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-31
Максимовський Назар Русланович
номер у списку групи: 13

Перевірив:
Порєв В.М.

Київ 2024

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)

        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()

        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() }

        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)

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

        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.*
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 coords = shape.getCoords()
        return
"${shape.name}\t${coords.left.toInt()}\t${coords.top.toInt()}\t${coords.right
.toInt()}\t${coords.bottom.toInt()}"
    }

    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()
        }
        return shapes.find { it.name == fields.name }?.getInstance()?.apply {
            setStart(fields.startX, fields.startY)
            setEnd(fields.endX, fields.endY)
        } ?: throw IllegalArgumentException("Unknown shape name:
```

```
${fields.name}")
    }

    fun serializeDrawing(): String = drawnShapes.joinToString("\n") {
serializeShape(it) }

    fun deserializeDrawing(serializedDrawing: String) {
        if (!isDrawingEmpty()) clearAll()
        serializedDrawing.lines().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>) {
        selectedShapesIndices.removeAll(indices)
        paintUtils.repaint()
    }

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

    fun isDrawingEmpty(): Boolean = drawnShapes.isEmpty()

    fun undo() {
        if (drawnShapes.isNotEmpty()) {
            drawnShapes.removeLast()
            onUndoListener()
        }
    }

    fun clearAll() {
        drawnShapes.clear()
        onClearAllListener()
        paintUtils.repaint()
    }

    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).also {
            it.associatedIds.putAll(this.associatedIds)
        }
    }

    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).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).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).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.*

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

    override fun isValid() = startX != endX || startY != endY

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

    override fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?) {
        if (!isValid()) return

        val ellipseRadius = 50F
        val startPoint = PointF(startX, startY)
        val endPoint = PointF(endX, endY)
        val distance = hypot(endX - startX, endY - startY)
        val angle = acos((endX - startX) / distance)
        val offset = PointF(ellipseRadius * cos(angle), ellipseRadius *
sin(angle))

        val tangentPoints = listOf(
            PointF(startX + offset.x, startY + offset.y),
            PointF(endX - offset.x, endY - offset.y)
        )

        showLine(context, canvas, outlinePaint, tangentPoints[0],
tangentPoints[1])
        showEllipse(context, canvas, outlinePaint, null, startPoint,
ellipseRadius)
        showEllipse(context, canvas, outlinePaint, null, endPoint,
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),
ShapeInterface {
    override val name = context.getString(R.string.cuboid)

    override fun isValid() = startX != endX || startY != endY

    override fun getInstance(): Shape = CuboidShape(context).also {
        it.associatedIds.putAll(associatedIds)
    }

    override fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?) {
        val frontRect = RectF(startX, startY, endX, endY)
        val backRect = RectF(frontRect).apply { offset(100F, -100F) }

        listOf(frontRect, backRect).forEach { rect ->
            showRect(context, canvas, outlinePaint, null, rect)
        }

        listOf(
            PointF(frontRect.right, frontRect.top) to PointF(backRect.right,
backRect.top),
            PointF(frontRect.right, frontRect.bottom) to
PointF(backRect.right, backRect.bottom),
            PointF(frontRect.left, frontRect.bottom) to PointF(backRect.left,
backRect.bottom),
            PointF(frontRect.left, frontRect.top) to PointF(backRect.left,
backRect.top)
        ).forEach { (start, end) ->
            showLine(context, canvas, outlinePaint, start, end)
        }
    }

    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 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
        }
    }

    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)
    }
}
```

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>

    fun initialize(editor: MyEditor,
                   onSelectListener: (Shape) -> Unit,
                   onCancelListener: () -> Unit) {
        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),
        )

        objButtons.forEachIndexed { index, button ->
            val shape = editor.shapes[index]
            shape.associatedIds["objButton"] = button.id
            button.setup(shape, onSelectListener, onCancelListener)
        }
    }

    fun onSelectObj(shape: Shape) {
        editor.currentShape?.associatedIds?.get("objButton")?.let { id ->
            findViewById<ObjectButton>(id)?.onCancelObj()
```

```
        }
        shape.associatedIds["objButton"]?.let { id ->
            findViewById<ObjectButton>(id)?.onSelectObj()
        }
    }

    fun onCancelObj() {
        editor.currentShape?.associatedIds?.get("objButton")?.let { id ->
            findViewById<ObjectButton>(id)?.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 by lazy { Tooltip(context).apply { create("") } }
}
    private val cancelTooltip by lazy { Tooltip(context).apply { create("") } }
}

    private val longPressThreshold = 1000L
    private var pressStartTime: Long = 0

    fun setup(shape: Shape,
              onSelectListener: (Shape) -> Unit,
              onCancelListener: () -> Unit) {
        this.shape = shape
        this.onSelectListener = onSelectListener
        this.onCancelListener = onCancelListener

        selectTooltip.updateText("Select: ${shape.name}")
        cancelTooltip.updateText("Cancel editing")
    }

    override fun onTouchEvent(event: MotionEvent): Boolean {
        when (event.action) {
            MotionEvent.ACTION_DOWN -> {
                markPressed()
                pressStartTime = System.currentTimeMillis()
            }
            MotionEvent.ACTION_UP -> {
                val pressDuration = System.currentTimeMillis() -
pressStartTime
                if (pressDuration < longPressThreshold) performClick() else
```

```
performLongClick()
                resetPressTime()
            }
        }
        return true
    }

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

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

    private fun resetPressTime() {
        pressStartTime = 0
    }

    private fun markPressed() {
        setBackgroundTint(R.color.pressed_btn_background_color)
    }

    private fun setBackgroundTint(colorRes: Int) {
        backgroundTintList = context.getColorStateList(colorRes)
    }

    private fun setIconTint(colorRes: Int) {
        colorFilter = PorterDuffColorFilter(
            context.getColor(colorRes), PorterDuff.Mode.SRC_IN
        )
    }

    fun onSelectObj() {
        isSelected = true
        setBackgroundTint(R.color.selected_btn_background_color)
        setIconTint(R.color.selected_btn_icon_color)
    }

    fun onCancelObj() {
        isSelected = false
        setBackgroundTint(R.color.transparent)
        setIconTint(R.color.on_objects_toolbar_color)
    }
}
```

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 {
        tooltip = Snackbar.make((context as
Activity).findViewById(android.R.id.content), "", Snackbar.LENGTH_LONG)

tooltip.view.setBackgroundColor(context.getColor(R.color.transparent))

        val layout = tooltip.view as ViewGroup
        val view = inflate(context, R.layout.tooltip, null).apply {
            findViewById<TextView>(R.id.tooltip_text).text = text
            findViewById<Button>(R.id.tooltip_hide).setOnClickListener {

it.setTextColor(context.getColor(R.color.tooltip_bnt_clicked_text_color))
                hide()
            }
        }
        layout.addView(view)
        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) {
    private var isDisplayed = false
    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

    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

    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 =
inflateBottomView(R.layout.table_default_bottom_view) {

findViewById<Button>(R.id.files_dialog_btn_hide).setOnClickListener {
                onHideTableListener?.invoke()
            }
        }

        selectBottomView =
inflateBottomView(R.layout.table_select_bottom_view) {

findViewById<Button>(R.id.files_dialog_btn_open).setOnClickListener {
                cancelRows(selectedRowsIndices.toList())
            }

findViewById<Button>(R.id.files_dialog_btn_delete).setOnClickListener {
                deleteRows(selectedRowsIndices.toList())
                onDeleteRowsListener?.invoke(selectedRowsIndices.toList())
            }
        }

        bottomView.addView(defaultBottomView)
    }

    private fun inflateBottomView(layoutId: Int, init: View.() -> Unit):
LinearLayout {
        return LinearLayout(context).apply {
            layoutInflater.inflate(layoutId, this, true)
            init()
        }
    }

    fun addRow(serializedShape: String) {
        val data = serializedShape.dropLast(1).split("\t")
        val row = TableRow(context).apply {
            layoutInflater.inflate(R.layout.table_row, this, true)
            findViewById<TextView>(R.id.table_shape_name).text = data[0]
            findViewById<TextView>(R.id.table_x1).text = data[1]
            findViewById<TextView>(R.id.table_y1).text = data[2]
            findViewById<TextView>(R.id.table_x2).text = data[3]
            findViewById<TextView>(R.id.table_y2).text = data[4]
            setOnClickListener {
toggleRowSelection(tableLayout.indexOfChild(this)) }
        }
        tableLayout.addView(row)
        tableLayout.children.firstOrNull()?.let { if (it is TextView)
tableLayout.removeView(it) }
        updateRowBgColor(row, tableLayout.indexOfChild(row))
        scrollView.scrollToDescendant(row)
    }

    private fun toggleRowSelection(index: Int) {
        if (selectedRowsIndices.contains(index)) {
            cancelRows(listOf(index))
        } else {
            selectRow(index)
        }
    }
```

```
    private fun selectRow(index: Int) {
        if (selectedRowsIndices.isEmpty())
switchBottomView(defaultBottomView, selectBottomView)
        selectedRowsIndices.add(index)
        updateRowBgColor(tableLayout.getChildAt(index), index, isSelected =
true)
        onSelectRowListener?.invoke(index)
    }

    private fun cancelRows(indices: List<Int>) {
        indices.forEach { index ->
            selectedRowsIndices.remove(index)
            updateRowBgColor(tableLayout.getChildAt(index), index)
        }
        if (selectedRowsIndices.isEmpty()) switchBottomView(selectBottomView,
defaultBottomView)
        onCancelRowsListener?.invoke(indices)
    }

    fun deleteRows(indices: List<Int>) {
        indices.sortedDescending().forEach {
            selectedRowsIndices.remove(it)
            tableLayout.removeViewAt(it)
        }
        refreshTableAfterDeletion()
    }

    private fun refreshTableAfterDeletion() {
        if (tableLayout.childCount == 0) {
            addEmptyTableMessage()
        } else {
            tableLayout.children.forEachIndexed { index, row ->
updateRowBgColor(row, index) }
        }
        if (selectedRowsIndices.isEmpty()) switchBottomView(selectBottomView,
defaultBottomView)
    }

    private fun addEmptyTableMessage() {
        tableLayout.addView(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
        })
    }

    private fun updateRowBgColor(row: View, index: Int, isSelected: Boolean =
false) {
        row.setBackgroundColor(
            requireActivity().getColor(
                if (isSelected) {
                    if (index % 2 == 0) R.color.table_selected_row_bg_color_1
else R.color.table_selected_row_bg_color_2
                } else {
                    if (index % 2 == 0) R.color.table_default_row_bg_color_1
else R.color.table_default_row_bg_color_2
                }
            )
```

```
        )
    }

    private fun switchBottomView(from: View, to: View) {
        bottomView.apply {
            removeView(from)
            addView(to)
        }
    }

    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.*

class FileManager(private val context: Context) {
    private val fileExtension: String by lazy {
context.getString(R.string.file_extension) }
    private val drawingsDir: File by lazy {
        File(context.getExternalFilesDir(null),
context.getString(R.string.drawings_dir_name)).apply { if (!exists())
mkdirs() }
    }
    private lateinit var currentFile: File

    private val createFileDialog = CreateFileDialog()
    private val 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 defaultFileName = getDefaultFileName()
        currentFile = File(drawingsDir, defaultFileName)
        startListener(defaultFileName)

        createFileDialog.setFileCreationListeners({}, ::createFile)
        filesDialog.setOnFileListeners(::openFile, ::deleteFile)
    }

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

```
    }

    fun save() {
        FileWriter(currentFile).use {
            it.write(onSaveFileListener())
        }
        Tooltip(context).create("Малюнок збережено у файлі
${currentFile.name}").display()
    }

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

    private fun getShortFileName(fileName: String) =
fileName.removeSuffix(fileExtension)

    private fun getShortFileNames() = drawingsDir.list()?.map {
getShortFileName(it) } ?: emptyList()

    private fun getDefaultFileName(): String {
        val nameStart = context.getString(R.string.default_short_file_name)
        var index = 1
        val shortFileNames = getShortFileNames()
        while ("$nameStart$index" in shortFileNames) index++
        return "$nameStart$index$fileExtension"
    }

    private fun openFile(fileName: String) {
        currentFile = File(drawingsDir, fileName)
        val content = currentFile.readText()
        onOpenFileListener(fileName, content)
    }

    private fun createFile(shortFileName: String): Pair<Boolean, String> {
        if (shortFileName.isBlank()) return false to "Порожнє ім'я"
        if (getShortFileNames().contains(shortFileName)) return false to
"Використане ім'я"

        val fileName = "$shortFileName$fileExtension"
        currentFile = File(drawingsDir, fileName)
        FileWriter(currentFile).use {
it.write(onCreateFileListener(fileName)) }
        return true to "Малюнок збережено у файлі $fileName"
    }

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

    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 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).apply {
            hint = this@CreateFileDialog.hint
        }

        view.findViewById<Button>(R.id.files_dialog_btn_open).apply {
            setOnClickListener {
                editText.text.clear()
                onCancelListener()
                dismiss()
            }
        }

        view.findViewById<Button>(R.id.files_dialog_btn_delete).apply {
            setOnClickListener {
                val text = editText.text.toString().also {
editText.text.clear() }
                val (isValid, message) = onConfirmListener(text)
                if (isValid) {
                    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>) {
        onCancelListener = cancelListener
        onConfirmListener = confirmListener
    }
}
```

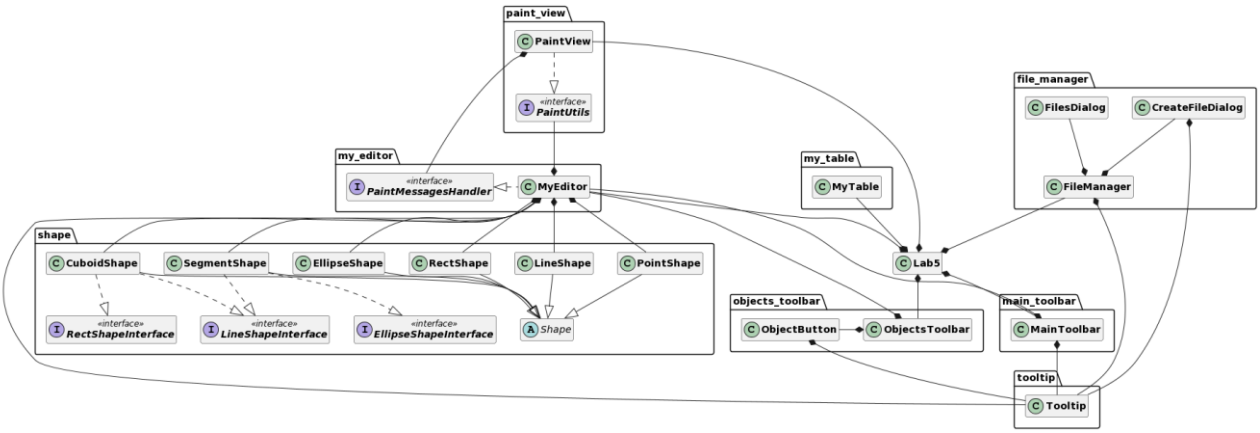
FilesDialog.kt

```
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.*
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 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 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).apply
{
            removeAllViews()
            if (currentFileList.isNotEmpty()) {
                currentFileList.forEach { file ->
                    val row = TableRow(context).apply {
                        layoutInflater.inflate(R.layout.files_dialog_row,
this, true)

findViewById<TextView>(R.id.files_dialog_row_name).text = file
                        setOnClickListener { handleRowClick(this, file) }
                    }
                    addView(row)
                }
            } else {
                onEmptyDir()
```

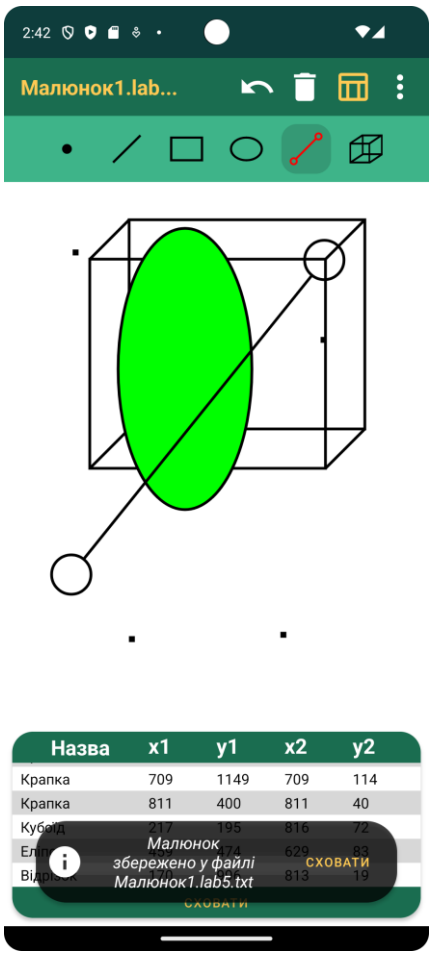
```
            }
        }

        setupBottomView(view)
    }

    private fun handleRowClick(row: TableRow, file: String) {
        selectedRow.view?.let { cancelRow() }
        selectRow(row, file)
    }

    private fun setupBottomView(view: View) {
        bottomView = view.findViewById(R.id.files_dialog_bottom_view)
        val defaultBottomView =
createBottomView(R.layout.files_dialog_default_bottom_view).apply {

findViewById<Button>(R.id.files_dialog_btn_hide).setOnClickListener {
dismiss() }
        }
        val selectBottomView =
createBottomView(R.layout.files_dialog_select_bottom_view).apply {

findViewById<Button>(R.id.files_dialog_btn_open).setOnClickListener {
                onOpenListener(selectedRow.fileName!!)
                cancelRow()
                dismiss()
            }

findViewById<Button>(R.id.files_dialog_btn_delete).setOnClickListener {
deleteRow() }
        }

        bottomView.addView(defaultBottomView)
    }

    private fun createBottomView(layoutRes: Int): LinearLayout {
        return LinearLayout(context).apply {
            layoutInflater.inflate(layoutRes, this, true)
        }
    }

    private fun selectRow(row: TableRow, file: String) {
        bottomView.removeAllViews()

bottomView.addView(createBottomView(R.layout.files_dialog_select_bottom_view)
)

row.setBackgroundColor(requireActivity().getColor(R.color.files_dialog_select
ed_row_bg_color))
        selectedRow.view = row
        selectedRow.fileName = file
    }

    private fun cancelRow() {
        bottomView.removeAllViews()

bottomView.addView(createBottomView(R.layout.files_dialog_default_bottom_view
))

selectedRow.view?.setBackgroundColor(requireActivity().getColor(R.color.files
_dialog_default_row_bg_color))
        selectedRow = object { var view: TableRow? = null; var fileName:
String? = null }
    }
```

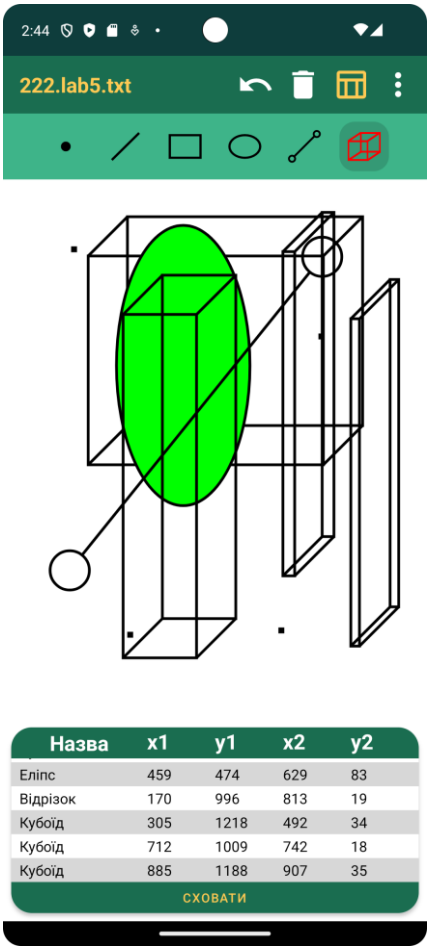
```
    private fun deleteRow() {
        bottomView.removeAllViews()

bottomView.addView(createBottomView(R.layout.files_dialog_default_bottom_view
))
        currentFileList.remove(selectedRow.fileName)
        tableLayout.removeView(selectedRow.view)
        onDeleteListener(selectedRow.fileName!!)
        selectedRow = object { var view: TableRow? = null; var fileName:
String? = null }
        if (currentFileList.isEmpty()) onEmptyDir()
    }

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

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

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

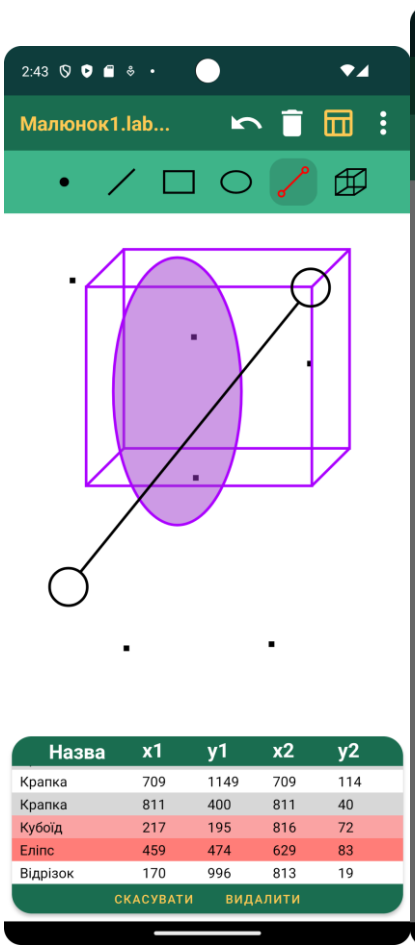
Діаграма класів програми

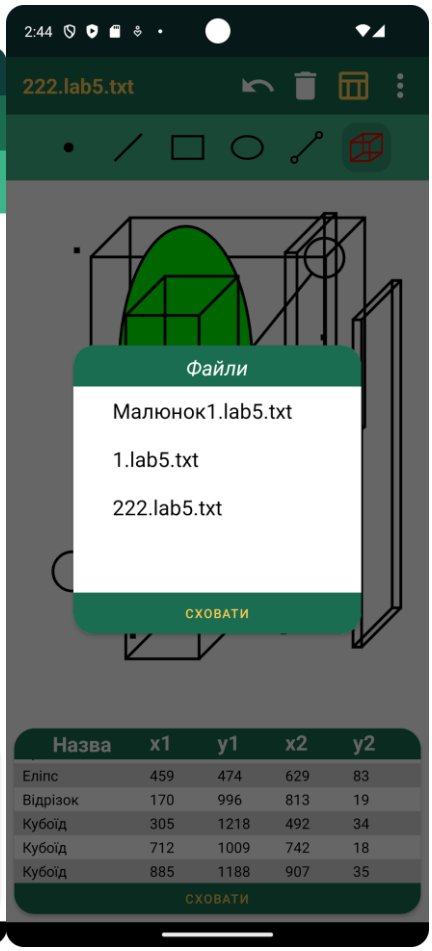


Ілюстрації виконання програми


2:42
Малюнок1.lab...
Назва x1 y1 x2 y2
Крапка 709 1149 709 114
Крапка 811 400 811 40
Малюнок збережено у файлі Малюнок1.lab5.txt
СХОВАТИ


2:44
222.lab5.txt
Назва x1 y1 x2 y2
Еліпс 459 474 629 83
Відрізок 170 996 813 19
Кубоїд 305 1218 492 34
Кубоїд 712 1009 742 18
Кубоїд 885 1188 907 35
СХОВАТИ


2:43
Малюнок1.lab...
Назва x1 y1 x2 y2
Крапка 709 1149 709 114
Крапка 811 400 811 40
Кубоїд 217 195 816 72
Еліпс 459 474 629 83
Відрізок 170 996 813 19
СКАСУВАТИ
ВИДАЛИТИ


2:44
222.lab5.txt
Файли
Малюнок1.lab5.txt
1.lab5.txt
222.lab5.txt
СХОВАТИ
Назва x1 y1 x2 y2
Еліпс 459 474 629 83
Відрізок 170 996 813 19
Кубоїд 305 1218 492 34
Кубоїд 712 1009 742 18
Кубоїд 885 1188 907 35
СХОВАТИ

Висновки

В процесі виконання цієї лабораторної роботи я створив багатовіконний інтерфейс для раніше розробленого графічного редактора на мові програмування Kotlin, який функціонує на платформі Android. Серед нововведень - вікно таблиці, яке дозволяє зручно керувати намальованими фігурами. Крім того, я додав можливість зберігати малюнки у файли формату .txt, впровадивши базові операції для роботи з файлами, такі як створення, читання, оновлення та видалення. Останнє нововведення стосується використання шаблону Singleton, який унеможливлює одночасне створення кількох екземплярів класу MyEditor.