**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»

Виконав: Перевірив:

Студент групи ІМ-22 Порєв В.М  
Тимофеєв Даниіл Костянтинович  
номер у списку групи: 23

Київ 2023

**Мета**: Мета роботи – 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. Перевірити роботу програми. Налагодити програму.

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.

**Варіанти** :

1. Для вибору варіанту використовується значення Ж = Жлаб2, де Жлаб2 – номер студента в журналі, який використовувався для попередньої лаб. роботи No = 23

2. Статичний масив Shape \*pcshape[N]; причому, кількість елементів масиву вказівників як для статичного, так і динамічного має бути N = 23+100 = 123.

3. “Гумовий” слід при вводі об’єкті - пунктирна лінія чорного кольору для варіантів

4. **Прямокутник**:

* Увід прямокутника - по двом протилежним кутам для варіантів (Ж mod 2 = 0)
* Відображення прямокутника - чорний контур прямокутника без заповнення для (Ж mod 5 = 3 або 4)

5. **Еліпс**:

* Від центру до одного з кутів охоплюючого прямокутника для варіантів (Ж mod 2 = 0)
* Відображення еліпсу - чорний контур з кольоровим заповненням для (Ж mod 5 = 3 або 4)
* Кольори заповнення еліпсу - сірий для (Ж mod 6 = 0)

6. Позначка поточного типу об’єкту, що вводиться - в меню (метод OnInitMenuPopup) для варіантів (Ж mod 2 = 0)

**Вихідний текст програм** :

**Shape.kt**

open class Shape (protected var paintSettings : Paint) {

protected var isEraserMode: Boolean = true

var startXCoordinate: Float = 0f

var startYCoordinate: Float = 0f

var endXCoordinate: Float = 0f

var endYCoordinate: Float = 0f

fun defineEraserMode (eraserMode: Boolean) {

isEraserMode = eraserMode

}

fun defineStartCoordinates (x: Float, y: Float) {

startXCoordinate = x

startYCoordinate = y

}

fun defineEndCoordinates (x: Float, y: Float) {

endXCoordinate = x

endYCoordinate = y

}

open fun draw (canvas: Canvas) {}

open fun configureDrawing () {

paintSettings.apply {

pathEffect = PathEffect()

}

}

fun defineEraserDrawingStyle () {

paintSettings.apply {

color = Color.BLACK

style = Paint.Style.STROKE

pathEffect = DashPathEffect(floatArrayOf(45f, 35f), 0f)

}

}

}  
**MyEditor.kt**

class MyEditor (private val painSettings: Paint, private val shapesList: MutableList<Shape>) {

private var initialShape = Shape(painSettings)

private val shapesLimit: Int = 123

fun onTouchDown (x: Float, y: Float) {

defineInitialShape(initialShape)

initialShape.defineStartCoordinates(x, y)

}

fun onTouchUp () {

initialShape.let {

if (shapesList.contains(it)) shapesList.remove(it)

it.defineEraserMode(false)

addShapeToEditor(it, shapesList)

}

}

fun handleMouseMovement (x: Float, y: Float) {

initialShape.let {

if (shapesList.contains(it)) shapesList.remove(it)

it.defineEndCoordinates(x, y)

addShapeToEditor(it, shapesList)

}

}

fun defineInitialShape (shape: Shape) {

initialShape = shape.javaClass.getDeclaredConstructor(Paint::class.java).newInstance(painSettings)

}

private fun addShapeToEditor (shape: Shape, shapes: MutableList<Shape>) {

if (shapes.lastIndex == shapesLimit - 1) {

shapes.removeAt(shapes.lastIndex)

}

shapes.add(shape)

}

}  
**DotShape.kt**

class DotShape (paintSettings: Paint) : Shape(paintSettings) {

override fun draw (canvas: Canvas) {

configureDrawing()

paintSettings.strokeWidth = 15f

canvas.drawPoint(startXCoordinate, startYCoordinate, paintSettings)

}

override fun configureDrawing () {

paintSettings.apply { color = Color.BLACK }

}

}  
**EllipseShape.kt**

class EllipseShape (paintSettings: Paint) : Shape(paintSettings) {

override fun draw(canvas: Canvas) {

val left = 2 \* startXCoordinate - endXCoordinate

val top = 2 \* startYCoordinate - endYCoordinate

val right = endXCoordinate

val bottom = endYCoordinate

val ovalRect = RectF(left, top, right, bottom)

if (!isEraserMode) {

configureFillStyle()

canvas.drawOval(ovalRect, paintSettings)

configureDrawing()

canvas.drawOval(ovalRect, paintSettings)

} else {

defineEraserDrawingStyle()

canvas.drawOval(ovalRect, paintSettings)

}

}

override fun configureDrawing () {

super.configureDrawing()

paintSettings.apply {

this.color = Color.BLACK

this.style = Paint.Style.STROKE

this.strokeWidth = 15f

}

}

private fun configureFillStyle () {

applyDrawingStyle(Color.rgb(128, 128, 128), Paint.Style.FILL)

}

private fun applyDrawingStyle (color: Int, style: Paint.Style) {

paintSettings.apply {

this.color = color

this.style = style

}

}

}  
**LineShape.kt**

open class LineShape (paintSettings: Paint) : Shape(paintSettings) {

override fun draw (canvas: Canvas) {

if (!isEraserMode) configureDrawing()

else defineEraserDrawingStyle()

canvas.drawLine(startXCoordinate, startYCoordinate, endXCoordinate, endYCoordinate, paintSettings)

}

override fun configureDrawing () {

super.configureDrawing()

paintSettings.apply {

color = Color.BLACK

style = Paint.Style.FILL\_AND\_STROKE

strokeWidth = 15f

}

}

}  
**RectangleShape.kt**

open class RectangleShape (paintSettings: Paint) : Shape(paintSettings) {

override fun draw (canvas: Canvas) {

configureDrawing()

if (isEraserMode) defineEraserDrawingStyle()

val rect = RectF(startXCoordinate, startYCoordinate, endXCoordinate, endYCoordinate)

canvas.drawRect(rect, paintSettings)

}

override fun configureDrawing () {

super.configureDrawing()

paintSettings.apply {

color = Color.BLACK

style = Paint.Style.STROKE

}

}

}  
**CubeShape.kt**

class CubeShape (paintSettings: Paint): RectangleShape(paintSettings) {

override fun draw(canvas: Canvas) {

if (!isEraserMode) configureDrawing()

else defineEraserDrawingStyle()

val halfWidth = abs(startXCoordinate - endXCoordinate) / 2

val heightDifference = abs(startYCoordinate - endYCoordinate)

val topLeftX = 2 \* startXCoordinate - endXCoordinate

val topLeftY = 2 \* startYCoordinate - endYCoordinate

drawRect(canvas, topLeftX, topLeftY, endXCoordinate, endYCoordinate)

drawRect(canvas, topLeftX + halfWidth, topLeftY - heightDifference, endXCoordinate + halfWidth, endYCoordinate - heightDifference)

drawLines(canvas, topLeftX, topLeftY, endXCoordinate, endYCoordinate, halfWidth, heightDifference)

}

private fun drawRect (canvas: Canvas, left: Float, top: Float, right: Float, bottom: Float) {

canvas.drawRect(left, top, right, bottom, paintSettings)

}

private fun drawLines (canvas: Canvas, x1: Float, y1: Float, x2: Float, y2: Float, halfWidth: Float, heightDifference: Float) {

canvas.drawLine(x1, y1, x1 + halfWidth, y1 - heightDifference, paintSettings)

canvas.drawLine(x2, y2, x2 + halfWidth, y2 - heightDifference, paintSettings)

canvas.drawLine(x1, y2, x1 + halfWidth, y2 - heightDifference, paintSettings)

canvas.drawLine(x2, y1, x2 + halfWidth, y1 - heightDifference, paintSettings)

}

}   
**LineWithCirclesShape.kt**

class LineWithCirclesShape (painSettings: Paint): LineShape(painSettings) {

private var circleRadius = 10f

init {

paintSettings.style = Paint.Style.FILL

}

override fun draw (canvas: Canvas) {

super.draw(canvas)

canvas.drawCircle(startXCoordinate, startYCoordinate, circleRadius, paintSettings)

canvas.drawCircle(endXCoordinate, endYCoordinate, circleRadius, paintSettings)

}

}

**CustomDrawingView.kt**

class CustomDrawingView(context: Context) : View(context) {

companion object {

private const val BACKGROUND\_COLOUR = Color.WHITE

private const val DRAWING\_COLOR = Color.BLACK

private const val STROKE\_WIDTH = 15f

}

private var drawingCanvas = Canvas()

private val shapeList = mutableListOf<Shape>()

private var currentX = 0f

private var currentY = 0f

val drawingSetting = Paint().apply {

color = DRAWING\_COLOR

strokeWidth = STROKE\_WIDTH

style = Paint.Style.STROKE

strokeCap = Paint.Cap.ROUND

strokeJoin = Paint.Join.ROUND

isAntiAlias = true

}

private var actualShapeEditor: MyEditor = MyEditor(drawingSetting, shapeList)

override fun onDraw(canvas: Canvas?) {

super.onDraw(canvas)

shapeList.forEach { it.draw(canvas!!) }

}

fun setShapePrimitiveEditor(shape: Shape) {

actualShapeEditor.defineInitialShape(shape)

}

private fun handleTouchUp() {

invalidate()

actualShapeEditor.onTouchUp()

}

private fun handleTouchMove() {

invalidate()

actualShapeEditor.handleMouseMovement(currentX, currentY)

}

private fun handleTouchStart() {

invalidate()

actualShapeEditor.onTouchDown(currentX, currentY)

}

override fun onTouchEvent(event: MotionEvent?): Boolean {

currentX = event!!.x

currentY = event.y

when (event.action) {

MotionEvent.ACTION\_MOVE -> handleTouchMove()

MotionEvent.ACTION\_UP -> handleTouchUp()

MotionEvent.ACTION\_DOWN -> handleTouchStart()

}

return true

}

override fun onSizeChanged(newWidth: Int, newHeight: Int, oldWidth: Int, oldHeight: Int) {

super.onSizeChanged(newWidth, newHeight, oldWidth, oldHeight)

drawingCanvas = Canvas()

drawingCanvas.drawColor(BACKGROUND\_COLOUR)

}

}   
**MainActivity.kt**

class MainActivity : AppCompatActivity() {

private lateinit var drawingView: CustomDrawingView

private lateinit var currentSelectedOption: MenuItem

private lateinit var mainMenu: Menu

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

drawingView = CustomDrawingView(this)

drawingView.setShapePrimitiveEditor(DotShape(drawingView.drawingSetting))

setContentView(drawingView)

showSystemBars()

}

override fun onCreateOptionsMenu(menu: Menu?): Boolean {

val mainMenuInflater: MenuInflater = menuInflater

mainMenuInflater.inflate(R.menu.main\_menu, menu)

mainMenu = menu!!

currentSelectedOption = mainMenu.findItem(R.id.dotIcon)

setPrimitiveIcon(currentSelectedOption, R.drawable.dot)

updateActionBarTitle(currentSelectedOption.title.toString())

currentSelectedOption.isChecked = true

return true

}

override fun onOptionsItemSelected(item: MenuItem): Boolean {

defineDisabledIcon(currentSelectedOption)

when (item.itemId) {

R.id.ellipseIcon, R.id.ellipseSelect -> {

currentSelectedOption = mainMenu.findItem(R.id.ellipseIcon)

drawingView.setShapePrimitiveEditor(EllipseShape(drawingView.drawingSetting))

currentSelectedOption.icon = ContextCompat.getDrawable(this, R.drawable.ellipse)

}

R.id.lineIcon, R.id.lineSelect -> {

currentSelectedOption = mainMenu.findItem(R.id.lineIcon)

drawingView.setShapePrimitiveEditor(LineShape(drawingView.drawingSetting))

currentSelectedOption.icon = ContextCompat.getDrawable(this, R.drawable.line)

}

R.id.dotIcon, R.id.dotSelect -> {

currentSelectedOption = mainMenu.findItem(R.id.dotIcon)

drawingView.setShapePrimitiveEditor(DotShape(drawingView.drawingSetting))

currentSelectedOption.icon = ContextCompat.getDrawable(this, R.drawable.dot)

}

R.id.rectangleIcon, R.id.rectangleSelect -> {

currentSelectedOption = mainMenu.findItem(R.id.rectangleIcon)

drawingView.setShapePrimitiveEditor(RectangleShape(drawingView.drawingSetting))

currentSelectedOption.icon = ContextCompat.getDrawable(this, R.drawable.rectangle)

}

R.id.cubeIcon, R.id.cubeSelect -> {

currentSelectedOption = mainMenu.findItem(R.id.cubeIcon)

drawingView.setShapePrimitiveEditor(CubeShape(drawingView.drawingSetting))

currentSelectedOption.icon = ContextCompat.getDrawable(this, R.drawable.cube)

}

R.id.lineswithcirclesIcon, R.id.lineswithcirclesSelect -> {

currentSelectedOption = mainMenu.findItem(R.id.lineswithcirclesIcon)

drawingView.setShapePrimitiveEditor(LineWithCirclesShape(drawingView.drawingSetting))

currentSelectedOption.icon = ContextCompat.getDrawable(this, R.drawable.linewithcircles)

}

}

updateActionBarTitle(currentSelectedOption.title.toString())

return super.onOptionsItemSelected(item)

}

private fun showSystemBars() {

WindowCompat.setDecorFitsSystemWindows(window, true)

WindowInsetsControllerCompat(window, drawingView).show(WindowInsetsCompat.Type.systemBars())

}

private fun setPrimitiveIcon(item: MenuItem, iconResourceId: Int) {

item.icon = ContextCompat.getDrawable(this, iconResourceId)

}

private fun defineDisabledIcon(item: MenuItem) {

when (item.itemId) {

R.id.ellipseIcon -> setPrimitiveIcon(item, R.drawable.ellipse\_disabled)

R.id.lineIcon -> setPrimitiveIcon(item, R.drawable.line\_disabled)

R.id.dotIcon -> setPrimitiveIcon(item, R.drawable.dot\_disabled)

R.id.rectangleIcon -> setPrimitiveIcon(item, R.drawable.rectangle\_disabled)

R.id.cubeIcon -> setPrimitiveIcon(item, R.drawable.cube\_disabled)

R.id.lineswithcirclesIcon -> setPrimitiveIcon(item, R.drawable.lineswithcircles\_disabled)

}

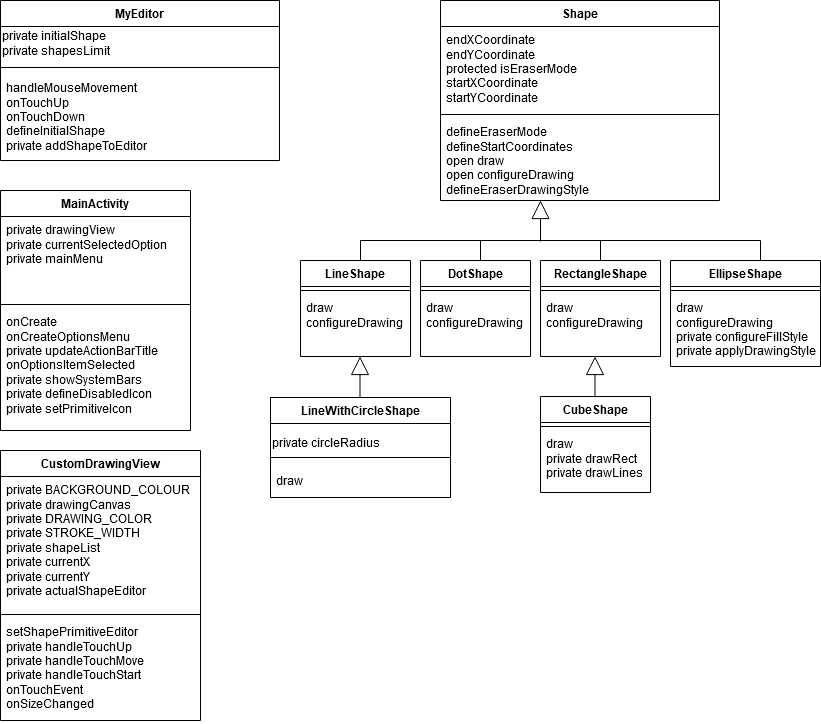
}

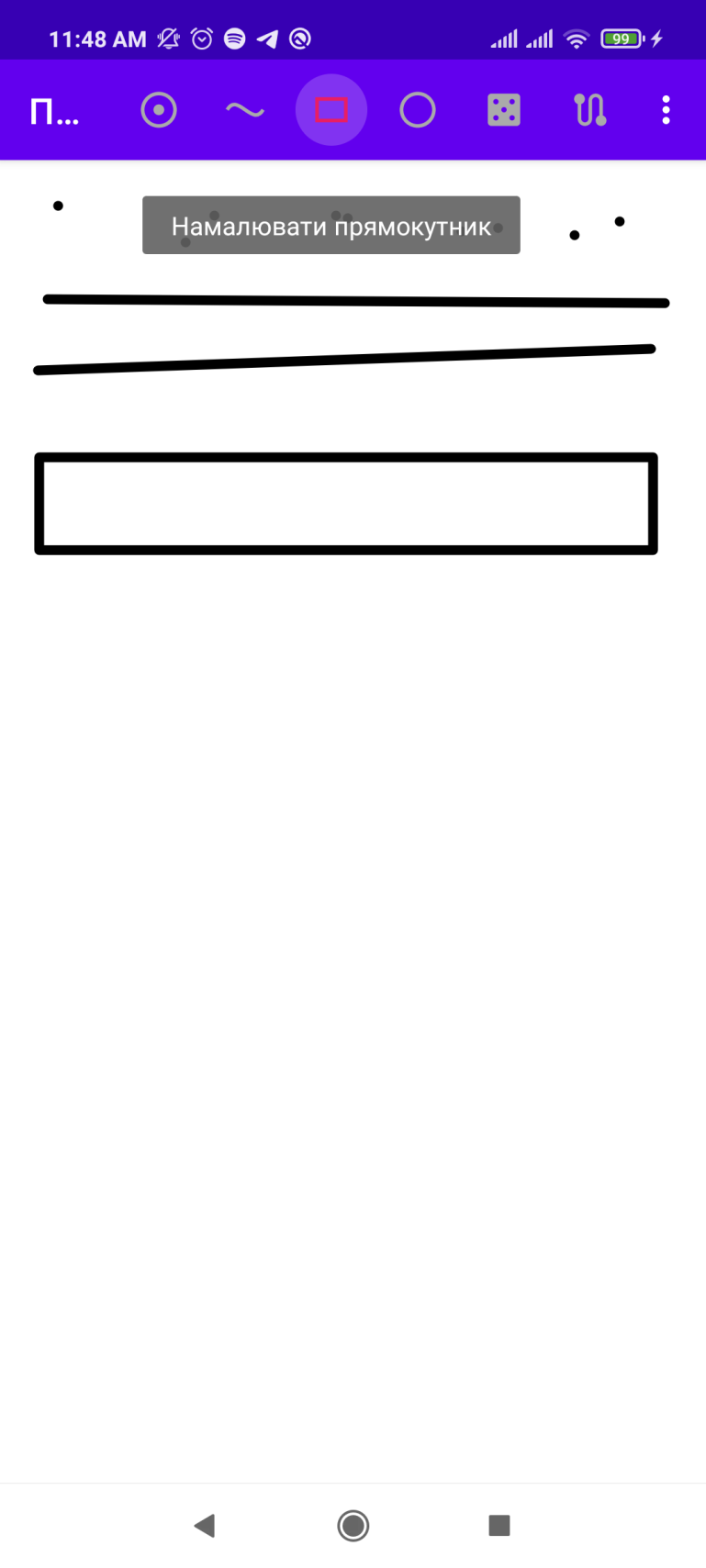
private fun updateActionBarTitle(title: String) {

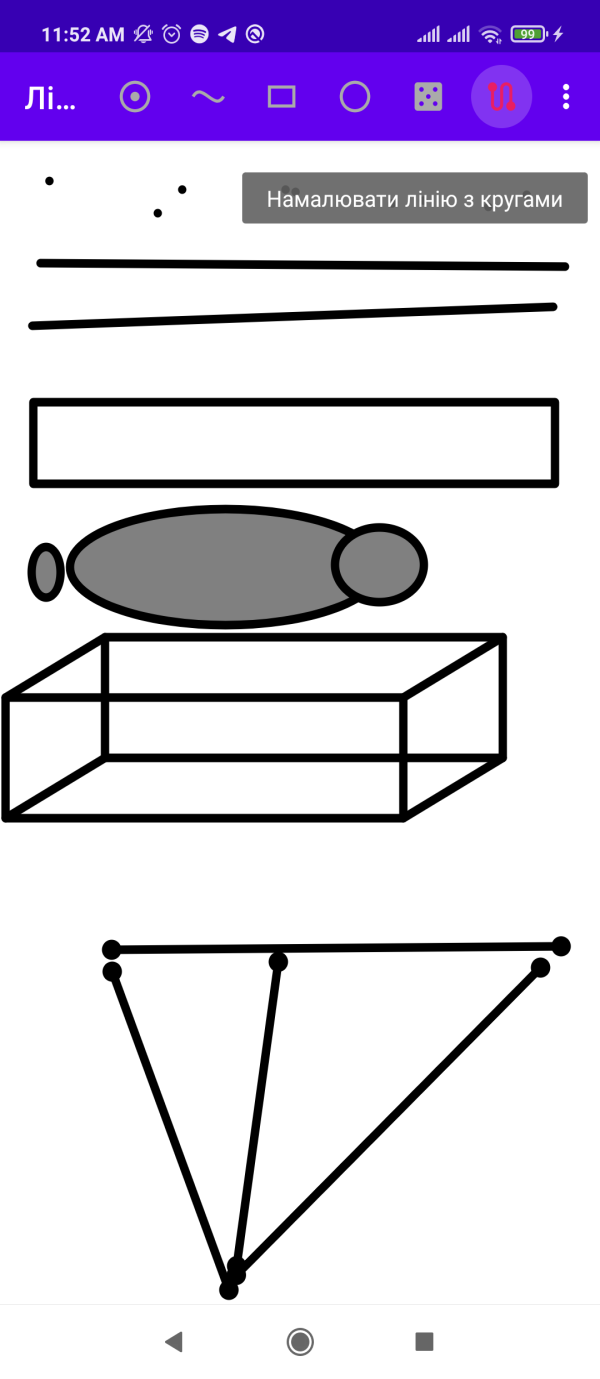
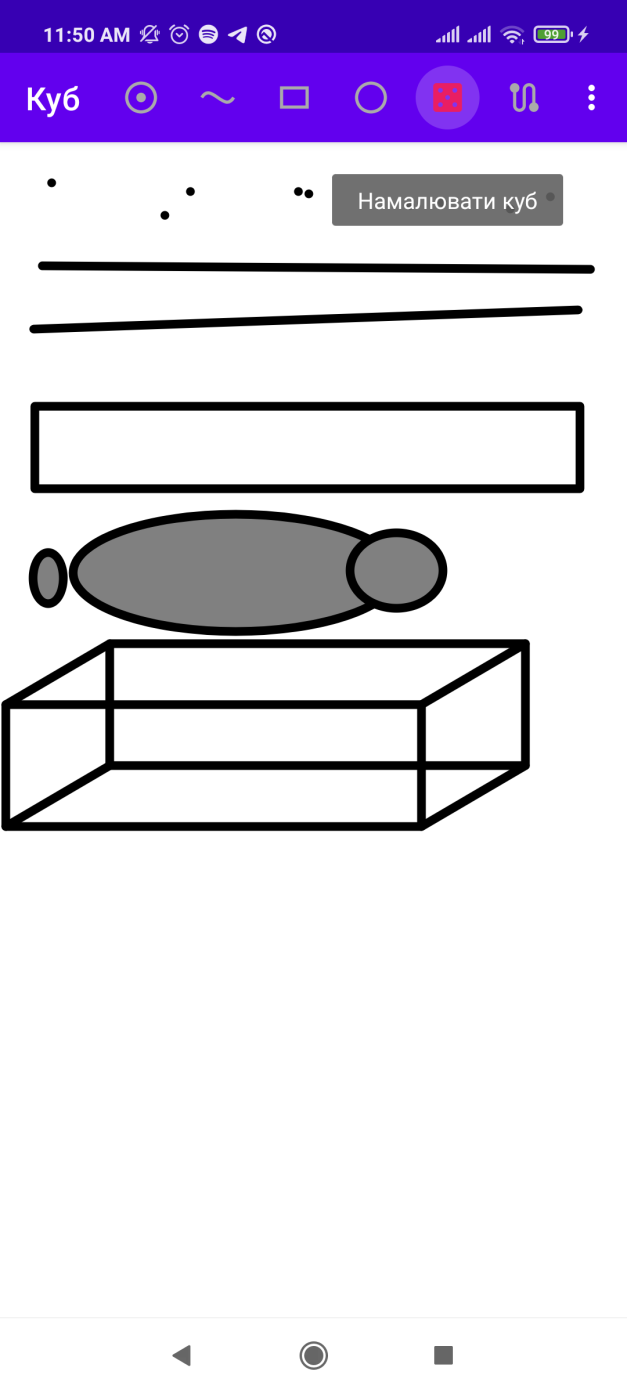
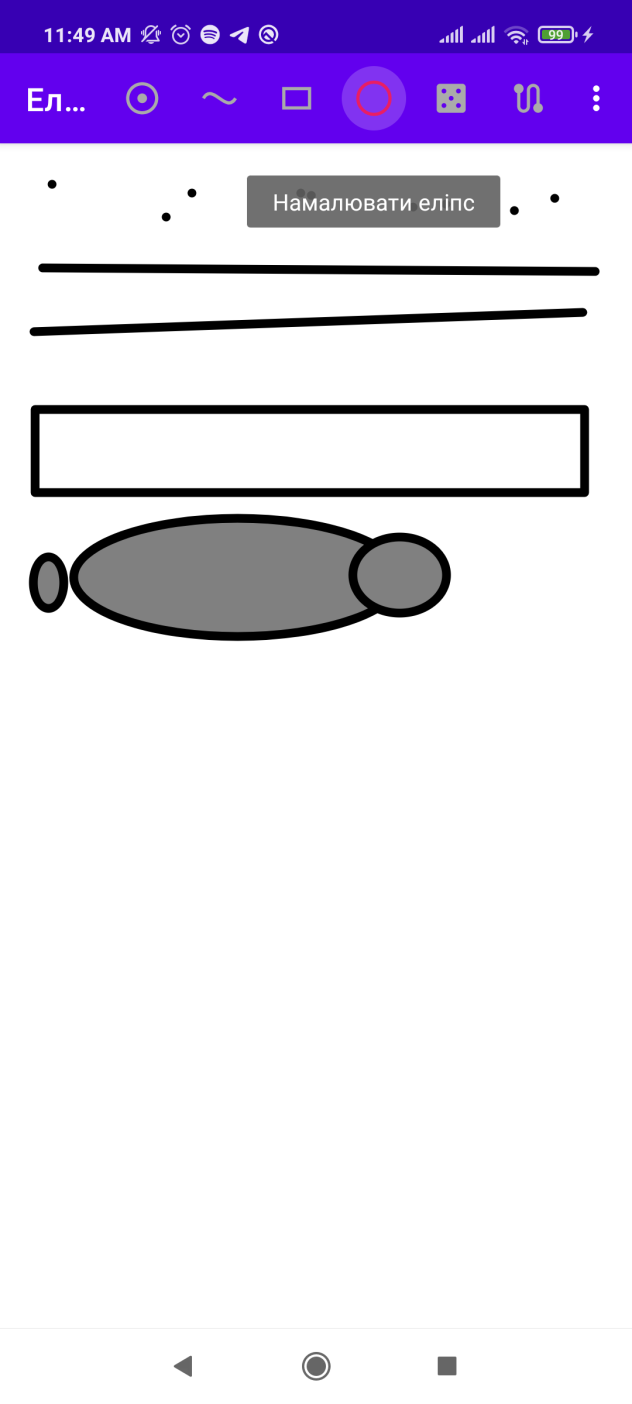
supportActionBar?.title = title

}

}   
**Діаграма класів :**



 **Тестування програми (скріншоти)**

****

**Висновки:** В ході виконання лабораторної роботи № 4 я розробляв графічний додаток для малювання різних фігур на екрані, який включає в себе елемент тулбара. У цій програмі широко використовується поліморфізм. Поліморфізм дозволяє об'єктам різних класів виконувати однакові дії, а також надає гнучкість і можливість розширення програми.

Клас Shape є абстрактним базовим класом для всіх графічних фігур і містить абстрактні методи, такі як draw і configureDrawing. Кожен конкретний клас фігури (наприклад, DotShape, LineShape, EllipseShape, RectangleShape) реалізує ці методи у власний спосіб. При цьому всі ці класи можуть бути використані як Shape, і їхні методи можуть бути викликані поліморфно.

У цій лабораторній роботі був введений рефакторинг коду. Розроблено новий головний клас графічного редактора MyEditor, який інкапсулює список об'єктів типу Shape, та містити інтерфейсні функції-члени, які повинні реалізовувати основні функції графічного редактора. Відповідно прибрали класи ShapeEditor