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 №2

з дисципліни
«Об’єктно орієнтоване програмування»
на тему
“Розробка 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 використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів C++, запрограмувавши простий графічний редактор в об’єктно-орієнтованому стилі.

Завдання

1. Створити у середовищі MS Visual Studio C++ проект типу Windows Desktop Application з ім’ям **Lab2**.
2. Скомпілювати проєкт і отримати виконуваний файл програми.
3. Перевірити роботу програми. Налагодити програму.
4. Проаналізувати та прокоментувати результати та вихідний текст програми.
5. Оформити звіт.

Умови завдання за варіантом (Ж = 13):

- Масив вказівників для динамічних об’єктів типу Shape: статичний масив для Shape обсягом 113 елементів *(13 mod 3 = 1)*
- "Гумовий" слід при вводі об’єктів: суцільна лінія червоного кольору *(13 mod 4 = 1)*
- Увід прямокутника: від центру до одного з кутів *(13 mod 2 = 1)*
- Відображення прямокутника: чорний контур прямокутника без заповнення *(13 mod 5 = 3)*
- Увід еліпса: по двом протилежним кутам охоплюючого прямокутника *(13 mod 2 = 1)*
- Відображення еліпса: чорний контур з кольоровим заповненням *(13 mod 5 = 3)*
- Колір заповнення еліпса: жовтий *(13 mod 6 = 1)*
- Позначка поточного типу об’єкту, що вводиться: в заголовку вікна *(13 mod 2 = 1)*

Вихідні тексти файлів програми

Lab2.kt

```
package com.oop.lab2

import android.os.Bundle
import android.view.Menu
import android.view.MenuItem
import android.view.View
import android.widget.PopupMenu
```

```
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.forEach

import com.oop.lab2.shape_editor.ShapeObjectsEditor
import com.oop.lab2.shape_editor.ShapeObjectsEditorInterface

class Lab2 : AppCompatActivity() {
    private var objectsPopupMenu: PopupMenu? = null
    private val shapeObjectsEditor: ShapeObjectsEditorInterface =
ShapeObjectsEditor()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main_activity)
        val paintView = findViewById<PaintView>(R.id.paint_view)
        paintView.shapeObjectsEditor = shapeObjectsEditor
        shapeObjectsEditor.paintView = paintView
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.actionbar_menu, menu)
        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        return when (item.itemId) {
            R.id.btn_file -> {
                Toast
                    .makeText(this, "Ви натиснули кнопку \"Файл\"",
Toast.LENGTH_SHORT)
                    .show()
                true
            }
            R.id.btn_objects -> {
                if (objectsPopupMenu == null) {
                    val btnObjects = findViewById<View>(R.id.btn_objects)
                    objectsPopupMenu = createObjectsPopupMenu(btnObjects)
                }
                showObjectsPopupMenu()
                true
            }
            R.id.btn_info -> {
                Toast
                    .makeText(this, "Ви натиснули кнопку \"Довідка\"",
Toast.LENGTH_SHORT)
                    .show()
                true
            }
            else -> {
                super.onOptionsItemSelected(item)
            }
        }
    }

    private fun createObjectsPopupMenu(anchor: View): PopupMenu {
```

```
        val popupMenu = PopupMenu(this, anchor)
        popupMenu.menuInflater.inflate(R.menu.objects_popup_menu,
popupMenu.menu)
        popupMenu.setOnMenuItemClickListener { menuItem ->
            popupMenu.menu.forEach {
                if (it.isChecked) {
                    it.isChecked = false
                }
            }
            menuItem.isChecked = true
            when (menuItem.itemId) {
                R.id.btn_point -> {
                    shapeObjectsEditor.startPointEditor()
                    supportActionBar?.title = getString(R.string.point)
                    true
                }
                R.id.btn_line -> {
                    shapeObjectsEditor.startLineEditor()
                    supportActionBar?.title = getString(R.string.line)
                    true
                }
                R.id.btn_rectangle -> {
                    shapeObjectsEditor.startRectEditor()
                    supportActionBar?.title = getString(R.string.rectangle)
                    true
                }
                R.id.btn_ellipse -> {
                    shapeObjectsEditor.startEllipseEditor()
                    supportActionBar?.title = getString(R.string.ellipse)
                    true
                }
                else -> false
            }
        }
        return popupMenu
    }

    private fun showObjectsPopupMenu() {
        objectsPopupMenu?.show()
    }
}
```

PaintViewInterface.kt

```
package com.oop.lab2

import android.graphics.Canvas
import android.graphics.Paint

import com.oop.lab2.shape_editor.ShapeObjectsEditorInterface

interface PaintViewInterface {
    val paint: Paint
```

```
    var shapeObjectsEditor: ShapeObjectsEditorInterface

    val drawnShapesCanvas: Canvas

    val rubberTraceCanvas: Canvas

    fun repaint()

    fun clearCanvas(canvas: Canvas)
}
```

PaintView.kt

```
package com.oop.lab2

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

import com.oop.lab2.shape_editor.ShapeObjectsEditorInterface

class PaintView(context: Context, attrs: AttributeSet?):
    View(context, attrs),
    PaintViewInterface {
    override val paint = Paint().apply {
        isAntiAlias = true
        strokeWidth = 7f
    }
    override lateinit var shapeObjectsEditor: ShapeObjectsEditorInterface
    override lateinit var drawnShapesCanvas: Canvas
    override lateinit var rubberTraceCanvas: Canvas

    private var drawnShapesBitmap: Bitmap? = null
    private var rubberTraceBitmap: Bitmap? = null

    override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
        super.onMeasure(widthMeasureSpec, heightMeasureSpec)
        val width = MeasureSpec.getSize(widthMeasureSpec)
        val height = MeasureSpec.getSize(heightMeasureSpec)
        if (drawnShapesBitmap == null &&
            rubberTraceBitmap == null) {
            drawnShapesBitmap = Bitmap.createBitmap(width, height,
Bitmap.Config.ARGB_8888)
            drawnShapesCanvas = Canvas(drawnShapesBitmap!!)
            rubberTraceBitmap = Bitmap.createBitmap(width, height,
Bitmap.Config.ARGB_8888)
            rubberTraceCanvas = Canvas(rubberTraceBitmap!!)
```

```
        }
    }

    override fun onDraw(canvas: Canvas) {
        super.onDraw(canvas)
        if (!shapeObjectsEditor.isRubberTraceModeOn) {
            shapeObjectsEditor.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 -> shapeObjectsEditor.onFingerTouch(x, y)
            MotionEvent.ACTION_MOVE -> shapeObjectsEditor.onFingerMove(x, y)
            MotionEvent.ACTION_UP -> shapeObjectsEditor.onFingerRelease()
        }
        return true
    }

    override fun repaint() {
        invalidate()
    }

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

ShapeObjectsEditorInterface.kt

```
package com.oop.lab2.shape_editor

import com.oop.lab2.PaintViewInterface

interface ShapeObjectsEditorInterface {
    var isRubberTraceModeOn: Boolean

    var paintView: PaintViewInterface

    fun startPointEditor()

    fun startLineEditor()

    fun startRectEditor()

    fun startEllipseEditor()
```

```
    fun onFingerTouch(x: Float, y: Float)

    fun onFingerMove(x: Float, y:Float)

    fun onFingerRelease()

    fun onPaint()
}
```

ShapeObjectsEditor.kt

```
package com.oop.lab2.shape_editor

import com.oop.lab2.PaintViewInterface

// Імпорт пакетів модуля editor
import com.oop.lab2.editor.ShapeEditor
import com.oop.lab2.editor.PointShapeEditor
import com.oop.lab2.editor.LineShapeEditor
import com.oop.lab2.editor.RectShapeEditor
import com.oop.lab2.editor.EllipseShapeEditor

// Імпорт пакетів модуля shape
import com.oop.lab2.shape.Shape
import com.oop.lab2.shape.PointShape
import com.oop.lab2.shape.LineShape
import com.oop.lab2.shape.RectShape
import com.oop.lab2.shape.EllipseShape

class ShapeObjectsEditor: ShapeObjectsEditorInterface {
    override var isRubberTraceModeOn = false
    override lateinit var paintView: PaintViewInterface

    private val drawnShapes = mutableListOf<Shape>()
    private lateinit var activeEditor: ShapeEditor
    private lateinit var currentShape: () -> Shape

    override fun startPointEditor() {
        activeEditor = PointShapeEditor()
        currentShape = { PointShape() }
    }

    override fun startLineEditor() {
        activeEditor = LineShapeEditor()
        currentShape = { LineShape() }
    }

    override fun startRectEditor() {
        activeEditor = RectShapeEditor()
        currentShape = { RectShape() }
    }

    override fun startEllipseEditor() {
```

```
            activeEditor = EllipseShapeEditor()
            currentShape = { EllipseShape() }
        }

        override fun onFingerTouch(x: Float, y: Float) {
            activeEditor.shape = currentShape()
            activeEditor.onFingerTouch(x, y)
            isRubberTraceModeOn = true
        }

        override fun onFingerMove(x: Float, y:Float) {
            paintView.clearCanvas(paintView.rubberTraceCanvas)
            activeEditor.onFingerMove(paintView.rubberTraceCanvas,
paintView.paint, x, y)
            paintView.repaint()
        }

        override fun onFingerRelease() {
            activeEditor.onFingerRelease(drawnShapes)
            isRubberTraceModeOn = false
            paintView.repaint()
        }

        override fun onPaint() {
            paintView.clearCanvas(paintView.rubberTraceCanvas)
            paintView.clearCanvas(paintView.drawnShapesCanvas)
            drawnShapes.forEach { it.show(paintView.drawnShapesCanvas,
paintView.paint) }
        }
}
```

Shape.kt

```
package com.oop.lab2.shape

import android.graphics.Canvas
import android.graphics.Paint

abstract class Shape {
    protected var startX = 0F
    protected var startY = 0F
    protected var endX = 0F
    protected var endY = 0F

    protected abstract val defaultOutlineColor: Int
    protected abstract val defaultFillingColor: Int
    protected var outlineColor: Int? = null
    protected var fillingColor: Int? = null

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

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

    fun setColors(outlineColor: Int, fillingColor: Int) {
        this.outlineColor = outlineColor
        this.fillingColor = fillingColor
    }

    abstract fun show(canvas: Canvas, paint: Paint)
}
```

PointShape.kt

```
package com.oop.lab2.shape

import android.graphics.Canvas
import android.graphics.Color
import android.graphics.Paint

class PointShape: Shape() {
    override val defaultOutlineColor = Color.argb(255, 0, 0, 0)
    override val defaultFillingColor = Color.TRANSPARENT

    override fun show(canvas: Canvas, paint: Paint) {
        if (outlineColor == null) {
            paint.color = defaultOutlineColor
        } else {
            paint.color = outlineColor as Int
            outlineColor = null
        }
        paint.style = Paint.Style.STROKE
        canvas.drawPoint(startX, startY, paint)
    }
}
```

LineShape.kt

```
package com.oop.lab2.shape

import android.graphics.Canvas
import android.graphics.Color
import android.graphics.Paint

class LineShape: Shape() {
    override val defaultOutlineColor = Color.argb(255, 0, 0, 0)
    override val defaultFillingColor = Color.TRANSPARENT

    override fun show(canvas: Canvas, paint: Paint) {
        if (outlineColor == null) {
            paint.color = defaultOutlineColor
```

```
        } else {
            paint.color = outlineColor as Int
            outlineColor = null
        }
        paint.style = Paint.Style.STROKE
        canvas.drawLine(startX, startY, endX, endY, paint)
    }
}
```

RectShape.kt

```
package com.oop.lab2.shape

import android.graphics.Canvas
import android.graphics.Color
import android.graphics.Paint
import android.graphics.Rect

class RectShape: Shape() {
    override val defaultOutlineColor = Color.argb(255, 0, 0, 0)
    override val defaultFillingColor = Color.TRANSPARENT

    override fun show(canvas: Canvas, paint: Paint) {
        val startX = startX.toInt()
        val startY = startY.toInt()
        val endX = endX.toInt()
        val endY = endY.toInt()
        val rect = Rect(startX, startY, endX, endY)
        // Малюємо контур прямокутника
        if (outlineColor == null) {
            paint.color = defaultOutlineColor
        } else {
            paint.color = outlineColor as Int
            outlineColor = null
        }
        paint.style = Paint.Style.STROKE
        canvas.drawRect(rect, paint)
        // Малюємо заповнення прямокутника
        if (fillingColor == null) {
            paint.color = defaultFillingColor
        } else {
            paint.color = fillingColor as Int
            fillingColor = null
        }
        paint.style = Paint.Style.FILL
        canvas.drawRect(rect, paint)
    }
}
```

EllipseShape.kt

```
package com.oop.lab2.shape
```

```
import android.graphics.Canvas
import android.graphics.Color
import android.graphics.Paint
import android.graphics.RectF

class EllipseShape: Shape() {
    override val defaultOutlineColor = Color.argb(255, 0, 0, 0)
    override val defaultFillingColor = Color.argb(255, 255, 255, 0)

    override fun show(canvas: Canvas, paint: Paint) {
        val rect = RectF(startX, startY, endX, endY)
        // Малюємо контур еліпса
        if (outlineColor == null) {
            paint.color = defaultOutlineColor
        } else {
            paint.color = outlineColor as Int
            outlineColor = null
        }
        paint.style = Paint.Style.STROKE
        canvas.drawOval(rect, paint)
        // Малюємо заповнення еліпса
        if (fillingColor == null) {
            paint.color = defaultFillingColor
        } else {
            paint.color = fillingColor as Int
            fillingColor = null
        }
        paint.style = Paint.Style.FILL
        canvas.drawOval(rect, paint)
    }
}
```

Editor.kt

```
package com.oop.lab2.editor

import android.graphics.Canvas
import android.graphics.Paint

import com.oop.lab2.shape.Shape

abstract class Editor {
    abstract fun onFingerTouch(x: Float, y: Float)

    abstract fun onFingerMove(canvas: Canvas, paint: Paint, x: Float, y:
Float)

    abstract fun onFingerRelease(drawnShapes: MutableList<Shape>)
}
```

ShapeEditor.kt

```
package com.oop.lab2.editor

import android.graphics.Color
import com.oop.lab2.shape.Shape

abstract class ShapeEditor: Editor() {
    lateinit var shape: Shape

    override fun onFingerRelease(drawnShapes: MutableList<Shape>) {
        drawnShapes.add(shape)
    }

    protected fun setRubberTraceMode() {
        val outlineColor = Color.argb(255, 255, 0, 0)
        val fillingColor = Color.TRANSPARENT
        shape.setColors(outlineColor, fillingColor)
    }
}
```

PointShapeEditor.kt

```
package com.oop.lab2.editor

import android.graphics.Canvas
import android.graphics.Paint

class PointShapeEditor: ShapeEditor() {
    override fun onFingerTouch(x: Float, y: Float) {
        shape.setStart(x, y)
    }

    override fun onFingerMove(canvas: Canvas, paint: Paint, x: Float, y:
Float) {
        setRubberTraceMode()
        shape.show(canvas, paint)
    }
}
```

LineShapeEditor.kt

```
package com.oop.lab2.editor

import android.graphics.Canvas
import android.graphics.Paint

class LineShapeEditor: ShapeEditor() {
    override fun onFingerTouch(x: Float, y: Float) {
        shape.setStart(x, y)
    }

    override fun onFingerMove(canvas: Canvas, paint: Paint, x: Float, y:
Float) {
        shape.setEnd(x, y)
```

```
        setRubberTraceMode()
        shape.show(canvas, paint)
    }
}
```

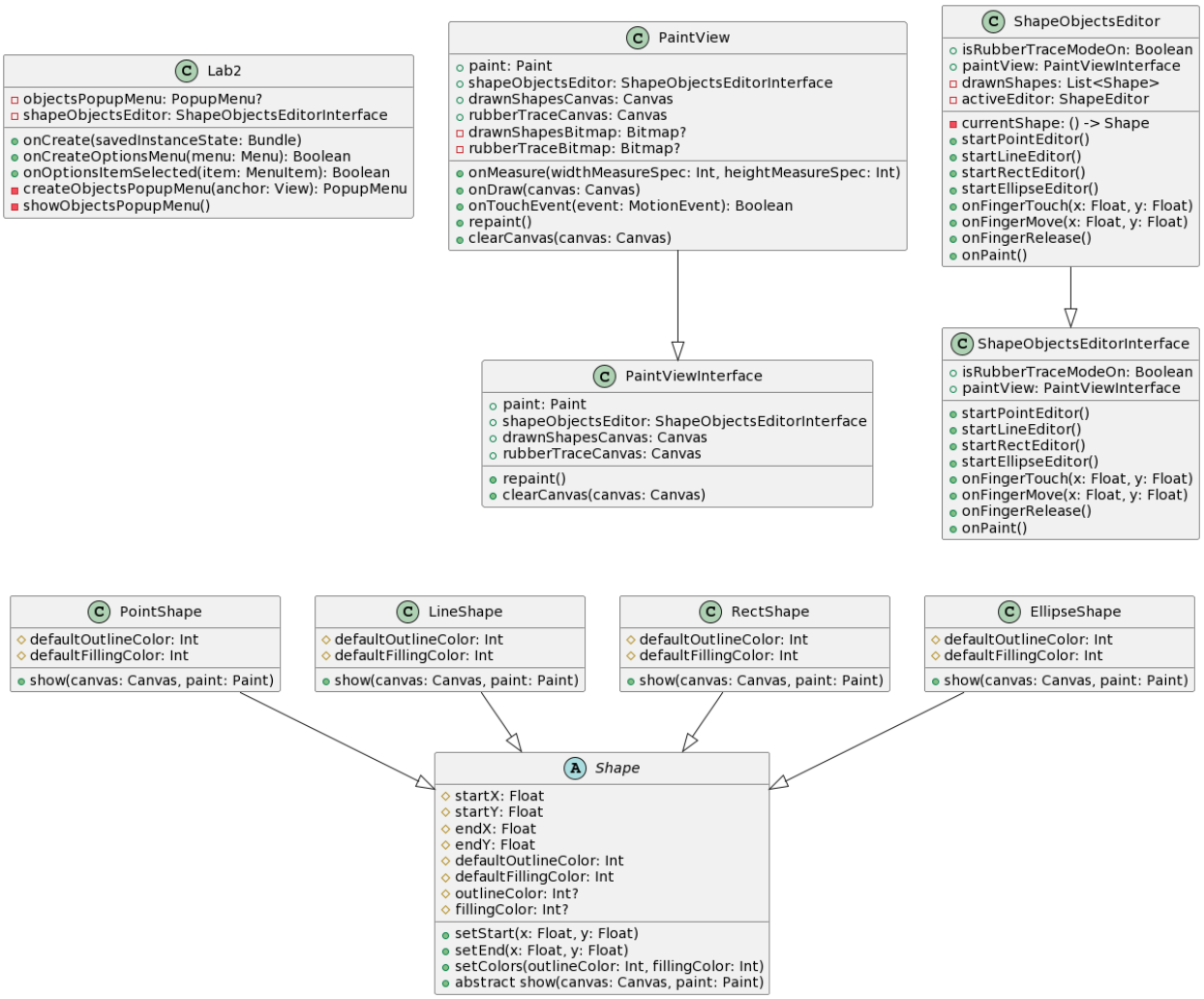
RectShapeEditor.kt

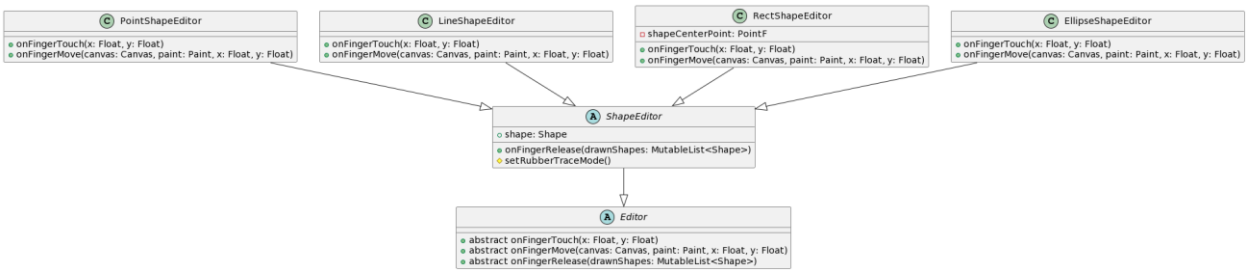
```
package com.oop.lab2.editor

import android.graphics.Canvas
import android.graphics.Paint
import android.graphics.PointF

class RectShapeEditor: ShapeEditor() {
    private val shapeCenterPoint = PointF()

    override fun onFingerTouch(x: Float, y: Float) {
        shapeCenterPoint.set(x, y)
    }

    override fun onFingerMove(canvas: Canvas, paint: Paint, x: Float, y:
Float) {
        val shapeStartX: Float
        val shapeStartY: Float
        val shapeEndX: Float
        val shapeEndY: Float

        val dx = x - shapeCenterPoint.x
        val oppositeX = shapeCenterPoint.x - dx
        if (x < shapeCenterPoint.x) {
            shapeStartX = x
            shapeEndX = oppositeX
        } else {
            shapeStartX = oppositeX
            shapeEndX = x
        }

        val dy = y - shapeCenterPoint.y
        val oppositeY = shapeCenterPoint.y - dy
        if (y < shapeCenterPoint.y) {
            shapeStartY = y
            shapeEndY = oppositeY
        } else {
            shapeStartY = oppositeY
            shapeEndY = y
        }

        shape.setStart(shapeStartX, shapeStartY)
        shape.setEnd(shapeEndX, shapeEndY)
        setRubberTraceMode()
        shape.show(canvas, paint)
    }
}
```

EllipseShapeEditor.kt

```
package com.oop.lab2.editor

import android.graphics.Canvas
import android.graphics.Paint

class EllipseShapeEditor : ShapeEditor() {
    override fun onFingerTouch(x: Float, y: Float) {
        shape.setStart(x, y)
    }

    override fun onFingerMove(canvas: Canvas, paint: Paint, x: Float, y:
Float) {
        shape.setEnd(x, y)
        setRubberTraceMode()
        shape.show(canvas, paint)
    }
}
```

Діаграма класів програми

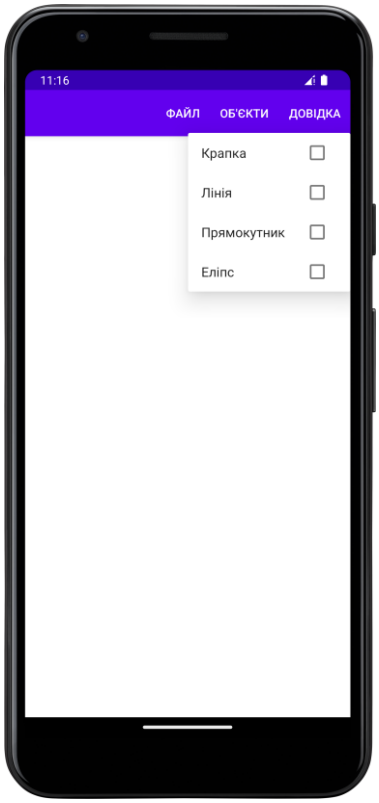


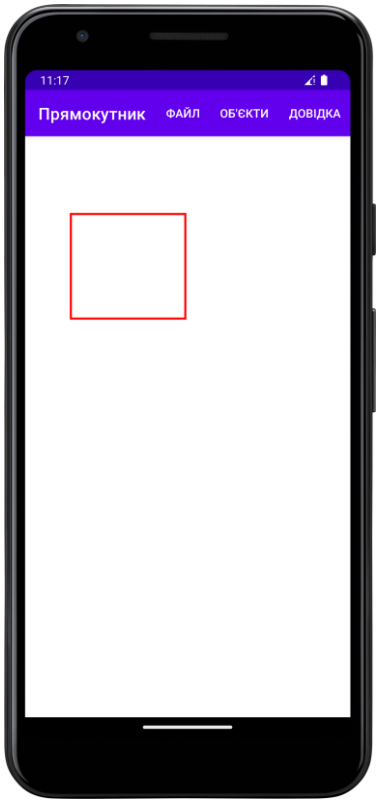


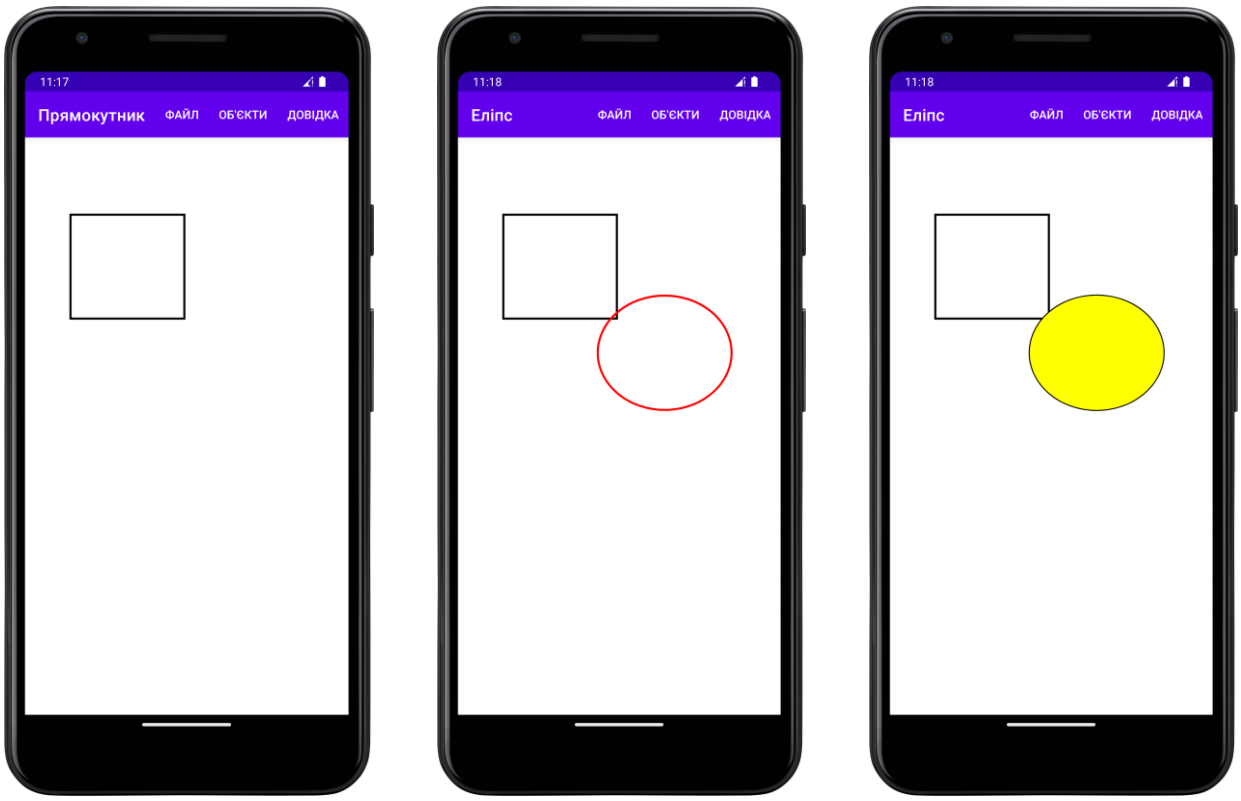
Позначення для модифікаторів видимості

Позначення для поля	*Позначення для методу*	*Модифікатор видимості*
		`private`
		`protected`
		`public`

Ілюстрації виконання програми







Висновки

Під час виконання цієї лабораторної роботи я навчився використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів мови програмування ***Kotlin***, запрограмувавши простий графічний редактор для платформи ***Android*** в об’єктно-орієнтованому стилі.