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
Максимовський Назар Русланович
номер у списку групи: 13

Перевірив:
Порєв В.М.

Київ 2024

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

```
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)
        setupToolbars()
        setupPaintView()
    }

    private fun setupToolbars() {
        mainToolbar = findViewById(R.id.main_toolbar)
        objectsToolbar = findViewById(R.id.objects_toolbar)

        mainToolbar.onCreate(editor)
        objectsToolbar.onCreate(editor)

        mainToolbar.setObjListeners(::onObjSelect, ::onObjCancel)
        objectsToolbar.setObjListeners(::onObjSelect, ::onObjCancel)
```

```
    }

    private fun setupPaintView() {
        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),
    PaintUtils {

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

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

```
    lateinit var handler: PaintMessagesHandler

    override fun onSizeChanged(w: Int, h: Int, oldw: Int, oldh: Int) {
        super.onSizeChanged(w, h, oldw, oldh)
        initBitmaps(w, h)
    }

    private fun initBitmaps(w: Int, h: Int) {
        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)
        canvas.drawBitmap(drawnShapesBitmap, 0F, 0F, null)
        if (handler.isRubberTraceModeOn) {
            canvas.drawBitmap(rubberTraceBitmap, 0F, 0F, null)
        } else {
            handler.onPaint()
        }
    }

    override fun onTouchEvent(event: MotionEvent): Boolean {
        super.onTouchEvent(event)
        when (event.action) {
            MotionEvent.ACTION_DOWN -> handler.onFingerTouch(event.x,
event.y)
            MotionEvent.ACTION_MOVE -> handler.onFingerMove(event.x, event.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.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.*

class MyEditor(context: Context) : PaintMessagesHandler {

    override var isRubberTraceModeOn = false

    lateinit var paintUtils: PaintUtils
    private val drawnShapes = mutableListOf<Shape>()

    private val shapes = arrayOf(
        PointShape(context),
        LineShape(context),
        RectShape(context),
        EllipseShape(context),
        SegmentShape(context),
        CuboidShape(context)
    )

    var currentShape: Shape? = null
        private set

    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?.let {
            isRubberTraceModeOn = false
            if (it.isValid()) drawnShapes.add(it)
            paintUtils.repaint()
            currentShape = it.getInstance()
        }
    }

    override fun onPaint() {
        paintUtils.clearCanvas(paintUtils.rubberTraceCanvas)
```

```
        paintUtils.clearCanvas(paintUtils.drawnShapesCanvas)
        drawnShapes.forEach { it.showDefault(paintUtils.drawnShapesCanvas) }
    }

    fun undo() {
        if (drawnShapes.isNotEmpty()) {
            drawnShapes.removeLast()
            paintUtils.repaint()
        }
    }

    fun clearAll() {
        if (drawnShapes.isNotEmpty()) {
            drawnShapes.clear()
            paintUtils.repaint()
        }
    }
}
```

Shape.kt

```
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

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

    protected var startX = 0F
    protected var startY = 0F
    protected var endX = 0F
    protected var endY = 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
    abstract fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?)
    abstract fun showDefault(canvas: Canvas)

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

    protected open fun getOutlinePaint() = Paint().apply {
        isAntiAlias = true
        style = Paint.Style.STROKE
```

```
        strokeWidth = 7F
        color = context.getColor(R.color.black)
    }

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

    protected open fun getRubberTracePaint(): Paint = getOutlinePaint().apply
{
        color = context.getColor(R.color.dark_blue)
        pathEffect = DashPathEffect(floatArrayOf(30F, 15F), 0F)
    }
}
```

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() = true

    override fun getInstance() = PointShape(context).also {
        it.associatedIds.putAll(this.associatedIds)
    }

    override fun getOutlinePaint() = super.getOutlinePaint().apply {
        strokeWidth = 15F
    }

    override fun getRubberTracePaint() = 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

```
// LineShape.kt
package com.oop.lab4.shape

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

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

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

    override fun getInstance() = 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(), null)
    }
}
```

RectShape.kt

```
package com.oop.lab4.shape

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

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

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

    override fun getInstance() = RectShape(context).also {
        it.associatedIds.putAll(this.associatedIds)
    }

    override fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?) {
        val rect = RectF(startX, startY, endX, endY).apply { sort() }
        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

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

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

```
    override fun getInstance() = EllipseShape(context).also {
        it.associatedIds.putAll(this.associatedIds)
    }

    override fun getFillingPaint() = super.getFillingPaint().apply {
        color = context.getColor(R.color.light_green)
    }

    override fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?) {
        val rect = RectF(startX, startY, 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

```
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

```
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

```
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

```
// SegmentShape.kt
package com.oop.lab4.shape

import android.content.Context
import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.PointF
import kotlin.math.acos
import kotlin.math.cos
import kotlin.math.sin
import kotlin.math.sqrt

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

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

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

    override fun show(canvas: Canvas, outlinePaint: Paint, fillingPaint:
Paint?) {
        val ellipseRadius = 50F
        val distance = sqrt((endX - startX).pow(2) + (endY - startY).pow(2))
        val angle = acos((endX - startX) / distance)
        val offset = PointF(ellipseRadius * cos(angle), ellipseRadius *
sin(angle))

        // Draw main segment
        canvas.drawLine(startX + offset.x, startY + offset.y, endX -
offset.x, endY - offset.y, outlinePaint)

        // Draw ellipses
```

```
        canvas.drawCircle(startX, startY, ellipseRadius, outlinePaint)
        canvas.drawCircle(endX, endY, ellipseRadius, outlinePaint)
    }

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

CuboidShape.kt

```
package com.oop.lab4.shape

import android.content.Context
import android.graphics.*
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() = startX != endX || startY != endY

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

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

        rectShapeShow(context, canvas, outlinePaint, null, frontRect)
        rectShapeShow(context, canvas, outlinePaint, null, backRect)

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

    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.*
import android.widget.*
```

```
import androidx.appcompat.widget.Toolbar
import com.oop.lab4.R
import com.oop.lab4.my_editor.MyEditor
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) {
        this.editor = editor
        val btnOptions = findViewById<ImageButton>(R.id.btn_options)
        currentObjTextView = findViewById(R.id.current_object)

        optionsMenu = createMenu(R.menu.main_toolbar_options_menu,
btnOptions) {
            when (it.itemId) {
                R.id.file -> fileSubmenu.show()
                R.id.objects -> objSubmenu.show()
                R.id.info -> Tooltip(context, null).create(this, "Ви
натиснули кнопку\n\"Довідка\"").show()
            }
        }

        fileSubmenu = createMenu(R.menu.main_toolbar_file_submenu,
btnOptions) {
            when (it.itemId) {
                R.id.undo -> editor.undo()
                R.id.clear_all -> editor.clearAll()
            }
        }

        objSubmenu = createMenu(R.menu.main_toolbar_objects_submenu,
btnOptions) { clickedItem ->
            objSubmenuItems.forEachIndexed { index, item ->
                if (item == clickedItem) {
                    val shape = editor.shapes[index]
                    if (!item.isChecked)
onObjSelectListener(shape.getInstance()) else onObjCancelListener()
                }
            }
        }

        objSubmenuItems = objSubmenu.menu.let { menu ->
            editor.shapes.mapIndexed { index, shape ->
                menu.findItem(R.id.item_point + index).also {
                    shape.associatedIds["objSubmenuItem"] = it.itemId
                }
            }.toTypedArray()
        }
    }

    private fun createMenu(menuRes: Int, anchor: View, listener: (MenuItem) -
```

```
> Unit): PopupMenu {
        return PopupMenu(context, anchor).apply {
            menuInflater.inflate(menuRes, menu)
            setOnMenuItemClickListener { listener(it); true }
        }
    }

    fun setObjListeners(onSelectListener: (Shape) -> Unit, onCancelListener:
() -> Unit) {
        onObjSelectListener = onSelectListener
        onObjCancelListener = onCancelListener
    }

    fun onObjSelect(shape: Shape) {
        updateObjTextView(shape.name)
        updateSelection(shape.associatedIds["objSubmenuItem"], isSelected =
true)
    }

    fun onObjCancel() {
        updateObjTextView("Не вибрано")

updateSelection(editor.currentShape?.associatedIds?.get("objSubmenuItem"),
isSelected = false)
    }

    private fun updateObjTextView(text: String) {
        currentObjTextView.text = text
    }

    private fun updateSelection(itemId: Int?, isSelected: Boolean) {
        itemId?.let { objSubmenu.menu.findItem(it).isChecked = isSelected }
    }
}
```

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 = editor.shapes.mapIndexed { index, shape ->
            findViewById<ObjectButton>(R.id.btn_point + index).also {
                shape.associatedIds["objButton"] = it.id
            }
        }.toTypedArray()
    }
```

```
    fun setObjListeners(
        onSelectListener: (Shape) -> Unit,
        onCancelListener: () -> Unit
    ) {
        onObjSelectListener = onSelectListener
        onObjCancelListener = onCancelListener

        objButtons.forEachIndexed { index, button ->
            val shape = editor.shapes[index]
            button.apply {
                onCreate(shape)
                setObjListeners(onSelectListener, onCancelListener)
            }
        }
    }

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

    fun onObjCancel() {
        editor.currentShape?.associatedIds?.get("objButton")?.let {
            findViewById<ObjectButton>(it).onObjCancel()
        }
    }
}
```

ObjectButton.kt

```
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
```

```
    fun onCreate(shape: Shape) {
        this.shape = shape
        selectTooltip.create(this, "Вибрати об'єкт\n\"${shape.name}\"")
        cancelTooltip.create(this, "Вимкнути режим\nредагування")
    }

    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 < timeOfLongPress) performClick() else
performLongClick()
                pressStartTime = 0
            }
        }
        return true
    }

    override fun performClick(): Boolean {
        super.performClick()
        if (isObjSelected) onObjCancelListener() else
onObjSelectListener(shape.getInstance())
        return true
    }

    override fun performLongClick(): Boolean {
        super.performLongClick()
        if (isObjSelected) {
            markSelected()
            cancelTooltip.show()
        } else {
            markNotPressed()
            selectTooltip.show()
        }
        return true
    }

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

    private fun markNotPressed() {
        backgroundTintList = context.getColorStateList(R.color.transparent)
    }

    private fun markSelected() {
        backgroundTintList =
context.getColorStateList(R.color.selected_btn_background_color)
        colorFilter = PorterDuffColorFilter(
            context.getColor(R.color.selected btn icon color),
            PorterDuff.Mode.SRC_IN
        )
    }

    private fun markNotSelected() {
        backgroundTintList = context.getColorStateList(R.color.transparent)
        colorFilter = PorterDuffColorFilter(
            context.getColor(R.color.on_objects_toolbar_color),
```

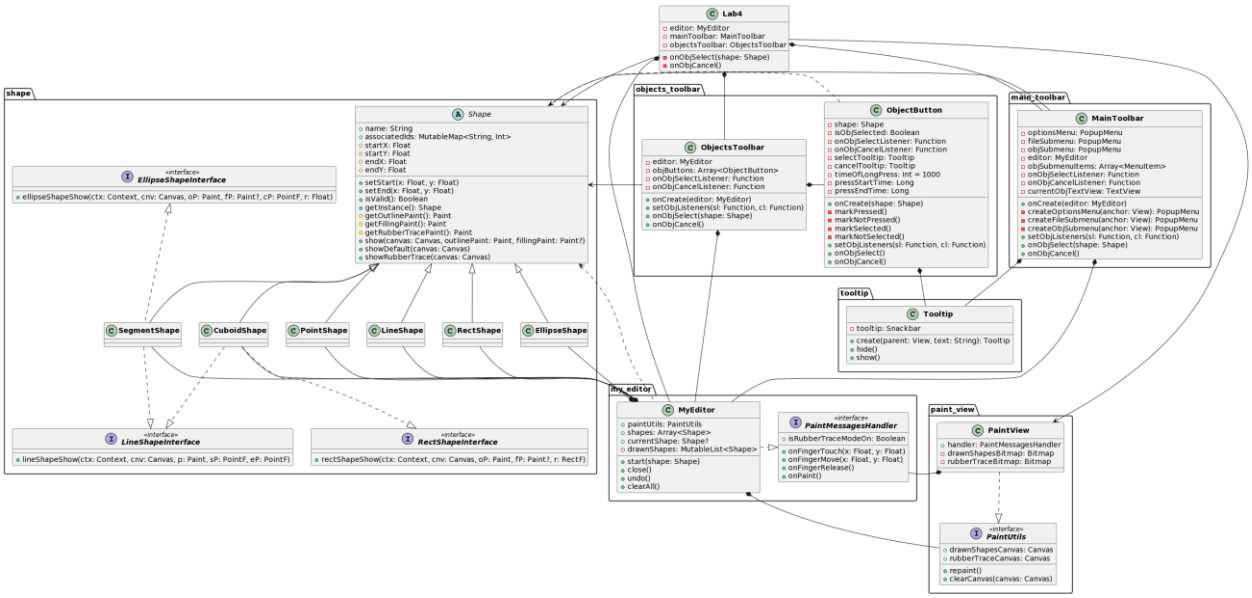
```
            PorterDuff.Mode.SRC_IN
        )
    }

    fun setObjListeners(
        onSelectListener: (Shape) -> Unit,
        onCancelListener: () -> Unit
    ) {
        onObjSelectListener = onSelectListener
        onObjCancelListener = onCancelListener
    }

    fun onObjSelect() {
        isObjSelected = true
        markSelected()
    }

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

Tooltip.kt

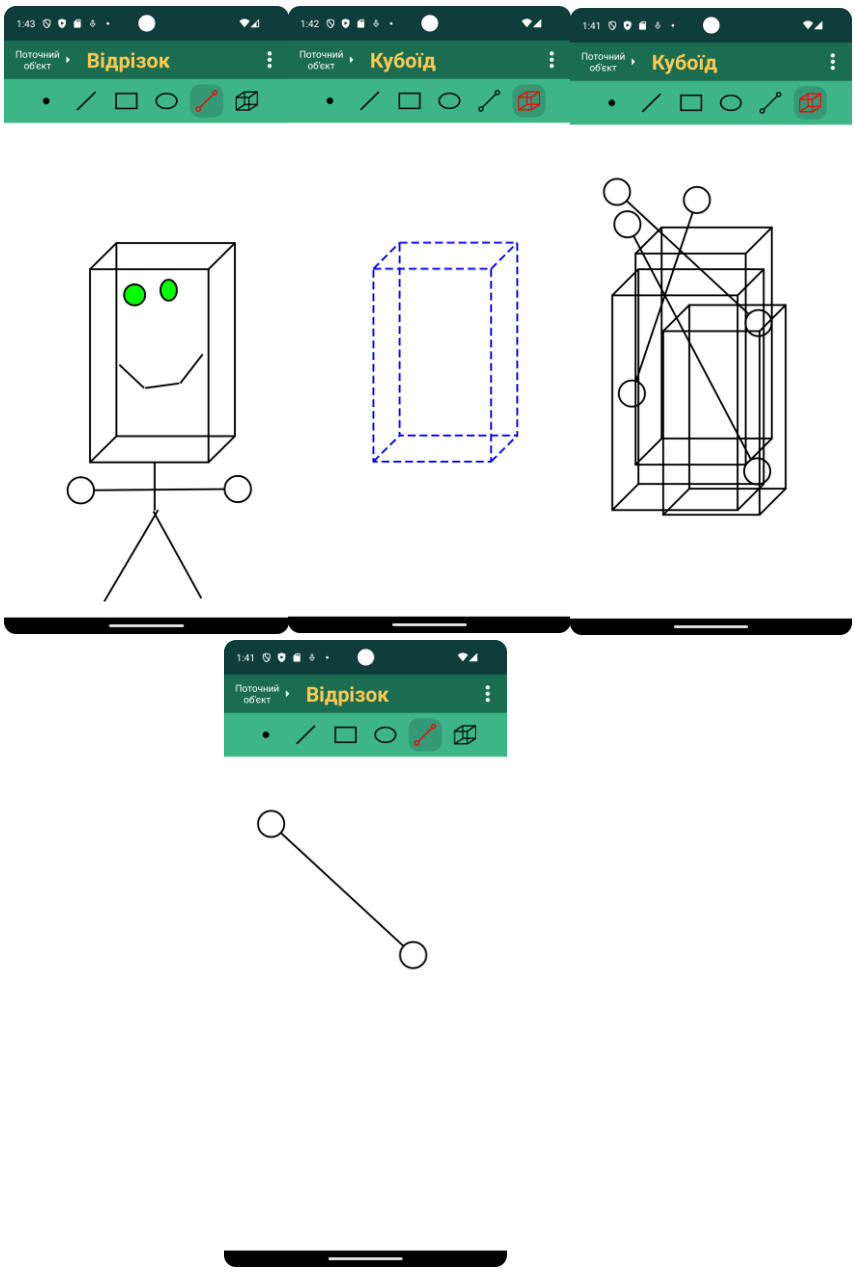
```
package com.oop.lab4.tooltip

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 {
        tooltip = Snackbar.make(parent, "", Snackbar.LENGTH_LONG).apply {
            view.setBackgroundColor(context.getColor(R.color.transparent))
            (view as Snackbar.SnackbarLayout).apply {
                val customView = inflate(context, R.layout.tooltip, null)
                addView(customView)

                customView.findViewById<TextView>(R.id.tooltip_text).text =
text
                customView.findViewById<Button>(R.id.tooltip_hide).apply {
                    setOnClickListener {

setTextColor(context.getColor(R.color.tooltip_bnt_clicked_text_color))
                        hide()
                    }
                }
            }
        }
        return this
    }

    fun hide() {
        tooltip.dismiss()
```

```
    }

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

Діаграма класів програми



Ілюстрації виконання програми



Висновки

Під час цієї лабораторної роботи я покращив код існуючого графічного редактора для Android, застосовуючи шаблони та практики об'єктно-орієнтованого програмування. Внесені зміни полегшують додавання нових типів об'єктів, як, наприклад, вже реалізовані "Відрізок" та "Кубоїд". Це стало можливим завдяки об'єднанню всіх редакторів об'єктів в одному класі MyEditor.