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-21
Сірик Максим Олександрович
номер у списку групи: 22

Перевірив:
Порєв Віктор Миколайович

Київ 2023

Зміст

1 Мета:

Мета роботи – отримати вміння та навички використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів C++, запрограмувавши графічний інтерфейс користувача.

2 Завдання:

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім'ям Lab3.
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.

2.1 Варіант:

Варіанти завдань та основні вимоги
1. У звіті повинна бути схема успадкування класів – діаграма класів
2. Усі методи-обробники повідомлень, зокрема, і метод OnNotify, повинні бути функціями-членами деякого класу (класів).
3. Для вибору типу об'єкту в графічному редакторі Lab3 повинно бути вікно Toolbar з кнопками відповідно типам об'єктів. Кнопки дублюють підпункти меню "Об'єкти". Кнопки мають бути з підказками (tooltips). Меню "Об'єкти"повинно бути праворуч меню "Файл"та ліворуч меню "Довідка". Підпункти меню "Об'єкти"містять назви геометричних форм українською мовою. Геометричні форми згідно варіанту завдання.
4. Для вибору варіанту використовується значення Ж = Жлаб2 + 1, де
Жлаб2 – номер студента в журналі, який використовувався для попередньої лаб. роботи No2.

$$J = 22 + 1 \tag{1}$$

$$J = 23 \tag{2}$$

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. $N = 23 + 100$
Динамічний масив обирають студенти, у яких варіант (Ж mod 3 = 0).
Решта студентів роблять статичний масив. Примітка. Позначка mod означає залишок від ділення. 23 mod 3 = 2
6. "Гумовий"слід при вводі об'єктів
- суцільна лінія чорного кольору для варіантів (Ж mod 4 = 0)
- суцільна лінія червоного кольору для (Ж mod 4 = 1)
- суцільна лінія синього кольору для (Ж mod 4 = 2)
- пунктирна лінія чорного кольору для (Ж mod 4 =3) 23 mod 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 mod 2 = 0)
- від центру до одного з кутів для (Ж mod 2 = 1) 23 mod 2 = 1
Відображення прямокутника:

- чорний контур з білим заповненням для (Ж mod 5 = 0)
- чорний контур з кольоровим заповненням для (Ж mod 5 = 1 або 2)
- чорний контур прямокутника без заповнення для (Ж mod 5 = 3 або 4) 23 mod 5 = 3
Кольори заповнення прямокутника:
- жовтий для (Ж mod 6 = 0)
- світло-зелений для (Ж mod 6 = 1)
- блакитний для (Ж mod 6 = 2)
- рожевий для (Ж mod 6 = 3)
- помаранчевий для (Ж mod 6 = 4)
- сірий для (Ж mod 6 = 5) 23 mod 6 = 5
7.2. Еліпс
Ввід еліпсу:
- по двом протилежним кутам охоплюючого прямокутника для варіантів (Ж mod 2 = 1) 23 mod 2 = 1
- від центру до одного з кутів охоплюючого прямокутника для варіантів (Ж mod 2 = 0)
Відображення еліпсу:
- чорний контур з білим заповненням для (Ж mod 5 = 1)
- чорний контур з кольоровим заповненням для (Ж mod 5 = 3 або 4) 23 mod 5 = 3
- чорний контур еліпсу без заповнення для (Ж mod 5 = 0 або 2)
Кольори заповнення еліпсу:
- жовтий для (Ж mod 6 = 1)
- світло-зелений для (Ж mod 6 = 2)
- блакитний для (Ж mod 6 = 3)
- рожевий для (Ж mod 6 = 4)
- помаранчевий для (Ж mod 6 = 5) 23 mod 6 = 5
- сірий для (Ж mod 6 = 0)
8. Позначка поточного типу об'єкту, що вводиться
- в меню (метод OnInitMenuPopup) для варіантів (Ж mod 2 = 0)
- в заголовку вікна для (Ж mod 2 = 1) 23 mod 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("Lab3") {
    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: MenuBar.kt

```
package com.github.erotourtes.view

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

const val g = 22 + 1
const val n = 100 + g

class MenuBar(shapes: List<MenuItemInfo>) : MenuBar() {
    init {
        menu("Файл") {
            item("New...")
            item("Open...")
            item("Save␣as...")
            separator()
            item("Print")
            separator()
            item("Exit")
        }
        menu("Objects") {
            for (shape in shapes) {
                radiomenuitem(shape.name, shape.group) {
                    action(shape.action)
```

```
                    isSelected = shape.selected
                }
            }
        }
        menu("Help") {
            item(
                """
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣0)␣Ж␣=␣$g
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣1)␣Статичнии␣масив␣(Ж␣mod␣3␣!=␣0)␣обсягом␣$g␣+␣100␣=␣$n.
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣2)␣"Гумовии"␣слід␣при␣вводі␣об'єктів␣-␣пунктирна␣лінія␣чорного␣кольору␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣3)␣Прямокутник:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Увід␣прямокутника:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣від␣центру␣до␣одного␣з␣кутів␣для␣(Ж␣mod␣2␣=␣1)␣g␣%␣2␣=␣${g␣%␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Відображення␣прямокутника:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣чорнии␣контур␣прямокутника␣без␣заповнення␣для␣(Ж␣mod␣5␣=␣3␣а
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Кольори␣заповнення␣прямокутника:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣сірии␣для␣(Ж␣mod␣6␣=␣5)␣g␣%␣6␣=␣${g␣%␣6}
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣4)␣Еліпс:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Ввід␣еліпсу:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣по␣двом␣протилежним␣кутам␣охоплюючого␣прямокутника␣для␣варіан
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Відображення␣еліпсу:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣чорнии␣контур␣з␣кольоровим␣заповненням␣для␣(Ж␣mod␣5␣=␣3␣або␣4
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Кольори␣заповнення␣еліпсу:
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣померанчевии␣для␣(Ж␣mod␣6␣=␣5)␣g␣%␣6␣=␣${g␣%␣6}
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣5)␣Позначка␣поточного␣типу␣об'єкту,␣що␣вводиться
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣-␣в␣заголовку␣вікна␣для␣(Ж␣mod␣2␣=␣1)␣g␣%␣2␣=␣${g␣%␣2}
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣""".trimIndent()
            )
        }
    }
}
```

Лістинг 4: ToolBar.kt

```
package com.github.erotourtes.view

class ToolBar {
}
```

3.3 Module: com.github.erotourtes.utils

Лістинг 5: 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)

    override fun toString(): String = "Dimension(x1=$x1, y1=$y1, x2=$x2, y2=$y2)"
}
```

Лістинг 6: 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) {
    save()
    lambda(this)
    restore()
}
```

Лістинг 7: 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 getToCornerDimension(dm: Dimension): Dimension {
    val w = dm.width
    val h = dm.height
    val (cx, cy) = dm.getRaw().first

    val sX = cx - w
    val sY = cy - h

    return Dimension().setStart(sX, sY).setEnd(sX + w * 2, sY + h * 2)
}
```

3.4 Module: com.github.erotourtes.drawing

Лістинг 8: CanvasHandler.kt

```
package com.github.erotourtes.drawing

import com.github.erotourtes.view.n
```

```kotlin
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(editorClas
        editorClass.getConstructor(ShapesList::class.java, GraphicsContext::class.java)
            .newInstance(shapes, canvas.graphicsContext2D)
    }
}
```

Лістинг 9: CanvasPane.kt

```kotlin
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())
    }
}
```

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

Лістинг 10: Shape.kt

```kotlin
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) {
    protected val dm = Dimension()

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

    abstract fun draw()
```

```
    open fun setDm(curDm: Dimension) = curDm.copyTo(dm)

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

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

Лістинг 11: 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.TRANSPARENT
        colorStroke = Color.BLACK
    }

    override fun setDm(curDm: Dimension) = dm.copyFrom(getToCornerDimension(curDm))
```

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

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

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

    override fun setDm(curDm: Dimension) = dm.copyFrom(getToCornerDimension(curDm))
}
```

3.6 Module: com.github.erotourtes.drawing.editor

Лістинг 12: 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(protected 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) {
        redraw()
        dm.setStart(e.x, e.y)
    }

    protected open fun onMouseDragged(e: MouseEvent) {
```

```
            redraw()

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

        protected open fun onMouseReleased(e: MouseEvent) {
            if (e.isDragDetect) return // returns if mouse was not dragged
            shape.setDm(dm)
            shapes.add(shape.copy())
            redraw()
        }

        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 fun redraw() {
            clear()
            drawAll()
        }

        abstract fun previewLine()

        protected fun setPreviewProperties() {
            gc.setLineDashes(5.0)
            gc.stroke = Color.BLACK
        }
    }
```

Лістинг 13: 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

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.setDm(dm)
    }

    override fun onMouseReleased(e: MouseEvent) {
        shapes.add(shape.copy())
        redraw()
```

```
    }

    override fun previewLine() {}
}

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

    override fun previewLine() = gc.drawOnce {
        setPreviewProperties()
        strokeLine(dm)
    }
}

class RectEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Rect(gc)
    override fun previewLine()  = gc.drawOnce {
        setPreviewProperties()
        strokeRect(getToCornerDimension(dm))
    }
}

class EllipseEditor(shapes: ShapesList, gc: GraphicsContext) : Editor(shapes, gc) {
    override val shape = Ellipse(gc)
    override fun previewLine() = gc.drawOnce {
        setPreviewProperties()
        strokeOval(getToCornerDimension(dm))
        strokeRect(getToCornerDimension(dm))
    }
}
```

Лістинг 14: 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 overflo
        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 Illeg
```

```
    }
    override fun toString(): String = "ShapesList(index=$shapeIndex)"
}
```

4 Ілюстрації:

5 Висновки:

Отже, я отримав вміння та навички використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів Kotlin, запрограмувавши графічний інтерфейс користувача.