Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**Лабораторна робота №1**

*з предмету: «ТПКС»*

*по темі:*

«Редактор блок-схем алгоритмів»

Виконав: студент ФІОТ

групи ІО-92

Петрук В.О.

Київ 2012р.

**Мета:** здобуття навичок з побудови редактора блок-схем алгоритмів. Розробка

інтерфейсу користувача та функціонального наповнення. Розробка засобів

перетворення форматів зберігання даних.

**Завдання**

1. Представити номер залікової книжки в двійковому вигляді:

(NNNN)10=(n15n14…n2n1)2 = 100100000000112

2. В залежності від молодшого розряду номера залікової книжки визначити тип редактора:

n1 = 1: Редактор логічних схем алгоритмів (ЛСА)

3. Розробити формат файлу для зберігання схеми алгоритму у матричному вигляді

(матриця зв’язків між вузлами та матриця співвідношення сигналів та вузлів):

n2=1: Бінарний

4. Реалізувати редактор алгоритму заданого типу (п.2) з можливістю

збереження/відновлення результатів роботи програми у матричному вигляді

згідно розробленого формату (п.3). Передбачити в редакторі наступні функції:

• створення нової блок схеми алгоритму;

• модифікація алгоритму (створення/видалення початкового, кінцевого,

логічних(X) та операційних(Y) вузлів, редагування сигналів вузлів,

створення/видалення зв’язків між вузлами);

• контроль вводу (тільки один початковий та один кінцевий вузли, логічні та

операційні вузли мають містити не менш одного сигналу, логічні вузли

містять тільки вхідні сигнали, а операційні – тільки вихідні, тощо);

• збереження алгоритму у матричному вигляді згідно (п.3);

• відновлення алгоритму з матричного вигляду згідно (п.3).

**Опис програми**

Вигляд головного вікна програми зображений на рис. 1.

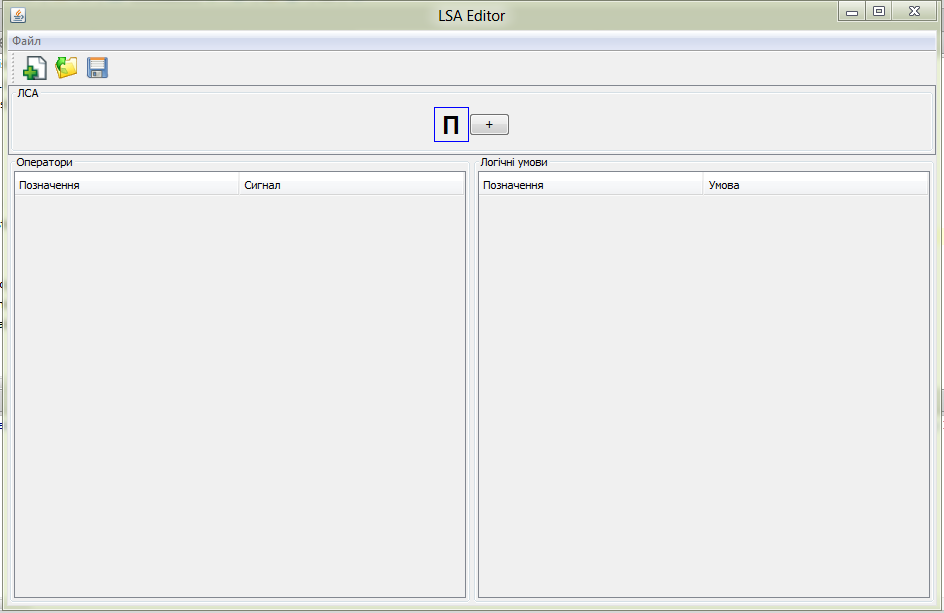


Рисунок 1 – Головне вікно програми

Головне вікно складається з:

1. головного меню програми
2. панелі інструментів
3. панелі рисування з кнопкою + (створення нового елементу ЛСА)
4. таблиць, де відображаються позначення та сигнали операторних та логічних вершин

Для додавання елементів ЛСА необхідно натиснути кнопку + панелі рисування. Буде відображено вікно створення нового елементу ЛСА (рис. 2).

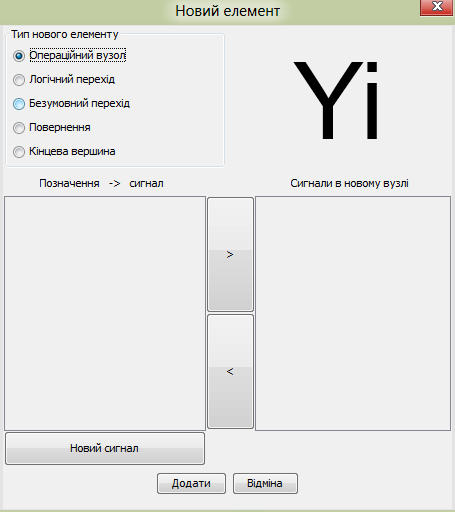


Рисунок 2 – Новий елемент ЛСА

Необхідно обрати необхідний тип нового елементу ЛСА та обрати які сигнали будуть в даній вершині. Якщо необхідні сигнали відсутні в списку, необхідно натиснути кнопку Новий сигнал(для операторних вершин) або Нова умова (для умовних вершин). Після цього на екрані буде відображено діалог створення нового сигналу чи умови (рис. 3).

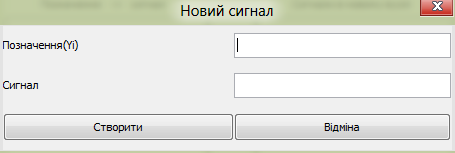
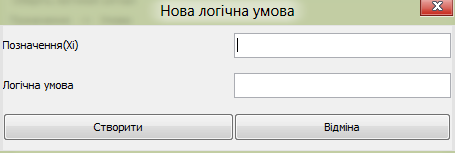
 

Рисунок 3 – Діалогові вікна створення нового сигналу та умови

Для створення нового алгоритму ЛСА необхідно натиснути на кнопку E:\KPI-edu\4 kurs\ТПКС\labs\src\lab1\icons\new.png на панелі інструментів або на відповідний пункт меню Файл.

Для відкриття раніше створеного проекту, необхідно натиснути на кнопку E:\KPI-edu\4 kurs\ТПКС\labs\src\lab1\icons\open.gif панелі інструментів або головного меню (рис. 4) та обрати необхідний файл.

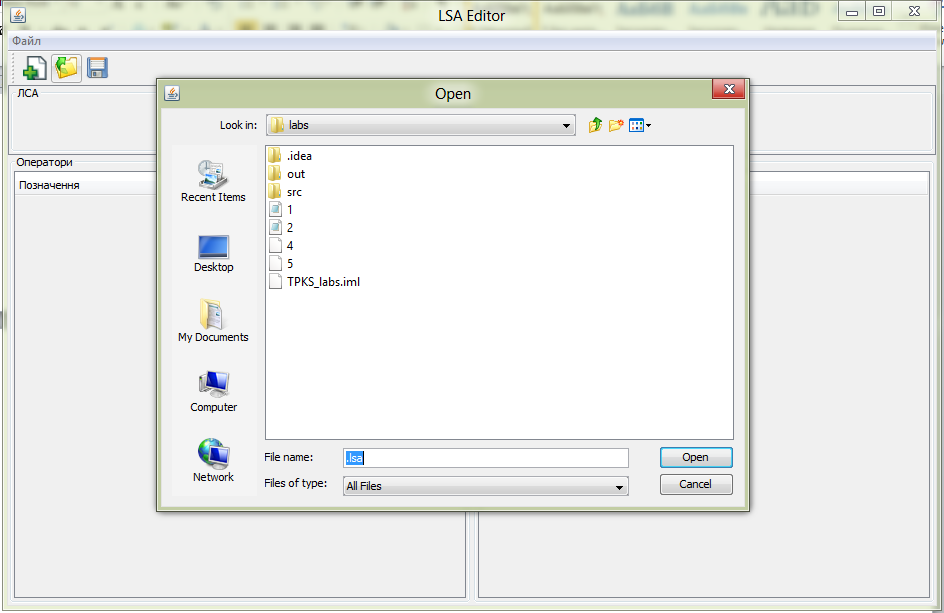


Рисунок 4 – Відкриття проекту

Для збереження проекту у файл необхідно натиснути на кнопку E:\KPI-edu\4 kurs\ТПКС\labs\src\lab1\icons\save.png панелі інструментів, або відповідній кнопці головного меню та ввести ім’я для файлу (рис 5).

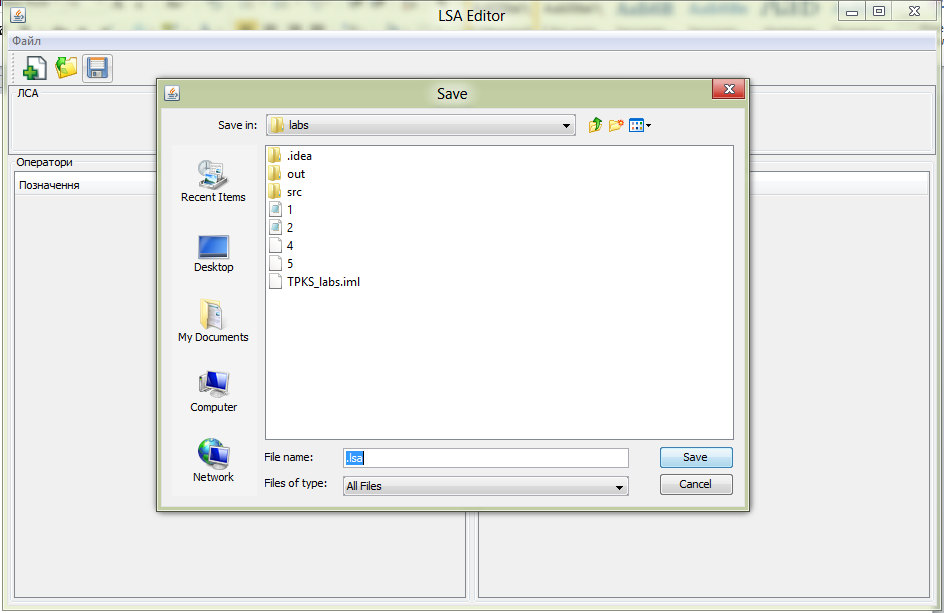


Рисунок 5 – Збереження проекту

Для редагування вершин алгоритму необхідно дві клацнути ЛКМ на елементі в панелі рисування, або натиснути ПКМ та обрати варіант Редагувати елемент (рис. 6)

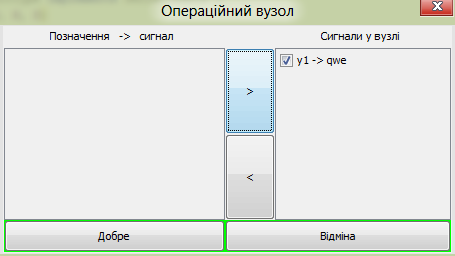


Рисунок 6 – Редагування елементу алгоритму

Для зміни порядку слідування вершин алгоритму необхідно вирізати елемент (ПКМ на елементі + вирізати) та вставити в необхідну позицію .

Для видалення елементу ЛСА необхідно клінути на елементі ПКМ та обрати пункт Видалити.

Для редагування сигналів алгоритму необхідно виділити відповідний чарунок у таблиці відповідності позначень та сигналів та ввести нове значення сигналу або позначення(рис 7).

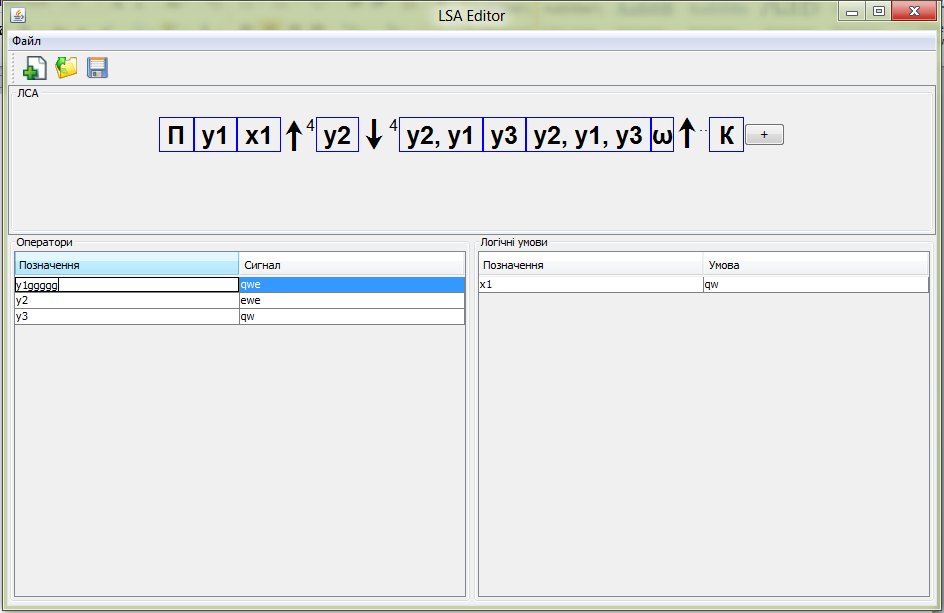


Рисунок 8 – Редагування сигналів алгоритму

Для виходу з програми можна вибрати меню Файл та натиснути Вихід або натиснути на  головного вікна програми.

**Код програми:**

package lab1;

import lab1.gui.MainFrame;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

public class Main{

private static MainFrame mFrame;

/\*\*

\* Точка запуску програми. Тут створюється головне вікно проограми та нова ЛСА.

\* @param args

\*/

public static void main(String args[]) {

try {

UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.WindowsLookAndFeel");

} catch (Exception e) {

e.printStackTrace();

}

mFrame = new MainFrame();

LSA lsa = new LSA();

mFrame.setLsa(lsa);

mFrame.createMainFrame();

}

}

package lab1;

import lab1.gui.MainFrame;

import java.util.ArrayList;

public class LSA {

private ArrayList<Unit> lsaItems = new ArrayList<Unit>();

/\*\*

\* Конструктор ЛСА без параметрів(створюється нова ЛСА).

\*/

public LSA() {

}

/\*\*

\* Створення ЛСА з раніше створеним набором елементів.

\*

\* @param lsaItems

\*/

public LSA(ArrayList<Unit> lsaItems) {

this.lsaItems = lsaItems;

}

/\*\*

\* Повертає нововстворений обєкт МСА на основі даної ЛСА.

\*

\* @return

\*/

public MSA getMsa() {

return (new MSA(this));

}

/\*\*

\* Додає новий елемент в дану ЛСА.

\*

\* @param unit

\*/

public void addItem(Unit unit) {

lsaItems.add(unit);

}

//------------------Operation -------------------------

/\*\*

\* Повертає ArrayList оперторних вершин даного ЛСА.

\*

\* @return

\*/

private ArrayList<Unit> getOperationUnits() {

ArrayList<Unit> operationUnits = new ArrayList<Unit>();

for (Unit unit : lsaItems) {

if (unit.getUnitType().equals(UnitType.O)) {

operationUnits.add(unit);

}

}

return operationUnits;

}

/\*\*

\* Повертає масив опертаторних вершин даного ЛСА.

\*

\* @return

\*/

public String[][] getOperationUnitsArray() {

ArrayList<Unit> operationUnits = getOperationUnits();

int size = operationUnits.size();

ArrayList<String> operMarks = new ArrayList<String>();

ArrayList<String> operSignals = new ArrayList<String>();

for (int i = 0; i < size; i++) {

int count = 0;

for (String s : operationUnits.get(i).getUnitMarking()) {

if (checkOperationSignal(operMarks, s)) {

operMarks.add(s);

operSignals.add(operationUnits.get(i).getUnitValue()[count]);

}

count++;

}

}

String[][] operUnits = new String[2][operMarks.size()];

for (int i = 0; i < operMarks.size(); i++) {

operUnits[0][i] = operMarks.get(i);

operUnits[1][i] = operSignals.get(i);

}

return operUnits;

}

/\*\*

\* Змінює поля операторної вершини ЛСА з позначенням marking.

\*

\* @param marking

\* @param newValue

\* @param changeMarking

\*/

public void changeOperationUnit(String marking, String newValue, boolean changeMarking) {

ArrayList<Unit> operationUnits = getOperationUnits();

for (Unit unit : operationUnits) {

for (int i = 0; i < unit.getUnitMarking().length; i++) {

if (unit.getUnitMarking()[i].equals(marking)) {

if (changeMarking) {

unit.getUnitMarking()[i] = newValue;

} else {

unit.getUnitValue()[i] = newValue;

}

}

}

}

}

//----------------------------Conditional ----------------------------------

/\*\*

\* Повертає ArrayList умовних вершин ЛСА.

\*

\* @return

\*/

private ArrayList<Unit> getConditionUnits() {

ArrayList<Unit> conditionUnits = new ArrayList<Unit>();

for (Unit unit : lsaItems) {

if (unit.getUnitType().equals(UnitType.C)) {

conditionUnits.add(unit);

}

}

return conditionUnits;

}

/\*\*

\* Повертає масив умовних вершин ЛСА,

\*

\* @return

\*/

public String[][] getConditionUnitsArray() {

ArrayList<Unit> conditionUnits = getConditionUnits();

int size = conditionUnits.size();

ArrayList<String> condMarks = new ArrayList<String>();

ArrayList<String> condSignals = new ArrayList<String>();

for (int i = 0; i < size; i++) {

int count = 0;

for (String s : conditionUnits.get(i).getUnitMarking()) {

if (checkOperationSignal(condMarks, s)) {

condMarks.add(s);

condSignals.add(conditionUnits.get(i).getUnitValue()[count]);

}

count++;

}

}

String[][] condUnits = new String[2][condMarks.size()];

for (int i = 0; i < condMarks.size(); i++) {

condUnits[0][i] = condMarks.get(i);

condUnits[1][i] = condSignals.get(i);

}

return condUnits;

}

/\*\*

\* Змінює поля умовної вершини ЛСА з позначенням marking.

\*

\* @param marking

\* @param newValue

\* @param changeMarking

\*/

public void changeConditionUnit(String marking, String newValue, boolean changeMarking) {

ArrayList<Unit> conditionUnits = getConditionUnits();

for (Unit unit : conditionUnits) {

if (unit.getUnitMarking()[0].equals(marking)) {

if (changeMarking) {

unit.getUnitMarking()[0] = newValue;

} else {

unit.getUnitValue()[0] = newValue;

}

}

}

}

/\*\*

\* Перевірка, чи є елемент з позначенням mark в масиві arrayOfMarks.

\*

\* @param arrayOfMarks

\* @param mark

\* @return

\*/

private boolean checkOperationSignal(ArrayList<String> arrayOfMarks, String mark) {

boolean flag = true;

for (String s : arrayOfMarks) {

if (mark.equals(s)) {

flag = false;

break;

}

}

return flag;

}

/\*\*

\* Повертає масив елементів ЛСА.

\*

\* @return

\*/

public ArrayList<Unit> getLsaItems() {

return lsaItems;

}

/\*\*

\* Виводить значення елементів ЛСА в стандартний потік виводу.

\*/

public void printLSA() {

for (Unit unit : lsaItems) {

System.out.println(unit.getUnitType() + " :");

for (int i = 0; i < unit.getUnitMarking().length; i++) {

System.out.print(" " + unit.getUnitMarking()[i] + " -> " + unit.getUnitValue()[i]);

}

System.out.println(" idx = " + unit.getIndex());

}

}

}

package lab1;

import java.io.Serializable;

public class Unit implements Serializable {

private UnitType uType;

private String[] marking;

private String[] value;

private int index;

/\*\*

\* Конструктор елементу ЛСА.

\* @param uType - тип елементу

\* @param marking - масив позначеннь елементу

\* @param value - масив сигналів елементу

\* @param idx - індекс переходу або повернення

\*/

public Unit(UnitType uType, String[] marking, String[] value, int idx) {

this.uType = uType;

this.marking = marking;

this.value = value;

this.index = idx;

}

/\*\*

\* @return тип елементу ЛСА

\*/

public UnitType getUnitType() {

return this.uType;

}

/\*\*

\* @return масив позначень елементу

\*/

public String[] getUnitMarking() {

return this.marking;

}

/\*\*

\* @param unitMarking - масив нових позначень даного елементу

\*/

public void setUnitMarking(String[] unitMarking) {

this.marking = unitMarking;

}

/\*\*

\* @return масив сигналів елементу

\*/

public String[] getUnitValue() {

return this.value;

}

/\*\*

\* @param unitValue - масив нових значень даного елементу

\*/

public void setUnitValue(String[] unitValue) {

this.value = unitValue;

}

/\*\*

\* @return індекс переходу або повернення

\*/

public int getIndex() {

return this.index;

}

/\*\*

\* @param index - новий індекс переходу або повернення

\*/

public void setIndex(int index) {

this.index =index;

}

}

package lab1;

import java.io.Serializable;

/\*\*

\* Типи елементів ЛСА.

\*/

public enum UnitType implements Serializable {

B, O, C, U, R, E;

}

package lab1;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import static java.lang.System.\*;

/\*\*

\* Перелік тестів, для контролю правильності введеного алгоритму.

\*/

enum TestType implements Serializable {

IS\_END, IS\_RETURN\_AFTER\_END, IS\_AFTER\_END\_UNITS\_REACHABLE, DUPLICATE\_RETURN\_INDEX, CONDITION\_INDEX\_CORRESPONDS\_RETURN\_INDEX

}

public class MSA implements Serializable {

private int[][] unitsUnion;

private Map<Integer, Unit> unitSignals = new HashMap<Integer, Unit>();

private int endIdx;

/\*\*

\* Конструктор МСА.

\* @param lsa

\*/

public MSA(LSA lsa) {

createMSA(lsa);

}

/\*\*

\* Повертає матрицю звязків вершин ЛСА.

\* @return

\*/

public int[][] getUnitsUnion(){

return unitsUnion;

}

/\*\*

\* Повертає матрицю свівідношення сигналів сугналів та вузлів.

\* @return

\*/

public Map<Integer, Unit> getUnitSignals(){

return unitSignals;

}

/\*\*

\* Створює МСА на основі ЛСА.

\* @param lsa

\*/

public void createMSA(LSA lsa) {

unitsUnion = new int[getMatrixSize(lsa)][getMatrixSize(lsa)];

for (int i = 0; i < unitsUnion.length; i++) {

// таблиця відповідності: номер сигналу <-> сигнал

unitSignals.put(i, getThisUnit(i, lsa));

// таблиця переходів

UnitType currentUnitType = getThisUnit(i, lsa).getUnitType();

switch (currentUnitType) {

case O:

setOperationGoTo(i, lsa);

break;

case C:

setConditionGoTo(i, lsa);

break;

}

}

}

/\*\*

\* Встановлює перехід операційної вершини на наступну вершину (в МСА).

\* @param i

\*/

private void setOperationGoTo(int i, LSA lsa) {

if (getNextUnit(getThisUnit(i, lsa), lsa) != (null)) {

UnitType nextUnitType = getNextUnit(getThisUnit(i, lsa), lsa).getUnitType();

switch (nextUnitType) {

case O:

case C:

setThisOne(i);

break;

case U:

int rightPos = findNextUnitByIndex(getNextUnit(getThisUnit(i, lsa), lsa).getIndex(), lsa);

if (rightPos != -1) {

unitsUnion[i][rightPos] = 1;

}

break;

case R:

int nextAbsPos = skipR(getAbsUnitPos(getThisUnit(i, lsa), lsa), lsa);

if (nextAbsPos != -1) {

unitsUnion[i][nextAbsPos] = 1;

}

break;

case E:

}

}

}

/\*\*

\* Встановлює перехід умовної вершини на наступну вершину (в МСА).

\* @param i

\*/

private void setConditionGoTo(int i, LSA lsa) {

setOperationGoTo(i, lsa);

int secondPos = findNextUnitByIndex(getThisUnit(i, lsa).getIndex(), lsa);

System.out.println("ni = " + secondPos);

if (secondPos != -1) {

unitsUnion[i][secondPos] = -1;

}

}

/\*\*

\* Визначає розмір матриці МСА.

\* @param lsa

\* @return

\*/

private int getMatrixSize(LSA lsa) {

int count = 0;

for (Unit unit : lsa.getLsaItems()) {

if ((unit.getUnitType().equals(UnitType.O)) || (unit.getUnitType().equals(UnitType.C))) {

count++;

}

}

return count;

}

/\*\*

\* Повертає елемент з ЛСА за заданою відносною позицією.

\* @param pos

\* @param lsa

\* @return

\*/

private Unit getThisUnit(int pos, LSA lsa) {

int count = -1;

for (Unit unit : lsa.getLsaItems()) {

if ((unit.getUnitType().equals(UnitType.O)) || (unit.getUnitType().equals(UnitType.C))) {

count++;

}

if (count == pos) {

return unit;

}

}

return null;

}

/\*\*

\* Повертає елемент, який розташовується в ЛСА справа від заданого.

\* @param unit

\* @param lsa

\* @return

\*/

private Unit getNextUnit(Unit unit, LSA lsa) {

ArrayList<Unit> list = lsa.getLsaItems();

for (int i = 0; i < list.size() - 1; i++) {

if (list.get(i).equals(unit)) {

return list.get(i + 1);

}

}

return null;

}

/\*\*

\* Шукає відносну позицію наступного елемента за даним, враховуючи елементи поверення та безумовного переходу.

\* @param index

\* @return відносну позицію шуканого елементу, або -1 у разі відсутності такого

\*/

private int findNextUnitByIndex(int index, LSA lsa) {

for (int i = 0; i < lsa.getLsaItems().size(); i++) {

if ((lsa.getLsaItems().get(i).getUnitType().equals(UnitType.R)) && (lsa.getLsaItems().get(i).getIndex() == index)) {

return skipR(i, lsa);

}

}

return -1;

}

/\*\*

\* Встановлює перехід на наступний елемент матриці.

\* @param pos

\*/

private void setThisOne(int pos) {

for (int i = 0; i < unitsUnion.length; i++) {

unitsUnion[pos][i] = 0;

}

unitsUnion[pos][pos + 1] = 1;

}

/\*\*

\* Повертає відносну позиція елемента.

\* @param needUnit

\* @return

\*/

private int getIndexOfUnit(Unit needUnit, LSA lsa) {

int count = 0;

for (Unit unit : lsa.getLsaItems()) {

if ((unit.getUnitType().equals(UnitType.O)) || (unit.getUnitType().equals(UnitType.C))) {

if (unit.equals(needUnit)) {

break;

}

count++;

}

}

return count;

}

/\*\*

\* Повертає абсолютну позицію елемента в масиві.

\* @param unit

\* @return

\*/

private int getAbsUnitPos(Unit unit, LSA lsa) {

for (int i = 0; i < lsa.getLsaItems().size(); i++) {

if (lsa.getLsaItems().get(i).equals(unit)) {

return i;

}

}

return -1;

}

/\*\*

\* Повертає відносну позицію елемента після даного елемента, враховуючи

\* всі елементи поверення та безумовних переходів.

\* @param thisAbsPos

\* @return

\*/

private int skipR(int thisAbsPos, LSA lsa) {

int needPos = -1;

for (int i = thisAbsPos + 1; i < lsa.getLsaItems().size(); i++) {

if (lsa.getLsaItems().get(i).getUnitType().equals(UnitType.U)) {

needPos = findNextUnitByIndex(lsa.getLsaItems().get(i).getIndex(), lsa);

break;

}

if ((lsa.getLsaItems().get(i).getUnitType().equals(UnitType.O)) || lsa.getLsaItems().get(i).getUnitType().equals(UnitType.C)) {

needPos = getIndexOfUnit(lsa.getLsaItems().get(i), lsa);

break;

}

if (lsa.getLsaItems().get(i).getUnitType().equals(UnitType.R)) {

needPos = skipR(i, lsa);

break;

}

}

return needPos;

}

/\*\*

\* Виведення в консоль матриці МСА

\*/

public void printMSA() {

System.out.println("----------MSA------------");

for (int i = 0; i < unitsUnion.length; i++) {

for (int j = 0; j < unitsUnion.length; j++) {

out.print(unitsUnion[i][j] + " ");

}

out.println();

}

}

/\*\*

\* Виведення в консоль значення вершин МСА

\*/

public void printSignals() {

for (int i = 0; i < unitSignals.size(); i++) {

System.out.println(unitSignals.get(i).getUnitType() + " :");

for (int j = 0; j < unitSignals.get(j).getUnitMarking().length; j++) {

System.out.print(" " + unitSignals.get(j).getUnitMarking()[j] + " -> " + unitSignals.get(j).getUnitValue()[j]);

}

System.out.println(" idx = " + unitSignals.get(i).getIndex());

}

}

}

package lab1;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.Serializable;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

import static java.lang.System.\*;

/\*\*

\* Перелік тестів, для контролю правильності введеного алгоритму.

\*/

enum TestType implements Serializable {

IS\_END, IS\_RETURN\_AFTER\_END, IS\_AFTER\_END\_UNITS\_REACHABLE, DUPLICATE\_RETURN\_INDEX, CONDITION\_INDEX\_CORRESPONDS\_RETURN\_INDEX

}

public class MSA implements Serializable {

private int[][] unitsUnion;

private Map<Integer, Unit> unitSignals = new HashMap<Integer, Unit>();

private int endIdx;

/\*\*

\* Конструктор МСА.

\* @param lsa

\*/

public MSA(LSA lsa) {

createMSA(lsa);

}

/\*\*

\* Повертає матрицю звязків вершин ЛСА.

\* @return

\*/

public int[][] getUnitsUnion(){

return unitsUnion;

}

/\*\*

\* Повертає матрицю свівідношення сигналів сугналів та вузлів.

\* @return

\*/

public Map<Integer, Unit> getUnitSignals(){

return unitSignals;

}

/\*\*

\* Створює МСА на основі ЛСА.

\* @param lsa

\*/

public void createMSA(LSA lsa) {

unitsUnion = new int[getMatrixSize(lsa)][getMatrixSize(lsa)];

for (int i = 0; i < unitsUnion.length; i++) {

// таблиця відповідності: номер сигналу <-> сигнал

unitSignals.put(i, getThisUnit(i, lsa));

// таблиця переходів

UnitType currentUnitType = getThisUnit(i, lsa).getUnitType();

switch (currentUnitType) {

case O:

setOperationGoTo(i, lsa);

break;

case C:

setConditionGoTo(i, lsa);

break;

}

}

}

/\*\*

\* Встановлює перехід операційної вершини на наступну вершину (в МСА).

\* @param i

\*/

private void setOperationGoTo(int i, LSA lsa) {

if (getNextUnit(getThisUnit(i, lsa), lsa) != (null)) {

UnitType nextUnitType = getNextUnit(getThisUnit(i, lsa), lsa).getUnitType();

switch (nextUnitType) {

case O:

case C:

setThisOne(i);

break;

case U:

int rightPos = findNextUnitByIndex(getNextUnit(getThisUnit(i, lsa), lsa).getIndex(), lsa);

if (rightPos != -1) {

unitsUnion[i][rightPos] = 1;

}

break;

case R:

int nextAbsPos = skipR(getAbsUnitPos(getThisUnit(i, lsa), lsa), lsa);

if (nextAbsPos != -1) {

unitsUnion[i][nextAbsPos] = 1;

}

break;

case E:

}

}

}

/\*\*

\* Встановлює перехід умовної вершини на наступну вершину (в МСА).

\* @param i

\*/

private void setConditionGoTo(int i, LSA lsa) {

setOperationGoTo(i, lsa);

int secondPos = findNextUnitByIndex(getThisUnit(i, lsa).getIndex(), lsa);

System.out.println("ni = " + secondPos);

if (secondPos != -1) {

unitsUnion[i][secondPos] = -1;

}

}

/\*\*

\* Визначає розмір матриці МСА.

\* @param lsa

\* @return

\*/

private int getMatrixSize(LSA lsa) {

int count = 0;

for (Unit unit : lsa.getLsaItems()) {

if ((unit.getUnitType().equals(UnitType.O)) || (unit.getUnitType().equals(UnitType.C))) {

count++;

}

}

return count;

}

/\*\*

\* Повертає елемент з ЛСА за заданою відносною позицією.

\* @param pos

\* @param lsa

\* @return

\*/

private Unit getThisUnit(int pos, LSA lsa) {

int count = -1;

for (Unit unit : lsa.getLsaItems()) {

if ((unit.getUnitType().equals(UnitType.O)) || (unit.getUnitType().equals(UnitType.C))) {

count++;

}

if (count == pos) {

return unit;

}

}

return null;

}

/\*\*

\* Повертає елемент, який розташовується в ЛСА справа від заданого.

\* @param unit

\* @param lsa

\* @return

\*/

private Unit getNextUnit(Unit unit, LSA lsa) {

ArrayList<Unit> list = lsa.getLsaItems();

for (int i = 0; i < list.size() - 1; i++) {

if (list.get(i).equals(unit)) {

return list.get(i + 1);

}

}

return null;

}

/\*\*

\* Шукає відносну позицію наступного елемента за даним, враховуючи елементи поверення та безумовного переходу.

\* @param index

\* @return відносну позицію шуканого елементу, або -1 у разі відсутності такого

\*/

private int findNextUnitByIndex(int index, LSA lsa) {

for (int i = 0; i < lsa.getLsaItems().size(); i++) {

if ((lsa.getLsaItems().get(i).getUnitType().equals(UnitType.R)) && (lsa.getLsaItems().get(i).getIndex() == index)) {

return skipR(i, lsa);

}

}

return -1;

}

/\*\*

\* Встановлює перехід на наступний елемент матриці.

\* @param pos

\*/

private void setThisOne(int pos) {

for (int i = 0; i < unitsUnion.length; i++) {

unitsUnion[pos][i] = 0;

}

unitsUnion[pos][pos + 1] = 1;

}

/\*\*

\* Повертає відносну позиція елемента.

\* @param needUnit

\* @return

\*/

private int getIndexOfUnit(Unit needUnit, LSA lsa) {

int count = 0;

for (Unit unit : lsa.getLsaItems()) {

if ((unit.getUnitType().equals(UnitType.O)) || (unit.getUnitType().equals(UnitType.C))) {

if (unit.equals(needUnit)) {

break;

}

count++;

}

}

return count;

}

/\*\*

\* Повертає абсолютну позицію елемента в масиві.

\* @param unit

\* @return

\*/

private int getAbsUnitPos(Unit unit, LSA lsa) {

for (int i = 0; i < lsa.getLsaItems().size(); i++) {

if (lsa.getLsaItems().get(i).equals(unit)) {

return i;

}

}

return -1;

}

/\*\*

\* Повертає відносну позицію елемента після даного елемента, враховуючи

\* всі елементи поверення та безумовних переходів.

\* @param thisAbsPos

\* @return

\*/

private int skipR(int thisAbsPos, LSA lsa) {

int needPos = -1;

for (int i = thisAbsPos + 1; i < lsa.getLsaItems().size(); i++) {

if (lsa.getLsaItems().get(i).getUnitType().equals(UnitType.U)) {

needPos = findNextUnitByIndex(lsa.getLsaItems().get(i).getIndex(), lsa);

break;

}

if ((lsa.getLsaItems().get(i).getUnitType().equals(UnitType.O)) || lsa.getLsaItems().get(i).getUnitType().equals(UnitType.C)) {

needPos = getIndexOfUnit(lsa.getLsaItems().get(i), lsa);

break;

}

if (lsa.getLsaItems().get(i).getUnitType().equals(UnitType.R)) {

needPos = skipR(i, lsa);

break;

}

}

return needPos;

}

/\*\*

\* Виведення в консоль матриці МСА

\*/

public void printMSA() {

System.out.println("----------MSA------------");

for (int i = 0; i < unitsUnion.length; i++) {

for (int j = 0; j < unitsUnion.length; j++) {

out.print(unitsUnion[i][j] + " ");

}

out.println();

}

}

/\*\*

\* Виведення в консоль значення вершин МСА

\*/

public void printSignals() {

for (int i = 0; i < unitSignals.size(); i++) {

System.out.println(unitSignals.get(i).getUnitType() + " :");

for (int j = 0; j < unitSignals.get(j).getUnitMarking().length; j++) {

System.out.print(" " + unitSignals.get(j).getUnitMarking()[j] + " -> " + unitSignals.get(j).getUnitValue()[j]);

}

System.out.println(" idx = " + unitSignals.get(i).getIndex());

}

}

}

package lab1.gui;

import lab1.LSA;

import lab1.Unit;

import lab1.UnitType;

import javax.swing.\*;

import javax.swing.border.TitledBorder;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.font.TextAttribute;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

public class DrawPanel extends JPanel {

private LSA lsa;

private Map<JLabel, Unit> labelUnitHashMap = new HashMap<JLabel, Unit>();

private JButton addButton;

private MainPanel parentPanel;

private DrawPanel thisPanel;

private LSAMouseListener lsaMouseListener;

private JLabel selectedLabel;

private JLabel bufferedLabel = null;

/\*\*

\* Конструктор панелі рисування елементів ЛСА.

\*

\* @param lsa

\* @param mainPanel

\*/

public DrawPanel(LSA lsa, MainPanel mainPanel) {

this.thisPanel = this;

lsaMouseListener = new LSAMouseListener(this);

setBorder(new TitledBorder("ЛСА"));

this.lsa = lsa;

this.parentPanel = mainPanel;

setLayout(new FlowLayout());

addButton();

String[] s = {"П"};

Unit beginUnit = new Unit(UnitType.B, s, s, 0);

lsa.addItem(beginUnit);

drawLsa(UnitType.B);

}

/\*\*

\* Додає на панель рисування кнопку створення нового елементу ЛСА.

\*/

public void addButton() {

addButton = new JButton("+");

FlowLayout flowLayout = new FlowLayout();

flowLayout.setHgap(0);

setLayout(flowLayout);

add(addButton);

addButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

deselect();

NewItemDialog newItemDialog = new NewItemDialog(thisPanel, lsa);

parentPanel.updateTables();

}

});

}

/\*\*

\* Рисує новий елемент ЛСА на панелі рисування.

\*

\* @param unitType - тип нового елементу.

\*/

public void drawLsa(UnitType unitType) {

ArrayList<Unit> lsaItems;

int size;

String[] marks;

String labelStr;

JLabel newLabel = null;

JLabel arrowUpLabel = new JLabel("↑");

JLabel indexLabel;

HashMap textAttributes;

switch (unitType) {

case B:

newLabel = new JLabel("П");

newLabel.setFont(new Font("TimesRoman", Font.BOLD, 25));

newLabel.setBorder(BorderFactory.createLineBorder(Color.blue));

newLabel.setPreferredSize(new Dimension(35, 35));

newLabel.setVerticalAlignment(JLabel.CENTER);

newLabel.setHorizontalAlignment(JLabel.CENTER);

this.selectedLabel = newLabel;

this.selectedLabel.setOpaque(true);

remove(addButton);

add(newLabel);

add(addButton);

repaint();

revalidate();

//додаємо елемент в hashmap

lsaItems = lsa.getLsaItems();

size = lsaItems.size();

labelUnitHashMap.put(newLabel, lsaItems.get(size - 1));

break;

case O:

lsaItems = lsa.getLsaItems();

size = lsaItems.size();

marks = lsaItems.get(size - 1).getUnitMarking();

labelStr = " ";

for (String s : marks) {

labelStr += (s + ", ");

}

newLabel = new JLabel(labelStr.substring(0, labelStr.length() - 2) + " ");

newLabel.setFont(new Font("TimesRoman", Font.BOLD, 25));

newLabel.setBorder(BorderFactory.createLineBorder(Color.blue));

newLabel.setOpaque(true);

newLabel.addMouseListener(lsaMouseListener);

remove(addButton);

add(newLabel);

add(addButton);

labelUnitHashMap.put(newLabel, lsaItems.get(size - 1));

break;

case C:

lsaItems = lsa.getLsaItems();

size = lsaItems.size();

marks = lsaItems.get(size - 1).getUnitMarking();

labelStr = " ";

for (String s : marks) {

labelStr += (s + ", ");

}

newLabel = new JLabel(labelStr.substring(0, labelStr.length() - 2) + " ");

newLabel.setFont(new Font("TimesRoman", Font.BOLD, 25));

newLabel.setBorder(BorderFactory.createLineBorder(Color.blue));

newLabel.setOpaque(true);

newLabel.addMouseListener(lsaMouseListener);

remove(addButton);

add(newLabel);

arrowUpLabel.setFont(new Font("TimesRoman", Font.BOLD, 40));

arrowUpLabel.setPreferredSize(new Dimension(25, 55));

arrowUpLabel.setVerticalAlignment(JLabel.BOTTOM);

arrowUpLabel.setHorizontalAlignment(JLabel.RIGHT);

indexLabel = new JLabel((lsaItems.get(size - 1).getIndex() + ""));

textAttributes = new HashMap();

textAttributes.put(TextAttribute.FONT, "TimesRoman");

textAttributes.put(TextAttribute.SIZE, new Float(25));

textAttributes.put(TextAttribute.SUPERSCRIPT, TextAttribute.SUPERSCRIPT\_SUPER);

indexLabel.setFont(new Font(textAttributes));

indexLabel.setVerticalTextPosition(JLabel.TOP);

indexLabel.setHorizontalAlignment(JLabel.LEFT);

indexLabel.setPreferredSize(new Dimension(((int) Math.log10((double) lsaItems.get(size - 1).getIndex()) + 1) \* 10, 50));

add(arrowUpLabel);

add(indexLabel);

add(addButton);

labelUnitHashMap.put(newLabel, lsaItems.get(size - 1));

break;

case U:

lsaItems = lsa.getLsaItems();

size = lsaItems.size();

labelStr = "ω";

newLabel = new JLabel(labelStr);

newLabel.setFont(new Font("TimesRoman", Font.BOLD, 25));

newLabel.setBorder(BorderFactory.createLineBorder(Color.blue));

newLabel.setOpaque(true);

newLabel.addMouseListener(lsaMouseListener);

remove(addButton);

add(newLabel);

arrowUpLabel.setFont(new Font("TimesRoman", Font.BOLD, 40));

arrowUpLabel.setPreferredSize(new Dimension(25, 50));

arrowUpLabel.setVerticalAlignment(JLabel.BOTTOM);

arrowUpLabel.setHorizontalAlignment(JLabel.RIGHT);

arrowUpLabel.setOpaque(true);

indexLabel = new JLabel((lsaItems.get(size - 1).getIndex() + ""));

textAttributes = new HashMap();

textAttributes.put(TextAttribute.FONT, "TimesRoman");

textAttributes.put(TextAttribute.SIZE, new Float(25));

textAttributes.put(TextAttribute.SUPERSCRIPT, TextAttribute.SUPERSCRIPT\_SUPER);

indexLabel.setFont(new Font(textAttributes));

indexLabel.setVerticalTextPosition(JLabel.TOP);

indexLabel.setHorizontalAlignment(JLabel.LEFT);

indexLabel.setPreferredSize(new Dimension(((int) Math.log10((double) lsaItems.get(size - 1).getIndex()) + 1) \* 10, 50));

//indexLabel.setOpaque(true);

add(arrowUpLabel);

add(indexLabel);

add(addButton);

labelUnitHashMap.put(newLabel, lsaItems.get(size - 1));

break;

case R:

lsaItems = lsa.getLsaItems();

size = lsaItems.size();

JLabel arrowDownLabel = new JLabel("↓");

arrowDownLabel.setFont(new Font("TimesRoman", Font.BOLD, 40));

arrowDownLabel.setPreferredSize(new Dimension(30, 50));

arrowDownLabel.setVerticalAlignment(JLabel.BOTTOM);

arrowDownLabel.setHorizontalAlignment(JLabel.RIGHT);

arrowDownLabel.setOpaque(true);

arrowDownLabel.addMouseListener(lsaMouseListener);

indexLabel = new JLabel((lsaItems.get(size - 1).getIndex() + ""));

textAttributes = new HashMap();

textAttributes.put(TextAttribute.FONT, "TimesRoman");

textAttributes.put(TextAttribute.SIZE, new Float(25));

textAttributes.put(TextAttribute.SUPERSCRIPT, TextAttribute.SUPERSCRIPT\_SUPER);

indexLabel.setFont(new Font(textAttributes));

indexLabel.setVerticalTextPosition(JLabel.TOP);

indexLabel.setHorizontalAlignment(JLabel.LEFT);

indexLabel.setPreferredSize(new Dimension(((int) Math.log10((double) lsaItems.get(size - 1).getIndex()) + 1) \* 10, 50));

indexLabel.setOpaque(true);

remove(addButton);

add(arrowDownLabel);

add(indexLabel);

add(addButton);

labelUnitHashMap.put(arrowDownLabel, lsaItems.get(size - 1));

break;

case E:

newLabel = new JLabel("К");

newLabel.setFont(new Font("TimesRoman", Font.BOLD, 25));

newLabel.setBorder(BorderFactory.createLineBorder(Color.blue));

newLabel.setPreferredSize(new Dimension(35, 35));

newLabel.setVerticalAlignment(JLabel.CENTER);

newLabel.setHorizontalAlignment(JLabel.CENTER);

newLabel.setOpaque(true);

newLabel.addMouseListener(lsaMouseListener);

remove(addButton);

add(newLabel);

add(addButton);

lsaItems = lsa.getLsaItems();

size = lsaItems.size();

labelUnitHashMap.put(newLabel, lsaItems.get(size - 1));

break;

}

}

/\*\*

\* Встановлює виділений елемент панелі рисування.

\*

\* @param jLabel

\*/

public void setSelectedLabel(JLabel jLabel) {

selectedLabel = jLabel;

}

/\*\*

\* Повретає елемент ЛСА з хештаблиці відповідності елементів ЛСА та елементів панелі рисування.

\*

\* @param jLabel

\* @return

\*/

public Unit getUnitFromHashmap(JLabel jLabel) {

return this.labelUnitHashMap.get(jLabel);

}

/\*\*

\* Повертає тип елементу ЛСА з хештаблиці відповідності елементів ЛСА та елементів панелі рисування.

\*

\* @param jLabel

\* @return

\*/

public UnitType getUnitTypeFromHashmap(JLabel jLabel) {

return this.labelUnitHashMap.get(jLabel).getUnitType();

}

/\*\*

\* Повертає виділений елемент панелі рисування.

\*

\* @return

\*/

public JLabel getSelectedLabel() {

return selectedLabel;

}

/\*\*

\* Повертає абсолютну позицію елемента панелі рисування з масиву елментів цієї панелі.

\*

\* @param jLabel

\* @return

\*/

public int getPositionOfElement(JLabel jLabel) {

int pos = 0;

for (int i = 0; i < getComponentCount(); i++) {

if (getComponent(i).equals(jLabel)) {

pos = i;

break;

}

}

return pos;

}

/\*\*

\* Зняття виділення обраного елементу панелі рисування.

\*/

public void deselect() {

if ((selectedLabel).getText().equals("↓")) {

getComponent(getPositionOfElement(selectedLabel) + 1).setBackground(getBackground());

}

selectedLabel.setBackground(getBackground());

}

/\*\*

\* Видаляє раніше обраний елемент з панелі рисування.

\*/

public void removeBufferedItem() {

if (bufferedLabel != null) {

if (!getUnitTypeFromHashmap(bufferedLabel).equals(UnitType.R)) {

bufferedLabel.setBorder(BorderFactory.createLineBorder(Color.blue));

} else {

bufferedLabel.setBorder(BorderFactory.createLineBorder(parentPanel.getBackground()));

}

bufferedLabel = null;

}

}

/\*\*

\* Повретає раніще обраний елемент області рисування.

\*

\* @return

\*/

public JLabel getBufferedLabel() {

return bufferedLabel;

}

/\*\*

\* Встановлює буферний елемент панелі рисуання для виконання подільних дій над ним.

\*

\* @param label

\*/

public void setBufferedLabel(JLabel label) {

this.bufferedLabel = label;

}

/\*\*

\* Видаляє елемент з панелі рисування.

\*

\* @param jLabel

\*/

public void removeLabel(JLabel jLabel) {

int posOfLabel = getPositionOfElement(jLabel);

UnitType unitType = getUnitTypeFromHashmap(jLabel);

switch (unitType) {

case C:

case U:

remove(posOfLabel);

case R:

remove(posOfLabel);

case E:

case O:

remove(posOfLabel);

}

repaint();

revalidate();

}

/\*\*

\* Додає новий елемент в ЛСА.

\*

\* @param jLabel

\*/

public void addUnitToLSA(JLabel jLabel) {

int pos = getPosOfItemInLSA(jLabel);

if (pos == lsa.getLsaItems().size()) {

lsa.getLsaItems().add(this.labelUnitHashMap.get(jLabel));

} else {

lsa.getLsaItems().add(pos, this.labelUnitHashMap.get(jLabel));

}

parentPanel.updateTables();

}

/\*\*

\* Видаляє елемнет з ЛСА.

\*

\* @param jLabel

\*/

public void removeUnitFromLSA(JLabel jLabel) {

lsa.getLsaItems().remove(this.labelUnitHashMap.get(jLabel));

parentPanel.updateTables();

}

/\*\*

\* Повретає абсолютну позицію елемента ЛСА, якому відповідає елемент панелі рисування.

\*

\* @param jLabel - елемент панелі рисування

\* @return - абсолютна позиція елементу в масиві елементів ЛСА

\*/

private int getPosOfItemInLSA(JLabel jLabel) {

int i = 0;

int newPos = 0;

while (!jLabel.equals(getComponent(i))) {

UnitType unitType = getUnitTypeFromHashmap((JLabel) getComponent(i));

switch (unitType) {

case B:

case O:

case E:

i++;

break;

case C:

case U:

i += 3;

break;

case R:

i += 2;

break;

}

newPos++;

}

return newPos;

}

/\*\*

\* Встановлення нового індексу елементу ЛСА.

\*

\* @param jLabel

\* @param newIndex

\* @param unitType

\*/

public void changeIndexUnit(JLabel jLabel, int newIndex, int unitType) {

//unitType == 1 : return item

//unitType ==2 : unconditional item

JLabel editedLabel = (JLabel) getComponent(getPositionOfElement(jLabel) + unitType);

editedLabel.setText("" + newIndex);

editedLabel.setPreferredSize(new Dimension(((int) Math.log10((double) newIndex) + 1) \* 10, 50));

}

/\*\*

\* Встановлення нового ЛСА.

\*

\* @param lsa

\*/

public void setLsa(LSA lsa) {

this.lsa = lsa;

}

/\*\*

\* Повертає дану ЛСА.

\*

\* @return

\*/

public LSA getLsa() {

return lsa;

}

/\*\*

\* Оновлює елементи ЛСА та таблиць відображення операторних та умовних вершин.

\*/

public void updateAll() {

parentPanel.updateTables();

parentPanel.updateLSAPanel();

}

}

package lab1.gui;

import lab1.LSA;

import lab1.Unit;

import lab1.UnitType;

import javax.swing.\*;

import javax.swing.border.TitledBorder;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.Set;

public class EditItem extends JDialog {

private GridBagConstraints gridBagConstraints;

private Unit editUnit;

private JLabel selectedLabel;

private DrawPanel drawPanel;

private Set<JCheckBox> checkBoxes;

private Set<JCheckBox> rightCheckBoxes = new HashSet<JCheckBox>();

private JPanel textFieldOperation = new JPanel();

private JPanel textFieldSignal = new JPanel();

private LSA lsa;

private String[][] operSignals;

private JPanel conditionsPanel = new JPanel();

private ButtonGroup buttonGroup = new ButtonGroup();

private JTextField jTextField;

private JTextField ifield;

private JRadioButton activeRadioButton;

private boolean isEdited = false;

/\*\*

\* Конструктор класу для редагування елементів ЛСА.

\*

\* @param unit

\* @param selectedLabel

\* @param drawPanel

\*/

public EditItem(Unit unit, JLabel selectedLabel, DrawPanel drawPanel) {

editUnit = unit;

this.selectedLabel = selectedLabel;

this.drawPanel = drawPanel;

lsa = drawPanel.getLsa();

setModal(true);

setResizable(false);

setLayout(new GridBagLayout());

gridBagConstraints = new GridBagConstraints();

UnitType unitType = unit.getUnitType();

switch (unitType) {

case O:

createOperationEditPanel();

break;

case C:

createConditionEditPanel();

break;

case U:

createUnconditionEditPanel();

break;

case R:

createReturnEditPanel();

break;

}

setVisible(true);

}

/\*\*

\* Створює панель редагування операційної вершини.

\*/

private void createOperationEditPanel() {

setTitle("Операційний вузол");

setSize\_Position(UnitType.O);

operSignals = lsa.getOperationUnitsArray();

setLayout(new GridBagLayout());

JLabel labelOperation = new JLabel("Позначення -> сигнал");

JScrollPane scrollPaneOperation = new JScrollPane(textFieldOperation);

textFieldOperation.setLayout(new BoxLayout(textFieldOperation, BoxLayout.Y\_AXIS));

checkBoxes = new HashSet<JCheckBox>();

for (int i = 0; i < lsa.getOperationUnitsArray()[0].length; i++) {

JCheckBox jCheckBox = new JCheckBox((lsa.getOperationUnitsArray()[0][i] + " -> " + lsa.getOperationUnitsArray()[1][i]), false);

if (isThisOperation(lsa.getOperationUnitsArray()[0][i])) {

textFieldSignal.add(jCheckBox);

rightCheckBoxes.add(jCheckBox);

jCheckBox.setSelected(true);

} else {

checkBoxes.add(jCheckBox);

textFieldOperation.add(jCheckBox);

}

}

JLabel labelSignal = new JLabel("Сигнали у вузлі");

JScrollPane scrollPaneSignal = new JScrollPane(textFieldSignal);

textFieldSignal.setLayout(new BoxLayout(textFieldSignal, BoxLayout.Y\_AXIS));

JPanel verticalPanel = new JPanel();

verticalPanel.setLayout(new GridLayout(2, 1));

JButton addButton = new JButton(">");

addButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

for (JCheckBox checkbox : checkBoxes) {

if (checkbox.isSelected()) {

textFieldSignal.add(checkbox);

rightCheckBoxes.add(checkbox);

isEdited = true;

}

}

repaint();

revalidate();

}

});

JButton removeButton = new JButton("<");

removeButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

Set<JCheckBox> tempCB = new HashSet<JCheckBox>();

for (JCheckBox checkbox : rightCheckBoxes) {

if (checkbox.isSelected()) {

textFieldOperation.add(checkbox);

checkBoxes.add(checkbox);

tempCB.add(checkbox);

isEdited = true;

}

}

for (JCheckBox jCheckBox : tempCB) {

rightCheckBoxes.remove(jCheckBox);

}

repaint();

revalidate();

}

});

verticalPanel.add(addButton);

verticalPanel.add(removeButton);

gridBagConstraints.anchor = GridBagConstraints.NORTH;

setGridBagParam(0, 0, 1, 1, 10, 1);

add(labelOperation, gridBagConstraints);

setGridBagParam(2, 0, 1, 1, 20, 1);

add(labelSignal, gridBagConstraints);

gridBagConstraints.fill = GridBagConstraints.BOTH;

setGridBagParam(0, 1, 1, 1, 20, 50);

add(scrollPaneOperation, gridBagConstraints);

setGridBagParam(1, 1, 1, 1, 1, 50);

add(verticalPanel, gridBagConstraints);

setGridBagParam(2, 1, 1, 1, 10, 50);

add(scrollPaneSignal, gridBagConstraints);

// setGridBagParam(0, 2, 1, 1, 1, 2);

// add(addNewSignalButton, gridBagConstraints);

JButton okButton = new JButton("Добре");

okButton.setBackground(Color.green);

okButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

if (isEdited) {

if (rightCheckBoxes.size() != 0) {

setThisItemSignals(rightCheckBoxes);

drawPanel.updateAll();

setVisible(false);

} else {

JOptionPane.showMessageDialog(null, "У вузлі відсутні сигнали! Додайте хоча б один сигнал.");

}

} else {

setVisible(false);

}

}

}

);

setGridBagParam(0, 3, 1, 1, 1, 2);

add(okButton, gridBagConstraints);

JButton cancelButton = new JButton("Відміна");

cancelButton.setBackground(Color.green);

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

setGridBagParam(1, 3, 2, 1, 1, 2);

add(cancelButton, gridBagConstraints);

}

/\*\*

\* Створює панель редагування логічних вершин.

\*/

private void createConditionEditPanel() {

setTitle("Редагування логічного вузла");

setSize\_Position(UnitType.C);

// setBorder(new TitledBorder("Оберіть логічний сигнал "));

// condSignals = lsa.getConditionUnitsArray();

setLayout(new GridBagLayout());

JLabel labelOperation = new JLabel("Позначення -> Умова");

JScrollPane scrollPaneOperation = new JScrollPane(conditionsPanel);

conditionsPanel.setLayout(new BoxLayout(conditionsPanel, BoxLayout.Y\_AXIS));

for (int i = 0; i < lsa.getConditionUnitsArray()[0].length; i++) {

final JRadioButton jRadioButton = new JRadioButton((lsa.getConditionUnitsArray()[0][i] + " -> " + lsa.getConditionUnitsArray()[1][i]), false);

if (lsa.getConditionUnitsArray()[0][i].equals(editUnit.getUnitMarking()[0])) {

jRadioButton.setSelected(true);

activeRadioButton = jRadioButton;

}

jRadioButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

activeRadioButton = jRadioButton;

}

});

buttonGroup.add(jRadioButton);

conditionsPanel.add(jRadioButton);

}

JLabel jConditionLabel = new JLabel("j = ");

jConditionLabel.setFont(new Font("TimesRoman", Font.BOLD, 20));

jTextField = new JTextField("" + editUnit.getIndex());

jTextField.setPreferredSize(new Dimension(getWidth() / 4, getHeight() / 6));

jTextField.setFont(new Font("TimesRoman", Font.BOLD, 20));

JPanel jPanel = new JPanel();

jPanel.add(jConditionLabel);

jPanel.add(jTextField);

gridBagConstraints.anchor = GridBagConstraints.NORTH;

setGridBagParam(0, 0, 2, 1, 10, 1);

add(labelOperation, gridBagConstraints);

gridBagConstraints.fill = GridBagConstraints.BOTH;

setGridBagParam(0, 1, 2, 1, 20, 50);

add(scrollPaneOperation, gridBagConstraints);

setGridBagParam(0, 3, 2, 1, 1, 2);

add(jPanel, gridBagConstraints);

JButton okButton = new JButton("Добре");

okButton.setBackground(Color.green);

okButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String[] temp = activeRadioButton.getText().split(" -> ", 2);

String[] newUnitMarking = {temp[0]};

String[] newUnitCondition = {temp[1]};

editUnit.setUnitMarking(newUnitMarking);

editUnit.setUnitValue(newUnitCondition);

try {

int newIndex = Integer.parseInt(jTextField.getText().trim());

if (newIndex < 0) {

throw new Exception();

}

editUnit.setIndex(newIndex);

drawPanel.updateAll();

setVisible(false);

} catch (Exception exc) {

JOptionPane.showMessageDialog(null, "Невірний формат j ! Введіть ціле додатнє число");

}

}

}

);

setGridBagParam(0, 4, 1, 1, 1, 1);

add(okButton, gridBagConstraints);

JButton cancelButton = new JButton("Відміна");

cancelButton.setBackground(Color.green);

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

setGridBagParam(1, 4, 1, 1, 1, 1);

add(cancelButton, gridBagConstraints);

}

/\*\*

\* Створює панель редагування безумовних переходів.

\*/

private void createUnconditionEditPanel() {

setTitle("Безумовний перехід");

setSize\_Position(UnitType.U);

JPanel unconditionPanel = new JPanel();

unconditionPanel.setBorder(new TitledBorder("Укажіть номер безумовного переходу"));

JLabel ilabel = new JLabel("i = ");

ilabel.setFont(new Font("TimesRoman", Font.BOLD, 25));

ifield = new JTextField("" + editUnit.getIndex());

ifield.setPreferredSize(new Dimension(getWidth() / 2, getHeight() / 3));

ifield.setFont(new Font("TimesRoman", Font.BOLD, 25));

ifield.setEditable(true);

unconditionPanel.add(ilabel);

unconditionPanel.add(ifield);

setGridBagParam(0, 0, 2, 1, 1, 5);

add(unconditionPanel, gridBagConstraints);

JButton okButton = new JButton("Добре");

okButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

try {

int newIndex = Integer.parseInt(ifield.getText().trim());

if (newIndex < 0) {

throw new Exception();

}

editUnit.setIndex(newIndex);

drawPanel.changeIndexUnit(selectedLabel, newIndex, 2);

setVisible(false);

} catch (Exception exc) {

JOptionPane.showMessageDialog(null, "Невірний формат і ! Введіть ціле додатнє число");

}

}

}

);

setGridBagParam(0, 1, 1, 1, 1, 1);

add(okButton, gridBagConstraints);

JButton cancelButton = new JButton("Відміна");

cancelButton.addActionListener(new

ActionListener() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

}

);

setGridBagParam(1, 1, 1, 1, 1, 1);

add(cancelButton, gridBagConstraints);

}

/\*\*

\* Створює панель редагування елементу повернення.

\*/

private void createReturnEditPanel() {

setTitle("Повернення");

setSize\_Position(UnitType.R);

JPanel unconditionPanel = new JPanel();

unconditionPanel.setBorder(new TitledBorder("Укажіть номер повернення"));

JLabel ilabel = new JLabel("i = ");

ilabel.setFont(new Font("TimesRoman", Font.BOLD, 25));

ifield = new JTextField("" + editUnit.getIndex());

ifield.setPreferredSize(new Dimension(getWidth() / 2, getHeight() / 3));

ifield.setFont(new Font("TimesRoman", Font.BOLD, 25));

ifield.setEditable(true);

unconditionPanel.add(ilabel);

unconditionPanel.add(ifield);

setGridBagParam(0, 0, 2, 1, 1, 5);

add(unconditionPanel, gridBagConstraints);

JButton okButton = new JButton("Добре");

okButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

try {

int newIndex = Integer.parseInt(ifield.getText().trim());

if (newIndex < 0) {

throw new Exception();

}

editUnit.setIndex(newIndex);

drawPanel.changeIndexUnit(selectedLabel, newIndex, 1);

setVisible(false);

} catch (Exception exc) {

JOptionPane.showMessageDialog(null, "Невірний формат і ! Введіть ціле додатнє число");

}

}

}

);

setGridBagParam(0, 1, 1, 1, 1, 1);

add(okButton, gridBagConstraints);

JButton cancelButton = new JButton("Відміна");

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

setGridBagParam(1, 1, 1, 1, 1, 1);

add(cancelButton, gridBagConstraints);

}

/\*\*

\* Встановлює парметри вікна редагування елементів.

\*

\* @param unitType

\*/

private void setSize\_Position(UnitType unitType) {

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

int divInt;

if (unitType.equals(UnitType.O) || unitType.equals(UnitType.C)) {

divInt = 6;

} else {

divInt = 9;

}

int dialogWidth = screenSize.width \* 2 / divInt;

int dialogHeight = screenSize.height \* 2 / divInt;

int leftCornerH = (screenSize.height - dialogHeight) / 2;

int leftCornerW = (screenSize.width - dialogWidth) / 2;

setBounds(leftCornerW, leftCornerH, dialogWidth, dialogHeight);

setBounds(leftCornerW, leftCornerH, dialogWidth, dialogHeight);

}

/\*\*

\* Встановлення параметрів GridBagConstraints компоновщика GridBag

\*

\* @param gridx

\* @param gridy

\* @param gridwidth

\* @param gridheight

\* @param weightx

\* @param weighty

\*/

private void setGridBagParam(int gridx, int gridy, int gridwidth, int gridheight, int weightx, int weighty) {

gridBagConstraints.gridx = gridx;

gridBagConstraints.gridy = gridy;

gridBagConstraints.gridwidth = gridwidth;

gridBagConstraints.gridheight = gridheight;

gridBagConstraints.weightx = weightx;

gridBagConstraints.weighty = weighty;

gridBagConstraints.ipadx = 5;

gridBagConstraints.ipady = 5;

}

/\*\*

\* Первіряє чи даний сигнал існує в масиві сигналів ЛСА.

\*

\* @param mark

\* @return

\*/

private boolean isThisOperation(String mark) {

boolean flag = false;

for (String s : editUnit.getUnitMarking()) {

if (s.equals(mark)) {

flag = true;

break;

}

}

return flag;

}

/\*\*

\* Встановлення відредагованих сигналів у вершині.

\*

\* @param checkBoxes

\*/

public void setThisItemSignals(Set<JCheckBox> checkBoxes) {

ArrayList<String> marking = new ArrayList<String>();

ArrayList<String> thisSignals = new ArrayList<String>();

for (JCheckBox jCheckBox : checkBoxes) {

marking.add(jCheckBox.getText().split(" -> ", 2)[0]);

thisSignals.add(jCheckBox.getText().split(" -> ", 2)[1]);

}

String[] marks = new String[marking.size()];

String[] signals = new String[thisSignals.size()];

for (int i = 0; i < marking.size(); i++) {

marks[i] = marking.get(i);

signals[i] = thisSignals.get(i);

}

editUnit.setUnitMarking(marks);

editUnit.setUnitValue(signals);

}

}

package lab1.gui;

import lab1.file.NewProjectListener;

import lab1.file.OpenButtonListener;

import lab1.file.SaveButtonListener;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class LSAMenuBar {

/\*\*

\* Створює випадаюче меню для управління проектом.

\*

\* @param mainPanel

\* @return

\*/

public JMenuBar getMenuBar(MainPanel mainPanel) {

JMenuBar jMenuBar = new JMenuBar();

JMenu jMenu = new JMenu("Файл");

Icon newIcon = new ImageIcon("src/lab1/icons/new.png");

JMenuItem newProjectItem = new JMenuItem("Створити проект", newIcon);

newProjectItem.setToolTipText("Створити новий проект");

newProjectItem.addActionListener(new NewProjectListener(mainPanel));

Icon openIcon = new ImageIcon("src/lab1/icons/open.gif");

JMenuItem openProjectItem = new JMenuItem("Відкрити проект", openIcon);

openProjectItem.setToolTipText("Відкрити раніше створений проект");

openProjectItem.addActionListener(new OpenButtonListener(mainPanel));

Icon saveIcon = new ImageIcon("src/lab1/icons/save.png");

JMenuItem saveProjectItem = new JMenuItem("Зберегти проект", saveIcon);

saveProjectItem.setToolTipText("Зберегти проект");

saveProjectItem.addActionListener(new SaveButtonListener(mainPanel));

JMenuItem exitItem = new JMenuItem("Вийти");

exitItem.setToolTipText("Не збережені дані будуть втрачені!!!");

exitItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

System.exit(0);

}

});

jMenu.add(newProjectItem);

jMenu.add(openProjectItem);

jMenu.add(saveProjectItem);

jMenu.addSeparator();

jMenu.add(exitItem);

jMenuBar.add(jMenu);

return jMenuBar;

}

}

package lab1.gui;

import lab1.UnitType;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseAdapter;

import java.awt.event.MouseEvent;

public class LSAMouseListener extends MouseAdapter {

private DrawPanel parentPanel;

private Color selectedItemBackground = Color.getHSBColor(240, 300, 50);

private JLabel selLabel;

private boolean isPopupMenu = false;

/\*\*

\* Конструктор слухача миші для елементів ЛСА на панелі рисування.

\* @param drawPanel

\*/

public LSAMouseListener(DrawPanel drawPanel) {

this.parentPanel = drawPanel;

}

/\*\*

\* Дія при натисенні ЛКМ на елемент ЛСА.

\* @param e

\*/

public void mouseClicked(MouseEvent e) {

showSelectitem(e);

if ((e.getClickCount() >=2)&&!(parentPanel.getUnitFromHashmap(selLabel)).getUnitType().equals(UnitType.E)){

EditItem editItem = new EditItem(parentPanel.getUnitFromHashmap(selLabel), selLabel, parentPanel);

}

if (SwingUtilities.isRightMouseButton(e)) {

showSelectitem(e);

isPopupMenu = true;

createPopupMenu().show(parentPanel.getSelectedLabel(), e.getX(), e.getY());

}

}

/\*\*

\* Дія при наведенні курсору миші на елемент ЛСА.

\* @param e

\*/

public void mouseEntered(MouseEvent e) {

if (!isPopupMenu) {

showSelectitem(e);

}

}

/\*\*

\* Дія при виведенні курсору миші з елементу ЛСА.

\* @param e

\*/

public void mouseExited(MouseEvent e) {

if (!isPopupMenu) {

parentPanel.deselect();

}

}

/\*\*

\* Виділяє обраний елемент ЛСА.

\* @param e

\*/

private void showSelectitem(MouseEvent e) {

selLabel = parentPanel.getSelectedLabel();

// прибараємо заливку раніше обраного елементу

selLabel.setBackground(parentPanel.getBackground());

if ((selLabel.getText().equals("↓"))) {

parentPanel.getComponent(parentPanel.getPositionOfElement(selLabel) + 1).setBackground(parentPanel.getBackground());

}

selLabel = (JLabel) e.getSource();

parentPanel.setSelectedLabel(selLabel);

// встановлюємо заливку обраного елементу

if ((selLabel).getText().equals("↓")) {

parentPanel.getComponent(parentPanel.getPositionOfElement(selLabel) + 1).setBackground(selectedItemBackground);

}

selLabel.setBackground(selectedItemBackground);

}

/\*\*

\* Стврює контексне меню для елементів ЛСА.

\* @return

\*/

private JPopupMenu createPopupMenu() {

JPopupMenu jPopupMenu = new JPopupMenu();

JMenuItem editItem = new JMenuItem("Редагувати елемент");

if (parentPanel.getUnitTypeFromHashmap(selLabel).equals(UnitType.E)){

editItem.setEnabled(false);

}

editItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

EditItem editItem = new EditItem(parentPanel.getUnitFromHashmap(selLabel), selLabel, parentPanel);

}

});

JMenuItem cutItem = new JMenuItem("Вирізати");

cutItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JLabel bLabel = parentPanel.getBufferedLabel();

if (!parentPanel.getUnitTypeFromHashmap(selLabel).equals(UnitType.B)) {

if (bLabel != null) {

if (!parentPanel.getUnitTypeFromHashmap(bLabel).equals(UnitType.R)) {

bLabel.setBorder(BorderFactory.createLineBorder(Color.blue));

} else {

bLabel.setBorder(BorderFactory.createLineBorder(parentPanel.getBackground()));

}

}

parentPanel.setBufferedLabel(selLabel);

selLabel.setBorder(BorderFactory.createLineBorder(Color.gray));

}

isPopupMenu = false;

}

});

JMenuItem pasteOnLeft = new JMenuItem("← Вставити зліва");

if (parentPanel.getBufferedLabel() == null) {

pasteOnLeft.setEnabled(false);

}

pasteOnLeft.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JLabel bLabel = parentPanel.getBufferedLabel();

UnitType unitType = parentPanel.getUnitTypeFromHashmap(bLabel);

switch (unitType) {

case O:

case E:

if (!bLabel.equals(selLabel)) {

parentPanel.removeLabel(bLabel);

parentPanel.removeUnitFromLSA(bLabel);

parentPanel.add(bLabel, parentPanel.getPositionOfElement(selLabel));

parentPanel.addUnitToLSA(bLabel);

}

parentPanel.removeBufferedItem();

break;

case C:

case U:

JLabel tempLabel1 = (JLabel) parentPanel.getComponent(parentPanel.getPositionOfElement(bLabel) + 1);

JLabel tempLabel2 = (JLabel) parentPanel.getComponent(parentPanel.getPositionOfElement(bLabel) + 2);

if (!bLabel.equals(selLabel)) {

parentPanel.removeLabel(bLabel);

parentPanel.removeUnitFromLSA(bLabel);

parentPanel.add(bLabel, parentPanel.getPositionOfElement(selLabel));

parentPanel.add(tempLabel1, parentPanel.getPositionOfElement(selLabel));

parentPanel.add(tempLabel2, parentPanel.getPositionOfElement(selLabel));

parentPanel.addUnitToLSA(bLabel);

}

parentPanel.removeBufferedItem();

break;

case R:

JLabel tempLabelR = (JLabel) parentPanel.getComponent(parentPanel.getPositionOfElement(bLabel) + 1);

if (!bLabel.equals(selLabel)) {

parentPanel.removeLabel(bLabel);

parentPanel.removeUnitFromLSA(bLabel);

parentPanel.add(bLabel, parentPanel.getPositionOfElement(selLabel));

parentPanel.add(tempLabelR, parentPanel.getPositionOfElement(selLabel));

parentPanel.addUnitToLSA(bLabel);

}

parentPanel.removeBufferedItem();

break;

}

isPopupMenu = false;

}

});

JMenuItem pasteOnRight = new JMenuItem("Вставити справа →");

if ((parentPanel.getBufferedLabel() == null)) {

pasteOnRight.setEnabled(false);

}

pasteOnRight.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JLabel bLabel = parentPanel.getBufferedLabel();

UnitType unitType = parentPanel.getUnitTypeFromHashmap(bLabel);

switch (unitType) {

case O:

case E:

if (!bLabel.equals(selLabel)) {

parentPanel.removeLabel(bLabel);

parentPanel.removeUnitFromLSA(bLabel);

if (getPositionOfSelectedItem() == -1) {

parentPanel.add(bLabel);

parentPanel.addUnitToLSA(bLabel);

} else {

parentPanel.add(bLabel, getPositionOfSelectedItem());

parentPanel.addUnitToLSA(bLabel);

}

}

parentPanel.removeBufferedItem();

break;

case C:

case U:

JLabel tempLabel1 = (JLabel) parentPanel.getComponent(parentPanel.getPositionOfElement(bLabel) + 1);

JLabel tempLabel2 = (JLabel) parentPanel.getComponent(parentPanel.getPositionOfElement(bLabel) + 2);

if (!bLabel.equals(selLabel)) {

parentPanel.removeLabel(bLabel);

parentPanel.removeUnitFromLSA(bLabel);

if (getPositionOfSelectedItem() == -1) {

parentPanel.add(bLabel);

parentPanel.add(tempLabel1);

parentPanel.add(tempLabel2);

} else {

parentPanel.add(bLabel, getPositionOfSelectedItem());

parentPanel.add(tempLabel1, getPositionOfSelectedItem() + 1);

parentPanel.add(tempLabel2, getPositionOfSelectedItem() + 2);

parentPanel.addUnitToLSA(bLabel);

}

}

parentPanel.removeBufferedItem();

break;

case R:

JLabel tempLabelR = (JLabel) parentPanel.getComponent(parentPanel.getPositionOfElement(bLabel) + 1);

if (!bLabel.equals(selLabel)) {

parentPanel.removeLabel(bLabel);

parentPanel.removeUnitFromLSA(bLabel);

if (getPositionOfSelectedItem() == -1) {

parentPanel.add(bLabel);

parentPanel.add(tempLabelR);

parentPanel.addUnitToLSA(bLabel);

} else {

parentPanel.add(bLabel, getPositionOfSelectedItem());

parentPanel.add(tempLabelR, getPositionOfSelectedItem() + 1);

parentPanel.addUnitToLSA(bLabel);

}

}

parentPanel.removeBufferedItem();

break;

}

isPopupMenu = false;

}

});

JMenuItem copyItem = new JMenuItem("Копіювати");

copyItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

JLabel bLabel = parentPanel.getBufferedLabel();

if (bLabel != null) {

if (!bLabel.getText().equals("↓")) {

parentPanel.getBufferedLabel().setBorder(BorderFactory.createLineBorder(Color.blue));

} else {

bLabel.setBorder(BorderFactory.createLineBorder(parentPanel.getBackground()));

}

}

parentPanel.setBufferedLabel(selLabel);

selLabel.setBorder(BorderFactory.createLineBorder(Color.pink));

isPopupMenu = false;

}

});

JMenuItem removeItem = new JMenuItem("Видалити");

removeItem.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

parentPanel.removeLabel(selLabel);

parentPanel.removeUnitFromLSA(selLabel);

parentPanel.removeBufferedItem();

isPopupMenu = false;

}

});

jPopupMenu.add(editItem);

jPopupMenu.add(new JSeparator());

jPopupMenu.add(cutItem);

jPopupMenu.add(pasteOnLeft);

jPopupMenu.add(pasteOnRight);

jPopupMenu.add(new JSeparator());

jPopupMenu.add(removeItem);

return jPopupMenu;

}

/\*\*

\* Повертає абсолютну позицію елементу в масиві елементів панелі рисування ЛСА.

\* @return

\*/

private int getPositionOfSelectedItem() {

int pos = parentPanel.getPositionOfElement(selLabel) + 1;

UnitType selUType = parentPanel.getUnitTypeFromHashmap(selLabel);

if (selUType.equals(UnitType.R)) {

pos++;

}

if ((selUType.equals(UnitType.U)) || (selUType.equals(UnitType.C))) {

pos += 2;

}

if (pos > parentPanel.getComponentCount()) {

return -1;

} else {

return pos;

}

}

}

package lab1.gui;

import lab1.LSA;

import lab1.Main;

import lab1.file.NewProjectListener;

import lab1.file.OpenButtonListener;

import lab1.file.SaveButtonListener;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

public class MainFrame extends JFrame {

private LSA lsa;

private MainPanel mainPanel;

public MainFrame() {

super();

}

/\*\*

\* Встановлення нового ЛСА для проекту.

\*

\* @param lsa

\*/

public void setLsa(LSA lsa) {

this.lsa = lsa;

}

/\*\*

\* Створення нового вікна проекту.

\*/

public void createMainFrame() {

setTitle("LSA Editor");

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setSize\_Position();

createMainPanel(lsa);

createMenuBar();

createToolBar();

pack();

setVisible(true);

}

/\*\*

\* Встановлення розмірів та позиції головного вікна.

\*/

private void setSize\_Position() {

// setResizable(false);

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

int frameWidth = screenSize.width \* 2 / 3;

int frameHeight = screenSize.height \* 2 / 3;

int leftCornerH = (screenSize.height - frameHeight) / 2;

int leftCornerW = (screenSize.width - frameWidth) / 2;

setBounds(leftCornerW, leftCornerH, frameWidth, frameHeight);

setLayout(new BorderLayout());

}

/\*\*

\* Створення головного меню.

\*/

private void createMenuBar() {

LSAMenuBar lsaMenuBar = new LSAMenuBar();

setJMenuBar(lsaMenuBar.getMenuBar(mainPanel));

}

/\*\*

\* Створення панелі інструментів.

\*/

private void createToolBar() {

JToolBar jtb = new JToolBar("Панель інструментів");

jtb.setOrientation(SwingConstants.HORIZONTAL);

//new

JButton newButton;

Icon newIcon = new ImageIcon("src/lab1/icons/new.png");

newButton = new JButton(newIcon);

newButton.setToolTipText("Створити новий проект");

newButton.addActionListener(new NewProjectListener(this.mainPanel));

jtb.add(newButton);

//open

JButton openButton;

Icon openIcon = new ImageIcon("src/lab1/icons/open.gif");

openButton = new JButton(openIcon);

openButton.setToolTipText("Відкрити раніше створений проект");

openButton.addActionListener(new OpenButtonListener(this.mainPanel));

jtb.add(openButton);

//save

Icon saveIcon = new ImageIcon("src/lab1/icons/save.png");

JButton saveButton = new JButton(saveIcon);

saveButton.setToolTipText("Зберегти проект");

saveButton.addActionListener(new SaveButtonListener(mainPanel));

jtb.add(saveButton);

JButton controlButton = new JButton("Перевірити алгоритм");

controlButton.setBorder(BorderFactory.createLineBorder(Color.YELLOW));

controlButton.setFont(new Font("TimesRoman", Font.BOLD, 14));

add(jtb, BorderLayout.NORTH);

}

/\*\*

\* Створення панелі для розміщення всіх елементів(панель рисуання, таблиці).

\*

\* @param lsa

\*/

private void createMainPanel(LSA lsa) {

MainPanel mainPanel = new MainPanel(lsa);

this.mainPanel = mainPanel;

add(mainPanel);

}

}

package lab1.gui;

import lab1.LSA;

import lab1.Unit;

import javax.swing.\*;

import java.awt.\*;

import java.util.ArrayList;

public class MainPanel extends JPanel {

private GridBagConstraints gridBagConstraints;

private TextPanel operationUnits;

private TextPanel conditionUnits;

private LSA thisLsa;

private DrawPanel drawPanel;

private JScrollPane scrollPane1;

/\*\*

\* Створення головної панелі.

\*

\* @param lsa

\*/

public MainPanel(LSA lsa) {

thisLsa = lsa;

GridBagLayout gridBagLayout = new GridBagLayout();

setLayout(gridBagLayout);

gridBagConstraints = new GridBagConstraints();

gridBagConstraints.fill = GridBagConstraints.BOTH;

// поле ЛСА

drawPanel = new DrawPanel(lsa, this);

scrollPane1 = new JScrollPane(drawPanel);

scrollPane1.setVerticalScrollBar(new JScrollBar());

setGridBagParam(0, 0, 2, 1, 2);

add(scrollPane1, gridBagConstraints);

//Поле операційних вершин

operationUnits = new TextPanel(TextPanelType.OPERATION, lsa, this);

setGridBagParam(0, 1, 1, 1, 5);

add(operationUnits, gridBagConstraints);

// Поле логічних умов

conditionUnits = new TextPanel(TextPanelType.CONDITION, lsa, this);

setGridBagParam(1, 1, 1, 1, 5);

add(conditionUnits, gridBagConstraints);

}

/\*\*

\* Встановлення параметрів GridBagConstraints компоновщика GridBag

\*

\* @param gridx

\* @param gridy

\* @param gridwidth

\* @param weightx

\* @param weighty

\*/

private void setGridBagParam(int gridx, int gridy, int gridwidth, int weightx, int weighty) {

gridBagConstraints.gridx = gridx;

gridBagConstraints.gridy = gridy;

gridBagConstraints.gridwidth = gridwidth;

gridBagConstraints.weightx = weightx;

gridBagConstraints.weighty = weighty;

}

/\*\*

\* Оновлення інформації в таблицях відповідності вершин та сигналів.

\*/

public void updateTables() {

operationUnits.updateTable(thisLsa);

conditionUnits.updateTable(thisLsa);

repaint();

revalidate();

}

/\*\*

\* Оновлення панелі рисування.

\*/

public void updateLSAPanel() {

drawPanel.setLsa(thisLsa);

drawPanel.removeAll();

ArrayList<Unit> lsaItems = (ArrayList<Unit>) thisLsa.getLsaItems().clone();

thisLsa.getLsaItems().clear();

for (Unit unit : lsaItems) {

thisLsa.addItem(unit);

drawPanel.drawLsa(unit.getUnitType());

}

repaint();

revalidate();

}

/\*\*

\* Встановлення нового ЛСА для проекту.

\*

\* @param lsa

\*/

public void setThisLsa(LSA lsa) {

thisLsa = lsa;

}

/\*\*

\* Повертає ЛСА цього проекту.

\*

\* @return

\*/

public LSA getThisLsa() {

return thisLsa;

}

/\*\*

\* Виводить ЛСА в стандартний потік виводу.

\*/

public void prLSA() {

thisLsa.printLSA();

}

}

package lab1.gui;

import lab1.LSA;

import lab1.Unit;

import lab1.UnitType;

import javax.swing.\*;

import javax.swing.border.TitledBorder;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.net.URL;

import java.util.ArrayList;

import java.util.Set;

public class NewItemDialog extends JDialog implements ActionListener {

private DrawPanel parentPanel;

private JLabel picture;

private NewItemsPanels newItemsPanels;

private JPanel tempPanel;

private JPanel operationPanel;

private JPanel conditionPanel;

private JPanel unconditionPanel;

private JPanel returnPanel;

private JPanel endPanel;

private GridBagConstraints gridBagConstraints;

private UnitType unitType = UnitType.O;

private LSA lsa;

private String[] marks = new String[0];

private String[] signals = new String[0];

/\*\*

\* Діалог для створення новго елементу ЛСА.

\* @param drawPanel

\* @param lsa

\*/

NewItemDialog(DrawPanel drawPanel, LSA lsa) {

this.parentPanel = drawPanel;

this.lsa = lsa;

setTitle("Новий елемент");

setModal(true);

setResizable(false);

setSize\_Position();

setLayout(new GridBagLayout());

gridBagConstraints = new GridBagConstraints();

createPanels();

addElementsToDialog();

setVisible(true);

}

/\*\*

\* Встановлення параметрів вікна створення нового елементу ЛСА.

\*/

private void setSize\_Position() {

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

int dialogWidth = screenSize.width \* 2 / 6;

int dialogHeight = screenSize.height \* 2 / 3;

int leftCornerH = (screenSize.height - dialogHeight) / 2;

int leftCornerW = (screenSize.width - dialogWidth) / 2;

setBounds(leftCornerW, leftCornerH, dialogWidth, dialogHeight);

setBounds(leftCornerW, leftCornerH, dialogWidth, dialogHeight);

}

/\*\*

\* Додає елементи редегування у вікно створення нового елементу.

\*/

private void addElementsToDialog() {

JPanel container = new JPanel();

container.setLayout(new GridLayout(1, 2));

JPanel typeOfNewElement = new JPanel();

typeOfNewElement.setBorder(new TitledBorder("Тип нового елементу"));

typeOfNewElement.setLayout(new GridLayout(5, 1));

JRadioButton operationButton = new JRadioButton("Операційний вузол", true);

JRadioButton conditionButton = new JRadioButton("Логічний перехід ", false);

JRadioButton gotoButton = new JRadioButton("Безумовний перехід", false);

JRadioButton returnButton = new JRadioButton("Повернення", false);

JRadioButton endButton = new JRadioButton("Кінцева вершина", false);

operationButton.setActionCommand(UnitType.O.toString());

conditionButton.setActionCommand(UnitType.C.toString());

gotoButton.setActionCommand(UnitType.U.toString());

returnButton.setActionCommand(UnitType.R.toString());

endButton.setActionCommand(UnitType.E.toString());

operationButton.addActionListener(this);

conditionButton.addActionListener(this);

gotoButton.addActionListener(this);

returnButton.addActionListener(this);

endButton.addActionListener(this);

ButtonGroup bgroup = new ButtonGroup();

bgroup.add(operationButton);

bgroup.add(conditionButton);

bgroup.add(gotoButton);

bgroup.add(returnButton);

bgroup.add(endButton);

typeOfNewElement.add(operationButton);

typeOfNewElement.add(conditionButton);

typeOfNewElement.add(gotoButton);

typeOfNewElement.add(returnButton);

typeOfNewElement.add(endButton);

setGridBagParam(0, 0, 1, 1, 1, 1);

gridBagConstraints.fill = GridBagConstraints.HORIZONTAL;

gridBagConstraints.anchor = GridBagConstraints.NORTH;

container.add(typeOfNewElement);

picture = new JLabel(createImageIcon(UnitType.O.toString()));

container.add(picture);

add(container, gridBagConstraints);

setGridBagParam(0, 1, 1, 1, 1, 50);

gridBagConstraints.fill = GridBagConstraints.BOTH;

gridBagConstraints.anchor = GridBagConstraints.CENTER;

tempPanel = operationPanel;

add(tempPanel, gridBagConstraints);

JPanel okCancelPanel = new JPanel();

JButton okButton = new JButton("Додати");

JButton cancelButton = new JButton("Відміна");

okCancelPanel.add(okButton);

okCancelPanel.add(cancelButton);

setGridBagParam(0, 2, 1, 1, 1, 1);

add(okCancelPanel, gridBagConstraints);

okButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

switch (unitType) {

case O:

if (marks.length != 0) {

createNewItem(0);

setVisible(false);

} else {

JOptionPane.showMessageDialog(null, "У вузлі немає жодного сигналу! Додайте хоча б один сигнал.");

}

break;

case C:

if (marks.length != 0) {

try {

String conditionNumb = newItemsPanels.getJString();

int cNumbInt = Integer.parseInt(conditionNumb);

if (cNumbInt <0){

throw new NumberFormatException();

}

createNewItem(cNumbInt);

setVisible(false);

} catch (NumberFormatException err) {

JOptionPane.showMessageDialog(null, "Невірний формат j! Введіть ціле додатнє число.");

}

} else {

JOptionPane.showMessageDialog(null, "Не вказана логічна умова!");

}

break;

case U:

try {

String unconditionNumb = newItemsPanels.getUnconditionalIndex();

int uncNumbInt = Integer.parseInt(unconditionNumb);

if (uncNumbInt <0){

throw new NumberFormatException();

}

createNewItem(uncNumbInt);

setVisible(false);

} catch (NumberFormatException err) {

JOptionPane.showMessageDialog(null, "Невірний формат i! Введіть ціле додатнє число.");

}

break;

case R:

try {

String returnNumb = newItemsPanels.getRetrunIndex();

int retNumbInt = Integer.parseInt(returnNumb);

if (retNumbInt <0){

throw new NumberFormatException();

}

//перевірка, чи не існує вже такий індекс

boolean isIndex = false;

for (Unit unit:lsa.getLsaItems()){

if ((unit.getUnitType().equals(UnitType.R))&&(unit.getIndex()==retNumbInt)){

isIndex = true;

break;

}

}

if (!isIndex){

createNewItem(retNumbInt);

setVisible(false);

} else {

JOptionPane.showMessageDialog(null, "Елемент поверення з індексом і = "+retNumbInt+" вже існує! Введіть інший індекс.");

}

} catch (NumberFormatException err) {

JOptionPane.showMessageDialog(null, "Невірний формат i! Введіть ціле додатнє число.");

}

break;

case E:

if (checkLsaForEndItem()) {

createNewItem(0);

setVisible(false);

} else {

JOptionPane.showMessageDialog(null, "Кінцева вершина вже існує! Оберіть інший тип вузла.");

}

break;

}

}

});

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

}

/\*\*

\* Обробка події вибору типу нового елементу ЛСА,

\* @param e

\*/

public void actionPerformed(ActionEvent e) {

picture.setIcon(createImageIcon(e.getActionCommand()));

switch (e.getActionCommand().charAt(0)) {

case 'O':

setPanel(operationPanel);

unitType = UnitType.O;

break;

case 'C':

setPanel(conditionPanel);

unitType = UnitType.C;

break;

case 'U':

setPanel(unconditionPanel);

unitType = UnitType.U;

break;

case 'R':

setPanel(returnPanel);

unitType = UnitType.R;

break;

case 'E':

setPanel(endPanel);

unitType = UnitType.E;

break;

}

}

/\*\*

\* Створення зображення для відповідного типу елементу ЛСА.

\* @param path

\* @return

\*/

private static ImageIcon createImageIcon(String path) {

URL imgURL = NewItemDialog.class.getResource("images/" + path + ".png");

ImageIcon imageIcon = new ImageIcon(imgURL);

return imageIcon;

}

/\*\*

\* Створення панелей для створення необхідного типу елементу ЛСА.

\*/

private void createPanels() {

newItemsPanels = new NewItemsPanels();

operationPanel = newItemsPanels.getOperationPanel(lsa, this);

conditionPanel = newItemsPanels.getConditionPanel(lsa, this);

unconditionPanel = newItemsPanels.getUnconditionPanel();

returnPanel = newItemsPanels.getReturnPanel();

endPanel = newItemsPanels.getEndPanel();

}

/\*\*

\* Встановлення параметрів GridBagConstraints компоновщика GridBag

\* @param gridx

\* @param gridy

\* @param gridwidth

\* @param gridheight

\* @param weightx

\* @param weighty

\*/

private void setGridBagParam(int gridx, int gridy, int gridwidth, int gridheight, int weightx, int weighty) {

gridBagConstraints.gridx = gridx;

gridBagConstraints.gridy = gridy;

gridBagConstraints.gridwidth = gridwidth;

gridBagConstraints.gridheight = gridheight;

gridBagConstraints.weightx = weightx;

gridBagConstraints.weighty = weighty;

gridBagConstraints.ipadx = 5;

gridBagConstraints.ipady = 5;

}

/\*\*

\* Додання панелі створення нового елементу ЛСА.

\* @param needPanel

\*/

private void setPanel(JPanel needPanel) {

remove(tempPanel);

tempPanel = needPanel;

setGridBagParam(0, 1, 1, 1, 1, 50);

gridBagConstraints.fill = GridBagConstraints.BOTH;

gridBagConstraints.anchor = GridBagConstraints.CENTER;

add(tempPanel, gridBagConstraints);

repaint();

revalidate();

}

/\*\*

\* Створює новий елемент ЛСА.

\* @param idx

\*/

public void createNewItem(int idx) {

Unit newUnit = new Unit(unitType, this.marks, this.signals, idx);

this.lsa.addItem(newUnit);

parentPanel.drawLsa(unitType);

}

/\*\*

\* Встановлення сигналів поточного лементу ЛСА.

\* @param checkBoxes

\*/

public void setThisItemSignals(Set<JCheckBox> checkBoxes) {

ArrayList<String> marking = new ArrayList<String>();

ArrayList<String> thisSignals = new ArrayList<String>();

for (JCheckBox jCheckBox : checkBoxes) {

marking.add(jCheckBox.getText().split(" -> ", 2)[0]);

thisSignals.add(jCheckBox.getText().split(" -> ", 2)[1]);

}

this.marks = new String[marking.size()];

this.signals = new String[thisSignals.size()];

for (int i = 0; i < marking.size(); i++) {

this.marks[i] = marking.get(i);

this.signals[i] = thisSignals.get(i);

}

}

/\*\*

\* Встановлення необхідної логічної умови в даному логічному елементі ЛСА.

\*/

public void setActiveCondition(JRadioButton radioButton) {

ArrayList<String> marking = new ArrayList<String>();

ArrayList<String> thisSignals = new ArrayList<String>();

marking.add(radioButton.getText().split(" -> ", 2)[0]);

thisSignals.add(radioButton.getText().split(" -> ", 2)[1]);

this.marks = new String[1];

this.signals = new String[1];

this.marks[0] = marking.get(0);

this.signals[0] = thisSignals.get(0);

}

/\*\*

\* Перевірка, чи є вже кінцева вершина в алгоритмі.

\* @return

\*/

private boolean checkLsaForEndItem() {

boolean flag = true;

int count = 0;

for (Unit unit : lsa.getLsaItems()) {

if (unit.getUnitType().equals(UnitType.E)) {

flag = false;

break;

}

}

return flag;

}

}

package lab1.gui;

import lab1.LSA;

import lab1.UnitType;

import javax.swing.\*;

import javax.swing.border.TitledBorder;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.HashSet;

import java.util.Set;

public class NewItemsPanels {

private GridBagConstraints gridBagConstraints = new GridBagConstraints();

private Set<JCheckBox> checkBoxes;

private Set<JCheckBox> rightCheckBoxes = new HashSet<JCheckBox>();

private JPanel operationPanel = new JPanel();

private JPanel conditionPanel = new JPanel();

private JPanel textFieldOperation = new JPanel();

private JPanel textFieldSignal = new JPanel();

private LSA lsa;

private String[][] operSignals;

private NewItemDialog newItemDialogParent;

private JPanel conditionsPanel = new JPanel();

private String[][] condSignals;

private JPanel textFieldCSignal = new JPanel();

private ButtonGroup buttonGroup = new ButtonGroup();

private JTextField jTextField;

private JTextField ifield;

JTextField rfield;

/\*\*

\* Повертає панель для створення опеційної вершини.

\*

\* @param lsaa

\* @param newItemDialog

\* @return

\*/

public JPanel getOperationPanel(LSA lsaa, NewItemDialog newItemDialog) {

this.lsa = lsaa;

this.newItemDialogParent = newItemDialog;

operSignals = lsa.getOperationUnitsArray();

operationPanel.setLayout(new GridBagLayout());

JLabel labelOperation = new JLabel("Позначення -> сигнал");

JScrollPane scrollPaneOperation = new JScrollPane(textFieldOperation);

textFieldOperation.setLayout(new BoxLayout(textFieldOperation, BoxLayout.Y\_AXIS));

checkBoxes = new HashSet<JCheckBox>();

for (int i = 0; i < lsa.getOperationUnitsArray()[0].length; i++) {

JCheckBox jCheckBox = new JCheckBox((lsa.getOperationUnitsArray()[0][i] + " -> " + lsa.getOperationUnitsArray()[1][i]), false);

checkBoxes.add(jCheckBox);

textFieldOperation.add(jCheckBox);

}

JButton addNewSignalButton = new JButton("Новий сигнал");

addNewSignalButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

int oldSize = operSignals[0].length;

NewSignalDialog newSignalDialog = new NewSignalDialog(operSignals, UnitType.O);

operSignals = newSignalDialog.getNewSignals();

if (oldSize != operSignals[0].length) {

JCheckBox checkBox = new JCheckBox((operSignals[0][oldSize] + " -> " + operSignals[1][oldSize]), true);

checkBoxes.add(checkBox);

textFieldOperation.add(checkBox);

operationPanel.repaint();

operationPanel.revalidate();

}

}

});

JLabel labelSignal = new JLabel("Сигнали в новому вузлі");

JScrollPane scrollPaneSignal = new JScrollPane(textFieldSignal);

textFieldSignal.setLayout(new BoxLayout(textFieldSignal, BoxLayout.Y\_AXIS));

JPanel verticalPanel = new JPanel();

verticalPanel.setLayout(new GridLayout(2, 1));

JButton addButton = new JButton(">");

addButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

for (JCheckBox checkbox : checkBoxes) {

if (checkbox.isSelected()) {

textFieldSignal.add(checkbox);

rightCheckBoxes.add(checkbox);

}

}

newItemDialogParent.setThisItemSignals(rightCheckBoxes);

operationPanel.repaint();

operationPanel.revalidate();

}

});

JButton removeButton = new JButton("<");

removeButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

Set<JCheckBox> tempCB = new HashSet<JCheckBox>();

for (JCheckBox checkbox : rightCheckBoxes) {

if (checkbox.isSelected()) {

textFieldOperation.add(checkbox);

checkBoxes.add(checkbox);

tempCB.add(checkbox);

}

}

for (JCheckBox jCheckBox : tempCB) {

rightCheckBoxes.remove(jCheckBox);

}

newItemDialogParent.setThisItemSignals(rightCheckBoxes);

operationPanel.repaint();

operationPanel.revalidate();

}

});

verticalPanel.add(addButton);

verticalPanel.add(removeButton);

gridBagConstraints.anchor = GridBagConstraints.NORTH;

setGridBagParam(0, 0, 1, 1, 10, 1);

operationPanel.add(labelOperation, gridBagConstraints);

setGridBagParam(2, 0, 1, 1, 20, 1);

operationPanel.add(labelSignal, gridBagConstraints);

gridBagConstraints.fill = GridBagConstraints.BOTH;

setGridBagParam(0, 1, 1, 1, 20, 50);

operationPanel.add(scrollPaneOperation, gridBagConstraints);

setGridBagParam(1, 1, 1, 1, 1, 50);

operationPanel.add(verticalPanel, gridBagConstraints);

setGridBagParam(2, 1, 1, 1, 10, 50);

operationPanel.add(scrollPaneSignal, gridBagConstraints);

setGridBagParam(0, 2, 1, 1, 1, 2);

operationPanel.add(addNewSignalButton, gridBagConstraints);

return operationPanel;

}

/\*\*

\* Повертає панель для створення логічної вершини.

\*

\* @param lsaa

\* @param newItemDialog

\* @return

\*/

public JPanel getConditionPanel(LSA lsaa, NewItemDialog newItemDialog) {

this.lsa = lsaa;

this.newItemDialogParent = newItemDialog;

conditionPanel.setBorder(new TitledBorder("Оберіть логічний сигнал "));

condSignals = lsa.getConditionUnitsArray();

conditionPanel.setLayout(new GridBagLayout());

JLabel labelOperation = new JLabel("Позначення -> Умова");

JScrollPane scrollPaneOperation = new JScrollPane(conditionsPanel);

conditionsPanel.setLayout(new BoxLayout(conditionsPanel, BoxLayout.Y\_AXIS));

for (int i = 0; i < lsa.getConditionUnitsArray()[0].length; i++) {

final JRadioButton jRadioButton = new JRadioButton((lsa.getConditionUnitsArray()[0][i] + " -> " + lsa.getConditionUnitsArray()[1][i]), false);

jRadioButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

newItemDialogParent.setActiveCondition(jRadioButton);

}

});

buttonGroup.add(jRadioButton);

conditionsPanel.add(jRadioButton);

}

JButton addNewSignalButton = new JButton("Нова умова");

addNewSignalButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

int oldSize = condSignals[0].length;

NewSignalDialog newSignalDialog = new NewSignalDialog(condSignals, UnitType.C);

condSignals = newSignalDialog.getNewSignals();

if (oldSize != condSignals[0].length) {

final JRadioButton jNewRadioButton = new JRadioButton((condSignals[0][oldSize] + " -> " + condSignals[1][oldSize]));

buttonGroup.add(jNewRadioButton);

jNewRadioButton.setSelected(true);

newItemDialogParent.setActiveCondition(jNewRadioButton);

conditionsPanel.add(jNewRadioButton);

jNewRadioButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

newItemDialogParent.setActiveCondition(jNewRadioButton);

}

});

conditionPanel.repaint();

conditionPanel.revalidate();

}

}

});

JLabel jConditionLabel = new JLabel("j = ");

jConditionLabel.setFont(new Font("TimesRoman", Font.BOLD, 20));

jTextField = new JTextField("1 ");

jTextField.setFont(new Font("TimesRoman", Font.BOLD, 20));

JPanel jPanel = new JPanel();

jPanel.add(jConditionLabel);

jPanel.add(jTextField);

gridBagConstraints.anchor = GridBagConstraints.NORTH;

setGridBagParam(0, 0, 1, 1, 10, 1);

conditionPanel.add(labelOperation, gridBagConstraints);

gridBagConstraints.fill = GridBagConstraints.BOTH;

setGridBagParam(0, 1, 1, 1, 20, 50);

conditionPanel.add(scrollPaneOperation, gridBagConstraints);

setGridBagParam(0, 2, 1, 1, 1, 2);

conditionPanel.add(addNewSignalButton, gridBagConstraints);

setGridBagParam(0, 3, 1, 1, 1, 2);

conditionPanel.add(jPanel, gridBagConstraints);

return conditionPanel;

}

/\*\*

\* Повертає панель для створення безумовного переходу.

\*

\* @return

\*/

public JPanel getUnconditionPanel() {

JPanel unconditionPanel = new JPanel();

unconditionPanel.setBorder(new TitledBorder("Укажіть номер безумовного переходу"));

JLabel ilabel = new JLabel("i = ");

ifield = new JTextField("1 ");

ifield.setEditable(true);

unconditionPanel.add(ilabel);

unconditionPanel.add(ifield);

return unconditionPanel;

}

/\*\*

\* Повертає панель для створення вершини повернення.

\*

\* @return

\*/

public JPanel getReturnPanel() {

JPanel returnPanel = new JPanel();

returnPanel.setBorder(new TitledBorder("Укажіть номер повернення"));

JLabel rlabel = new JLabel("i = ");

rfield = new JTextField("1 ");

rfield.setEditable(true);

returnPanel.add(rlabel);

returnPanel.add(rfield);

return returnPanel;

}

/\*\*

\* Повертає панель для створення кінцевої вершини.

\*

\* @return

\*/

public JPanel getEndPanel() {

JPanel endPanel = new JPanel();

return endPanel;

}

/\*\*

\* Встановлення параметрів GridBagConstraints компоновщика GridBag

\*

\* @param gridx

\* @param gridy

\* @param gridwidth

\* @param gridheight

\* @param weightx

\* @param weighty

\*/

private void setGridBagParam(int gridx, int gridy, int gridwidth, int gridheight, int weightx, int weighty) {

gridBagConstraints.gridx = gridx;

gridBagConstraints.gridy = gridy;

gridBagConstraints.gridwidth = gridwidth;

gridBagConstraints.gridheight = gridheight;

gridBagConstraints.weightx = weightx;

gridBagConstraints.weighty = weighty;

gridBagConstraints.ipadx = 5;

gridBagConstraints.ipady = 5;

}

/\*\*

\* Повертає текстове значення поля для введення індексу умовного переходу.

\*

\* @return

\*/

public String getJString() {

return this.jTextField.getText().trim();

}

/\*\*

\* Повертає текстове значення поля для введення індексу безумовного переходу.

\*

\* @return

\*/

public String getUnconditionalIndex() {

return this.ifield.getText().trim();

}

/\*\*

\* Повертає текстове значення поля для введення індексу повернення.

\*

\* @return

\*/

public String getRetrunIndex() {

return this.rfield.getText().trim();

}

}

package lab1.gui;

import lab1.UnitType;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class NewSignalDialog extends JDialog {

private GridBagConstraints gridBagConstraints = new GridBagConstraints();

private String[][] newSignals;

private JTextField markingField;

private String dialogName;

private String markingLabelName;

private String signalLabelName;

private String error1;

private String error2\_1;

private String error2\_2;

public NewSignalDialog(String[][] signals, UnitType unitType) {

this.newSignals = signals;

setStrings(unitType);

setTitle(dialogName);

setModal(true);

setResizable(false);

setSize\_Position();

setLayout(new GridBagLayout());

gridBagConstraints.anchor = GridBagConstraints.WEST;

gridBagConstraints.fill = GridBagConstraints.HORIZONTAL;

JLabel markingLabel = new JLabel(markingLabelName);

markingField = new JTextField();

JLabel signalLabel = new JLabel(signalLabelName);

final JTextField signalField = new JTextField();

JButton okButton = new JButton("Створити");

okButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String ff = checkSignal(markingField.getText());

if (ff.equals("0")) {

String[][] newOperSignals = new String[2][newSignals[0].length + 1];

for (int i = 0; i < newSignals[0].length; i++) {

newOperSignals[0][i] = newSignals[0][i];

newOperSignals[1][i] = newSignals[1][i];

}

newOperSignals[0][newSignals[0].length] = markingField.getText();

newOperSignals[1][newSignals[0].length] = signalField.getText();

newSignals = newOperSignals;

setVisible(false);

} else {

if (ff.equals("1")) {

JOptionPane.showMessageDialog(null, error1);

} else {

JOptionPane.showMessageDialog(null, (error2\_1 + markingField.getText() + error2\_2));

}

}

}

});

JButton cancelButton = new JButton("Відміна");

cancelButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

setVisible(false);

}

});

setGridBagParam(0, 0, 1, 1, 1, 1);

add(markingLabel, gridBagConstraints);

setGridBagParam(1, 0, 1, 1, 1, 1);

add(markingField, gridBagConstraints);

setGridBagParam(0, 1, 1, 1, 1, 1);

add(signalLabel, gridBagConstraints);

setGridBagParam(1, 1, 1, 1, 1, 1);

add(signalField, gridBagConstraints);

setGridBagParam(0, 2, 1, 1, 1, 1);

add(okButton, gridBagConstraints);

setGridBagParam(1, 2, 1, 1, 1, 1);

add(cancelButton, gridBagConstraints);

setVisible(true);

}

private void setSize\_Position() {

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

int dialogWidth = screenSize.width \* 2 / 6;

int dialogHeight = screenSize.height \* 2 / 10;

int leftCornerH = (screenSize.height - dialogHeight) / 2;

int leftCornerW = (screenSize.width - dialogWidth) / 2;

setBounds(leftCornerW, leftCornerH, dialogWidth, dialogHeight);

setBounds(leftCornerW, leftCornerH, dialogWidth, dialogHeight);

}

private void setGridBagParam(int gridx, int gridy, int gridwidth, int gridheight, int weightx, int weighty) {

gridBagConstraints.gridx = gridx;

gridBagConstraints.gridy = gridy;

gridBagConstraints.gridwidth = gridwidth;

gridBagConstraints.gridheight = gridheight;

gridBagConstraints.weightx = weightx;

gridBagConstraints.weighty = weighty;

gridBagConstraints.ipadx = 5;

gridBagConstraints.ipady = 5;

}

private String checkSignal(String marking) {

String flag = "0";

if (marking.equals("")) {

flag = "1";

}

for (int i = 0; i < this.newSignals[0].length; i++) {

if (marking.equals(newSignals[0][i])) {

flag = "2";

break;

}

}

return flag;

}

public String[][] getNewSignals() {

return newSignals;

}

private void setStrings(UnitType unitType) {

if (unitType.equals(UnitType.O)) {

dialogName = "Новий сигнал";

markingLabelName = "Позначення(Yi)";

signalLabelName = "Сигнал";

error1 = "Позначення сигналу не може бути порожнім! Введіть нове позначення";

error2\_1 = "Сигнал з позначенням ";

error2\_2 = " вже існує! Введіть інше позначення";

} else {

dialogName = "Нова логічна умова";

markingLabelName = "Позначення(Xi)";

signalLabelName = "Логічна умова";

error1 = "Позначення логічної умови не може бути порожнім! Введіть нове позначення";

error2\_1 = "Логічна умова з позначенням ";

error2\_2 = " вже існує! Введіть інше позначення";

}

}

}

package lab1.gui;

import lab1.LSA;

import javax.swing.\*;

import javax.swing.border.TitledBorder;

import java.awt.\*;

import java.awt.event.\*;

/\*\*

\* Типи таблиць відповідності позачень та сигналів.

\*/

enum TextPanelType {OPERATION, CONDITION};

public class TextPanel extends JPanel implements MouseListener, KeyListener {

private TextPanelType textPanelType;

private String[] columnNames = new String[2];

private String[][] data = new String[0][0];//{{"", ""}};

private LSA lsa;

private MainPanel parentPanel;

private TextPanelType itsTextPanelType;

private JTable table;

private JScrollPane scrollPane;

private int selectRow;

private int selectColumn;

private String selectCellString = null;

private String selectRowMarking = null;

private boolean isDoubleClick = false;

/\*\*

\* Створення таблиць відповідності позначень та сигналів.

\* @param textPanelType

\* @param lsa

\* @param pPanel

\*/

public TextPanel(TextPanelType textPanelType, LSA lsa, MainPanel pPanel) {

this.textPanelType = textPanelType;

this.itsTextPanelType = textPanelType;

this.lsa = lsa;

this.parentPanel = pPanel;

if (textPanelType.equals(TextPanelType.OPERATION)) {

if (lsa.getOperationUnitsArray()[0].length != 0) {

this.data = lsa.getOperationUnitsArray();

}

}

if (textPanelType.equals(TextPanelType.CONDITION)) {

if (lsa.getOperationUnitsArray()[0].length != 0) {

this.data = lsa.getConditionUnitsArray();

}

}

//--------------------

setLayout(new CardLayout());

if (textPanelType.equals(TextPanelType.OPERATION)) {

columnNames[0] = "Позначення";

columnNames[1] = "Сигнал";

setBorder(new TitledBorder("Оператори"));

} else {

columnNames[0] = "Позначення";

columnNames[1] = "Умова";

setBorder(new TitledBorder("Логічні умови"));

}

table = new JTable(data, columnNames);

table.addMouseListener(this);

table.addKeyListener(this);

scrollPane = new JScrollPane(table);

add(scrollPane);

}

/\*\*

\* Оновлення інформації в таблиці.

\* @param nlsa

\*/

public void updateTable(LSA nlsa) {

this.lsa = nlsa;

if (this.itsTextPanelType.equals(TextPanelType.OPERATION)) {

this.data = convertArray(lsa.getOperationUnitsArray());

} else {

this.data = convertArray(lsa.getConditionUnitsArray());

}

//

remove(scrollPane);

scrollPane.remove(table);

table = new JTable(data, columnNames);

table.addMouseListener(this);

table.addKeyListener(this);

scrollPane = new JScrollPane(table);

add(scrollPane);

repaint();

revalidate();

}

/\*\*

\* Транспонування матриці сигналів.

\* @param strings

\* @return

\*/

private String[][] convertArray(String[][] strings) {

String[][] tempStr = new String[strings[0].length][strings.length];

for (int i = 0; i < strings[0].length; i++) {

for (int j = 0; j < strings.length; j++) {

tempStr[i][j] = strings[j][i];

}

}

return tempStr;

}

/\*\*

\* Обробка події натиснення ЛКМ на чарунку таблиці.

\* @param e

\*/

public void mouseClicked(MouseEvent e) {

if (e.getButton() == MouseEvent.BUTTON1) {

if (selectCellString != null) {

table.setValueAt(selectCellString, selectRow, selectColumn);

}

selectColumn = table.columnAtPoint(e.getPoint());

selectRow = table.rowAtPoint(e.getPoint());

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

selectRowMarking = (String) table.getValueAt(selectRow, 0);

}

}

/\*\*

\* Обробка події натиснення ЛКМ на чарунку таблиці.

\* @param event

\*/

public void mousePressed(MouseEvent event) {

if (event.getClickCount() >= 2) {

isDoubleClick = true;

}

}

public void mouseReleased(MouseEvent e) {

}

public void mouseEntered(MouseEvent e) {

}

public void mouseExited(MouseEvent e) {

}

public void keyTyped(KeyEvent e) {

}

public void keyPressed(KeyEvent e) {

}

/\*\*

\* Обробка події натиснення клавіш на чарунку таблиці.

\* @param e

\*/

public void keyReleased(KeyEvent e) {

switch (e.getKeyCode()) {

case KeyEvent.VK\_ENTER:

// перевірка, чи був змінений текст в попередньому чарунку

String newStr = (String) table.getValueAt(selectRow, selectColumn);

boolean isError = false;

if (!newStr.equals(selectCellString)) {

boolean isChangeMarking = true;

if (selectColumn == 1) {

isChangeMarking = false;

}

if (textPanelType.equals(TextPanelType.OPERATION)) {

if (isChangeMarking) {

System.out.println("isChangeMarking");

if (newStr.equals("")) {

JOptionPane.showMessageDialog(null, ("Позначення сигналу не може бути порожнім! Введіть інше позначення"));

isError = true;

table.setValueAt(selectRowMarking, selectRow, selectColumn);

} else {

if (!checkOperationItems(newStr)) {

JOptionPane.showMessageDialog(null, ("Сигнал з позначенням '" + newStr + "' вже існує! Введіть інше позначення"));

isError = true;

table.setValueAt(selectRowMarking, selectRow, selectColumn);

} else {

lsa.changeOperationUnit(selectRowMarking, newStr, isChangeMarking);

parentPanel.updateLSAPanel();

}

}

} else {

lsa.changeOperationUnit(selectRowMarking, newStr, isChangeMarking);

}

} else {

if (isChangeMarking) {

if (newStr.equals("")) {

JOptionPane.showMessageDialog(null, ("Позначення умови не може бути порожнім! Введіть інше позначення"));

isError = true;

table.setValueAt(selectCellString, selectRow, selectColumn);

} else {

if (!checkConditionItems(newStr)) {

JOptionPane.showMessageDialog(null, ("Логічна умова з позначенням '" + newStr + "' вже існує! Введіть інше позначення"));

isError = true;

table.setValueAt(selectCellString, selectRow, selectColumn);

} else {

lsa.changeConditionUnit(selectRowMarking, newStr, isChangeMarking);

parentPanel.updateLSAPanel();

}

}

} else {

lsa.changeConditionUnit(selectRowMarking, newStr, isChangeMarking);

}

}

}

int oldSelectRow = selectRow;

int oldSelectColumn = selectColumn;

if (!isError) {

selectRow++;

if (selectRow == table.getRowCount()) {

selectRow = 0;

selectColumn++;

}

if (selectColumn == table.getColumnCount()) {

selectColumn = 0;

}

}

if (isDoubleClick) {

selectRow = oldSelectRow;

selectColumn = oldSelectColumn;

if (!isError) {

isDoubleClick = false;

}

}

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

selectRowMarking = (String) table.getValueAt(selectRow, 0);

break;

case KeyEvent.VK\_LEFT:

table.setValueAt(selectCellString, selectRow, selectColumn);

if (selectColumn != 0) {

selectColumn--;

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

}

break;

case KeyEvent.VK\_RIGHT:

table.setValueAt(selectCellString, selectRow, selectColumn);

if (selectColumn != table.getColumnCount() - 1) {

selectColumn++;

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

}

break;

case KeyEvent.VK\_UP:

table.setValueAt(selectCellString, selectRow, selectColumn);

if (selectRow != 0) {

selectRow--;

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

selectRowMarking = (String) table.getValueAt(selectRow, 0);

}

break;

case KeyEvent.VK\_DOWN:

table.setValueAt(selectCellString

, selectRow, selectColumn);

if (selectRow != table.getRowCount() - 1) {

selectRow++;

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

selectRowMarking = (String) table.getValueAt(selectRow, 0);

}

break;

case KeyEvent.VK\_TAB: //! зміна нумерації чарунків

table.setValueAt(selectCellString, selectRow, selectColumn);

if (selectColumn != table.getColumnCount() - 1) {

selectColumn++;

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

}

break;

}

}

/\*\*

\* Перевірка чи існує дане позначення операторної вершини.

\* @param newStr

\* @return

\*/

private boolean checkOperationItems(String newStr) {

boolean flag = true;

for (int i = 0; i < lsa.getOperationUnitsArray()[0].length; i++) {

if (newStr.equals(lsa.getOperationUnitsArray()[0][i])) {

flag = false;

break;

}

}

return flag;

}

/\*\*

\* Перевірка чи існує дане позначення умовної вершини.

\* @param newStr

\* @return

\*/

private boolean checkConditionItems(String newStr) {

boolean flag = true;

for (int i = 0; i < lsa.getConditionUnitsArray()[0].length; i++) {

if (newStr.equals(lsa.getConditionUnitsArray()[0][i])) {

flag = false;

break;

}

}

return flag;

}

}

package lab1.gui;

import lab1.LSA;

import javax.swing.\*;

import javax.swing.border.TitledBorder;

import java.awt.\*;

import java.awt.event.\*;

/\*\*

\* Типи таблиць відповідності позачень та сигналів.

\*/

enum TextPanelType {OPERATION, CONDITION};

public class TextPanel extends JPanel implements MouseListener, KeyListener {

private TextPanelType textPanelType;

private String[] columnNames = new String[2];

private String[][] data = new String[0][0];//{{"", ""}};

private LSA lsa;

private MainPanel parentPanel;

private TextPanelType itsTextPanelType;

private JTable table;

private JScrollPane scrollPane;

private int selectRow;

private int selectColumn;

private String selectCellString = null;

private String selectRowMarking = null;

private boolean isDoubleClick = false;

/\*\*

\* Створення таблиць відповідності позначень та сигналів.

\* @param textPanelType

\* @param lsa

\* @param pPanel

\*/

public TextPanel(TextPanelType textPanelType, LSA lsa, MainPanel pPanel) {

this.textPanelType = textPanelType;

this.itsTextPanelType = textPanelType;

this.lsa = lsa;

this.parentPanel = pPanel;

if (textPanelType.equals(TextPanelType.OPERATION)) {

if (lsa.getOperationUnitsArray()[0].length != 0) {

this.data = lsa.getOperationUnitsArray();

}

}

if (textPanelType.equals(TextPanelType.CONDITION)) {

if (lsa.getOperationUnitsArray()[0].length != 0) {

this.data = lsa.getConditionUnitsArray();

}

}

//--------------------

setLayout(new CardLayout());

if (textPanelType.equals(TextPanelType.OPERATION)) {

columnNames[0] = "Позначення";

columnNames[1] = "Сигнал";

setBorder(new TitledBorder("Оператори"));

} else {

columnNames[0] = "Позначення";

columnNames[1] = "Умова";

setBorder(new TitledBorder("Логічні умови"));

}

table = new JTable(data, columnNames);

table.addMouseListener(this);

table.addKeyListener(this);

scrollPane = new JScrollPane(table);

add(scrollPane);

}

/\*\*

\* Оновлення інформації в таблиці.

\* @param nlsa

\*/

public void updateTable(LSA nlsa) {

this.lsa = nlsa;

if (this.itsTextPanelType.equals(TextPanelType.OPERATION)) {

this.data = convertArray(lsa.getOperationUnitsArray());

} else {

this.data = convertArray(lsa.getConditionUnitsArray());

}

//

remove(scrollPane);

scrollPane.remove(table);

table = new JTable(data, columnNames);

table.addMouseListener(this);

table.addKeyListener(this);

scrollPane = new JScrollPane(table);

add(scrollPane);

repaint();

revalidate();

}

/\*\*

\* Транспонування матриці сигналів.

\* @param strings

\* @return

\*/

private String[][] convertArray(String[][] strings) {

String[][] tempStr = new String[strings[0].length][strings.length];

for (int i = 0; i < strings[0].length; i++) {

for (int j = 0; j < strings.length; j++) {

tempStr[i][j] = strings[j][i];

}

}

return tempStr;

}

/\*\*

\* Обробка події натиснення ЛКМ на чарунку таблиці.

\* @param e

\*/

public void mouseClicked(MouseEvent e) {

if (e.getButton() == MouseEvent.BUTTON1) {

if (selectCellString != null) {

table.setValueAt(selectCellString, selectRow, selectColumn);

}

selectColumn = table.columnAtPoint(e.getPoint());

selectRow = table.rowAtPoint(e.getPoint());

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

selectRowMarking = (String) table.getValueAt(selectRow, 0);

}

}

/\*\*

\* Обробка події натиснення ЛКМ на чарунку таблиці.

\* @param event

\*/

public void mousePressed(MouseEvent event) {

if (event.getClickCount() >= 2) {

isDoubleClick = true;

}

}

public void mouseReleased(MouseEvent e) {

}

public void mouseEntered(MouseEvent e) {

}

public void mouseExited(MouseEvent e) {

}

public void keyTyped(KeyEvent e) {

}

public void keyPressed(KeyEvent e) {

}

/\*\*

\* Обробка події натиснення клавіш на чарунку таблиці.

\* @param e

\*/

public void keyReleased(KeyEvent e) {

switch (e.getKeyCode()) {

case KeyEvent.VK\_ENTER:

// перевірка, чи був змінений текст в попередньому чарунку

String newStr = (String) table.getValueAt(selectRow, selectColumn);

boolean isError = false;

if (!newStr.equals(selectCellString)) {

boolean isChangeMarking = true;

if (selectColumn == 1) {

isChangeMarking = false;

}

if (textPanelType.equals(TextPanelType.OPERATION)) {

if (isChangeMarking) {

System.out.println("isChangeMarking");

if (newStr.equals("")) {

JOptionPane.showMessageDialog(null, ("Позначення сигналу не може бути порожнім! Введіть інше позначення"));

isError = true;

table.setValueAt(selectRowMarking, selectRow, selectColumn);

} else {

if (!checkOperationItems(newStr)) {

JOptionPane.showMessageDialog(null, ("Сигнал з позначенням '" + newStr + "' вже існує! Введіть інше позначення"));

isError = true;

table.setValueAt(selectRowMarking, selectRow, selectColumn);

} else {

lsa.changeOperationUnit(selectRowMarking, newStr, isChangeMarking);

parentPanel.updateLSAPanel();

}

}

} else {

lsa.changeOperationUnit(selectRowMarking, newStr, isChangeMarking);

}

} else {

if (isChangeMarking) {

if (newStr.equals("")) {

JOptionPane.showMessageDialog(null, ("Позначення умови не може бути порожнім! Введіть інше позначення"));

isError = true;

table.setValueAt(selectCellString, selectRow, selectColumn);

} else {

if (!checkConditionItems(newStr)) {

JOptionPane.showMessageDialog(null, ("Логічна умова з позначенням '" + newStr + "' вже існує! Введіть інше позначення"));

isError = true;

table.setValueAt(selectCellString, selectRow, selectColumn);

} else {

lsa.changeConditionUnit(selectRowMarking, newStr, isChangeMarking);

parentPanel.updateLSAPanel();

}

}

} else {

lsa.changeConditionUnit(selectRowMarking, newStr, isChangeMarking);

}

}

}

int oldSelectRow = selectRow;

int oldSelectColumn = selectColumn;

if (!isError) {

selectRow++;

if (selectRow == table.getRowCount()) {

selectRow = 0;

selectColumn++;

}

if (selectColumn == table.getColumnCount()) {

selectColumn = 0;

}

}

if (isDoubleClick) {

selectRow = oldSelectRow;

selectColumn = oldSelectColumn;

if (!isError) {

isDoubleClick = false;

}

}

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

selectRowMarking = (String) table.getValueAt(selectRow, 0);

break;

case KeyEvent.VK\_LEFT:

table.setValueAt(selectCellString, selectRow, selectColumn);

if (selectColumn != 0) {

selectColumn--;

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

}

break;

case KeyEvent.VK\_RIGHT:

table.setValueAt(selectCellString, selectRow, selectColumn);

if (selectColumn != table.getColumnCount() - 1) {

selectColumn++;

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

}

break;

case KeyEvent.VK\_UP:

table.setValueAt(selectCellString, selectRow, selectColumn);

if (selectRow != 0) {

selectRow--;

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

selectRowMarking = (String) table.getValueAt(selectRow, 0);

}

break;

case KeyEvent.VK\_DOWN:

table.setValueAt(selectCellString

, selectRow, selectColumn);

if (selectRow != table.getRowCount() - 1) {

selectRow++;

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

selectRowMarking = (String) table.getValueAt(selectRow, 0);

}

break;

case KeyEvent.VK\_TAB: //! зміна нумерації чарунків

table.setValueAt(selectCellString, selectRow, selectColumn);

if (selectColumn != table.getColumnCount() - 1) {

selectColumn++;

selectCellString = (String) table.getValueAt(selectRow, selectColumn);

}

break;

}

}

/\*\*

\* Перевірка чи існує дане позначення операторної вершини.

\* @param newStr

\* @return

\*/

private boolean checkOperationItems(String newStr) {

boolean flag = true;

for (int i = 0; i < lsa.getOperationUnitsArray()[0].length; i++) {

if (newStr.equals(lsa.getOperationUnitsArray()[0][i])) {

flag = false;

break;

}

}

return flag;

}

/\*\*

\* Перевірка чи існує дане позначення умовної вершини.

\* @param newStr

\* @return

\*/

private boolean checkConditionItems(String newStr) {

boolean flag = true;

for (int i = 0; i < lsa.getConditionUnitsArray()[0].length; i++) {

if (newStr.equals(lsa.getConditionUnitsArray()[0][i])) {

flag = false;

break;

}

}

return flag;

}

}

package lab1.file;

import lab1.LSA;

import lab1.MSA;

import lab1.Unit;

import lab1.UnitType;

import lab1.gui.MainPanel;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.util.ArrayList;

import java.util.Map;

public class NewProjectListener implements ActionListener {

private MainPanel mainPanel;

private ArrayList<Unit> unitArrayList;

/\*\*

\* Конструктор обробника кнопки створення нового проекту.

\*

\* @param mainPanel

\*/

public NewProjectListener(MainPanel mainPanel) {

this.mainPanel = mainPanel;

}

/\*\*

\* Обробник кнопки створення нового проекту.

\*/

public void actionPerformed(ActionEvent e) {

unitArrayList = new ArrayList<Unit>();

Unit beginUnit = new Unit(UnitType.B, new String[0], new String[0], -1);

unitArrayList.add(beginUnit);

LSA lsa = new LSA(unitArrayList);

mainPanel.setThisLsa(lsa);

mainPanel.updateTables();

mainPanel.updateLSAPanel();

}

}

package lab1.file;

import lab1.LSA;

import lab1.MSA;

import lab1.Unit;

import lab1.UnitType;

import lab1.gui.MainFrame;

import lab1.gui.MainPanel;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.ObjectInputStream;

import java.util.ArrayList;

import java.util.Map;

public class OpenButtonListener implements ActionListener {

private MainPanel mainPanel;

private ArrayList<Unit> unitArrayList;

private ArrayList<Integer> freeIndexed;

/\*\*

\* Конструктор обробника кнопки відкриття файлу.

\*

\* @param mainPanel

\*/

public OpenButtonListener(MainPanel mainPanel) {

this.mainPanel = mainPanel;

}

/\*\*

\* Обробник кнопки відкриття файлу.

\*

\* @param e

\*/

public void actionPerformed(ActionEvent e) {

JFileChooser chooser = new JFileChooser();

chooser.setCurrentDirectory(new File("."));

chooser.setSelectedFile(new File(".lsa"));

int result = chooser.showOpenDialog(null);

if (result != JFileChooser.CANCEL\_OPTION) {

String fileName = chooser.getSelectedFile().getPath();

try {

deserialize(fileName);

} catch (IOException e1) {

JOptionPane.showMessageDialog(null, ("Не вдалося відкрити файл " + fileName));

} catch (ClassNotFoundException e1) {

e1.printStackTrace(); //To change body of catch statement use File | Settings | File Templates.

}

}

}

/\*\*

\* Десереалізує обєкт класу MSA з файлу.

\*

\* @param fileName

\* @throws IOException

\* @throws ClassNotFoundException

\*/

private void deserialize(String fileName) throws IOException, ClassNotFoundException {

FileInputStream fis = new FileInputStream(fileName);

ObjectInputStream oin = new ObjectInputStream(fis);

MSA msa = (MSA) oin.readObject();

System.out.println(mainPanel);

mainPanel.setThisLsa(createLSAfromFile(msa));

mainPanel.updateTables();

mainPanel.updateLSAPanel();

}

/\*\*

\* Створює масив елементів ЛСА з матриці МСА.

\*

\* @param msa

\* @return

\*/

private LSA createLSAfromFile(MSA msa) {

unitArrayList = new ArrayList<Unit>();

freeIndexed = new ArrayList<Integer>();

Unit beginUnit = new Unit(UnitType.B, new String[0], new String[0], -1);

unitArrayList.add(beginUnit);

int[][] unitsUnion = msa.getUnitsUnion();

Map<Integer, Unit> unitSignals = msa.getUnitSignals();

for (int i = 0; i < unitsUnion.length; i++) {

unitArrayList.add(unitSignals.get(i));

}

for (int i = 0; i < unitsUnion.length; i++) {

Unit thisUnit = unitSignals.get(i);

UnitType thisUnitType = thisUnit.getUnitType();

switch (thisUnitType) {

case O:

pasteOperationElement(i, unitsUnion, unitSignals, thisUnit);

break;

case C:

pasteConditionElement(i, unitsUnion, unitSignals, thisUnit);

}

}

LSA lsa = new LSA(unitArrayList);

return lsa;

}

/\*\*

\* Повретає позицію елемента з масиву елементів ЛСА.

\*

\* @param unit

\* @return

\*/

private int getPosOfUnit(Unit unit) {

for (int i = 0; i < unitArrayList.size(); i++) {

if (unitArrayList.get(i).equals(unit)) {

return i + 1; // +1 - так як є початковий елемент ЛСА

}

}

return -1;

}

/\*\*

\* Повертає позицію, куди йде перехід оперційної вершини в МСА.

\*

\* @param unitsUnion

\* @param line

\* @return

\*/

private int whereGoInLine(int[][] unitsUnion, int line) {

for (int i = 0; i < unitsUnion.length; i++) {

if (unitsUnion[line][i] == 1) {

return i;

}

}

return -1;

}

/\*\*

\* Повертає позицію, куди йде 2й перехід логічної вершини в МСА.

\*

\* @param unitsUnion

\* @param line

\* @return

\*/

private int whereGoSecondInLine(int[][] unitsUnion, int line) {

for (int i = 0; i < unitsUnion.length; i++) {

if (unitsUnion[line][i] == -1) {

return i;

}

}

return -1;

}

/\*\*

\* Створює новий індекс для безумовного переходу. При цьому індекс не дублює раніше створні індекси або індекси умовних переходів.

\*/

private int getFreeIndex() {

boolean isFree = false;

int freeIndex = 0;

while (!isFree) {

isFree = true;

for (int i = 1; i < unitArrayList.size(); i++) {

if (unitArrayList.get(i).getIndex() == freeIndex) {

isFree = false;

freeIndex++;

break;

}

}

if (isFree) {

for (Integer integer : freeIndexed) {

if (integer == freeIndex) {

isFree = false;

freeIndex++;

break;

}

}

}

}

freeIndexed.add(freeIndex);

return freeIndex;

}

private void pasteOperationElement(int i, int[][] unitsUnion, Map<Integer, Unit> unitSignals, Unit thisUnit) {

int nextElemPos = whereGoInLine(unitsUnion, i);

if (nextElemPos != -1) {

if (nextElemPos != (i + 1)) { //перехід не на наступний елемент - вставляємо безумовний перехід і елемент повернення

int freeIndex = getFreeIndex();

unitArrayList.add(getPosOfUnit(thisUnit), new Unit(UnitType.U, new String[0], new String[0], freeIndex));

unitArrayList.add(getPosOfUnit(unitSignals.get(nextElemPos)) - 1, new Unit(UnitType.R, new String[0], new String[0], freeIndex));

}

} else { //переходу немає, або кінцева вершина

if (i != unitsUnion.length - 1) { // переходу немає - вставляємо безумовний перехід в нікуди

unitArrayList.add(new Unit(UnitType.U, new String[0], new String[0], -1));

} else { // вставляємо кінцеву вершину

unitArrayList.add(new Unit(UnitType.E, new String[0], new String[0], -1));

}

}

}

private void pasteConditionElement(int i, int[][] unitsUnion, Map<Integer, Unit> unitSignals, Unit thisUnit) {

pasteOperationElement(i, unitsUnion, unitSignals, thisUnit);

int secondElemPos = whereGoSecondInLine(unitsUnion, i);

if (secondElemPos != -1) {

unitArrayList.add(getPosOfUnit(unitSignals.get(secondElemPos)) - 1, new Unit(UnitType.R, new String[0], new String[0], thisUnit.getIndex()));

}

}

}

package lab1.file;

import lab1.LSA;

import lab1.gui.MainPanel;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectOutputStream;

public class SaveButtonListener implements ActionListener {

private MainPanel mainPanel;

public SaveButtonListener(MainPanel mainPanel) {

this.mainPanel = mainPanel;

}

/\*\*

\* Обробник кнопки збереження МСА в файл.

\*

\* @param e

\*/

public void actionPerformed(ActionEvent e) {

JFileChooser chooser = new JFileChooser();

chooser.setCurrentDirectory(new File("."));

chooser.setSelectedFile(new File(".lsa"));

int result = chooser.showSaveDialog(null);

if (result != JFileChooser.CANCEL\_OPTION) {

String fileName = chooser.getSelectedFile().getPath();

try {

serialize(fileName);

} catch (IOException e1) {

e1.printStackTrace();

}

}

}

/\*\*

\* Серіалізує матрицю МСА в файл.

\*

\* @param fileName

\* @throws IOException

\*/

private void serialize(String fileName) throws IOException {

FileOutputStream fos = new FileOutputStream(fileName);

ObjectOutputStream oos = new ObjectOutputStream(fos);

LSA lsa = mainPanel.getThisLsa();

oos.writeObject(lsa.getMsa());

oos.flush();

oos.close();

}

}