

Matrix3×3 Class

```
/*
 *
 * Gareth Evans
 *
 * Started:      23rd October 2000
 * Last Modified: 30th October 2000 (Revision 1.1)
 *
 * Group Classification for 3×3 matrices
 *
 * Matrix3x3 Class
 *
 */

public class Matrix3x3
{ // start Matrix3x3

    // Define Variables for a 3x3 Matrix

    int[][] elements;
    String name = new String();
    Matrix3x3 nextElement;

    // Constructor for creating a new 3x3 Matrix

    Matrix3x3()
    {
        elements = new int[3][3]; // 3×3 matrix
        name = null;
        nextElement = null;
    }

    // INSERT METHOD

    public void insert(Matrix3x3 list, Matrix3x3 toInsert)
    { // start insert

        if (list.nextElement == null) // Checks next element in list
        {
            list.elements = toInsert.elements; // Add matrix to list
            list.name = toInsert.name; // Add matrix's name to list
            list.nextElement = new Matrix3x3(); // Create a fresh new node
        }
        else
        {
            insert(list.nextElement, toInsert); // Calls the method again recursively
        }
    } // end insert

    // NEXT METHOD

    public Matrix3x3 nextMatrix(Matrix3x3 list) // pass in a node, want to return the next node
    {
        return list.nextElement;
    }

    // Method for working out the determinant of a 3×3 Matrix
    // We will use the following notation: A = (a b c)
    // (d e f)
    // (g h i)
    // So det(A) = a(ei-hf)-d(bi-hc)+g(bf-ec)

    public int Det(Matrix3x3 workmatrix)
    { // start Det

        int result;
        int a,b,c,d,e,f,g,h,i;

        // For simplification, assign letters to elements of the matrix
        a = workmatrix.elements[0][0];
        b = workmatrix.elements[0][1];
        c = workmatrix.elements[0][2];
        d = workmatrix.elements[1][0];
        e = workmatrix.elements[1][1];
        f = workmatrix.elements[1][2];
        g = workmatrix.elements[2][0];
        h = workmatrix.elements[2][1];
        i = workmatrix.elements[2][2];

        // Work out the determinant and return the result
        result = a*((e*i)-(h*f)) - d*((b*i)-(h*c)) + g*((b*f)-(e*c));
        return result;
    } // end Det
}
```

```

// Method for comparing two matrices to see if they are the same

public boolean Same(Matrix3x3 one, Matrix3x3 two)
{ // start Same

    // Assume matrices are the same to begin with
    boolean result = true;

    // Now compare each individual element, returning false if
    // any two elements differ
    for(int i=0; i<=2; i++)
    {
        for(int j=0; j<=2; j++)
        {
            if((one.elements[i][j])!=(two.elements[i][j]))
            {
                result = false;
            } // end if
        } // end for(j)
    } // end for(i)
    return result;
} // end Same

// Method for multiplying to matrices

public Matrix3x3 Multiply(Matrix3x3 one, Matrix3x3 two)
{ // start Multiply

    // Define Variables
    Matrix3x3 result = new Matrix3x3();
    int i,j,k,sum;

    for(i=0; i<=2; i++)
    {
        for(j=0; j<=2; j++)
        {
            sum = 0;
            for(k=0; k<=2; k++)
            {
                sum = sum + (one.elements[i][k])*(two.elements[k][j]);
            }
            result.elements[i][j] = sum;
        }
    }

    result.name = (one.name + two.name);

    return result;
} // end Multiply
} // end Matrix3x3

```

GroupApplet class

```
/*
 *
 * Gareth Evans
 *
 * Started:      23rd October 2000
 * Last Modified: 6th November 2000 (Revision 1.6)
 *
 * Group Classification for 3x3 matrices
 *
 * Main Class (Applet)
 *
 */

import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class GroupApplet extends Applet { // begin GroupApplet

    // My Variables
    String messageStore = new String(); // for "Status Bar" use
    static int memlimit = 500; // to trap infinite groups

    // Variables for Master Matrix List
    static Matrix3x3 MatrixList = new Matrix3x3(); // Holds group matrices
    static int count = 1; // Keeps track of the size of MatrixList

    // Variables for use in the display section
    static Matrix3x3 DisplayList = new Matrix3x3(); // Holds display matrices
    static int displaycount = 1; // keeps track of DisplayList size
    static int displayposition = 1; // keeps track of which matrix is shown now
    static String displayString = new String(); // used for output status bar

    // Variables for use with button click handling
    static int customfunction = -1;
    static boolean buttonpressed = false;

    // JBuilder Variables
    boolean isStandalone = false;

    // Various Layouts for "Look" of Applet
    BorderLayout MasterBorderLayout = new BorderLayout();
    GridLayout IOGridLayout = new GridLayout();
    BorderLayout InputBorderLayout = new BorderLayout();
    GridLayout inputGrid = new GridLayout();
    BorderLayout outputBorderLayout = new BorderLayout();
    BorderLayout outputSubBorderLayout = new BorderLayout();
    GridLayout outputMatGrid = new GridLayout();

    // Applet Panels
    Panel IOPanel = new Panel();
    Panel inputPanel = new Panel();
    Panel outputPanel = new Panel();
    Panel inputSub = new Panel();
    Panel outputSub = new Panel();
    Panel outputMatBase = new Panel();
    Panel belowIdentifier = new Panel();

    // TextFields for providing output and obtaining input
    TextField Message = new TextField();
    static TextField inputMessage = new TextField();
    static TextField outputMessage = new TextField();
    static TextField OA = new TextField();
    static TextField OB = new TextField();
    static TextField OC = new TextField();
    static TextField OD = new TextField();
    static TextField OE = new TextField();
    static TextField OF = new TextField();
    static TextField OG = new TextField();
    static TextField OH = new TextField();
    static TextField OI = new TextField();
    static TextField Identifier = new TextField();
    static TextField IA = new TextField();
    static TextField IB = new TextField();
    static TextField IC = new TextField();
    static TextField ID = new TextField();
    static TextField IE = new TextField();
    static TextField IF = new TextField();
    static TextField IG = new TextField();
    static TextField IH = new TextField();
    static TextField II = new TextField();
}
```

```

// Buttons for use input
static Button Add = new Button();
static Button Display = new Button();
static Button CalcGroup = new Button();
static Button CalcSubgroup = new Button();
static Button CalcElement = new Button();
static Button Next = new Button();
static Button Custom = new Button();

// Get a parameter value
public String getParameter(String key, String def) {
    return isStandalone ? System.getProperty(key, def) :
        (getParameter(key) != null ? getParameter(key) : def);
}

// Construct the applet
public GroupApplet() {
}

// Initialize the applet
public void init() { // Start Init
    insertIdentity(MatrixList); // Insert Identity Matrix into List

    // set certain buttons disabled to start with
    CalcGroup.setEnabled(false);
    CalcSubgroup.setEnabled(false);
    CalcElement.setEnabled(false);
    Next.setEnabled(false);
    Custom.setEnabled(false);

    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }

    try
    {
        Display(MatrixList, "Matrix", count);
    }
    catch (Exception exp) { }
} // End Init

// GET INPUT METHOD
public static Matrix3x3 GetInput() throws Exception
{ // Start GetInput
    Matrix3x3 result = new Matrix3x3();
    result.elements[0][0] = Integer.parseInt(IA.getText());
    result.elements[1][0] = Integer.parseInt(IB.getText());
    result.elements[2][0] = Integer.parseInt(IC.getText());
    result.elements[0][1] = Integer.parseInt(ID.getText());
    result.elements[1][1] = Integer.parseInt(IE.getText());
    result.elements[2][1] = Integer.parseInt(IF.getText());
    result.elements[0][2] = Integer.parseInt(IG.getText());
    result.elements[1][2] = Integer.parseInt(IH.getText());
    result.elements[2][2] = Integer.parseInt(II.getText());

    result.name = Identifier.getText();

    return result;
} // End GetInput

// DISPLAY (SINGLE) MATRIX METHOD
public static void DisplayMatrix(Matrix3x3 single, int position, int total, String descriptor) throws
Exception
{ // Start DisplayMatrix

    // Initialise Display
    outputMessage.setText(descriptor + " " + position + " of " + total + ": " + single.name);
    if (position==total)
    {
        Next.setEnabled(false);
    }
    else
    {
        Next.setEnabled(true);
    }
}

```

```

// Transfer Data to Screen
OA.setText("" + single.elements[0][0]);
OB.setText("" + single.elements[1][0]);
OC.setText("" + single.elements[2][0]);
OD.setText("" + single.elements[0][1]);
OE.setText("" + single.elements[1][1]);
OF.setText("" + single.elements[2][1]);
OG.setText("" + single.elements[0][2]);
OH.setText("" + single.elements[1][2]);
OI.setText("" + single.elements[2][2]);
} // End DisplayMatrix

// DISPLAY MATRIX METHOD
public static void Display(Matrix3x3 disp, String descriptor, int size) throws Exception

// 3 parametres: (1) the 3x3 matrix to display on screen
//              (2) descriptor to describe what is shown e.g. generators, groups,...
//              (3) an integer to denote how many matrices to display

{ // Start Display

    if (size <=1)
    {
        Next.setEnabled(false);
    }
    else
    {
        Next.setEnabled(true);
    }

    DisplayList = disp;
    displaycount = size;
    displayString = descriptor;
    displayposition = 1;
    DisplayMatrix(DisplayList, 1, size, descriptor);
} // End Display

// METHOD TO CHECK WHETHER A MATRIX EXISTS IN A MATRIX LIST ALREADY
public static boolean AlreadyInList(Matrix3x3 check, Matrix3x3 fulllist, int loopduration)
{ // Start AlreadyInList
    // CHECK NOT ALREADY IN LIST
    boolean already = false; // assume new matrix not in list
    Matrix3x3 temp = new Matrix3x3();
    temp = fulllist;

    for(int i=1; i<=loopduration; i++)
    {
        if ((temp.Same(temp, check))==true) // invokes method in Matrix3x3 class
        {
            already = true;
        }
        temp = temp.nextMatrix(temp);
    }
    return already;
} // End AlreadyInList

// METHOD TO ADD A (VALID) MATRIX TO A MATRIX LIST
public static void AddMatrix() throws Exception
{ // Start Add Matrix
    Matrix3x3 check = GetInput();

    // CHECK DETERMINANT
    if(((check.Det(check))==1)|((check.Det(check))== -1))
    {
        // CHECK NOT ALREADY IN LIST
        boolean test = AlreadyInList(check, MatrixList, count);

        if (test == true)
        {
            inputMessage.setText(Identifier.getText() + " Already in the List. Try Again.");
        }
        else // Add to list
        {
            inputMessage.setText("Det(" + Identifier.getText() + ") = " + check.Det(check));
            MatrixList.insert(MatrixList, check);
            count++;
            unDim();
            Display(MatrixList, "Matrix", count); // update Display
        }
    }
    else
    {
        // Display Error Message
        inputMessage.setText("Invalid Determinant!");
    }
} // End AddMatrix

```

```

// METHOD TO CALCULATE THE GROUP OF ANY GIVEN SET OF MATRICES
public static void CalcGroup(Matrix3x3 Manip, String descriptor, int calccount, boolean incrementOK) throws
Exception
{ // Start CalcGroup

// To calculate the group, calculate all products of
// matrices in the list so far and see if they already exist.

Matrix3x3 testi = new Matrix3x3();
Matrix3x3 testj = new Matrix3x3();
Matrix3x3 testixj = new Matrix3x3();
testi = Manip;
testj = Manip;
boolean nomore = false;
int added = 0;

while (nomore == false) // Execute until whole group is closed under multiplication
{ // start while
    nomore = true; // assume no more to add

    for(int i=1; i<=calccount; i++)
    { // start for(i)

        for(int j=1; j<=calccount; j++)
        { // start for(j)

            testixj = testi.Multiply(testi, testj);
            boolean result = AlreadyInList(testixj, Manip, calccount);

            if (result == false) // i.e. not in list already
            { // start if
                Manip.insert(Manip, testixj);
                added++;
                inputMessage.setText("Matrices Added = " + added);
                nomore = false;

                if (calccount <= memlimit)
                {
                    calccount++;
                }
                else // i.e. dealing with a possible infinite group
                {
                    // exit from loops
                    inputMessage.setText("MEMORY ERROR!");
                    j = calccount;
                    i = calccount;
                    nomore = true;
                }
                if(incrementOK==true) {count++;}

                // NOTE: incrementOK is used to denote whether we are calculating a group
                // from generators (true, increment count) OR calculating e.g.
                // subgroups. (false, do not increment count)

            } // end if
            testj = testj.nextMatrix(testj);
        } // end for(j)
        testi = testi.nextMatrix(testi);
        testj = Manip; // reset
    } // end for(i)
    testi = Manip; // reset
} // end while

if (incrementOK==true)
{
    MatrixList = Manip;
    Display(MatrixList, descriptor, calccount);
}
else
{
    Display(Manip, descriptor, calccount);
}
} // end CalcGroup

// METHOD TO DISABLE OTHER BUTTONS WHILE WAITING FOR CUSTOM TO BE PRESSED
public static void Dim()
{ // start Dim

    Add.setEnabled(false);
    Display.setEnabled(false);
    CalcGroup.setEnabled(false);
    CalcSubgroup.setEnabled(false);
    CalcElement.setEnabled(false);

} // end Dim

```

```

// METHOD TO ENABLE OTHER BUTTONS
public static void unDim()
{ // start unDim

    Add.setEnabled(true);
    Display.setEnabled(true);
    CalcGroup.setEnabled(true);
    CalcSubgroup.setEnabled(true);
    CalcElement.setEnabled(true);

} // end unDim

// METHOD TO CALCULATE SUBGROUP GENERATED BY AN ELEMENT OF A GROUP
public static void Element() throws Exception
{ // start Element
    // Generates all subgroups generated by an element of MatrixList

    Matrix3x3 generator = new Matrix3x3();
    Matrix3x3 carrier = new Matrix3x3();
    Matrix3x3 swap = new Matrix3x3();
    carrier = MatrixList;
    int generatorsize;

    // Disable all other buttons, enable Custom
    Dim();
    Custom.setEnabled(true);

    for (int i=1; i<=count; i++)
    { // start for(i)
        generator.nextElement = null; // reset matrix
        swap.elements = carrier.elements;
        swap.name = carrier.name;
        generatorsize = 1;

        insertIdentity(generator); // Insert Identity Matrix

        if (i>1) // i.e. Generator is not the Identity matrix
        {
            generator.insert(generator, swap);
            generatorsize = 2;
            DisplayMatrix(generator.nextMatrix(generator), i, count, "Generator");
        }
        else
        {
            DisplayMatrix(generator, i, count, "Generator");
        }

        // LISTEN FOR BUTTON PRESS - "Calculate Subgroup"
        Custom.setLabel("Calculate Subgroup");
        buttonpressed = false;
        Next.setEnabled(false);

        //while (buttonpressed == false)
        //{
        for(int delay = 0; delay <=3000; delay++)
        {System.out.println(delay);}
        //}

        Next.setEnabled(true);
        CalcGroup(generator, "Subgroup", generatorsize, false);
        carrier = carrier.nextMatrix(carrier); // cycle

        //LISTEN FOR BUTTON PRESS - "Next Subgroup"
        Custom.setLabel("Next Subgroup");
        buttonpressed = false;
        //while(buttonpressed == false)
        //{
        for(int delay = 0; delay <=5000; delay++)
        {System.out.println(delay);}
        //}

    } // end for(i)

    // Reset Custom's Label and Enable all other buttons again
    Custom.setLabel("");
    Custom.setEnabled(false);
    unDim();
} // end Element

```

```

// METHOD TO CALCULATE THE SUBGROUPS OF A GROUP
public static void Subgroup()
{ // start Subgroup
    for (int size = 1; size <=count; size++)
    {
        // TO BE COMPLETED
    }
    inputMessage.setText("TO BE DONE");
} // end Subgroup

// METHOD TO CALCULATE THE HASSER DIAGRAM OF A SUBGROUP STRUCTURE
public static void Hasser()
{ // start Hasser
    inputMessage.setText("TO BE DONE");
} // end Hasser

// METHOD TO INSERT AN IDENTITY MATRIX INTO ANY MATRIX LIST
public static void insertIdentity(Matrix3x3 input)
{ // start insertIdentity
    int[][] id_el;
    id_el = new int[3][3];
    id_el[0][0] = 1; id_el[0][1] = 0; id_el[0][2] = 0;
    id_el[1][0] = 0; id_el[1][1] = 1; id_el[1][2] = 0;
    id_el[2][0] = 0; id_el[2][1] = 0; id_el[2][2] = 1;

    String id_str = new String();
    id_str = "I";

    Matrix3x3 id_m = new Matrix3x3();
    id_m.elements = id_el;
    id_m.name = id_str;
    input.insert(input, id_m);
} // end InsertIdentity

// Component initialization
private void jbInit() throws Exception { // start jbInit

    // begin by setting Master Layout
    this.setLayout(MasterBorderLayout);

    // set Main Layouts & Panels
    MasterBorderLayout.setVgap(2);
    IOPanel.setBackground(Color.lightGray);
    IOPanel.setLayout(IOGridLayout);
    IOGridLayout.setColumns(2);

    // Initialise Input Layouts & Panels
    inputPanel.setBackground(Color.lightGray);
    inputPanel.setLayout(InputBorderLayout);

    // Initialise Output Layouts & Panels
    outputPanel.setBackground(Color.lightGray);
    outputPanel.setLayout(outputBorderLayout);
    outputSub.setLayout(outputSubBorderLayout);

    // Initialise Main Status Bar
    Message.setBackground(Color.darkGray);
    Message.setFont(new java.awt.Font("Serif", 1, 12));
    Message.setForeground(Color.white);
    Message.setEditable(false);

    // Initialise Input Area
    inputSub.setBackground(Color.lightGray);
    inputSub.setLayout(inputGrid);
    inputGrid.setColumns(4);
    inputGrid.setHgap(2);
    inputGrid.setRows(4);
    inputGrid.setVgap(2);

    // Initialise Output Area
    outputMatBase.setBackground(Color.lightGray);
    outputMatBase.setLayout(outputMatGrid);
    outputMatGrid.setColumns(3);
    outputMatGrid.setHgap(2);
    outputMatGrid.setRows(3);
    outputMatGrid.setVgap(2);

    // Initialise Input Status Bar
    inputMessage.setBackground(Color.gray);
    inputMessage.setForeground(Color.white);
    inputMessage.setEditable(false);

    // Initialise Output Status Bar
    outputMessage.setBackground(Color.gray);
    outputMessage.setForeground(Color.white);
    outputMessage.setEditable(false);
}

```

```

// Initialise Input Display Area
IA.setBackground(Color.yellow);
IA.setText("0");
IB.setBackground(Color.yellow);
IB.setText("0");
IC.setBackground(Color.yellow);
IC.setText("0");
ID.setBackground(Color.cyan);
ID.setText("0");
IE.setBackground(Color.cyan);
IE.setText("0");
IF.setBackground(Color.cyan);
IF.setText("0");
IG.setBackground(Color.green);
IG.setText("0");
IH.setBackground(Color.green);
IH.setText("0");
II.setBackground(Color.green);
II.setText("0");

// Initialise Output Display Area
OA.setBackground(Color.yellow);
OA.setEditable(false);
OA.setText("0");
OB.setBackground(Color.yellow);
OB.setEditable(false);
OB.setText("0");
OC.setBackground(Color.yellow);
OC.setEditable(false);
OC.setText("0");
OD.setBackground(Color.cyan);
OD.setEditable(false);
OD.setText("0");
OE.setBackground(Color.cyan);
OE.setEditable(false);
OE.setText("0");
OF.setBackground(Color.cyan);
OF.setEditable(false);
OF.setText("0");
OG.setBackground(Color.green);
OG.setEditable(false);
OG.setText("0");
OH.setBackground(Color.green);
OH.setEditable(false);
OH.setText("0");
OI.setBackground(Color.green);
OI.setEditable(false);
OI.setText("0");

// ADD ELEMENTS TO APPLET
this.add(IOPanel, BorderLayout.CENTER);
    IOPanel.add(inputPanel, null);
        inputPanel.add(inputSub, BorderLayout.CENTER);
            inputSub.add(IA, null);
            inputSub.add(ID, null);
            inputSub.add(IG, null);
            inputSub.add(Add, null);
            inputSub.add(IB, null);
            inputSub.add(IE, null);
            inputSub.add(IH, null);
            inputSub.add(Identifier, null);
            inputSub.add(IC, null);
            inputSub.add(IF, null);
            inputSub.add(II, null);
            inputSub.add(belowIdentifier, null);
            inputSub.add(Display, null);
            inputSub.add(CalcGroup, null);
            inputSub.add(CalcSubgroup, null);
            inputSub.add(CalcElement, null);
        inputPanel.add(inputMessage, BorderLayout.SOUTH);
    IOPanel.add(outputPanel, null);
        outputPanel.add(outputMessage, BorderLayout.SOUTH);
        outputPanel.add(outputSub, BorderLayout.CENTER);
            outputSub.add(Next, BorderLayout.SOUTH);
            outputSub.add(Custom, BorderLayout.NORTH);
            outputSub.add(outputMatBase, BorderLayout.CENTER);
                outputMatBase.add(OA, null);
                outputMatBase.add(OD, null);
                outputMatBase.add(OG, null);
                outputMatBase.add(OB, null);
                outputMatBase.add(OE, null);
                outputMatBase.add(OH, null);
                outputMatBase.add(OC, null);
                outputMatBase.add(OF, null);
                outputMatBase.add(OI, null);
    this.add(Message, BorderLayout.SOUTH);

```

```

// EVENT CODE

// ADD BUTTON EVENTS
Add.setFont(new java.awt.Font("Dialog", 1, 12));
Add.setLabel("+");
Add.addMouseListener(new java.awt.event.MouseAdapter() { // start addMouseListener

    public void mouseEntered(MouseEvent e) {
        Add_mouseEntered(e);
    }

    public void mouseExited(MouseEvent e) {
        Add_mouseExited(e);
    }

    public void mouseClicked(MouseEvent e) {
        Add_mouseClicked(e);
    }
}); // end addMouseListener for Add

// IDENTIFIER BUTTON EVENTS
Identifier.setBackground(Color.darkGray);
Identifier.setFont(new java.awt.Font("Dialog", 1, 12));
Identifier.setForeground(Color.white);
Identifier.setText("A");
Identifier.addMouseListener(new java.awt.event.MouseAdapter() { // start addMouseListener

    public void mouseEntered(MouseEvent e) {
        Identifier_mouseEntered(e);
    }

    public void mouseExited(MouseEvent e) {
        Identifier_mouseExited(e);
    }
}); // end addMouseListener for Identifier

// DISPLAY BUTTON EVENTS
Display.setFont(new java.awt.Font("Dialog", 1, 12));
Display.setLabel("D");
Display.addMouseListener(new java.awt.event.MouseAdapter() { // start addMouseListener

    public void mouseEntered(MouseEvent e) {
        Display_mouseEntered(e);
    }

    public void mouseExited(MouseEvent e) {
        Display_mouseExited(e);
    }

    public void mouseClicked(MouseEvent e) {
        try
        {
            Display_mouseClicked(e);
        }
        catch (Exception exp) { }
    }
}); // end addMouseListener for Display

// CALCULATE GROUP EVENTS
CalcGroup.setFont(new java.awt.Font("Dialog", 1, 12));
CalcGroup.setLabel("G");
CalcGroup.addMouseListener(new java.awt.event.MouseAdapter() { // start addMouseListener

    public void mouseClicked(MouseEvent e) {
        CalcGroup_mouseClicked(e);
    }
}); // end addMouseListener (1) for CalcGroup

CalcGroup.addMouseListener(new java.awt.event.MouseAdapter() { // start addMouseListener

    public void mouseExited(MouseEvent e) {
        CalcGroup_mouseExited(e);
    }

    public void mouseEntered(MouseEvent e) {
        CalcGroup_mouseEntered(e);
    }
}); // end addMouseListener (2) for CalcGroup

```

```

// CALCULATE SUBGROUP EVENTS
CalcSubgroup.setFont(new java.awt.Font("Dialog", 1, 12));
CalcSubgroup.setLabel("S");
CalcSubgroup.addMouseListener(new java.awt.event.MouseAdapter() { // start addMouse Listener

    public void mouseExited(MouseEvent e) {
        CalcSubgroup_mouseExited(e);
    }

    public void mouseEntered(MouseEvent e) {
        CalcSubgroup_mouseEntered(e);
    }

    public void mouseClicked(MouseEvent e) {
        CalcSubgroup_mouseClicked(e);
    }
}); // end addMouseListener for CalcSubgroup

// CALCULATE ELEMENT GENERATED SUBGROUPS EVENTS
CalcElement.setFont(new java.awt.Font("Dialog", 1, 12));
CalcElement.setLabel("E");
CalcElement.addMouseListener(new java.awt.event.MouseAdapter() { // start addMouseListener

    public void mouseExited(MouseEvent e) {
        CalcElement_mouseExited(e);
    }

    public void mouseEntered(MouseEvent e) {
        CalcElement_mouseEntered(e);
    }

    public void mouseClicked(MouseEvent e) {
        CalcElement_mouseClicked(e);
    }
}); // end addMouseListener for CalcElement

// NEXT BUTTON EVENTS
Next.setBackground(Color.lightGray);
Next.setActionCommand("Down");
Next.setLabel("Next");
Next.addMouseListener(new java.awt.event.MouseAdapter() { // start addMouseListener

    public void mouseExited(MouseEvent e) {
        Next_mouseExited(e);
    }

    public void mouseEntered(MouseEvent e) {
        Next_mouseEntered(e);
    }

    public void mouseClicked(MouseEvent e) {
        Next_mouseClicked(e);
    }
}); // end addMouseListener for Next

// CUSTOM BUTTON EVENTS
Custom.setBackground(Color.lightGray);
Custom.setActionCommand("Up");
Custom.addMouseListener(new java.awt.event.MouseAdapter() { // start addMouseListener

    public void mouseExited(MouseEvent e) {
        Custom_mouseExited(e);
    }

    public void mouseEntered(MouseEvent e) {
        Custom_mouseEntered(e);
    }

    public void mouseClicked(MouseEvent e) {
        Custom_mouseClicked(e);
    }
}); // end addMouseListener for Custom

// END OF EVENTS
} // end jbInit

// Get Applet information
public String getAppletInfo() {
    return "Applet Information";
}

// Get parameter info
public String[][] getParameterInfo() {
    return null;
}

```

```

//
// ADD BUTTON
//

// MOUSE ENTERS ADD BUTTON
void Add_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText("Add Matrix to Basis Vectors");
}

// MOUSE EXITS ADD BUTTON
void Add_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

// MOUSE CLICKS ADD BUTTON
void Add_mouseClicked(MouseEvent e) {

// ADD MATRIX TO LIST OF GENERATORS
try
{
    AddMatrix();
}
catch (Exception exp) { }
}

//
// IDENTIFIER TEXT FIELD
//

// MOUSE ENTERS TEXT FIELD
void Identifier_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText("Matrix Identifier");
}

// MOUSE EXITS TEXT FIELD
void Identifier_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

//
// DISPLAY BUTTON
//

// MOUSE ENTERS DISPLAY BUTTON
void Display_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText("Display Current Matrix List");
}

// MOUSE EXITS DISPLAY BUTTON
void Display_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

// MOUSE CLICKS DISPLAY BUTTON
void Display_mouseClicked(MouseEvent e) throws Exception {
Next.setEnabled(true);
// DISPLAY LIST OF MATRICES
Display(MatrixList, "Matrix", count);
}

//
// CALCULATE GROUP BUTTON
//

// MOUSE ENTERS CALCULATE GROUP BUTTON
void CalcGroup_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText("Calculate Group");
}

// MOUSE EXITS CALCULATE GROUP BUTTON
void CalcGroup_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

```

```

// MOUSE CLICKS CALCULATE GROUP BUTTON
void CalcGroup_mouseClicked(MouseEvent e) {

Next.setEnabled(true);
// CALCULATE GROUP
try
{
    CalcGroup(MatrixList, "Group", count, true);
}
catch (Exception exp) { }
}

//
// CALCULATE SUBGROUPS BUTTON
//

// MOUSE ENTERS CALCULATE SUBGROUP BUTTON
void CalcSubgroup_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText("Calculate Subgroups");
}

// MOUSE EXITS CALCULATE SUBGROUP BUTTON
void CalcSubgroup_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

// MOUSE CLICKS CALCULATE SUBGROUP BUTTON
void CalcSubgroup_mouseClicked(MouseEvent e) {

// CALCULATE SUBGROUP
try
{
    Subgroup();
}
catch (Exception exp) { }
}

//
// CALCULATE ELEMENT GENERATED SUBGROUPS BUTTON
//

// MOUSE ENTERS ELEMENT BUTTON
void CalcElement_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText("Element Generated Subgroups");
}

// MOUSE EXITS ELEMENT BUTTON
void CalcElement_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

// MOUSE CLICKS ELEMENT BUTTON
void CalcElement_mouseClicked(MouseEvent e) {

// CALCULATE SUBGROUPS GENERATED BY AN ELEMENT OF THE GROUP
try
{
    Element();
}
catch (Exception exp) { }
}

//
// CUSTOM BUTTON
//

// MOUSE ENTERS CUSTOM BUTTON
void Custom_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText(Custom.getLabel());
}

// MOUSE EXITS CUSTOM BUTTON
void Custom_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

// MOUSE CLICKS CUSTOM BUTTON
void Custom_mouseClicked(MouseEvent e) {

// Indicate Button has been pressed
buttonpressed = true;
}

```

```
//  
// NEXT BUTTON  
//  
// MOUSE ENTERS NEXT BUTTON  
void Next_mouseEntered(MouseEvent e) {  
messageStore = Message.getText();  
Message.setText("Next Matrix");  
}  
  
// MOUSE EXITS NEXT BUTTON  
void Next_mouseExited(MouseEvent e) {  
Message.setText(messageStore);  
}  
  
// MOUSE CLICKS NEXT BUTTON  
void Next_mouseClicked(MouseEvent e) {  
try  
{  
// SHOW NEXT MATRIX  
if (displayposition < displaycount)  
{  
DisplayList = DisplayList.nextMatrix(DisplayList);  
displayposition++;  
DisplayMatrix(DisplayList, displayposition, displaycount, displayString);  
}  
else  
{  
Next.setEnabled(false);  
}  
}  
catch (Exception exp) { }  
} // End GroupApplet
```