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»

Виконав:

Студент групи ІМ-22
Тимофеєв Даниіл Костянтинович
номер у списку групи: 23

Перевірив:

Порєв В.М

Київ 2023

Мета: Мета роботи – отримати вміння та навички використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів C++, 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. 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 + 1, де Жлаб2 – номер студента в журналі, який використовувався для попередньої лаб. роботи No2 = 24

2. Статичний масив Shape *pcshape[N]; причому, кількість елементів масиву вказівників як для статичного, так і динамічного має бути N = 24+100 = 124.

3. “Гумовий” слід при вводі об’єкті - суцільна лінія чорного кольору для варіантів (Ж mod 4 = 0)

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

```
abstract class Shape (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
 }



 abstract fun draw (canvas: Canvas)



 abstract fun configureDrawing ()

}
```

Editor.kt

```
abstract class Editor {
  abstract fun handleMouseMovement (x: Float, y: Float)
  abstract fun onTouchUp ()
  abstract fun onTouchDown (x: Float, y: Float)

}
```

ShapeEditor.kt

```
abstract class ShapeEditor (paintSettings: Paint, shapesList:
MutableList<Shape>) : Editor() {
  val paint: Paint = paintSettings
  val shapes: MutableList<Shape> = shapesList
  private val shapesLimit: Int = 124
  protected fun addShapeToEditor (shape: Shape, shapes: MutableList<Shape>) {
    if (shapes.lastIndex == shapesLimit - 1) {
      shapes.removeAt(shapes.lastIndex)
    }
```

```
    shapes.add(shape)
  }
  override fun onTouchUp () {}

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

  override fun handleMouseMovement (x: Float, y: Float) {}
}
```

DotShape.kt

```
class DotShape (paintSettings: Paint) : Shape(paintSettings) {
  private val paint: Paint = paintSettings

  override fun draw (canvas: Canvas) {
    configureDrawing()
    paint.strokeWidth = 20f
    canvas.drawPoint(startXCoordinate, startYCoordinate, paint)
  }

  override fun configureDrawing () {
    paint.apply { color = Color.BLACK }
  }

}
```

EllipseShape.kt

```
class EllipseShape (paintSettings: Paint) : Shape(paintSettings) {
  private val paint = Paint()

  override fun draw (canvas: Canvas) {
    if (!isEraserMode) {
```

```
    configureFillStyle()
    canvas.drawOval(
      2*startXCoordinate - endXCoordinate,
      2*startYCoordinate - endYCoordinate,
      endXCoordinate,
      endYCoordinate,
      paint,
    )
  }

  configureDrawing()
  canvas.drawOval(
    2*startXCoordinate - endXCoordinate,
    2*startYCoordinate - endYCoordinate,
    endXCoordinate,
    endYCoordinate,
    paint,
  )
}

override fun configureDrawing () {
  paint.apply {
    this.color = Color.BLACK
    this.style = Paint.Style.STROKE
    this.strokeWidth = 20f
  }
}

private fun configureFillStyle () {
```

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

  private fun applyDrawingStyle (color: Int, style: Paint.Style) {
    paint.apply {
      this.color = color
      this.style = style
    }
  }
}
```

LineShape.kt

```
class LineShape (paintSettings: Paint) : Shape(paintSettings) {
  private val paint = Paint()

  override fun draw (canvas: Canvas) {
    configureDrawing()
    canvas.drawLine(startXCoordinate, startYCoordinate, endXCoordinate,
endYCoordinate, paint)
  }

  override fun configureDrawing () {
    paint.apply {
      color = Color.BLACK
      style = Paint.Style.FILL_AND_STROKE
      strokeWidth = 20f
    }
  }
}
```

RectangleShape.kt

```
class RectangleShape (initialPaintSettings: Paint) : Shape(initialPaintSettings) {
  private val paint = Paint()

  override fun draw (canvas: Canvas) {
    configureDrawing()
    val rect = RectF(startXCoordinate, startYCoordinate, endXCoordinate,
endYCoordinate)

    canvas.drawRect(rect, paint)
  }

  override fun configureDrawing () {
    paint.apply {
      color = Color.BLACK
      style = Paint.Style.STROKE
      strokeWidth = 20f
    }
  }
}
```

DotEditor.kt

```
class DotEditor (paintSettings: Paint, shapesList: MutableList<Shape>) :
ShapeEditor(paintSettings, shapesList) {
  private var currentDotShape: DotShape? = null

  override fun onTouchDown (x: Float, y: Float) {
    currentDotShape = DotShape(paint)
    currentDotShape?.defineStartCoordinates(x, y)
  }

  override fun onTouchUp () {
```

```
    currentDotShape?.let {
      addShapeToEditor(it, shapes)
      currentDotShape = null
    }
  }
}
```

EllipseEditor.kt

```
class EllipseEditor (private val paintSettings: Paint, private val shapesList:
MutableList<Shape>) : ShapeEditor(paintSettings, shapesList) {
  private var ellipseShape: EllipseShape? = null

  override fun onTouchDown (x: Float, y: Float) {
    ellipseShape = EllipseShape(paintSettings)
    ellipseShape?.defineStartCoordinates(x, y)
  }

  override fun onTouchUp () {
    ellipseShape?.let {
      if (shapesList.contains(it)) shapesList.remove(it)
      it.defineEraserMode(false)
      addShapeToEditor(it, shapesList)
    }
    ellipseShape = null
  }

  override fun handleMouseMovement (x: Float, y: Float) {
    ellipseShape?.let {
      if (shapesList.contains(it)) shapesList.remove(it)
      it.defineEndCoordinates(x, y)
```

```
      addShapeToEditor(it, shapesList)
    }
  }
}
```

LineEditor.kt

```
class LineEditor (private val paintSettings: Paint, private val shapesList:
MutableList<Shape>) : ShapeEditor(paintSettings, shapesList) {
  private var currentLine: LineShape? = null

  override fun onTouchDown (x: Float, y: Float) {
    currentLine = LineShape(paintSettings)
    currentLine?.defineStartCoordinates(x, y)
  }

  override fun onTouchUp () {
    currentLine?.let {
      addShapeToEditor(it, shapesList)
    }
    currentLine = null
  }

  override fun handleMouseMovement (x: Float, y: Float) {
    currentLine?.let { lineShape ->
      shapesList.remove(lineShape)
      lineShape.defineEndCoordinates(x, y)
      addShapeToEditor(lineShape, shapesList)
    }
  }
}
```

RectangleEditor.kt

```
class RectangleEditor (private val paintSettings: Paint, private val shapesList: MutableList<Shape>) : ShapeEditor(paintSettings, shapesList) {
  private var rectShape: RectangleShape? = null



  override fun onTouchDown (x: Float, y: Float) {
    rectShape = RectangleShape(paintSettings)
    rectShape!!.defineStartCoordinates(x, y)
  }

  override fun onTouchUp () {
    rectShape?.let {
      addShapeToEditor(it, shapesList)
    }
    rectShape = null
  }

  override fun handleMouseMovement (x: Float, y: Float) {
    rectShape?.let {
      shapesList.remove(it)

      it.defineEndCoordinates(x, y)
      addShapeToEditor(it, shapesList)
    }
  }
}
```

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 = 20f
  }

  private var drawingCanvas = Canvas()
  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 actualShape: ShapeEditor = DotEditor(drawingSetting, shapeList)


  override fun onDraw(canvas: Canvas?) {
    super.onDraw(canvas)
    shapeList.forEach { it.draw(canvas!!) }
  }
```

```
fun setShapePrimitiveEditor(primitiveEditor: ShapeEditor) {
  actualShape = primitiveEditor
}

private fun handleTouchUp() {
  invalidate()
  actualShape.onTouchUp()
}

private fun handleTouchMove() {
  invalidate()
  actualShape.handleMouseMovement(currentX, currentY)
}

private fun handleTouchStart() {
  invalidate()
  actualShape.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

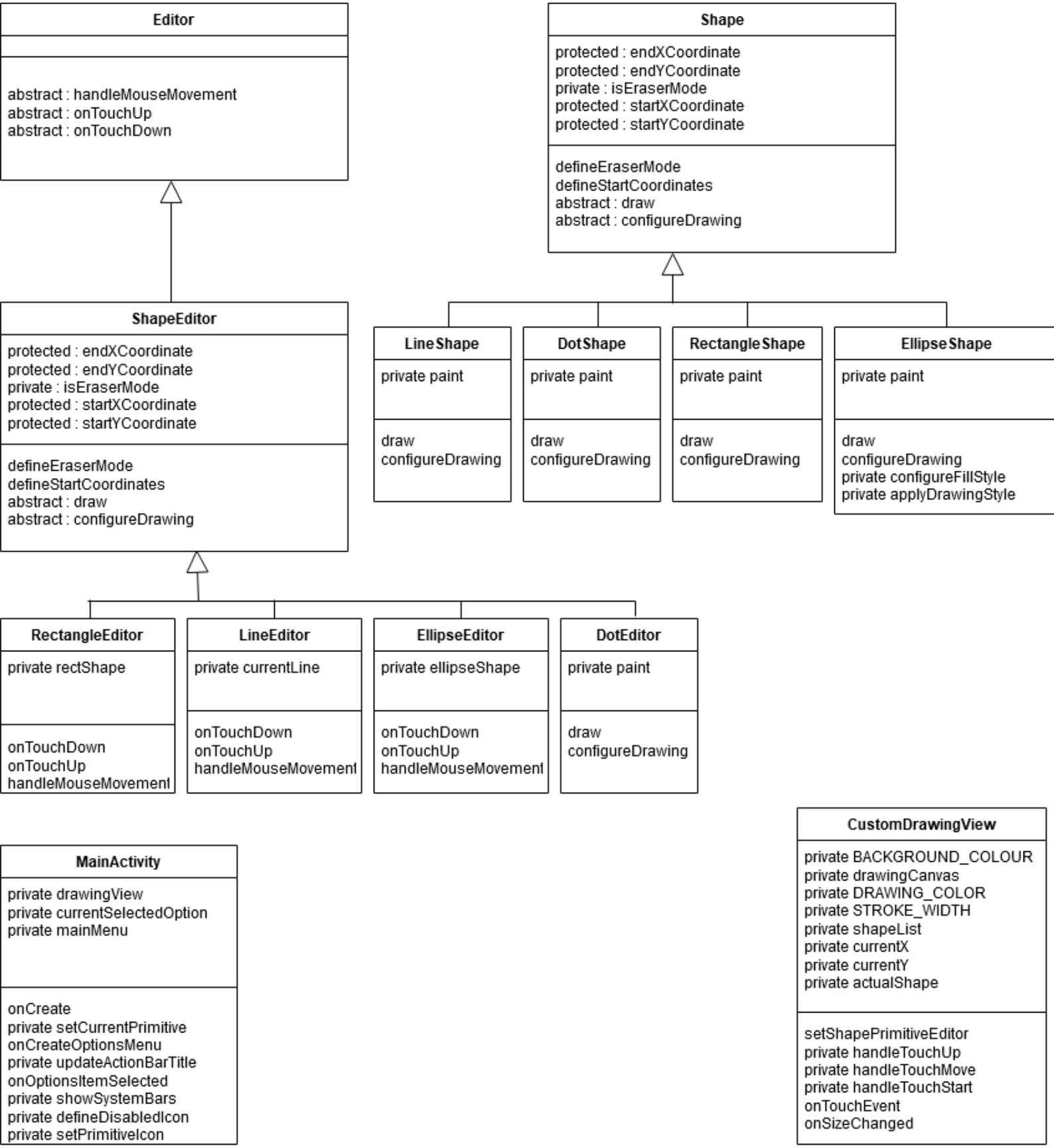
```
class MainActivity : AppCompatActivity() {
  private lateinit var drawingView: CustomDrawingView
  private lateinit var currentSelectedOption: MenuItem
  private lateinit var mainMenu: Menu

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    drawingView = CustomDrawingView(this)
    val initialEditor = DotEditor(drawingView.drawingSetting,
drawingView.shapeList)
    drawingView.setShapePrimitiveEditor(initialEditor)
    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)
    currentSelectedOption = mainMenu.findItem(item.itemId)
    val optionTitle = currentSelectedOption.title.toString()

    val primitiveEditor: ShapeEditor? = when (item.itemId) {
      R.id.ellipseIcon, R.id.ellipseSelect -> EllipseEditor(
        drawingView.drawingSetting,
        drawingView.shapeList
      )

      R.id.lineIcon, R.id.lineSelect -> LineEditor(
        drawingView.drawingSetting,
        drawingView.shapeList
      )

      R.id.dotIcon, R.id.dotSelect -> DotEditor(drawingView.drawingSetting,
drawingView.shapeList)
      R.id.rectangleIcon, R.id.rectangleSelect -> RectangleEditor(
        drawingView.drawingSetting,
        drawingView.shapeList
      )
```

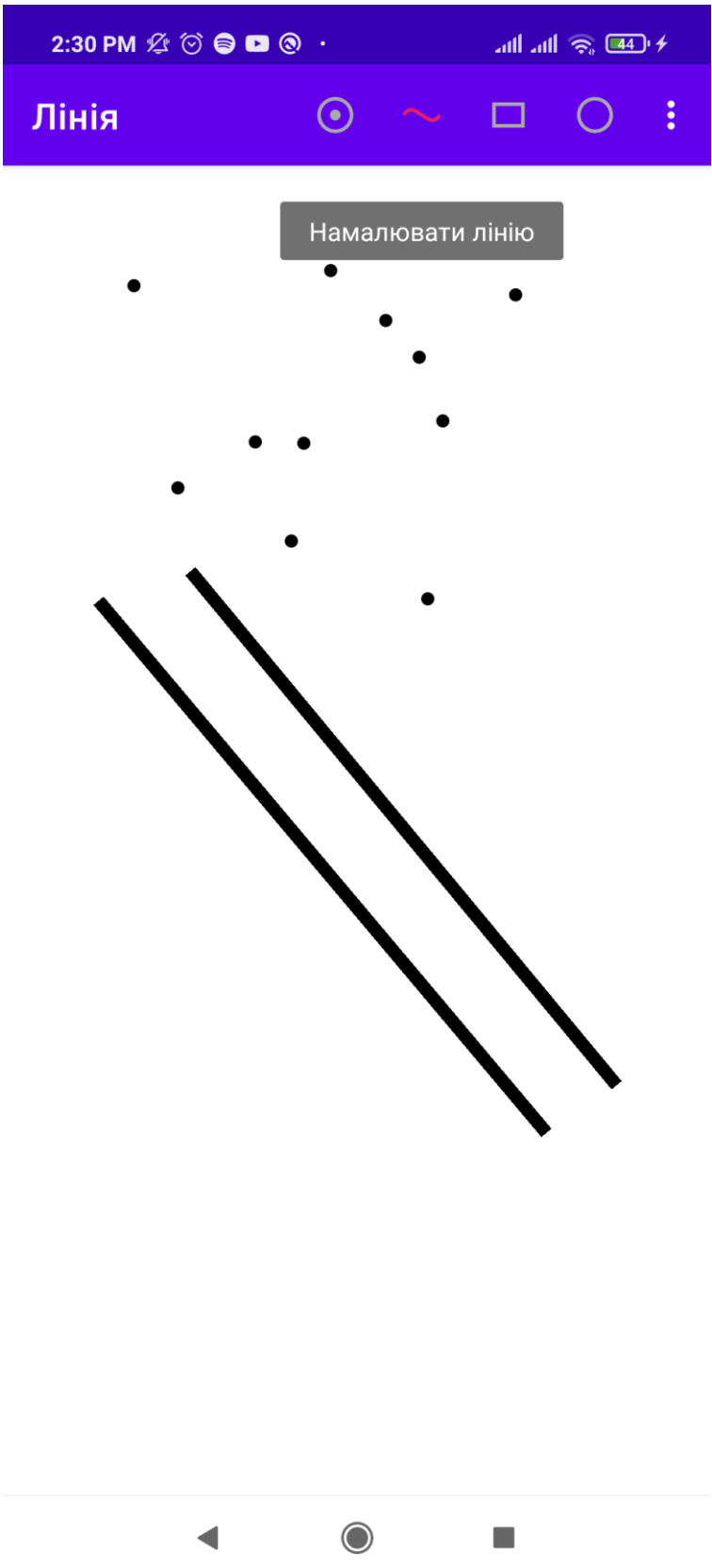
```
    else -> null
  }

  primitiveEditor?.let { editor ->
    item.isChecked = true
    mainMenu.findItem(R.id.ellipseSelect).isChecked = false
    mainMenu.findItem(R.id.lineSelect).isChecked = false
    mainMenu.findItem(R.id.rectangleSelect).isChecked = false
    mainMenu.findItem(R.id.dotSelect).isChecked = false

    val iconResourceId = when (editor) {
      is EllipseEditor -> R.drawable.ellipse
      is LineEditor -> R.drawable.line
      is DotEditor -> R.drawable.dot
      is RectangleEditor -> R.drawable.rectangle
      else -> 0
    }

    setCurrentPrimitive(editor, optionTitle, iconResourceId)
  }

  updateActionBarTitle(optionTitle)
  currentSelectedOption.isChecked = true

  return super.onOptionsItemSelected(item)
}


private fun showSystemBars() {
```

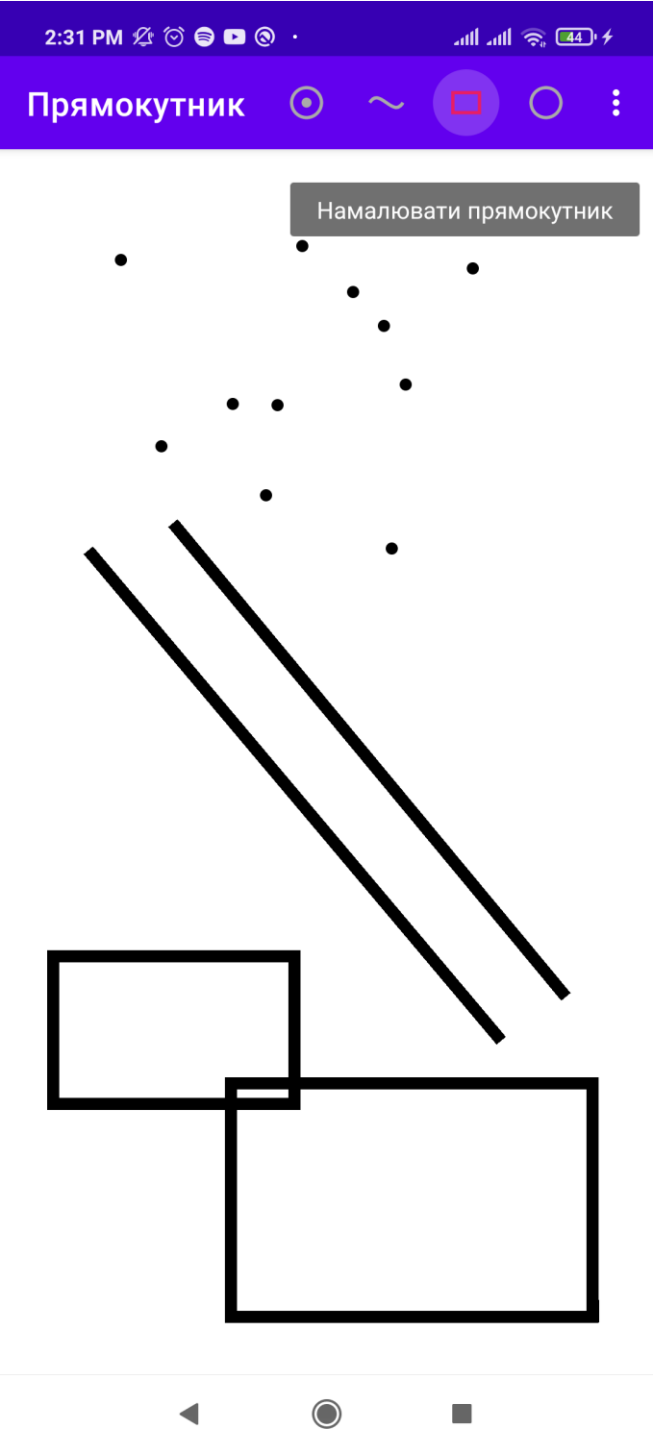
```
    WindowCompat.setDecorFitsSystemWindows(window, true)
    WindowInsetsControllerCompat(window,
drawingView).show(WindowInsetsCompat.Type.systemBars())
  }

  private fun setPrimitiveIcon(item: MenuItem, iconResourceId: Int) {
    item.icon = ContextCompat.getDrawable(this, iconResourceId)
  }

  private fun setCurrentPrimitive(primitive: ShapeEditor, title: String,
iconResourceId: Int) {
    drawingView.setShapePrimitiveEditor(primitive)
    currentSelectedOption.icon = ContextCompat.getDrawable(this,
iconResourceId)
    currentSelectedOption.title = title
  }

  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)
    }
  }

  private fun updateActionBarTitle(title: String) {
    supportActionBar?.title = title
  }
}
```

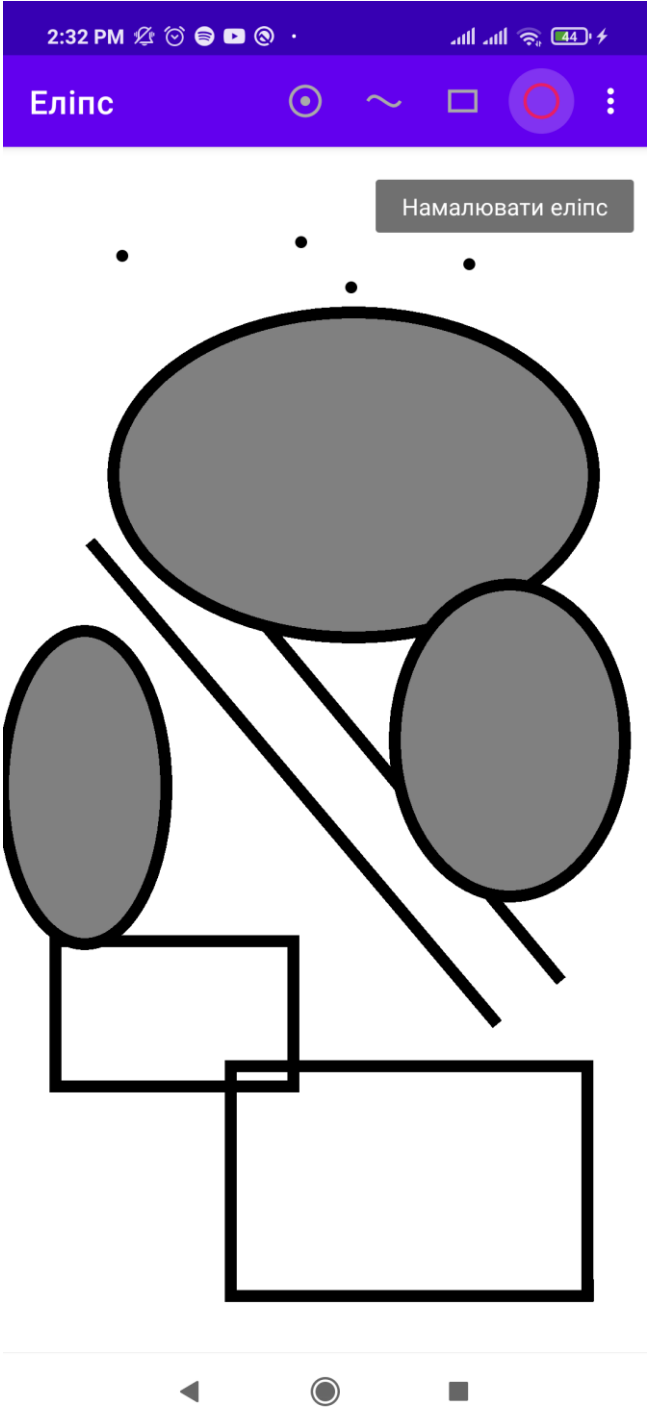
Діаграма класів :



Тестування програми (скріншоти)







Висновки: В ході виконання лабораторної роботи № 3 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.

Клас ShapeEditor також є абстрактним і містить методи, такі як onTouchDown, onTouchUp і handleMouseMovement, які реалізовані у конкретних редакторах (наприклад, DotEditor, LineEditor, EllipseEditor, RectangleEditor). Поліморфізм дозволяє використовувати об'єкти різних редакторів, які можуть обробляти події та взаємодіяти з користувачем через інтерфейс тулбара.