Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №4

з дисципліни
«Об’єктно орієнтоване програмування»
на тему
“Вдосконалення структури коду графічного редактора об’єктів на C++”

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

Перевірив:
Порєв В.М.

Київ 2023

Мета

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

Завдання

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

Умови завдання за варіантом (Ж = 13):

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

Вихідні тексти файлів програми

Lab4.kt

```kotlin
package com.oop.lab4

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

import com.oop.lab4.shape.Shape
import com.oop.lab4.my_editor.MyEditor
import com.oop.lab4.paint_view.PaintView
import com.oop.lab4.main_toolbar.MainToolbar
import com.oop.lab4.objects_toolbar.ObjectsToolbar


class Lab4 : AppCompatActivity() {
    private lateinit var editor: MyEditor
    private lateinit var mainToolbar: MainToolbar
    private lateinit var objectsToolbar: ObjectsToolbar


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

        editor = MyEditor(this)

        mainToolbar = findViewById(R.id.main_toolbar)
        mainToolbar.onCreate(editor)
        mainToolbar.setObjListeners(::onObjSelect, ::onObjCancel)

        objectsToolbar = findViewById(R.id.objects_toolbar)
        objectsToolbar.onCreate(editor)
        objectsToolbar.setObjListeners(::onObjSelect, ::onObjCancel)

        val paintView = findViewById<PaintView>(R.id.paint_view)
        paintView.handler = editor
        editor.paintUtils = paintView
    }

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

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

PaintUtils.kt

```
package com.oop.lab4.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.lab4.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.lab4.shape_editor.PaintMessagesHandler

class PaintView(context: Context, attrs: AttributeSet?):
    View(context, attrs),
```

```kotlin
    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

```kotlin
package com.oop.lab4.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.lab4.my_editor

import android.content.Context
import com.oop.lab4.paint_view.PaintUtils
import com.oop.lab4.shape.Shape
import com.oop.lab4.shape.PointShape
import com.oop.lab4.shape.LineShape
import com.oop.lab4.shape.RectShape
import com.oop.lab4.shape.EllipseShape
import com.oop.lab4.shape.SegmentShape
import com.oop.lab4.shape.CuboidShape

class MyEditor(context: Context): PaintMessagesHandler {
    lateinit var paintUtils: PaintUtils
    override var isRubberTraceModeOn = false

    val shapes = arrayOf(
        PointShape(context),
        LineShape(context),
        RectShape(context),
        EllipseShape(context),
        SegmentShape(context),
        CuboidShape(context),
    )
    var currentShape: Shape? = null
        private set
    private val drawnShapes = mutableListOf<Shape>()

    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)
        }
    }
```

```kotlin
    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()) {
                drawnShapes.add(it)
            }
            paintUtils.repaint()
            it.getInstance()
        }
    }

    override fun onPaint() {
        paintUtils.clearCanvas(paintUtils.rubberTraceCanvas)
        paintUtils.clearCanvas(paintUtils.drawnShapesCanvas)
        drawnShapes.forEach {
            it.showDefault(paintUtils.drawnShapesCanvas)
        }
    }

    fun undo() {
        if (drawnShapes.size > 0) {
            drawnShapes.removeLast()
            paintUtils.repaint()
        }
    }

    fun clearAll() {
        if (drawnShapes.size > 0) {
            drawnShapes.clear()
            paintUtils.repaint()
        }
    }
}
```

Shape.kt

```kotlin
package com.oop.lab4.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.DashPathEffect
import android.graphics.Paint
import com.oop.lab4.R
```

```kotlin
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

    protected open fun getOutlinePaint(): Paint {
        return Paint().apply {
            isAntiAlias = true
            style = Paint.Style.STROKE
            strokeWidth = 7F
            color = context.getColor(R.color.black)
        }
    }

    protected open fun getFillingPaint(): Paint {
        return Paint().apply {
            isAntiAlias = true
            style = Paint.Style.FILL
        }
    }

    protected open fun getRubberTracePaint(): Paint {
        val paint = getOutlinePaint()
        paint.color = context.getColor(R.color.dark_blue)
        val dashLen = 30F
        val spaceLen = 15F
        val dashDensity = floatArrayOf(dashLen, spaceLen, dashLen, spaceLen)
        paint.pathEffect = DashPathEffect(dashDensity, 0F)
        return paint
    }

    abstract fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?)

    abstract fun showDefault(canvas: Canvas)

    fun showRubberTrace(canvas: Canvas) {
        show(canvas, getRubberTracePaint(), null)
    }
}
```

PointShape.kt

```
package com.oop.lab4.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import com.oop.lab4.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(), null)
    }
}
```

LineShape.kt

```
package com.oop.lab4.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import com.oop.lab4.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(), null)
    }
}
```

RectShape.kt

```
package com.oop.lab4.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.RectF
import com.oop.lab4.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(), null)
    }
}
```

EllipseShape.kt

```
package com.oop.lab4.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.RectF
import com.oop.lab4.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(): Paint {
        return super.getFillingPaint().apply {
            color = context.getColor(R.color.light_green)
        }
    }

    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(), getFillingPaint())
    }
}
```

LineShapeInterface.kt

```kotlin
package com.oop.lab4.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

```kotlin
package com.oop.lab4.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

```kotlin
package com.oop.lab4.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.lab4.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.PointF
import com.oop.lab4.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(), null)
    }
}
```

CuboidShape.kt

```
package com.oop.lab4.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.PointF
import android.graphics.RectF
import com.oop.lab4.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(), null)
    }
}
```

MainToolbar.kt

```
package com.oop.lab4.main_toolbar

import android.content.Context
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.lab4.R

import com.oop.lab4.my_editor.MyEditor
```

```kotlin
import com.oop.lab4.shape.Shape
import com.oop.lab4.tooltip.Tooltip

class MainToolbar(context: Context, attrs: AttributeSet?):
    Toolbar(context, attrs) {
    private lateinit var optionsMenu: PopupMenu
    private lateinit var fileSubmenu: PopupMenu
    private lateinit var objSubmenu: PopupMenu

    private lateinit var editor: MyEditor
    private lateinit var objSubmenuItems: Array<MenuItem>

    private lateinit var onObjSelectListener: (Shape) -> Unit
    private lateinit var onObjCancelListener: () -> Unit

    private lateinit var currentObjTextView: TextView

    fun onCreate(editor: MyEditor) {
        val btnOptions = findViewById<ImageButton>(R.id.btn_options)
        btnOptions.setOnClickListener {
            optionsMenu.show()
        }
        optionsMenu = createOptionsMenu(btnOptions)
        fileSubmenu = createFileSubmenu(btnOptions)
        objSubmenu = createObjSubmenu(btnOptions)
        this.editor = editor
        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
        }
        currentObjTextView = findViewById(R.id.current_object)
    }

    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 -> {
                    val tooltip = Tooltip(context, attrs = null)
                    val text = "Ви натиснули кнопку\n\"Довідка\""
                    tooltip.create(this, text).show()
                    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.undo -> {
                    editor.undo()
                    true
                }
                R.id.clear_all -> {
                    editor.clearAll()
                    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]
                        onObjSelectListener(shape.getInstance())
                    } else {
                        onObjCancelListener()
                    }
                }
            }
            true
        }
        return popupMenu
    }
    fun setObjListeners(
        onSelectListener: (Shape) -> Unit,
        onCancelListener: () -> Unit
    ) {
        onObjSelectListener = onSelectListener
        onObjCancelListener = onCancelListener
    }

    fun onObjSelect(shape: Shape) {
        currentObjTextView.text = shape.name
        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 onObjCancel() {
        currentObjTextView.text = "Не вибрано"
        editor.currentShape?.let {
            val id = it.associatedIds["objSubmenuItem"]
            val item = objSubmenu.menu.findItem(id!!)
            item.isChecked = false
        }
    }
}
```

ObjectsToolbar.kt

```
package com.oop.lab4.objects_toolbar

import android.content.Context
import android.util.AttributeSet
import androidx.appcompat.widget.Toolbar
import com.oop.lab4.R

import com.oop.lab4.my_editor.MyEditor
import com.oop.lab4.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 onObjSelectListener: (Shape) -> Unit
    private lateinit var onObjCancelListener: () -> 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
    ) {
        onObjSelectListener = onSelectListener
        onObjCancelListener = onCancelListener

        for (index in objButtons.indices) {
            val button = objButtons[index]
            val shape = editor.shapes[index]
            button.onCreate(shape)
            button.setObjListeners(onObjSelectListener, onObjCancelListener)
        }
    }

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

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

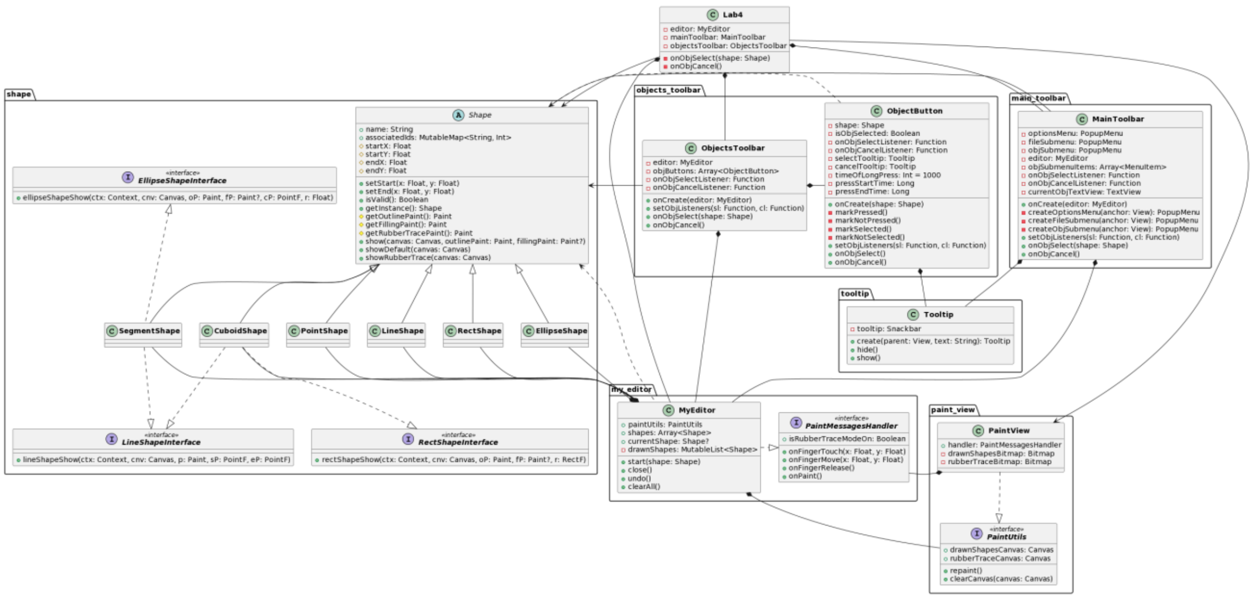
ObjectButton.kt

```kotlin
package com.oop.lab4.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.lab4.R

import com.oop.lab4.shape.Shape
import com.oop.lab4.tooltip.Tooltip

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

    private var isObjSelected = false
    private lateinit var onObjSelectListener: (Shape) -> Unit
    private lateinit var onObjCancelListener: () -> Unit

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

    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(this, selectTooltipText)
        val cancelTooltipText = "Вимкнути режим\nредагування"
        cancelTooltip.create(this, 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 (!isObjSelected) {
            onObjSelectListener(shape.getInstance())
        } else {
            onObjCancelListener()
        }
        return true
    }

    override fun performLongClick(): Boolean {
        super.performLongClick()
        if (!isObjSelected) {
            markNotPressed()
            selectTooltip.show()
        } else {
            markSelected()
            cancelTooltip.show()
        }
        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
    ) {
        onObjSelectListener = onSelectListener
        onObjCancelListener = onCancelListener
    }
    fun onObjSelect() {
        isObjSelected = true
        markSelected()
    }

    fun onObjCancel() {
        isObjSelected = false
        markNotSelected()
    }
}
```

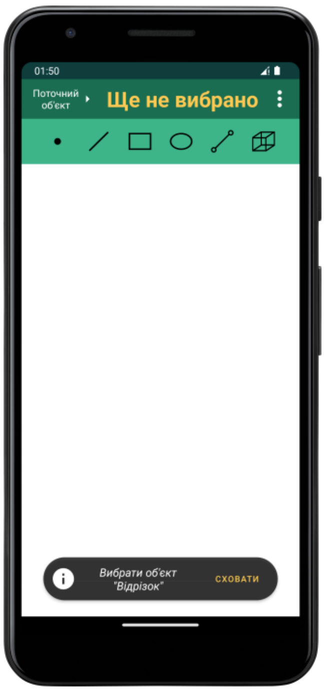
Tooltip.kt

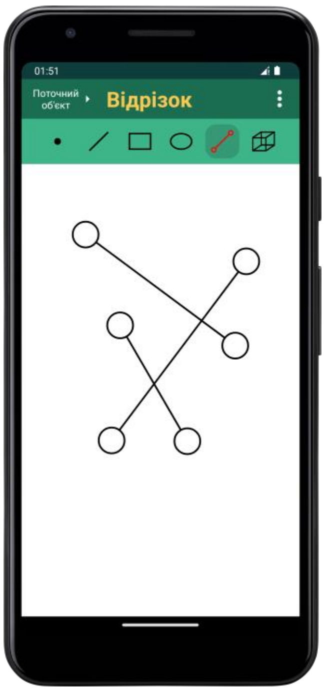
```
package com.oop.lab4.tooltip

import android.content.Context
import android.util.AttributeSet
import android.view.View
import android.widget.Button
import android.widget.TextView
import com.google.android.material.snackbar.Snackbar
import com.oop.lab4.R

class Tooltip(context: Context, attrs: AttributeSet?): View(context, attrs) {
    private lateinit var tooltip: Snackbar

    fun create(parent: View, text: String): Tooltip {
        val displayDuration = Snackbar.LENGTH_LONG
        tooltip = Snackbar.make(parent, "", displayDuration)

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

        val layout = tooltip.view as Snackbar.SnackbarLayout
        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 show() {
        tooltip.show()
    }
}
```

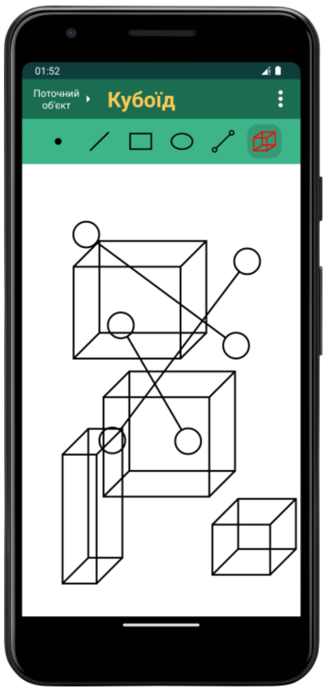
Діаграма класів програми

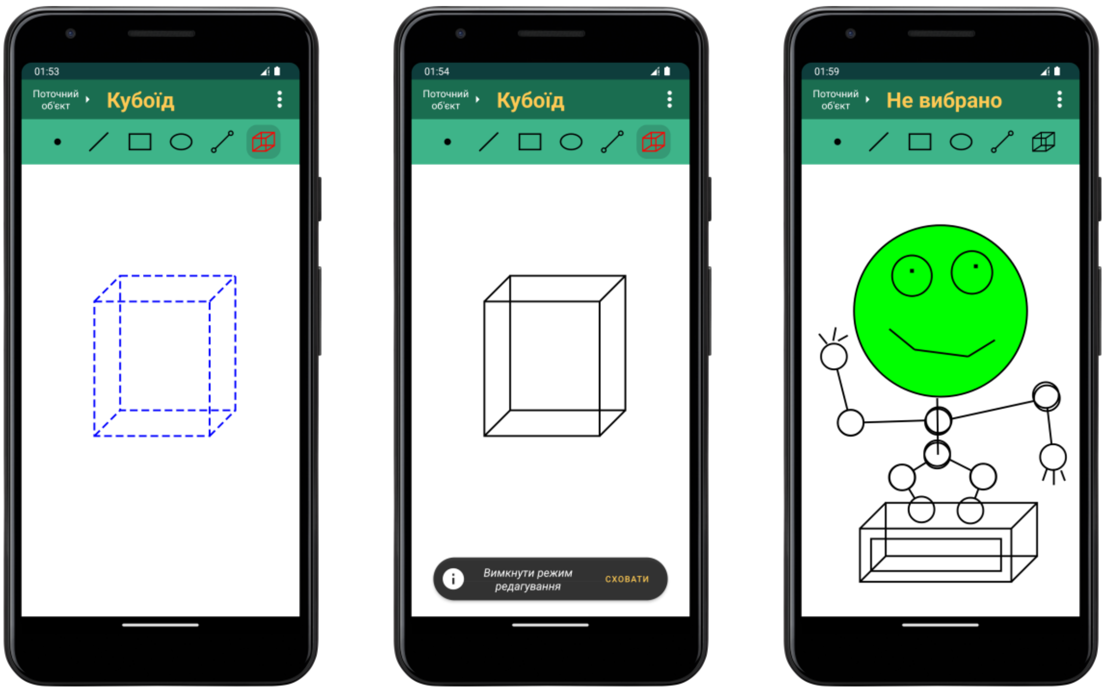

Lab4
editor: MyEditor
mainToolbar: MainToolbar
objectsToolbar: ObjectsToolbar
onObjSelect(shape: Shape)
onObjCancel()
shape
Shape
name: String
associatedIds: MutableMap<String, Int>
startX: Float
startY: Float
endX: Float
endY: Float
setStart(x: Float, y: Float)
setEnd(x: Float, y: Float)
isValid(): Boolean
getInstance(): Shape
getOutlinePaint(): Paint
getFillingPaint(): Paint
getRubberTracePaint(): Paint
show(canvas: Canvas, outlinePaint: Paint, fillingPaint: Paint?)
showDefault(canvas: Canvas)
showRubberTrace(canvas: Canvas)
«interface»
EllipseShapeInterface
ellipseShapeShow(ctx: Context, cnv: Canvas, oP: Paint, fP: Paint?, cP: PointF, r: Float)
SegmentShape
CuboidShape
PointShape
LineShape
RectShape
EllipseShape
«interface»
LineShapeInterface
lineShapeShow(ctx: Context, cnv: Canvas, p: Paint, sP: PointF, eP: PointF)
«interface»
RectShapeInterface
rectShapeShow(ctx: Context, cnv: Canvas, oP: Paint, fP: Paint?, r: RectF)
objects_toolbar
ObjectsToolbar
editor: MyEditor
objButtons: Array<ObjectButton>
onObjSelectListener: Function
onObjCancelListener: Function
onCreate(editor: MyEditor)
setObjListeners(sl: Function, cl: Function)
onObjSelect(shape: Shape)
onObjCancel()
ObjectButton
shape: Shape
isObjSelected: Boolean
onObjSelectListener: Function
onObjCancelListener: Function
selectTooltip: Tooltip
cancelTooltip: Tooltip
timeOfLongPress: Int = 1000
pressStartTime: Long
pressEndTime: Long
onCreate(shape: Shape)
markPressed()
markNotPressed()
markSelected()
markNotSelected()
setObjListeners(sl: Function, cl: Function)
onObjSelect()
onObjCancel()
main_toolbar
MainToolbar
optionsMenu: PopupMenu
fileSubmenu: PopupMenu
objSubmenu: PopupMenu
editor: MyEditor
objSubmenuItems: Array<MenuItem>
onObjSelectListener: Function
onObjCancelListener: Function
currentObjTextView: TextView
onCreate(editor: MyEditor)
createOptionsMenu(anchor: View): PopupMenu
createFileSubmenu(anchor: View): PopupMenu
createObjSubmenu(anchor: View): PopupMenu
setObjListeners(sl: Function, cl: Function)
onObjSelect(shape: Shape)
onObjCancel()
tooltip
Tooltip
tooltip: Snackbar
create(parent: View, text: String): Tooltip
hide()
show()
my_editor
MyEditor
paintUtils: PaintUtils
shapes: Array<Shape>
currentShape: Shape?
drawnShapes: MutableList<Shape>
start(shape: Shape)
close()
undo()
clearAll()
«interface»
PaintMessagesHandler
isRubberTraceModeOn: Boolean
onFingerTouch(x: Float, y: Float)
onFingerMove(x: Float, y: Float)
onFingerRelease()
onPaint()
paint_view
PaintView
handler: PaintMessagesHandler
drawnShapesBitmap: Bitmap
rubberTraceBitmap: Bitmap
«interface»
PaintUtils
drawnShapesCanvas: Canvas
rubberTraceCanvas: Canvas
repaint()
clearCanvas(canvas: Canvas)

Ілюстрації виконання програми









Висновки

Під час виконання цієї лабораторної роботи я удосконалив код раніше створеного графічного редактора для платформи ***Android*** за допомогою шаблонів та практик об'єктно-орієнтованого програмування. Внесені зміни забезпечать зручне додавання нових типів об'єктів. Наочним доказом цього слугують уже додані об'єкти *"Відрізок"* та *"Кубоїд"*. Це стало можливим завдяки об'єднанню усіх редакторів об'єктів в один клас ***MyEditor.***