**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 №3**

з дисципліни

«Об’єктно орієнтоване програмування»

на тему

“Розробка інтерфейсу користувача на C++”

Виконав: Перевірив:

Студент групи ІМ-22 Порєв В.М.

Кушнір Микола Миколайович

номер у списку групи: 13

Київ 2023

**Мета**

Отримати вміння та навички використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів С++, запрограмувавши графічний інтерфейс користувача.

**Завдання**

**1.** Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям **Lab3**.

**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 = Жлаб2 + 1 = 13 + 1 = 14):**

* Масив вказівників для динамічних об’єктів типу Shape: статичний масив для Shapeобсягом 114 елементів *(14 mod 3 = 2)*
* "Гумовий" слід при вводі об’єктів: суцільна лінія синього кольору *(14 mod 4 = 2)*
* Увід прямокутника: по двом протилежним кутам *(14 mod 2 = 0)*
* Відображення прямокутника: чорний контур прямокутника без заповнення *(14 mod 5 = 4)*
* Увід еліпса: від центру до одного з кутів охоплюючого прямокутника *(14 mod 2 = 0)*
* Відображення еліпса: чорний контур з кольоровим заповненням *(14 mod 5 = 4)*
* Колір заповнення еліпса: світло-зелений *(14 mod 6 = 2)*
* Позначка поточного типу об’єкту, що вводиться: в меню *(14 mod 2 = 0)*

**Вихідні тексти файлів програми**

**Lab3.kt**

package com.oop.lab3  
  
import android.os.Bundle  
import androidx.appcompat.app.AppCompatActivity  
  
import com.oop.lab3.shape\_editor.ShapeObjectsEditor  
import com.oop.lab3.shape.Shape  
import com.oop.lab3.paint\_view.PaintView  
import com.oop.lab3.main\_toolbar.MainToolbar  
import com.oop.lab3.objects\_toolbar.ObjectsToolbar  
  
class Lab3 : AppCompatActivity() {  
 private lateinit var shapeObjEditor: ShapeObjectsEditor  
 private lateinit var mainToolbar: MainToolbar  
 private lateinit var objectsToolbar: ObjectsToolbar  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.main\_activity)  
  
 shapeObjEditor = ShapeObjectsEditor(this)  
  
 setupToolbar(mainToolbar, shapeObjEditor)  
 setupToolbar(objectsToolbar, shapeObjEditor)  
  
 val paintView = findViewById<PaintView>(R.id.paint\_view).apply {  
 handler = shapeObjEditor  
 shapeObjEditor.paintUtils = this  
 }  
 }  
  
 private fun setupToolbar(toolbar: ObjectsToolbar, editor: ShapeObjectsEditor) {  
 toolbar.onCreate(editor)  
 toolbar.setObjListeners(::onObjSelect, ::onObjCancel)  
 }  
  
 private fun onObjSelect(shape: Shape) {  
 mainToolbar.onObjSelect(shape)  
 objectsToolbar.onObjSelect(shape)  
 shapeObjEditor.startEditor(shape)  
 }  
  
 private fun onObjCancel() {  
 mainToolbar.onObjCancel()  
 objectsToolbar.onObjCancel()  
 shapeObjEditor.closeEditor()  
 }  
}

**PaintUtils.kt**

package com.oop.lab3.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.lab3.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.lab3.shape\_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)  
 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)  
 val (x, y) = event.x to 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.lab3.shape\_editor  
  
interface PaintMessagesHandler {  
 var isRubberTraceModeOn: Boolean  
 fun onFingerTouch(x: Float, y: Float)  
 fun onFingerMove(x: Float, y: Float)  
 fun onFingerRelease()  
 fun onPaint()  
}

**ShapeObjectsEditor.kt**

package com.oop.lab3.shape\_editor  
  
import android.content.Context  
  
import com.oop.lab3.shape.Shape  
import com.oop.lab3.shape.PointShape  
import com.oop.lab3.shape.LineShape  
import com.oop.lab3.shape.RectShape  
import com.oop.lab3.shape.EllipseShape  
  
import com.oop.lab3.editor.ShapeEditor  
import com.oop.lab3.editor.PointShapeEditor  
import com.oop.lab3.editor.LineShapeEditor  
import com.oop.lab3.editor.RectShapeEditor  
import com.oop.lab3.editor.EllipseShapeEditor  
  
import com.oop.lab3.paint\_view.PaintUtils  
  
class ShapeObjectsEditor(context: Context): PaintMessagesHandler {  
 lateinit var paintUtils: PaintUtils  
 override var isRubberTraceModeOn = false  
  
 val shapes = arrayOf(  
 PointShape(context, PointShapeEditor()),  
 LineShape(context, LineShapeEditor()),  
 RectShape(context, RectShapeEditor()),  
 EllipseShape(context, EllipseShapeEditor())  
 )  
 private var currentShape: Shape? = null  
 private val drawnShapes = mutableListOf<Shape>()  
 private var activeEditor: ShapeEditor? = null  
  
 fun startEditor(shape: Shape) {  
 currentShape = shape  
 activeEditor = shape.editor  
 }  
  
 fun closeEditor() {  
 currentShape = null  
 activeEditor = null  
 }  
  
 override fun onFingerTouch(x: Float, y: Float) {  
 activeEditor?.onFingerTouch(x, y)  
 }  
  
 override fun onFingerMove(x: Float, y: Float) {  
 activeEditor?.let {  
 isRubberTraceModeOn = true  
 paintUtils.clearCanvas(paintUtils.rubberTraceCanvas)  
 it.onFingerMove(paintUtils.rubberTraceCanvas, x, y)  
 paintUtils.repaint()  
 }  
 }  
  
 override fun onFingerRelease() {  
 activeEditor?.let {  
 isRubberTraceModeOn = false  
 it.onFingerRelease(drawnShapes)  
 paintUtils.repaint()  
 }  
 }  
  
 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() {  
 drawnShapes.clear()  
 paintUtils.repaint()  
 }  
}

**Shape.kt**

package com.oop.lab3.shape  
  
import android.content.Context  
import android.graphics.Canvas  
import android.graphics.Paint  
import com.oop.lab3.R  
  
import com.oop.lab3.editor.ShapeEditor  
  
abstract class Shape(private val context: Context) {  
 abstract val name: String  
 val associatedIds = mutableMapOf<String, Int>()  
 abstract val editor: ShapeEditor  
  
 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 createPaint(strokeWidth: Float, color: Int): Paint {  
 return Paint().apply {  
 isAntiAlias = true  
 style = Paint.Style.STROKE  
 this.strokeWidth = strokeWidth  
 color = context.getColor(color)  
 }  
 }  
  
 protected open fun getOutlinePaint(): Paint = createPaint(7F, R.color.black)  
  
 protected open fun getFillingPaint(): Paint = createPaint(0F, 0) // Default no color  
  
 protected open fun getRubberTracePaint(): Paint = createPaint(7F, R.color.dark\_blue)  
  
 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.lab3.shape  
  
import android.content.Context  
import android.graphics.Canvas  
import android.graphics.Paint  
import com.oop.lab3.R  
import com.oop.lab3.editor.ShapeEditor  
  
class PointShape(private val context: Context, override val editor: ShapeEditor) : Shape(context) {  
 init {  
 editor.shape = this  
 }  
  
 override val name = context.getString(R.string.point)  
  
 override fun isValid() = true  
  
 override fun getInstance() = PointShape(context, editor).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**

package com.oop.lab3.shape  
  
import android.content.Context  
import android.graphics.Canvas  
import android.graphics.Paint  
import com.oop.lab3.R  
import com.oop.lab3.editor.ShapeEditor  
  
class LineShape(private val context: Context, override val editor: ShapeEditor) : Shape(context) {  
 init {  
 editor.shape = this  
 }  
  
 override val name = context.getString(R.string.line)  
  
 override fun isValid() = startX != endX || startY != endY  
  
 override fun getInstance() = 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.lab3.shape  
  
import android.content.Context  
import android.graphics.Canvas  
import android.graphics.Paint  
import android.graphics.RectF  
import com.oop.lab3.R  
import com.oop.lab3.editor.ShapeEditor  
  
class RectShape(private val context: Context, override val editor: ShapeEditor) : Shape(context) {  
 init {  
 editor.shape = this  
 }  
  
 override val name = context.getString(R.string.rectangle)  
  
 override fun isValid() = startX != endX || startY != endY  
  
 override fun getInstance() = 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.lab3.shape  
  
import android.content.Context  
import android.graphics.Canvas  
import android.graphics.Paint  
import android.graphics.RectF  
import com.oop.lab3.R  
import com.oop.lab3.editor.ShapeEditor  
  
class EllipseShape(private val context: Context, override val editor: ShapeEditor) : Shape(context) {  
 init {  
 editor.shape = this  
 }  
  
 override val name = context.getString(R.string.ellipse)  
  
 override fun isValid() = startX != endX || startY != endY  
  
 override fun getInstance() = EllipseShape(context, editor).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)  
 fillingPaint?.let { canvas.drawOval(rect, it) }  
 canvas.drawOval(rect, outlinePaint)  
 }  
  
 override fun showDefault(canvas: Canvas) {  
 show(canvas, getOutlinePaint(), getFillingPaint())  
 }  
}

**Editor.kt**

package com.oop.lab3.editor  
  
import android.graphics.Canvas  
  
import com.oop.lab2.shape.Shape  
  
abstract class Editor {  
 abstract fun onFingerTouch(x: Float, y: Float)  
  
 abstract fun onFingerMove(canvas: Canvas, x: Float, y: Float)  
  
 abstract fun onFingerRelease(drawnShapes: MutableList<Shape>)  
}

**ShapeEditor.kt**

package com.oop.lab3.editor  
  
import com.oop.lab2.shape.Shape  
  
abstract class ShapeEditor : Editor() {  
  
 lateinit var shape: Shape  
  
 override fun onFingerRelease(drawnShapes: MutableList<Shape>) {  
 shape.takeIf { it.isValid() }?.let(drawnShapes::add)  
 shape = shape.getInstance()  
 }  
}

**PointShapeEditor.kt**

package com.oop.lab3.editor  
  
import android.graphics.Canvas  
  
class PointShapeEditor: ShapeEditor() {  
 override fun onFingerTouch(x: Float, y: Float) {  
 shape.setStart(x, y)  
 }  
  
 override fun onFingerMove(canvas: Canvas, x: Float, y: Float) {  
 shape.showRubberTrace(canvas)  
 }  
}

**LineShapeEditor.kt**

package com.oop.lab3.editor  
  
import android.graphics.Canvas  
  
class LineShapeEditor: ShapeEditor() {  
 override fun onFingerTouch(x: Float, y: Float) {  
 shape.setStart(x, y)  
 shape.setEnd(x, y)  
 }  
  
 override fun onFingerMove(canvas: Canvas, x: Float, y: Float) {  
 shape.setEnd(x, y)  
 shape.showRubberTrace(canvas)  
 }  
}

**RectShapeEditor.kt**

package com.oop.lab3.editor  
  
import android.graphics.Canvas  
  
class RectShapeEditor: ShapeEditor() {  
 override fun onFingerTouch(x: Float, y: Float) {  
 shape.setStart(x, y)  
 shape.setEnd(x, y)  
 }  
  
 override fun onFingerMove(canvas: Canvas, x: Float, y: Float) {  
 shape.setEnd(x, y)  
 shape.showRubberTrace(canvas)  
 }  
}

**EllipseShapeEditor.kt**

package com.oop.lab3.editor  
  
import android.graphics.Canvas  
import android.graphics.PointF  
import android.graphics.RectF  
  
class EllipseShapeEditor : ShapeEditor() {  
  
 private val centerPoint = PointF()  
  
 override fun onFingerTouch(x: Float, y: Float) {  
 centerPoint.set(x, y)  
 shape.setStart(x, y)  
 shape.setEnd(x, y)  
 }  
  
 override fun onFingerMove(canvas: Canvas, x: Float, y: Float) {  
 val rect = RectF(  
 centerPoint.x - (x - centerPoint.x),  
 centerPoint.y - (y - centerPoint.y),  
 x,  
 y  
 ).apply { sort() }  
  
 shape.setStart(rect.left, rect.top)  
 shape.setEnd(rect.right, rect.bottom)  
 shape.showRubberTrace(canvas)  
 }  
}

**MainToolbar.kt**

package com.oop.lab3.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.lab3.R  
  
import com.oop.lab3.shape\_editor.ShapeObjectsEditor  
import com.oop.lab3.shape.Shape  
import com.oop.lab3.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 shapeObjEditor: ShapeObjectsEditor  
 private lateinit var objSubmenuItems: Array<MenuItem>  
  
 private lateinit var onObjSelectListener: (Shape) -> Unit  
 private lateinit var onObjCancelListener: () -> Unit  
  
 private lateinit var currentObjTextView: TextView  
  
 fun onCreate(shapeObjEditor: ShapeObjectsEditor) {  
 val btnOptions = findViewById<ImageButton>(R.id.btn\_options)  
 btnOptions.setOnClickListener {  
 optionsMenu.show()  
 }  
 optionsMenu = createOptionsMenu(btnOptions)  
 fileSubmenu = createFileSubmenu(btnOptions)  
 objSubmenu = createObjSubmenu(btnOptions)  
 this.shapeObjEditor = shapeObjEditor  
 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),  
 )  
 for (index in objSubmenuItems.indices) {  
 val shape = shapeObjEditor.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 -> {  
 shapeObjEditor.undo()  
 true  
 }  
 R.id.clear\_all -> {  
 shapeObjEditor.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 = shapeObjEditor.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  
 shapeObjEditor.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"  
 shapeObjEditor.currentShape?.let {  
 val id = it.associatedIds["objSubmenuItem"]  
 val item = objSubmenu.menu.findItem(id!!)  
 item.isChecked = false  
 }  
 }  
}

**ObjectsToolbar.kt**

package com.oop.lab3.objects\_toolbar  
  
import android.content.Context  
import android.util.AttributeSet  
import androidx.appcompat.widget.Toolbar  
import com.oop.lab3.R  
  
import com.oop.lab3.shape\_editor.ShapeObjectsEditor  
import com.oop.lab3.shape.Shape  
  
class ObjectsToolbar(context: Context, attrs: AttributeSet?):  
 Toolbar(context, attrs) {  
 private lateinit var shapeObjEditor: ShapeObjectsEditor  
 private lateinit var objButtons: Array<ObjectButton>  
  
 private lateinit var onObjSelectListener: (Shape) -> Unit  
 private lateinit var onObjCancelListener: () -> Unit  
  
 fun onCreate(shapeObjEditor: ShapeObjectsEditor) {  
 this.shapeObjEditor = shapeObjEditor  
 objButtons = arrayOf(  
 findViewById(R.id.btn\_point),  
 findViewById(R.id.btn\_line),  
 findViewById(R.id.btn\_rectangle),  
 findViewById(R.id.btn\_ellipse),  
 )  
 for (index in objButtons.indices) {  
 val shape = shapeObjEditor.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 = shapeObjEditor.shapes[index]  
 button.onCreate(shape)  
 button.setObjListeners(onObjSelectListener, onObjCancelListener)  
 }  
 }  
  
 fun onObjSelect(shape: Shape) {  
 shapeObjEditor.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() {  
 shapeObjEditor.currentShape?.let {  
 val id = it.associatedIds["objButton"]  
 val button = findViewById<ObjectButton>(id!!)  
 button.onObjCancel()  
 }  
 }  
}

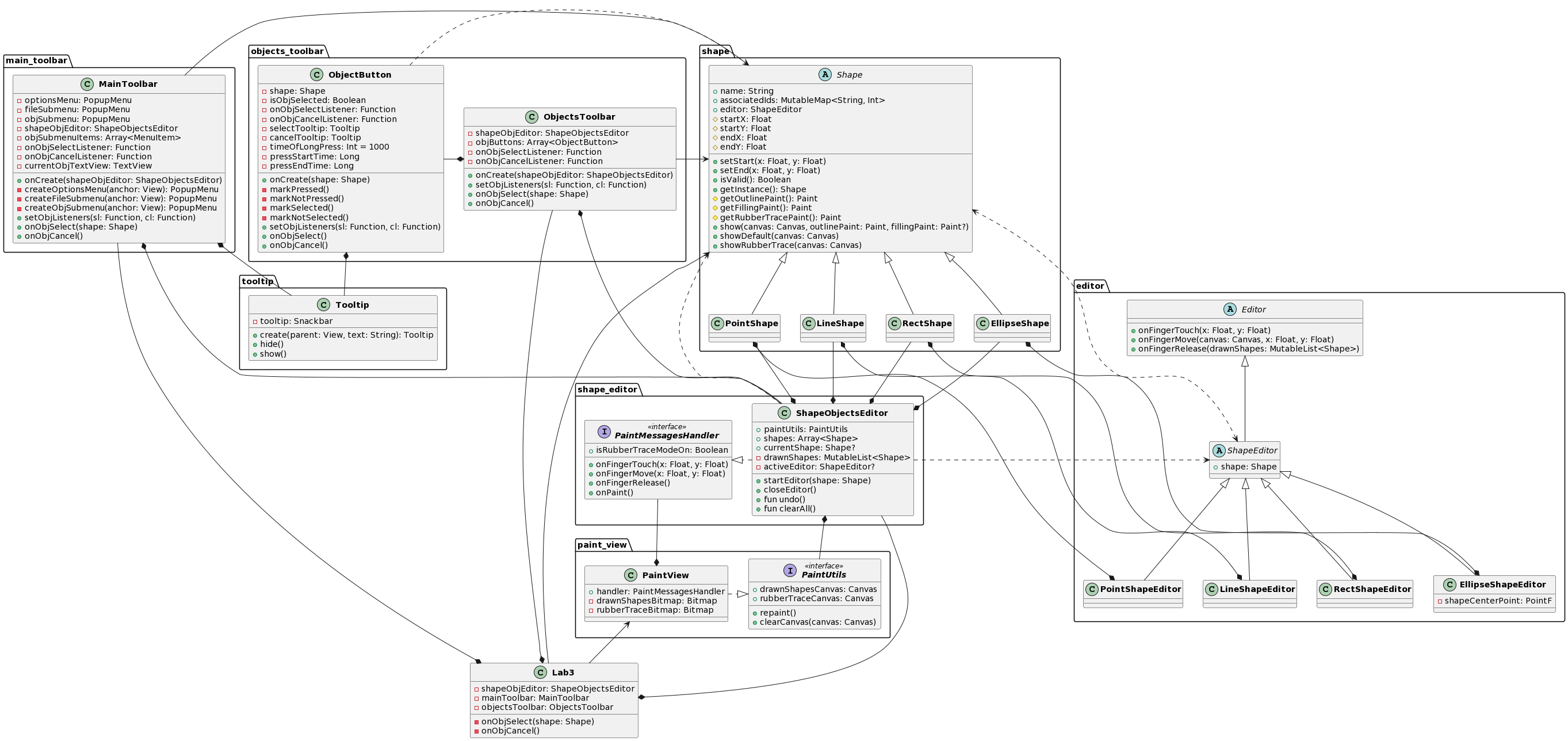
**ObjectButton.kt**

package com.oop.lab3.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.lab3.R  
  
import com.oop.lab3.shape.Shape  
import com.oop.lab3.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()  
 }  
}

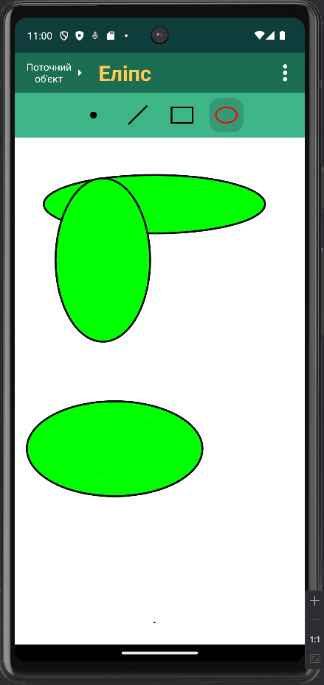
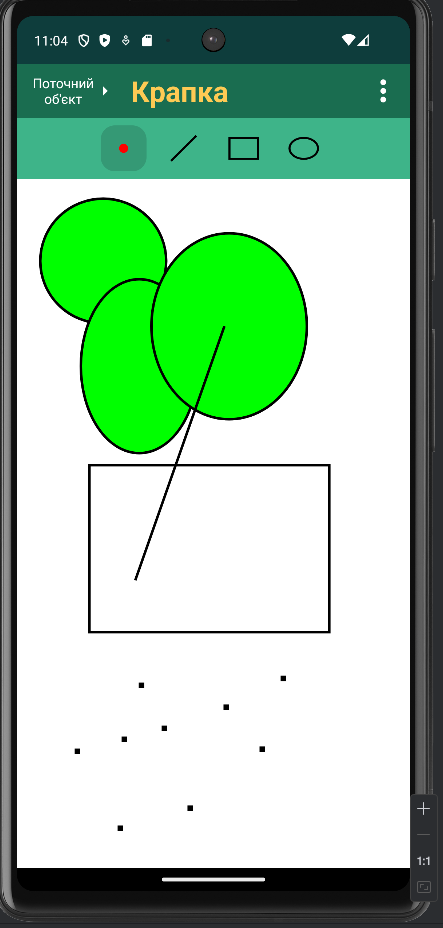
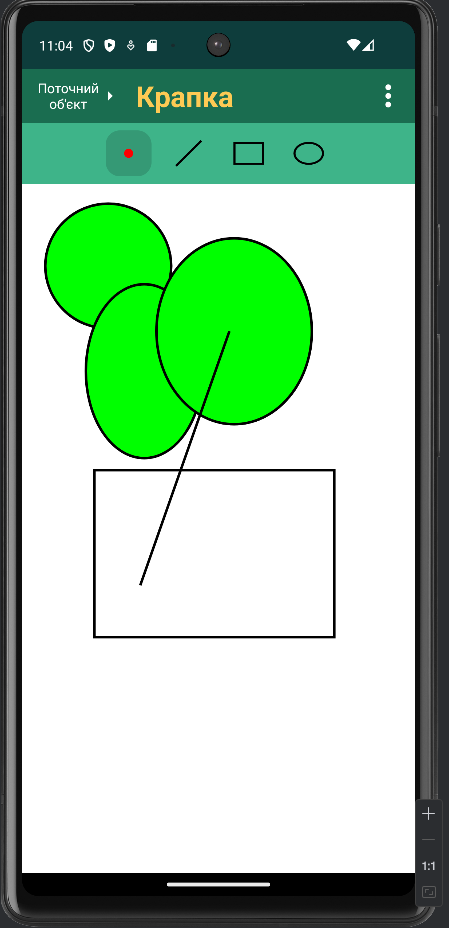
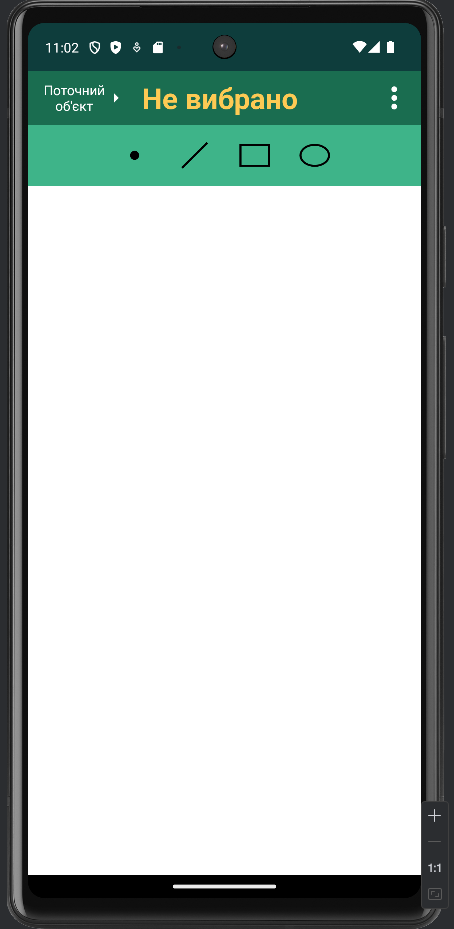
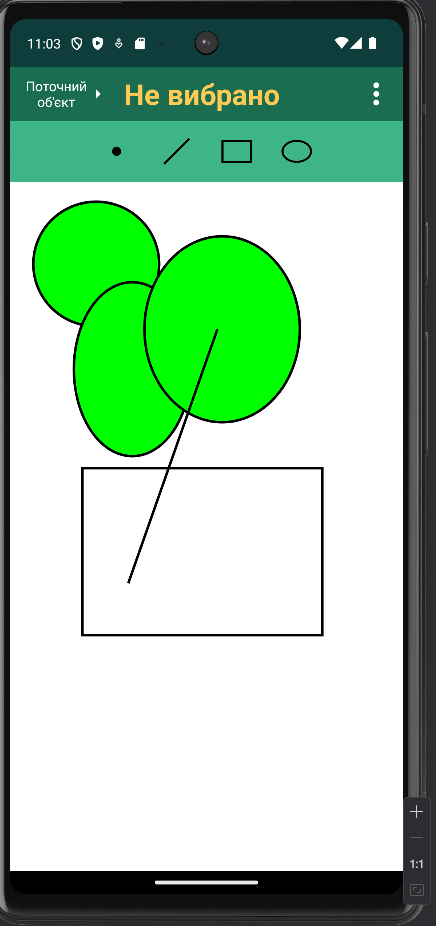
**Tooltip.kt**

package com.oop.lab3.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.lab3.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**



**Ілюстрації виконання програми**

**Висновки**