**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 №5**

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 програмувати

багатовіконний інтерфейс програми на С++ в об’єктно-орієнтованому стилі.

**Завдання :**

1. Створити у середовищі MS Visual Studio C++ проект Desktop

Application з ім’ям Lab5.

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.

**Варіанти** :

1. Для усіх варіантів завдань необхідно дотримуватися вимог та положень, викладених вище у порядку виконання роботи та методичних рекомендаціях.

2. Номер варіанту завдання дорівнює номеру зі списку студентів у журналі. Студенти з непарним номером (1, 3, 5, . . .) програмують глобальний статичний об'єкт класу MyEditor у вигляді Singleton Меєрса. Студенти з парним номером (2, 4, 6, . . .) програмують об’єкт класу MyEditor на основі класичної реалізації Singleton.

3. Усі кольори та стилі геометричних форм – як у попередньої лаб. роботі No4.

4. Запрограмувати вікно таблиці. Для його відкриття та закриття передбачити окремий пункт меню. Вікно таблиці повинно автоматично закриватися при виході з програми.

5. Вікно таблиці – немодальне вікно діалогу. Таблиця повинна бути запрограмована як клас у окремому модулі. Інтерфейс модуля у вигляді оголошення класу таблиці

6. Запрограмувати запис файлу множини об’єктів, що вводяться

7. Оголошення класів для усіх типів об'єктів робити у окремих заголовочних файлах \*.h, а визначення функцій членів – у окремих файлах \*.cpp. Таким чином, програмний код для усіх наявних типів об'єктів розподілюється по множині окремих модулів.

8. Ієрархія класів та побудова модулів повинні бути зручними для можливостей додавання нових типів об'єктів без переписування коду вже існуючих модулів.

9. У звіті повинна бути схема успадкування класів – діаграма класів. Побудувати діаграму класів засобами Visual Studio C++.

10. Бонуси-заохочення, які можуть суттєво підвищити оцінку лабораторної роботи. Оцінка підвищується за виконання кожного пункту, з наведених нижче:

* Якщо у вікні таблиці буде передбачено, щоб користувач міг виділити курсором рядок таблиці і відповідний об’єкт буде якось виділятися на зображенні у головному вікні.
* Якщо у вікні таблиці користувач може виділити курсором рядок таблиці і відповідний об’єкт буде вилучено з масиву об’єктів.
* Якщо програма не тільки записує у файл опис множини об’єктів, аще й здатна завантажити такий файл і відобразити відповідні об’єкти у головному вікні та вікні таблиці

**Вихідний текст програм** :

**Shape.kt**

open class Shape (protected var paintSettings : Paint) {

protected var isEraserMode: Boolean = true

protected var isHighlightingMode: Boolean = false

var startXCoordinate: Float = 0f

var startYCoordinate: Float = 0f

var endXCoordinate: Float = 0f

var endYCoordinate: Float = 0f

fun defineEraserMode (eraserMode: Boolean) {

isEraserMode = eraserMode

}

fun defineHighlightingMode () {

isHighlightingMode = !isHighlightingMode

}

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) {

if (isHighlightingMode) {

paintSettings.apply { color = Color.GREEN }

}

}

open fun configureDrawing () {

paintSettings.apply {

pathEffect = PathEffect()

}

}

fun defineEraserDrawingStyle () {

paintSettings.apply {

color = Color.BLACK

style = Paint.Style.STROKE

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

}

}

private fun alignCoordinates (coordinate: Float): String {

val decimalFormat = DecimalFormat("0.00")

return decimalFormat.format(coordinate) + "\t".repeat(5)

}

fun receiveHighlightingCondition (): Boolean {

return isHighlightingMode

}

private fun getClassNameString (): String = this::class.java.name + "\t".repeat(5)

open fun toShapeCoordinate (): ShapeCoordinate {

return ShapeCoordinate(

getClassNameString(),

alignCoordinates(startXCoordinate),

alignCoordinates(startYCoordinate),

alignCoordinates(endXCoordinate),

alignCoordinates(endYCoordinate),

this)

}

}

**MyEditor.kt**

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

private var initialShape = Shape(painSettings)

private val shapesLimit: Int = 123

private lateinit var table: MyTable

companion object {

private var instance: MyEditor? = null

fun getInstance (painSettings: Paint, shapesList: MutableList<Shape>): MyEditor {

if (instance == null) instance = MyEditor(painSettings, shapesList)

return instance!!

}

}

fun defineTable (table: MyTable) {

this.table = table

}

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)

table.insertRow(initialShape.toShapeCoordinate())

}

}

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()

super.draw(canvas)

paintSettings.strokeWidth = 10f

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()

super.draw(canvas)

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 = 10f

}

}

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()

super.draw(canvas)

} 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 = 10f

}

}

}

**RectangleShape.kt**

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

override fun draw (canvas: Canvas) {

configureDrawing()

if (isEraserMode) defineEraserDrawingStyle()

super.draw(canvas)

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)

if (isHighlightingMode) paintSettings.apply { color = Color.GREEN }

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 circleSize = 10f

init {

paintSettings.style = Paint.Style.FILL

}

override fun draw (canvas: Canvas) {

super.draw(canvas)

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

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

}

}

**CustomDrawingView.kt**

class CustomDrawingView (context: Context, attributeSet: AttributeSet): View(context, attributeSet) {

companion object {

private const val BACKGROUND\_COLOUR = Color.WHITE

private const val DRAWING\_COLOR = Color.BLACK

private const val STROKE\_WIDTH = 10f

}

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.getInstance(drawingSetting, shapeList)

private lateinit var table: MyTable

override fun onDraw(canvas: Canvas) {

super.onDraw(canvas)

shapeList.forEach { it.draw(canvas) }

}

fun setShapePrimitiveEditor(shape: Shape) {

actualShapeEditor.defineInitialShape(shape)

}

fun defineTable (table: MyTable) {

this.table = table

actualShapeEditor.defineTable(table)

}

fun receiveShapes (): MutableList<Shape> {

return shapeList

}

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

}

fun retrieveData (data: List<String>) {

shapeList.clear()

data.forEach { string ->

val fields = string.split("\t")

processFields(fields)

}

}

private fun processFields (fields: List<String>) {

val shape = createShapeFromFields(fields)

shape.let {

table.insertRow(it.toShapeCoordinate())

shapeList.add(it)

invalidate()

}

}

private fun createShapeFromFields (fields: List<String>): Shape {

val shapeType = Class.forName(fields[0])

val constructor = shapeType.getConstructor(Paint::class.java)

val shape = constructor.newInstance(drawingSetting) as Shape

shape.apply {

defineStartCoordinates(fields[1].toFloat(), fields[2].toFloat())

defineEndCoordinates(fields[3].toFloat(), fields[4].toFloat())

defineEraserMode(false)

}

return shape

}

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 frameLayout: FrameLayout

private lateinit var table: MyTable

private lateinit var mainMenu: Menu

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

drawingView = findViewById(R.id.mainCanvas)

drawingView.setShapePrimitiveEditor(LineShape(drawingView.drawingSetting))

frameLayout = findViewById(R.id.tableContainer)

table = MyTable()

val ft: FragmentTransaction = supportFragmentManager.beginTransaction()

ft.add(R.id.tableContainer, table)

ft.commit()

}

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\_casino)

}

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)

}

R.id.tableIcon, R.id.tableSelect -> {

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

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

if (frameLayout.visibility == View.GONE) frameLayout.visibility = View.VISIBLE

else frameLayout.visibility = View.GONE

}

}

updateActionBarTitle(currentSelectedOption.title.toString())

return super.onOptionsItemSelected(item)

}

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\_casino\_disabled)

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

}

}

private fun updateActionBarTitle(title: String) {

supportActionBar?.title = title

}

}

**MyTable.kt:**

class MyTable : Fragment() {

private val shapeCoordinates = mutableListOf<ShapeCoordinate>()

private lateinit var tableLayout: TableLayout

private lateinit var view: CustomDrawingView

private lateinit var fragmentView: View

private lateinit var saveImageButton: ImageButton

private lateinit var loadImageButton: ImageButton

override fun onCreateView (

inflater: LayoutInflater,

container: ViewGroup?,

savedInstanceState: Bundle?

): View {

fragmentView = inflater.inflate(R.layout.table, container, false)

return fragmentView

}

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {

super.onViewCreated(view, savedInstanceState)

initializeViews()

setupListeners()

}

fun insertRow (shapeCoordinate: ShapeCoordinate) {

shapeCoordinates.add(shapeCoordinate)

val shapeRow = createRowView(shapeCoordinate)

configureRowEvents(shapeRow, shapeCoordinate)

tableLayout.addView(shapeRow)

}

private fun initializeViews () {

view = requireActivity().findViewById(R.id.mainCanvas)

tableLayout = fragmentView.findViewById(R.id.tableLayout)

defineLayout(tableLayout)

view.defineTable(this)

loadImageButton = fragmentView.findViewById(R.id.loadImageButton)

saveImageButton = fragmentView.findViewById(R.id.saveImageButton)

}

private fun defineLayout (layout: TableLayout) {

this.tableLayout = layout

val header = createTableHeaderRow()

this.tableLayout.addView(header)

}

private fun setupListeners () {

loadImageButton.setOnClickListener { retrieveData() }

saveImageButton.setOnClickListener { saveData()}

}

private fun saveData() {

val filePath = context?.filesDir

val file = File(filePath, "SHAPES")

file.delete()

receiveRows().forEach { row ->

file.appendText("${row.shapeType.trim()}\t${row.startX.trim()}\t${row.startY.trim()}\t${row.endX.trim()}\t${row.endY}\n")

}

}

private fun retrieveData() {

emptyRows()

defineLayout(tableLayout)

val filePath = context?.filesDir

val file = File(filePath, "SHAPES")

val inputAsString = FileInputStream(file).bufferedReader().use { it.readText() }

val splitInput = inputAsString.split("\n")

val shapesData = splitInput.filter { splitInput.indexOf(it) != splitInput.lastIndex }

view.retrieveData(shapesData)

}

private fun createTableHeaderRow (): TableRow {

val tableHeaderRow = TableRow(tableLayout.context)

val shapeName = createTextView(getString(R.string.label\_name))

val startX = createTextView(getString(R.string.label\_x1))

val startY = createTextView(getString(R.string.label\_y1))

val endX = createTextView(getString(R.string.label\_x2))

val endY = createTextView(getString(R.string.label\_y2))

tableHeaderRow.addView(shapeName )

tableHeaderRow.addView(startX)

tableHeaderRow.addView(startY)

tableHeaderRow.addView(endX)

tableHeaderRow.addView(endY)

return tableHeaderRow

}

private fun createRowView (shapeCoordinate: ShapeCoordinate): TableRow {

val shapeRow = TableRow(tableLayout.context)

val shapeName = createTextView(shapeCoordinate.shapeType.split(".").last())

val startX = createTextView(shapeCoordinate.startX)

val startY = createTextView(shapeCoordinate.startY)

val endX = createTextView(shapeCoordinate.endX)

val endY = createTextView(shapeCoordinate.endY)

shapeRow.addView(shapeName)

shapeRow.addView(startX)

shapeRow.addView(startY)

shapeRow.addView(endX)

shapeRow.addView(endY)

return shapeRow

}

private fun createTextView (text: String): TextView {

return TextView(tableLayout.context).apply { this.text = text }

}

private fun removeAndShift (index: Int, shapes: MutableList<Shape>) {

shapes.removeAt(index)

}

private fun configureRowEvents (shapeRow: TableRow, shapeCoordinate: ShapeCoordinate) {

resetTextViewColors(shapeRow)

shapeRow.setOnLongClickListener {

handleRowLongClick(shapeRow, shapeCoordinate)

true

}

shapeRow.setOnClickListener {

handleRowClick(shapeRow, shapeCoordinate)

}

}

private fun resetTextViewColors (shapeRow: TableRow) {

for (i in 0 until shapeRow.childCount) {

val textView: TextView = shapeRow.getChildAt(i) as TextView

textView.setTextColor(Color.BLACK)

}

}

private fun handleRowLongClick (shapeRow: TableRow, shapeCoordinate: ShapeCoordinate) {

removeAndShift(view.receiveShapes().indexOf(shapeCoordinate.shapeReference), view.receiveShapes())

tableLayout.removeViewInLayout(shapeRow)

shapeCoordinates.remove(shapeCoordinate)

tableLayout.invalidate()

view.invalidate()

}

private fun handleRowClick (shapeRow: TableRow, shapeCoordinate: ShapeCoordinate) {

for (i in 0 until shapeRow.childCount) {

val textView: TextView = shapeRow.getChildAt(i) as TextView

val textColor = if (!shapeCoordinate.shapeReference.receiveHighlightingCondition()) Color.GREEN else Color.BLACK

textView.setTextColor(textColor)

}

shapeCoordinate.shapeReference.defineHighlightingMode()

view.invalidate()

}

private fun emptyRows () {

shapeCoordinates.clear()

tableLayout.removeAllViews()

}

private fun receiveRows (): MutableList<ShapeCoordinate> {

return shapeCoordinates

}

}

**ShapeCoordinate.kt:**

data class ShapeCoordinate (

val shapeType: String,

val startX: String,

val startY: String,

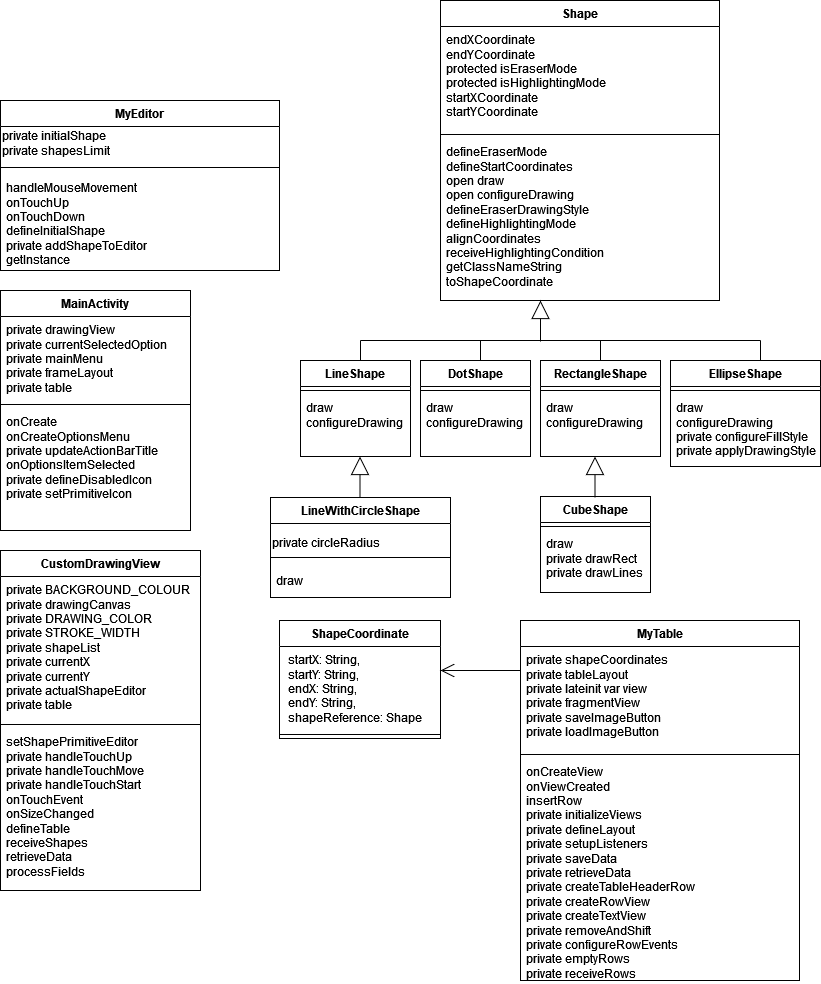
val endX: String,

val endY: String,

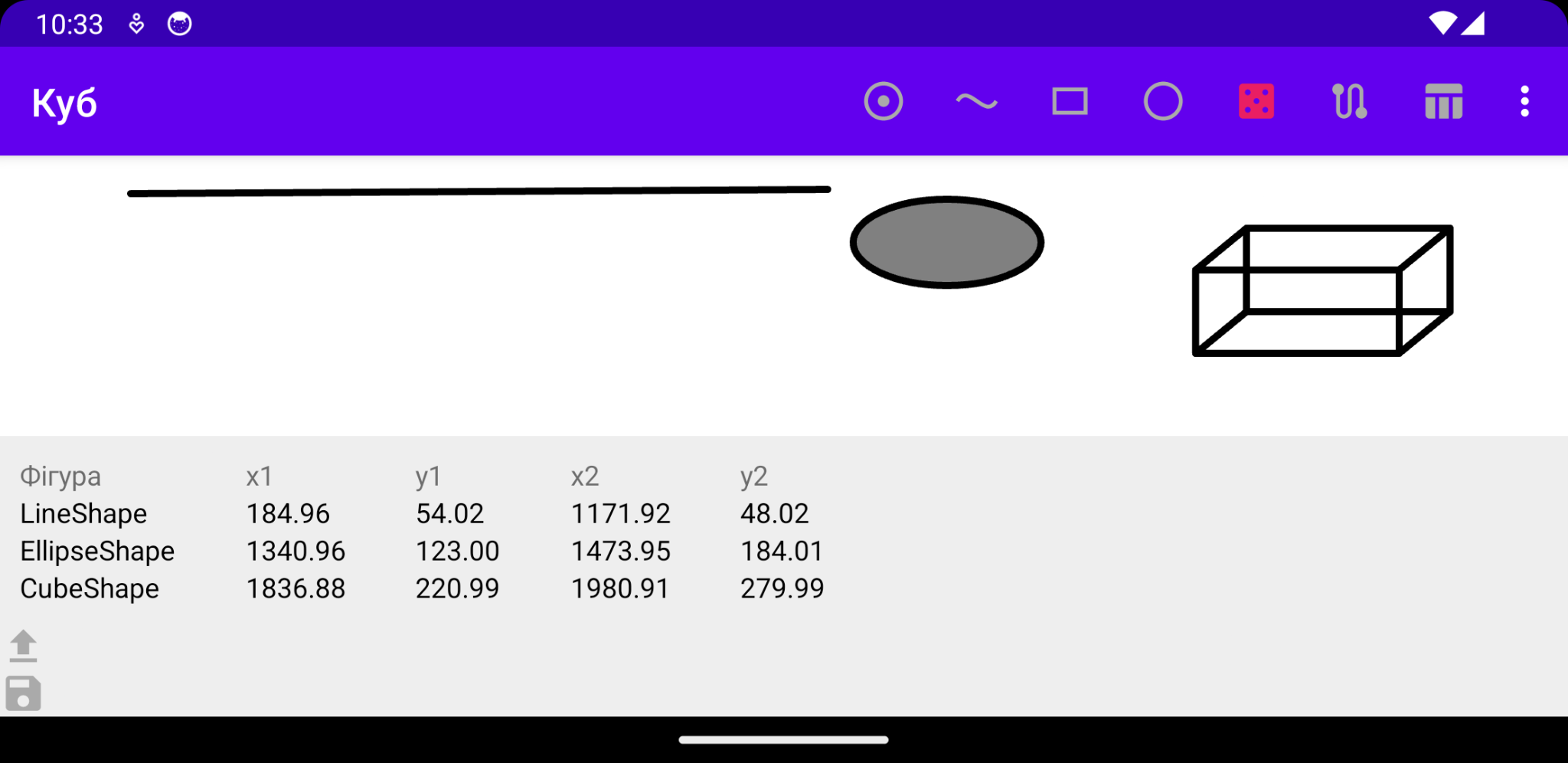
val shapeReference: Shape

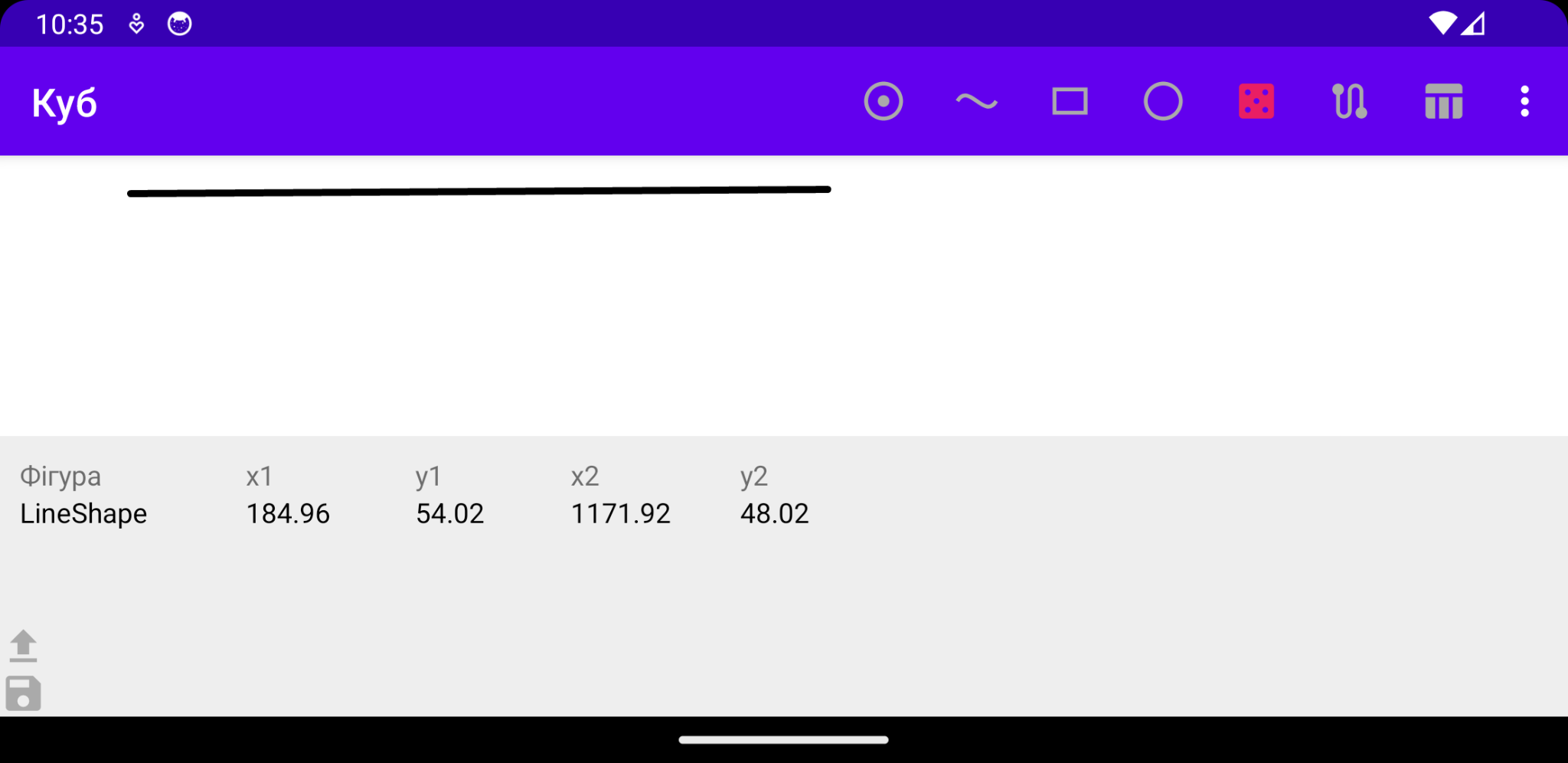
)

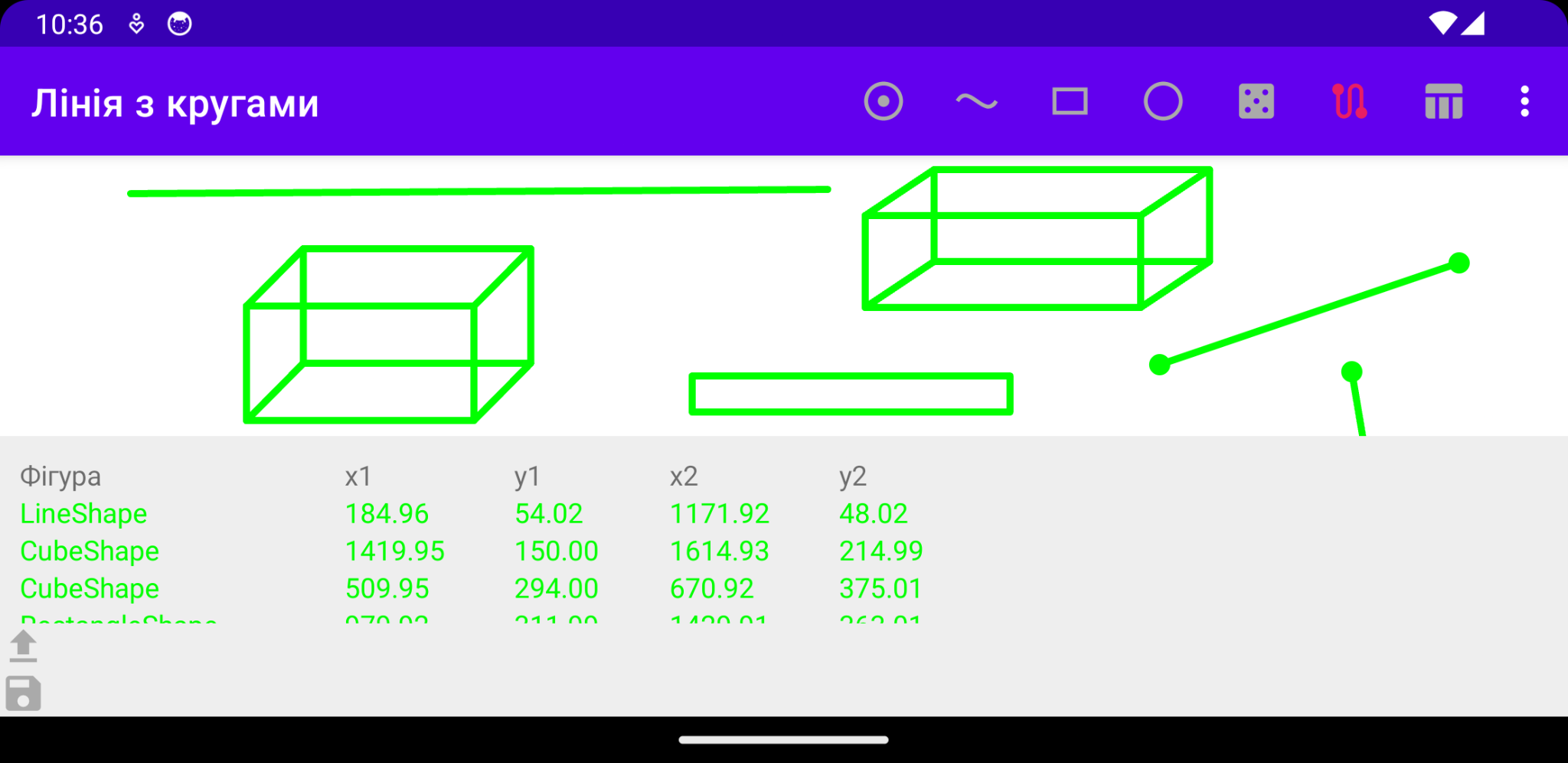
**Діаграма класів :**

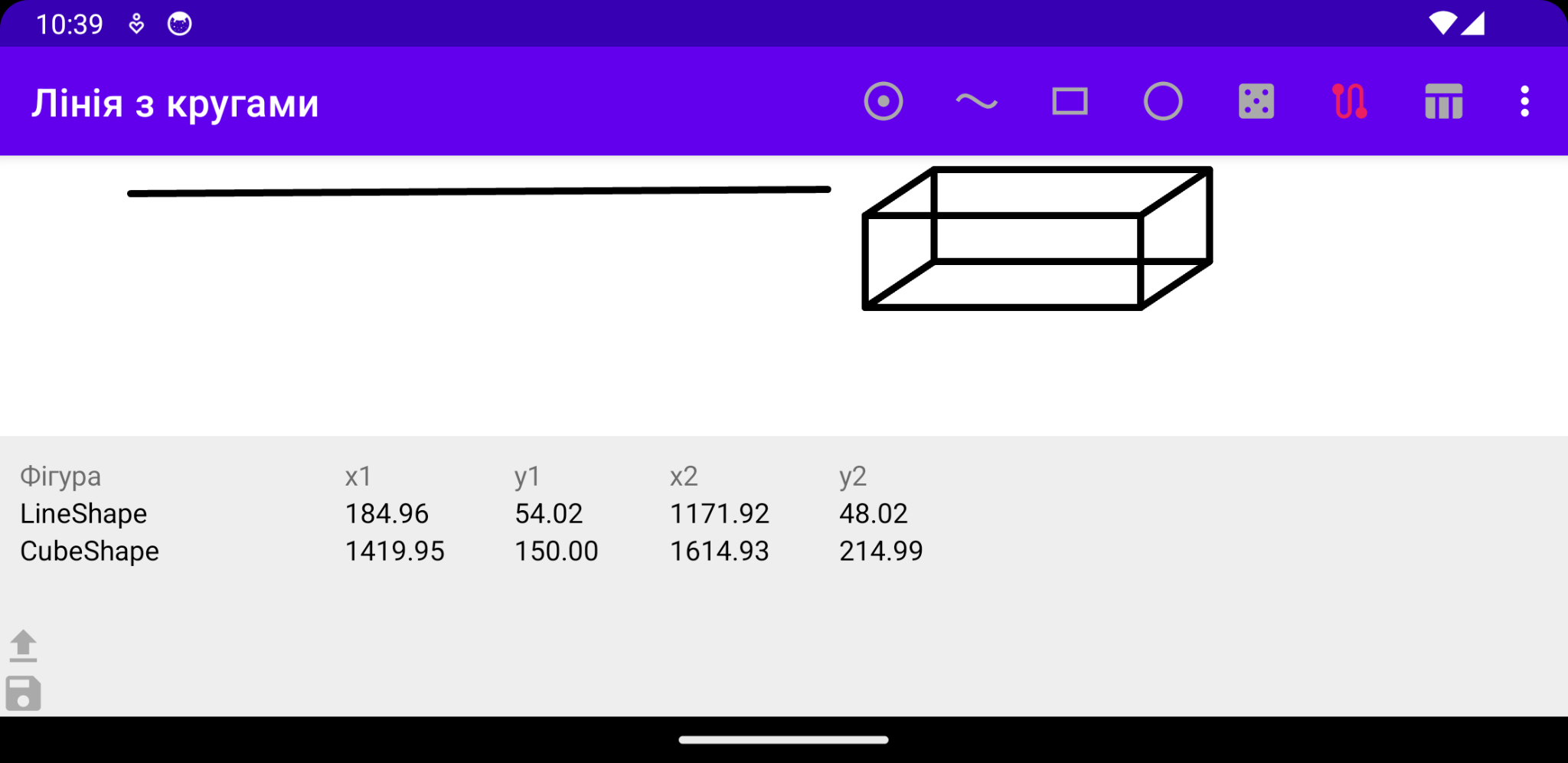
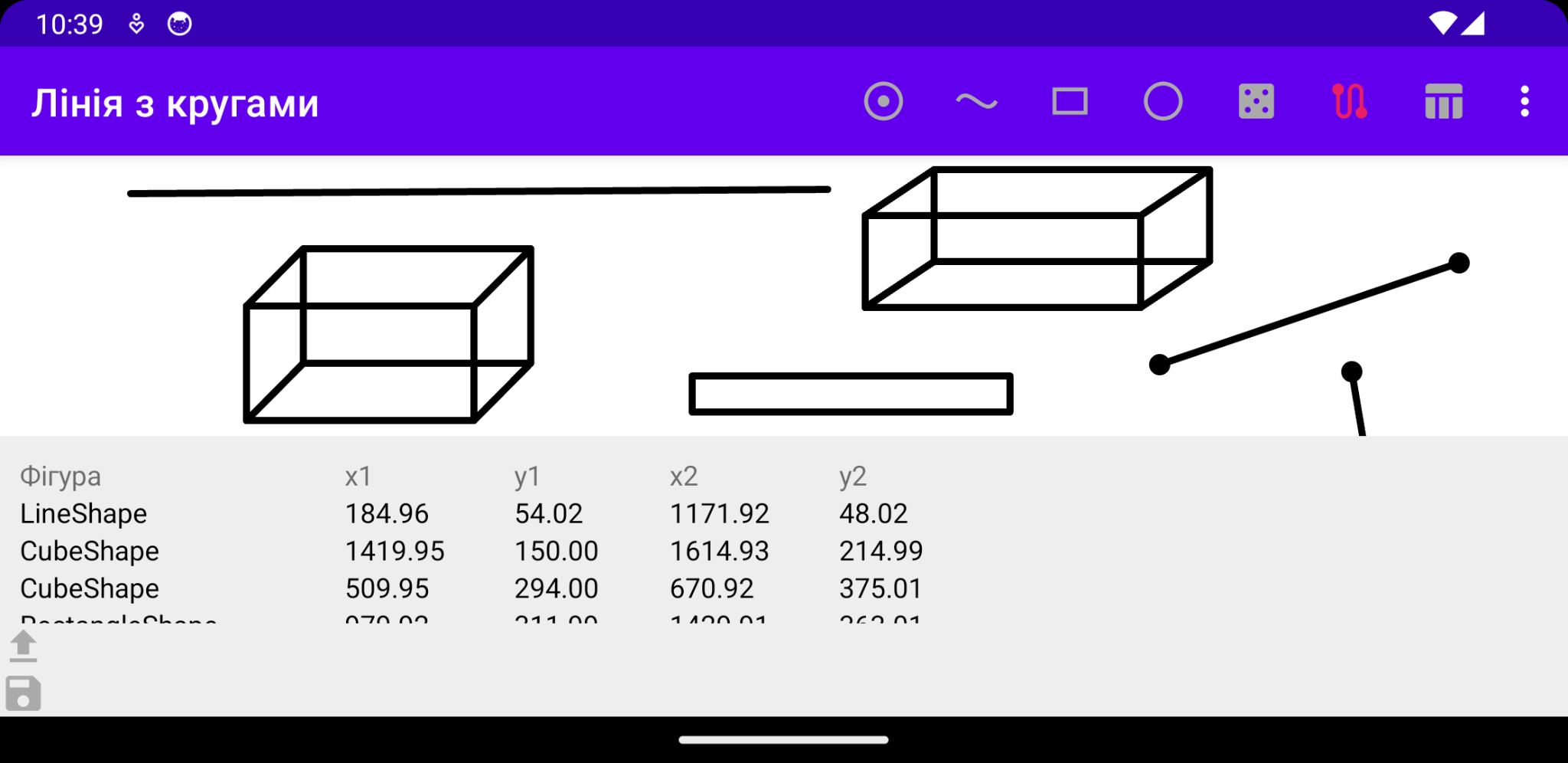


**Тестування програми (скріншоти)**

**Додавання в таблицю**

**Видалення з таблиці**

**Підсвітка фігур**

**Збереження та завантаження файлу**

**Висновки:** В ході виконання лабораторної роботи № 5 я розробляв графічний додаток для малювання різних фігур на екрані, який включає в себе таблицюз фігурами . У цій програмі широко використовується поліморфізм. Поліморфізм дозволяє об'єктам різних класів виконувати однакові дії, а також надає гнучкість і можливість розширення програми.

Крім того, був застосований патерн Singleton для класу MyEditor. Цей підхід дозволяє гарантувати, що у програмі існує тільки один екземпляр MyEditor , забезпечуючи централізований доступ до графічного редактора та його функціональності.

Клас MyTable виконує функцію керування та відображенням таблиці координат фігур в графічному редакторі. Реалізовані функції дозволяють вставляти нові рядки у таблицю, видаляти їх, зберігати та завантажувати дані з файлу. Також відображається заголовок таблиці з інформацією про тип та координати фігур.