

**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»

Виконав:  
Студент групи ІМ-21  
Сірик Максим Олександрович  
номер у списку групи: 22

Перевірив:  
Порєв Віктор Миколайович

Київ 2023

# Зміст

<b>1</b>	<b>Мета:</b>	<b>2</b>
<b>2</b>	<b>Завдання:</b>	<b>2</b>
2.1	Варіанти завдань та основні вимоги . . . . .	2
<b>3</b>	<b>Текст програми:</b>	<b>3</b>
3.1	Module: com.github erotourtes.app . . . . .	3
3.2	Module: com.github erotourtes.view . . . . .	3
3.3	Module: com.github erotourtes.drawing . . . . .	5
3.4	Module: com.github erotourtes.drawing.editor . . . . .	5
3.5	Module: com.github erotourtes.drawing.shape . . . . .	8
3.6	Module: com.github erotourtes.utils . . . . .	9
<b>4</b>	<b>Ілюстрації:</b>	<b>12</b>
<b>5</b>	<b>Висновки:</b>	<b>12</b>

# 1 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.

## 2 Завдання:

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

### 2.1 Варіанти завдань та основні вимоги

1. Для усіх варіантів завдань необхідно дотримуватися вимог та положень, викладених вище у порядку виконання роботи та методичних рекомендаціях.
2. У звіті повинна бути схема успадкування класів – діаграма класів
3. Для вибору типу об'єкта в графічному редакторі Lab2 повинно бути меню "Об'єкти" з чотирма підпунктами. Меню "Об'єкти" повинно бути праворуч меню "Файл" та ліворуч меню "Довідка". Підпункти меню "Об'єкти" містять назви українською мовою геометричних форм – так, як наведено вище у порядку виконання роботи та методичних рекомендаціях. Геометричні форми згідно варіанту завдання.
4. Для вибору варіанту використовується Ж – номер студента в журналі.
5. Масив вказівників для динамічних об'єктів типу Shape - динамічний масив Shape `**pcshape`; - статичний масив Shape `*pcshape[N]`; причому, кількість елементів масиву вказівників як для статичного, так і динамічного має бути  $N = Ж + 100$ . Динамічний масив обирають студенти, у яких варіант  $(Ж \bmod 3 = 0)$ . Решта студентів – статичний масив. Позначка `mod` означає залишок від ділення.
6. "Гумовий" слід при вводі об'єктів - суцільна лінія чорного кольору для варіантів  $(Ж \bmod 4 = 0)$ 
  - суцільна лінія червоного кольору для варіантів  $(Ж \bmod 4 = 1)$
  - суцільна лінія синього кольору для варіантів  $(Ж \bmod 4 = 2)$
  - пунктирна лінія чорного кольору для варіантів  $(Ж \bmod 4 = 3)$
7. Чотири геометричні форми (крапка, лінія, прямокутник, еліпс) можуть мати наступні різновиди вводу та відображення.
  - 7.1. Прямокутник Увід прямокутника: - по двом протилежним кутам для варіантів  $(Ж \bmod 2 = 0)$  - від центру до одного з кутів для варіантів  $(Ж \bmod 2 = 1)$  Відображення прямокутника: - чорний контур з білим заповненням для  $(Ж \bmod 5 = 0)$  - чорний контур з кольоровим заповненням для  $(Ж \bmod 5 = 1 \text{ або } 2)$  - чорний контур прямокутника без заповнення для  $(Ж \bmod 5 = 3 \text{ або } 4)$  Кольори заповнення прямокутника: - жовтий для  $(Ж \bmod 6 = 0)$  - світло-зелений для  $(Ж \bmod 6 = 1)$  - блакитний для  $(Ж \bmod 6 = 2)$  - рожевий для  $(Ж \bmod 6 = 3)$  - померанчевий для  $(Ж \bmod 6 = 4)$  - сірий для  $(Ж \bmod 6 = 5)$
  - 7.2. Еліпс Увід еліпсу: - по двом протилежним кутам охоплюючого прямокутника для варіантів  $(Ж \bmod 2 = 1)$  - від центру до одного з кутів охоплюючого прямокутника для варіантів  $(Ж \bmod 2 = 0)$  Відображення еліпсу: - чорний контур з білим заповненням для  $(Ж \bmod 5 = 1)$  - чорний контур з кольоровим заповненням для  $(Ж \bmod 5 = 3 \text{ або } 4)$  - чорний контур еліпсу без заповнення для  $(Ж \bmod 5 = 0 \text{ або } 2)$  Кольори заповнення еліпсу: - жовтий для  $(Ж \bmod 6 = 1)$  - світло-зелений для  $(Ж \bmod 6 = 2)$  - блакитний для  $(Ж \bmod 6 = 3)$  - рожевий для  $(Ж \bmod 6 = 4)$  - померанчевий для  $(Ж \bmod 6 = 5)$  - сірий для  $(Ж \bmod 6 = 0)$
8. Позначка поточного типу об'єкту, що вводиться - в меню (метод `OnInitMenuPopup`) для варіантів  $(Ж \bmod 2 = 0)$  - в заголовку вікна для  $(Ж \bmod 2 = 1)$

9. Приклад вибору варіанту. Для 9-го студента у списку ( $Ж = 9$ ) буде: - динамічний масив для Shape ( $9 \bmod 3 = 0$ ) обсягом 109 об'єктів - "гумовий" слід ( $9 \bmod 4 = 1$ ) - суцільна лінія червоного кольору - прямокутник: - ввід від центру до одного з кутів ( $9 \bmod 2 = 1$ ) - чорний контур прямокутника без заповнення ( $9 \bmod 5 = 4$ ) - еліпс: - по двом протилежним кутам охопл. прямокутника ( $9 \bmod 2 = 1$ ) - чорний контур з кольоровим заповненням ( $9 \bmod 5 = 4$ ) - колір заповнення: блакитний ( $9 \bmod 6 = 3$ ) - позначка поточного типу об'єкту: в заголовку вікна ( $9 \bmod 2 = 1$ ) Примітка. Визначення кольорів та інші параметри варіантів можуть бути змінені викладачем шляхом оголошення студентам відповідного повідомлення завчасно перед постановкою завдань.

## 3 Текст програми:

### 3.1 Module: com.github.erotourtes.app

Лістинг 1: MyApp.kt

```
package com.github.erotourtes.app

import com.github.erotourtes.view.MainView
import tornadofx.App

class MyApp: App(MainView::class)
```

### 3.2 Module: com.github.erotourtes.view

Лістинг 2: MainView.kt

```
package com.github.erotourtes.view

import com.github.erotourtes.drawing.CanvasHandler
import com.github.erotourtes.drawing.CanvasPane
import com.github.erotourtes.drawing.editor.*
import com.github.erotourtes.utils.MenuItemInfo
import javafx.scene.control.ToggleGroup
import tornadofx.*

class MainView : View("Lab2") {
    private val canvasHandler = CanvasHandler()

    override val root = borderpane {
        top = MenuBar(createMenu())
        center = CanvasPane(canvasHandler.canvas)
    }

    private fun createMenu(): List<MenuItemInfo> {
        val group = ToggleGroup()
        val menuList = with(canvasHandler) {
            listOf(
                MenuItemInfo("точка", { useEditor<PointEditor>() }, group, true),
                MenuItemInfo("лінія", { useEditor<LineEditor>() }, group),
                MenuItemInfo("прямокутник", { useEditor<RectEditor>() }, group),
                MenuItemInfo("еліпс", { useEditor<EllipseEditor>() }, group),
            )
        }
    }
}
```

```

        menuList.find { it.selected }?.action?.let { it() }

        return menuList
    }
}

```

Лістинг 3: MainView.kt

```

package com.github.erotourtes.view

```

```

import com.github.erotourtes.utils.MenuItemInfo
import javafx.scene.control.MenuBar
import tornadofx.*

```

```

const val g = 22
const val n = 100 + g

```

```

class MenuBar(shapes : List<MenuItemInfo>) : MenuBar() {

```

```

    init {
        menu("Файл") {
            item("New")
            item("Open")
            item("Save")
            item("Save_as")
            separator()
            item("Exit")
        }
        menu("Об'єкти") {
            for (shape in shapes) {
                radiomenuitem(shape.name, shape.group) {
                    action(shape.action)
                    isSelected = shape.selected
                }
            }
        }
        menu("Довідка") {
            item(
                """

```

```

0) Ж_=$g
1) Статичний масив (Ж_mod_3!=0) обсягом $g+100=$n.
2) "Гумовий" слід при вводі об'єктів - суцільна лінія синього кольору
3) Позначка поточного типу об'єкту, що вводиться - в меню (метод OnInit)
4) Прямокутник:
    по двом протилежним кутам для варіантів (Ж_mod_2=0)
    чорний контур з кольоровим заповненням для (Ж_mod_5=1 або 2)
    померанчевий для (Ж_mod_6=4)
5) Еліпс
    від центру до одного з кутів охоплюючого прямокутника для варіантів
    чорний контур еліпсу без заповнення для (Ж_mod_5=0 або 2)
    рожевий для (Ж_mod_6=4)
        """.trimIndent()
    )
        }
    }
}

```

### 3.3 Module: com.github.erotourtes.drawing

Лістинг 4: CanvasHandler.kt

```
package com.github.erotourtes.drawing

import com.github.erotourtes.view.n
import com.github.erotourtes.drawing.editor.Editor
import com.github.erotourtes.drawing.editor.ShapesList
import javafx.scene.canvas.Canvas
import javafx.scene.canvas.GraphicsContext

class CanvasHandler {
    val canvas = Canvas()
    private val shapes = ShapesList(n)
    private var map: MutableMap<Class<out Editor>, Editor> = mutableMapOf()

    inline fun <reified T : Editor> useEditor() {
        val editorClass = T::class.java as Class<out Editor>
        val editor = getOrCreateEditor(editorClass)
        editor.listenToEvents()
    }

    fun getOrCreateEditor(editorClass: Class<out Editor>): Editor = map.getOrPut(editorClass, {
        editorClass.getConstructor(ShapesList::class.java, GraphicsContext::class.java)
            .newInstance(shapes, canvas.graphicsContext2D)
    })
}
```

Лістинг 5: CanvasPane.kt

```
package com.github.erotourtes.drawing

import javafx.scene.canvas.Canvas
import javafx.scene.layout.Pane

class CanvasPane(canvas: Canvas) : Pane() {
    init {
        children.add(canvas)

        canvas.widthProperty().bind(this.widthProperty())
        canvas.heightProperty().bind(this.heightProperty())
    }
}
```

### 3.4 Module: com.github.erotourtes.drawing.editor

Лістинг 6: Editor.kt

```
package com.github.erotourtes.drawing.editor

import com.github.erotourtes.drawing.shape.*
import javafx.scene.paint.Color
import com.github.erotourtes.utils.*
import javafx.scene.canvas.GraphicsContext
import javafx.scene.input.MouseEvent
```

```

abstract class Editor(private val shapes: ShapesList, protected val gc: GraphicsContext) {
    protected val dm = Dimension()
    protected abstract val shape: Shape

    open fun listenToEvents() {
        val c = gc.canvas
        c.setOnMousePressed(this::onMousePressed)
        c.setOnMouseDragged(this::onMouseDragged)
        c.setOnMouseReleased(this::onMouseReleased)
    }

    protected open fun onMousePressed(e: MouseEvent) {
        clear()
        drawAll()
        dm.setStart(e.x, e.y)
    }

    protected open fun onMouseDragged(e: MouseEvent) {
        clear()
        drawAll()

        dm.setEnd(e.x, e.y)

        drawShowLine()
        shape.draw(dm)
    }

    protected open fun onMouseReleased(e: MouseEvent) {
        shapes.add(shape.copy())
    }

    private fun drawAll() {
        for (shape in shapes) shape.draw()
    }

    private fun clear() = gc.clearRect(0.0, 0.0, gc.canvas.width, gc.canvas.height)

    protected open fun drawShowLine() {
        gc.drawOnce {
            gc.stroke = Color.BLUE
            gc.strokeRect(dm)
        }
    }
}

```

Лістинг 7: Editors.kt

```

package com.github.erotourtes.drawing.editor

import com.github.erotourtes.drawing.shape.*
import com.github.erotourtes.utils.*
import javafx.scene.canvas.GraphicsContext
import javafx.scene.input.MouseEvent
import javafx.scene.paint.Color

```

```

class PointEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Point(gc)

    override fun onMouseDragged(e: MouseEvent) {}

    override fun onMousePressed(e: MouseEvent) {
        dm.setEnd(e.x, e.y)
        super.onMousePressed(e)
        shape.draw(dm)
    }
}

class LineEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Line(gc)
}

class RectEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Rect(gc)
    override fun drawShowLine() {}
}

class EllipseEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Ellipse(gc)
    override fun drawShowLine() = gc.drawOnce {
        gc.stroke = Color.BLUE
        strokeRect(getEllipseDimensions(dm))
    }
}

```

#### ЛІСТИНГ 8: ShapesList.kt

```

package com.github.erotourtes.drawing.editor

import com.github.erotourtes.drawing.shape.Shape

class ShapesList(n: Int) : Iterable<Shape> {
    private val shapeArr = Array<Shape?>(n) { null }
    private var shapeIndex = 0

    val size: Int
        get() = shapeIndex

    fun add(sh: Shape) {
        if (shapeIndex == shapeArr.size) throw IllegalArgumentException("History is overflowed")
        shapeArr[shapeIndex++] = sh
    }

    override fun iterator(): Iterator<Shape> = ShapeIterator()

    inner class ShapeIterator : Iterator<Shape> {
        private var curIndex = 0
        override fun hasNext(): Boolean = curIndex < size
        override fun next(): Shape = if (hasNext()) shapeArr[curIndex++]!! else throw Illegal
    }
}

```



```
}
```

### 3.5 Module: com.github.erotourtes.drawing.shape

Лістинг 9: Shape.kt

```
package com.github.erotourtes.drawing.shape

import com.github.erotourtes.utils.Dimension
import javafx.scene.canvas.GraphicsContext
import javafx.scene.paint.Color

abstract class Shape(val gc: GraphicsContext) {
    val dm = Dimension()

    var colorFill: Color = Color.BLACK
    var colorStroke: Color = Color.BLACK

    abstract fun draw()

    open fun draw(curDm: Dimension) {
        curDm.copyTo(dm)
        draw()
    }

    fun setProperties() {
        with(gc) {
            fill = colorFill
            stroke = colorStroke
        }
    }

    fun copy(): Shape {
        val shape = this::class.java.getConstructor(GraphicsContext::class.java).newInstance()
        shape.dm.copyFrom(dm)
        shape.colorFill = colorFill
        shape.colorStroke = colorStroke
        return shape
    }
}
```

Лістинг 10: Shapes.kt

```
package com.github.erotourtes.drawing.shape

import com.github.erotourtes.utils.*
import javafx.scene.canvas.GraphicsContext
import javafx.scene.paint.Color

class Point(gc: GraphicsContext) : Shape(gc) {
    override fun draw() {
        val radius = 12.0
        gc.drawOnce {
            setProperties()
            val (x, y) = dm.getBoundaries().first
        }
    }
}
```

```

        fillOval(x, y, radius, radius)
    }
}

class Line(gc: GraphicsContext) : Shape(gc) {
    override fun draw() {
        gc.drawOnce {
            setProperties()
            strokeLine(dm)
        }
    }
}

class Rect(gc: GraphicsContext) : Shape(gc) {
    init {
        colorFill = Color.ORANGE
        colorStroke = Color.BLACK
    }

    override fun draw() {
        gc.drawOnce {
            setProperties()
            fillRect(dm)
            strokeRect(dm)
        }
    }
}

class Ellipse(gc: GraphicsContext) : Shape(gc) {
    init {
        colorFill = Color.TRANSPARENT
        colorStroke = Color.BLACK
    }

    override fun draw() {
        gc.drawOnce {
            setProperties();
            fillOval(dm)
            strokeOval(dm)
        }
    }

    override fun draw(curDm: Dimension) {
        dm.copyFrom(getEllipseDimensions(curDm))
        draw()
    }
}

```

### 3.6 Module: com.github.erotourtes.utils

Лістинг 11: Dimension.kt

```
package com.github.erotourtes.utils
```

```

import kotlin.math.abs

class Dimension {
    private var x1: Double = 0.0
    private var y1: Double = 0.0
    private var x2: Double = 0.0
    private var y2: Double = 0.0

    val width: Double
        get() = abs(x2 - x1)
    val height: Double
        get() = abs(y2 - y1)

    fun setStart(x: Double, y: Double): Dimension {
        x1 = x
        y1 = y
        return this
    }

    fun setEnd(x: Double, y: Double): Dimension {
        x2 = x
        y2 = y
        return this
    }

    fun copyTo(dst: Dimension) {
        dst.x1 = x1
        dst.y1 = y1
        dst.x2 = x2
        dst.y2 = y2
    }

    fun copyFrom(src: Dimension) = src.copyTo(this)

    fun getBoundaries(): Pair<Point, Point> {
        return Pair(
            Point(x1.coerceAtMost(x2), y1.coerceAtMost(y2)),
            Point(x1.coerceAtLeast(x2), y1.coerceAtLeast(y2))
        )
    }

    fun getRaw(): Pair<Point, Point> {
        return Pair(
            Point(x1, y1),
            Point(x2, y2)
        )
    }

    data class Point(val x: Double, val y: Double)
}

```

Лістинг 12: ExtensionFunctions.kt

```
package com.github.erotourtes.utils
```

```

import javafx.scene.canvas.GraphicsContext

fun GraphicsContext.fillRect(dm: Dimension) {
    val (s, e) = dm.getBoundaries()
    fillRect(s.x, s.y, e.x - s.x, e.y - s.y)
}

fun GraphicsContext.strokeRect(dm: Dimension) {
    val (s, e) = dm.getBoundaries()
    strokeRect(s.x, s.y, e.x - s.x, e.y - s.y)
}

fun GraphicsContext.fillOval(dm: Dimension) {
    val (s, e) = dm.getBoundaries()
    fillOval(s.x, s.y, e.x - s.x, e.y - s.y)
}

fun GraphicsContext.strokeOval(dm: Dimension) {
    val (s, e) = dm.getBoundaries()
    strokeOval(s.x, s.y, e.x - s.x, e.y - s.y)
}

fun GraphicsContext.strokeLine(dm: Dimension) {
    val (s, e) = dm.getRaw()
    strokeLine(s.x, s.y, e.x, e.y)
}

inline fun GraphicsContext.drawOnce(lambda: GraphicsContext.() -> Unit) {
    val oldFill = fill
    val oldStroke = stroke
    lambda(this)
    fill = oldFill
    stroke = oldStroke
}

```

Лістинг 13: Utils.kt

```

package com.github.erotourtes.utils

import javafx.scene.control.ToggleGroup

data class MenuItemInfo(
    val name: String,
    val action: () -> Unit,
    val group: ToggleGroup? = null,
    var selected: Boolean = false
)

fun getEllipseDimensions(dm: Dimension): Dimension {
    val w = dm.width
    val h = dm.height
    val (cx, cy) = dm.getRaw().first

    val sX = cx - w

```

