oop.lab5.shape.Shape
import com.oop.lab5.shape.PointShape
import com.oop.lab5.shape.LineShape
import com.oop.lab5.shape.RectShape
import com.oop.lab5.shape.EllipseShape
import com.oop.lab5.shape.SegmentShape
import com.oop.lab5.shape.CuboidShape
```

```
import com.oop.lab5.tooltip.Tooltip

class MyEditor private constructor(): PaintMessagesHandler {
    companion object {
        @Volatile
        private lateinit var instance: MyEditor

        fun getInstance(): MyEditor {
            synchronized(this) {
                if (!::instance.isInitialized) instance = MyEditor()
                return instance
            }
        }
    }

    lateinit var paintUtils: PaintUtils
    override var isRubberTraceModeOn = false

    lateinit var shapes: Array<Shape>
    var currentShape: Shape? = null
        private set
    private val drawnShapes = mutableListOf<Shape>()
    private val selectedShapesIndices = mutableListOf<Int>()

    private var onNewShapeListener: ((Shape) -> Unit)? = null

    private lateinit var onUndoListener: () -> Unit
    private lateinit var onClearAllListener: () -> Unit

    private lateinit var emptyDrawingTooltip: Tooltip

    fun onCreate(context: Context) {
        shapes = arrayOf(
            PointShape(context),
            LineShape(context),
            RectShape(context),
            EllipseShape(context),
            SegmentShape(context),
            CuboidShape(context),
        )
        emptyDrawingTooltip = Tooltip(context)
    }

    fun start(shape: Shape) {
        currentShape = shape
    }

    fun close() {
        currentShape = null
    }

    override fun onFingerTouch(x: Float, y: Float) {
        currentShape?.apply {
            setStart(x, y)
            setEnd(x, y)
        }
```

```
    }

    override fun onFingerMove(x: Float, y: Float) {
        currentShape?.let {
            isRubberTraceModeOn = true
            paintUtils.clearCanvas(paintUtils.rubberTraceCanvas)
            it.setEnd(x, y)
            it.showRubberTrace(paintUtils.rubberTraceCanvas)
            paintUtils.repaint()
        }
    }

    override fun onFingerRelease() {
        currentShape = currentShape?.let {
            isRubberTraceModeOn = false
            if (it.isValid()) addShape(it)
            paintUtils.repaint()
            it.getInstance()
        }
    }

    override fun onPaint() {
        paintUtils.clearCanvas(paintUtils.rubberTraceCanvas)
        paintUtils.clearCanvas(paintUtils.drawnShapesCanvas)
        drawnShapes.forEach {
            if (selectedShapesIndices.contains(drawnShapes.indexOf(it))) {
                it.showSelected(paintUtils.drawnShapesCanvas)
            } else {
                it.showDefault(paintUtils.drawnShapesCanvas)
            }
        }
    }

    fun serializeShape(shape: Shape): String {
        val str = StringBuilder()
        val coords = shape.getCoords()
        val fields = arrayOf(
            shape.name,
            coords.left.toInt(),
            coords.top.toInt(),
            coords.right.toInt(),
            coords.bottom.toInt()
        )
        (0..<(fields.size - 1)).forEach {
            str.append("${fields[it]}\t")
        }
        str.append("${fields.last()}")
        return str.toString()
    }

    fun deserializeShape(serializedShape: String): Shape {
        val data = serializedShape.split("\t")
        val fields = object {
            val name = data[0]
            val startX = data[1].toFloat()
            val startY = data[2].toFloat()
```

```
            val endX = data[3].toFloat()
            val endY = data[4].toFloat()
        }
        val shape = shapes.find { fields.name == it.name }!!.getInstance()
        shape.setStart(fields.startX, fields.startY)
        shape.setEnd(fields.endX, fields.endY)
        return shape
    }

    fun serializeDrawing(): String {
        val str = StringBuilder()
        drawnShapes.forEach {
            str.append("${serializeShape(it)}\n")
        }
        return str.toString()
    }

    fun deserializeDrawing(serializedDrawing: String) {
        if (!isDrawingEmpty()) clearAll()
        val serializedShapes = serializedDrawing.dropLast(1).split("\n")
        serializedShapes.forEach {
            addShape(deserializeShape(it))
        }
        paintUtils.repaint()
    }

    fun addShape(shape: Shape) {
        drawnShapes.add(shape)
        onNewShapeListener?.invoke(shape)
    }

    fun selectShape(index: Int) {
        selectedShapesIndices.add(index)
        paintUtils.repaint()
    }

    fun cancelShapes(indices: List<Int>) {
        for (index in indices) {
            selectedShapesIndices.remove(index)
        }
        paintUtils.repaint()
    }

    fun deleteShapes(indices: List<Int>) {
        if (!isDrawingEmpty()) {
            for (index in indices.sorted().sortedDescending()) {
                selectedShapesIndices.remove(index)
                drawnShapes.removeAt(index)
            }
            paintUtils.repaint()
        } else {
            emptyDrawingTooltip.create("Полотно уже порожнє").display()
        }
    }
```

```
    fun isDrawingEmpty(): Boolean {
        return drawnShapes.isEmpty()
    }

    fun undo() {
        deleteShapes(listOf(drawnShapes.size - 1))
        onUndoListener()
    }

    fun clearAll() {
        deleteShapes((0..<drawnShapes.size).toList())
        onClearAllListener()
    }

    fun setOnNewShapeListener(listener: ((Shape) -> Unit)?) {
        onNewShapeListener = listener
    }

    fun setOnUndoListener(listener: () -> Unit) {
        onUndoListener = listener
    }

    fun setOnClearAllListener(listener: () -> Unit) {
        onClearAllListener = listener
    }
}
```

Shape.kt

```
package com.oop.lab5.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.DashPathEffect
import android.graphics.Paint
import android.graphics.RectF
import com.oop.lab5.R

abstract class Shape(private val context: Context) {
    abstract val name: String
    val associatedIds = mutableMapOf<String, Int>()

    protected var startX: Float = 0F
    protected var startY: Float = 0F
    protected var endX: Float = 0F
    protected var endY: Float = 0F

    fun setStart(x: Float, y: Float) {
        startX = x
        startY = y
    }

    fun setEnd(x: Float, y: Float) {
        endX = x
```

```
            endY = y
        }

        abstract fun isValid(): Boolean

        abstract fun getInstance(): Shape

    fun getCoords(): RectF {
        return RectF(startX, startY, endX, endY)
    }

    protected open fun getOutlinePaint(mode: String): Paint {
        return Paint().apply {
            isAntiAlias = true
            style = Paint.Style.STROKE
            strokeWidth = 7F
            val modeActions = mapOf(
                "default" to {
                    color = context.getColor(R.color.black)
                },
                "selected" to {
                    color = context.getColor(R.color.selected_outline_color)
                },
                "rubberTrace" to {
                    color = context.getColor(R.color.dark_blue)
                    val dashLen = 30F
                    val spaceLen = 15F
                    val dashDensity = floatArrayOf(dashLen, spaceLen, dashLen,
    spaceLen)
                    pathEffect = DashPathEffect(dashDensity, 0F)
                },
            )
            modeActions[mode]?.invoke()
        }
    }

    protected open fun getFillingPaint(mode: String): Paint {
        return Paint().apply {
            isAntiAlias = true
            style = Paint.Style.FILL
        }
    }
        abstract fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
    Paint?)

        abstract fun showDefault(canvas: Canvas)

        abstract fun showSelected(canvas: Canvas)

        fun showRubberTrace(canvas: Canvas) {
            show(canvas, getOutlinePaint("rubberTrace"), null)
        }
    }
```

PointShape.kt

```
package com.oop.lab5.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import com.oop.lab5.R

class PointShape(private val context: Context): Shape(context) {
    override val name = context.getString(R.string.point)

    override fun isValid(): Boolean {
        return true
    }

    override fun getInstance(): Shape {
        return PointShape(context, editor).also {
            it.associatedIds.putAll(this.associatedIds)
        }
    }

    override fun getOutlinePaint(): Paint {
        return super.getOutlinePaint().apply {
            strokeWidth = 15F
        }
    }
    override fun getRubberTracePaint(): Paint {
        return super.getRubberTracePaint().apply {
            strokeWidth = 15F
        }
    }

    override fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?) {
        canvas.drawPoint(startX, startY, outlinePaint)
    }

    override fun showDefault(canvas: Canvas) {
        show(canvas, getOutlinePaint("default"), null)
    }

    override fun showSelected(canvas: Canvas) {
        show(canvas, getOutlinePaint("selected"), null)
    }
}
```

LineShape.kt

```
package com.oop.lab5.shape

import android.content.Context
import android.graphics.Canvas
```

```
import android.graphics.Paint
import com.oop.lab5.R

class LineShape(private val context: Context): Shape(context) {
    override val name = context.getString(R.string.line)

    override fun isValid(): Boolean {
        return (startX != endX || startY != endY)
    }

    override fun getInstance(): Shape {
        return LineShape(context, editor).also {
            it.associatedIds.putAll(this.associatedIds)
        }
    }

    override fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?) {
        canvas.drawLine(startX, startY, endX, endY, outlinePaint)
    }

    override fun showDefault(canvas: Canvas) {
        show(canvas, getOutlinePaint("default"), null)
    }

    override fun showSelected(canvas: Canvas) {
        show(canvas, getOutlinePaint("selected"), null)
    }
}
```

RectShape.kt

```
package com.oop.lab5.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.RectF
import com.oop.lab5.R

class RectShape(private val context: Context): Shape(context) {
    override val name = context.getString(R.string.rectangle)

    override fun isValid(): Boolean {
        return (startX != endX || startY != endY)
    }

    override fun getInstance(): Shape {
        return RectShape(context, editor).also {
            it.associatedIds.putAll(this.associatedIds)
        }
    }

    override fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
```

```
Paint?) {
        val rect = RectF(startX, startY, endX, endY)
        fillingPaint?.let {
            canvas.drawRect(rect, it)
        }
        canvas.drawRect(rect, outlinePaint)
    }

    override fun showDefault(canvas: Canvas) {
        show(canvas, getOutlinePaint("default"), null)
    }

    override fun showSelected(canvas: Canvas) {
        show(canvas, getOutlinePaint("selected"), null)
    }
}
```

EllipseShape.kt

```
package com.oop.lab5.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.RectF
import com.oop.lab5.R

class EllipseShape(private val context: Context): Shape(context) {
    override val name = context.getString(R.string.ellipse)

    override fun isValid(): Boolean {
        return (startX != endX || startY != endY)
    }

    override fun getInstance(): Shape {
        return EllipseShape(context, editor).also {
            it.associatedIds.putAll(this.associatedIds)
        }
    }

    override fun getFillingPaint(mode: String): Paint {
        return super.getFillingPaint(mode).apply {
            val modeActions = mapOf(
                "default" to {
                    color = context.getColor(R.color.light_green)
                },
                "selected" to {
                    color = context.getColor(R.color.selected_filling_color)
                },
            )
            modeActions[mode]?.invoke()
        }
    }
```

```
    override fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?) {
        val dx = endX -  startX
        val dy = endY - startY
        val rect = RectF(startX - dx, startY - dy, endX, endY).apply { sort()
}
        fillingPaint?.let {
            canvas.drawOval(rect, it)
        }
        canvas.drawOval(rect, outlinePaint)
    }

    override fun showDefault(canvas: Canvas) {
        show(canvas, getOutlinePaint("default"), getFillingPaint("default"))
    }

    override fun showSelected(canvas: Canvas) {
        show(canvas, getOutlinePaint("selected"),
getFillingPaint("selected"))
    }
}
```

LineShapeInterface.kt

```
package com.oop.lab5.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.PointF

interface LineShapeInterface {
    fun lineShapeShow(context: Context, canvas: Canvas, paint: Paint,
                      startPoint: PointF, endPoint: PointF) {
        val lineShape = LineShape(context)
        lineShape.setStart(startPoint.x, startPoint.y)
        lineShape.setEnd(endPoint.x, endPoint.y)
        lineShape.show(canvas, paint, null)
    }
}
```

RectShapeInterface.kt

```
package com.oop.lab5.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.RectF

interface RectShapeInterface {
    fun rectShapeShow(context: Context, canvas: Canvas,
                      outlinePaint: Paint, fillingPaint: Paint?,
```

```
                        rect: RectF) {
        val rectShape = RectShape(context)
        rectShape.setStart(rect.left, rect.top)
        rectShape.setEnd(rect.right, rect.bottom)
        rectShape.show(canvas, outlinePaint, fillingPaint)
    }
}
```

EllipseShapeInterface.kt

```
package com.oop.lab5.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.PointF

interface EllipseShapeInterface {
    fun ellipseShapeShow(context: Context, canvas: Canvas,
                         outlinePaint: Paint, fillingPaint: Paint?,
                         centerPoint: PointF, radius: Float) {
        val ellipseShape = EllipseShape(context)
        ellipseShape.setStart(centerPoint.x, centerPoint.y)
        ellipseShape.setEnd(centerPoint.x + radius, centerPoint.y + radius)
        ellipseShape.show(canvas, outlinePaint, fillingPaint)
    }
}
```

SegmentShape.kt

```
package com.oop.lab5.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.PointF
import com.oop.lab5.R
import kotlin.math.abs
import kotlin.math.acos
import kotlin.math.cos
import kotlin.math.sin
import kotlin.math.sqrt

class SegmentShape(private val context: Context):
    Shape(context),
    LineShapeInterface,
    EllipseShapeInterface {
    override val name = context.getString(R.string.segment)

    override fun isValid(): Boolean {
        return (startX != endX || startY != endY)
    }
```

```
    override fun getInstance(): Shape {
        return SegmentShape(context).also {
            it.associatedIds.putAll(this.associatedIds)
        }
    }

    override fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?) {
        if (!isValid()) return
        val ellipseRadius = 50F
        val startEllipseCenter = PointF(startX, startY)
        val endEllipseCenter = PointF(endX, endY)

        val dx = abs(endX - startX)
        val dy = abs(endY - startY)
        val distance = sqrt(dx * dx + dy * dy)
        val angle = acos(dx / distance)
        val offset = PointF(ellipseRadius * cos(angle), ellipseRadius *
sin(angle))

        val startTangentPoint = PointF()
        val endTangentPoint = PointF()
        if (startX < endX) {
            startTangentPoint.x = startX + offset.x
            endTangentPoint.x = endX - offset.x
        } else {
            startTangentPoint.x = startX - offset.x
            endTangentPoint.x = endX + offset.x
        }
        if (startY < endY) {
            startTangentPoint.y = startY + offset.y
            endTangentPoint.y = endY - offset.y
        } else {
            startTangentPoint.y = startY - offset.y
            endTangentPoint.y = endY + offset.y
        }
        lineShapeShow(context, canvas, outlinePaint, startTangentPoint,
endTangentPoint)
        ellipseShapeShow(context, canvas, outlinePaint, null,
            startEllipseCenter, ellipseRadius)
        ellipseShapeShow(context, canvas, outlinePaint, null,
            endEllipseCenter, ellipseRadius)
    }

    override fun showDefault(canvas: Canvas) {
        show(canvas, getOutlinePaint("default"), null)
    }

    override fun showSelected(canvas: Canvas) {
        show(canvas, getOutlinePaint("selected"), null)
    }
}
```

CuboidShape.kt

```
package com.oop.lab5.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.PointF
import android.graphics.RectF
import com.oop.lab5.R

class CuboidShape(private val context: Context):
    Shape(context),
    LineShapeInterface,
    RectShapeInterface {
    override val name = context.getString(R.string.cuboid)

    override fun isValid(): Boolean {
        return (startX != endX || startY != endY)
    }
    override fun getInstance(): Shape {
        return CuboidShape(context).also {
            it.associatedIds.putAll(this.associatedIds)
        }
    }

    override fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?) {
        val frontRect = RectF(startX, startY, endX, endY)
        rectShapeShow(context, canvas, outlinePaint, null, frontRect)
        val offset = 100F
        val backRect = RectF(frontRect).apply {
            offset(offset, -offset)
        }
        rectShapeShow(context, canvas, outlinePaint, null, backRect)
        frontRect.sort()
        backRect.sort()
        lineShapeShow(context, canvas, outlinePaint,
            PointF(frontRect.right, frontRect.top),
            PointF(backRect.right, backRect.top)
        )
        lineShapeShow(context, canvas, outlinePaint,
            PointF(frontRect.right, frontRect.bottom),
            PointF(backRect.right, backRect.bottom)
        )
        lineShapeShow(context, canvas, outlinePaint,
            PointF(frontRect.left, frontRect.bottom),
            PointF(backRect.left, backRect.bottom)
        )
        lineShapeShow(context, canvas, outlinePaint,
            PointF(frontRect.left, frontRect.top),
            PointF(backRect.left, backRect.top)
        )
    }
```

```
    override fun showDefault(canvas: Canvas) {
        show(canvas, getOutlinePaint("default"), null)
    }

    override fun showSelected(canvas: Canvas) {
        show(canvas, getOutlinePaint("selected"), null)
    }
}
```

MainToolbar.kt

```
package com.oop.lab5.main_toolbar

import android.content.Context
import android.graphics.PorterDuff
import android.graphics.PorterDuffColorFilter
import android.util.AttributeSet
import android.view.MenuItem
import android.view.View
import android.widget.ImageButton
import android.widget.PopupMenu
import android.widget.TextView
import androidx.appcompat.widget.Toolbar
import com.oop.lab5.R

import com.oop.lab5.my_editor.MyEditor
import com.oop.lab5.shape.Shape
import com.oop.lab5.tooltip.Tooltip

class MainToolbar(context: Context, attrs: AttributeSet?):
    Toolbar(context, attrs) {
    private lateinit var editor: MyEditor

    private lateinit var optionsMenu: PopupMenu
    private lateinit var fileSubmenu: PopupMenu
    private lateinit var objSubmenu: PopupMenu
    private lateinit var objSubmenuItems: Array<MenuItem>
    private lateinit var btnTable: ImageButton

    private lateinit var fileNameView: TextView

    private lateinit var onShowHideTableListener: () -> Unit

    private lateinit var onFilesListener: () -> Unit
    private lateinit var onSaveListener: () -> Unit
    private lateinit var onSaveAsListener: () -> Unit

    private lateinit var onSelectObjListener: (Shape) -> Unit
    private lateinit var onCancelObjListener: () -> Unit

    fun onCreate(editor: MyEditor) {
        this.editor = editor
        fileNameView = findViewById(R.id.current_file_name)
```

```
        val btnUndo = findViewById<ImageButton>(R.id.btn_undo)
        btnUndo.setOnClickListener {
            this.editor.undo()
        }
        val btnClearAll = findViewById<ImageButton>(R.id.btn_clear_all)
        btnClearAll.setOnClickListener {
            this.editor.clearAll()
        }
        btnTable = findViewById(R.id.btn_table)
        btnTable.setOnClickListener { onShowHideTableListener() }
        val btnOptions = findViewById<ImageButton>(R.id.btn_options)
        btnOptions.setOnClickListener {
            optionsMenu.show()
        }
        optionsMenu = createOptionsMenu(btnOptions)
        fileSubmenu = createFileSubmenu(btnOptions)
        objSubmenu = createObjSubmenu(btnOptions)
        objSubmenuItems = arrayOf(
            objSubmenu.menu.findItem(R.id.item_point),
            objSubmenu.menu.findItem(R.id.item_line),
            objSubmenu.menu.findItem(R.id.item_rectangle),
            objSubmenu.menu.findItem(R.id.item_ellipse),
            objSubmenu.menu.findItem(R.id.item_segment),
            objSubmenu.menu.findItem(R.id.item_cuboid),
        )
        for (index in objSubmenuItems.indices) {
            val shape = editor.shapes[index]
            val item = objSubmenuItems[index]
            shape.associatedIds["objSubmenuItem"] = item.itemId
        }
    }

    private fun createOptionsMenu(anchor: View): PopupMenu {
        val popupMenu = PopupMenu(context, anchor)
        popupMenu.menuInflater.inflate(R.menu.main_toolbar_options_menu,
popupMenu.menu)
        popupMenu.setOnMenuItemClickListener { item ->
            when(item.itemId) {
                R.id.file -> {
                    fileSubmenu.show()
                    true
                }
                R.id.objects -> {
                    objSubmenu.show()
                    true
                }
                R.id.info -> {
                    Tooltip(context)
                        .create("Ви натиснули кнопку\n\"Довідка\"")
                        .display()
                    true
                }
                else -> {
                    false
                }
            }
```

```
        }
        return popupMenu
    }

    private fun createFileSubmenu(anchor: View): PopupMenu {
        val popupMenu = PopupMenu(context, anchor)
        popupMenu.menuInflater.inflate(R.menu.main_toolbar_file_submenu,
popupMenu.menu)
        popupMenu.setOnMenuItemClickListener { item ->
            when(item.itemId) {
                R.id.files -> {
                    onFilesListener()
                    true
                }
                R.id.save -> {
                    onSaveListener()
                    true
                }
                R.id.save_as -> {
                    onSaveAsListener()
                    true
                }
                else -> {
                    false
                }
            }
        }
        return popupMenu
    }

    private fun createObjSubmenu(anchor: View): PopupMenu {
        val popupMenu = PopupMenu(context, anchor)
        popupMenu.menuInflater.inflate(R.menu.main_toolbar_objects_submenu,
popupMenu.menu)
        popupMenu.setOnMenuItemClickListener { clickedItem ->
            for (index in objSubmenuItems.indices) {
                val item = objSubmenuItems[index]
                if (item == clickedItem) {
                    if (!item.isChecked) {
                        val shape = editor.shapes[index]
                        onSelectObjListener(shape.getInstance())
                    } else {
                        onCancelObjListener()
                    }
                }
            }
            true
        }
        return popupMenu
    }

    fun setTableListener(listener: () -> Unit) {
        onShowHideTableListener = listener
    }
```

```
    fun onShowTable () {
        val iconColor =
context.getColor(R.color.on_main_toolbar_selected_btn_icon_color)
        btnTable.colorFilter = PorterDuffColorFilter(iconColor,
PorterDuff.Mode.SRC_IN)
    }

    fun onHideTable() {
        val iconColor = context.getColor(R.color.on_main_toolbar_color)
        btnTable.colorFilter = PorterDuffColorFilter(iconColor,
PorterDuff.Mode.SRC_IN)
    }

    fun setFileListeners(
        filesListener: () -> Unit,
        saveListener: () -> Unit,
        saveAsListener: () -> Unit
    ) {
        onFilesListener = filesListener
        onSaveListener = saveListener
        onSaveAsListener = saveAsListener
    }

    fun setFileName(fileName: String) {
        val maxFileNameLength = 12
        fileNameView.text =
            if (fileName.length <= maxFileNameLength) {
                fileName
            } else {
                "${fileName.substring(0..<maxFileNameLength)}..."
            }
    }

    fun setObjListeners(
        onSelectListener: (Shape) -> Unit,
        onCancelListener: () -> Unit
    ) {
        onSelectObjListener = onSelectListener
        onCancelObjListener = onCancelListener
    }

    fun onSelectObj(shape: Shape) {
        editor.currentShape?.let {
            val id = it.associatedIds["objSubmenuItem"]
            val item = objSubmenu.menu.findItem(id!!)
            item.isChecked = false
        }
        val id = shape.associatedIds["objSubmenuItem"]
        val item = objSubmenu.menu.findItem(id!!)
        item.isChecked = true
    }

    fun onCancelObj() {
        editor.currentShape?.let {
            val id = it.associatedIds["objSubmenuItem"]
            val item = objSubmenu.menu.findItem(id!!)
```

```
                item.isChecked = false
            }
        }
    }
```

ObjectsToolbar.kt

```
package com.oop.lab5.objects_toolbar

import android.content.Context
import android.util.AttributeSet
import androidx.appcompat.widget.Toolbar
import com.oop.lab5.R

import com.oop.lab5.my_editor.MyEditor
import com.oop.lab5.shape.Shape

class ObjectsToolbar(context: Context, attrs: AttributeSet?):
    Toolbar(context, attrs) {
    private lateinit var editor: MyEditor
    private lateinit var objButtons: Array<ObjectButton>

    private lateinit var onSelectObjListener: (Shape) -> Unit
    private lateinit var onCancelObjListener: () -> Unit

    fun onCreate(editor: MyEditor) {
        this.editor = editor
        objButtons = arrayOf(
            findViewById(R.id.btn_point),
            findViewById(R.id.btn_line),
            findViewById(R.id.btn_rectangle),
            findViewById(R.id.btn_ellipse),
            findViewById(R.id.btn_segment),
            findViewById(R.id.btn_cuboid),
        )
        for (index in objButtons.indices) {
            val shape = editor.shapes[index]
            val button = objButtons[index]
            shape.associatedIds["objButton"] = button.id
        }
    }

    fun setObjListeners(
        onSelectListener: (Shape) -> Unit,
        onCancelListener: () -> Unit
    ) {
        onSelectObjListener = onSelectListener
        onCancelObjListener = onCancelListener

        for (index in objButtons.indices) {
            val button = objButtons[index]
            val shape = editor.shapes[index]
            button.onCreate(shape)
            button.setObjListeners(onSelectObjListener, onCancelObjListener)
```

```
        }
    }

    fun onSelectObj(shape: Shape) {
        editor.currentShape?.let {
            val id = it.associatedIds["objButton"]
            val button = findViewById<ObjectButton>(id!!)
            button.onCancelObj()
        }
        val id = shape.associatedIds["objButton"]
        val button = findViewById<ObjectButton>(id!!)
        button.onSelectObj()
    }

    fun onCancelObj() {
        editor.currentShape?.let {
            val id = it.associatedIds["objButton"]
            val button = findViewById<ObjectButton>(id!!)
            button.onCancelObj()
        }
    }
}
```

ObjectButton.kt

```
package com.oop.lab5.objects_toolbar

import android.content.Context
import android.graphics.PorterDuff
import android.graphics.PorterDuffColorFilter
import android.util.AttributeSet
import android.view.MotionEvent
import com.oop.lab5.R

import com.oop.lab5.shape.Shape
import com.oop.lab5.tooltip.Tooltip

class ObjectButton(context: Context, attrs: AttributeSet?):
    androidx.appcompat.widget.AppCompatImageButton(context, attrs) {
    private lateinit var shape: Shape

    private var isSelected = false
    private lateinit var onSelectListener: (Shape) -> Unit
    private lateinit var onCancelListener: () -> Unit

    private val selectTooltip = Tooltip(context)
    private val cancelTooltip = Tooltip(context)

    private val timeOfLongPress = 1000
    private var pressStartTime: Long = 0
    private var pressEndTime: Long = 0

    fun onCreate(shape: Shape) {
        this.shape = shape
```

```
        val selectTooltipText = "Вибрати об\'єкт\n\"${shape.name}\""
        selectTooltip.create(selectTooltipText)
        val cancelTooltipText = "Вимкнути режим\nредагування"
        cancelTooltip.create(cancelTooltipText)
    }

    override fun onTouchEvent(event: MotionEvent): Boolean {
        when (event.action) {
            MotionEvent.ACTION_DOWN -> {
                markPressed()
                pressStartTime = System.currentTimeMillis()
            }
            MotionEvent.ACTION_UP -> {
                pressEndTime = System.currentTimeMillis()
                val pressDuration = pressEndTime - pressStartTime
                if (pressDuration < timeOfLongPress) {
                    performClick()
                } else {
                    performLongClick()
                }
                pressStartTime = 0
                pressEndTime = 0
            }
        }
        return true
    }

    override fun performClick(): Boolean {
        super.performClick()
        if (!isSelected) {
            onSelectListener(shape.getInstance())
        } else {
            onCancelListener()
        }
        return true
    }

    override fun performLongClick(): Boolean {
        super.performLongClick()
        if (!isSelected) {
            markNotPressed()
            selectTooltip.display()
        } else {
            markSelected()
            cancelTooltip.display()
        }
        return true
    }

    private fun markPressed() {
        val backgroundColorId = R.color.pressed_btn_background_color
        backgroundTintList = context.getColorStateList(backgroundColorId)
    }

    private fun markNotPressed() {
        val backgroundColorId = R.color.transparent
```

```
            backgroundTintList = context.getColorStateList(backgroundColorId)
        }

        private fun markSelected() {
            val backgroundColorId = R.color.selected_btn_background_color
            backgroundTintList = context.getColorStateList(backgroundColorId)
            val iconColor = context.getColor(R.color.selected_btn_icon_color)
            colorFilter = PorterDuffColorFilter(iconColor,
PorterDuff.Mode.SRC_IN)
        }

        private fun markNotSelected() {
            val backgroundColorId = R.color.transparent
            backgroundTintList = context.getColorStateList(backgroundColorId)
            val iconColor = context.getColor(R.color.on_objects_toolbar_color)
            colorFilter = PorterDuffColorFilter(iconColor,
PorterDuff.Mode.SRC_IN)
        }

        fun setObjListeners(
            onSelectListener: (Shape) -> Unit,
            onCancelListener: () -> Unit
        ) {
            this.onSelectListener = onSelectListener
            this.onCancelListener = onCancelListener
        }

        fun onSelectObj() {
            isSelected = true
            markSelected()
        }

        fun onCancelObj() {
            isSelected = false
            markNotSelected()
        }
    }
```

Tooltip.kt

```
package com.oop.lab5.tooltip

import android.app.Activity
import android.content.Context
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.TextView
import com.google.android.material.snackbar.Snackbar
import com.oop.lab5.R

class Tooltip(context: Context): View(context) {
    private lateinit var tooltip: Snackbar
```

```
    fun create(text: String): Tooltip {
        val activityView =
            (context as Activity).findViewById<View>(android.R.id.content)
        val displayDuration = Snackbar.LENGTH_LONG
        tooltip = Snackbar.make(activityView, "", displayDuration)

        val backgroundColor = context.getColor(R.color.transparent)
        tooltip.view.setBackgroundColor(backgroundColor)

        val layout = tooltip.view as ViewGroup
        val view = inflate(context, R.layout.tooltip, null)
        layout.addView(view)
        val textView = view.findViewById<TextView>(R.id.tooltip_text)
        textView.text = text

        val btnHide = view.findViewById<Button>(R.id.tooltip_hide)
        btnHide.setOnClickListener {
            val textColor =
                context.getColor(R.color.tooltip_bnt_clicked_text_color)
            btnHide.setTextColor(textColor)
            hide()
        }
        return this
    }

    fun hide() {
        tooltip.dismiss()
    }

    fun display() {
        tooltip.show()
    }
}
```

MyTable.kt

```
package com.oop.lab5.my_table

import android.graphics.Typeface
import android.os.Bundle
import android.view.Gravity
import android.view.View
import android.widget.Button
import android.widget.LinearLayout
import android.widget.ScrollView
import android.widget.TableLayout
import android.widget.TableRow
import android.widget.TextView
import androidx.core.view.children
import androidx.fragment.app.Fragment
import com.oop.lab5.R
```

```
class MyTable: Fragment(R.layout.table) {
    var isDisplayed = false

    private lateinit var scrollView: ScrollView
    private lateinit var tableLayout: TableLayout

    private lateinit var bottomView: LinearLayout
    private lateinit var defaultBottomView: LinearLayout
    private lateinit var selectBottomView: LinearLayout

    private val selectedRowsIndices = mutableListOf<Int>()
    private var onHideTableListener: (() -> Unit)? = null
    private var onSelectRowListener: ((Int) -> Unit)? = null
    private var onCancelRowsListener: ((List<Int>) -> Unit)? = null
    private var onDeleteRowsListener: ((List<Int>) -> Unit)? = null

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        scrollView = view.findViewById(R.id.table_scroll_view)
        tableLayout = view.findViewById(R.id.table_table_layout)
        bottomView = view.findViewById(R.id.files_dialog_bottom_view)

        defaultBottomView = LinearLayout(context)
        layoutInflater.inflate(
            R.layout.table_default_bottom_view, defaultBottomView, true
        )
        val buttonHide = defaultBottomView
            .findViewById<Button>(R.id.files_dialog_btn_hide)
        buttonHide.setOnClickListener {
            onHideTableListener?.invoke()
        }

        selectBottomView = LinearLayout(context)
        layoutInflater.inflate(
            R.layout.table_select_bottom_view, selectBottomView, true
        )
        val buttonCancel = selectBottomView
            .findViewById<Button>(R.id.files_dialog_btn_open)
        buttonCancel.setOnClickListener {
            cancelRows(selectedRowsIndices.toList())
        }
        val buttonDelete = selectBottomView
            .findViewById<Button>(R.id.files_dialog_btn_delete)
        buttonDelete.setOnClickListener {
            val indices = selectedRowsIndices.toList()
            deleteRows(indices)
            onDeleteRowsListener?.invoke(indices)
        }

        bottomView.addView(defaultBottomView)
    }

    fun addRow(serializedShape: String) {
        val data = serializedShape.dropLast(1).split("\t")
        val fields = object {
```

```
            val name = data[0]
            val x1 = data[1]
            val y1 = data[2]
            val x2 = data[3]
            val y2 = data[4]
        }
        val row = TableRow(context)
        layoutInflater.inflate(R.layout.table_row, row, true)
        row.findViewById<TextView>(R.id.table_shape_name).text = fields.name
        row.findViewById<TextView>(R.id.table_x1).text = fields.x1
        row.findViewById<TextView>(R.id.table_y1).text = fields.y1
        row.findViewById<TextView>(R.id.table_x2).text = fields.x2
        row.findViewById<TextView>(R.id.table_y2).text = fields.y2

        row.setOnClickListener {
            val rowIndex = tableLayout.indexOfChild(it)
            if (!selectedRowsIndices.contains(rowIndex)) {
                selectRow(rowIndex)
            } else {
                cancelRows(listOf(rowIndex))
            }
        }
        tableLayout.addView(row)
        val firstChild = tableLayout.children.first()
        if (firstChild is TextView) {
            tableLayout.removeView(firstChild)
        }
        setDefaultRowBgColor(tableLayout.indexOfChild(row))
        scrollView.scrollToDescendant(row)
    }

    private fun selectRow(index: Int) {
        if (selectedRowsIndices.isEmpty()) {
            bottomView.removeView(defaultBottomView)
            bottomView.addView(selectBottomView)
        }
        selectedRowsIndices.add(index)
        setSelectedRowBgColor(index)
        onSelectRowListener?.invoke(index)
    }

    private fun cancelRows(indices: List<Int>) {
        for (index in indices) {
            selectedRowsIndices.remove(index)
            setDefaultRowBgColor(index)
        }
        if (selectedRowsIndices.isEmpty()) {
            bottomView.removeView(selectBottomView)
            bottomView.addView(defaultBottomView)
        }
        onCancelRowsListener?.invoke(indices)
    }

    fun deleteRows(indices: List<Int>) {
        for (index in indices.sorted().sortedDescending()) {
            selectedRowsIndices.remove(index)
```

```
            val row = tableLayout.getChildAt(index)
            tableLayout.removeView(row)
        }
        (indices.min()..<tableLayout.childCount).forEach {
            setDefaultRowBgColor(it)
        }
        if (tableLayout.childCount == 0) {
            if (bottomView.children.first() == selectBottomView) {
                bottomView.removeView(selectBottomView)
                bottomView.addView(defaultBottomView)
            }
            val textView = TextView(context).apply {
                layoutParams = LinearLayout.LayoutParams(
                    LinearLayout.LayoutParams.MATCH_PARENT,
                        resources
                            .getDimension(
                                R.dimen.table_content_height
                            ).toInt()
                )
                text = "Полотно порожнє"
                textSize = 20F
                setTypeface(null, Typeface.ITALIC)
                gravity = Gravity.CENTER
            }
            tableLayout.addView(textView)
        } else if (selectedRowsIndices.isEmpty()) {
            if (bottomView.children.first() == selectBottomView) {
                bottomView.removeView(selectBottomView)
                bottomView.addView(defaultBottomView)
            }
        }
    }

    private fun setDefaultRowBgColor(index: Int) {
        val row = tableLayout.getChildAt(index)
        row.setBackgroundColor(
            if (index % 2 == 0) {
                requireActivity()
                    .getColor(R.color.table_default_row_bg_color_1)
            } else {
                requireActivity()
                    .getColor(R.color.table_default_row_bg_color_2)
            }
        )
    }

    private fun setSelectedRowBgColor(index: Int) {
        val row = tableLayout.getChildAt(index)
        row.setBackgroundColor(
            if (index % 2 == 0) {
                requireActivity()
                    .getColor(R.color.table_selected_row_bg_color_1)
            } else {
                requireActivity()
                    .getColor(R.color.table_selected_row_bg_color_2)
            }
```

```
            )
        }

        fun onUndo() {
            deleteRows(listOf(tableLayout.childCount - 1))
        }

        fun onClearAll() {
            deleteRows((0..<tableLayout.childCount).toList())
        }

        fun setOnHideTableListener(listener: () -> Unit) {
            onHideTableListener = listener
        }

        fun setOnSelectRowListener(listener: (Int) -> Unit) {
            onSelectRowListener = listener
        }

        fun setOnCancelRowsListener(listener: (List<Int>) -> Unit) {
            onCancelRowsListener = listener
        }

        fun setOnDeleteRowsListener(listener: (List<Int>) -> Unit) {
            onDeleteRowsListener = listener
        }
    }
```

FileManager.kt

```
package com.oop.lab5.file_manager

import android.content.Context
import androidx.fragment.app.FragmentManager
import com.oop.lab5.R
import com.oop.lab5.tooltip.Tooltip
import java.io.BufferedReader
import java.io.File
import java.io.FileReader
import java.io.FileWriter

class FileManager(private val context: Context) {
    private lateinit var fileExtension: String
    private lateinit var path: String
    private lateinit var drawingsDir: File
    private lateinit var currentFile: File

    private lateinit var createFileDialog: CreateFileDialog
    private lateinit var filesDialog: FilesDialog

    private lateinit var onCreateFileListener: (String) -> String
    private lateinit var onOpenFileListener: (String, String) -> Unit
    private lateinit var onSaveFileListener: () -> String
```

```
    private lateinit var onDeleteFileListener: (String, String?) -> Unit

    fun onCreate(startListener: (String) -> Unit) {
        val root = context.getExternalFilesDir(null)
        val drawingsDirName = context.getString(R.string.drawings_dir_name)
        drawingsDir = File(root, drawingsDirName)
        if (!drawingsDir.exists()) drawingsDir.mkdirs()
        path = drawingsDir.absolutePath
        fileExtension = context.getString(R.string.file_extension)
        val fileName = getDefaultFileName()
        currentFile = File(path, fileName)
        startListener(fileName)

        createFileDialog = CreateFileDialog()
        createFileDialog.setFileCreationListeners({}, ::createFile)

        filesDialog = FilesDialog()
        filesDialog.setOnFileListeners(::openFile, ::deleteFile)
    }

    fun files(manager: FragmentManager) {
        filesDialog.display(manager, drawingsDir.list())
    }

    fun save() {
        val str = onSaveFileListener()
        val writer = FileWriter(currentFile)
        writer.append(str)
        writer.flush()
        writer.close()
        Tooltip(context)
            .create("Малюнок збережено у файлі ${currentFile.name}")
            .display()
    }

    fun saveAs(manager: FragmentManager) {
        createFileDialog.display(manager,
getShortFileName(getDefaultFileName()))
    }

    private fun getShortFileName(fileName: String): String {
        return fileName.removeSuffix(fileExtension)
    }

    private fun getShortFileNames(): List<String>? {
        return drawingsDir.list()?.map {
            getShortFileName(it)
        }
    }

    private fun getDefaultFileName(): String {
        val nameStart = context.getString(R.string.default_short_file_name)
        var nameEnd = 1
        var name = "$nameStart$nameEnd"
        val shortFileNames = getShortFileNames()
        if (shortFileNames != null) {
```

```
            while (name in shortFileNames) {
                nameEnd++
                name = "$nameStart$nameEnd"
            }
        }
        return name + fileExtension
    }

    private fun openFile(fileName: String) {
        val serializedDrawing = StringBuilder()
        currentFile = File(path, fileName)
        val bufferReader = BufferedReader(FileReader(currentFile))
        var text: String? = bufferReader.readLine()
        while (text != null) {
            serializedDrawing.append("$text\n")
            text = bufferReader.readLine()
        }
        bufferReader.close()
        onOpenFileListener(fileName, serializedDrawing.toString())
    }

    private fun createFile(shortFileName: String): Pair<Boolean, String> {
        val shortFileNames = getShortFileNames()
        return if (shortFileName == "") {
            false to "Порожнє ім'я"
        } else if (
            shortFileNames != null &&
            shortFileNames.contains(getShortFileName(shortFileName))
            ) {
            false to "Використане ім'я"
        } else {
            val fileName = shortFileName + fileExtension
            currentFile = File(path, fileName)
            val str = onCreateFileListener(fileName)
            val writer = FileWriter(currentFile)
            writer.append(str)
            writer.flush()
            writer.close()
            true to "Малюнок збережено у файлі $fileName"
        }
    }

    private fun deleteFile(fileName: String) {
        val file = File(path, fileName)
        file.delete()
        onDeleteFileListener(fileName,
            if (file.name != currentFile.name) null
            else getDefaultFileName()
        )
    }

    fun setOnFileListeners(
        createListener: (String) -> String,
        openListener: (String, String) -> Unit,
        saveListener: () -> String,
        deleteListener: (String, String?) -> Unit
```

```
    ) {
        onCreateFileListener = createListener
        onOpenFileListener = openListener
        onSaveFileListener = saveListener
        onDeleteFileListener = deleteListener
    }
}
```

CreateFileDialog.kt

```
package com.oop.lab5.file_manager

import android.app.Dialog
import android.graphics.Color
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.EditText
import androidx.fragment.app.DialogFragment
import androidx.fragment.app.FragmentManager
import com.oop.lab5.R

import com.oop.lab5.tooltip.Tooltip

class CreateFileDialog: DialogFragment(R.layout.create_file_dialog) {
    private lateinit var editText: EditText
    private var hint = ""

    private lateinit var onCancelListener: () -> Unit
    private lateinit var onConfirmListener: (String) -> Pair<Boolean, String>

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setStyle(STYLE_NORMAL, R.style.Dialog)
    }

    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        return super.onCreateDialog(savedInstanceState).apply {
            setCancelable(false)
            setCanceledOnTouchOutside(false)
        }
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        editText = view.findViewById(R.id.enter_file_name)
        editText.hint = hint
        val buttonCancel = view
            .findViewById<Button>(R.id.files_dialog_btn_open)
        buttonCancel.setOnClickListener {
            editText.text.clear()
            onCancelListener()
            dismiss()
```

```
        }
        val buttonOkay = view
            .findViewById<Button>(R.id.files_dialog_btn_delete)
        buttonOkay.setOnClickListener {
            val text = editText.text.toString()
            editText.text.clear()
            val (isNameValid, message) = onConfirmListener(text)
            if (isNameValid) {
                dismiss()
                Tooltip(requireActivity()).create(message).display()
            } else {
                editText.setHintTextColor(Color.RED)
                editText.hint = message
            }
        }
    }

    fun display(manager: FragmentManager, nameHint: String) {
        hint = nameHint
        show(manager, "create_file_dialog")
    }

    fun setFileCreationListeners(
        cancelListener: () -> Unit,
        confirmListener: (String) -> Pair<Boolean, String>
    ) {
        onConfirmListener = confirmListener
        onCancelListener = cancelListener
    }
}
```

FilesDialog.kt

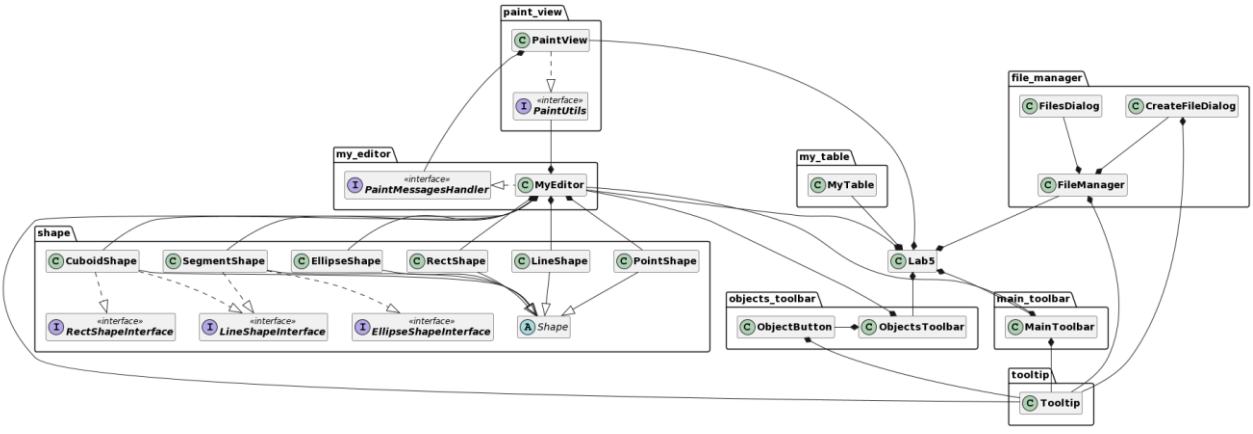
```
package com.oop.lab5.file_manager

import android.app.Dialog
import android.graphics.Typeface
import android.os.Bundle
import android.view.Gravity
import android.view.View
import android.widget.Button
import android.widget.LinearLayout
import android.widget.TableLayout
import android.widget.TableRow
import android.widget.TextView
import androidx.fragment.app.DialogFragment
import androidx.fragment.app.FragmentManager
import com.oop.lab5.R

class FilesDialog : DialogFragment(R.layout.files_dialog) {
    private lateinit var tableLayout: TableLayout
    private lateinit var bottomView: LinearLayout
    private lateinit var defaultBottomView: LinearLayout
```

```
private lateinit var selectBottomView: LinearLayout

private var currentFileList = mutableListOf<String>()
private var selectedRow = object {
    var view: TableRow? = null
    var fileName: String? = null
}

private lateinit var onOpenListener: (String) -> Unit
private lateinit var onDeleteListener: (String) -> Unit

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setStyle(STYLE_NORMAL, R.style.Dialog)
}

override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
    return super.onCreateDialog(savedInstanceState).apply {
        setCancelable(false)
        setCanceledOnTouchOutside(false)
    }
}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    tableLayout = view.findViewById(R.id.files_dialog_table_layout)
    tableLayout.removeAllViews()
    if (currentFileList.isNotEmpty()) {
        for (file in currentFileList) {
            val row = TableRow(context)
            layoutInflater.inflate(R.layout.files_dialog_row, row, true)
            row.findViewById<TextView>(R.id.files_dialog_row_name).text =
                file
            row.setOnClickListener {
                val selectedRowView = selectedRow.view
                if (selectedRowView != null) cancelRow()
                if (selectedRowView != it) selectRow(it as TableRow)
            }
            tableLayout.addView(row)
        }
    } else {
        onEmptyDir()
    }
    bottomView = view.findViewById(R.id.files_dialog_bottom_view)
    defaultBottomView = LinearLayout(context)
    layoutInflater.inflate(
        R.layout.files_dialog_default_bottom_view,
        defaultBottomView, true
    )
    val buttonHide = defaultBottomView
        .findViewById<Button>(R.id.files_dialog_btn_hide)
    buttonHide.setOnClickListener { dismiss() }
    selectBottomView = LinearLayout(context)
    layoutInflater.inflate(
        R.layout.files_dialog_select_bottom_view,
```

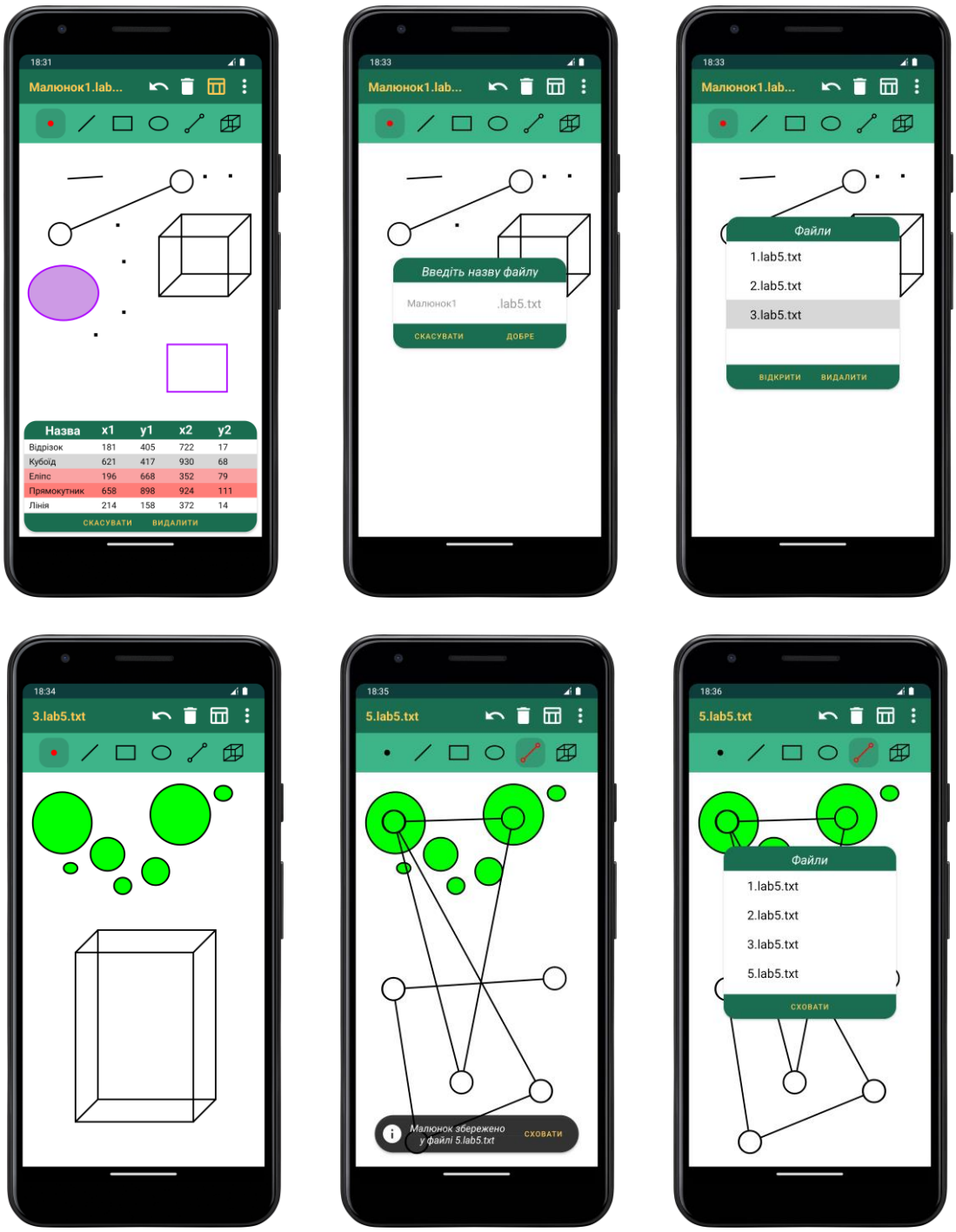
```
            selectBottomView, true
        )
        val buttonOpen = selectBottomView
            .findViewById<Button>(R.id.files_dialog_btn_open)
        buttonOpen.setOnClickListener {
            onOpenListener(selectedRow.fileName!!)
            cancelRow()
            dismiss()
        }
        val buttonDelete = selectBottomView
            .findViewById<Button>(R.id.files_dialog_btn_delete)
        buttonDelete.setOnClickListener {
            deleteRow()
        }
        bottomView.addView(defaultBottomView)
    }

    fun display(manager: FragmentManager, fileList: Array<String>?) {
        currentFileList.clear()
        if (!fileList.isNullOrEmpty()) {
            currentFileList.addAll(fileList)
        }
        show(manager, "files_dialog")
    }

    private fun onEmptyDir() {
        val textView = TextView(context).apply {
            layoutParams = LinearLayout.LayoutParams(
                LinearLayout.LayoutParams.MATCH_PARENT,
                resources.getDimension(
                    R.dimen.files_dialog_content_height
                ).toInt()
            )
            gravity = Gravity.CENTER
            text = context.getString(R.string.files_dialog_default_text)
            textSize = 20F
            setTypeface(null, Typeface.ITALIC)
        }
        tableLayout.addView(textView)
    }

    private fun selectRow(row: TableRow) {
        bottomView.removeView(defaultBottomView)
        bottomView.addView(selectBottomView)
        row.setBackgroundColor(
            requireActivity()
                .getColor(R.color.files_dialog_selected_row_bg_color)
        )
        selectedRow.view = row
        selectedRow.fileName = row
            .findViewById<TextView>(R.id.files_dialog_row_name)
            .text
            .toString()
    }
```

```
    private fun cancelRow() {
        bottomView.removeView(selectBottomView)
        bottomView.addView(defaultBottomView)
        selectedRow.view!!.setBackgroundColor(
            requireActivity()
                .getColor(R.color.files_dialog_default_row_bg_color)
        )
        selectedRow.view = null
        selectedRow.fileName = null
    }

    private fun deleteRow() {
        bottomView.removeView(selectBottomView)
        bottomView.addView(defaultBottomView)

        currentFileList.remove(selectedRow.fileName)
        tableLayout.removeView(selectedRow.view)
        onDeleteListener(selectedRow.fileName!!)
        selectedRow.view = null
        selectedRow.fileName = null
        if (currentFileList.isEmpty()) onEmptyDir()
    }

    fun setOnFileListeners(
        openListener: (String) -> Unit,
        deleteListener: (String) -> Unit
    ) {
        onOpenListener = openListener
        onDeleteListener = deleteListener
    }
}
```

Діаграма класів програми



Ілюстрації виконання програми



Висновки

Під час виконання цієї лабораторної роботи я запрограмував багатовіконний інтерфейс в об’єктно-орієнтованому стилі для раніше створеного графічного редактора, написаного на мові програмування ***Kotlin*** та доступного для платформи ***Android***. Зокрема було реалізовано вікно таблиці, що дозволяє зручно маніпулювати уже намальованими фігурами. Окрім цього тепер користувач матиме можливість зберігати малюнки у файли (формату ***.txt***), оскільки я додав до наявного функціоналу програми базові операції для роботи з файлами (створення, читання, оновлення та

видалення). Останнім нововведенням було унеможливлення одночасного створення декількох екземплярів класу ***MyEditor*** за допомогою шаблону ***Singleton***.