public class TimeDiagramPanel extends JPanel implements Scrollable {

private ModellingModel modellingModel;

public TimeDiagramPanel(ModellingModel modellingModel) {

this.modellingModel = modellingModel;

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

Graphics2D g2 = (Graphics2D) g;

setBackground(modellingModel.getBackgroundColor());

FontRenderContext context = g2.getFontRenderContext();

g2.setFont(modellingModel.getFont());

ArrayList<String> names = modellingModel.getNames();

int maxWidth = 0;

int maxHeight = 0;

Rectangle2D bounds;

for (String name : names) {

bounds = getFont().getStringBounds(name, context);

if (bounds.getWidth() > maxWidth) {

maxWidth = (int) bounds.getWidth();

}

if (bounds.getHeight() > maxHeight) {

maxHeight = (int) bounds.getHeight();

}

}

bounds = getFont().getStringBounds(ModellingModel.TIME\_STRING, context);

if (bounds.getWidth() > maxWidth) {

maxWidth = (int) bounds.getWidth();

}

modellingModel.setDiagramSize(new Dimension(maxWidth + ModellingModel.INTERVAL\_WIDTH \* modellingModel.getT() + ModellingModel.DIAGRAM\_BORDER \* 3,

maxHeight \* (names.size() + 1) + ModellingModel.DIAGRAM\_Y\_INTERVAL \* names.size() + ModellingModel.DIAGRAM\_BORDER \* 2));

setSize(modellingModel.getDiagramSize());

g2.setColor(modellingModel.getTextColor());

int y = ModellingModel.DIAGRAM\_BORDER;

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

LineMetrics metrics = modellingModel.getFont().getLineMetrics(names.get(i), context);

g2.drawString(names.get(i), ModellingModel.DIAGRAM\_BORDER, (int) (y + metrics.getAscent()));

y += maxHeight + ModellingModel.DIAGRAM\_Y\_INTERVAL;

}

if (modellingModel.getT() > 0) {

ArrayList<HashMap<Integer, Character>> values = modellingModel.getDiagrams();

y = ModellingModel.DIAGRAM\_BORDER;

g2.setColor(modellingModel.getStandartColor());

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

int x = 2 \* ModellingModel.DIAGRAM\_BORDER + maxWidth;

for (int t = 0; t < modellingModel.getT(); t++) {

if (values.get(i).get(t).compareTo('1') == 0) {

g2.fillRect(x, y, ModellingModel.INTERVAL\_WIDTH, maxHeight);

} else {

g2.drawLine(x, y + maxHeight, x + ModellingModel.INTERVAL\_WIDTH, y + maxHeight);

}

x += ModellingModel.INTERVAL\_WIDTH;

}

y += maxHeight + ModellingModel.DIAGRAM\_Y\_INTERVAL;

}

}

g2.setColor(modellingModel.getTextColor());

int x = ModellingModel.DIAGRAM\_BORDER;

LineMetrics metrics = modellingModel.getFont().getLineMetrics(ModellingModel.TIME\_STRING, context);

g2.drawString(ModellingModel.TIME\_STRING, x, y + metrics.getAscent());

x += ModellingModel.DIAGRAM\_BORDER + maxWidth;

for (int t = 0; t < modellingModel.getT(); t++) {

String tString = String.valueOf(t);

bounds = modellingModel.getFont().getStringBounds(tString, context);

metrics = modellingModel.getFont().getLineMetrics(tString, context);

g2.drawString(tString, (int) (x - bounds.getWidth() / 2), (int) (y + metrics.getAscent()));

x += ModellingModel.INTERVAL\_WIDTH;

}

}

@Override

public Dimension getPreferredSize() {

return modellingModel.getDiagramSize();

}

public Dimension getPreferredScrollableViewportSize() {

return getPreferredSize();

}

public int getScrollableUnitIncrement(Rectangle visibleRect, int orientation, int direction) {

return 1;

}

public int getScrollableBlockIncrement(Rectangle visibleRect, int orientation, int direction) {

return 10;

}

public boolean getScrollableTracksViewportWidth() {

return false;

}

public boolean getScrollableTracksViewportHeight() {

return false;

}

}

public class SchemeTableModel extends AbstractTableModel {

private String[] inNames;

private String[] elementNames;

private LogicFunctionsWorker.LogicFunction[] logicFunctions;

private int[] inertiaDelay;

private int[] dynamicDelay;

private String[] outNames;

private int[][] connectivityMatrix;

public SchemeTableModel(String[] inNames, String[] elementNames, LogicFunctionsWorker.LogicFunction[] logicFunctions,

int[] inertiaDelay, int[] dynamicDelay, String[] outNames, int[][] connectivityMatrix) {

this.inNames = inNames;

this.elementNames = elementNames;

this.logicFunctions = logicFunctions;

this.inertiaDelay = inertiaDelay;

this.dynamicDelay = dynamicDelay;

this.outNames = outNames;

this.connectivityMatrix = connectivityMatrix;

}

public int getInCount() {

return inNames.length;

}

public String[] getInNames() {

return inNames;

}

public String[] getElementNames() {

return elementNames;

}

public LogicFunctionsWorker.LogicFunction[] getLogicFunctions() {

return logicFunctions;

}

public int[] getInertiaDelay() {

return inertiaDelay;

}

public int[] getDynamicDelay() {

return dynamicDelay;

}

public int[][] getConnectivityMatrixExceptOut() {

int[][] result = new int[inNames.length + elementNames.length][];

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

result[i] = new int[result.length];

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

result[i][j] = connectivityMatrix[i][j];

}

}

return result;

}

@Override

public boolean isCellEditable(int rowIndex, int columnIndex) {

return false;

}

public int getRowCount() {

return connectivityMatrix.length + 1;

}

public int getColumnCount() {

return (inNames.length + elementNames.length + outNames.length + 1);

}

public Object getValueAt(int rowIndex, int columnIndex) {

if (rowIndex == 0 && columnIndex == 0) {

return "";

}

if (rowIndex == 0) {

int temp = columnIndex - 1;

if (temp >= inNames.length) {

temp -= inNames.length;

if (temp < elementNames.length) {

return elementNames[temp];

} else {

temp -= elementNames.length;

return outNames[temp];

}

} else {

return inNames[temp];

}

}

if (columnIndex == 0) {

int temp = rowIndex - 1;

if (temp >= inNames.length) {

temp -= inNames.length;

if (temp < elementNames.length) {

return elementNames[temp];

} else {

temp -= elementNames.length;

return outNames[temp];

}

} else {

return inNames[temp];

}

}

return connectivityMatrix[rowIndex - 1][columnIndex - 1];

}

public int getElementCount() {

return elementNames.length;

}

}

public class ModellingPanel extends JPanel {

private JLabel inputSetLabel;

private JComboBox inputSetBox;

private JLabel startSetLabel;

private JComboBox startSetBox;

private JButton stepButton;

private JButton modellingButton;

private JButton allModellingButton;

private JButton clearButton;

private JSplitPane splitPane;

private JTable modellingTable;

private TimeDiagramPanel diagramPanel;

private ModellingModel modellingModel;

public ModellingPanel(SchemeTableModel schemeTableModel, Color backgroundColor, Color standartColor, Color textColor,

Font font) {

super();

modellingModel = new ModellingModel(schemeTableModel.getInNames(), schemeTableModel.getElementNames(),

schemeTableModel.getInertiaDelay(), schemeTableModel.getDynamicDelay(),

schemeTableModel.getLogicFunctions(), schemeTableModel.getConnectivityMatrixExceptOut(),

backgroundColor, standartColor, textColor, font);

StringBuilder builder = new StringBuilder();

builder.append("Вхідний набір( ");

for (String s : schemeTableModel.getInNames()) {

builder.append(s);

builder.append(" ");

}

builder.append(")");

inputSetLabel = new JLabel(builder.toString());

inputSetBox = new JComboBox();

inputSetBox.setEditable(false);

for (int i = 0; i < Math.pow(2, schemeTableModel.getInCount()); i++) {

String temp = Integer.toBinaryString(i);

while (temp.length() < schemeTableModel.getInCount()) {

temp = "0" + temp;

}

inputSetBox.addItem(temp);

}

builder = new StringBuilder();

builder.append("Установчий набір ( ");

for (String s : schemeTableModel.getInNames()) {

builder.append(s);

builder.append(" ");

}

builder.append(")");

startSetLabel = new JLabel(builder.toString());

startSetBox = new JComboBox();

startSetBox.setEditable(false);

for (int i = 0; i < Math.pow(2, schemeTableModel.getInCount()); i++) {

String temp = Integer.toBinaryString(i);

while (temp.length() < schemeTableModel.getInCount()) {

temp = "0" + temp;

}

startSetBox.addItem(temp);

}

stepButton = new JButton(new AbstractAction() {

public void actionPerformed(ActionEvent e) {

if (modellingModel.isBusy()) {

modellingModel.step();

} else {

if (modellingModel.isClear()) {

modellingModel.step((String) inputSetBox.getSelectedItem(), (String) startSetBox.getSelectedItem());

} else {

modellingModel.step((String) inputSetBox.getSelectedItem(), true);

}

}

modellingModel.fireTableDataChanged();

revalidate();

repaint();

}

});

stepButton.setText("Крок");

modellingButton = new JButton(new AbstractAction() {

public void actionPerformed(ActionEvent e) {

modellingModel.modelling((String) inputSetBox.getSelectedItem(), (String) startSetBox.getSelectedItem());

modellingModel.fireTableDataChanged();

revalidate();

repaint();

}

});

modellingButton.setText("Моделювати");

allModellingButton = new JButton(new AbstractAction() {

public void actionPerformed(ActionEvent e) {

for (int i = 0; i < inputSetBox.getItemCount(); i++) {

modellingModel.modelling((String) inputSetBox.getItemAt(i), (String) startSetBox.getSelectedItem());

}

modellingModel.fireTableDataChanged();

revalidate();

repaint();

}

});

allModellingButton.setText("Моделювати всі набори");

clearButton = new JButton(new AbstractAction() {

public void actionPerformed(ActionEvent e) {

modellingModel.clear();

modellingModel.fireTableDataChanged();

revalidate();

repaint();

}

});

clearButton.setText("Очистити");

modellingTable = new JTable(modellingModel);

modellingTable.setDragEnabled(false);

diagramPanel = new TimeDiagramPanel(modellingModel);

splitPane = new JSplitPane(JSplitPane.VERTICAL\_SPLIT);

splitPane.setTopComponent(new JScrollPane(modellingTable));

splitPane.setBottomComponent(new JScrollPane(diagramPanel));

splitPane.setDividerLocation(splitPane.getPreferredSize().height / 2);

int strutSize = 5;

Box hBox = Box.createHorizontalBox();

hBox.add(Box.createHorizontalStrut(strutSize));

hBox.add(inputSetLabel);

hBox.add(Box.createHorizontalStrut(strutSize));

hBox.add(inputSetBox);

hBox.add(Box.createHorizontalStrut(strutSize));

hBox.add(startSetLabel);

hBox.add(Box.createHorizontalStrut(strutSize));

hBox.add(startSetBox);

hBox.add(Box.createHorizontalStrut(strutSize));

hBox.add(stepButton);

hBox.add(Box.createHorizontalStrut(strutSize));

hBox.add(modellingButton);

hBox.add(Box.createHorizontalStrut(strutSize));

hBox.add(allModellingButton);

hBox.add(Box.createHorizontalStrut(strutSize));

hBox.add(clearButton);

hBox.add(Box.createHorizontalStrut(strutSize));

Box vBox = Box.createVerticalBox();

vBox.add(Box.createVerticalStrut(2 \* strutSize));

vBox.add(hBox);

vBox.add(Box.createVerticalStrut(2 \* strutSize));

setLayout(new BorderLayout());

add(new JScrollPane(vBox), BorderLayout.NORTH);

add(splitPane);

}

}

public class ModellingModel extends AbstractTableModel {

public static final int INTERVAL\_WIDTH = 20;

public static final int DIAGRAM\_BORDER = 20;

public static final int DIAGRAM\_Y\_INTERVAL = 15;

public static final String TIME\_STRING = "Час t";

private static final String[] LAST\_COLUMN\_NAMES = {"ТТС", "t", "ТБС"};

private Color backgroundColor;

private Color standartColor;

private Color textColor;

private Font font;

private String[] inNames;

private String[] elementNames;

private int[] inertiaDelay;

private int[] dynamicDelay;

private LogicFunctionsWorker.LogicFunction[] logicFunctions;

private int[][] connectivityMatrix;

private int t;

private HashMap<Integer, LinkedList<Integer>> futureEventsTable;

private int lastEventT;

private LinkedList<String> previousFutureEvents;

private Dimension diagramSize;

private int graphicHeight;

private ArrayList<HashMap<Integer, Character>> diagrams;

private LinkedList<String[]> rows;

private int columnCount;

private boolean isBusy;

private boolean isClear;

public ModellingModel(String[] inNames, String[] elementNames, int[] inertiaDelay, int[] dynamicDelay,

LogicFunctionsWorker.LogicFunction[] logicFunctions, int[][] connectivityMatrix,

Color backgroundColor, Color standartColor, Color textColor, Font font) {

this.inNames = inNames;

this.elementNames = elementNames;

this.inertiaDelay = inertiaDelay;

this.dynamicDelay = dynamicDelay;

this.logicFunctions = logicFunctions;

this.connectivityMatrix = connectivityMatrix;

this.backgroundColor = backgroundColor;

this.standartColor = standartColor;

this.textColor = textColor;

this.font = font;

columnCount = inNames.length + elementNames.length + 3;

rows = new LinkedList<String[]>();

isBusy = false;

isClear = true;

diagramSize = new Dimension(100, 100);

t = 0;

lastEventT = 0;

futureEventsTable = new HashMap<Integer, LinkedList<Integer>>();

previousFutureEvents = new LinkedList<String>();

diagrams = new ArrayList<HashMap<Integer, Character>>();

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

diagrams.add(new HashMap<Integer, Character>());

}

}

private boolean hasFutureEvents() {

if (futureEventsTable.isEmpty()) {

return false;

}

return true;

}

private void addDiagramValues(int t) {

int index = 0;

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

int j = rows.size() - 1;

boolean found = false;

while (!found && j >= 0) {

if (rows.get(j)[index].length() > 0 && rows.get(j)[index].compareTo(" ") != 0) {

found = true;

}

j--;

}

diagrams.get(index).put(t, rows.get(++j)[index].charAt(0));

index++;

}

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

int j = rows.size() - 1;

boolean found = false;

while (!found && j >= 0) {

if (rows.get(j)[index].length() > 0) {

found = true;

}

j--;

}

diagrams.get(index).put(t, rows.get(++j)[index].charAt(0));

index++;

}

}

private String getInputSet(ArrayList<Integer> numbers) {

StringBuffer buffer = new StringBuffer();

for (int index : numbers) {

int j = rows.size() - 1;

boolean found = false;

while (!found && j >= 0) {

if (rows.get(j)[index].length() > 0 && rows.get(j)[index].compareTo(" ") != 0) {

found = true;

}

j--;

}

buffer.append(rows.get(++j)[index]);

}

return buffer.toString();

}

private String[] getStartCondition(String startSet) {

ArrayList<String[]> conditions = new ArrayList<>();

String[] condition = new String[elementNames.length];

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

condition[i] = "0";

}

conditions.add(condition);

boolean endFlag;

do {

condition = new String[elementNames.length];

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

ArrayList<Integer> inputElements = new ArrayList<Integer>();

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

if (connectivityMatrix[j][i + inNames.length] > 0) {

for (int k = connectivityMatrix[j][i + inNames.length]; k > 0; k--) {

inputElements.add(j);

}

}

}

StringBuffer inputBuffer = new StringBuffer();

for (int index : inputElements) {

if (index < inNames.length) {

inputBuffer.append(startSet.substring(index, index + 1));

} else {

inputBuffer.append(conditions.get(conditions.size() - 1)[index - inNames.length]);

}

}

StringBuffer buffer = new StringBuffer();

buffer.append(LogicFunctionsWorker.getValue(logicFunctions[i], inputBuffer.toString()));

condition[i] = buffer.toString();

}

String[] previousCondition = conditions.get(conditions.size() - 1);

endFlag = true;

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

if (condition[i].compareTo(previousCondition[i]) != 0) {

endFlag = false;

break;

}

}

if (!endFlag) {

conditions.add(condition);

}

} while (!endFlag);

return conditions.get(conditions.size() - 1);

}

public boolean isBusy() {

return isBusy;

}

public boolean isClear() {

return isClear;

}

public void step(String inputSet, String startSet) {

String[] row = new String[columnCount];

int index = 0;

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

row[index++] = "";

}

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

row[index++] = "0";

}

for (int i = 0; i < LAST\_COLUMN\_NAMES.length; i++) {

row[index++] = "";

}

rows.add(row);

row = new String[columnCount];

index = 0;

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

row[index++] = startSet.substring(i, i + 1);

}

String[] startCondition = getStartCondition(startSet);

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

row[index++] = startCondition[i];

}

row[index++] = "";

row[index++] = "";

StringBuilder stringBuilder = new StringBuilder();

boolean isFirst = true;

for (int i = inNames.length; i < inNames.length + elementNames.length; i++) {

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

if (connectivityMatrix[j][i] > 0) {

if (!isFirst) {

stringBuilder.append("-");

}

stringBuilder.append(elementNames[i - inNames.length]);

if (futureEventsTable.get(0) == null) {

futureEventsTable.put(0, new LinkedList<Integer>());

}

futureEventsTable.get(0).add(i);

isFirst = false;

break;

}

}

}

row[index] = stringBuilder.toString();

previousFutureEvents.add(stringBuilder.toString());

rows.add(row);

isClear = false;

step(inputSet, false);

}

public void step() {

StringBuffer buffer = new StringBuffer();

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

buffer.append(" ");

}

step(buffer.toString(), false);

}

public void step(String inputSet, boolean newSet) {

if (newSet) {

for (int i = inNames.length; i < inNames.length + elementNames.length; i++) {

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

if (connectivityMatrix[j][i] > 0) {

if (futureEventsTable.get(t) == null) {

futureEventsTable.put(t, new LinkedList<Integer>());

}

futureEventsTable.get(t).add(i);

break;

}

}

}

}

String[] row = new String[columnCount];

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

row[i] = inputSet.substring(i, i + 1);

}

for (int i = inNames.length; i < inNames.length + elementNames.length; i++) {

row[i] = "";

}

LinkedList<Integer> currentEvents = futureEventsTable.get(t);

futureEventsTable.remove(t);

if (currentEvents != null) {

StringBuilder stringBuilder = new StringBuilder();

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

stringBuilder.append(elementNames[currentEvents.get(i) - inNames.length]);

if (i < currentEvents.size() - 1) {

stringBuilder.append("-");

}

}

row[columnCount - 3] = stringBuilder.toString();

String tempString = stringBuilder.toString();

row[columnCount - 2] = String.valueOf(t);

rows.add(row);

stringBuilder = new StringBuilder();

boolean isFirst = true;

for (int ii = 0; ii < currentEvents.size(); ii++) {

int index = currentEvents.get(ii);

//TODO: I changed delay calculation

// int delay = dynamicDelay[index - inNames.length];

int delay = inertiaDelay[index - inNames.length] + dynamicDelay[index - inNames.length];

ArrayList<Integer> inputElements = new ArrayList<Integer>();

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

if (connectivityMatrix[i][index] > 0) {

for (int j = connectivityMatrix[i][index]; j > 0; j--) {

inputElements.add(i);

}

}

}

StringBuffer buffer = new StringBuffer();

buffer.append(LogicFunctionsWorker.getValue(logicFunctions[index - inNames.length], getInputSet(inputElements)));

row[index] = buffer.toString();

for (int i = inNames.length; i < connectivityMatrix[index].length; i++) {

if (connectivityMatrix[index][i] > 0) {

if (!isFirst) {

stringBuilder.append("-");

}

stringBuilder.append(elementNames[i - inNames.length]);

isFirst = false;

boolean addFlag = true;

for (int eventT = t; eventT <= lastEventT; eventT++) {

if (futureEventsTable.get(eventT) != null) {

if (futureEventsTable.get(eventT).contains(i)) {

if (eventT < t + delay) {

futureEventsTable.get(eventT).remove(new Integer(i));

} else {

addFlag = false;

}

}

}

}

if (addFlag) {

if (delay == 0) {

currentEvents.add(i);

} else {

if (futureEventsTable.get(t + delay) == null) {

futureEventsTable.put(t + delay, new LinkedList<Integer>());

}

if (!futureEventsTable.get(t + delay).contains(i)) {

futureEventsTable.get(t + delay).add(i);

if (t + delay > lastEventT) {

lastEventT = t + delay;

}

}

}

}

}

}

}

row[columnCount - 1] = stringBuilder.toString();

isBusy = hasFutureEvents();

previousFutureEvents.add(tempString);

for (String e : previousFutureEvents) {

if (e.compareTo(stringBuilder.toString()) == 0 && stringBuilder.toString().length() > 0) {

isBusy = false;

break;

}

}

if (!isBusy) {

previousFutureEvents = new LinkedList<String>();

lastEventT = 0;

} else {

previousFutureEvents.add(stringBuilder.toString());

}

} else {

row[inNames.length + elementNames.length + 1] = String.valueOf(t);

rows.add(row);

}

addDiagramValues(t);

t++;

}

public void modelling(String inputSet, String startSet) {

if (!isBusy) {

if (isClear) {

step(inputSet, startSet);

} else {

step(inputSet, true);

}

}

while (isBusy) {

step();

}

}

public void clear() {

t = 0;

lastEventT = 0;

futureEventsTable = new HashMap<Integer, LinkedList<Integer>>();

previousFutureEvents = new LinkedList<String>();

isBusy = false;

isClear = true;

rows = new LinkedList<String[]>();

}

public int getT() {

return t;

}

public void setDiagramSize(Dimension diagramSize) {

this.diagramSize = diagramSize;

graphicHeight = (int) (diagramSize.getHeight() / (inNames.length + elementNames.length));

}

public ArrayList<String> getNames() {

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

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

names.add(inNames[i]);

}

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

names.add(elementNames[i]);

}

return names;

}

public ArrayList<HashMap<Integer, Character>> getDiagrams() {

return diagrams;

}

public Dimension getDiagramSize() {

return diagramSize;

}

public int getGraphicHeight() {

return graphicHeight;

}

public Color getBackgroundColor() {

return backgroundColor;

}

public Color getStandartColor() {

return standartColor;

}

public Color getTextColor() {

return textColor;

}

public Font getFont() {

return font;

}

@Override

public String getColumnName(int column) {

int temp = column;

if (temp < inNames.length) {

return inNames[temp];

} else {

temp -= inNames.length;

if (temp < elementNames.length) {

return elementNames[temp];

} else {

temp -= elementNames.length;

return LAST\_COLUMN\_NAMES[temp];

}

}

}

public int getRowCount() {

return rows.size();

}

public int getColumnCount() {

return columnCount;

}

public Object getValueAt(int rowIndex, int columnIndex) {

return rows.get(rowIndex)[columnIndex];

}

}

public class LogicFunctionsWorker {

public enum LogicFunction {AND, OR, NAND, NOR, XOR}

private static final char ONE = '1';

private static final char ZERO = '0';

public static char getValue(LogicFunction function, String inputSet) {

switch (function) {

case AND: {

return and(inputSet);

}

case OR: {

return or(inputSet);

}

case NAND: {

return nand(inputSet);

}

case NOR: {

return nor(inputSet);

}

case XOR: {

return xor(inputSet);

}

default: {

return ZERO;

}

}

}

private static char and(String inputSet) {

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

if (inputSet.charAt(i) == ZERO) {

return ZERO;

}

}

return ONE;

}

private static char or(String inputSet) {

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

if (inputSet.charAt(i) == ONE) {

return ONE;

}

}

return ZERO;

}

private static char nand(String inputSet) {

if (and(inputSet) == ONE) {

return ZERO;

}

return ONE;

}

private static char nor(String inputSet) {

if (or(inputSet) == ONE) {

return ZERO;

}

return ONE;

}

private static char xor(String inputSet) {

char result = xor2(inputSet.charAt(0), inputSet.charAt(1));

for (int i = 2; i < inputSet.length(); i++) {

result = xor2(result, inputSet.charAt(i));

}

return result;

}

private static char xor2(char ch1, char ch2) {

if (ch1 == ch2) {

return ZERO;

}

return ONE;

}

}

public interface LIDElement {

public int getInertiaDelay();

public int getDynamicDelay();

public void setInertiaDelay(int inertiaDelay);

public void setDynamicDelay(int dynamicDelay);

}

public interface VisualElement {

public void draw(Graphics2D g2);

public void move(int x, int y);

public void setSelected(boolean selected);

public boolean contains(Point p);

public boolean isOverlaped(Rectangle r);

}

public interface Exitable {

public boolean isInExit(Point point);

public Point getExitPoint(Point point) throws ConnectionAlreadyExistException;

public String getName();

public void addExitConnectionName(String name, Point point);

public void removeExitConnectionName(String name);

}

public interface Enterable {

public boolean isInEnter(Point point);

public Point getEnterPoint(Point point) throws ConnectionAlreadyExistException;

public String getName();

public void addEnterConnectionName(String name, Point point);

public void removeEnterConnectionName(String name);

}

public abstract class LogicElement extends ClosedElement implements LIDElement, Enterable, Exitable {

@XStreamAlias("inertiadelay")

protected int inertiaDelay;

@XStreamAlias("dynamicdelay")

protected int dynamicDelay;

@XStreamAlias("incount")

protected int inCount;

@XStreamAlias("innames")

protected String[] inNames;

@XStreamAlias("outnames")

protected String outName;

public LogicElement() {

super();

}

protected LogicElement(int x, int y, int width, int height, Color standartColor, Color selectedColor,

Color textColor, Font font, String name, int inertiaDelay, int dynamicDelay, int inCount) {

super(x, y, width, height, standartColor, selectedColor, textColor, font, name);

this.inertiaDelay = inertiaDelay;

this.dynamicDelay = dynamicDelay;

this.inCount = inCount;

inNames = new String[this.inCount];

}

public void draw(Graphics2D g2) {

g2.setColor(color);

g2.fillRect(x + width / 6, y, width / 3 \* 2, height);

int inDistance = height / (inCount + 1);

int inY = y;

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

inY += inDistance;

g2.drawLine(x, inY, x + width / 6, inY);

}

g2.drawLine(x + width / 6 \* 5, y + height / 2, x + width, y + height / 2);

g2.setColor(textColor);

g2.setFont(font);

FontRenderContext context = g2.getFontRenderContext();

Rectangle2D stringBounds = font.getStringBounds(name, context);

g2.drawString(name, x + width / 6 + (int) ((width / 3 \* 2 - stringBounds.getWidth()) / 2), y + height / 12 \* 11);

}

public int getInertiaDelay() {

return inertiaDelay;

}

public int getDynamicDelay() {

return dynamicDelay;

}

public void setInertiaDelay(int inertiaDelay) {

this.inertiaDelay = inertiaDelay;

}

public void setDynamicDelay(int dynamicDelay) {

this.dynamicDelay = dynamicDelay;

}

public int getInCount() {

return inCount;

}

public void setInCount(int inCount) {

if (inCount < this.inCount) {

String[] newInNames = new String[inCount];

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

newInNames[i] = inNames[i];

}

inNames = newInNames;

}

this.inCount = inCount;

}

public Point getEnterPoint(Point point) throws ConnectionAlreadyExistException {

int enterDistance = height / (inCount + 1);

if ((point.x >= x) && (point.x <= x + width / 6) &&

(point.y >= y + enterDistance / 2) && (point.y <= y + height - enterDistance / 2)) {

int inNumber = (point.y - y - enterDistance / 2) / enterDistance;

if (inNames[inNumber] != null) {

throw new ConnectionAlreadyExistException(true);

}

return new Point(x, y + enterDistance \* (inNumber + 1));

} else {

return null;

}

}

public Point getExitPoint(Point point) throws ConnectionAlreadyExistException {

if ((point.x >= x + width / 6 \* 5) && (point.x <= x + width) && (point.y >= y) && (point.y <= y + height)) {

if (outName != null) {

throw new ConnectionAlreadyExistException(false);

}

return new Point(x + width, y + height / 2);

} else {

return null;

}

}

public boolean isInEnter(Point point) {

try {

if (getEnterPoint(point) != null) {

return true;

}

return false;

} catch (ConnectionAlreadyExistException e) {

return false;

}

}

public boolean isInExit(Point point) {

try {

if (getExitPoint(point) != null) {

return true;

}

return false;

} catch (ConnectionAlreadyExistException e) {

return false;

}

}

public void addEnterConnectionName(String name, Point point) {

int enterDistance = height / (inCount + 1);

if ((point.x >= x) && (point.x <= x + width / 6) &&

(point.y >= y + enterDistance / 2) && (point.y <= y + height - enterDistance / 2)) {

int inNumber = (point.y - y - enterDistance / 2) / enterDistance;

inNames[inNumber] = name;

}

}

public void addExitConnectionName(String name, Point point) {

outName = name;

}

public void removeEnterConnectionName(String name) {

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

if (inNames[i] != null && inNames[i] == name) {

inNames[i] = null;

}

}

}

public void removeExitConnectionName(String name) {

outName = null;

}

}

public class OrElement extends LogicElement {

private static final String OR\_TEXT = "1";

public OrElement() {

super();

}

public OrElement(int x, int y, int width, int height, Color standartColor, Color selectedColor, Color textColor,

Font font, String name, int inertiaDelay, int dynamicDelay, int inCount) {

super(x, y, width, height, standartColor, selectedColor, textColor, font, name, inertiaDelay, dynamicDelay,

inCount);

}

@Override

public void draw(Graphics2D g2) {

super.draw(g2);

g2.setColor(textColor);

g2.setFont(font);

FontRenderContext context = g2.getFontRenderContext();

Rectangle2D stringBounds = font.getStringBounds(OR\_TEXT, context);

g2.drawString(OR\_TEXT, x + width / 6 + (int) ((width / 3 \* 2 - stringBounds.getWidth()) / 2),

(int) (y + stringBounds.getHeight()));

}

}

public class NorElement extends OrElement {

public NorElement() {

super();

}

public NorElement(int x, int y, int width, int height, Color standartColor, Color selectedColor, Color textColor,

Font font, String name, int inertiaDelay, int dynamicDelay, int inCount) {

super(x, y, width, height, standartColor, selectedColor, textColor, font, name, inertiaDelay, dynamicDelay,

inCount);

}

@Override

public void draw(Graphics2D g2) {

super.draw(g2);

g2.setColor(textColor);

g2.drawOval(x + width / 6 \* 5 - width / 12, y + height / 2 - width / 12, width / 6, width / 6);

}

}

public class AndElement extends LogicElement {

private static final String AND\_TEXT = "&";

public AndElement() {

super();

}

public AndElement(int x, int y, int width, int height, Color standartColor, Color selectedColor,

Color textColor, Font font, String name, int inertiaDelay, int dynamicDelay, int inCount) {

super(x, y, width, height, standartColor, selectedColor, textColor, font, name, inertiaDelay, dynamicDelay,

inCount);

}

@Override

public void draw(Graphics2D g2) {

super.draw(g2);

g2.setColor(textColor);

g2.setFont(font);

FontRenderContext context = g2.getFontRenderContext();

Rectangle2D stringBounds = font.getStringBounds(AND\_TEXT, context);

g2.drawString(AND\_TEXT, x + width / 6 + (int) ((width / 3 \* 2 - stringBounds.getWidth()) / 2),

(int) (y + stringBounds.getHeight()));

}

}

public class Program {

private static final File CONFIGURATION\_FILE = new File("conf.xml");

private static Configuration configuration;

private static void readConfigurationFile() {

try {

BufferedReader input = new BufferedReader(new FileReader(CONFIGURATION\_FILE));

StringBuilder builder = new StringBuilder();

String line;

while ((line = input.readLine()) != null) {

builder.append(line);

}

input.close();

XStream xStream = new XStream();

xStream.alias("configuration", Configuration.class);

configuration = (Configuration) xStream.fromXML(builder.toString());

} catch (IOException e) {

Toolkit kit = Toolkit.getDefaultToolkit();

Dimension screenSize = kit.getScreenSize();

configuration = new Configuration(new Rectangle((int) ((screenSize.getWidth() - MainFrame.MIN\_WIDTH) / 2),

(int) ((screenSize.getHeight() - MainFrame.MIN\_HEIGHT) / 2), MainFrame.MIN\_WIDTH, MainFrame.MIN\_HEIGHT));

}

}

public static void writeConfigurationFile(Rectangle mainFrameBounds) {

try {

configuration = new Configuration(mainFrameBounds);

XStream xStream = new XStream();

xStream.alias("configuration", Configuration.class);

String xml = xStream.toXML(configuration);

PrintWriter output = new PrintWriter(CONFIGURATION\_FILE);

output.write(xml);

output.close();

} catch (FileNotFoundException e) {

JOptionPane.showMessageDialog(null, "Can not save settings", "Error!",

JOptionPane.ERROR\_MESSAGE);

}

}

public static void main(String[] args) {

try {

// Set System L&F

UIManager.setLookAndFeel(

UIManager.getSystemLookAndFeelClassName());

}

catch (UnsupportedLookAndFeelException e) {

e.printStackTrace();

}

catch (ClassNotFoundException e) {

e.printStackTrace();

}

catch (InstantiationException e) {

e.printStackTrace();

}

catch (IllegalAccessException e) {

e.printStackTrace();

}

readConfigurationFile();

SwingUtilities.invokeLater(new Runnable() {

public void run() {

MainFrame mainFrame = new MainFrame(configuration.getMainFrameBounds());

mainFrame.setVisible(true);

}

});

}

}