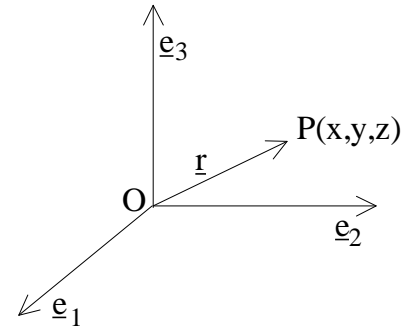


## Project Progress Report: Week 5

Consider any point in space represented with respect to an orthonormal basis with three vectors,  $\{\underline{e}_1, \underline{e}_2, \underline{e}_3\}$ . For *any two* basis vectors,  $\underline{e}_i \cdot \underline{e}_j = \delta_{ij}$ . In the diagram, we have

$$\underline{r} = \overrightarrow{OP} = x\underline{e}_1 + y\underline{e}_2 + z\underline{e}_3.$$



Consider that we want to change the representation of our vector  $\underline{r}$  from one basis to another. For example, consider that the diagram shown represents the usual basis we take ( $\underline{i}$ ,  $\underline{j}$  and  $\underline{k}$ ). We may represent this basis by the *following* matrix, where the **columns** represent  $\underline{e}_1$ ,  $\underline{e}_2$  and  $\underline{e}_3$ :

$$\underline{A} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Keeping the *origin* where it is, consider that we reflect the  $\underline{e}_1$  and  $\underline{e}_2$  axes so we obtain the following matrix:

$$\underline{B} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The general idea is that we use a  $3 \times 3$  matrix to represent any basis which may be obtained by rotation or reflection from the normal basis  $\underline{A}$ . Let these matrices be generators, and consider that we are given an *arbitrary number* of generators. The task is to generate a matrix group from a given set of generators. This should be implemented in the Java programming language.

When the program starts, it should ask the user to provide a set of generators. The program should check to see whether these generators are valid, and this is done by calculating the determinant and checking to see whether it is  $\pm 1$ . Assume that if you multiply any two matrices already in the group and obtain a matrix not already in the group, that matrix is a valid basis and may be added to the matrix group.

The program should calculate the matrix group of any given set of generators and return the **order** of the matrix group. In terms of calculation, when the program has found the matrix group, then it should be closed under multiplication, meaning that the product of any two matrices in the group should already be in the group.

Implementing this problem, the program will depend heavily on the construction of a class to represent a  $3 \times 3$  matrix. This class should have methods to find the determinant of such a matrix, the product of two such matrices and a method to see whether two such matrices are the same.

## Source Code: DOS Program

```

/*
 *
 * Gareth Evans
 *
 * Started:      23rd October 2000
 * Last Modified: 30th October 2000 (Revision 1.1)
 *
 * Group Classification for 3x3 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]; // 3x3 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 3x3 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 deteminant 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;
        }
    }

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

```

```

/*
 *
 * Gareth Evans
 *
 * Started:      23rd October 2000
 * Last Modified: 30th October 2000 (Revision 1.2c)
 *
 * Group Classification for 3x3 matrices
 *
 * Main Class
 *
 */

import java.bangor.*; // for the BasicIo class

public class groupcalc
{ // start groupcalc

    // GLOBAL VARIABLES
    static Matrix3x3 MatrixList = new Matrix3x3(); // Holds group matrices
    static int count = 1; // Keeps track of the size of MatrixList

    // GET INTEGER METHOD - gets an integer from the end user
    public static int GetInt() throws Exception
    { // start GetInt

        int Choice = 0; // returns 0 if error or no input

        try
        {
            Choice = BasicIo.readInteger();
        }
        catch (Exception exp)
        {
            Choice = GetInt();
        }
        return Choice;
    } // end GetInt

    // GET INPUT METHOD
    public static Matrix3x3 GetInput() throws Exception
    { // start GetInput

        Matrix3x3 result = new Matrix3x3();

        System.out.println("Please enter the elements of your 3x3 matrix");
        System.out.println();

        for(int i=0; i<=2; i++)
        {
            for(int j=0; j<=2; j++)
            {
                System.out.print("Row " + (j+1) + ", Column " + (i+1) + ": ");
                result.elements[j][i] = GetInt();
                // Enter numbers by column
            }
        }
        System.out.println();
        System.out.print("Please enter a descriptor for your matrix e.g. A, B, P, Q,...");
        try
        {
            result.name = BasicIo.readString();
        }
        catch (Exception exp) { }

        return result;
    } // end GetInput

```

```

// DISPLAY (SINGLE) MATRIX METHOD
public static void DisplayMatrix(Matrix3x3 single) throws Exception
{ // start DisplayMatrix

    System.out.println("Matrix Identifier: " + single.name);
    System.out.println();
    System.out.println(single.elements[0][0] + "          " + single.elements[0][1] + "
        " + single.elements[0][2]);
    System.out.println(single.elements[1][0] + "          " + single.elements[1][1] + "
        " + single.elements[1][2]);
    System.out.println(single.elements[2][0] + "          " + single.elements[2][1] + "
        " + single.elements[2][2]);

} // end DisplayMatrix

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

    System.out.println();
    System.out.println("Number of matrices in the " + descriptor + ": " + dispcount);

    Matrix3x3 temp = new Matrix3x3();
    temp = disp;
    int scrollstop = 0; // scrollstop used to control large output as in dir/p in DOS

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

        scrollstop++;
        System.out.println();
        System.out.print("List Number: " + i + ". ");
        DisplayMatrix(temp);
        temp = temp.nextMatrix(temp);

        if(scrollstop==3)
        { // start if

            System.out.println();
            System.out.print("Press Enter to continue....");
            String sponge = BasicIo.readString();
            scrollstop = 0;
        } // end if
    } // end for(i)

    System.out.println();
    System.out.println("Number of matrices in the " + descriptor + ": " + dispcount);

} // 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++)
    { // start for(i)

        if ((temp.Same(temp, check))==true) // invokes method in Matrix3x3 class
        { // start if

            already = true;

        } // end if
        temp = temp.nextMatrix(temp);
    } // end for(i)
    return already;
} // end AlreadyInList

```

```

// METHOD TO ADD A (VALID) MATRIX TO A MATRIX LIST
public static void AddMatrix() throws Exception
{ // start AddMatrix

    Matrix3x3 check = GetInput();
    System.out.println();
    System.out.println("The Determinant of the matrix is " + check.Det(check));

    // CHECK DETERMINANT
    if(((check.Det(check))==1)|((check.Det(check))== -1))
    { // start if

        // CHECK NOT ALREADY IN LIST
        boolean test = AlreadyInList(check, MatrixList, count);

        if (test == true)
        {
            System.out.println("Your Matrix is already in the list. Please try again.");
        }
        else // Add to list
        {
            System.out.println("Now adding the matrix to the list.");
            MatrixList.insert(MatrixList, check);
            count++;
        }
    } // end if
    else
    {
        // Display Error Message
        System.out.println("You have entered an invalid matrix whose determinant is
                            not 1 or -1");
        System.out.println("Please try again.");
    }
} // 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;

    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);
                    calccount++;
                    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)

                    nomore = false;
                } // 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 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;

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

        System.out.println();
        generator.nextElement = null; // reset matrix
        swap.elements = carrier.elements;
        swap.name = carrier.name;
        generatorsize = 1;

        insertIdentity(generator); // Insert Identity Matrix
        System.out.println();
        System.out.print("Generator: ");

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

        System.out.println();
        System.out.println("Press Enter to continue");
        String temp = BasicIo.readString();
        CalcGroup(generator, "subgroup", generatorsize, false);
        carrier = carrier.nextMatrix(carrier); // cycle

    } // end for(i)
} // 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
    }

    System.out.println();
    System.out.println("TO BE COMPLETED");
    System.out.println();
} // end Subgroup

```

```

// METHOD TO CALCULATE THE HASSER DIAGRAM OF A SUBGROUP STRUCTURE
public static void Hasser()
{ // start Hasser

    System.out.println();
    System.out.println("TO BE COMPLETED");
    System.out.println();

} // 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

// MAIN METHOD
public static void main(String args[]) throws Exception
{ // start MAIN

    int Choice = -1;

    // insert Identity Matrix into the list
    insertIdentity(MatrixList);

    // MENU

    do // repeat this until option 7 is chosen
    { // start DO

        Choice = -1;

        System.out.println();
        System.out.println("MAIN MENU - PLEASE ENTER YOUR CHOICE.");
        System.out.println("-----");
        System.out.println();
        System.out.println("1. Display the List");
        System.out.println("2. Add a Matrix to the list");
        System.out.println("3. Calculate the Group");
        System.out.println("4. Calculate the Subgroups Generated by an Element");
        System.out.println("5. Calculate the Subgroups");
        System.out.println("6. Hasser Diagram of the Subgroups");
        System.out.println("7: Exit the Program");
        System.out.println();

        try
        {
            Choice = BasicIo.readInteger();
        }

        catch (Exception exp) { }

        switch(Choice)
        { // start SWITCH

            case 1: Display(MatrixList, "list", count);
                    break;

            case 2: AddMatrix();
                    break;

            case 3: CalcGroup(MatrixList, "list", count, true);
                    break;

```

```
case 4: Element();
        break;

case 5: Subgroup();
        break;

case 6: Hasser();
        break;

case 7: System.out.println("The Program will now close...");
        break;

default: System.out.println("Sorry, Input is incorrect. Please Try Again. ");

        } // end SWITCH
    } // end DO
while (Choice != 7);

} // end MAIN

} // end groupcalc
```

# Testing the Program

```
>java groupcalc
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

- ```
-----
1. Display the List
2. Add a Matrix to the list
3. Calculate the Group
4. Calculate the Subgroups Generated by an Element
5. Calculate the Subgroups
6. Hasser Diagram of the Subgroups
7: Exit the Program
```

```
1
```

```
Number of matrices in the list: 1
```

```
List Number: 1. Matrix Identifier: I
```

```
1      0      0
0      1      0
0      0      1
```

```
Number of matrices in the list: 1
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

- ```
-----
1. Display the List
2. Add a Matrix to the list
3. Calculate the Group
4. Calculate the Subgroups Generated by an Element
5. Calculate the Subgroups
6. Hasser Diagram of the Subgroups
7: Exit the Program
```

```
2
```

```
Please enter the elements of your 3x3 matrix
```

```
Row 1, Column 1: -1
Row 2, Column 1: 0
Row 3, Column 1: 0
Row 1, Column 2: 0
Row 2, Column 2: -1
Row 3, Column 2: 0
Row 1, Column 3: 0
Row 2, Column 3: 0
Row 3, Column 3: 1
```

```
Please enter a descriptor for your matrix e.g. A, B,
P, Q,...P
```

```
The Determinant of the matrix is 1
Now adding the matrix to the list.
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

- ```
-----
1. Display the List
2. Add a Matrix to the list
3. Calculate the Group
4. Calculate the Subgroups Generated by an Element
5. Calculate the Subgroups
6. Hasser Diagram of the Subgroups
7: Exit the Program
```

```
1
```

```
Number of matrices in the list: 2
```

```
List Number: 1. Matrix Identifier: I
```

```
1      0      0
0      1      0
0      0      1
```

```
List Number: 2. Matrix Identifier: P
```

```
-1     0     0
0     -1    0
0      0     1
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

- ```
-----
1. Display the List
2. Add a Matrix to the list
3. Calculate the Group
4. Calculate the Subgroups Generated by an Element
5. Calculate the Subgroups
6. Hasser Diagram of the Subgroups
7: Exit the Program
```

```
2
```

```
Please enter the elements of your 3x3 matrix
```

```
Row 1, Column 1: 0
Row 2, Column 1: 1
Row 3, Column 1: 0
Row 1, Column 2: 0
Row 2, Column 2: 0
Row 3, Column 2: 1
Row 1, Column 3: 1
Row 2, Column 3: 0
Row 3, Column 3: 0
```

```
Please enter a descriptor for your matrix e.g. A, B,
P, Q,...Q
```

```
The Determinant of the matrix is 1
Now adding the matrix to the list.
```

```
MAIN MENU - PLEASE ENTER YOUR CHOICE.
```

- ```
-----
1. Display the List
2. Add a Matrix to the list
3. Calculate the Group
4. Calculate the Subgroups Generated by an Element
5. Calculate the Subgroups
6. Hasser Diagram of the Subgroups
7: Exit the Program
```

```
3
```

```
Number of matrices in the list: 12
```

```
List Number: 1. Matrix Identifier: I
```

```
1      0      0
0      1      0
0      0      1
```

```
List Number: 2. Matrix Identifier: P
```

```
-1     0     0
0     -1    0
0      0     1
```

```
List Number: 3. Matrix Identifier: Q
```

```
0      0      1
1      0      0
0      1      0
```

```
Press Enter to continue....
```

```
List Number: 4. Matrix Identifier: PQ
```

```
0      0     -1
-1     0     0
0      1     0
```

```
List Number: 5. Matrix Identifier: QP
```

```
0      0      1
-1     0      0
0     -1      0
```

```
List Number: 6. Matrix Identifier: QQ
```

```
0      1      0
0      0      1
```

Number of matrices in the list: 2

List Number: 7. Matrix Identifier: QPQ

```

0      1      0
0      0      -1
-1     0      0
    
```

List Number: 8. Matrix Identifier: QQP

```

0      -1     0
0      0      1
-1     0      0
    
```

List Number: 9. Matrix Identifier: QQPQ

```

-1     0      0
0      1      0
0      0      -1
    
```

Press Enter to continue....

List Number: 10. Matrix Identifier: PQP

```

0      0      -1
1      0      0
0      -1     0
    
```

List Number: 11. Matrix Identifier: PQQ

```

0      -1     0
0      0      -1
1      0      0
    
```

List Number: 12. Matrix Identifier: PQQPQ

```

1      0      0
0      -1     0
0      0      -1
    
```

Press Enter to continue....

Number of matrices in the list: 12

MAIN MENU - PLEASE ENTER YOUR CHOICE.

- ```

-----
1. Display the List
2. Add a Matrix to the list
3. Calculate the Group
4. Calculate the Subgroups Generated by an Element
5. Calculate the Subgroups
6. Hasser Diagram of the Subgroups
7: Exit the Program
    
```

4

Generator: Matrix Identifier: I

```

1      0      0
0      1      0
0      0      1
    
```

Press Enter to continue

Number of matrices in the subgroup: 1

List Number: 1. Matrix Identifier: I

```

1      0      0
0      1      0
0      0      1
    
```

Number of matrices in the subgroup: 1

```

1      0      0
    
```

Press Enter to continue....

Generator: Matrix Identifier: P

```

-1     0      0
0      -1     0
0      0      1
    
```

Press Enter to continue

Number of matrices in the subgroup: 2

List Number: 1. Matrix Identifier: I

```

1      0      0
0      1      0
0      0      1
    
```

List Number: 2. Matrix Identifier: P

```

-1     0      0
0      -1     0
0      0      1
    
```

Number of matrices in the subgroup: 2

Generator: Matrix Identifier: Q

```

0      0      1
1      0      0
0      1      0
    
```

Press Enter to continue

Number of matrices in the subgroup: 3

List Number: 1. Matrix Identifier: I

```

1      0      0
0      1      0
0      0      1
    
```

List Number: 2. Matrix Identifier: Q

```

0      0      1
1      0      0
0      1      0
    
```

List Number: 3. Matrix Identifier: QQ

```

0      1      0
0      0      1
1      0      0
    
```

Press Enter to continue....

Generator: Matrix Identifier: PQ

```

0      0      -1
-1     0      0
0      1      0
    
```

Press Enter to continue

etc.

MAIN MENU - PLEASE ENTER YOUR CHOICE.

- ```

-----
1. Display the List
2. Add a Matrix to the list
3. Calculate the Group
4. Calculate the Subgroups Generated by an Element
5. Calculate the Subgroups
6. Hasser Diagram of the Subgroups
7: Exit the Program
    
```

7

The Program will now close...

>

Source Code: Applet (29/10/2000)

```

/*
 *
 * Gareth Evans
 *
 * Started:      23rd October 2000
 * Last Modified: 29th October 2000 (Revision 1.4)
 *
 * Group Classification for 3x3 matrices
 *
 * Main Class (Applet)
 *
 */

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

public class GroupApplet extends Applet {

    // My Variables
    String messageStore = new String();
    static Matrix3x3 MatrixList = new Matrix3x3(); // Holds group matrices
    static Matrix3x3 DisplayList = new Matrix3x3();
    static int count = 1; // Keeps track of the size of MatrixList
    static int displaycount = 1;
    static int displayposition = 1;
    static String displayString = new String();
    static int customfunction = -1;

    // JBuilder Variables
    boolean isStandalone = false;
    BorderLayout MasterBorderLayout = new BorderLayout();
    Panel IOPanel = new Panel();
    GridLayout IOGridLayout = new GridLayout();
    Panel inputPanel = new Panel();
    Panel outputPanel = new Panel();
    TextField Message = new TextField();
    static TextField inputMessage = new TextField();
    Panel inputSub = new Panel();
    BorderLayout InputBorderLayout = new BorderLayout();
    BorderLayout borderLayout2 = new BorderLayout();
    static TextField outputMessage = new TextField();
    Panel outputSub = new Panel();
    BorderLayout borderLayout3 = new BorderLayout();
    Button Next = new Button();
    static Button Custom = new Button();
    Panel outputMatBase = new Panel();
    GridLayout gridLayout1 = new GridLayout();
    static TextField OA = new TextField();
    static TextField OE = new TextField();
    static TextField OF = new TextField();
    static TextField OD = new TextField();
    static TextField OC = new TextField();
    static TextField OI = new TextField();
    static TextField OG = new TextField();
    static TextField OH = new TextField();
    static TextField OB = new TextField();
    GridLayout inputGrid = new GridLayout();
    static TextField IA = new TextField();
    static TextField II = new TextField();
    static TextField IF = new TextField();
    static TextField Identifier = new TextField();
    static TextField IH = new TextField();
    static TextField IC = new TextField();
    static TextField IE = new TextField();
    static TextField IB = new TextField();
    static TextField IG = new TextField();
    Panel panell = new Panel();
    static TextField ID = new TextField();
    Button Add = new Button();
    Button Display = new Button();
    Button CalcSubgroup = new Button();
    Button CalcGroup = new Button();
    Button CalcElement = 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() {
    insertIdentity(MatrixList);
    try {
        jbInit();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}

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

    result.name = Identifier.getText();

    return result;
}

// DISPLAY (SINGLE) MATRIX METHOD
public static void DisplayMatrix(Matrix3x3 single, int position, int total, String
    descriptor) throws Exception
{
    outputMessage.setText(descriptor + " " + position + " of " + total + ": " + single.name);
    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]);
}

// 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

{
    DisplayList = disp;
    displaycount = size;
    displayString = descriptor;
    displayposition = 1;
    DisplayMatrix(DisplayList, 1, size, descriptor);
}

// METHOD TO CHECK WHETHER A MATRIX EXISTS IN A MATRIX LIST ALREADY
public static boolean AlreadyInList(Matrix3x3 check, Matrix3x3 fulllist, int loopduration)
{
    // 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;
}

// METHOD TO ADD A (VALID) MATRIX TO A MATRIX LIST
public static void AddMatrix() throws Exception
{
    Matrix3x3 check = GetInput();
    inputMessage.setText("Det(" + Identifier.getText() + "): " + check.Det(check));

    // 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
        {
            MatrixList.insert(MatrixList, check);
            count++;
        }
    }
    else
    {
        // Display Error Message
        inputMessage.setText("Invalid Determinant!");
    }
}

// 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
{
    // 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;

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

        for(int i=1; i<=calccount; i++)
        {
            for(int j=1; j<=calccount; j++)
            {
                testixj = testi.Multiply(testi, testj);
                boolean result = AlreadyInList(testixj, Manip, calccount);

                if (result == false) // i.e. not in list already
                {
                    Manip.insert(Manip, testixj);
                    calccount++;
                    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)

                    nomore = false;
                }
                testj = testj.nextMatrix(testj);
            }
        }
    }
}

```

```

    }
    testi = testi.nextMatrix(testi);
    testj = Manip; // reset
    }
    testi = Manip; // reset
}

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

// METHOD TO CALCULATE SUBGROUP GENERATED BY AN ELEMENT OF A GROUP
public static void Element() throws Exception
{
    // 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;

    for (int i=1; i<=count; 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));
        }
        else
        {
            //DisplayMatrix(generator);
        }

        // LISTEN FOR BUTTON PRESS - "Calculate Subgroup"
        Custom.setLabel("Calculate Subgroup");

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

        //LISTEN FOR BUTTON PRESS - "Next Subgroup"
        Custom.setLabel("Next Subgroup");
    }
    Custom.setLabel("");
}

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

    inputMessage.setText("TO BE DONE");
}

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

```

```

// METHOD TO INSERT AN IDENTITY MATRIX INTO ANY MATRIX LIST
public static void insertIdentity(Matrix3x3 input)
{
    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);
}

//Component initialization
private void jbInit() throws Exception {
    this.setLayout(MasterBorderLayout);
    MasterBorderLayout.setVgap(2);
    IOPanel.setBackground(Color.white);
    IOPanel.setLayout(IOGridLayout);
    IOGridLayout.setColumns(2);
    outputPanel.setBackground(Color.red);
    outputPanel.setLayout(borderLayout2);
    inputPanel.setBackground(Color.green);
    inputPanel.setLayout(InputBorderLayout);
    Message.setBackground(Color.darkGray);
    Message.setFont(new java.awt.Font("Serif", 1, 12));
    Message.setForeground(Color.white);
    Message.setEditable(false);
    inputMessage.setBackground(Color.gray);
    inputMessage.setForeground(Color.white);
    inputMessage.setEditable(false);
    outputMessage.setBackground(Color.gray);
    outputMessage.setForeground(Color.white);
    outputMessage.setEditable(false);
    outputSub.setLayout(borderLayout3);
    Next.setBackground(Color.lightGray);
    Next.setActionCommand("Down");
    Next.setLabel("Next");
    Next.addMouseListener(new java.awt.event.MouseAdapter() {

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

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

        public void mouseClicked(MouseEvent e) {
            Next_mouseClicked(e);
        }
    });
    Custom.setBackground(Color.lightGray);
    Custom.setActionCommand("Up");
    Custom.addMouseListener(new java.awt.event.MouseAdapter() {

        public void mouseEntered(MouseEvent e) {
            Custom_mouseEntered(e);
        }
    });
    Custom.addMouseListener(new java.awt.event.MouseAdapter() {

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

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

        public void mouseClicked(MouseEvent e) {
            Custom_mouseClicked(e);
        }
    });
}

```

```

outputMatBase.setBackground(Color.lightGray);
outputMatBase.setLayout(gridLayout1);
gridLayout1.setColumns(3);
gridLayout1.setHgap(2);
gridLayout1.setRows(3);
gridLayout1.setVgap(2);
OA.setBackground(Color.yellow);
OA.setEditable(false);
OA.setText("0");
OE.setBackground(Color.cyan);
OE.setEditable(false);
OE.setText("0");
OF.setBackground(Color.cyan);
OF.setEditable(false);
OF.setText("0");
OD.setBackground(Color.cyan);
OD.setEditable(false);
OD.setText("0");
OC.setBackground(Color.yellow);
OC.setEditable(false);
OC.setText("0");
OI.setBackground(Color.green);
OI.setEditable(false);
OI.setText("0");
OG.setBackground(Color.green);
OG.setEditable(false);
OG.setText("0");
OH.setBackground(Color.green);
OH.setEditable(false);
OH.setText("0");
OB.setBackground(Color.yellow);
OB.setEditable(false);
OB.setText("0");
inputSub.setBackground(Color.lightGray);
inputSub.setLayout(inputGrid);
inputGrid.setColumns(4);
inputGrid.setHgap(2);
inputGrid.setRows(4);
inputGrid.setVgap(2);
IA.setBackground(Color.yellow);
IA.setText("0");
II.setBackground(Color.green);
II.setText("0");
IF.setBackground(Color.cyan);
IF.setText("0");
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() {

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

    public void mouseExited(MouseEvent e) {
        Identifier_mouseExited(e);
    }
});
IH.setBackground(Color.green);
IH.setText("0");
IC.setBackground(Color.yellow);
IC.setText("0");
IE.setBackground(Color.cyan);
IE.setText("0");
IB.setBackground(Color.yellow);
IB.setText("0");
IG.setBackground(Color.green);
IG.setText("0");
ID.setBackground(Color.cyan);
ID.setText("0");
Add.setFont(new java.awt.Font("Dialog", 1, 12));
Add.setLabel("+");
Add.addMouseListener(new java.awt.event.MouseAdapter() {

    public void mouseClicked(MouseEvent e) {
        Add_mouseClicked(e);
    }
});

```

```

Add.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        Add_mouseClicked(e);
    }
});
Add.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseEntered(MouseEvent e) {
        Add_mouseEntered(e);
    }

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

    public void mouseClicked(MouseEvent e) {
        Add_mouseClicked(e);
    }
});
Display.setFont(new java.awt.Font("Dialog", 1, 12));
Display.setLabel("D");
Display.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        try
        {
            Display_mouseClicked(e);
        }
        catch (Exception exp) { }
    }
});
Display.addMouseListener(new java.awt.event.MouseAdapter() {
    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) { }
    }
});
CalcSubgroup.setFont(new java.awt.Font("Dialog", 1, 12));
CalcSubgroup.setLabel("S");
CalcSubgroup.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseExited(MouseEvent e) {
        CalcSubgroup_mouseExited(e);
    }

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

    public void mouseClicked(MouseEvent e) {
        CalcSubgroup_mouseClicked(e);
    }
});
CalcGroup.setFont(new java.awt.Font("Dialog", 1, 12));
CalcGroup.setLabel("G");
CalcGroup.addMouseListener(new java.awt.event.MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        CalcGroup_mouseClicked(e);
    }
});

```

```

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

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

    public void mouseEntered(MouseEvent e) {
        CalcGroup_mouseEntered(e);
    }
});
CalcElement.setFont(new java.awt.Font("Dialog", 1, 12));
CalcElement.setLabel("E");
CalcElement.addMouseListener(new java.awt.event.MouseAdapter() {

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

    public void mouseExited(MouseEvent e) {
        CalcElement_mouseExited(e);
    }
});
CalcElement.addMouseListener(new java.awt.event.MouseAdapter() {

    public void mouseEntered(MouseEvent e) {
        CalcElement_mouseEntered(e);
    }
});
CalcElement.addMouseListener(new java.awt.event.MouseAdapter() {

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

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

    public void mouseClicked(MouseEvent e) {
        CalcElement_mouseClicked(e);
    }
});
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(panell, 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);
}

```

```

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

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

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

void Add_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

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

void Display_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

void CalcGroup_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

void CalcSubgroup_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

void CalcElement_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

void Custom_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

void Next_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

void CalcGroup_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText("Calculate Group");
}

void CalcSubgroup_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText("Calculate Subgroups");
}

void Identifier_mouseExited(MouseEvent e) {
Message.setText(messageStore);
}

void CalcElement_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText("Element Generated Subgroups");
messageStore = null;
}

void Custom_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText(Custom.getLabel());
}

void Next_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText("Next Matrix");
}

```

```

void Identifier_mouseEntered(MouseEvent e) {
messageStore = Message.getText();
Message.setText("Matrix Identifier");
}

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

void Display_mouseClicked(MouseEvent e) throws Exception {
// DISPLAY LIST OF MATRICES
Display(MatrixList, "Matrix", count);
}

void CalcGroup_mouseClicked(MouseEvent e) {
// CALCULATE GROUP
try
{
    CalcGroup(MatrixList, "Group", count, true);
}
catch (Exception exp) { }
}

void CalcSubgroup_mouseClicked(MouseEvent e) {
// CALCULATE SUBGROUP
try
{
    Subgroup();
}
catch (Exception exp) { }
}

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

void Custom_mouseClicked(MouseEvent e) {
// CUSTOM FUNCTION

switch(customfunction)
{
    case 1: //CUSTOM FUNCTION 1
        break;

    case 2: //CUSTOM FUNCTION 2
        break;

    default:
}
}

void Next_mouseClicked(MouseEvent e) {
try
{
// SHOW NEXT MATRIX
if (displayposition < displaycount)
{
    DisplayList = DisplayList.nextMatrix(DisplayList);
    displayposition++;
    DisplayMatrix(DisplayList, displayposition, displaycount, displayString);
}
}
catch (Exception exp) { }
}
}

```