**REPORT**

**JAVA MINI PROJECT**

**TOPIC: SIMULATION OF A BREADBOARD**

INDEX

|  |  |  |
| --- | --- | --- |
| Sr. No | Topic | Page no. |
| 1 | INTRODUCTION | 2 |
| 2 | PROBLEM DEFINITION | 3 |
| 3 | CONCEPTS OF OOPM USED | 3 |
| 4 | MODULES | 8 |
| 5 | PROGRAM | 9 |
| 6 | SCREENSHOTS | 54 |
| 7 | CONCLUSION | 56 |
| 8 | ACKNOWLEDGEMENT | 56 |

**INTRODUCTION:**

# Soldering every circuit you build probably isn’t practical.  At some point you are probably going to want to use some type of temporary method to connect different components together.  One of the popular methods is using a breadboard.  This simple (and cool) looking device only needs a few instructions before you can begin the use of breadboards (sometimes known as Blue-Boards or Plug-Boards) in electronics labs is well established. However, it is also a problem, in that students may not be able to experiment outside of scheduled lab time without supervision. If a breadboard simulation were available online it would greatly enhance learning resources for these students.

# Layout of a Breadboard

There are various colors, shapes, and sizes that breadboards come in.  However, they all generally look about the same and have the same basic functionality.  The board shown in Picture 1 is a pretty common “medium” sized breadboard.



Generally there will be some labels for the rows and columns of the breadboard.  On higher quality boards these letters will be silkscreened for better readability.  Just a personal note, I hardly ever reference these designators.  You could, for example, use them to keep track of pins on an IC.  However, I find it less error prone to just count the pins on an IC than do offset math.

**PROBLEM DEFINITION:**

The Objective of the project would be to develop a JAVA Applet which displays the image of a breadboard and allows components to be selected for the circuit. Once a circuit has been defined, the user can then select various input from a set of available inputs. The project could focus exclusively on analog or digital circuits, or take a general approach and allow both types of components to be used. There will be some graphics involved as part of the JAVA programming.

However, the problem using a real-life breadboard is that,students may not be able to experiment outside of scheduled lab time without supervision. If a breadboard simulation were available online it would greatly enhance learning resources for these students.

Also it prevents from any kind of electrical issues like short-ciruit and very well prevents the threat to life.

Most bread boards can handle up to 1A through their connections at digital-level voltages (5V and below).  Even so, I would really try to limit current through any breadboard to the 500mA range.

Over time, the little clips and springs that make up the connections inside the breadboard wear out.  If you push a wire in too far (and too hard) you may make unintended connections.  As you change connections around, you might bump a wire or resistor sticking up which may make (or break) a connection.

**CONCEPTS OF OOPM USED FOR THIS PROJECT:**

**Applet**

**Why use Java Applet?**

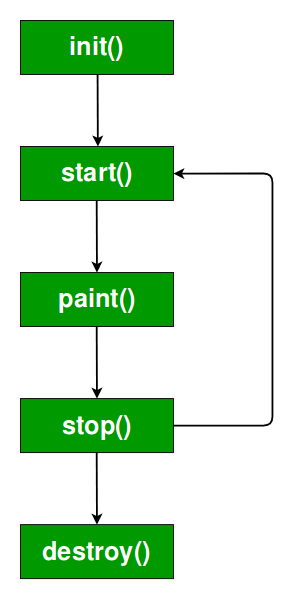
Applet is a Java program that can be embedded into a web page. It runs inside the web browser and works at client side. Applet is embedded in a HTML page using the APPLET or OBJECT tag and hosted on a web server.

Applets are used to make the web site more dynamic and entertaining.

**Some important points :**

1. All applets are sub-classes (either directly or indirectly) of [*java.applet.Applet*](https://docs.oracle.com/javase/7/docs/api/java/applet/Applet.html) class.
2. Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
3. In general, execution of an applet does not begin at main() method.  
   Output of an applet window is not performed by *System.out.println()*. Rather it is handled with various AWT methods, such as *drawString()*.

**Life cycle of an applet :**



**Features of Applets over HTML**

* + Displaying dynamic web pages of a web application.
  + Playing sound files.
  + Displaying documents
  + Playing animations

**Event Handling**

|  |
| --- |
| Changing the state of an object is known as an event. For example, click on button, dragging mouse etc. The java.awt.event package provides many event classes and Listener interfaces for event handling. |

**Java Event classes and Listener interfaces**

|  |  |
| --- | --- |
| **Event Classes** | **Listener Interfaces** |
| ActionEvent | ActionListener |
| MouseEvent | MouseListener and MouseMotionListener |
| MouseWheelEvent | MouseWheelListener |
| KeyEvent | KeyListener |
| ItemEvent | ItemListener |
| TextEvent | TextListener |
| AdjustmentEvent | AdjustmentListener |
| WindowEvent | WindowListener |
| ComponentEvent | ComponentListener |
| ContainerEvent | ContainerListener |
| FocusEvent | FocusListener |

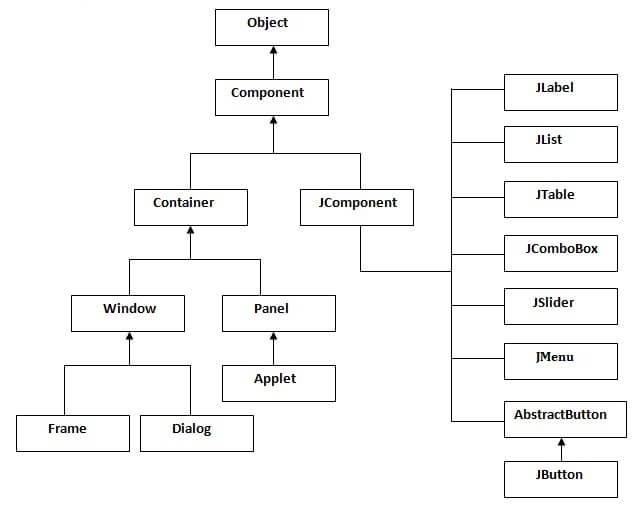
**SWING IN JAVA**

**Java Swing tutorial** is a part of Java Foundation Classes (JFC) that is used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

The hierarchy of java swing API is given below.



**DIFFERENCE BETWEEN AWT AND SWING**

|  |  |
| --- | --- |
| **AWT** | **Swing** |
| AWT stands for Abstract windows toolkit. | Swing is also called as JFC’s (Java Foundation classes). |
| AWT components are called Heavyweight component. | Swings are called light weight component because swing components sits on the top of AWT components and do the work. |
| AWT components require java.awt package. | Swing components require javax.swing package. |
| AWT components are platform dependent. | Swing components are made in purely java and they are platform independent. |
| This feature is not supported in AWT. | We can have different look and feel in Swing. |
| These feature is not available in AWT. | Swing has many advanced features like JTabel, Jtabbed pane which is not available in AWT. Also. Swing components are called "lightweight" because they do not require a native OS object to implement their functionality. JDialog and JFrame are heavyweight, because they do have a peer. So components like JButton, JTextArea, etc., are lightweight because they do not have an OS peer. |
| With AWT, you have 21 "peers" (one for each control and one for the dialog itself). A "peer" is a widget provided by the operating system, such as a button object or an entry field object. | With Swing, you would have only one peer, the operating system's window object. All of the buttons, entry fields, etc. are drawn by the Swing package on the drawing surface provided by the window object. This is the reason that Swing has more code. It has to draw the button or other control and implement its behavior instead of relying on the host operating system to perform those functions. |
| AWT is a thin layer of code on top of the OS. | Swing is much larger. Swing also has very much richer functionality. |
| Using AWT, you have to implement a lot of things yourself. | Swing has them buil |

**CLASS as a User-Defined Data type**

We have used various classes as a data type to store various data types. It acts as a container and its objects are referenced throughout the program to simplify the code and reduce Complexity.

Classes Used:

* *Attribute =>* int link, int value, Color color  
  ~> Acts as the attribute of a Pin which helps us access it easily. It stores  
  the *link*ed value and the *value* or voltage of the pin.
* *Cordinate =>* int s, int r, int c  
  ~> It acts as the coordinate system to access a pin from the breadboard.
* *Pins =>* Cordinate src, Attribute ob, Pins next~> Acts as the overall description of a Pin, storing its Cordinate, its  
  Attributes and the object of the pin existing after this.
* *color =>* int r, int g, int b  
  ~> Acts as a coordinate system for the random color combinations.

**Encapsulation and Data Hiding**

The above classes used are the perfect example of *Encapsulation* and *Data Hiding*.

**Vectors**

Throughout the program, we have used various vectors to store various types of data. This has helped us make variables easily accessible throughout the program. It has also helped us with the storage of dynamically created Pins.

**MODULES:**

The problem statement is divided into smaller parts, to make it easier to understand and implement it in various places.

*Various Classes:*

* *BreadBoard.java =>* It implements the basic layout of the breadboard structure and stores the most important Data Members and Member functions which are used throughout the program.
* *Connect.java =>* It implements the second window which pops up after clicking any of the pins of the board. It gives the user choices and further classes are called based on the choice.
* *ICOption.java =>* It implements the window which gives user the choice as to which IC he would like to add.
* *Gates.java =>* It implements the back-end logic of each IC chosen by the user, which calculates and stores the final output.
* *PinOption =>* It implements the connection of one pin to another and based on the entries given by the user, performs the required task.
* *Display =>* It implements the final display window which shows the different inputs and outputs inserted in a program.
* *Simulator.java =>* Implements the main method having a JFrame which contains the Applet.

**PROGRAM:**

*BreadBoard.java*

package tryout;

import java.applet.Applet;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.Random;

import java.util.Vector;

import javax.swing.\*;

class Attribute {

int link;

int value;

Color color;

Attribute(int l, Color clr) {

link = l;

color = clr;

value = -1;

}

Attribute(int l, int v) {

link = l;

value = v;

}

}

class Cordinate {

int s, r, c;

Cordinate(int s, int r, int c) {

this.s = s;

this.r = r;

this.c = c;

}

}

class Pins {

Cordinate src;

Attribute ob;

Pins next;

Pins(Cordinate src) {

this.src = src;

}

void add(Pins ob) {

Pins tmp = this;

while(tmp.next!=null) {

tmp = tmp.next;

}

tmp.next = ob;

}

}

class color {

int r, g, b;

}

//<applet code="BreadBoard.class" width = 1750 height = 400> </applet>

public class BreadBoard extends Applet implements ActionListener{

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

JLabel lblbt[], lblb[];

char row[];

JLabel tmplbl[];

JLabel lbl;

static Vector<color> vcolor = new Vector<color>();

static JButton pin[][][];

static Attribute ofpin[][][];

static Vector<JButton> icgate = new Vector<JButton>();

@SuppressWarnings("rawtypes")

static Vector gate = new Vector();

static Vector<Cordinate> inpt = new Vector<Cordinate>();

static Vector<Cordinate> otpt = new Vector<Cordinate>();

static Vector<Cordinate> vcc = new Vector<Cordinate>();

static Vector<Cordinate> gnd = new Vector<Cordinate>();

static Vector<Pins> wire = new Vector<Pins>();

static Applet board;

JPanel brd, pnl1, pnl2, pnl3, pnl0;

static JPanel pnlpn;

static ImageIcon icon;

ImageIcon ic;

Dimension dmpn;

Rectangle btn;

boolean flag, counter;

static boolean canDisplay;

public void init() {

counter = true;

board = this;

setSize(1750, 300);

icon = new ImageIcon(ClassLoader.getSystemResource("breadboard\_pin.png"));

ic = new ImageIcon("");

tmplbl = new JLabel[12];

row = new char[]{' ', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', ' '};

for(int i=0;i<12;i++) {

tmplbl[i] = new JLabel(" " + row[i]);

tmplbl[i].setFont(new Font("SansSerif", Font.BOLD, 9));

tmplbl[i].setHorizontalTextPosition(JLabel.CENTER);

}

lblb = new JLabel[64];

pin = new JButton[2][5][64];

ofpin = new Attribute[2][5][64];

lblbt = new JLabel[64];

lbl = new JLabel();

brd = new JPanel();

pnl0 = new JPanel();

pnl1 = new JPanel();

pnl2 = new JPanel();

pnl3 = new JPanel();

pnlpn = new JPanel();

pnl0.setLayout(new GridLayout(1, 64));

pnl0.setBackground(new Color(255, 250, 255));

pnl1.setLayout(new GridLayout(5, 64));

pnl1.setBackground(new Color(255, 250, 240));

pnl2.setLayout(new GridLayout(5, 64));

pnl2.setBackground(new Color(255, 250, 240));

pnl3.setLayout(new GridLayout(1, 64));

pnl3.setBackground(new Color(255, 250, 255));

pnlpn.setLayout(null);

pnl0.add(tmplbl[0]);

pnl3.add(tmplbl[11]);

for(int i=0;i<5;i++) {

pnl1.add(tmplbl[i+1]);

pnl2.add(tmplbl[i+6]);

for(int j=0;j<64;j++) {

pin[0][i][j] = new JButton();

pin[0][i][j].setIcon(icon);

pin[0][i][j].setBorderPainted(false);

pin[0][i][j].setContentAreaFilled(false);

pin[0][i][j].addActionListener(this);

pnl1.add(pin[0][i][j]);

pin[1][i][j] = new JButton();

pin[1][i][j].setIcon(icon);

pin[1][i][j].setBorderPainted(false);

pin[1][i][j].setContentAreaFilled(false);

pin[1][i][j].addActionListener(this);

pnl2.add(pin[1][i][j]);

ofpin[0][i][j] = new Attribute(-1, -1);

ofpin[1][i][j] = new Attribute(-1, -1);

if(i==0) {

if(j<10) {

lblb[j] = new JLabel(" "+j);

lblbt[j] = new JLabel(" "+j);

} else {

lblb[j] = new JLabel(" " +j);

lblbt[j] = new JLabel(" " +j);

}

pnl0.add(lblb[j]);

pnl3.add(lblbt[j]);

}

}

}

setLayout(new FlowLayout());

pnl0.setPreferredSize(new Dimension(1500, 5));

pnl3.setPreferredSize(new Dimension(1500, 5));

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

brd.setPreferredSize(new Dimension(1500, 250));

brd.add(pnl0);

brd.add(pnl1);

brd.add(pnlpn);

brd.add(pnl2);

brd.add(pnl3);

add(brd);

}

@Override

public void actionPerformed(ActionEvent arg0) {

JButton prsd = (JButton)arg0.getSource();

Cordinate src = getButton(prsd);

Object[] options;

int ch;

if(prsd.getIcon()==null) {

options = new Object[]{"Remove connection", "Change Connection", "Return"};

ch = JOptionPane.showOptionDialog(null, "Would you like to ...?", "PIN CONNECTION",

JOptionPane.DEFAULT\_OPTION, JOptionPane.INFORMATION\_MESSAGE, null, options, null);

switch(ch) {

case 0:

//Remove

break;

case 1:

//Remove

linkPin(src);

}

} else if(prsd.getIcon().toString().endsWith("vcc.png")) {

options = new Object[]{"Yes", "No"};

ch = JOptionPane.showOptionDialog(null, "Would you like to Remove Vcc?", "PIN CONNECTION",

JOptionPane.DEFAULT\_OPTION, JOptionPane.INFORMATION\_MESSAGE, null, options, null);

if(ch==0) {

BreadBoard.removeFrom(src, vcc);

BreadBoard.removeValue(src);

BreadBoard.ofpin[src.s][src.r][src.c] = new Attribute(-1, -1);

BreadBoard.getPin(src).setIcon(icon);

}

} else if(prsd.getIcon().toString().endsWith("gnd.png")) {

options = new Object[]{"Yes", "No"};

ch = JOptionPane.showOptionDialog(null, "Would you like to Remove Ground?", "PIN CONNECTION",

JOptionPane.DEFAULT\_OPTION, JOptionPane.INFORMATION\_MESSAGE, null, options, null);

if(ch==0) {

BreadBoard.removeFrom(src, gnd);

BreadBoard.removeValue(src);

BreadBoard.ofpin[src.s][src.r][src.c] = new Attribute(-1, -1);

BreadBoard.getPin(src).setIcon(icon);

}

} else if(prsd.getIcon().toString().endsWith("otpt.png")) {

options = new Object[]{"Yes", "No"};

ch = JOptionPane.showOptionDialog(null, "Would you like to Remove Output?", "PIN CONNECTION",

JOptionPane.DEFAULT\_OPTION, JOptionPane.INFORMATION\_MESSAGE, null, options, null);

if(ch==0) {

BreadBoard.removeFrom(src, otpt);

BreadBoard.removeValue(src);

BreadBoard.ofpin[src.s][src.r][src.c] = new Attribute(-1, -1);

BreadBoard.getPin(src).setIcon(icon);

}

} else if(prsd.getIcon().toString().endsWith("inpt.png")) {

options = new Object[]{"Yes", "No"};

ch = JOptionPane.showOptionDialog(null, "Would you like to Remove Input?", "PIN CONNECTION",

JOptionPane.DEFAULT\_OPTION, JOptionPane.INFORMATION\_MESSAGE, null, options, null);

if(ch==0) {

BreadBoard.removeFrom(src, inpt);

BreadBoard.removeValue(src);

BreadBoard.ofpin[src.s][src.r][src.c] = new Attribute(-1, -1);

BreadBoard.getPin(src).setIcon(icon);

}

} else if(prsd.getIcon().toString().endsWith("pin.png")) {

linkPin(src);

}

}

static void addICGate(Cordinate src, String ic) {

ImageIcon imgIcn = new ImageIcon(ClassLoader.getSystemResource("breadboard\_ic.png"));

JButton icgate = new JButton(ic, imgIcn);

icgate.setForeground(new Color(255, 255, 255));

icgate.setHorizontalTextPosition(JButton.CENTER);

icgate.setVerticalTextPosition(JButton.CENTER);

icgate.setBounds(27+23\*src.c, 0, 161, 50);

pnlpn.add(icgate);

icgate.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

for(Cordinate tmp : vcc) {

addValue(tmp, 1);

}

for(Cordinate tmp : inpt) {

BreadBoard.ofpin[tmp.s][tmp.r][tmp.c].value = 0;

}

for(Cordinate tmp : otpt) {

BreadBoard.ofpin[tmp.s][tmp.r][tmp.c].value = getValue(tmp);

}

execute();

if(canDisplay) {

Display ob = new Display(inpt.size(), otpt.size());

ob.init();

}

}

});

BreadBoard.icgate.add(icgate);

icgate.requestFocus();

}

static void linkPin(Cordinate src) {

try {

BreadBoard.board.setEnabled(false);

Connect ob = new Connect(src);

ob.init();

} catch (Exception e) {

System.out.println("Not Found!");

}

}

Cordinate getButton(JButton prsd) {

Cordinate src = new Cordinate(0, 0, 0);

for(int i=0;i<5;i++) {

for(int j=0;j<64;j++) {

if(prsd.equals(pin[0][i][j])) {

src.s = 0;

src.r = i;

src.c = j;

} else if(prsd.equals(pin[1][i][j])) {

src.s = 1;

src.r = i;

src.c = j;

}

}

}

return src;

}

static boolean isIn(Cordinate src, Vector<Cordinate> v) {

boolean flag = false;

for(Cordinate tmp : v) {

if(src.s==tmp.s && src.r==tmp.r && src.c==tmp.c) {

flag = true;

break;

}

}

return flag;

}

static void removeFrom(Cordinate src, Vector<Cordinate> v) {

for(Cordinate tmp : v) {

if(src.s==tmp.s && src.r==tmp.r && src.c==tmp.c) {

v.remove(tmp);

break;

}

}

}

static Color getColor() {

Color clr;

color clrt = new color();

Random rand = new Random();

int rd, gr, bl;

do {

rd = rand.nextInt(255);

}while(isIn(rd, 0));

do {

gr = rand.nextInt(255);

}while(isIn(gr, 0));

do {

bl = rand.nextInt(255);

}while(isIn(bl, 0));

clrt.r = rd;

clrt.g = gr;

clrt.b = bl;

vcolor.add(clrt);

clr = new Color(rd, gr, bl);

return clr;

}

static JButton getPin(Cordinate src) {

return BreadBoard.pin[src.s][src.r][src.c];

}

static Attribute getOf(Cordinate src) {

Attribute ob = null;

for(int i=0;i<5;i++) {

ob = BreadBoard.ofpin[src.s][i][src.c];

if(ob!=null) {

break;

}

}

return ob;

}

static boolean isOutput(Cordinate src) {

boolean flag = false;

JButton btn;

int i, k=0;

for(i=0;i<5;i++) {

btn = BreadBoard.pin[src.s][i][src.c];

if(btn.getIcon()!=icon) {

flag = true;

break;

}

}

if(flag) {

do {

btn = BreadBoard.pin[src.s][i][src.c-k-1];

k++;

}while(btn.getIcon()!=icon && src.c-k!=0);

}

return flag;

}

static boolean checkValue(Cordinate src, int value) {

int i, tmp=-1;

for(i=0;i<5;i++) {

if(src.r!=i) {

tmp = BreadBoard.ofpin[src.s][i][src.c].value;

if(tmp!=-1) {

break;

}

}

}

if(value == 2) {

if(i==5) {

return false;

}

BreadBoard.ofpin[src.s][src.r][src.c] = BreadBoard.ofpin[src.s][i][src.c];

return true;

} else if(tmp==1 || tmp==0 || tmp==-2) {

return false;

} else {

addValue(src, value);

return true;

}

}

static void addValue(Cordinate src, int value) {

Attribute tmp;

for(int i=0;i<5;i++) {

tmp = BreadBoard.ofpin[src.s][i][src.c];

if(src.r!=i && tmp.link!=-1) {

tmp.value = value;

break;

}

}

}

static int getValue(Cordinate src) {

Attribute tmp;

System.out.print(src.s + "," + src.r + "," + src.c + "\t");

for(int i=0;i<5;i++) {

tmp = BreadBoard.ofpin[src.s][i][src.c];

if(src.r!=i && tmp.link!=-1 && tmp.value!=2) {

System.out.print(src.s + "," + i + "," + src.c + "\n");

return tmp.value;

}

}

System.out.print("Fail\n");

return 0;

}

static void removeValue(Cordinate src) {

int tmp;

for(int i=0;i<5;i++) {

tmp = BreadBoard.ofpin[src.s][i][src.c].link;

if(src.r!=i && tmp!=-1) {

BreadBoard.ofpin[src.s][i][src.c].value = -1;

break;

}

}

}

static boolean isIn(int c, int ch) {

boolean flag = false;

int cmp=0;

for(color clr : vcolor) {

switch(ch) {

case 1:

cmp = clr.r;

break;

case 2:

cmp = clr.g;

break;

case 3:

cmp = clr.b;

break;

}

if(c==cmp) {

flag = true;

break;

}

}

return flag;

}

static void execute() {

String s;

canDisplay = true;

for(Object ob : gate) {

s = ob.toString().substring(7);

if(s.startsWith("AND")) {

AND ic = (AND) ob;

ic.start();

} else if(s.startsWith("OR")) {

OR ic = (OR) ob;

ic.start();

} else if(s.startsWith("NOT")) {

NOT ic = (NOT) ob;

ic.start();

} else if(s.startsWith("NAND")) {

NAND ic = (NAND) ob;

ic.start();

} else if(s.startsWith("NOR")) {

NOR ic = (NOR) ob;

ic.start();

} else if(s.startsWith("XOR")) {

XOR ic = (XOR) ob;

ic.start();

}

}

}

}

*Connect.java*

package tryout;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowEvent;

import java.awt.event.WindowListener;

import javax.swing.\*;

public class Connect extends JFrame implements ActionListener {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

JFrame frame;

JLabel ptr[];

JLabel spc[];

JButton btn[];

JPanel pnl[];

private int Frame\_Width;

private int Frame\_Height;

private int Frame\_X;

private int Frame\_Y;

boolean flag;

Cordinate src;

Connect(Cordinate src) {

this.src = src;

flag = true;

}

public void init() {

frame = new JFrame();

Frame\_Width = 480; Frame\_Height = 450;

Frame\_X = 635; Frame\_Y = 204;

frame.setLayout(new FlowLayout());

String s[] = {"Add IC", "Add Input", "Add Output", "Add Vcc", "Add Ground", "Connect another Pin"};

spc = new JLabel[6];

for(int i=0;i<6;i++) {

spc[i] = new JLabel(" ");

spc[i].setFont(new Font("SansSerif", 0, 30));

}

ptr = new JLabel[6];

for(int i=0;i<6;i++) {

ptr[i] = new JLabel("~~>");

ptr[i].setFont(new Font("SansSerif", 0, 30));

}

btn = new JButton[6];

for(int i=0;i<6;i++) {

btn[i] = new JButton(s[i]);

btn[i].setPreferredSize(new Dimension(300, 50));

btn[i].setFont(new Font("SansSerif", 0, 20));

btn[i].addActionListener(this);

}

pnl = new JPanel[6];

for(int i=0;i<6;i++) {

pnl[i] = new JPanel();

pnl[i].setLayout(new FlowLayout());

pnl[i].add(ptr[i]);

pnl[i].add(btn[i]);

pnl[i].add(spc[i]);

frame.add(pnl[i]);

}

jFrame(frame);

}

//Create a generalized Frame:

private void jFrame(final JFrame frame) {

frame.setVisible(true);

frame.setTitle("Pin Configuration");

frame.setSize(Frame\_Width, Frame\_Height);

frame.setLocation(Frame\_X, Frame\_Y);

frame.addWindowListener(new WindowListener() {

public void windowActivated(WindowEvent arg0) {

}

public void windowClosed(WindowEvent arg0) {

}

public void windowClosing(WindowEvent arg0) {

if(flag) {

BreadBoard.board.setEnabled(true);

}

}

public void windowDeactivated(WindowEvent arg0) {

}

public void windowDeiconified(WindowEvent arg0) {

}

public void windowIconified(WindowEvent arg0) {

}

public void windowOpened(WindowEvent arg0) {

}

});

}

@Override

public void actionPerformed(ActionEvent arg0) {

JButton prsd = (JButton)arg0.getSource();

for(int i=0;i<6;i++) {

if(prsd.equals(btn[i])) {

call(i);

break;

}

}

frame.dispatchEvent(new WindowEvent(frame, WindowEvent.WINDOW\_CLOSING));

}

Cordinate isFree() {

Cordinate flag = null, tmp;

int s, r, c;

for(int pin=0;pin<14;pin++) {

s = (pin<7)?1:0;

r = (pin<7)?0:4;

c = src.c + ((pin<7)?pin:(13-pin));

tmp = new Cordinate(s, r, c);

if(!BreadBoard.getPin(tmp).getIcon().toString().endsWith("pin.png")) {

flag = tmp;

break;

}

}

return flag;

}

void call(int o) {

JButton tmp;

Icon icon;

Cordinate pin;

char ch;

int v;

try {

switch(o) {

case 0:

if(src.s!=0 || src.r!=4) {

display("An IC can only be added on the E Row.");

} else if(o==0 && src.c>57) {

display("An IC has 7 pin. Select a column less than 58.");

} else {

pin = isFree();

if (pin!=null) {

ch = (char)(65+pin.r+5\*pin.s);

display("Cannot place IC. Connection found at " + ch + "-" + pin.c + ".");

} else {

ICOption ob = new ICOption(src);

flag = false;

ob.init();

}

}

break;

case 1:

v = 0;

if(BreadBoard.checkValue(src, v)) {

icon = new ImageIcon(ClassLoader.getSystemResource("breadboard\_inpt.png"));

BreadBoard.inpt.add(src);

tmp = BreadBoard.getPin(src);

tmp.setIcon(icon);

BreadBoard.ofpin[src.s][src.r][src.c] = new Attribute(-2, v);

} else {

display("Error! Another Connection already Exists!");

}

break;

case 2:

v = 2;

BreadBoard.checkValue(src, v);

icon = new ImageIcon(ClassLoader.getSystemResource("breadboard\_otpt.png"));

BreadBoard.otpt.add(src);

tmp = BreadBoard.getPin(src);

tmp.setIcon(icon);

BreadBoard.ofpin[src.s][src.r][src.c] = new Attribute(-2, v);

break;

case 3:

v = 1;

if(BreadBoard.checkValue(src, v)) {

icon = new ImageIcon(ClassLoader.getSystemResource("breadboard\_vcc.png"));

BreadBoard.vcc.add(src);

tmp = BreadBoard.getPin(src);

tmp.setIcon(icon);

BreadBoard.ofpin[src.s][src.r][src.c] = new Attribute(-2, v);

} else {

display("Error! Another Connection already Exists!");

}

break;

case 4:

v = -2;

if(BreadBoard.checkValue(src, v)) {

icon = new ImageIcon(ClassLoader.getSystemResource("breadboard\_gnd.png"));

BreadBoard.gnd.add(src);

tmp = BreadBoard.getPin(src);

tmp.setIcon(icon);

BreadBoard.ofpin[src.s][src.r][src.c] = new Attribute(-2, v);

} else {

display("Error! Another Connection already Exists!");

}

break;

case 5:

flag = false;

PinOption ob = new PinOption(src);

ob.init();

}

} catch (Exception e) {

System.out.println(o);

}

}

public void display(String s) {

JOptionPane.showMessageDialog(frame, s, "BREADBOARD", JOptionPane.INFORMATION\_MESSAGE, null);

}

}

*ICOption.java*

package tryout;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowEvent;

import java.awt.event.WindowListener;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

public class ICOption implements ActionListener {

JFrame frame;

JLabel ptr[];

JLabel spc[];

JButton btn[];

JPanel pnl[];

private int Frame\_Width;

private int Frame\_Height;

private int Frame\_X;

private int Frame\_Y;

Cordinate src;

public ICOption(Cordinate src) {

this.src = src;

}

public void init() {

frame = new JFrame();

Frame\_Width = 480; Frame\_Height = 450;

Frame\_X = 635; Frame\_Y = 200;

frame.setLayout(new FlowLayout());

String s[] = {"AND", "OR", "NOT", "NAND", "NOR", "X-OR"};

spc = new JLabel[6];

for(int i=0;i<6;i++) {

spc[i] = new JLabel(" ");

spc[i].setFont(new Font("SansSerif", 0, 30));

}

ptr = new JLabel[6];

for(int i=0;i<6;i++) {

ptr[i] = new JLabel("~~>");

ptr[i].setFont(new Font("SansSerif", 0, 30));

}

btn = new JButton[6];

for(int i=0;i<6;i++) {

btn[i] = new JButton(s[i]);

btn[i].setPreferredSize(new Dimension(300, 50));

btn[i].setFont(new Font("SansSerif", 0, 20));

btn[i].addActionListener(this);

}

pnl = new JPanel[6];

for(int i=0;i<6;i++) {

pnl[i] = new JPanel();

pnl[i].setLayout(new FlowLayout());

pnl[i].add(ptr[i]);

pnl[i].add(btn[i]);

pnl[i].add(spc[i]);

frame.add(pnl[i]);

}

jFrame(frame);

}

private void jFrame(final JFrame frame) {

frame.setVisible(true);

frame.setTitle("Pin Configuration");

frame.setSize(Frame\_Width, Frame\_Height);

frame.setLocation(Frame\_X, Frame\_Y);

frame.addWindowListener(new WindowListener() {

public void windowActivated(WindowEvent arg0) {

}

public void windowClosed(WindowEvent arg0) {

}

public void windowClosing(WindowEvent arg0) {

BreadBoard.board.setEnabled(true);

}

public void windowDeactivated(WindowEvent arg0) {

}

public void windowDeiconified(WindowEvent arg0) {

}

public void windowIconified(WindowEvent arg0) {

}

public void windowOpened(WindowEvent arg0) {

}

});

}

@SuppressWarnings("unchecked")

@Override

public void actionPerformed(ActionEvent arg0) {

JButton prsd = (JButton)arg0.getSource();

if(prsd.equals(btn[0])) {

AND ic = new AND(src);

ic.init();

BreadBoard.gate.add(ic);

BreadBoard.addICGate(src, "IC - 7408");

} else if(prsd.equals(btn[1])) {

OR ic = new OR(src);

ic.init();

BreadBoard.gate.add(ic);

BreadBoard.addICGate(src, "IC - 7432");

} else if(prsd.equals(btn[2])) {

NOT ic = new NOT(src);

ic.init();

BreadBoard.gate.add(ic);

BreadBoard.addICGate(src, "IC - 7404");

} else if(prsd.equals(btn[3])) {

NAND ic = new NAND(src);

ic.init();

BreadBoard.gate.add(ic);

BreadBoard.addICGate(src, "IC - 7400");

} else if(prsd.equals(btn[4])) {

NOR ic = new NOR(src);

ic.init();

BreadBoard.gate.add(ic);

BreadBoard.addICGate(src, "IC - 7402");

} else {

XOR ic = new XOR(src);

ic.init();

BreadBoard.gate.add(ic);

BreadBoard.addICGate(src, "IC - 7486");

}

frame.dispatchEvent(new WindowEvent(frame, WindowEvent.WINDOW\_CLOSING));

}

}

*Gates.java*

package tryout;

import javax.swing.ImageIcon;

import javax.swing.JOptionPane;

class ICGate {

Cordinate src;

boolean isGnd() {

Cordinate tmp = new Cordinate(1, 0, src.c+6);

if(BreadBoard.getValue(tmp)==-2) {

return true;

}

return false;

}

boolean check(int ch) {

boolean flag = true;

Attribute tmp[] = new Attribute[3];

int i, s, r, c, t, k;

t = (ch==0)?7408 : ((ch==1)?7432 : ((ch==3)?7400 : 7486));

k = 0;

if(BreadBoard.getValue(src)!=1) {

display("Pin 14 should have Vcc connected of IC " + t);

return false;

}

if(!isGnd()) {

display("Pin 7 should have Ground connected of IC " + t);

return false;

}

for(i=0;i<13;i+=3) {

if(i==6) {

i++;

}

s = (i<7)?1:0;

r = (i<7)?0:4;

for(int j=0;j<3;j++) {

c = src.c + ((i+j<7)?i+j:(13-i-j));

tmp[j] = BreadBoard.ofpin[s][r][c];

}

if(tmp[0].link!=-1 && tmp[1].link!=-1 && tmp[2].link!=-1){

if(tmp[0].value!=-1 && tmp[1].value!=-1 && tmp[2].value!=-1) {

calculate(i, ch);

}

k++;

} else if(!(tmp[0].link==-1 && tmp[1].link==-1 && tmp[2].link==-1)) {

display("Incomplete Connection at Gate " + (((i+1)/4)+1) + " of IC " + t);

return false;

}

}

if(k==0) {

display("Redundant IC " + t);

return false;

}

return flag;

}

void calculate(int pin, int ch) {

Attribute ob;

int s, r, c, input1, input2;

s = (pin<7)?1:0;

r = (pin<7)?0:4;

c = src.c + ((pin<7)?pin:(13-pin));

if(pin<7) {

input1 = BreadBoard.getValue(new Cordinate(s, r, c));

input2 = BreadBoard.getValue(new Cordinate(s, r, c+1));

ob = BreadBoard.ofpin[s][r][c+2];

ob.link = -2;

ob.value = getOutput(input1, input2, ch);

BreadBoard.addValue(new Cordinate(s, r, c+2), ob.value);

} else {

input1 = BreadBoard.getValue(new Cordinate(s, r, c-2));

input2 = BreadBoard.getValue(new Cordinate(s, r, c-1));

ob = BreadBoard.ofpin[s][r][c];

ob.link = -2;

ob.value = getOutput(input1, input2, ch);

BreadBoard.addValue(new Cordinate(s, r, c), ob.value);

}

}

int getOutput(int i1, int i2, int ch) {

boolean input1 = (i1==1);

boolean input2 = (i2==1);

boolean output = false;

switch(ch) {

case 0:

output = input1&&input2;

break;

case 1:

output = input1||input2;

break;

case 3:

output = !(input1&&input2);

break;

case 4:

output = !(input1||input2);

break;

case 5:

output = (input1!=input2);

}

return (output)?1:0;

}

int getOutput(int i) {

boolean input = (i==1);

boolean output = !input;

return (output)?1:0;

}

void link(int pin) {

int s, r, c, tmp;

String url = "breadboard\_pin";

url += ((pin<7)?"up":"down") + ".png";

ImageIcon icon = new ImageIcon(ClassLoader.getSystemResource(url));

s = (pin<7)?1:0;

r = (pin<7)?0:4;

c = src.c + ((pin<7)?pin:(13-pin));

Attribute ob = null;

BreadBoard.pin[s][r][c].setIcon(icon);

for(int i=0;i<5;i++) {

tmp = r + i\*((pin<7)?1:-1);

ob = BreadBoard.ofpin[s][tmp][c];

if(ob.link!=-1) {

break;

}

}

BreadBoard.ofpin[s][r][c] = ob;

}

public void display(String s) {

JOptionPane.showMessageDialog(null, s, "BREADBOARD", JOptionPane.INFORMATION\_MESSAGE, null);

}

}

class AND extends ICGate{

AND(Cordinate src) {

this.src = src;

}

void init() {

for(int i=0;i<14;i++) {

link(i);

}

}

void start() {

init();

BreadBoard.canDisplay = BreadBoard.canDisplay && check(0);

}

}

class OR extends ICGate{

OR(Cordinate src) {

this.src = src;

}

void init() {

for(int i=0;i<14;i++) {

link(i);

}

}

void start() {

init();

BreadBoard.canDisplay = BreadBoard.canDisplay && check(1);

}

}

class NOT extends ICGate{

NOT(Cordinate src) {

this.src = src;

}

void init() {

for(int i=0;i<14;i++) {

link(i);

}

}

void start() {

init();

BreadBoard.canDisplay = BreadBoard.canDisplay && check();

}

boolean check() {

boolean flag = true;

Attribute tmp[] = new Attribute[2];

int i, s, r, c, k=0;

if(BreadBoard.getValue(src)!=1) {

display("Pin 14 should have Vcc connected of IC 7404");

return false;

}

if(!isGnd()) {

display("Pin 7 should have Ground connected of IC 7404");

return false;

}

for(i=0;i<13;i+=2) {

if(i==6) {

i++;

}

s = (i<7)?1:0;

r = (i<7)?0:4;

for(int j=0;j<2;j++) {

c = src.c + ((i+j<7)?i+j:(13-i-j));

tmp[j] = BreadBoard.ofpin[s][r][c];

}

if(tmp[0].link!=-1 && tmp[1].link!=-1) {

if(tmp[0].value!=-1 && tmp[1].value!=-1) {

calculate(i);

}

k++;

} else if(!(tmp[0].link==-1 && tmp[1].link==-1)) {

display("Incomplete Connection at Gate " + ((i/2)+1) + " of IC 7404");

return false;

} else {

tmp[(i<7)?1 : 0].value = 3;

}

}

if(k==0) {

display("Redundant IC 7404");

return false;

}

return flag;

}

void calculate(int pin) {

Attribute ob;

int s, r, c, input;

s = (pin<7)?1:0;

r = (pin<7)?0:4;

c = src.c + ((pin<7)?pin:(13-pin));

if(pin<7) {

input = BreadBoard.getValue(new Cordinate(s, r, c));

ob = BreadBoard.ofpin[s][r][c+1];

ob.link = -2;

ob.value = getOutput(input);

BreadBoard.addValue(new Cordinate(s, r, c+1), ob.value);

} else {

input = BreadBoard.getValue(new Cordinate(s, r, c-1));

ob = BreadBoard.ofpin[s][r][c];

ob.link = -2;

ob.value = getOutput(input);

BreadBoard.addValue(new Cordinate(s, r, c), ob.value);

}

}

}

class NAND extends ICGate{

NAND(Cordinate src) {

this.src = src;

}

void init() {

for(int i=0;i<14;i++) {

link(i);

}

}

void start() {

init();

BreadBoard.canDisplay = BreadBoard.canDisplay && check(3);

}

}

class NOR extends ICGate{

NOR(Cordinate src) {

this.src = src;

}

void init() {

for(int i=0;i<14;i++) {

link(i);

}

}

void start() {

init();

BreadBoard.canDisplay = BreadBoard.canDisplay && check();

}

boolean check() {

boolean flag = true;

Attribute tmp[] = new Attribute[3];

int i, s, r, c, k=0;

if(BreadBoard.getValue(src)!=1) {

display("Pin 14 should have Vcc connected of IC 7402");

return false;

}

if(!isGnd()) {

display("Pin 7 should have Ground connected of IC 7402");

return false;

}

for(i=0;i<13;i+=3) {

if(i==6) {

i++;

}

s = (i<7)?1:0;

r = (i<7)?0:4;

for(int j=0;j<3;j++) {

c = src.c + ((i+j<7)?i+j:(13-i-j));

tmp[j] = BreadBoard.ofpin[s][r][c];

}

if(tmp[0].link!=-1 && tmp[1].link!=-1 && tmp[2].link!=-1){

if(tmp[0].value!=-1 && tmp[1].value!=-1 && tmp[2].value!=-1) {

calculate(i);

}

k++;

} else if(!(tmp[0].link==-1 && tmp[1].link==-1 && tmp[2].link==-1)) {

display("Incomplete Connection at Gate " + (((i+1)/4)+1) + " of IC 7402");

return false;

} else {

tmp[(i<7)?0 : 2].value = 3;

}

}

if(k==0) {

display("Redundant IC 7402");

return false;

}

return flag;

}

void calculate(int pin) {

Attribute ob;

int s, r, c, input1, input2;

s = (pin<7)?1:0;

r = (pin<7)?0:4;

c = src.c + ((pin<7)?pin:(13-pin));

if(pin<7) {

input1 = BreadBoard.getValue(new Cordinate(s, r, c-2));

input2 = BreadBoard.getValue(new Cordinate(s, r, c-1));

ob = BreadBoard.ofpin[s][r][c];

ob.link = -2;

ob.value = getOutput(input1, input2, 4);

BreadBoard.addValue(new Cordinate(s, r, c), ob.value);

} else {

input1 = BreadBoard.getValue(new Cordinate(s, r, c));

input2 = BreadBoard.getValue(new Cordinate(s, r, c+1));

ob = BreadBoard.ofpin[s][r][c+2];

ob.link = -2;

ob.value = getOutput(input1, input2, 4);

BreadBoard.addValue(new Cordinate(s, r, c+2), ob.value);

}

}

}

class XOR extends ICGate{

XOR(Cordinate src) {

this.src = src;

}

void init() {

for(int i=0;i<14;i++) {

link(i);

}

}

void start() {

init();

BreadBoard.canDisplay = BreadBoard.canDisplay && check(5);

}

}

*PinOption.java*

package tryout;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.WindowEvent;

import java.awt.event.WindowListener;

import javax.swing.\*;

import javax.swing.border.EtchedBorder;

public class PinOption extends JFrame implements ActionListener {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

JFrame frame;

JLabel intr;

JLabel tmp1, tmp2, tmp3, tmp4;

JLabel rlbl, clbl;

JTextArea rin, cin;

JButton sbmt, btn1, btn2;

JPanel tpnl, rpnl, cpnl, bpnl;

private int Frame\_Width;

private int Frame\_Height;

private int Frame\_X;

private int Frame\_Y;

Cordinate src, src1;

PinOption(Cordinate src) {

this.src = src;

btn1 = BreadBoard.getPin(src);

src1 = new Cordinate(0, 0, 0);

}

public void init() {

frame = new JFrame();

Frame\_Width = 540; Frame\_Height = 280;

Frame\_X = 600; Frame\_Y = 280;

frame.setLayout(new FlowLayout());

intr = new JLabel(" Enter the co-ordinates of the breadboard:");

intr.setFont(new Font("SansSerif", 0, 20));

intr.setPreferredSize(new Dimension(450, 50));

rlbl = new JLabel(" ROW");

rlbl.setFont(new Font("SansSerif", 0, 18));

rlbl.setPreferredSize(new Dimension(100, 25));

clbl = new JLabel("COLUMN");

clbl.setFont(new Font("SansSerif", 0, 18));

clbl.setPreferredSize(new Dimension(100, 25));

tmp1 = new JLabel("");

tmp1.setPreferredSize(new Dimension(75, 25));

tmp2 = new JLabel("");

tmp2.setPreferredSize(new Dimension(75, 25));

tmp3 = new JLabel("");

tmp3.setPreferredSize(new Dimension(75, 25));

tmp4 = new JLabel("");

tmp4.setPreferredSize(new Dimension(75, 25));

rin = new JTextArea();

rin.setFont(new Font("SansSerif", 0, 18));

rin.setPreferredSize(new Dimension(100, 25));

cin = new JTextArea();

cin.setFont(new Font("SansSerif", 0, 18));

cin.setPreferredSize(new Dimension(100, 25));

tpnl = new JPanel();

rpnl = new JPanel();

cpnl = new JPanel();

bpnl = new JPanel();

sbmt = new JButton("Submit");

sbmt.setFont(new Font("SansSerif", 0, 21));

sbmt.setPreferredSize(new Dimension(125, 30));

sbmt.addActionListener(this);

rpnl.setLayout(new FlowLayout());

rpnl.setPreferredSize(new Dimension(250, 50));

rpnl.add(rlbl);

rpnl.add(rin);

rpnl.setBorder(new EtchedBorder());

cpnl.setLayout(new FlowLayout());

cpnl.setPreferredSize(new Dimension(250, 50));

cpnl.add(clbl);

cpnl.add(cin);

cpnl.setBorder(new EtchedBorder());

bpnl.setLayout(new GridBagLayout());

GridBagConstraints gbc = new GridBagConstraints();

gbc.anchor = GridBagConstraints.CENTER;

bpnl.setPreferredSize(new Dimension(250, 50));

bpnl.add(sbmt, gbc);

tpnl.setLayout(new FlowLayout());

tpnl.setPreferredSize(new Dimension(450, 250));

tpnl.add(intr);

tpnl.add(tmp1);

tpnl.add(rpnl);

tpnl.add(tmp2);

tpnl.add(tmp3);

tpnl.add(cpnl);

tpnl.add(tmp4);

tpnl.add(bpnl);

frame.add(tpnl);

jFrame(frame);

}

//Create a generalized Frame:

private void jFrame(final JFrame frame) {

frame.setVisible(true);

frame.setTitle("Connect Configuration");

frame.setSize(Frame\_Width, Frame\_Height);

frame.setLocation(Frame\_X, Frame\_Y);

frame.addWindowListener(new WindowListener() {

public void windowActivated(WindowEvent arg0) {

}

public void windowClosed(WindowEvent arg0) {

}

public void windowClosing(WindowEvent arg0) {

BreadBoard.board.setEnabled(true);

}

public void windowDeactivated(WindowEvent arg0) {

}

public void windowDeiconified(WindowEvent arg0) {

}

public void windowIconified(WindowEvent arg0) {

}

public void windowOpened(WindowEvent arg0) {

}

});

}

@Override

public void actionPerformed(ActionEvent arg0) {

String rs, cs;

rs = rin.getText();

cs = cin.getText();

boolean flag = false;

try {

src1.c = Integer.valueOf(cs);

} catch(Exception e) {

flag = true;

}

if(rs.length()==0) {

display("Row should not be empty!");

frame.requestFocus();

} else if(rs.length()>1) {

display("Row should be a single character!");

frame.requestFocus();

} else if(rs.charAt(0)<'A' || rs.charAt(0)>'J') {

display("Row should be between A and J!");

frame.requestFocus();

} else if(flag) {

display("Column should be a number!");

frame.requestFocus();

} else if(src1.c<0 || src1.c>63) {

display("Column should be between 0-63");

frame.requestFocus();

} else {

src1.r = (rs).charAt(0) - 65;

if(src1.r>=5) {

src1.s = 1;

src1.r -= 5;

}

if((BreadBoard.getPin(src1)).getIcon()!=BreadBoard.icon) {

display("Pin already connected.");

frame.requestFocus();

} else if(isRedundant()) {

display("Redundant connection. Select another column!");

frame.requestFocus();

} else {

addPins();

frame.dispatchEvent(new WindowEvent(frame, WindowEvent.WINDOW\_CLOSING));

}

}

}

boolean isRedundant() {

Attribute ob1, ob2;

ob1 = getAttr(src);

ob2 = getAttr(src1);

return (ob1.link==ob2.link && ob1.link!=-1);

}

Attribute getAttr(Cordinate src) {

Attribute ob = null;

for(int i=0;i<5;i++) {

ob = BreadBoard.ofpin[src.s][i][src.c];

if(ob.link!=-1) {

break;

}

}

return ob;

}

void addPins() {

int l = BreadBoard.wire.size();

Attribute ob1 = getAttr(src), ob2 = getAttr(src1);

Pins p1, p2;

p1 = new Pins(src);

p2 = new Pins(src1);

p1.add(p2);

if(ob1.link==-2 && ob2.link==-2) {

display("Connection Failed! There exists different connections on both sides.");

} else if(ob1.link==-2) {

ob1.link = l;

ob1.color = BreadBoard.getColor();

addTo(ob1, p1, 0);

} else if(ob2.link==-2) {

ob2.link = l;

ob2.color = BreadBoard.getColor();

addTo(ob2, p1, 0);

} else if(ob1.link==-1 && ob2.link==-1) {

ob1 = new Attribute(l, BreadBoard.getColor());

addTo(ob1, p1, 0);

} else if(ob1.link==-1) {

addTo(ob2, p1, 1);

} else if(ob2.link==-1) {

addTo(ob1, p1, 1);

} else {

l = ob2.link;

p1.add(BreadBoard.wire.get(l));

addTo(ob1, p1, 1);

BreadBoard.wire.remove(l);

}

}

void addTo(Attribute ob, Pins p, int k) {

int l = ob.link;

p.ob = ob;

updateButton(p);

BreadBoard.ofpin[src.s][src.r][src.c] = ob;

BreadBoard.ofpin[src1.s][src1.r][src1.c] = ob;

if(k==0) {

BreadBoard.wire.add(p);

} else {

BreadBoard.wire.get(l).add(p);

}

}

void updateButton(Pins p) {

Color color = p.ob.color;

JButton btn;

while(p!=null) {

btn = BreadBoard.getPin(p.src);

btn.setBackground(color);

btn.setContentAreaFilled(false);

btn.setOpaque(true);

btn.setIcon(null);

p = p.next;

}

}

public void display(String s) {

JOptionPane.showMessageDialog(frame, s, "BREADBOARD", JOptionPane.INFORMATION\_MESSAGE, null);

}

}

*Display.java*

package tryout;

import java.awt.Dimension;

import java.awt.FlowLayout;

import java.awt.Font;

import java.awt.GridLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.MouseEvent;

import java.awt.event.MouseMotionAdapter;

import java.awt.event.WindowEvent;

import java.awt.event.WindowListener;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

public class Display extends JFrame implements ActionListener {

/\*\*

\*

\*/

private static final long serialVersionUID = 1L;

JFrame frame;

int linpt, lotpt;

JLabel lblinpt[];

JButton btninpt[];

boolean flag[];

JPanel pnlinpt[];

JPanel inpt[];

JPanel input;

JLabel lblotpt[];

JLabel tfotpt[];

JPanel pnlotpt[];

JPanel otpt[];

JPanel output;

private int Frame\_Width;

private int Frame\_Height;

private int Frame\_X;

private int Frame\_Y;

Display(int i, int o) {

linpt = i;

lotpt = o;

lblinpt = new JLabel[linpt];

btninpt = new JButton[linpt];

flag = new boolean[linpt];

inpt = new JPanel[linpt];

pnlinpt = new JPanel[linpt];

input = new JPanel();

lblotpt = new JLabel[lotpt];

tfotpt = new JLabel[lotpt];

otpt = new JPanel[lotpt];

pnlotpt = new JPanel[lotpt];

output = new JPanel();

}

String getCordinate(Cordinate src) {

char ch = (char)(src.s\*5 + src.r + 65);

String s = ch + "-" + src.c;

return s;

}

public void init() {

int wd = Math.max(linpt, lotpt);

wd = (wd<=8)?wd:(wd-1)%8+1;

int ht = ((linpt-1)/8) + ((lotpt-1)/8) + 2;

frame = new JFrame();

Frame\_Width = 175\*wd; Frame\_Height = 175\*ht + 75;

Frame\_X = 960 - Frame\_Width/2; Frame\_Y = 150;

FlowLayout fl = new FlowLayout();

fl.setVgap(25);

frame.setLayout(fl);

input.setLayout(new FlowLayout());

output.setLayout(new FlowLayout());

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

lblinpt[i] = new JLabel("INPUT at " + getCordinate(BreadBoard.inpt.get(i)) + ":");

lblinpt[i].setFont(new Font("SansSerif", 0, 20));

lblinpt[i].setHorizontalAlignment(JLabel.CENTER);

lblinpt[i].setPreferredSize(new Dimension(150, 30));

btninpt[i] = new JButton();

btninpt[i].setIcon(new ImageIcon(ClassLoader.getSystemResource("input\_off.png")));

btninpt[i].setPreferredSize(new Dimension(50, 50));

btninpt[i].addActionListener(this);

inpt[i] = new JPanel();

inpt[i].setLayout(new FlowLayout());

inpt[i].add(btninpt[i]);

pnlinpt[i] = new JPanel();

pnlinpt[i].setLayout(new GridLayout(2, 1));

pnlinpt[i].add(lblinpt[i]);

pnlinpt[i].add(inpt[i]);

frame.add(pnlinpt[i]);

}

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

lblotpt[i] = new JLabel("OUTPUT at " + getCordinate(BreadBoard.otpt.get(i)) + ":");

lblotpt[i].setFont(new Font("SansSerif", 0, 16));

lblotpt[i].setHorizontalAlignment(JLabel.CENTER);

lblotpt[i].setPreferredSize(new Dimension(150, 30));

tfotpt[i] = new JLabel();

tfotpt[i].setIcon(new ImageIcon(ClassLoader.getSystemResource("output\_off.png")));

tfotpt[i].setPreferredSize(new Dimension(100, 122));

otpt[i] = new JPanel();

otpt[i].setLayout(new FlowLayout());

otpt[i].add(tfotpt[i]);

pnlotpt[i] = new JPanel();

pnlotpt[i].setLayout(new FlowLayout());

pnlotpt[i].setPreferredSize(new Dimension(150, 175));

pnlotpt[i].add(lblotpt[i]);

pnlotpt[i].add(otpt[i]);

frame.add(pnlotpt[i]);

}

int i=0;

for(Cordinate src : BreadBoard.otpt) {

int tmp = BreadBoard.getValue(src);

if(tmp==1) {

tfotpt[i].setIcon(new ImageIcon(ClassLoader.getSystemResource("output\_on.png")));

} else {

tfotpt[i].setIcon(new ImageIcon(ClassLoader.getSystemResource("output\_off.png")));

}

i++;

}

jFrame(frame);

}

private void jFrame(final JFrame frame) {

frame.setVisible(true);

frame.setTitle("Pin Configuration");

frame.setSize(Frame\_Width, Frame\_Height);

frame.setLocation(Frame\_X, Frame\_Y);

frame.addMouseMotionListener(new MouseMotionAdapter() {

public void mouseDragged(MouseEvent e) {

System.out.println(frame.getLocationOnScreen());

System.out.println(frame.getSize());

}

});

frame.addWindowListener(new WindowListener() {

public void windowActivated(WindowEvent arg0) {

}

public void windowClosed(WindowEvent arg0) {

}

public void windowClosing(WindowEvent arg0) {

BreadBoard.board.setEnabled(true);

}

public void windowDeactivated(WindowEvent arg0) {

}

public void windowDeiconified(WindowEvent arg0) {

}

public void windowIconified(WindowEvent arg0) {

}

public void windowOpened(WindowEvent arg0) {

}

});

}

@Override

public void actionPerformed(ActionEvent arg0) {

Cordinate isrc;

JButton prsd = (JButton) arg0.getSource();

int i = 0, tmp;

for(JButton btn : btninpt) {

if(btn.equals(prsd)) {

if(!flag[i]) {

btninpt[i].setIcon(new ImageIcon(ClassLoader.getSystemResource("input\_on.png")));

isrc = BreadBoard.inpt.get(i);

BreadBoard.ofpin[isrc.s][isrc.r][isrc.c].value = 1;

flag[i] = true;

} else {

btninpt[i].setIcon(new ImageIcon(ClassLoader.getSystemResource("input\_off.png")));

isrc = BreadBoard.inpt.get(i);

BreadBoard.ofpin[isrc.s][isrc.r][isrc.c].value = 0;

flag[i] = false;

}

break;

}

i++;

}

BreadBoard.execute();

i=0;

for(Cordinate src : BreadBoard.otpt) {

tmp = BreadBoard.getValue(src);

if(tmp==1) {

tfotpt[i].setIcon(new ImageIcon(ClassLoader.getSystemResource("output\_on.png")));

} else {

tfotpt[i].setIcon(new ImageIcon(ClassLoader.getSystemResource("output\_off.png")));

}

i++;

}

}

}

*Simulator.java*

package tryout;

import java.awt.event.WindowEvent;

import javax.swing.JFrame;

public class Simulator {

static JFrame frame;

public static void main(String[] args) {

BreadBoard board = new BreadBoard();

frame = new JFrame("BreadBoard");

frame.setSize(1530, 310);

frame.setLocation(230, 180);

frame.setVisible(true);

frame.add(board);

board.init();

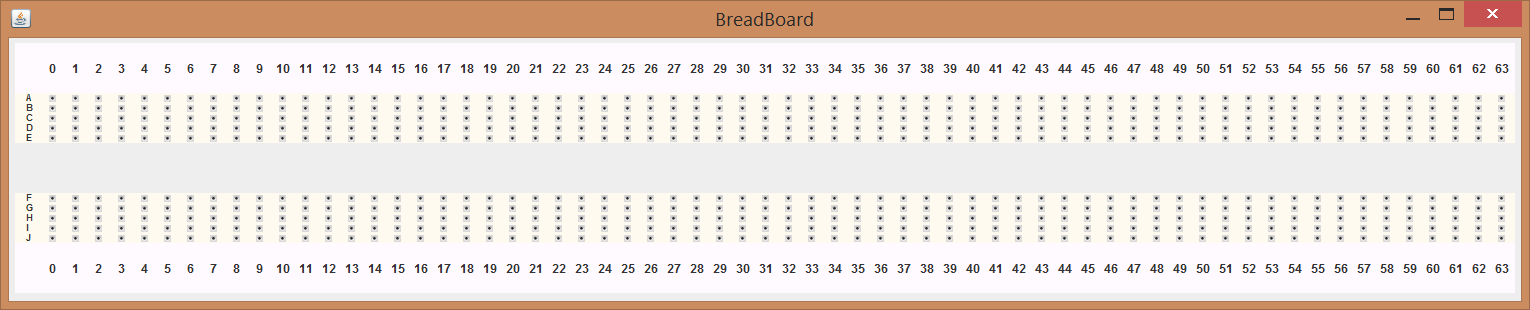
frame.dispatchEvent(new WindowEvent(frame, WindowEvent.COMPONENT\_RESIZED));

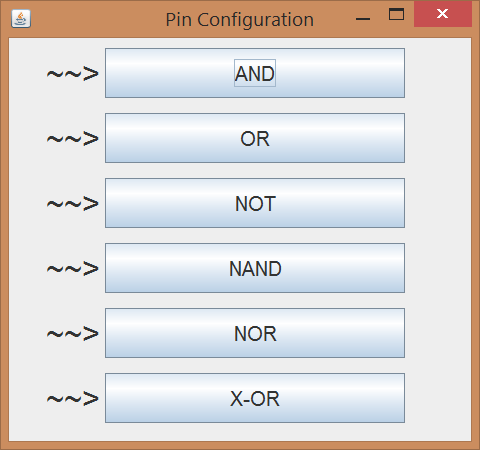
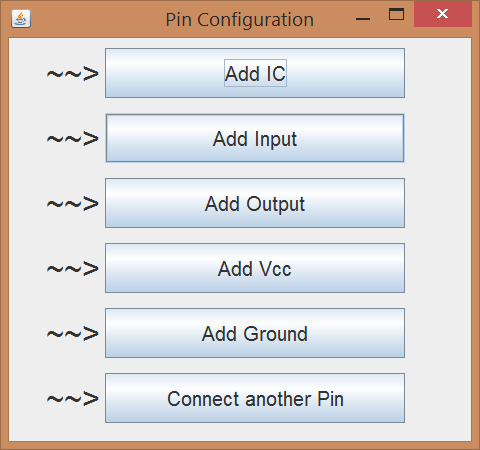
frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

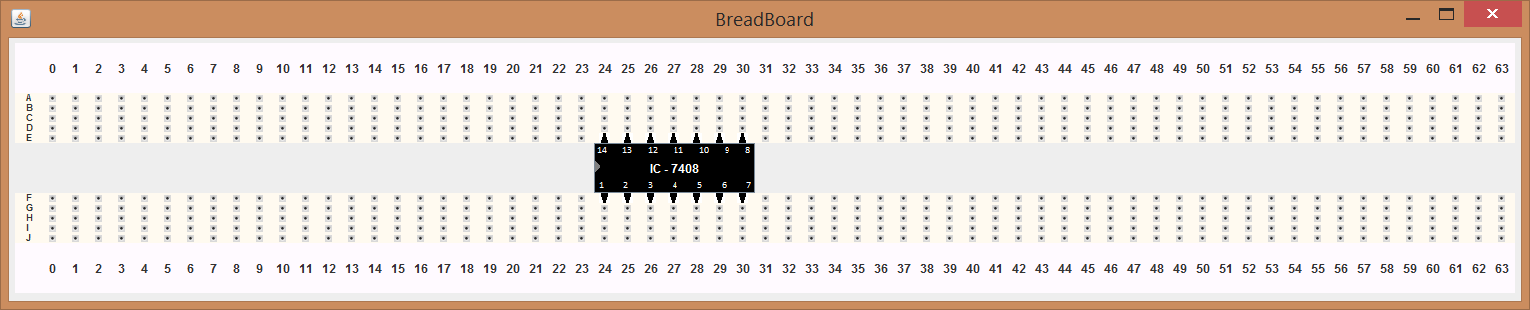
}

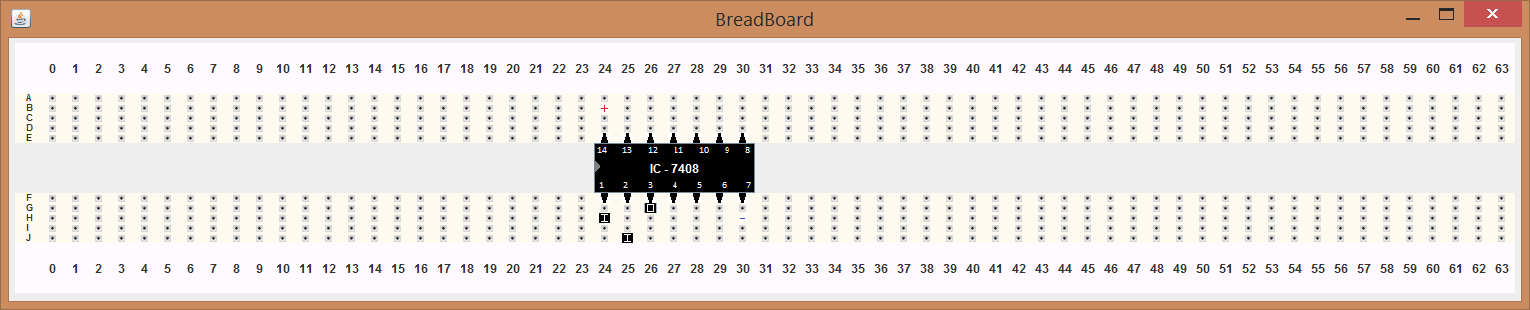
}

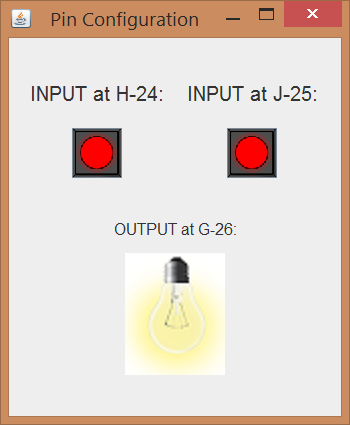
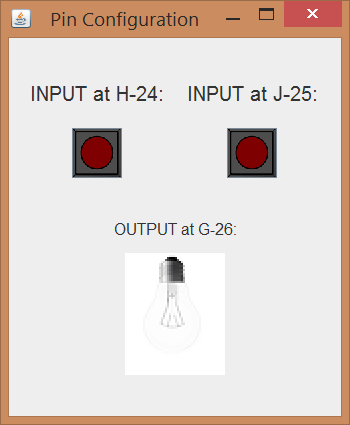
**SCREENSHOTS:**

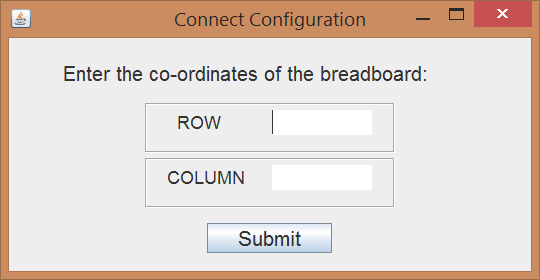
****

****

****

****

****

****

**CONCLUSION:**

Thus this program will help us in simulating a breadboard and prevent all the possible drawbacks while using a real life breadboard. This will help the student and other members to understand their connection very well.

**ACKNOWLEDGEMENT:**

We, the students of Se-Comp, DBIT would like to Thank Mayura Ma’am for giving this opportunity to create something out of our curriculum and motivating us into doing this project.