

COMPUTATIONAL TOOLS FOR GROUP THEORY

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

by

Jeffrey H. Barr

Spring 2005

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Jeffrey H. Barr:

Computational Tools for Group Theory

Carl Eckberg, Chair
Computer Science

William Root
Computer Science

Marcus Greferath
Mathematics and Statistics

Approval Date

Copyright © 2005

by

Jeffrey H. Barr

All Rights Reserved

DEDICATION

This thesis is dedicated to my family. I could not have accomplished this without the support and love of Becca and Daniel.

ABSTRACT OF THE THESIS

Computational Tools for Group Theory
by
Jeffrey H. Barr
Master of Science in Computer Science
San Diego State University, 2005

This thesis describes the refinement and extension of code that was originally developed as part of a 1987 Math 797 project by David Gibbs, "Computer Generation and Identification of Groups of Order 2 to 31." The purpose of the code was to generate, identify, and analyze groups presented in the form of a Cayley Table. Gibbs' code was transferred from Pascal to Java. Objects were created to improve the code design and allow for better interaction between the generation, identification, analysis, and visualization sections of code.

The code for this thesis allows cyclic groups to easily be generated, along with groups created via defined relationships and the cross product of multiple groups. A user interface was added to the system to assist the user when utilizing the code as well as visualizing the groups that are generated. Functionality to allow a user to manually enter a Cayley Table for analysis was also added to the system.

The generation code is no longer limited to groups of order less than 31. Improvements were made to the identification code so that the system can identify all Abelian groups including those created via the cross product of groups. Additionally, many non-Abelian groups of order 32 were added to the list of groups which could be identified.

Analysis functionality was added including the identification of whether a table actually represents a group as well as if the group is Abelian. Also, functionality was added to calculate the inner automorphism group of the group being displayed. The analysis functionality will provide users the ability to analyze groups that the code cannot yet identify.

TABLE OF CONTENTS

	PAGE
ABSTRACT	v
LIST OF TABLES.....	vii
LIST OF FIGURES	ix
ACKNOWLEDGEMENTS	x
CHAPTER	
1 INTRODUCTION	1
2 BASIC GROUP THEORY.....	3
3 FINITE ABELIAN GROUPS	6
4 GENERATORS AND RELATIONS	10
5 HISTORICAL BACKGROUND.....	14
Program gen_zn.pas	15
Program def_rel.pas.....	15
Program cross_pr.pas	17
Program ident_gp.pas	18
6 USER INTERFACE.....	21
Classes groupMainFrame and groupMain.....	22
Group Generation Buttons	22
Group Analysis Buttons.....	26
Class groupPanel.....	28
Class groupTableModel.....	29
7 CODE DESCRIPTION	32
Class groupMatrix.....	32
Class groupCreator	33
createCyclicGroup Method.....	34
createDefineRelationshipGroup Method	34
createXProdGroup Method.....	35
createInnerAutGroup Method	35

Class groupIdentify	36
8 ADDITIONAL WORK.....	38
9 FUTURE WORK.....	41
APPENDICES.....	45
A ALL GROUPS OF ORDER 2 TO 31 AND SOME GROUPS OF ORDER 32	45
B CODE.....	51
Main.java	52
groupMain.java	53
groupMain.java	69
groupPanel.java.....	87
groupTableModel.java.....	90
groupRelation.java	93
groupMatrix.java.....	96
groupCreator.java.....	104
groupIdentify.java	114
groupPermutation.java.....	129
PermutationGenerator.java	132

LIST OF TABLES

	PAGE
Table 1. Cayley Table for $Z_4 \oplus Z_2$	9
Table 2. Cayley Table for D_4	13
Table 3. Number of Cayley Tables for Groups of Order < 6	39
Table 4. All Groups of Order 2 to 31 and Some Groups of Order 32'.....	46

LIST OF FIGURES

	PAGE
Figure 1. Cayley tables for Z_2	5
Figure 2. Sample view of the user interface including the group D_4	21
Figure 3. Cyclic input dialog box.	23
Figure 4. Error message dialog box.	28
Figure 5. Tables of order 3 with identity fixed at element 0.....	38

ACKNOWLEDGEMENTS

I would like to thank and acknowledge the contributions of the members of my thesis committee: Dr. Carl Eckberg, Mr. William Root, and Dr. Marcus Greferath.

CHAPTER 1

INTRODUCTION

The purpose of this thesis was to refine and extend the capabilities of a 1987 Math 797 project by David Gibbs, “Computer Generation and Identification of Groups of Order 2 to 31.” The code was transferred from Pascal to Java in order to ease the addition of a user interface that could be turned into an applet. The user interface was added to aid in visualizing the groups as a Cayley table as well as to assist the user when utilizing the code to generate and analyze groups. Also, Java enabled the ability to make the code object oriented so that it could be easily extended and different objects could be used in other future applications. The code to generate groups was made more flexible in order to enable generation of groups with orders greater than 31 including defined relationship groups utilizing more than four generators. Furthermore, the code can now identify all abelian groups including those created via the cross product of groups, while many non-abelian groups of order 32 were added to the list of non-abelian groups which could be identified. Finally, additional functionality was added to the code in order to allow for further analysis of groups that are entered into the system.

The code utilizes a `groupMatrix` object that contains a container class that holds the actual group as an array of values similar to a Cayley table. In addition, the methodologies to analyze the group are also contained in the `groupMatrix` class. Utilizing this single class allowed the code to transfer an object that would contain the group from the generator object (`groupCreator`) to both the user interface object (a `JTable` defined by an `AbstractTableModel` extension `groupTableModel`) and the identifier object (`groupIdentify`). Further discussion of the code, as well as how to use the system, will be provided in the User Interface and Code Description chapters.

According to Gibbs, there are 92 groups of order 2 to 31 through isomorphism. There are an additional 51 groups of order 32, 44 of which are non-abelian.¹ Gibbs was able to classify or identify all groups of order 2 to 31 up to isomorphism when the identity of the group was element 0. However, Gibbs' code did not first determine if the Cayley Table that was entered represented an actual group before attempting to identify the group. Further description of Gibbs' method of generating and identifying groups will occur in the History chapter, while description of changes to Gibbs' methods is given in the Code Description chapter.

Some of the additional functionality added to the code included the ability for a user to enter a Cayley table, and functions to determine if a table was a group and if a group was Abelian. Also, functionality was added to determine characteristics of a group including the inner automorphisms of a group. How this new functionality works is described in the User Interface and Code Description chapters.

¹ Marshall Hall, Jr and James K. Senior, *The Groups of Order 2^n ($n \leq 6$)*, (New York: Macmillan, 1964), 2.

CHAPTER 2

BASIC GROUP THEORY

There are four characteristic properties that a nonempty set of elements G must have in order to be group under a specific binary operation \bullet (e.g. multiplication or addition):

1. Associativity. For all a, b, c in G , $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.
2. Identity. There is an element $\lambda \in G$ such that $a\lambda = \lambda a = a$ for all $a \in G$.
3. Inverses. For each element a in G , there is an element b in G (called an inverse of a) such that $ab = ba = \lambda$.²

Finite groups are groups that have a finite number of elements. The number of elements is the order of the group and is labeled $|G|$, where G is the group. Each element g in a group G also has an order, $|g|$, equal to “the smallest positive integer n such that $g^n = \lambda$ ”.³ For a group to be considered Abelian, all elements must commute with all other elements in group; that is, $ab = ba$ for all a and b in G .

A subgroup of a group G is a subset of elements of G that form a group using the same binary operation as G . Both the sets containing just the identity element λ , and the entire set of elements in G are subgroups of G . A subgroup H is a normal subgroup of G if all the elements in G , commute with the subgroup or $xH=Hx$ for all $x \in G$, while the center of G , $Z(G)$, is the set of all elements in G that commute with all other elements in G or $\{a \in G \mid ax=xa \text{ for every } x \in G\}$.⁴ Note, the center of an Abelian group is the group because every element commutes with all of the other elements.

A Cayley Table, or multiplication table, is a method for displaying a group and the result of operations on the elements. The table consists of n rows and columns, where n is the order of the group and each column and row represents an element. The first row and column each represent the first element; the second row and column represent the second

² Joseph Gallian, *Contemporary Abstract Algebra*, 5th ed. (New York: Houghton Mifflin Co., 2001), 43.

³ Joseph Gallian, *Contemporary Abstract Algebra*, 5th ed. (New York: Houghton Mifflin Co., 2001), 58.

⁴ Jimmie Gilbert and Linda Gilbert, *Elements of Modern Algebra*, 5th ed. (Pacific Grove, CA: Brooks Cole, 2000), 122, 130, 178.

element, etc. Each cell in the Cayley Table represents one of the possible operations in the group where the result of the binary operation (row element • column element) is in the cell. Since the operations in a group are closed, all cells are filled with elements of the group. Also, since a group is not necessarily commutative, the result of (row element • column element) does not necessarily equal (column element • row element).

A group G is a cyclic group “if there is an element a in G such that $G = \{a^n \mid n \in \mathbb{Z}\}$.”⁵ Element a is called a generator of G and $|a|=|G|$. The group generated by a is $\langle a \rangle$. The sets of the form $\{0, 1, \dots, n-1\}$ are all cyclic groups under the binary operation addition modulus n . These groups are usually written \mathbb{Z}_n , where n is also the order of the group. A group \mathbb{Z}_n can have many generators, but they each form the same group. For example the group \mathbb{Z}_6 has two elements that are generators, element 1 and element 5, where element 5 utilizes addition modulus 6 to create the set $\{0, 5, 4, 3, 2, 1\}$ which is identical to the set created by element one $\{0, 1, 2, 3, 4, 5\}$.

When there are two groups G and H which may or may not have the same binary operation, a function $\phi: G \rightarrow H$ is a homomorphism from G to H such that $\phi(ab) = \phi(a)\phi(b)$ for all a and b in G . When the homomorphism is one-to-one and onto, or bijective, the homomorphism is called an isomorphism from $G \rightarrow H$, G and H are said to be isomorphic. In order to prove two groups G and H are isomorphic, there are four requirements:⁶

1. Mapping: there is a function ϕ from $G \rightarrow H$.
2. One-to-one: if $\phi(a) = \phi(b)$, then $a = b$.
3. Onto: for any element $h \in H$, there is an element $g \in G$ such that $\phi(g) = h$.
4. Operation Preserving: $\phi(ab) = \phi(a)\phi(b)$ for all $a, b \in G$

The use of isomorphisms is useful in studying groups because groups that look very different can still be isomorphic, allowing them to be described by the same group. In fact the two groups shown in the Cayley Tables in Figure 1 are both isomorphic to \mathbb{Z}_2 and only differ in their respective identity element.

⁵ Jimmie Gilbert and Linda Gilbert, *Elements of Modern Algebra*, 5th ed. (Pacific Grove, CA: Brooks Cole, 2000), 128.

⁶ Joseph Gallian, *Contemporary Abstract Algebra*, 5th ed. (New York: Houghton Mifflin Co., 2001), 120.

	0	1
0	0	1
1	1	0

	0	1
0	1	0
1	0	1

Figure 1. Cayley tables for Z_2 .

An automorphism is an isomorphism of a group G onto itself. The set of all automorphisms of a group G form a group that is referred to as $\text{Aut}(G) = \{\phi: G \rightarrow G \mid \phi \text{ is an isomorphism}\}$. For some element a in group G , conjugation by a of element $b \in G$ is a mapping $\phi_a(b) = aba^{-1}$. ϕ_a is a particular automorphism of G induced by a because it is a mapping of G onto itself and has a special name, inner automorphism. The set of all inner automorphisms of a group G form a group that is referred to as $\text{Inn}(G)$ and is a normal subgroup of $\text{Aut}(G)$. Also, for any group G , $\text{Inn}(G) \cong G/Z(G)$. Therefore, since the center of an Abelian group is the group, $\text{Inn}(G) = \{\lambda\}$ or Z_1 for an Abelian group G .⁷

⁷ Charles Lanski, *Concepts in Abstract Algebra*, (Belmont, CA: Brooks Cole, 2005), 239-240.

CHAPTER 3

FINITE ABELIAN GROUPS

The groups that were identified in the code can be separated into two basic sections, Abelian and non-Abelian groups. The code is able to use the order of the elements of the group to identify the cyclic elements because if an element of a group is the order of the group it must be the generator of the group and the group is cyclic. The other two fundamental routines used to identify groups of order $2*p$ and p^2 are based upon the uniqueness of the order of the groups and the fact that only a specific group, other than the cyclic group, could be of that order. However, all of the groups identified by the cyclic, $2*p$, and p^2 methodologies could have been identified utilizing the remaining techniques.

The remaining groups were separated based upon whether they were Abelian. The identification methods for the remaining Abelian groups utilize the Fundamental Theorem of Finite Abelian Groups:

Every finite Abelian group is the direct product of cyclic groups of prime-power order. Moreover, the number of terms in the product and the orders of the cyclic groups are uniquely determined by the group.⁸

Note that this theorem includes the cyclic groups because the cyclic groups are all finite Abelian groups. In fact, the direct product of cyclic groups is a cyclic group if the orders of the cyclic groups are relatively prime. Also, the direct product in this definition is the same as the cross product used to generate groups in the code where \oplus is used in this document to represent the direct or cross product. A group of prime power order can be written Z_p^n where p is some prime taken to the power of n .

The power of the Fundamental Theorem is its application to the construction of all Abelian groups of a specific order. Firstly, for all Abelian groups of the order p^k , there would only be one group for each set of positive integers which sum k :

that is, if k can be written as

⁸ Joseph Gallian, *Contemporary Abstract Algebra*, 5th ed. (New York: Houghton Mifflin Co., 2001), 211.

$$k = n_1 + n_2 + \dots + n_t$$

where each n_i is a positive integer; then

$$Z_{p^{n_1}} \oplus Z_{p^{n_2}} \oplus \dots \oplus Z_{p^{n_t}}$$

is an Abelian group of order p^k .⁹

Secondly, the uniqueness portion of the Fundamental Theorem guarantees distinct isomorphism classes such that $Z_4 \oplus Z_3$ is not isomorphic to $Z_2 \oplus Z_2 \oplus Z_3$ even though both groups are of order 12. This leads to the next observation that the number of isomorphisms of Abelian groups is directly related to the prime factorization of the order of the group. Thus, through the determination of the prime factorization of the order of a group, the number of Abelian groups of that order can be determined. For example, the prime factorization of 24 is $2^3 * 3^1$ and these numbers can only be combined in three methods, while the prime-power order requirement is maintained and the total value of the combination equals 24. Thus, the only possible Abelian groups of order 24 are $Z_8 \oplus Z_3$, $Z_4 \oplus Z_2 \oplus Z_3$, and $Z_2 \oplus Z_2 \oplus Z_2 \oplus Z_3$. Note that since 24 is a product of two primes, 24 is not a prime-power order, so one of the three identified Abelian groups ($Z_8 \oplus Z_3$) must be isomorphic to the cyclic group Z_{24} . This also shows how groups of much higher order still have a limited number of Abelian groups. For example, the prime factorization of 120 is $2^3 * 3^1 * 5^1$ which can only be combined in three methods, similar to groups of order 24. Therefore, the three Abelian groups of order 120 are $Z_8 \oplus Z_5 \oplus Z_3$, $Z_4 \oplus Z_2 \oplus Z_5 \oplus Z_3$, and $Z_2 \oplus Z_2 \oplus Z_2 \oplus Z_5 \oplus Z_3$.

The fact that Z_{24} is isomorphic to $Z_8 \oplus Z_3$ leads to the ability to combine cyclic factors if they are relatively prime. To combine the cyclic factors in a logical manner, “obtain a direct product of the form $Z_{n_1} \oplus Z_{n_2} \oplus \dots \oplus Z_{n_k}$, where n_i divides n_{i-1} .”¹⁰ Therefore, instead of combining $Z_4 \oplus Z_2 \oplus Z_5 \oplus Z_3$ into $Z_{30} \oplus Z_4$, it would be described as $Z_{60} \oplus Z_2$ since 4 does not divide 30 but 2 divides 60. Nevertheless, these three groups that are

⁹ Joseph Gallian, *Contemporary Abstract Algebra*, 5th ed. (New York: Houghton Mifflin Co., 2001), 212.

¹⁰ Joseph Gallian, *Contemporary Abstract Algebra*, 5th ed. (New York: Houghton Mifflin Co., 2001), 215.

generated through the direct product of different cyclic groups are still isomorphic. The combining and naming notation are for ease of use only.

The order of an element in a group can at most be the maximum order of the group; however, we know for any element a in a group G , $\langle a \rangle$ is a subgroup of G , where $\langle a \rangle$ is the set $\{a^n \mid n \text{ is an integer}\}$ and the order of a , $|a|$, in G is also the order of a in $\langle a \rangle$. Therefore, the order of any cyclic group Z_k in the group $Z_{n_1} \oplus Z_{n_2} \oplus \dots \oplus Z_{n_k}$ is k since the order of any cyclic group Z_k is k and the maximum order of any element is the order of the group Z_k in the direct product with the highest order since the cyclic groups are all subgroups of the original group. Using this knowledge, the Fundamental Theorem can be converted into an algorithm to identify the cyclic elements that compose the direct product used to generate the group.

The Greedy Algorithm for an Abelian Group of order p^n shown below is an algorithm that utilizes the Fundamental Theorem to identify the factors used in the direct product for the group. This is very similar to the methodology used in the code to identify each of the factors that comprise the Abelian groups in the identification of remaining Abelian groups section of code. The steps of the algorithm are:¹¹

1.

The list of G_i , where i goes from 1 to k , is the list of the k cyclic groups that comprise the direct product used to generate the Abelian group G .

An example of the calculation of the direct product for $Z_4 \oplus Z_2$ is shown to further display how the groups and more specifically the Cayley Table created in the code is generated. The elements of Z_4 are $\{0, 1, 2, 3\}$, while the elements of Z_2 are $\{0, 1\}$. The element 0 is the identity element for both groups. In the direct product, these two groups combine their elements to form the new elements, $\{(0, 0), (0, 1), (1, 0), (1, 1), (2, 0), (2, 1), (3, 0), (3, 1)\}$ where the element $(0, 0)$ is the new identity element. The operation used to combine the elements in Z_4 and Z_2 is maintained so we can use the Cayley tables for each group to determine the combinations. Thus, if element $(1,0)$ is to be combined with element $(3,1)$, we would use the Cayley Table for Z_4 to combine 1 and 3, while we would use the Cayley Table for Z_2 to combine 0 and 1 in order to find element $(0, 1)$. Each of the new

¹¹ Joseph Gallian, *Contemporary Abstract Algebra*, 5th ed. (New York: Houghton Mifflin Co., 2001), 214-215.

elements of the group $Z_4 \oplus Z_2$ can be transformed into a single number creating an isomorphism of the original group that was generated through the direct product of the two groups. Thus, the new set of elements becomes $\{0, 1, 2, 3, 4, 5, 6, 7\}$ with a one-one correspondence with the original elements. The Cayley Table for $Z_4 \oplus Z_2$ with the direct product elements is shown in Table 1.

Table 1. Cayley Table for $Z_4 \oplus Z_2$

	(0, 0)	(0, 1)	(1, 0)	(1, 1)	(2, 0)	(2, 1)	(3, 0)	(3, 1)
(0, 0)	(0, 0)	(0, 1)	(1, 0)	(1, 1)	(2, 0)	(2, 1)	(3, 0)	(3, 1)
(0, 1)	(0, 1)	(0, 0)	(1, 1)	(1, 0)	(2, 1)	(2, 0)	(3, 1)	(3, 0)
(1, 0)	(1, 0)	(1, 1)	(2, 0)	(2, 1)	(3, 0)	(3, 1)	(0, 0)	(0, 1)
(1, 1)	(1, 1)	(1, 0)	(2, 1)	(2, 0)	(3, 1)	(3, 0)	(0, 1)	(0, 0)
(2, 0)	(2, 0)	(2, 1)	(3, 0)	(3, 1)	(0, 0)	(0, 1)	(1, 0)	(1, 1)
(2, 1)	(2, 1)	(2, 0)	(3, 1)	(3, 0)	(0, 1)	(0, 0)	(1, 1)	(1, 0)
(3, 0)	(3, 0)	(3, 1)	(0, 0)	(0, 1)	(1, 0)	(1, 1)	(2, 0)	(2, 1)
(3, 1)	(3, 1)	(3, 0)	(0, 1)	(0, 0)	(1, 1)	(1, 0)	(2, 1)	(2, 0)

The list of all Abelian groups of order less than or equal to 32 is included in the list of all groups of order less than 32 Table 4 in Appendix A.

CHAPTER 4

GENERATORS AND RELATIONS

A group can also be created through the use of generators and relations. Generators are the set of elements that are used in “a set of equations (called relations) that specify the conditions that these generators are to satisfy.”¹² The set of elements are defined as the generators of a group G if “every element of G is expressible as a finite product of their powers.”¹³ While the relations could include how any combination of the generators equal other combinations, usual relations show the minimum number of equations needed in order to derive every other relation between the generators. One of the more interesting aspects of generators and relations is that given a set of relations, any set of generators used that satisfy only the given relations will form a group isomorphic to any other set of generators that use those same relations.

Generators and relations are capable of creating all groups, including Abelian and non-Abelian. The generators and relations have no indication which is being created, only the differences between the relations identify the differences in the groups. For example, both the groups D_4 and $Z_4 \oplus Z_2$ utilize two generators and are order 8. However, the groups are distinctly different including the fact that D_4 is non-Abelian, while $Z_4 \oplus Z_2$ is Abelian. The generators and relations for each group of order 8 are:

- Z_8 : $a^8 = \lambda$ (identity element)
- $Z_4 \oplus Z_2$: $a^4 = b^2 = \lambda$ and $ba = ab$
- $Z_2 \oplus Z_2 \oplus Z_2$: $a^2 = b^2 = c^2 = \lambda$, $ba = ab$, $ca = ac$, and $cb = bc$
- D_4 : $a^4 = b^2 = (ab)^2 = \lambda$
- Quaternion: $a^4 = (ab)^2 = \lambda$ and $b^2 = a^2$

¹² Joseph Gallian, *Contemporary Abstract Algebra*, 5th ed. (New York: Houghton Mifflin Co., 2001), 433.

¹³ H.S.M. Coxeter and W.O.J. Moser, *Generators and Relations for Discrete Groups*, 4th ed. (New York: Springer-Verlag, 1980), 1.

Note that in D_4 and $Z_4 \oplus Z_2$, element a is order 4, while element b is order 2. The only real difference in the relations is that while $ba = ab$ in $Z_4 \oplus Z_2$, $ba = aaab$ in D_4 . However, the differences between the two relations cause significant differences in the groups.

The use of generators and relations was particularly useful in generating the non-Abelian groups that can be identified in the code. In particular, specific classification of groups with similar relations could be identified as classified based upon the relations between the generators. For example, dihedral groups of the form D_q are of the order $2q$ and have the relations $a^q = b^2 = (ab)^2 = \lambda$ with elements a and b of the cyclic groups Z_q and Z_2 , respectively, are the generators. What is actually occurring is that Z_q has an automorphism of order 2 that “transforms every element into its inverse.”¹⁴ Thus, adjoining the group Z_q with an element b of period 2, the cyclic group Z_q is transformed into the dihedral group D_q defined by the equation $b^{-1}ab = a^{-1}$ which is equivalent to the relation $(ab)^2 = \lambda$ given above.

Another convenient classification group is the dicyclic groups. These are identified as the groups of the form $\langle 2, 2, m \rangle$, where the order of the group is $4m$. Instead of the relations given by the dihedral group, the relations are slightly changed. This relationship is also based upon the same automorphism that transformed the element $a \rightarrow a^{-1}$ from a cyclic group of the form Z_{2m} . However, instead of adjoining the group with an element of period 2, an element b of period 4 is used.¹⁵ Thus, for the group $\langle 2, 2, m \rangle$, while the equation of the mapping is still $b^{-1}ab = a^{-1}$, or $(ab)^2 = \lambda$, the adjoining element is now of period 4 and instead of $b^2 = \lambda$, $b^4 = \lambda$. However, since $b^4 = \lambda$, and by definition of Z_{2m} , $a^{2m} = \lambda$, the square of b now equals a^m , or $b^2 = a^m$. Therefore, the relationships for the groups of the form $\langle 2, 2, m \rangle$ are

These groups lead to a more generalized relationship that provides an identification scheme for many of the non-Abelian groups of order less than 32. The relationship $S^m = T^n = e$, $T^{-1}ST = S^r$ provides a means to generate the groups:¹⁶

¹⁴ H.S.M. Coxeter and W.O.J. Moser, *Generators and Relations for Discrete Groups*, 4th ed. (New York: Springer-Verlag, 1980), 6.

¹⁵ H.S.M. Coxeter and W.O.J. Moser, *Generators and Relations for Discrete Groups*, 4th ed. (New York: Springer-Verlag, 1980), 7-8.

¹⁶ H.S.M. Coxeter and W.O.J. Moser, *Generators and Relations for Discrete Groups*, 4th ed. (New York: Springer-Verlag, 1980), 11.

- $Z_m \oplus Z_n$ when $r = 1$
- Z_{2n} when $r = -1$, $m=2$, and n is odd
- $D_m \oplus Z_{n/2}$ when $r = -1$, and $n/2$ is odd
- $\langle 2, 2, m \rangle$ when $r = -1$, $n = 4$, and m is odd
- $Z_2 \oplus \langle 2, 2, m/2 \rangle$ when $r = -1$, $n = 4$, and $m/2$ is odd

Also, notice in the equation $D_m \oplus Z_{n/2}$, when $n=2$ the equation gives the dihedral group D_m since Z_1 is the identity and the relations become $S^m = T^2 = \lambda$, $T^{-1}ST = S^{-1}$ which is equivalent to $S^m = T^2 = (ST)^2 = \lambda$.

An example of the generation of the group D_4 is shown to further display how the groups and more specifically the Cayley Table created in the code is generated. The relations for D_4 are $a^4 = b^2 = \lambda$, $ba = a^{-1}b$, where a^4 and b^2 represent the generators used along with their orders and inverses, while $ba = a^{-1}b$ is the “pseudo” commutative relationship used by the code. Starting with the group Z_4 generated by element a , the element b , which has a period of 2, is adjoined to Z_4 . Therefore, the elements of the group D_4 are $\{\lambda, a, aa, aaa, b, ab, aab, aaab\}$. The relations are then used to reorder the elements when an operation between two elements results in a combination that is not one of the elements of the group. For example if aab is combined with ab , the result is $aabab$. This new result can then be transformed utilizing the defined relationships whereby $aabab = aaa^{-1}bb$ because $ba = a^{-1}b$, $aaa^{-1}bb = abb$ because $aa^{-1} = a^4 = \lambda$, and $abb = a$ because $b^2 = \lambda$ so the final result of the combination $aab \bullet ab = a$. The final Cayley Table for D_4 is shown in Table 2 along with the relationships used to derive the final table.

Table 2. Cayley Table for D_4

	λ	a	aa	aaa	b	ab	aab	aaab
λ	λ	a	aa	aaa	b	ab	aab	aaab
a	a	aa	aaa	aaaa = λ	ab	aab	aaab	aaaab = b
aa	aa	aaa	aaaa = λ	aaaaa = a	aab	aaab	aaaab = b	aaaaab = ab
aaa	aaa	aaaa = λ	aaaaa = a	aaaaaa = aaa	aaab	aaaab = b	aaaaab = ab	aaaaaab = aab
b	b	ba = aaab	baa = aab	baaa = ab	bb = λ	bab = aaa	baab = aa	baaab = a
ab	ab	aba = b	abaa = aaab	abaaa = aab	abb = a	abab = λ	abaab = aaa	abaaab = aa
aab	aab	aaba = ab	aabaa = b	aabaaa = aaab	aabb = aa	aabab = a	aabaab = λ	aabaaab = aaa
aaab	aaab	aaaba = aab	aaaba = ab	aaabaaa = b	aaabb = aaa	aaabab = aa	aaabaab = a	aaabaaab = λ

All the groups of order less than or equal to 32 along with many of their generators and relations are shown in Table 4 in Appendix A.

CHAPTER 5

HISTORICAL BACKGROUND

David Gibbs', "Computer Generation and Identification of Groups of Order 2 to 31" goal was to generate and identify all 92 groups through order 31 including the 45 non-abelian groups.¹⁷ The code was originally written in Pascal in four distinct code pieces that ran independently. Three of the sections dealt with generating different types of groups, while the remaining section was for identifying the groups. Generated groups were stored as files that could be read by the identification code. No user interface was available except for command line instructions. All information for this chapter was taken from Gibbs' paper except where otherwise noted.

Gibbs was able to identify all groups of order 2 to 31 up to isomorphism when the identity of the group was element 0. By forcing element 0 to be the identity element, Gibbs also reduced the number of possible isomorphisms by a large amount. Since Gibbs' code did not determine if the table being identified actually represented a group, the user would have to ensure the table represented a group with element 0 as the identity before running the code.

There were three distinct sections of code that were created by Gibbs in order to generate the different groups. He separated the group generation into cyclic groups, groups formed by defined relationships, and cross product groups. Each of the generator programs created text files that contained the group or groups created. Each group in the text file consisted of single line that contained the order of the group and a comment field followed by n lines which contained n space delimited numbers. This $n \times n$ table represented Cayley Table for the group that was generated. These text files, along with files of similar structure, were then used as input to the identification program. Gibbs was able to identify generators for all of the groups of order less than 32.

¹⁷ David Gibbs, "Computer Generation and Identification of Groups of Order 2 to 31" (unpublished Math 797 Project, San Diego State University, 1984), 4-9.

PROGRAM GEN_ZN.PAS

Cyclic groups are groups that can be generated by a single element. Gibbs utilized modulus arithmetic to generate an example of the cyclic groups by adding the row and column for a particular cell in the Cayley Table and taking the modulus of the result with respect to the order of the group to be generated. Thus, for group Z_7 , the result in row 4, column 5 would be $(4+5) \bmod 7$ or element 2. Gibbs' program `gen_zn.pas` automatically created all 30 cyclic groups of order 2 to 31 when run. All 30 groups were stored in a single text file starting with group Z_2 through group Z_{31} , with each Cayley Table labeled with its order n and name Z_n .

Each cell could be calculated independently by utilizing the row and column numbers of the cell. Therefore, by looping through each row and column the entire Cayley Table could be generated in two for loops. Each cell could then be calculated with one addition operation and one division, yielding $f(n) = 2n^2$ operations. Therefore, the generation of Cayley Tables for cyclic groups ran in $O(n^2)$ time.

PROGRAM DEF_REL.PAS

Groups formed by defined relationships were created utilizing two to four generators (letters) that are combined based upon the order of the generators as well as what Gibbs called "pseudo" commutative substitution instances. Defining relationships were documented by Gibbs for 34 of the 45 non-abelian groups of order less than 32. The remaining 11 relationships were generated utilizing the cross product code. The order of the generator defined the maximum number of elements of that generator as well as the inverse element of the generator. The "pseudo" commutative substitution instances defined how the generators combined so that they could always be placed in the same order as well as any other relations between the generators. For example, if there were two generators, "a" and "b", a "pseudo" commutative substitution instance would be required to determine how to switch the order of the generators. Therefore, a combination of the generators, "ba", would have a substitution where the "a" generator was first. This combination of generator characteristics and "pseudo" commutative substitution instances are combined into a substitution list.

...the elements of the set were created by appending all possible ordered combinations of letters and stored in the array `elt_list`. Then, the group table is built, each element being formed by the concatenation with another element in the table. The resulting string is then examined against the substitution list for possible simplifications, and this process continues until the string is reduced to an element in the set (i.e. one in the array `elt_list`). It is then replaced by the corresponding integer subscript of the array `elt_list`, and the resulting group table once again consists of integers.¹⁸

Gibbs' `def_rel.pas` program used a command line entry system for its user interface. On startup, the program will first requests the user enters two file names, one for the Cayley Table to be stored in the same format as in the cyclic tables and the second for the defining relationships to be used to form each group. The program would then request the user to enter the order of the group that the user wished to create along with a name to be stored in the comment field. The program would then request the number of generators, assuming a number between 2 and 4 was entered. No error checking was used throughout the program in order to determine if the user entered a legal value. The order of each of the 2 to 4 generators, labeled a, b, c, and d, was then requested. The program would then request any relationships which were not pseudo-commutative that were used for substitution relationships. These would be relations of the form $bb=aa$, as used by the Quaternion group for replacing instances of bb in an evaluation of some combination of two elements with aa . The user would enter the left side of the equation and then the right at separate prompts using the value 1 for the identity element. The user would then enter the right side of the equation for a predefined list of "pseudo" commutative relationships. These would be relationships of the form $ba=a^w b^x c^y d^z$, where the left hand side represents all possible 2-element combinations of the generators used in reverse alphabetical. The values for w , x , y , and z actually represent the number of a's, b's, c's, and d's used in the equation. Therefore, the "pseudo" commutative relationship for the group D_8 which only has two generators would be $ba=aaab$. Also, since Gibbs' code understood the inverse to be the capitalization of the letter representing the generator, the "pseudo" commutative relationship for the group D_8 could also be represented as $ba=Ab$. If there were three generators, an additional two "pseudo" commutative relationships were needed for ca and cb , while four generators would require

¹⁸ David Gibbs, "Computer Generation and Identification of Groups of Order 2 to 31" (unpublished Math 797 Project, San Diego State University, 1984), 12.

even more relationships for da, db, and dc. The program will then list all of the relationships it is to use for generating the new group for the user to approve. If the user does not approve of the relationships, all data must be reentered. If the defined relationships were approved the program would store the relationships along with the order and comment field in a text file and then generate the group. Once the group was generated, it would be stored in the same format as the cyclic group with the order and comment field on one line followed by the Cayley Table on n lines, where n is the order of the group. The user can then exit the program or begin again and generate a new group to be stored in the same file.

According to Gibbs, the creation of the final element list would require at most $4n$ string concatenations and n array assignments. To combine each of the elements in the Cayley Table takes n^2 concatenations, one for each cell in the grid. In order to simplify a single element generated via the concatenations would require, in the worst case, substitutions on the order of $O(nm)$ where m is the number of generators. Since substitutions would be checked on all cells generated, the generation of defined relationships would run in $O(n^4)$.¹⁹

PROGRAM CROSS_PR.PAS

Cross product groups were formed by taking two groups and forming a new group based upon the combination of elements from both groups. The cross product code was used to create the remaining 28 groups, 17 abelian and 11 non-abelian. All of the groups used in the cross products are generated by either the cyclic method or the defined relationship method. For two groups of size n and m , the order for the corresponding cross product group would be $n*m$. Each element in the new group would initially be represented as a combination pair (e.g. $(0,0)$, $(0,1)$, ..., $(0,m-1)$, $(1, 0)$, ..., $(1, m-1)$, ..., $(n-1,0)$, $(n-1,1)$, ..., $(n-1,m-1)$) that is later mapped to its corresponding position in the array within which the pairs are stored. The cells in the Cayley table are calculated by utilizing the row and column values along with the array of pairs to determine positions in the original group tables that are then used to calculate new pairs whose position in the array is used as the cell value. For example, when calculating $Z_2 \times Z_4$, the elements 0 to 7 in the coordinate array would be $(0,0)$, $(0,1)$, $(0,2)$, $(0,3)$, $(1,0)$, $(1,1)$, $(1,2)$, and $(1,3)$. To calculate the value of the cell $(2,7)$

¹⁹ David Gibbs, "Computer Generation and Identification of Groups of Order 2 to 31" (unpublished Math 797 Project, San Diego State University, 1984), 11-12.

we would use elements 2 and 7, (0,2) and (1,3) respectively. The new cell would use the value of cells (0,1) from the Z2 table and the value of cell (2,3) from the Z4 table to give a new coordinate pair of (1,1) which is element 5 in the array. Therefore, cell (2,7) in the Cayley table for Z2 x Z4 would equal 5.

Gibbs' cross_pr.pas program would only generate the cross product of two groups. In order to generate cross products of more than two groups, the program had to be run multiple times to allow the results of one cross product to be combined with another group in order to build larger order cross product groups. The program would request, via the command line, the user enter the name of two files, each containing one group in the format that was previously described. The program would then use these two groups to generate a cross product, while calculating the order of the new group by determining the elements of the new group. This new group would be stored in the same format as the cyclic and defined relationship groups in a file named output.dat. The comment for the new group would consist of the comment from the first group, an x, and the comment from the second group. The program would then inquire whether the user wanted to create another cross product group or exit the program.

In order to load the two groups into memory, the program takes n_1^2 and n_2^2 operations where n_1 and n_2 are the order of the two groups and the order of the cross product group is $n=n_1*n_2$. It takes $2n$ operations to generate the cross product combination pairs. It then requires $2n^2$ operations to create the Cayley Table because each element in a pair has to be separately calculated. Finally, once each combination pair is generated, an additional maximum n operations are required to associate the combination pair with its array position. Thus, a total worst case has the program running in $O(n^2)$ time.²⁰

PROGRAM IDENT_GP.PAS

The program that was used for identifying a group, ident_gp.pas, requested from the user the name of a text file that was in the form used by the generator programs. It would then request the user decide if output would be sent to the screen or a file name the user chose. The program would then read the first group stored in the file, output the order and

²⁰ David Gibbs, "Computer Generation and Identification of Groups of Order 2 to 31" (unpublished Math 797 Project, San Diego State University, 1984), 10-11.

comment on the first line of the input file to the screen or output file and store the Cayley Table in a 128x128 integer array, severely wasting space. After identifying the group, the program would also output the name of the group to the screen or output file. If the input file was not empty, and contained another group, the program would then repeat the process.

The code goes through a specific five step process for identification after first calculating the number of elements of each order. No effort is made to determine if the Cayley Table actually represents a group before the identification process begins.

1. The code checks if the groups are cyclic by determining if the order of any of the elements in the group equals the order of the group. There are a total of 30 cyclic groups for groups of orders 2 to 31, one for each order.
2. The code then determines if the order of the group is of the form 2^*p , where p is prime (e.g. group order is 6 where $p=3$). If the group order is of the form 2^*p , the group is either cyclic which was determined in step 1, is the Klein-4 group if order 4, or one of the prime number dihedral groups (D3, D5, D7, D11, or D13) that have order less than 32. The specific group is determined by the value of p .
3. The code then determines if the order of the group is of the form p^2 , where p is prime (e.g. group order is 25 where $p=5$). If the group order is of the form p^2 , the group is either cyclic which was determined in step 1, or a cross product of the cyclic groups of order p . The only two groups with order less than 32 that this would include are $Z3 \times Z3$ (order 9) and $Z5 \times Z5$ (order 25) and the specific group is determined by the value of p .
4. The code then determines if the group is abelian by determining if all elements in the group commute. If the group is abelian, the code calculates the cross product group by first determining the highest order element of the group and using that value as the order of the first cyclic group in the cross product. The value is also divided into the total order of the group giving a remaining value of the group that is the factored into its prime factorization. This prime factorization is used along with the number of elements of the respective prime value to determine the number and value of the remaining cyclic groups in the cross product. Thus, if the prime factorization yields p^x , and there are n elements of order p in the group, the number of factors necessary to equal p^x and be used in the cross product is $\log_2(n+1) - 1$. There are a total of 14 remaining abelian cross product groups with order less than 32 that can be determined by this method.
5. The remaining groups that have not been identified are the non-abelian groups other than the dihedral groups of the form D_p , where p is prime identified in step 2. In all but two exceptions, the order of the elements of the group is used to uniquely identify the group. These groups are separated by group order then by the order of the elements of the group utilizing a case statement. The two exceptions are for two pairs of groups of order 16. Each pair of groups has the exact same order structure for the elements in each group. In order to differentiate each group, one pair can be differentiated by determining which group has the Klein-4 group as its center, while

the second pair can be differentiated by determining which group has a non-normal cyclic subgroup. There are a total of 40 additional non-abelian groups that are identified with this method.

The determination of the number of elements of each order is done in $2n^2$ time. Once the order of the elements is established, the determination of whether a group is cyclic can be done in n time. The code to check for whether the order is $2*p$ or p^2 is \sqrt{n} , while determination of whether a group is Abelian is n^2 . The identification of an Abelian group occurs in $3n$ operations, while the identification of the non-Abelian group is about $n+n^2+2n^3$ operations. However, only the differentiation of the non-abelian groups $Z_2 \oplus$ Quaternion and $Z_4 \times Z_4$ based upon determination of the normality of their subgroups requires $2n^3$ operations. All other non-abelian groups are also found in about $n+n^2$ operations. Therefore, the entire identification process runs in:²¹

$$\begin{array}{ll} f(n) = 2n^2 + n + 2*\text{sqrt}(n) + n^2 + 3n: & O(n^2) \text{ for Abelian groups} \\ f(n) = 2n^2 + n + 2*\text{sqrt}(n) + n^2 + n + 2n^3: & O(n^3) \text{ for non-Abelian groups.} \end{array}$$

²¹ David Gibbs, "Computer Generation and Identification of Groups of Order 2 to 31" (unpublished Math 797 Project, San Diego State University, 1984), 19-20.

CHAPTER 6

USER INTERFACE

The user interface was added to aid in visualizing the groups as a Cayley table as well as to assist the user when utilizing the code to generate and analyze groups. The user interface consists of a main panel that contains a series of buttons, labels, and a secondary panel that contains a table representing the group as a Cayley table. The interface was developed utilizing Java Swing components. A view of the interface is shown in Figure 2.

	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	1	0	7	6	5	4	3	2
2	2	3	4	5	6	7	0	1
3	3	2	1	0	7	6	5	4
4	4	5	6	7	0	1	2	3
5	5	4	3	2	1	0	7	6
6	6	7	0	1	2	3	4	5
7	7	6	5	4	3	2	1	0

The table is not an Abelian Group because it is not commutative

Check Group Properties:

Check if Group	Check if Abelian	Find Group Name	Find Inner Automorphism
----------------	------------------	-----------------	-------------------------

Choose Type of Group to Enter:

Cyclic Group	Cross Product Group	User Defined Group	Defined Relationship Group
--------------	---------------------	--------------------	----------------------------

Figure 2. Sample view of the user interface including the group D4

CLASSES GROUPMAINFRAME AND GROUPMAIN

Two separate main panels were created depending on the desired use of the software. A main extended JFrame, groupMainFrame, for use as a standalone utility and an extended JApplet, groupMain, for use as an applet as part of a website which are identical were both created. A section of JButtons is on each main panel, each of which has a method associated with it when the JButton's action listener is activated. There are two hidden JLabels that are only shown when in use in addition to the two labels describing the button sections. A secondary extended JPanel named groupPanel is an added component to groupMainFrame and groupMain. This extended JPanel contains the view of the actual Cayley table utilizing a JTable and the instance of groupPanel in the main frame or applet is called myGroup.

In addition to the Swing objects there are also additional objects that are used to create and identify the groups being displayed in the user interface. A groupCreator object, myCreator, contains methods to generate the groups and stores the group once it is generated. A groupIdentify object, groupNamer, contains the methods used to identify the groups on display and stores the name of the group once it is identified. Both of these classes use a common class called groupMatrix in order to store the group being generated, displayed, and identified. More detail on these objects can be found in the Code Description chapter.

Group Generation Buttons

The user interface has eight buttons that are JButton components. Four of the buttons are utilized to generate groups, while the other four buttons are used to analyze the groups that have been generated. Three of the group generation buttons are associated with actions that directly interface with code that is based upon the original Gibbs group generation code. This is code that is used to create the cyclic, defined relationship, and cross product groups. The methods that are called have functionality directly related to user interaction, such as requesting the order of the group to be generated, the number of groups to be used in the defined relationship or cross product, or the actual relationships between the groups. This data is input via input dialog boxes, while error messages questioning the input from the user is displayed in message dialog boxes. The limitations that Gibbs had in his software,

including the size and number of groups the user can use in generating new groups has been eliminated. The methods that call methods in the groupCreator object that directly generate, and store the groups, methods in the groupIdentify object to identify the groups, and methods in the groupPanel object to update the display of the groups. The generation, storage, and identification classes will be discussed in the next chapter, Code Description. The final group generation button is associated with a method that allows the user to directly input a Cayley table in the user interface.

The “Cyclic Group” JButton, btnZnGroup, is associated with a Java ActionListener that determines when the button is pressed and calls the method createZnGroup which is a method in the main panel. The method will then call the method createNextCyclicGroup which returns a boolean that signals successful generation of a cyclic group. The method createNextCyclicGroup opens a JOptionPane Input Dialog box, shown in Figure 3, that requests the user enter the order of the cyclic group to generate.

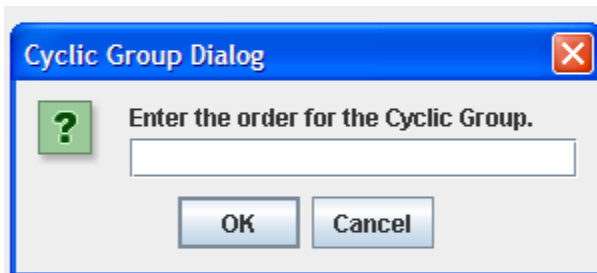


Figure 3. Cyclic input dialog box.

Error checking is done to ensure the user enters an integer. Unlike Gibbs, any order cyclic group can be generated. If an integer greater than 0 is entered, the method in groupCreator used to generate a cyclic group (createCyclicGroup) is called and the group generated is stored in the groupCreator object. If createNextCyclicGroup is ever unsuccessful, a JOptionPane message dialog box is displayed with the error that occurred and then the method returns failure to the method that called it. Upon successful creation of a group the createZnGroup method will update the groupMatrix object stored in myGroup and then call the findGroupName method that is discussed later in this chapter.

The “Defined Relationship Group” JButton, btnDefRelnGroup, is associated with a Java ActionListener that determines when the button is pressed and calls the method createDefinedRelationshipGroup which is a method in the main panel. The method opens a

JOptionPane Input Dialog box requesting the user enter the number of generators to use. Error checking is done to ensure an integer greater than or equal to two is entered. If there is an error at this point, the method returns without creating the new group. Unlike Gibbs, more than four generators can be used. The method will then request the order of each of the generators utilizing error checking to ensure an integer greater than 0 is added. If an error occurs, the system will continue to ask for a legal value until one is entered.

Each of these orders is added to an ArrayList containing the orders of each of the generators of the group and a groupRelation object is instantiated. An ArrayList is a Java collection class similar to an array that also allows the code to dynamically allocate space as more objects are added to the collection. A groupRelation object is a special object that contains two strings, left and right, which represent a relationship used in the generation of a defined relationship group. The groupRelation for each generator contains the generator repeated the order of the generator times for the left string and an empty string for the right string. For example the left string for a generator of order 3 would be "000" if it was the first generator, while the right string would be "". Each string can be retrieved separately for the groupRelation. Each groupRelation object is stored in an ArrayList called definedRelationships. A better description of the groupRelation object is given in the Code Description Chapter.

The method createDefinedRelationshipGroup than requests the user answer whether they have any additional relationships to add which were not pseudo-commutative that were used for substitution relationships. These would be relations of the form $bb=aa$, as used by the Quaternion group for replacing instances of bb in an evaluation of some combination of two elements with aa . All letter combinations are immediately converted to their corresponding number combinations, where $a=0$, $b=1$, etc. If the user enters "yes", the method would request the user enter a relationship in the form $left=right$, where "left" and "right" represent lists of generators. This relationship would be stored in another groupRelation object. If the left string already equals the same as a previously generated relation stored in the definedRelationships ArrayList, the right string is replaced. For example, the Quaternion group's second generator is order 2 giving the relationship $11 = ""$; however, the relationship $bb=aa$, would replace the relationship previously created for the order of the second generator so that the new relationship would be $11=00$. If the left string

has not been used in a previous relation, the new `groupRelation` is added to the `definedRelationship ArrayList`. The user is allowed to enter as many special relationships as desired. After the substitution relationships have been entered, the method then calls the `getNextRelationship` method for each “pseudo” commutative relationship to be added to the `definedRelationship ArrayList`. The `getNextRelationship` requests the “pseudo” commutative relationship for a defined left string (e.g. `ba`, `ca`, `cb`, etc.) and then performs error checking to ensure the order of the generators used in the relationship is alphabetical. This was because the elements of the defined relationship group always would always have the generators in alphabetical order. Therefore, a `groupRelation` whose right string that looked like “`aba`”, would not be able to create an element where the `a` generators never followed the `b` generators. Once all of the “pseudo” commutative relationships were defined the method in `groupCreator` used to generate a defined relationship group (`createDefineRelationshipGroup`) is called and the group generated is stored in the `groupCreator` object. Upon successful creation of a group, the `createDefinedRelationshipGroup` method will update the `groupMatrix` object stored in `myGroup` and then call the `findGroupName` method that is discussed later in this chapter.

The “Cross Product Group” `JButton`, `btnXProdGroup`, is associated with a Java `ActionListener` that determines when the button is pressed and calls the method `createXProdGroup` which is a method in the main panel. The method opens a `JOptionPane` Input Dialog box requesting the user enter the number of groups in the cross product. Error checking is done to ensure an integer greater than or equal to two is entered. If there is an error at this point, the method returns without creating the new group. Unlike Gibbs, more than two groups can be used to create the group. The user is then asked to enter 1 if the next group in the cross product is a cyclic group and a 2 if the next group is a defined relationship group. If a 1 is entered, the method calls `createNextCyclicGroup` described above, while if a 2 is entered, the method calls `createDefinedRelationshipGroup`. If something else is entered, the program returns without creating a new group. The group created is stored in a temporary instance of `groupMatrix`, `myGroup1`. Subsequent groups are requested from the user in the same manner and temporarily stored in another `groupMatrix` instance, `myGroup2`. Once two groups have been created, the method in `groupCreator` used to generate a cross product group (`createXProdGroup`) is called and the group generated is stored in the

groupCreator object. If another group needs to be added to the cross product, the previously created group stored in the groupCreator object is moved to myGroup1 and the process is repeated. Once the final cross product has been performed, the createXProdGroup method will update the groupMatrix object stored in myGroup and then call the findGroupName method that is discussed later in this chapter.

The final group generation button is the “User Defined Group” JButton, btnUserDefinedGroup, is associated with a Java ActionListener that determines when the button is pressed and calls the method createUserEntryGroup which is a method in the main panel. The method opens a JOptionPane Input Dialog box requesting the user enter the order of group the user will create. Error checking is done to ensure an integer greater than one is entered. If there is an error at this point, the method returns without changing the currently displayed group. The method in groupCreator used to create an empty group (createEmptyGroup) is called and the empty group is stored in the groupCreator object. The empty group is usually a table filled with the value -1 in each cell. The createUserEntryGroup method will then update the groupMatrix object stored in myGroup while allowing the user the ability to edit the group, unlike the other generation methods. The JLabel above the group, lblGroupName, is then set to “Group Table: User Defined” instead of calling the findGroupName method to set it.

Group Analysis Buttons

The remaining four buttons on the user interface are for analyzing the groups that are on display. They are particularly useful for the user entered groups. The first two buttons determine if the Cayley table that is displayed is actually a group and whether that group is abelian. There are two JLabels that are hidden from the user’s view unless specific information needs to be displayed. This information is usually the results of the analysis JButtons or the methods associated with the analysis JButtons. The JLabel, lblGroupName, is at the top of the user interface and displays the name of the current group being displayed. The JLabel, lblResultsOfAnalysis, is just above all the JButtons and displays the results of the analysis buttons.

The methods associated with the action of the first two buttons directly interact with the groupMatrix object, as that class has characteristic group tests associated with it. The

JButtons Check if Group, btnCheckGroup, and Check if Abelian, btnCheckAbel, are associated with Java ActionListener's that determine when the buttons are pressed and call the methods checkIfGroup and checkIfAbelian. Both methods would first replace the group stored in the groupCreator object with the one being displayed in the groupPanel object in case the group was edited. The methods would then use the groupMatrix object functionality to ensure the group represented by the table had an identity element, had inverses for all the elements, and was associative. The checkIfAbelian method would also ensure the group represented by the table was commutative. If any of these tests failed, the JLabel, lblResultsOfAnalysis would be updated with the cause of the failure in red text. Otherwise, success of the test would be reported via lblResultsOfAnalysis in blue text.

The Find Group Name JButton, btnCheckName, is associated with a Java ActionListener that determines when the button is pressed and calls the method findGroupName. This is the same method called by createZnGroup, createXProdGroup, and createDefinedRelationshipGroup methods. Similar to the checkIfGroup method, this method would first replace the group stored in the groupCreator object with the one being displayed in the groupPanel object in case the group was edited. Unlike Gibbs, the method would also ensure the group represented by the table was a group by determining if it had an identity element, had inverses for all the elements, and was associative. If the table being displayed is not a group the JLabel lblResultsOfAnalysis would be activated with message, "Group Table: This is not a Group." This was because the groupIdentify object has no error checking to determine if a group was a class before it tried to identify the group. The method would then replace the group stored in the groupIdentify object with the new group in the groupCreator object which would start the identification process by groupIdentify object. Once the groupIdentify object was able to determine a name, the JLabel lblGroupName would be set with that name.

The "Find Inner Automorphism" button is associated with a method is associated with a Java ActionListener that determines when the button is pressed and calls the method createInnerAutGroup. Similar to the checkIfGroup method, this method would first replace the group stored in the groupCreator object with the one being displayed in the groupPanel object in case the group was edited. The method would also ensure the group represented by the table was a group by determining if it had an identity element, had inverses for all the

elements, and was associative. If the table displayed is not a group, a JOptionPane with an error message dialog box would be activated as shown in Figure 4.

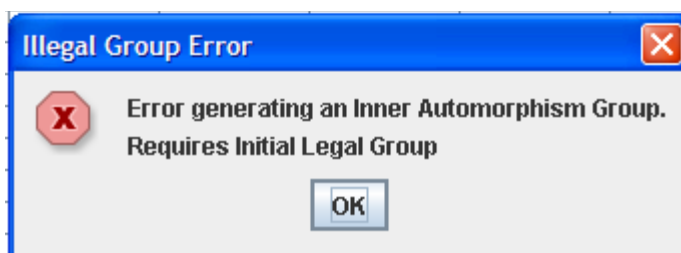


Figure 4. Error message dialog box.

The method in groupCreator used to create an inner automorphism group (createInnerAutGroup) is called and the group is stored in the groupCreator object. Upon successful creation of a group, the createInnerAutGroup method will update the groupMatrix object stored in myGroup and then call the findGroupName method. Finally, the JLabel lblResultsOfAnalysis is also updated and displayed with the name of the group retrieved from the groupIdentify object that was updated by the findGroupName method.

CLASS GROUPPANEL

The secondary panel is a separate class used to display the Cayley table to the user. This class is an extended JPanel class that is called groupPanel. It contains a JTable with a JScrollPane so that the entire JTable can be viewed. Also, the groupPanel object is passed and stores an instance of the groupMatrix object as a private variable myGroup. This object contains the actual group to be displayed. The class groupPanel allows changes to specific cell values in the Cayley Table stored in myGroup via a public method changeValueInGroup. Also, myGroup can be retrieved via the public method getGroup. The class groupTableModel, which is an extended AbstractTableModel, is also a field in groupPanel. When it is instantiated, it is passed myGroup and then set as the JTable's model. It uses the data in myGroup to fill the columns and rows in the JTable.

When a new group is generated via the JButtons on the main applet or frame, the groupPanel's updateTable method is called to reset the myGroup as well as pass it to the groupTableModel. The first column and header of the JTable are then set to different colors because the first column contains the element represented for each of the rows in the Cayley

Table on display. The column is set to a different color via a `DefaultTableCellRenderer`, a class used to displaying individual cells in a `JTable`. The `DefaultTableCellRenderer` is instantiated and its background, foreground, and font are set. This specific renderer is then set as the specific cell renderer for column 0 of the `JTable`.

CLASS GROUPTABLEMODEL

The `groupTableModel` class is an extension to the Java class `AbstractTableModel` and describes how the user interface displays the Cayley Table stored in a `groupMatrix` object. The `groupPanel` object being displayed as part of the user interface instantiated a `groupTableModel` object. The `JTable` in `groupPanel` then uses this local instance of `groupTableModel` as its model for where its data is located and how it is formatted.

The `AbstractTableModel` is an abstract class. An abstract class in Java contains abstract methods or methods that have not been defined. A class that extends an abstract class must implement the abstract methods in the abstract class it is extending in order to be declared legal. The three abstract methods that must be implemented in a class that extends an `AbstractTableModel` are:

- `public int getColumnCount()`
- `public int getRowCount()`
- `public Object getValueAt(int row, int column)`

The `AbstractTableModel` also implements most of the default methods of the `TableModel` interface. The `TableModel` interface provides the other default methods that help to define the way data is displayed in the `JTable`. Nevertheless, any class that extends an `AbstractTableModel` can store its data in any type of data structure or outside source such as a database or the class could generate the data real time. It only needs to be able to implement the `getValueAt` method in order to retrieve the data for a particular position in the `JTable`.

The `groupTableModel` uses a private `ArrayList` of `ArrayList` of strings called `tableRows` to store the data it parses from the `groupMatrix` object that it is passed. This `ArrayList` also represents the Cayley Table; however, an extra column is inserted at the beginning to represent the row names which are the elements in the group. Therefore, the first column has a 0 in the first row, a 1 in the second row, etc., while the column names are

set to 0 for the second column, 1 for the third column, etc. The `groupTableModel` uses a separate private `ArrayList` of strings called `colHeaders` to store the column names. A private boolean value called `canEditTable` that is set based upon whether the `JTable` should be editable, while a private integer is used to store the order of the displayed Cayley Table. The `groupTableModel` class is not passed a `groupMatrix` object when it is instantiated. Therefore, it initially instantiates `tableRows` and `colHeaders` empty so the `JTable` is empty. However, a pointer to the `groupPanel` class that instantiated `groupTableModel` is passed to the `groupTableModel` when it is constructed so that it can pass back changes to the Cayley Table in the `setValueAt` method that is overridden.

When a new group is generated, the `groupPanel`'s `updateTable` method is called by the methods used to generate new groups and is passed the new `groupMatrix` object and a boolean value as to whether the `JTable` should be editable as its parameters. This method in turn calls the `groupTableModel`'s `displayTable` method with the same parameters. The `displayTable` method will set the boolean `canEditTable` with the boolean parameter and order with the order of the Cayley Table stored in the `groupMatrix` object. The `displayTable` method will then fill each `ArrayList` in `tableRows` with the corresponding `ArrayList` from the `groupMatrix` object, adding the extra element to the front of each `ArrayList` that contains the element name from the group corresponding to rows in the Cayley Table.

The abstract methods `getColumnCount` and `getRowCount` retrieve their data from the value of order, where the column count is $order+1$ and row count is order. The `getValueAt` method retrieves the `ArrayList` of strings from `tableRows` corresponding to a specific row and then returns a string corresponding to a specific column in that `ArrayList` as the Object returned. The method `getColumnName` is overridden because the column names, which are the same as the elements in the Cayley Table, are shifted one column over so row names can be placed in the first column.

Since some of the displayed groups can be edited, both the `isCellEditable` and `setValueAt` methods were overridden. The `isCellEditable` returns the value set in the `canEditCell` boolean except in the case of cells in column 0, which can never be edited. The `setValueAt` method is used to update the value stored in `tableRows` when the user edits the `JTable` manually. This method ensures only legal values are set on the Cayley Table, integers from 0 to $order-1$. Also, the `setValueAt` updates the `groupMatrix` object stored in the

groupPanel object that instantiated groupTableModel. This is to ensure that the table can later be analyzed using the analysis functionality in the main frame or applet.

CHAPTER 7

CODE DESCRIPTION

Gibbs' code was transferred from Pascal to Java in order to ease the addition of the user interface described previously as well as enabling the ability to make the code object oriented so that it could be easily extended and different objects could be used in other future applications. The code utilizes a `groupMatrix` object that stores the group as its Cayley Table. This allows the group to be stored independent of the operation that is performed on the elements as well as any relationships between the elements. In addition, the methodologies to analyze the group are also contained in the `groupMatrix` class. Utilizing this single class allowed the code to transfer an object that would contain the group from the generator object (`groupCreator`) to both the user interface object (the `AbstractTableModel` extension `groupTableModel`) and the identifier object (`groupIdentify`) along with the methods those objects use to analyze the group.

CLASS GROUPMATRIX

The `groupMatrix` class contains the order of the group and an `ArrayList` that holds an `ArrayList` of `String` objects as private fields. Each value in the Cayley Table is represented by a single `String` object that is the string representation of an integer value. Each row in the Cayley Table is stored as an `ArrayList` of `String` objects and the entire group is stored as an `ArrayList` of the rows of `ArrayList`s. The `groupMatrix` class has public methods to get and set individual cells in the Cayley Table as well as determine if the current group is equal to another. The class also has public methods that allow the group to be reset to an entirely new group or a new order. The class can be used to determine if the table is an actual group because it has public methods to check for an identity element, an inverse for all elements, and whether the table is associative as well as whether the table is commutative in order to determine if the group is abelian. The class also has public methods to calculate and return the identity element as well as the inverse of any other element. An additional public method

is available that has not been utilized that analyzes the stored Cayley Table to ensure that none of the values in a row or column is repeated.

The determination of the identity element is particularly different because Gibbs' `ident_gp.pas` program assumed that the identity element was always element 0. In order to identify the identity element the process is:

1. Determine the row in the Cayley Table where the row element, e_i , combines with every column element, e_j , such that $e_i \bullet e_j = e_j$. This is the left identity of the Cayley Table.
2. Determine the column in the Cayley Table where the column element, e_j , combines with every row element, e_i , such that $e_i \bullet e_j = e_i$. This is the right identity of the Cayley Table.
3. If the left and right identity elements are equal than the Cayley Table has an identity that is equal to that element.

The identification of the identity element would take, at worst case, about $2n^2$ operations because determining the left and right identities would each take n^2 operations to check every column in every row, and vice versa. Therefore, the total order for identifying the identity element would be $O(n^2)$.

CLASS GROUPCREATOR

The `groupCreator` class is used to generate new groups that can be displayed and analyzed via the user interface. Much of the code for this class was adapted from Gibbs' original Pascal code. The `groupCreator` class has a private field of type `groupMatrix`, labeled `group`, which is either sent to the `groupCreator` directly through the public `resetGroup` method or built in `groupCreator` utilizing one of the group generator methods described below. Once a group is generated, it can be returned through the public `getGroup` method. There are now five group generator methods. The first three, `createCyclicGroup`, `createXProdGroup`, and `createDefineRelationGroup` are all based upon Gibbs' code and generate cyclic, cross product, and defined relationship groups, respectively, using the approach Gibbs used in his original code. The major difference between these methods and Gibbs' generation programs are the number of groups that can be generated. There is also a `createEmptyGroup` method which fills all the cells of a $n \times n$ Table with the value -1, where n is the order of the table to be entered by a user. The final generator method is `createInnerAutGroup` which creates an

inner automorphism group based upon the current instance of group. Descriptions of how each method works is given below.

createCyclicGroup Method

The method `createCyclicGroup` is nearly identical to the code used in Gibbs' `create_zn.pas` program. The limit of order less than 32 for the cyclic groups was removed. The order of the new group to be generated is passed in as a parameter to the method. If the order of group in the current instance of the `groupCreator` class is different, group is reset with the new order utilizing the `groupMatrix resetSize` method. Each entry in group's Cayley Table is then updated utilizing the `setEntry` method with a value corresponding to $(\text{row} + \text{column}) \bmod \text{order}$. A check is then done to ensure all the cells in the Cayley Table have a value before returning.

createDefineRelationshipGroup Method

The `createDefineRelationshipGroup` method is passed an `ArrayList` of generators with their orders, `generatorList`, and an `ArrayList` of `groupRelation` objects, relationships, containing the defined relationships for the group. The orders of all the generators are multiplied to determine the order of group which is then reset to that size. The private method `createFinalElementList` is then called with `generatorList` as its parameter. The method first creates an `ArrayList` of `ArrayList` of `String` objects where `ArrayList` of `Strings` would represent the different generators and the `Strings` in each `ArrayList` would represent the elements that could be created from that generator. Therefore, each row could have a different number of columns. For example, if generator 0 was of order 3, the possible elements would be "", "0", and "00"; whereas if generator 1 was of order 2, the possible generators would be "" and "1". The element from each generator would then be combined in all possible combinations while ensuring generator 0 elements are always before generator 1 elements that are always before generator 2 elements, etc. Therefore, for the two generators given above the final element list would be "", "0", "00", "1", "01", and "001", representing values 0-5 in the final Cayley Table. When the final element list is returned, the `createDefineRelationshipGroup` method uses each of these elements as a row and column in the Cayley Table. For each possible cell in the table, the elements in the table are combined

and they resulting value is reduced using the defined relationships stored in relationships to reduce the results until it equals one of the final elements. The process for reducing the combined elements to a final element is:

1. Combine two elements into a temporary word
2. Begin reduction loop
3. Loop through each groupRelation in relationship
4. Compare left string of current groupRelation to temporary word
5. If left is in temporary word replace left in temporary word in string, replace left in temporary word with right from the groupRelation and go to 2.
6. If another groupRelation is in the ArrayList relationship go to 3.
7. Reduction done, compare final temporary word to element list and set entry in table for combination of the two elements to the element's position in the final element list.

Once all of the possible row, column combinations are calculated, the Cayley Table is complete. A check is done to ensure all the cells in the Cayley Table have a value before returning.

createXProdGroup Method

The createXProdGroup is passed two groupMatrix objects g1 and g2 as parameters. The order of the new group is calculated by multiplying the order of the two parameter groups. The final elements of the group are then determined by combining each of the terms in each of the groups as a combination pair as described in the Historical Description chapter and storing the result in a nx2 integer array, where n is the order of the cross product group. Every combination of the final elements is performed where the result of joining two combination pairs is the result of joining the elements of the pairs separately as also described in the Historical Description chapter. Therefore, every combination of final elements results in another final element. The entry in the Cayley Table for each combination is equal to the position of the resulting final element in the array of final elements.

createInnerAutGroup Method

The final group generator method calculates the inner automorphism of group. The method first calculates the inner automorphism for each element in the group creating at most

n temporary groupMatrix objects. Each temporary groupMatrix object is calculated by the method createInnerAutFromElement which is passed the element to induce the inner automorphism as an integer parameter called element. The temporary groupMatrix is then created by calculating every new element of the Cayley Table by replacing the current value of the table, x, with the value $\text{element} \bullet x \bullet \text{element}^{-1}$. Each of the temporary groupMatrix objects are compared to previous ones created to ensure that they are all unique. Each of the groupMatrix objects is considered a final element in the inner automorphism group and is added to an ArrayList of groupMatrix objects called innerGroups.

The elements are combined using the method createInnerAutFromTwoElements into a temporary groupMatrix object nextGroup. Each combination represents one cell in the Cayley Table where element in the Cayley Table coincides with a groupMatrix object in the ArrayList innerGroups. Each groupMatrix object in the final element list can be represented by an element from the original group. These elements are the elements used to form the inner automorphism induced by that element. The method createInnerAutFromTwoElements is passed the two elements, elmtA and elmtB, representing the groupMatrix objects being combined. A new temporary group is created and returned by the method createInnerAutFromTwoElements by replacing every value x in the current Cayley Table with the value $\text{elmtA} \bullet \text{elmtB} \bullet x \bullet \text{elmtB}^{-1} \bullet \text{elmtA}^{-1}$. The groupMatrix object returned from the method createInnerAutFromTwoElements is then compared to all of the groupMatrix objects stored in innerGroups. The entry in the Cayley Table for the inner automorphism group nextGroup for each combination is equal to the position of the resulting final element in the ArrayList of final elements. The instance of group is then updated to the inner automorphism group, groupNext which is used to update the display and is identified by the method that called createInnerAutGroup.

CLASS GROUPIDENTIFY

The groupIdentify class identifies the group that is passed to the class via a groupMatrix object. Most of the code for this class was adapted directly from Gibbs' original Pascal code and the methods in the class mirror most of the procedures and functions in Gibbs' original code. The method for identifying groups mirrors the five step process outlined in the Historical Description process. Some of the major changes to the code

involve the use of the `groupMatrix` object and its analysis functionality particularly its ability to determine if a group is commutative, identification of the identity element, and identification of a specific element's inverse.

The identification process is also more generic so that more isomorphisms could be identified. In particular, the identification of the groups no longer requires the identity element be element 0 and many of the non-abelian groups of order 32 were added utilizing their Hall-Senior numbers. In Gibbs' code, the procedures to calculate the order structure of the group, to calculate the centers of the group, and to determine the normality of the subgroups specifically fixed the identity element as 0. The code for the `groupIdentify` class utilized the `groupMatrix` method `findIdentity` method to make the process more generic. Once the identity of the group is identified, there were two possible avenues to adjust the Gibbs' code in order to use any element as the identity. The first method would replace every element in the zeroth row and column with the corresponding element in the identity elements row and column. Thus, if the identity element was e_i , every element in row e_i would be swapped with every element in row 0, as well as every element in column e_i would be swapped with every element in column 0. This would create a group isomorphic to the original group, with the identity element as the zeroth element. Swapping the rows and columns would take an additional $6n$ operations because each swap takes 3 operations and there are n row elements and n column elements to swap. Thus, because the `findIdentity` method runs in $O(n^2)$, the overall operation also runs in $O(n^2)$. The advantage of using this method is the remainder of Gibbs' code could be used to further identify the group. However, by just using the `findIdentity` method and adjusting Gibbs' code to use the results of the `findIdentity` method, the group table did not have to be changed during the identification process.

Once a group has been identified, its name is stored in a private `String` field called `name` that can be retrieved with the method `getName`. If the group cannot be identified, an error message is stored in `name` and a private boolean field, `identified`, is set to `false` to indicate the identification routine failed. The results stored in `identified` can be retrieved via the method `isIdentified`.

Appendix B, Cayley Table Code, is the code for the entire project. It contains more information about each of the classes along with their fields and methods.

CHAPTER 8

ADDITIONAL WORK

Another class was created in order to analyze the creation of groups via brute force and determine the number of isomorphisms there are when the identity is fixed as element 0. For example, the four Cayley Tables of order 3, when the identity is fixed at element 0, are shown in Figure 5.

	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

	0	1	2
0	0	1	2
1	1	0	0
2	2	2	1

	0	1	2
0	0	1	2
1	1	2	1
2	2	0	0

	0	1	2
0	0	1	2
1	1	0	1
2	2	2	0

Figure 5. Tables of order 3 with identity fixed at element 0.

However, only the table in the upper left hand corner is an actual group (Z_3). The other tables, while each possesses an identity element, are not groups because they are all not associative. For example, in the table in the upper right hand corner, the value of $(1 \bullet 2) \bullet 2 = 2$, while the value of $1 \bullet (2 \bullet 2) = 0$.

The class `groupPermutation` was created to determine the number of Cayley Tables that were isomorphic to groups of a specific order. Testing occurred for groups up to order 6. Further testing could not be performed on higher order systems because there were $(n-1)!(n-1)$ tables generated and analyzed for each order where n is the order of the group. Therefore, while for order 3, there were $(3-1)!(3-1) = 4$ total tables to generate and analyze, for order 6 there were $(6-1)!(6-1) = 24,883,200,000$ tables to generate and analyze.

The tables were built by generating the $n!$ permutations of a set of n numbers and using each of those sets of permutations in each row of the table. This would fix the total number of tables to be generated at $n!^n$. However, only $(n-1)$ elements needed to be permuted for groups of order n because the identity element was kept fixed at element 0.

Therefore, each row needed $(n-1)!$ permutations generated for the remaining $n-1$ elements in each row. Also, there were now only $(n-1)$ total rows that required the permutations, since the first row was fixed due to the identity, the total number of tables that needed to be generated was reduced to $(n-1)!(^{n-1})$. Another methods used to reduce the number of tables generated was to ensure the columns contained elements as each row was added to the table. Nevertheless, the number of tables to analyze and identify as groups increased dramatically for every incremental increase in order of the groups. All groups of the same order were analyzed and identified in one run. Therefore, the results for groups Z_4 and $Z_2 \oplus Z_2$ were attained during the same test run of the class `groupPermutation`, while the results for the groups Z_6 and D_3 were also gathered at the same time. The results for the groups of order less than six are shown in Table 3.

Table 3. Number of Cayley Tables for Groups of Order < 6

Group Name	Order	Total Number of Cayley Tables Generated	Number of Cayley Tables
Z_3	3	4	1
Z_4	4	216	3
$Z_2 \oplus Z_2$, Klein-4	4	216	1
Z_5	5	331,776	6
Z_6	6	24,883,200,000	60
D_3, S_3	6	24,883,200,000	20

The class `groupPermutation` used the class `PermutationGenerator`²² to identify and create an `ArrayList` of n `PermutationGenerator` objects that contained all permutations of the integers 0 to $n-1$, where n is the order of the group. The class `groupPermutation` also contains a private `groupMatrix` object, `myGroup`, to store the current Cayley Table being generated and a private `groupIdentify` object, `groupNamer`, to identify the name of the Cayley Table stored in `myGroup` if it contains a group. Another private integer, `groupCount`, is also a field in `groupPermutation` that is used to keep track of the total number of groups identified

²² Gilleland, Michael. "Permutation Generator." [<http://www.merriampark.com/perm.htm>]. March 2005.

for a particular order. Once the ArrayList of PermutationGroup objects is created, a method in groupPermutation is then called recursively to fill myGroup row by row. The method's parameters are the ArrayList of PermutationGroup objects and the number of rows in myGroup to fill. Each row utilizes one of the PermutationGenerator objects and loops until all of the permutations generated by that object are tried. After all the rows are filled, the groupMatrix analysis methods are checked to determine if it contains a group by ensuring the Cayley Table stored in myGroup had an identity element, had inverses for all the elements, and was associative. If myGroup contained a group, it was then passed to groupNamer to be identified and the results were output to the screen.

Some short cuts were developed to reduce the total number of Cayley Tables that had to be checked. The first row was always kept as the identity and therefore, no other permutations were tried. The permutations for each row had to start with the row number in order to ensure column 0 contained the identity element. All other permutations for that row were skipped before filling in additional rows. Also, as each new row was added, a check was performed to ensure that no element is repeated in a row. If a column has an element repeated, the last row is removed and the next permutation for that row is tried.

Due to the number of permutations of Cayley Tables that are possible only tables up to order 6 were tried. It took 26 minutes for all permutations of groups of order 6 with the identity fixed at element 0 to be found. In contrast, it took less than 1 second for all permutations of groups of order 5 with the identity fixed at element 0 to be found.

CHAPTER 9

FUTURE WORK

There are two main areas of focus for future work: expansion of the code, its capabilities, and usefulness and further analysis of groups and group theory to expand the current capability of the code to identify more groups.

Currently, the code has basic analysis functionality to determine if a Cayley Table stored in a `groupMatrix` object contains a group, if the elements commute (i.e. the group is Abelian), as well as the ability to determine the inner automorphism of a group. Expanding these capabilities with new functionality could quickly yield the ability to identify the center of a group as well as determine the subgroups of a group and if a subgroup is normal. Algorithms for these functions are already partially written in the `groupIdentify` code and used to analyze some of the non-Abelian groups of order 16. For example, the method `computeCenter` in the class `groupIdentify` already determines the elements of a group that are in the center of the group by calculating which elements commute with all other elements in a group. Based upon the elements found to be in the center and the current Cayley Table an algorithm to generate the center of a group could quickly be added to the code and used for further analysis of higher order groups.

An algorithm to determine the automorphism group of a group G , $\text{Aut}(G)$, would also be useful. Further investigation would be necessary to expand the capabilities of the `groupCreator` class to generate the automorphism group; however, automorphism groups are important factors in determining semi-direct products of a group as well as useful in further study abstract algebra concepts such as rings and fields.

Expanding the capability of the class `groupIdentify` to identify groups with orders similar to the functionality that tests for groups of order $2*p$ and p^2 . This would yield quicker analysis of some groups than the current methodology for determining groups that are non-Abelian which focuses on order structure of the groups. This is because the differentiation of the groups by order structure breaks down at groups of order 16, and totally falls apart with groups of order 32 in which there are order structures that define six groups

that are isomorphically distinct. One example of a new analysis method focuses on the theorem, if group G is of the order $|pq|$, where p, q are prime and $p < q$, G is isomorphic to either Z_{pq} or if $p \mid (q-1)$ then G could be isomorphic to $Z_p \times Z_q$.²³ Since the first step of the groupIdentify identification process is to check for cyclic groups, this method could be used to determine specific non-Abelian groups before checking if the group is Abelian. For the groups of order less than 32, this theorem works for not only the groups of order $2 \cdot p$, but the group $Z_3 \times Z_7$ which is of order 21, while it shows why there is only the cyclic group for order 15.

Further group theory work is also necessary to complete and expand the list of groups that can be generated and identified. Determining all generators and relationships for the non-Abelian groups of order 32 will require a much greater understanding of group theory; yet, remains an ideal goal to achieve. Also, distinguishing between the groups and identifying the groups of order 32 will be a necessary feature of the groupIdentify class to complete. While some groups of order 32 can now be identified, the current identification utilizes the numbering system of Hall and Senior²⁴ to identify the classes with unique order structure. Connecting the groups of order 32 with names based upon cross products or group characteristics (e.g. dihedral and dicyclic) remains to be done. Also, only 10 of the 45 non-Abelian groups had a unique order structure that allow for identification based upon the techniques used by Gibbs. When the order structure and family number from Hall and Senior²⁵ were combined to determine a group, 18 out of the total 45 non-Abelian groups could be identified. Interestingly, one of the characteristics of groups with the same family number was that they had the same inner automorphism group. Utilizing the inner automorphism group along with new functionality that could quickly be added to the Cayley Table code, such as group center and normality of subgroups, a larger number of the groups of order 32 should be easily identified.

Nevertheless, much work still remains because the order structure, center, inner automorphism, and determination of the normality of subgroups does not necessarily mean

²³ Charles Lanski, *Concepts in Abstract Algebra*, (Belmont, CA: Brooks Cole, 2005), 287.

²⁴ Marshall Hall, Jr and James K. Senior, *The Groups of Order 2^n ($n \leq 6$)*, (New York: Macmillan, 1964).

²⁵ Marshall Hall, Jr and James K. Senior, *The Groups of Order 2^n ($n \leq 6$)*, (New York: Macmillan, 1964).

two groups are isomorphic. For example, the group formed by the cross product of $Z_2 \oplus Z_2 \oplus \text{Quaternion}$, the group formed by the defined relationships $a^8 = b^4 = \lambda$ and $ba = a^3b$ and the group formed by the defined relationships $a^8 = b^4 = \lambda$ and $ba = a^{-1}b$, all form groups whose order structure (3 elements of order 2, 20 elements of order 4, and 8 elements of order 8), center group ($Z_2 \oplus Z_2$), and inner automorphism (D_4) are identical as well as have subgroups that are not normal. However, their Cayley Tables are very different. The result is that we do not know if these groups are isomorphic. This is made even more complicated because there are actually four groups through isomorphism with that order structure and inner automorphism.²⁶ Functionality that tests two groups of the same order to determine if they are isomorphic would be extremely useful in determining if new defined relationships for higher order groups define groups which have not been identified.

²⁶ Marshall Hall, Jr and James K. Senior, *The Groups of Order 2^n ($n \leq 6$)*, (New York: Macmillan, 1964).

REFERENCES

- Coxeter, H.S.M. and Moser, W.O.J. *Generators and Relations for Discrete Groups*. 4th ed. New York: Springer-Verlag, 1980.
- Gallian, Joseph. *Contemporary Abstract Algebra*. 5th ed. New York: Houghton Mifflin Co., 2001
- Gibbs, David. "Computer Generation and Identification of Groups of Order 2 to 31." Unpublished Math 797 Project, San Diego State University, 1984.
- Gilbert, Jimmie and Gilbert, Linda. *Elements of Modern Algebra*. 5th ed. Pacific Grove, CA: Brooks Cole, 2000.
- Gilleland, Michael. "Permutation Generator." [<http://www.merriampark.com/perm.htm>]. March 2005.
- Hall, Marshall, Jr. and Senior, James K. *The Groups of Order 2^n ($n \leq 6$)*. New York: Macmillan, 1964.
- Lanski, Charles. *Concepts in Abstract Algebra*. Belmont, CA: Brooks Cole, 2005.
- Valero-Elizondo, Luis. "The Mod-2 Cohomology of 2-Groups (by Jon Carlson)." [<http://www.math.uga.edu/~lvalero/cohointro.htmlz>]. May 2001.
- Wavrik, John J. "Groups15 and Groups32." [http://www.math.ucsd.edu/~jwavrik/G15_G32.html]. September 2003.
- Weisstein, Eric W. "Mathworld—A Wolfram Web Resource." [<http://mathworld.wolfram.com/>]. March 2005.

APPENDIX A**ALL GROUPS OF ORDER 2 TO 31 AND SOME
GROUPS OF ORDER 32**

Table 4. All Groups of Order 2 to 31 and Some Groups of Order 32^{27,28}

Order	Group Name	Abelian	Defining Relations
2	Z_2	Yes	$a^2 = \lambda$
3	Z_3	Yes	$a^3 = \lambda$
4	Z_4	Yes	$a^4 = \lambda$
	$Z_2 \oplus Z_2$, Klein-4	Yes	$a^2 = b^2 = \lambda, ba=ab$
5	Z_5	Yes	$a^5 = \lambda$
6	Z_6	Yes	$a^6 = \lambda$
	D_3, S_3	No	$a^3 = b^2 = \lambda, ba=a^{-1}b$
7	Z_7	Yes	$a^7 = \lambda$
8	Z_8	Yes	$a^8 = \lambda$
	$Z_4 \oplus Z_2$	Yes	$a^4 = b^2 = \lambda, ba=ab$
	$Z_2 \oplus Z_2 \oplus Z_2$	Yes	$a^2 = b^2 = c^2 = \lambda, ba=ab, ca=ac, cb=bc$
	D_4	No	$a^4 = b^2 = \lambda, ba=a^{-1}b$
	$\langle 2,2,2 \rangle$ or Quaternion	No	$a^4 = \lambda, b^2 = a^2, ba=a^{-1}b$
9	Z_9	Yes	$a^9 = \lambda$
	$Z_3 \oplus Z_3$	Yes	$a^3 = b^3 = \lambda, ba=ab$
10	Z_{10}	Yes	$a^{10} = \lambda$
	D_5	No	$a^5 = b^2 = \lambda, ba=a^{-1}b$
11	Z_{11}	Yes	$a^{11} = \lambda$
12	Z_{12}	Yes	$a^{12} = \lambda$
	$Z_6 \oplus Z_2$	Yes	$a^6 = b^2 = \lambda, ba=ab$
	D_6	No	$a^6 = b^2 = \lambda, ba=a^{-1}b$
	A_4	No	$a^2 = b^2 = c^3 = \lambda, ba=ab, ca = bc, cb = abc$

²⁷ David Gibbs, "Computer Generation and Identification of Groups of Order 2 to 31" (unpublished Math 797 Project, San Diego State University, 1984), 5-9,13-15

²⁸ H.S.M. Coxeter and W.O.J. Moser, *Generators and Relations for Discrete Groups*, 4th ed. (New York: Springer-Verlag, 1980), 134-135.

	$\langle 2,2,3 \rangle$	No	$a^3 = b^4 = \lambda, ba=a^{-1}b$
13	Z_{13}	Yes	$a^{13} = \lambda$
14	Z_{14}	Yes	$a^{14} = \lambda$
	D_7	No	$a^7 = b^2 = \lambda, ba=a^{-1}b$
15	Z_{15}	Yes	$a^{15} = \lambda$
16	Z_{16}	Yes	$a^{16} = \lambda$
	$Z_8 \oplus Z_2$	Yes	$a^8 = b^2 = \lambda, ba=ab$
	$Z_4 \oplus Z_4$	Yes	$a^4 = b^4 = \lambda, ba=ab$
	$Z_4 \oplus Z_2 \oplus Z_2$	Yes	$a^4 = b^2 = c^2 = \lambda, ba=ab, ca=ac, cb=bc$
	$Z_2 \oplus Z_2 \oplus Z_2 \oplus Z_2$	Yes	$a^4 = b^2 = c^2 = d^2 = \lambda, ba=ab, ca=ac, da=ad, cb=bc, db=bd, dc=cd$
	$Z_2 \oplus D_4$	No	cross product
	$Z_2 \oplus \text{Quaternion}$	No	cross product
	D_8	No	$a^8 = b^2 = \lambda, ba=a^{-1}b$
	$\langle 2, 2, 4 \rangle$ or Q_4	No	$a^8 = \lambda, b^2 = a^4, ba=a^{-1}b$
	$Z_4 \times Z_4$	No	$a^4 = b^4 = \lambda, ba=a^{-1}b$
	$Z_8 \times Z_2$	No	$a^8 = b^2 = \lambda, ba=a^5b$
	$Z_8 \times Z_2$	No	$a^8 = b^2 = \lambda, ba=a^3b$
	Weird1	No	$a^4 = b^2 = c^2 = \lambda. ba=ab, ca=ac, cb=aabc$
	Weird1	No	$a^4 = b^2 = c^2 = \lambda. ba=ab, ca=abc^{-1}, cb=bc$
17	Z_{17}	Yes	$a^{17} = \lambda$
18	Z_{18}	Yes	$a^{18} = \lambda$
	$Z_6 \oplus Z_3$	Yes	$a^6 = b^3 = \lambda, ba=ab$
	$Z_3 \oplus D_3$	No	cross product
	D_9	No	$a^9 = b^2 = \lambda, ba=a^{-1}b$
	$\langle\langle 3,3,3;2 \rangle\rangle$	No	$a^3 = b^3 = c^2 = \lambda. ba=ab, ca=a^{-1}, cb=b^{-1}c$
19	Z_{19}	Yes	$a^{19} = \lambda$
20	Z_{20}	Yes	$a^{20} = \lambda$

	$Z_{10} \oplus Z_2$	Yes	$a^{10} = b^2 = \lambda, ba=ab$
	D_{10}	No	$a^{10} = b^2 = \lambda, ba=a^{-1}b$
	K-Metacyclic 20	No	$a^5 = b^4 = \lambda, ba=aab$
	$\langle 2,2,5 \rangle$	No	$a^5 = b^4 = \lambda, ba=a^{-1}b$
21	Z_{21}	Yes	$a^{21} = \lambda$
	$Z_7 \times_0 Z_3$	No	$a^7 = b^3 = \lambda, ba=aab$
22	Z_{22}	Yes	$a^{22} = \lambda$
	D_{11}	No	$a^{11} = b^2 = \lambda, ba=a^{-1}b$
23	Z_{23}	Yes	$a^{23} = \lambda$
24	Z_{24}	Yes	$a^{24} = \lambda$
	$Z_{12} \oplus Z_2$	Yes	$a^{12} = b^2 = \lambda, ba=ab$
	$Z_6 \oplus Z_2 \oplus Z_2$	Yes	$a^6 = b^2 = c^2 = \lambda, ba=ab, ca=ac, cb=bc$
	$A_4 \oplus Z_2$	No	Cross Product
	$D_6 \oplus Z_2$	No	Cross Product
	$D_4 \oplus Z_3$	No	Cross Product
	Quaternion $\oplus Z_3$	No	Cross Product
	$D_3 \oplus Z_4$	No	Cross Product
	$\langle 2,2,3 \rangle \oplus Z_2$	No	Cross Product
	D_{12}	No	$a^{12} = b^2 = \lambda, ba=a^{-1}b$
	S_4	No	$a^2 = b^2 = c^3 = d^2 = \lambda, ba=ab, ca=abc,$ $da=ad, cb=ac, db=abd, dc=c^{-1}d$
	$\langle 2,3,3 \rangle$	No	$a^4 = c^3 = \lambda, b^2=a^2, ba=a^{-1}b, ca=bc, cb=abc$
	$\langle 4, 6 \mid 2, 2 \rangle$	No	$a^4 = b^6 = \lambda, ba^{-1}=ab^{-1}, ba=a^{-1}b^{-1}$
	$\langle -2, 2, 3 \rangle$	No	$a^3 = b^8 = \lambda, ba=a^{-1}b$
	$\langle 2, 2, 6 \rangle$	No	$a^{12} = \lambda, b^2 = a^6, ba=a^{-1}b$
25	Z_{25}	Yes	$a^{25} = \lambda$
	$Z_5 \oplus Z_5$	Yes	$a^5 = b^5 = \lambda, ba=ab$
26	Z_{26}	Yes	$a^{26} = \lambda$

	D_{13}	No	$a^{13} = b^2 = \lambda, ba=a^{-1}b$
27	Z_{27}	Yes	$a^{27} = \lambda$
	$Z_9 \oplus Z_3$	Yes	$a^9 = b^3 = \lambda, ba=ab$
	$Z_3 \oplus Z_3 \oplus Z_3$	Yes	$a^3 = b^3 = c^3 = \lambda, ba=ab, ca=ac, cb=bc$
	$\langle 3, 3 \mid 3, 3 \rangle$	No	$a^3 = b^3 = c^3 = \lambda, ba=ab, ca=ac, cb=aabbc$
	Weird27	No	$a^9 = b^3 = \lambda, ba=aaaab$
28	Z_{28}	Yes	$a^{28} = \lambda$
	$Z_{14} \oplus Z_2$	Yes	$a^{14} = b^2 = \lambda, ba=ab$
	D_{14}	No	$a^{14} = b^2 = \lambda, ba=a^{-1}b$
	$\langle 2, 2, 7 \rangle$	No	$a^7 = b^4 = \lambda, ba=a^{-1}b$
29	Z_{29}	Yes	$a^{29} = \lambda$
30	Z_{30}	Yes	$a^{30} = \lambda$
	$D_5 \oplus Z_3$	No	Cross Product
	$D_3 \oplus Z_5$	No	Cross Product
	D_{15}	No	$a^{15} = b^2 = \lambda, ba=a^{-1}b$
31	Z_{31}	Yes	$a^{31} = \lambda$
32	Z_{32}	Yes	$a^{32} = \lambda$
	$Z_{16} \oplus Z_2$	Yes	$a^{16} = b^2 = \lambda, ba=ab$
	$Z_8 \oplus Z_4$	Yes	$a^8 = b^4 = \lambda, ba=ab$
	$Z_8 \oplus Z_2 \oplus Z_2$	Yes	$a^8 = b^2 = c^2 = \lambda, ba=ab, ca=ac, cb=bc$
	$Z_4 \oplus Z_4 \oplus Z_2$	Yes	$a^4 = b^4 = c^2 = \lambda, ba=ab, ca=ac, cb=bc$
	$Z_4 \oplus Z_2 \oplus Z_2 \oplus Z_2$	Yes	$a^4 = b^2 = c^2 = d^2 = \lambda, ba=ab, ca=ac,$ $da=ad, cb=bc, db=bd, dc=cd$
	$Z_2 \oplus Z_2 \oplus Z_2 \oplus Z_2 \oplus Z_2$	Yes	$a^2 = b^2 = c^2 = d^2 = f^2 = \lambda, ba=ab, ca=ac,$ $da=ad, fa=af, cb=bc, db=bd, fb=bf,$ $dc=cd, fc=cf, fd=df$
	$Z_2 \oplus Z_2 \oplus D_4$	No	Cross Product
	$Z_2 \oplus Z_2 \oplus \text{Quaternion}$	No	Cross Product

	$Z_2 \oplus D_8$	No	Cross Product
	$Z_2 \oplus Q_4$	No	Cross Product
	$Z_2 \oplus Z_4 \times Z_4$	No	Cross Product
	$Z_2 \oplus Z_8 \times Z_2$	No	Cross Product
	$Z_2 \oplus Z_8 \times Z_2$	No	Cross Product
	$Z_2 \oplus \text{Weird1}$	No	Cross Product
	$Z_2 \oplus \text{Weird1}$	No	Cross Product
	$Z_4 \oplus D_4$	No	Cross Product
	$Z_4 \oplus \text{Quaternion}$	No	Cross Product
	D_{16}	No	$a^{16} = b^2 = \lambda, ba = a^{-1}b$
	$\langle 2, 2, 8 \rangle$	No	$a^{16} = \lambda, b^2 = a^8, ba = a^{-1}b$
	$Z_{16} \times Z_2$	No	$a^{16} = b^2 = \lambda, ba = a^9b$
	$Z_{16} \times Z_2$	No	$a^{16} = b^2 = \lambda, ba = a^7b$

APPENDIX B

CODE

MAIN.JAVA

```

/*
 * Main.java
 *
 * Created on January 16, 2005, 1:12 PM
 */

package cayleytable;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
/**
 * Main class for use in running the Cayley Table generation and
 * identification software from a command line.
 * @author Jeffrey Barr
 */
public class Main {

    /**
     * Main class instantiates a <CODE>groupMainFrame</CODE> class to open
     * the user interface via a call to the Main class when the jar file
     is
     * added to the class path.
     */
    public Main() {
        groupMainFrame app = new groupMainFrame();
        app.addWindowListener(new WindowAdapter()
            {
                public void windowClosing( WindowEvent e )
                {
                    System.exit(0);
                }
            }
        );
    }
    /**
     * Main class instantiates a <CODE>groupMainFrame</CODE> class to open
     * the user interface via a call from the command line.
     * @param args Command line arguments
     */
    public static void main(String[] args) {
        groupMainFrame app = new groupMainFrame();
        app.addWindowListener(new WindowAdapter()
            {
                public void windowClosing( WindowEvent e )
                {
                    System.exit(0);
                }
            }
        );
    }
}

```

```

        );
    }
}

```

GROUPMAIN.JAVA

```

/*
 * groupMain.java
 *
 * Created on January 16, 2005, 1:19 PM
 */

package cayleytable;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
/**
 * Main Frame of Cayley table viewer for use when tool is run as a
 * standalone program. See <CODE>groupMain</CODE> for identical version
 * in applet format. User interface that contains <CODE>groupPanel</CODE>
 * that displays Cayley Table and buttons to generate and analyze a
 * Cayley Table.
 * @author Jeffrey Barr
 */
public class groupMain extends javax.swing.JApplet {

    /**
     * Initializes the applet <CODE>groupMainFrame</CODE> through call to
     initialize all
     * of the components in the applet.
     */
    public void init() {
        try {
            java.awt.EventQueue.invokeAndWait(new Runnable() {
                public void run() {
                    initComponents();
                }
            });
        } catch (Exception ex) {
            ex.printStackTrace();
            System.out.println("Error: " + ex.getLocalizedMessage());
        }
    }

    /**
     * This method is called from within the <CODE>init()</CODE>method to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.

```

```

* Code generated via Netbeans.
*/
private void initComponents() { //GEN-BEGIN:initComponents
    java.awt.GridBagConstraints gridBagConstraints;

    groupNamer = new cayleytable.groupIdentify();
    myCreator = new cayleytable.groupCreator();
    lblGroupName = new javax.swing.JLabel();
    myGroup = new cayleytable.groupPanel();
    lblResultsOfAnalysis = new javax.swing.JLabel();
    lblPropertyButtons = new javax.swing.JLabel();
    btnCheckGroup = new javax.swing.JButton();
    btnCheckAbel = new javax.swing.JButton();
    btnCheckName = new javax.swing.JButton();
    btnInnerAut = new javax.swing.JButton();
    lblGeneratorButtons = new javax.swing.JLabel();
    btnZnGroup = new javax.swing.JButton();
    btnXProdGroup = new javax.swing.JButton();
    btnUserDefinedGroup = new javax.swing.JButton();
    btnDefRelnGroup = new javax.swing.JButton();

    getContentPane().setLayout(new java.awt.GridBagLayout());

    lblGroupName.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
    lblGroupName.setText(groupNamer.getName());
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 0;
    gridBagConstraints.gridwidth =
java.awt.GridBagConstraints.REMAINDER;
    gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
    gridBagConstraints.weightx = 1.0;
    getContentPane().add(lblGroupName, gridBagConstraints);

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 1;
    gridBagConstraints.gridwidth =
java.awt.GridBagConstraints.REMAINDER;
    gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
    gridBagConstraints.weighty = 1.0;
    getContentPane().add(myGroup, gridBagConstraints);

    lblResultsOfAnalysis.setHorizontalAlignment(javax.swing.SwingConstants.CEN
TER);
    lblResultsOfAnalysis.setEnabled(false);
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 11;
    gridBagConstraints.gridwidth =
java.awt.GridBagConstraints.REMAINDER;
    gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
    gridBagConstraints.weightx = 1.0;
    getContentPane().add(lblResultsOfAnalysis, gridBagConstraints);

```

```

lblPropertyButtons.setHorizontalAlignment(javax.swing.SwingConstants.LEFT)
;
    lblPropertyButtons.setText("Check Group Properties:");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 12;
    gridBagConstraints.gridwidth =
java.awt.GridBagConstraints.RELATIVE;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(lblPropertyButtons, gridBagConstraints);

    btnCheckGroup.setText("Check if Group");
    btnCheckGroup.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            checkIfGroup(evt);
        }
    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 13;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(btnCheckGroup, gridBagConstraints);

    btnCheckAbel.setText("Check if Abelian");
    btnCheckAbel.addActionListener(new java.awt.event.ActionListener()
{
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            checkIfAbelian(evt);
        }
    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 13;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(btnCheckAbel, gridBagConstraints);

    btnCheckName.setText("Find Group Name");
    btnCheckName.addActionListener(new java.awt.event.ActionListener()
{
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            findGroupName(evt);
        }
    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 13;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(btnCheckName, gridBagConstraints);

    btnInnerAut.setText("Find Inner Automorphism");
    btnInnerAut.addActionListener(new java.awt.event.ActionListener()
{
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            createInnerAutGroup(evt);
        }
    });

```

```

    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 13;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(btnInnerAut, gridBagConstraints);

lblGeneratorButtons.setHorizontalAlignment(javax.swing.SwingConstants.LEFT
);
    lblGeneratorButtons.setText("Choose Type of Group to Enter:");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 14;
    gridBagConstraints.gridwidth =
java.awt.GridBagConstraints.RELATIVE;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(lblGeneratorButtons, gridBagConstraints);

    btnZnGroup.setText("Cyclic Group");
    btnZnGroup.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            createZnGroup(evt);
        }
    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 15;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(btnZnGroup, gridBagConstraints);

    btnXProdGroup.setText("Cross Product Group");
    btnXProdGroup.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            createXProdGroup(evt);
        }
    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 15;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(btnXProdGroup, gridBagConstraints);

    btnUserDefinedGroup.setText("User Defined Group");
    btnUserDefinedGroup.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            createUserEntryGroup(evt);
        }
    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 15;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;

```

```

        getContentPane().add(btnUserDefinedGroup, gridBagConstraints);

        btnDefRelnGroup.setText("Defined Relationship Group");
        btnDefRelnGroup.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                createDefinedRelationshipGroup(evt);
            }
        });

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridy = 15;
        gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
        getContentPane().add(btnDefRelnGroup, gridBagConstraints);

    } //GEN-END:initComponents

    /**
     * Method to create the groups based upon a defined relationship calls
     * the <CODE>groupCreator</CODE> object functionality to create and
     store the group.
     *
     * The group is named by <CODE>groupNamer</CODE>
     <CODE>groupIdentify</CODE> object after creation.
     * @param evt Launched by the push of <CODE>btnDefRelnGroup</CODE>.
     */
    private void createDefinedRelationshipGroup(java.awt.event.ActionEvent
    evt) { //GEN-FIRST:event_createDefinedRelationshipGroup
        ArrayList<String> generatorList = new ArrayList<String>();
        ArrayList<groupRelation> definedRelationships = new
        ArrayList<groupRelation>();

        String defRelName = "Group Table:  Defined Relationship";

        // Determine number of generators to be used and error check that it
        is a legal value
        String defRelString = JOptionPane.showInputDialog(null,
            "Enter the number of generators
in the defined relationship. ",
            "Defined Relationship Dialog",
            JOptionPane.QUESTION_MESSAGE);

        int defRelNumOfGenerators = -1;
        int numRelationships;
        try
        {
            defRelNumOfGenerators = Integer.parseInt(defRelString, 10);
        }
        catch (NumberFormatException e)
        {
            JOptionPane.showMessageDialog(null,
                defRelString + " is not a legal
number!" +
                "\nError: " + e.getMessage(),
                "Number Error",
                JOptionPane.ERROR_MESSAGE);
            blnGroupCreated = false;

```

```

        return;
    }

    numRelationships = factorial(defRelNumOfGenerators);
    System.out.println("Defining Relationships");
    if (defRelNumOfGenerators < 2)
    {
        JOptionPane.showMessageDialog(null,
            "You cannot create a defined
relationship with less" +
            " than two groups.", "Defined
Relationship Error",
            JOptionPane.ERROR_MESSAGE);
        blnGroupCreated = false;
        return;
    }

    // Determine the number of elements in each generator
    String elementString;
    for (int n=0; n<defRelNumOfGenerators; n++)
    {
        elementString = JOptionPane.showInputDialog(null,
            "Enter the number of
elements in generator " + n,
            "Element Query Dialog",
            JOptionPane.QUESTION_MESSAGE);

        // Ensure element string is an integer
        try
        {
            Integer.parseInt(elementString, 10);
        }
        catch (NumberFormatException e)
        {
            JOptionPane.showMessageDialog(null,
                elementString + " is not a legal
number!" +
                "\nError: " + e.getMessage(),
                "Number Error",
                JOptionPane.ERROR_MESSAGE);
            n = n-1;
            continue;
        }

        generatorList.add(elementString);
        definedRelationships.add(new groupRelation(n,
            Integer.parseInt(elementString,10)));
    }

    boolean extraRelation = true;
    while (extraRelation)
    {
        String question = JOptionPane.showInputDialog(null,
            "Do you have an extra
relationship between generators?\n"

```

```

+ " Enter yes/no. ",
"Relationship Dialog",

JOptionPane.QUESTION_MESSAGE);
    question = question.toLowerCase();
    if (question.compareTo("yes") == 0 || question.compareTo("y")
== 0)
        {
            String relationship = JOptionPane.showInputDialog(null,
relationship.",
Dialog",
"Relationship
JOptionPane.QUESTION_MESSAGE);
            groupRelation extra = new groupRelation(relationship,
generatorList);
            boolean foundMatch = false;
            for (int i=0; i<definedRelationships.size(); i++)
                {
                    groupRelation tempRel = (groupRelation)
definedRelationships.get(i);
                    if (extra.getLeft().compareTo(tempRel.getLeft())
== 0)
                        {
                            definedRelationships.set(i, extra);
                            foundMatch = true;
                        }
                }
            if (!foundMatch) definedRelationships.add(extra);
        }
    else
        extraRelation = false;
    }

    for (int i=0; i<defRelNumOfGenerators-1; i++)
        {
            for (int j=i+1; j<defRelNumOfGenerators; j++)
                {
                    definedRelationships.add(new groupRelation(j, i,
getNextRelationship(i,j), generatorList));
                }
        }

    System.out.println("Relationships Defined");
    for (int j=0; j<definedRelationships.size(); j++)
        {
            groupRelation temp = (groupRelation)
definedRelationships.get(j);
            System.out.println(j + ": " + temp.getLeft() + " = " +
temp.getRight());
        }

    System.out.println("Relationships Defined twice");

```

```

        if (myCreator.createDefineRelationGroup(generatorList,
definedRelationships))
        {
            myGroup.updateTable(myCreator.getGroup(), false);
            System.out.println("Attempting to identify group");
            findGroupName(evt);
            lblResultsOfAnalysis.setEnabled(false);
            lblResultsOfAnalysis.setText("");
            blnGroupCreated = true;
        }
        else
        {
            lblResultsOfAnalysis.setEnabled(true);
            lblResultsOfAnalysis.setBackground(Color.white);
            lblResultsOfAnalysis.setForeground(Color.RED);
            lblResultsOfAnalysis.setText("Error: No Defined Relationship
Group Found");
            System.out.println("No Defined Relationship Group Found");
            blnGroupCreated = false;
        }

    } //GEN-LAST:event_createDefinedRelationshipGroup

    /**
     * Method to allow user to enter group of defined order calls
     * the <CODE>groupCreator</CODE> object functionality to create and
store the group.
     * @param evt Launched by the push of
<CODE>btnUserDefinedGroup</CODE>.
     */
    private void createUserEntryGroup(java.awt.event.ActionEvent evt)
    { //GEN-FIRST:event_createUserEntryGroup
        String orderString = JOptionPane.showInputDialog(null,
"Enter the size of the group
you would like to create.",
"User Group Dialog",
JOptionPane.QUESTION_MESSAGE);

        int order = 0;
        try
        {
            order = Integer.parseInt(orderString, 10);
        }
        catch (NumberFormatException e)
        {
            JOptionPane.showMessageDialog(null,
orderString + " is not a legal
number!" +
"\nError: " + e.getMessage(),
"Number Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }
        if (order > 1)
        {
            if (!myCreator.createEmptyGroup(order))
            {

```

```

        JOptionPane.showMessageDialog(null,
                                     "Error generating an empty group
of order " +
GENERATED",
                                     orderString + ".\nNO GROUP WAS
                                     "Group Generation Error",
                                     JOptionPane.ERROR_MESSAGE);
    }
    myGroup.updateTable(myCreator.getGroup(), true);
    lblResultsOfAnalysis.setText("");
}
else
{
    JOptionPane.showMessageDialog(null,
greater than 1.",
    "The group order must be an integer
    "Number Error",
JOptionPane.ERROR_MESSAGE);
}

    lblGroupName.setText("Group Table:  User Defined");
    lblResultsOfAnalysis.setEnabled(false);
    lblResultsOfAnalysis.setText("");
} //GEN-LAST:event_createUserEntryGroup

/**
 * Method to create the groups based upon a cross product of at
 * least two other groups calls the <CODE>groupCreator</CODE> object
 * functionality to create and store the group.
 *
 * The group is named by <CODE>groupNamer</CODE>
<CODE>groupIdentify</CODE> object after creation.
 * @param evt Launched by the push of <CODE>btnXProdGroup</CODE>.
 */
private void createXProdGroup(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_createXProdGroup
    cayleytable.groupMatrix myGroup1 = new cayleytable.groupMatrix(0);
    cayleytable.groupMatrix myGroup2 = new cayleytable.groupMatrix(0);

    String XProdName = "Group Table:  ";
    String XProdString = JOptionPane.showInputDialog(null,
the cross product. ",
    "Enter the number of groups in
    "Cross Product Dialog",
    JOptionPane.QUESTION_MESSAGE);

    int XProdNumOfGroups = -1;
    try
    {
        XProdNumOfGroups = Integer.parseInt(XProdString, 10);
    }
    catch (NumberFormatException e)
    {
        JOptionPane.showMessageDialog(null,
number!" +
    XProdString + " is not a legal
    "\nError: " + e.getMessage(),

```



```

JOptionPane.QUESTION_MESSAGE);
if (choice.equals("1"))
{
    if (createNextCyclicGroup())
        myGroup2.resetGroup(myCreator.getGroup());
    else
        return;
}
else if (choice.equals("2"))
{
    createDefinedRelationshipGroup(evt);
    if (blnGroupCreated)
        myGroup2.resetGroup(myCreator.getGroup());
    else
        return;
}
else
{
    JOptionPane.showMessageDialog(null,
group or defined relationship",
        "You must select either a Cyclic
        "Cross Product Error",
        JOptionPane.ERROR_MESSAGE);
    return;
}

if (!myCreator.createXProdGroup(myGroup1, myGroup2))
{
    JOptionPane.showMessageDialog(null,
Product" +
GENERATED",
        "Error generating a Cross
        " Group.\nNO GROUP WAS
        "Group Generation Error",
        JOptionPane.ERROR_MESSAGE);
    return;
}
XProdNumOfGroups = XProdNumOfGroups - 1;
}
myGroup.updateTable(myCreator.getGroup(), false);
findGroupName(evt);
blnGroupCreated = true;
lblResultsOfAnalysis.setEnabled(false);
lblResultsOfAnalysis.setText("");
} //GEN-LAST:event_createXProdGroup

/**
 * Method to create a cyclic group of a user defined size calls the
 * <CODE>groupCreator</CODE> object functionality to create and store
the group.
 *
 * The group is named by <CODE>groupNamer</CODE>
<CODE>groupIdentify</CODE> object after creation.
 * @param evt Launched by the push of <CODE>btnZnGroup</CODE>.
 */

```

```

private void createZnGroup(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_createZnGroup
    if (createNextCyclicGroup())
    {
        myGroup.updateTable(myCreator.getGroup(), false);
        findGroupName(evt);
        blnGroupCreated = true;
    }
    lblResultsOfAnalysis.setEnabled(false);
    lblResultsOfAnalysis.setText("");
} //GEN-LAST:event_createZnGroup

/**
 * Method to call functionality in <CODE>groupCreator</CODE> to create
the
 * inner automorphism of the current group currently stored in
<CODE>myCreator</CODE>
 * <CODE>groupMatrix</CODE> object.
 *
 * The group replaces what is stored in <CODE>myGroup</CODE>
<CODE>groupCreator</CODE> object
 * and named by <CODE>groupIdentify</CODE> after creation.
 * @param evt Launched by the push of <CODE>btnInnerAut</CODE>.
 */
private void createInnerAutGroup(java.awt.event.ActionEvent evt)
{ //GEN-FIRST:event_createInnerAutGroup
    if (!myCreator.resetGroup(myGroup.getGroup())
        || !myCreator.getGroup().checkForIdentity()
        || !myCreator.getGroup().checkForInverse()
        || !myCreator.getGroup().checkIfAssociative())
    {
        JOptionPane.showMessageDialog(null,
Automorphism" +
Group",
        "Error generating an Inner
        " Group.\nRequires Initial Legal
        "Illegal Group Error",
        JOptionPane.ERROR_MESSAGE);
        return;
    }

    if (myCreator.createInnerAutGroup())
    {
        myGroup.updateTable(myCreator.getGroup(), false);
        findGroupName(evt);
        lblResultsOfAnalysis.setBackground(Color.white);
        lblResultsOfAnalysis.setForeground(Color.BLUE);
        lblResultsOfAnalysis.setEnabled(true);
        lblResultsOfAnalysis.setText("Inner Automorphism Found: " +
groupNamer.getName());
    }
    else
    {
        lblResultsOfAnalysis.setEnabled(true);
        lblResultsOfAnalysis.setBackground(Color.white);
        lblResultsOfAnalysis.setForeground(Color.RED);
    }
}

```

```

        lblResultsOfAnalysis.setText("Error: No Inner Automorphism
Group Found");
        System.out.println("No Inner Automorphism Group Found");
    }

    }//GEN-LAST:event_createInnerAutGroup

    /**
     * Method to name the group stored in <CODE>myCreator</CODE>
<CODE>groupCreator</CODE> object with the
     * <CODE>groupNamer</CODE> <CODE>groupIdentify</CODE> object and
displays the results on
     * <CODE>lblGroupName</CODE> JLabel object.
     * @param evt Launched by the push of <CODE>btnCheckName</CODE>.
     */
    private void findGroupName(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_findGroupName
        lblResultsOfAnalysis.setEnabled(true);
        lblGroupName.setEnabled(false);
        lblResultsOfAnalysis.setBackground(Color.yellow);
        lblResultsOfAnalysis.setForeground(Color.red);
        if (!myCreator.resetGroup(myGroup.getGroup()))
            lblGroupName.setText("Group Table: This is not a Group");
        else if (!myCreator.getGroup().checkForIdentity())
            lblGroupName.setText("Group Table: This is not a Group");
        else if (!myCreator.getGroup().checkForInverse())
            lblGroupName.setText("Group Table: This is not a Group");
        else if (!myCreator.getGroup().checkIfAssociative())
            lblGroupName.setText("Group Table: This is not a Group");
        else
        {
            lblResultsOfAnalysis.setEnabled(false);
            groupNamer.resetGroupIdentify(myCreator.getGroup());
            lblGroupName.setEnabled(true);
            lblGroupName.setText("Group Table: " + groupNamer.getName());
        }
    } //GEN-LAST:event_findGroupName

    /**
     * Method to determine if the table stored in the
<CODE>myCreator</CODE> <CODE>groupCreator</CODE>
     * object is an Abelian group using the analysis functions that are a
part of <CODE>groupMatrix</CODE>
     * and displays the results on <CODE>lblResultsOfAnalysis</CODE>
JLabel object.
     * @param evt Launched by the push of <CODE>btnCheckAbel</CODE>.
     */
    private void checkIfAbelian(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_checkIfAbelian
        lblResultsOfAnalysis.setEnabled(true);
        lblResultsOfAnalysis.setBackground(Color.YELLOW);
        lblResultsOfAnalysis.setForeground(Color.RED);

        if (!myCreator.resetGroup(myGroup.getGroup()))
            lblResultsOfAnalysis.setText("The table is not a group because
there currently is no table to check!");
    }

```

```

        else if (!myCreator.getGroup().checkForIdentity())
            lblResultsOfAnalysis.setText("The table is not a group because
it does not have an identity element");
        else if (!myCreator.getGroup().checkForInverse())
            lblResultsOfAnalysis.setText("The table is not a group because
it does not have an inverse for each element");
        else if (!myCreator.getGroup().checkIfAssociative())
            lblResultsOfAnalysis.setText("The table is not a group because
it is not associative");
        else if (!myCreator.getGroup().checkIfCommutative())
            lblResultsOfAnalysis.setText("The table is not an Abelian
Group because it is not commutative");
        else
        {
            lblResultsOfAnalysis.setForeground(Color.BLUE);
            lblResultsOfAnalysis.setText("The table is an Abelian
Group.");
        }

    } //GEN-LAST:event_checkIfAbelian

/**
 * Method to determine if the table stored in the
<CODE>myCreator</CODE> <CODE>groupCreator</CODE>
 * object is a group using the analysis functions that are a part of
<CODE>groupMatrix</CODE>
 * and displays the results on <CODE>lblResultsOfAnalysis</CODE>
JLabel object.
 * @param evt Launched by the push of <CODE>btnCheckGroup</CODE>.
 */
private void checkIfGroup(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_checkIfGroup
    lblResultsOfAnalysis.setEnabled(true);
    lblResultsOfAnalysis.setBackground(Color.yellow);
    lblResultsOfAnalysis.setForeground(Color.red);

    if (!myCreator.resetGroup(myGroup.getGroup()))
        lblResultsOfAnalysis.setText("The table is not a group because
there currently is no table to check!");
    else if (!myCreator.getGroup().checkForIdentity())
        lblResultsOfAnalysis.setText("The table is not a group because
it does not have an identity element");
    else if (!myCreator.getGroup().checkForInverse())
        lblResultsOfAnalysis.setText("The table is not a group because
it does not have an inverse for each element");
    else if (!myCreator.getGroup().checkIfAssociative())
        lblResultsOfAnalysis.setText("The table is not a group because
it is not associative");
    else
    {
        lblResultsOfAnalysis.setForeground(Color.blue);
        lblResultsOfAnalysis.setText("The table is a group.");
    }
} //GEN-LAST:event_checkIfGroup

```

```

// Variables declaration - do not modify//GEN-BEGIN:variables
private javax.swing.JButton btnCheckAabel;
private javax.swing.JButton btnCheckGroup;
private javax.swing.JButton btnCheckName;
private javax.swing.JButton btnDefRelnGroup;
private javax.swing.JButton btnInnerAut;
private javax.swing.JButton btnUserDefinedGroup;
private javax.swing.JButton btnXProdGroup;
private javax.swing.JButton btnZnGroup;
private cayleytable.groupIdentify groupNamer;
private javax.swing.JLabel lblGeneratorButtons;
private javax.swing.JLabel lblGroupName;
private javax.swing.JLabel lblPropertyButtons;
private javax.swing.JLabel lblResultsOfAnalysis;
private cayleytable.groupCreator myCreator;
private cayleytable.groupPanel myGroup;
// End of variables declaration//GEN-END:variables
/**
 * Boolean variable used by
<CODE>createDefinedRelationshipGroup</CODE>,
 * <CODE>createXProdGroup</CODE>, and <CODE>createZnGroup</CODE>
methods to identify if
 * a new group was successfully created.
 */
private boolean blnGroupCreated;

/**
 * Method used to request the "pseudo" commutative defined
relationships
 * from the user for use in the
<CODE>createDefinedRelationshipGroup</CODE> method.
 * @param a Value representing one subgroup in relationship ba=?
 * where ?? is the new relationship.
 * @param b Value representing second subgroup in relationship ba=?
where ?? is the new relationship.
 * @return String object containing the right hand side of the "pseudo"
commutative relationship
 */
public String getNextRelationship(int a, int b)
{
    String relationship = JOptionPane.showInputDialog(null,
        "Enter the relationship for " +
(char)((char)b+97) + (char)((char)a+97) +
        "\na represents group 1, b represents
group 2, etc." +
        "\nA is the inverse of a, B is the
inverse of b, etc.",
        "Relationship Dialog",
        JOptionPane.QUESTION_MESSAGE);

    int length = relationship.length();

    if (length < 2) return relationship;

    for (int i=0; i<length-1; i++)
    {

```

```

        for (int j=i+1; j<length; j++)
            {
                if ((int) relationship.toLowerCase().charAt(i) > (int)
relationship.toLowerCase().charAt(j))
                    {
                        JOptionPane.showMessageDialog(null,
"Error in relationship
order. Must order elements " +
alphabetical order.",
"so that letters are in
"Relationship Error",
JOptionPane.ERROR_MESSAGE);
                    }
                return null;
            }
        }
    }

    return relationship;
}

/**
 * Method used to create the actual cyclic group that is used by
 * the <CODE>createZnGroup</CODE> and <CODE>createXProdGroup</CODE>
methods.
 * @return Result identifying if a new Cyclic group was successfully
created.
 */
private boolean createNextCyclicGroup()
{
    String ZnOrderString = JOptionPane.showInputDialog(null,
"Enter the order for the Cyclic
Group. ",
"Cyclic Group Dialog",
JOptionPane.QUESTION_MESSAGE);

    int ZnOrder = -1;

    try
    {
        ZnOrder = Integer.parseInt(ZnOrderString, 10);
    }
    catch (NumberFormatException e)
    {
        JOptionPane.showMessageDialog(null,
ZnOrderString + " is not a legal
number!" +
"\nError: " + e.getMessage(),
"Number Error",
JOptionPane.ERROR_MESSAGE);
        return false;
    }
    if (ZnOrder > 0)
    {
        if (!myCreator.createCyclicGroup(ZnOrder))

```

```

        {
            JOptionPane.showMessageDialog(null,
                "Error generating a Z" +
ZnOrderString +
                " Group.\nNO GROUP WAS
GENERATED",
                "Group Generation Error",
                JOptionPane.ERROR_MESSAGE);
            return false;
        }
        lblResultsOfAnalysis.setText("");
        return true;
    }
    else return false;
}

/**
 * Method to calculate the factorial of an integer
 * @param n Integer for which the factorial is calculated
 * @return Factorial of n
 */
private int factorial(int n)
{
    if (n <= 0) return 1;
    else return n * factorial(n-1);
}
}
}

```

GROUPMAIN.JAVA

```

/*
 * groupMain.java
 *
 * Created on January 16, 2005, 1:19 PM
 */

package cayleytable;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

/**
 * Main Frame of Cayley table viewer for use when tool is run as a
 * standalone program. See <CODE>groupMain</CODE> for identical version
 * in applet format. User interface that contains <CODE>groupPanel</CODE>
 * that displays Cayley Table and buttons to generate and analyze a
 * Cayley Table.
 * @author Jeffrey Barr
 */
public class groupMainFrame extends javax.swing.JFrame {

```

```

/**
 * Initializes the Frame <CODE>groupMainFrame</CODE> through call to
 initialize all
 * of the components in the Frame.
 */
public groupMainFrame() {
    initComponents();
    setVisible(true);
}

/**
 * This method is called from within the
<CODE>groupMainFrame()</CODE>method to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 * Code generated via Netbeans.
 */
private void initComponents() { //GEN-BEGIN:initComponents
    java.awt.GridBagConstraints gridBagConstraints;

    groupNamer = new cayleytable.groupIdentify();
    myCreator = new cayleytable.groupCreator();
    lblGroupName = new javax.swing.JLabel();
    myGroup = new cayleytable.groupPanel();
    lblResultsOfAnalysis = new javax.swing.JLabel();
    lblPropertyButtons = new javax.swing.JLabel();
    btnCheckGroup = new javax.swing.JButton();
    btnCheckAbel = new javax.swing.JButton();
    btnCheckName = new javax.swing.JButton();
    btnInnerAut = new javax.swing.JButton();
    lblGeneratorButtons = new javax.swing.JLabel();
    btnZnGroup = new javax.swing.JButton();
    btnXProdGroup = new javax.swing.JButton();
    btnUserDefinedGroup = new javax.swing.JButton();
    btnDefRelnGroup = new javax.swing.JButton();

    getContentPane().setLayout(new java.awt.GridBagLayout());

    lblGroupName.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
    lblGroupName.setText(groupNamer.getName());
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 0;
    gridBagConstraints.gridwidth =
java.awt.GridBagConstraints.REMAINDER;
    gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
    gridBagConstraints.weightx = 1.0;
    getContentPane().add(lblGroupName, gridBagConstraints);

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 1;
    gridBagConstraints.gridwidth =
java.awt.GridBagConstraints.REMAINDER;

```

```

gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
gridBagConstraints.weighty = 1.0;
getContentPane().add(myGroup, gridBagConstraints);

lblResultsOfAnalysis.setHorizontalAlignment(javax.swing.SwingConstants.CEN
TER);
    lblResultsOfAnalysis.setEnabled(false);
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 11;
    gridBagConstraints.gridwidth =
java.awt.GridBagConstraints.REMAINDER;
    gridBagConstraints.fill = java.awt.GridBagConstraints.BOTH;
    gridBagConstraints.weightx = 1.0;
    getContentPane().add(lblResultsOfAnalysis, gridBagConstraints);

lblPropertyButtons.setHorizontalAlignment(javax.swing.SwingConstants.LEFT)
;
    lblPropertyButtons.setText("Check Group Properties:");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 12;
    gridBagConstraints.gridwidth =
java.awt.GridBagConstraints.RELATIVE;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(lblPropertyButtons, gridBagConstraints);

    btnCheckGroup.setText("Check if Group");
    btnCheckGroup.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            checkIfGroup(evt);
        }
    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 13;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(btnCheckGroup, gridBagConstraints);

    btnCheckAbel.setText("Check if Abelian");
    btnCheckAbel.addActionListener(new java.awt.event.ActionListener()
{
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            checkIfAbelian(evt);
        }
    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 13;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(btnCheckAbel, gridBagConstraints);

    btnCheckName.setText("Find Group Name");

```

```

        btnCheckName.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            findGroupName(evt);
        }
    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 13;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(btnCheckName, gridBagConstraints);

    btnInnerAut.setText("Find Inner Automorphism");
    btnInnerAut.addActionListener(new java.awt.event.ActionListener()
    {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            createInnerAutGroup(evt);
        }
    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 13;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(btnInnerAut, gridBagConstraints);

    lblGeneratorButtons.setHorizontalAlignment(javax.swing.SwingConstants.LEFT
);
    lblGeneratorButtons.setText("Choose Type of Group to Enter:");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 14;
    gridBagConstraints.gridwidth =
java.awt.GridBagConstraints.RELATIVE;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(lblGeneratorButtons, gridBagConstraints);

    btnZnGroup.setText("Cyclic Group");
    btnZnGroup.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            createZnGroup(evt);
        }
    });

    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridy = 15;
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    getContentPane().add(btnZnGroup, gridBagConstraints);

    btnXProdGroup.setText("Cross Product Group");
    btnXProdGroup.addActionListener(new
java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            createXProdGroup(evt);
        }
    });

```

```

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridy = 15;
        gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
        getContentPane().add(btnXProdGroup, gridBagConstraints);

        btnUserDefinedGroup.setText("User Defined Group");
        btnUserDefinedGroup.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                createUserEntryGroup(evt);
            }
        });

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridy = 15;
        gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
        getContentPane().add(btnUserDefinedGroup, gridBagConstraints);

        btnDefRelnGroup.setText("Defined Relationship Group");
        btnDefRelnGroup.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                createDefinedRelationshipGroup(evt);
            }
        });

        gridBagConstraints = new java.awt.GridBagConstraints();
        gridBagConstraints.gridy = 15;
        gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
        getContentPane().add(btnDefRelnGroup, gridBagConstraints);

        java.awt.Dimension screenSize =
java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-650)/2, (screenSize.height-700)/2,
650, 700);
    } //GEN-END: initComponents

    /**
     * Method to create the groups based upon a defined relationship calls
     * the <CODE>groupCreator</CODE> object functionality to create and
     * store the group.
     *
     * The group is named by <CODE>groupNamer</CODE>
     * <CODE>groupIdentify</CODE> object after creation.
     * @param evt Launched by the push of <CODE>btnDefRelnGroup</CODE>.
     */
    private void createDefinedRelationshipGroup(java.awt.event.ActionEvent
evt) { //GEN-FIRST:event_createDefinedRelationshipGroup
        ArrayList<String> generatorList = new ArrayList<String>();
        ArrayList<groupRelation> definedRelationships = new
ArrayList<groupRelation>();

        String defRelName = "Group Table:  Defined Relationship";

```

```

        // Determine number of generators to be used and error check that it
is a legal value
        String defRelString = JOptionPane.showInputDialog(null,
                "Enter the number of generators
in the defined relationship. ",
                "Defined Relationship Dialog",
                JOptionPane.QUESTION_MESSAGE);

        int defRelNumOfGenerators = -1;
        int numRelationships;
        try
        {
            defRelNumOfGenerators = Integer.parseInt(defRelString, 10);
        }
        catch (NumberFormatException e)
        {
            JOptionPane.showMessageDialog(null,
                defRelString + " is not a legal
integer!" +
                "\nError: " + e.getMessage(),
                "Number Error",
                JOptionPane.ERROR_MESSAGE);
            blnGroupCreated = false;
            return;
        }

        numRelationships = factorial(defRelNumOfGenerators);
        System.out.println("Defining Relationships");
        if (defRelNumOfGenerators < 2)
        {
            JOptionPane.showMessageDialog(null,
                "You cannot create a defined
relationship with less" +
                " than two groups.", "Defined
Relationship Error",
                JOptionPane.ERROR_MESSAGE);
            blnGroupCreated = false;
            return;
        }

        // Determine the number of elements in each generator
        String elementString;
        for (int n=0; n<defRelNumOfGenerators; n++)
        {
            elementString = JOptionPane.showInputDialog(null,
                "Enter the number of
elements in generator " + n,
                "Element Query Dialog",
                JOptionPane.QUESTION_MESSAGE);

            // Ensure element string is an integer
            try
            {
                int temp = Integer.parseInt(elementString, 10);
                if (temp < 1)
                {

```

```

        JOptionPane.showMessageDialog(null,
                                     "The order of a generator must
be greater than 0!",
                                     "Number Error",
JOptionPane.ERROR_MESSAGE);
        n = n-1;
        continue;
    }
}
catch (NumberFormatException e)
{
    JOptionPane.showMessageDialog(null,
integer!" +
                                     elementString + " is not a legal
                                     "\nError: " + e.getMessage(),
                                     "Number Error",
JOptionPane.ERROR_MESSAGE);
        n = n-1;
        continue;
}

    generatorList.add(elementString);
    definedRelationships.add(new groupRelation(n,
Integer.parseInt(elementString,10)));
}

    boolean extraRelation = true;
    while (extraRelation)
    {
        String question = JOptionPane.showInputDialog(null,
relationship between generators?\n"
                                     "Do you have an extra
                                     + " Enter yes/no. ",
                                     "Relationship Dialog",

JOptionPane.QUESTION_MESSAGE);
        question = question.toLowerCase();
        if (question.compareTo("yes") == 0 || question.compareTo("y")
== 0)
        {
            String relationship = JOptionPane.showInputDialog(null,
relationship.",
                                     "Enter the extra
Dialog",
                                     "Relationship

JOptionPane.QUESTION_MESSAGE);
            groupRelation extra = new groupRelation(relationship,
generatorList);
            boolean foundMatch = false;
            for (int i=0; i<definedRelationships.size(); i++)
            {
                groupRelation tempRel = (groupRelation)
definedRelationships.get(i);

```

```

        if (extra.getLeft().compareTo(tempRel.getLeft())
== 0)
            {
                definedRelationships.set(i, extra);
                foundMatch = true;
            }
        }
        if (!foundMatch) definedRelationships.add(extra);
    }
    else
        extraRelation = false;
}

for (int i=0; i<defRelNumOfGenerators-1; i++)
{
    for (int j=i+1; j<defRelNumOfGenerators; j++)
    {
        definedRelationships.add(new groupRelation(j, i,
getNextRelationship(i,j), generatorList));
    }
}

System.out.println("Relationships Defined");
for (int j=0; j<definedRelationships.size(); j++)
{
    groupRelation temp = (groupRelation)
definedRelationships.get(j);
    System.out.println(j + ": " + temp.getLeft() + " = " +
temp.getRight());
}

System.out.println("Relationships Defined twice");
if (myCreator.createDefineRelationGroup(generatorList,
definedRelationships))
{
    myGroup.updateTable(myCreator.getGroup(), false);
    System.out.println("Attempting to identify group");
    findGroupName(evt);
    lblResultsOfAnalysis.setEnabled(false);
    lblResultsOfAnalysis.setText("");
    blnGroupCreated = true;
}
else
{
    lblResultsOfAnalysis.setBackground(Color.white);
    lblResultsOfAnalysis.setForeground(Color.RED);
    lblResultsOfAnalysis.setEnabled(true);
    lblResultsOfAnalysis.setText("Error: No Defined Relationship
Group Found");
    System.out.println("No Defined Relationship Group Found");
    blnGroupCreated = false;
}

} //GEN-LAST:event_createDefinedRelationshipGroup

```

```

/**
 * Method to allow user to enter group of defined order calls
 * the <CODE>groupCreator</CODE> object functionality to create and
store the group.
 * @param evt Launched by the push of
<CODE>btnUserDefinedGroup</CODE>.
 */
private void createUserEntryGroup(java.awt.event.ActionEvent evt)
{ //GEN-FIRST:event_createUserEntryGroup
    String orderString = JOptionPane.showInputDialog(null,
you would like to create.",
                                                "Enter the size of the group
                                                "User Group Dialog",
                                                JOptionPane.QUESTION_MESSAGE);

    int order = 0;
    try
    {
        order = Integer.parseInt(orderString, 10);
    }
    catch (NumberFormatException e)
    {
        JOptionPane.showMessageDialog(null,
integer!" +
                                                "\nError: " + e.getMessage(),
                                                "Number Error",
JOptionPane.ERROR_MESSAGE);
        return;
    }
    if (order > 1)
    {
        if (!myCreator.createEmptyGroup(order))
        {
            JOptionPane.showMessageDialog(null,
of order " +
                                                "Error generating an empty group
                                                orderString + ".\nNO GROUP WAS
GENERATED",
                                                "Group Generation Error",
                                                JOptionPane.ERROR_MESSAGE);
        }
        myGroup.updateTable(myCreator.getGroup(), true);
        lblResultsOfAnalysis.setText("");
    }
    else
    {
        JOptionPane.showMessageDialog(null,
greater than 1.",
                                                "The group order must be an integer
                                                "Number Error",
JOptionPane.ERROR_MESSAGE);
        return;
    }

    lblGroupName.setText("Group Table: User Defined");
}

```

```

        lblResultsOfAnalysis.setEnabled(false);
        lblResultsOfAnalysis.setText("");
    } //GEN-LAST:event_createUserEntryGroup

/**
 * Method to create the groups based upon a cross product of at
 * least two other groups calls the <CODE>groupCreator</CODE> object
 * functionality to create and store the group.
 *
 * The group is named by <CODE>groupNamer</CODE>
<CODE>groupIdentify</CODE> object after creation.
 * @param evt Launched by the push of <CODE>btnXProdGroup</CODE>.
 */
private void createXProdGroup(java.awt.event.ActionEvent evt) { //GEN-FIRST:event_createXProdGroup
    cayleytable.groupMatrix myGroup1 = new cayleytable.groupMatrix(0);
    cayleytable.groupMatrix myGroup2 = new cayleytable.groupMatrix(0);

    String XProdName = "Group Table: ";
    String XProdString = JOptionPane.showInputDialog(null,
        "Enter the number of groups in
the cross product. ",
        "Cross Product Dialog",
        JOptionPane.QUESTION_MESSAGE);

    int XProdNumOfGroups = -1;
    try
    {
        XProdNumOfGroups = Integer.parseInt(XProdString, 10);
    }
    catch (NumberFormatException e)
    {
        JOptionPane.showMessageDialog(null,
            XProdString + " is not a legal
integer!" +
            "\nError: " + e.getMessage(),
            "Number Error",
            JOptionPane.ERROR_MESSAGE);
        return;
    }

    if (XProdNumOfGroups <= 1)
    {
        JOptionPane.showMessageDialog(null,
            "You cannot create a cross product
with less" +
            " than two groups.", "Cross Product
Error",
            JOptionPane.ERROR_MESSAGE);
        return;
    }

    String choice = JOptionPane.showInputDialog(null,
        "Enter 1 if first group is a
Cyclic group, else enter 2 for a Defined Relationship. ",
        "Cross Product Choice Dialog",

```

```

JOptionPane.QUESTION_MESSAGE);
if (choice.equals("1"))
{
    if (createNextCyclicGroup())
        XProdNumOfGroups = XProdNumOfGroups - 1;
    else
        return;
}
else if (choice.equals("2"))
{
    createDefinedRelationshipGroup(evt);
    if (blnGroupCreated)
        XProdNumOfGroups = XProdNumOfGroups - 1;
    else
        return;
}
else
{
    JOptionPane.showMessageDialog(null,
or defined relationship",
        "You must select either a Cyclic group
        "Cross Product Error",
        JOptionPane.ERROR_MESSAGE);
    return;
}

while (XProdNumOfGroups > 0)
{
    myGroup1.resetGroup(myCreator.getGroup());

    choice = JOptionPane.showInputDialog(null,
Cyclic group, else enter 2 for a Defined Relationship. ",
        "Enter 1 if next group is a
        "Cross Product Choice Dialog",
        JOptionPane.QUESTION_MESSAGE);

    if (choice.equals("1"))
    {
        if (createNextCyclicGroup())
            myGroup2.resetGroup(myCreator.getGroup());
        else
            return;
    }
    else if (choice.equals("2"))
    {
        createDefinedRelationshipGroup(evt);
        if (blnGroupCreated)
            myGroup2.resetGroup(myCreator.getGroup());
        else
            return;
    }
    else
    {
        JOptionPane.showMessageDialog(null,
group or defined relationship",
        "You must select either a Cyclic

```

```

        "Cross Product Error",
        JOptionPane.ERROR_MESSAGE);
    }
    return;
}

if (!myCreator.createXProdGroup(myGroup1, myGroup2))
{
    JOptionPane.showMessageDialog(null,
        "Error generating a Cross
Product" +
GENERATED",
        " Group.\nNO GROUP WAS
        "Group Generation Error",
        JOptionPane.ERROR_MESSAGE);
    return;
}
XProdNumOfGroups = XProdNumOfGroups - 1;
}
myGroup.updateTable(myCreator.getGroup(), false);
findGroupName(evt);
    blnGroupCreated = true;
    lblResultsOfAnalysis.setEnabled(false);
    lblResultsOfAnalysis.setText("");
} //GEN-LAST:event_createXProdGroup

/**
 * Method to create a cyclic group of a user defined size calls the
 * <CODE>groupCreator</CODE> object functionality to create and store
the group.
 *
 * The group is named by <CODE>groupNamer</CODE>
<CODE>groupIdentify</CODE> object after creation.
 * @param evt Launched by the push of <CODE>btnZnGroup</CODE>.
 */
private void createZnGroup(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_createZnGroup
    if (createNextCyclicGroup())
    {
        myGroup.updateTable(myCreator.getGroup(), false);
        findGroupName(evt);
        blnGroupCreated = true;
    }
    lblResultsOfAnalysis.setEnabled(false);
    lblResultsOfAnalysis.setText("");
} //GEN-LAST:event_createZnGroup

/**
 * Method to call functionality in <CODE>groupCreator</CODE> to create
the
 * inner automorphism of the current group currently stored in
<CODE>myCreator</CODE>
 * <CODE>groupMatrix</CODE> object.
 *
 * The group replaces what is stored in <CODE>myGroup</CODE>
<CODE>groupCreator</CODE> object
 * and named by <CODE>groupIdentify</CODE> after creation.

```

```

    * @param evt Launched by the push of <CODE>btnInnerAut</CODE>.
    */
    private void createInnerAutGroup(java.awt.event.ActionEvent evt)
    { //GEN-FIRST:event_createInnerAutGroup
        if (!myCreator.resetGroup(myGroup.getGroup()))
            | | !myCreator.getGroup().checkForIdentity()
            | | !myCreator.getGroup().checkForInverse()
            | | !myCreator.getGroup().checkIfAssociative()
            {
                JOptionPane.showMessageDialog(null,
                    "Error generating an Inner
Automorphism" +
                    " Group.\nRequires Initial Legal
Group",
                    "Illegal Group Error",
                    JOptionPane.ERROR_MESSAGE);

                return;
            }

        if (myCreator.createInnerAutGroup())
            {
                myGroup.updateTable(myCreator.getGroup(), false);
                findGroupName(evt);
                lblResultsOfAnalysis.setBackground(Color.white);
                lblResultsOfAnalysis.setForeground(Color.BLUE);
                lblResultsOfAnalysis.setEnabled(true);
                if (groupNamer.isIdentified())
                    lblResultsOfAnalysis.setText("Inner Automorphism
Found: " + groupNamer.getName());
                else
                    lblResultsOfAnalysis.setText("Inner Automorphism
Found: Unknown Group Name");
                System.out.println("Inner Automorphism Group Found");
            }
        else
            {
                lblResultsOfAnalysis.setBackground(Color.white);
                lblResultsOfAnalysis.setForeground(Color.RED);
                lblResultsOfAnalysis.setEnabled(true);
                lblResultsOfAnalysis.setText("Error: No Inner
Automorphism Group Found");
                System.out.println("No Inner Automorphism Group Found");
            }
    } //GEN-LAST:event_createInnerAutGroup

    /**
     * Method to name the group stored in <CODE>myCreator</CODE>
     <CODE>groupCreator</CODE> object with the
     * <CODE>groupNamer</CODE> <CODE>groupIdentify</CODE> object and
     displays the results on
     * <CODE>lblGroupName</CODE> JLabel object.
     * @param evt Launched by the push of <CODE>btnCheckName</CODE>.
     */
    private void findGroupName(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_findGroupName

```

```

        lblResultsOfAnalysis.setEnabled(true);
        lblGroupName.setEnabled(false);
        lblResultsOfAnalysis.setBackground(Color.yellow);
        lblResultsOfAnalysis.setForeground(Color.red);
        if (!myCreator.resetGroup(myGroup.getGroup()))
            lblGroupName.setText("Group Table: This is not a Group");
        else if (!myCreator.getGroup().checkForIdentity())
            lblGroupName.setText("Group Table: This is not a Group");
        else if (!myCreator.getGroup().checkForInverse())
            lblGroupName.setText("Group Table: This is not a Group");
        else if (!myCreator.getGroup().checkIfAssociative())
            lblGroupName.setText("Group Table: This is not a Group");
        else
        {
            lblResultsOfAnalysis.setEnabled(false);
            groupNamer.resetGroupIdentify(myCreator.getGroup());
            lblGroupName.setEnabled(true);
            lblGroupName.setText("Group Table: " + groupNamer.getName());
        }
    } //GEN-LAST:event_findGroupName

/**
 * Method to determine if the table stored in the
<CODE>myCreator</CODE> <CODE>groupCreator</CODE>
 * object is an Abelian group using the analysis functions that are a
part of <CODE>groupMatrix</CODE>
 * and displays the results on <CODE>lblResultsOfAnalysis</CODE>
JLabel object.
 * @param evt Launched by the push of <CODE>btnCheckAbel</CODE>.
 */
private void checkIfAbelian(java.awt.event.ActionEvent evt) { //GEN-FIRST:event_checkIfAbelian
    lblResultsOfAnalysis.setEnabled(true);
    lblResultsOfAnalysis.setBackground(Color.YELLOW);
    lblResultsOfAnalysis.setForeground(Color.RED);

    if (!myCreator.resetGroup(myGroup.getGroup()))
        lblResultsOfAnalysis.setText("The table is not a group because
there currently is no table to check!");
    else if (!myCreator.getGroup().checkForIdentity())
        lblResultsOfAnalysis.setText("The table is not a group because
it does not have an identity element");
    else if (!myCreator.getGroup().checkForInverse())
        lblResultsOfAnalysis.setText("The table is not a group because
it does not have an inverse for each element");
    else if (!myCreator.getGroup().checkIfAssociative())
        lblResultsOfAnalysis.setText("The table is not a group because
it is not associative");
    else if (!myCreator.getGroup().checkIfCommutative())
        lblResultsOfAnalysis.setText("The table is not an Abeliain
Group because it is not communative");
    else
    {
        lblResultsOfAnalysis.setForeground(Color.BLUE);
        lblResultsOfAnalysis.setText("The table is an Abelian
Group.");
    }
}

```

```

    }

} //GEN-LAST:event_checkIfAbelian

/**
 * Method to determine if the table stored in the
<CODE>myCreator</CODE> <CODE>groupCreator</CODE>
 * object is a group using the analysis functions that are a part of
<CODE>groupMatrix</CODE>
 * and displays the results on <CODE>lblResultsOfAnalysis</CODE>
JLabel object.
 * @param evt Launched by the push of <CODE>btnCheckGroup</CODE>.
 */
private void checkIfGroup(java.awt.event.ActionEvent evt) { //GEN-
FIRST:event_checkIfGroup
    lblResultsOfAnalysis.setEnabled(true);
    lblResultsOfAnalysis.setBackground(Color.yellow);
    lblResultsOfAnalysis.setForeground(Color.red);

    if (!myCreator.resetGroup(myGroup.getGroup()))
        lblResultsOfAnalysis.setText("The table is not a group because
there currently is no table to check!");
    else if (!myCreator.getGroup().checkForIdentity())
        lblResultsOfAnalysis.setText("The table is not a group because
it does not have an identity element");
    else if (!myCreator.getGroup().checkForInverse())
        lblResultsOfAnalysis.setText("The table is not a group because
it does not have an inverse for each element");
    else if (!myCreator.getGroup().checkIfAssociative())
        lblResultsOfAnalysis.setText("The table is not a group because
it is not associative");
    else
    {
        lblResultsOfAnalysis.setForeground(Color.blue);
        lblResultsOfAnalysis.setText("The table is a group.");
    }
} //GEN-LAST:event_checkIfGroup

// Variables declaration - do not modify //GEN-BEGIN:variables
/**
 * JButton object to launch <CODE>checkIfAbel</CODE> method.
 */
private javax.swing.JButton btnCheckAbel;
/**
 * JButton object to launch <CODE>checkIfGroup</CODE> method.
 */
private javax.swing.JButton btnCheckGroup;
/**
 * JButton object to launch <CODE>findGroupName</CODE> method.
 */
private javax.swing.JButton btnCheckName;
/**
 * JButton object to launch
<CODE>createDefinedRelationshipGroup</CODE> method.
 */

```

```

private javax.swing.JButton btnDefRelnGroup;
/**
 * JButton object to launch <CODE>createInnerAutGroup</CODE> method.
 */
private javax.swing.JButton btnInnerAut;
/**
 * JButton object to launch <CODE>createUserEntryGroup</CODE> method.
 */
private javax.swing.JButton btnUserDefinedGroup;
/**
 * JButton object to launch <CODE>createXProdGroup</CODE> method.
 */
private javax.swing.JButton btnXProdGroup;
/**
 * JButton object to launch <CODE>createZnGroup</CODE> method.
 */
private javax.swing.JButton btnZnGroup;
/**
 * <CODE>groupIdentify</CODE> object called to determine the name of
the current
 * group that is stored in <CODE>myCreator</CODE>
<CODE>groupCreator</CODE> object and displayed
 * in <CODE>myGroup</CODE> <CODE>groupPanel</CODE> object.
 */
private cayleytable.groupIdentify groupNamer;
/**
 * JLabel that describes functionality of
<CODE>btnDefRelnGroup</CODE>,
 * <CODE>btnUserDefinedGroup</CODE>, <CODE>btnXProdGroup</CODE>, and
<CODE>btnZnGroup</CODE>.
 */
private javax.swing.JLabel lblGeneratorButtons;
/**
 * JLabel used to display results of <CODE>findGroupName</CODE> method
with the current
 * name of the group displayed in <CODE>myGroup</CODE>
<CODE>groupPanel</CODE> object.
 */
private javax.swing.JLabel lblGroupName;
/**
 * JLabel that describes functionality of <CODE>btnCheckGroup</CODE>,
 * <CODE>btnCheckAabel</CODE>, <CODE>btnCheckName</CODE>, and
<CODE>btnInnerAut</CODE>.
 */
private javax.swing.JLabel lblPropertyButtons;
/**
 * JLabel used to display results of <CODE>checkGroup</CODE>,
<CODE>checkAabel</CODE>
 * and <CODE>createInnerAutGroup</CODE> methods with respect to the
current
 * group displayed in <CODE>myGroup</CODE> <CODE>groupPanel</CODE>
object.
 */
private javax.swing.JLabel lblResultsOfAnalysis;
/**

```

```

    * groupCreator object used to create and store group via the
<CODE>createDefinedRelationshipGroup</CODE>,
    * <CODE>createUserEntryGroup</CODE>, <CODE>createXProdGroup</CODE>,
<CODE>createZnGroup</CODE>, and <CODE>createInnerAutGroup</CODE> methods.
    *
    * Also, stores group that is analyzed via the
<CODE>checkIfGroup</CODE> and <CODE>checkIfAbelian</CODE> methods.
    *
    * This is the group that is also sent to the <CODE>myGroup</CODE>
<CODE>groupPanel</CODE> object and
    * the <CODE>groupNamer</CODE> <CODE>groupIdentify</CODE> object for
use in displaying the Cayley Table
    * and finding the name of the group via the
<CODE>findGroupName</CODE> method.
    */
    private cayleytable.groupCreator myCreator;
    /**
    * <CODE>groupPanel</CODE> object used for displaying the Cayley Table
that it
    * is passed via a <CODE>groupMatrix</CODE> object.
    */
    private cayleytable.groupPanel myGroup;
    // End of variables declaration//GEN-END:variables
    /**
    * Boolean variable used by
<CODE>createDefinedRelationshipGroup</CODE>,
    * <CODE>createXProdGroup</CODE>, and <CODE>createZnGroup</CODE>
methods to identify if
    * a new group was successfully created.
    */
    private boolean blnGroupCreated;

    /**
    * Method used to request the "pseudo" commutative defined
relationships
    * from the user for use in the
<CODE>createDefinedRelationshipGroup</CODE> method.
    * @param a Value representing one subgroup in relationship ba=??
where ?? is the new relationship.
    * @param b Value representing second subgroup in relationship
ba=?? where ?? is the new relationship.
    * @return String object containing the right hand side of the
"pseudo" commutative relationship
    */
    private String getNextRelationship(int a, int b)
    {
        String relationship = JOptionPane.showInputDialog(null,
            "Enter the relationship for " +
(char)((char)b+97) + (char)((char)a+97) +
            "\na represents group 1, b represents
group 2, etc." +
            "\nA is the inverse of a, B is the
inverse of b, etc.",
            "Relationship Dialog",
            JOptionPane.QUESTION_MESSAGE);

```

```

int length = relationship.length();

if (length < 2) return relationship;

for (int i=0; i<length-1; i++)
    {
        for (int j=i+1; j<length; j++)
            {
                if ((int) relationship.toLowerCase().charAt(i) > (int)
relationship.toLowerCase().charAt(j))
                    {
                        JOptionPane.showMessageDialog(null,
order. Must order elements " +
alphabetical order.",
                        "Error in relationship
                        "so that letters are in
                        "Relationship Error",
JOptionPane.ERROR_MESSAGE);
                        return null;
                    }
            }
    }

return relationship;
}

/**
 * Method used to create the actual cyclic group that is used by
 * the <CODE>createZnGroup</CODE> and <CODE>createXProdGroup</CODE>
methods.
 * @return Result identifying if a new Cyclic group was successfully
created.
 */
private boolean createNextCyclicGroup()
{
    String ZnOrderString = JOptionPane.showInputDialog(null,
Group. ",
        "Enter the order for the Cyclic
        "Cyclic Group Dialog",
        JOptionPane.QUESTION_MESSAGE);

    int ZnOrder = -1;

    try
    {
        ZnOrder = Integer.parseInt(ZnOrderString, 10);
    }
    catch (NumberFormatException e)
    {
        JOptionPane.showMessageDialog(null,
integer!" +
            ZnOrderString + " is not a legal
            "\nError: " + e.getMessage(),

```

```

        "Number Error",
JOptionPane.ERROR_MESSAGE);
        return false;
    }
    if (ZnOrder > 0)
    {
        if (!myCreator.createCyclicGroup(ZnOrder))
        {
            JOptionPane.showMessageDialog(null,
                "Error generating a Z" +
ZnOrderString +
                " Group.\nNO GROUP WAS
GENERATED",
                "Group Generation Error",
                JOptionPane.ERROR_MESSAGE);
            return false;
        }
        lblResultsOfAnalysis.setText("");
        return true;
    }
    JOptionPane.showMessageDialog(null,
        ZnOrderString + " is not a legal integer!\n"
+
        " Order of a group must be greater than 0",
        "Group Generation Error",
        JOptionPane.ERROR_MESSAGE);
    return false;
}

/**
 * Method to calculate the factorial of an integer
 * @param n Integer for which the factorial is calculated
 * @return Factorial of n
 */
private int factorial(int n)
{
    if (n <= 0) return 1;
    else return n * factorial(n-1);
}
}

```

GROUPPANEL.JAVA

```

/*
 * groupPanel.java
 *
 * Created on January 16, 2005, 1:20 PM
 */

package cayleytable;

import javax.swing.table.*;
import java.awt.*;

```

```

/**
 * Extended JPanel object that is used to display the actual Cayley
 * Table stored in the <CODE>myGroup</CODE> <CODE>groupMatrix</CODE>
object and displayed as
 * via the <CODE>tblGroupTable</CODE> JTable object using the
<CODE>gtmGroupModel</CODE>
 * <CODE>groupTableModel</CODE> object as the format for viewing.
 *
 * @author Jeffrey Barr
 */
public class groupPanel extends javax.swing.JPanel {

    /**
     * Creates new extended JPanel <CODE>groupPanel</CODE>
     */
    public groupPanel() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to
     * initialize groupPanel.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     * This code is autogenerated by the Netbeans code.
     */
    private void initComponents() { //GEN-BEGIN:initComponents
        myGroup = new cayleytable.groupMatrix();
        spnGroupTable = new javax.swing.JScrollPane();
        tblGroupTable = new javax.swing.JTable();

        setLayout(new java.awt.BorderLayout());

        setAutoscrolls(true);
        tblGroupTable.setFont(new java.awt.Font("Microsoft Sans Serif", 0,
12));
        gtmGroupModel = new cayleytable.groupTableModel(this);
        tblGroupTable.setModel(gtmGroupModel);

        tblGroupTable.setAutoResizeMode(javax.swing.JTable.AUTO_RESIZE_OFF);
        tblGroupTable.setAutoscrolls(false);
        tblGroupTable.setRowSelectionAllowed(false);
        spnGroupTable.setViewportView(tblGroupTable);
        tblGroupTable.getAccessibleContext().setAccessibleParent(this);

        add(spnGroupTable, java.awt.BorderLayout.CENTER);

    } //GEN-END:initComponents

    // Variables declaration - do not modify //GEN-BEGIN:variables
    /**
     * <CODE>groupMatrix</CODE> object used to store the actual Cayley
Table being displayed
     * on the panel via the <CODE>tblGroupTable</CODE> JTable object.
     */
}

```

```

private cayleytable.groupMatrix myGroup;
/**
 * JScrollPane object used to extend the size of usable space
available
 * for the <CODE>tblGroupTable</CODE> JTable object being displayed on
the user
 * interface so that groups of higher order can be displayed.
 */
private javax.swing.JScrollPane spnGroupTable;
/**
 * JTable object used to display the actual Cayley Table stored in
<CODE>myGroup</CODE>
 * <CODE>groupMatrix</CODE> object. Formatting of the table is done
via the <CODE>gtmGroupModel</CODE>.
 */
private javax.swing.JTable tblGroupTable;
// End of variables declaration//GEN-END:variables
/**
 * AbstractTableModel object used to describe how the group stored in
 * <CODE>myGroup</CODE> <CODE>groupMatrix</CODE> object is displayed
using the <CODE>tblGroupTable</CODE> <CODE>JTable</CODE>
 * object.
 */
private cayleytable.groupTableModel gtmGroupModel;

/**
 * Method to update both the <CODE>tblGroupTable</CODE> JTable object
and the
 * <CODE>myGroup</CODE> <CODE>groupMatrix</CODE> object with a new
group that is passed in.
 * @param newTable New <CODE>groupMatrix</CODE> object used to update
objects in <CODE>groupPanel</CODE>
 * @param makeEditable Boolean variable used to determine if
individual cells in the table can be changed manually through the user
interface.
 */
public void updateTable(groupMatrix newTable, boolean makeEditable) {
    myGroup.resetGroup(newTable);
    gtmGroupModel.displayTable(newTable, makeEditable);
    DefaultTableCellRenderer renderer = new
DefaultTableCellRenderer();
    renderer.setBackground(Color.BLACK);
    renderer.setForeground(Color.YELLOW);
    renderer.setFont(new Font("SansSerif", Font.PLAIN, 12));

tblGroupTable.getColumnModel().getColumn(0).setCellRenderer(renderer);
tblGroupTable.getTableHeader().setBackground(Color.BLACK);
tblGroupTable.getTableHeader().setForeground(Color.YELLOW);
}

/**
 * Method used by the user interface to update values in
<CODE>myGroup</CODE> <CODE>groupMatrix</CODE>
 * object when individual cells in the <CODE>tblGroupTable</CODE>
JTable object are edited by the user.
 * @param row Current row in Cayley Table that is to be changed.

```

```

    * @param col Current column in Cayley Table that is to be changed.
    * @param value New value to change in the Cayley Table.
    */
    public void changeValueInGroup(int row, int col, int value) {
        myGroup.setEntry(row, col, value);
    }

    /**
    * Returns the <CODE>myGroup</CODE> <CODE>groupMatrix</CODE> object
that is
    * stored and displayed in <CODE>groupPanel</CODE>.
    * @return Current <CODE>myGroup</CODE> <CODE>groupMatrix</CODE>
object that is displayed in <CODE>groupPanel</CODE>.
    */
    public groupMatrix getGroup() {
        return myGroup;
    }
}

```

GROUPTABLEMODEL.JAVA

```

/*
 * groupTableModel.java
 *
 * Created on January 16, 2005, 2:04 PM
 */

package cayleytable;

import javax.swing.table.AbstractTableModel;
import javax.swing.*;
import java.lang.Integer;
import java.util.*;
/**
 * Extended AbstractTableModel class describing how a group stored in
 * a <CODE>groupMatrix</CODE> object is displayed in a JTable object.
 * @author Jeffrey Barr
 */
public class groupTableModel extends AbstractTableModel {

    /**
    * int that contains the current order of the group on display
    */
    private int order;
    /**
    * boolean that determines if the JTable can be manually edited
    */
    private boolean canEditTable;
    /**
    * ArrayList of ArrayList of String objects which contains every cell
    * in the JTable which is the <CODE>groupMatrix</CODE> object passed
to the <CODE>groupTableModel</CODE>
    * constructor.
    */
    private ArrayList<ArrayList<String>> tableRows;

```

```

/**
 * <CODE>groupPanel</CODE> object that contains the JTable that the
current
 * instantiated of <CODE>groupTableModel</CODE> defines.
 */
private cayleytable.groupPanel topPanel;

/**
 * Creates a new instance of groupTableModel with an empty group of
order 0
 * @param inputPanel groupPanel object which contains the JTable
object
 */
public groupTableModel(groupPanel inputPanel)
{
    order = 0;
    canEditTable = false;
    tableRows = new ArrayList<ArrayList<String>>();
    topPanel = inputPanel;
}

/**
 * Method that fills the column headers and cells in the JTable with
the
 * newest <CODE>groupMatrix</CODE> object.
 * @param matrix <CODE>groupMatrix</CODE> object that contains the
newest group to be displayed.
 * @param makeEditable Boolean variable that defines if the newest
group can be manually edited in the JTable
 */
public void displayTable(groupMatrix matrix, boolean makeEditable)
{
    order = matrix.getOrder();
    canEditTable = makeEditable;
    tableRows.clear();
    tableRows = new ArrayList<ArrayList<String>>(order);

    for (int row=0; row<order; row++)
    {
        ArrayList<String> tableCols = new ArrayList<String>(order+1);
        while (tableCols.size() < order+1)
tableCols.add(Integer.toString(-1));
        for (int col=0; col<order+1; col++)
        {
            if (col == 0) tableCols.set(col, Integer.toString(row));
            else tableCols.set(col, Integer.toString(
matrix.getEntry(row, col-1)));
        }
        tableRows.add(row, tableCols);
    }

    fireTableChanged(null);
}

/**
 * Method used to return value to display as column name

```

```

    * for a specific column in the JTable.
    * @param column int value of column name to retrieve
    * @return Name of column based upon position
    */
    public String getColumnName(int column)
    {
        if (column == 0) return (null);
        return Integer.toString(column-1);
    }

    /**
     * Method to return number of rows in the JTable.
     * @return int value containing the order of the group being
displayed.
    */
    public int getRowCount()
    {
        return order;
    }

    /**
     * Method to return number of columns in the JTable.
     * @return int value containing the order of the group + one for each
element in the group and the first column containing the element names.
    */
    public int getColumnCount()
    {
        return (order+1);
    }

    /**
     * Method to set the value of a cell in the JTable as well as resets
     * the value of the same cell in the group stored in
<CODE>topPanel</CODE>.
     * @param aValue Object that is the new value of the cell in the
table.
     * @param rowIndex integer containing row of the cell to change in the
JTable
     * @param columnIndex integer containing column of the cell to change
in the JTable
    */
    public void setValueAt(Object aValue, int rowIndex, int columnIndex)
    {
        try
        {
            int value = Integer.decode((String) aValue).intValue();
            if (value < 0 || value >= order)
                return;
            else
                topPanel.changeValueInGroup(rowIndex, columnIndex-1,
value);
        }
        catch (NumberFormatException e)
        {
            return;
        }
    }

```

```

        ArrayList<String> tableCols;
        tableCols = tableRows.get(rowIndex);
        tableCols.set(columnIndex, (String) aValue);
        tableRows.set(rowIndex, tableCols);

        // Notify table that new data is available for a specific cell
        // Table is then refreshed with this new data.
        fireTableCellUpdated(rowIndex, columnIndex);

    }

    /**
     * Method used to return value to display for a specific row, column
     in the JTable.
     * @param rowIndex integer containing row of cell in JTable to
     retrieve.
     * @param columnIndex integer containing column of cell in JTable to
     retrieve.
     * @return Value of cell in JTable that was retrieved
     */
    public Object getValueAt(int rowIndex, int columnIndex)
    {
        ArrayList tableCols = (ArrayList) tableRows.get(rowIndex);

        if (tableCols.get(columnIndex) == "-1") return "";
        return tableCols.get(columnIndex);
    }

    /**
     * Method to determine if a specific cell is editable on the JTable.
     * @param rowIndex integer containing row of cell in JTable to check.
     * @param columnIndex integer containing column of cell in JTable to
     check.
     * @return Boolean that determines if cell is editable
     */
    public boolean isCellEditable(int rowIndex, int columnIndex)
    {
        if (canEditTable)
            return (columnIndex != 0);

        return false;
    }
}

```

GROUPRELATION.JAVA

```

/*
 * groupRelation.java
 *
 * Created on January 17, 2005, 3:29 PM
 */

```

```

package cayleytable;

import java.util.*;
import java.io.*;
/**
 * Class used to store the relationships between generators of groups
 * being created in the
<CODE>groupCreator.createDefineRelationGroup</CODE> method.
 * @author Jeffrey Barr
 */
public class groupRelation {

    /**
     * String Object that stores the LHS of the relationship equation
     */
    String left;
    /**
     * String Object that stores the RHS of the relationship equation
     */
    String right;

    /**
     * Creates a new instance of <CODE>groupRelation</CODE> containing a
generator
     * raised to some power being equal to the identity element or an
     * empty RHS.
     * @param generator Generator for which the <CODE>groupRelation</CODE>
is being created
     * @param genSize Order of generator for which <CODE>relation</CODE>
is being generated.
     */
    public groupRelation(int generator, int genSize)
    {
        left = new String("");
        right = new String("");

        for (int i=0; i<genSize; i++)
            {
                left = left + Integer.toString(generator, 10);
            }

    }

    /**
     * Creates a new instance of <CODE>groupRelation</CODE> containing a
     * relationship defined completely by the user through the user
interface.
     * @param relationship String object that represents the relationship
being created.
     * @param generatorList ArrayList of String objects that are the order
of each of the generators of the group for use in determining the inverse
of a generator.
     */
    public groupRelation(String relationship, ArrayList generatorList)
    {
        left = new String("");

```

```

right = new String("");
boolean onLeft = true;

char[] relChars = relationship.toCharArray();
for (int i=0; i<relationship.length(); i++)
    {
        int position;
        if (relChars[i] == '=') onLeft = false;
        else if (relChars[i] == ' ') continue;
        else if ((int) relChars[i] >= (int)'a' && (int) relChars[i] <=
(int)'z')
            {
                position = relChars[i] - (int)'a';
                if (onLeft)
                    left = left + Integer.toString(position, 10);
                else
                    right = right + Integer.toString(position, 10);
            }
        else
            {
                position = relChars[i] - (int)'A';
                int size = Integer.parseInt((String)
generatorList.get(position));
                for (int j=0; j < size-1; j++)
                    {
                        if (onLeft)
                            left = left + Integer.toString(position, 10);
                        else
                            right = right + Integer.toString(position,
10);
                    }
            }
    }

}

/**
 * Creates a new instance of <CODE>groupRelation</CODE> containing
 * a "pseudo" commutative defined relationship of the group.
 * @param gen1 Integer representing the first generator in the
"pseudo" commutative defined relationship
 * @param gen2 Integer representing the second generator in the
"pseudo" commutative defined relationship
 * @param rgt String object containing the RHS of the "pseudo"
commutative defined relationship
 * @param generatorList ArrayList of String objects that are the order
of each of the generators of the group for use in determining the inverse
of a generator.
 */
public groupRelation(int gen1, int gen2, String rgt, ArrayList
generatorList)
    {
        left = new String("");
        right = new String("");

        left = left + Integer.toString(gen1, 10);

```

```

left = left + Integer.toString(gen2, 10);

char[] rightChars = rgt.toCharArray();
for (int i=0; i<rgt.length(); i++)
{
    int position;
    if ((int) rightChars[i] >= (int)'a' && (int) rightChars[i] <=
(int)'z')
        {
            position = rightChars[i] - (int)'a';
            right = right + Integer.toString(position, 10);
        }
    else
        {
            position = rightChars[i] - (int)'A';
            int size = Integer.parseInt((String)
generatorList.get(position));
            for (int j=0; j < size-1; j++)
                {
                    right = right + Integer.toString(position, 10);
                }
        }
}

/**
 * Returns LHS of the <CODE>groupRelation</CODE>.
 * @return String object containing LHS of relation equation.
 */
public String getLeft()
{
    return left;
}

/**
 * Returns RHS of the <CODE>groupRelation</CODE>.
 * @return String object containing RHS of relation equation.
 */
public String getRight()
{
    return right;
}
}

```

GROUPMATRIX.JAVA

```

/*
 * groupMatrix.java
 *
 * Created on January 16, 2005, 8:07 PM
 */

```

```

package cayleytable;

import java.util.*;
/**
 * Class used as a storage device for all groups.  Stores the Cayley
 * table as an ArrayList of ArrayList of Strings where each row in
 * the table is one of the ArrayList of Strings and each cell is
 * represented
 * by one String.  The class also has analysis functions to determine
 * if the Cayley Table has group characteristics such as an Identity
 * element, inverse for each element, and associativity.
 * @author Jeffrey Barr
 */
public class groupMatrix {

    /**
     * ArrayList of ArrayList of Strings used for storing the Cayley Table
     * where each row in the table is one of the ArrayList of Strings and
     * each cell is represented by one String.
     */
    private ArrayList<ArrayList<String>> matrixRows;
    /**
     * Order of the group being stored refers to the number of rows and
     * columns in the Cayley Table.
     */
    private int order;

    /**
     * Creates a new instance of groupMatrix with an empty Cayley Table
     * (order 0).
     */
    public groupMatrix() {
        matrixRows = new ArrayList<ArrayList<String>>();
        order = 0;
        for (int row=0; row < order; row++)
            {
                ArrayList<String> matrixCols = new ArrayList<String>();
                for (int col=0; col < order; col++)
                    matrixCols.add("-1");
                matrixRows.add(matrixCols);
            }
    }

    /**
     * Creates a new instance of groupMatrix with a Cayley Table of a
     * specific given order.
     * @param n Order of group to create.
     */
    public groupMatrix(int n) {
        matrixRows = new ArrayList<ArrayList<String>>();
        order = n;
        for (int row=0; row < order; row++)
            {
                ArrayList<String> matrixCols = new ArrayList<String>();
                for (int col=0; col < order; col++)
                    matrixCols.add("-1");
            }
    }
}

```

```

        matrixRows.add(matrixCols);
    }
}

/**
 * Method to update the <CODE>matrixRows</CODE> and <CODE>order</CODE>
of
 * the current group with the values from a new
<CODE>groupMatrix</CODE> object.
 * @param reset <CODE>groupMatrix</CODE> object that contains new
group to update the current group.
 */
public void resetGroup(cayleytable.groupMatrix reset)
{
    resetSize(reset.getOrder());
    for (int row=0; row < order; row++)
    {
        for (int col=0; col < order; col++)
        {
            setEntry(row, col, reset.getEntry(row,col));
        }
    }
}

/**
 * Method to determine if another <CODE>groupMatrix</CODE> object is
equal to the
 * current group stored in this <CODE>groupMatrix</CODE> instance.
 * @param check <CODE>groupMatrix</CODE> object to compare to this
instance.
 * @return Boolean value whose result determines if the two groups
being compared are equal.
 */
public boolean isEqual(cayleytable.groupMatrix check)
{
    if (order != check.getOrder()) return false;
    for (int row=0; row < order; row++)
    {
        for (int col=0; col < order; col++)
        {
            if (getEntry(row, col) != check.getEntry(row,col))
                return false;
        }
    }
    return true;
}

/**
 * Method to reset the groupMatrix object to an Cayley Table of a
given
 * order filled with all -1 values.
 * @param size New order of the Cayley Table.
 */
public void resetSize(int size)
{
    matrixRows.clear();
}

```

```

order = size;
for (int row=0; row < order; row++)
{
    ArrayList<String> matrixCols = new ArrayList<String>();
    try {
        for (int col=0; col < order; col++)
            matrixCols.add("-1");
        matrixRows.add(matrixCols);
    }
    catch (IndexOutOfBoundsException e) {
        System.out.println("Error Index out of Bounds: " +
e.getMessage());
    }
}

/**
 * Method to set the value of a specific cell in the Cayley Table to
 * a given value.
 * @param row integer containing row of the cell to change in the
Cayley Table
 * @param col integer containing column of the cell to change in the
Cayley Table
 * @param value integer containing new value of the cell stored in the
Cayley Table
 */
public void setEntry(int row, int col, int value)
{
    ArrayList<String> matrixCols = matrixRows.get(row);
    matrixCols.set(col, Integer.toString(value));
    matrixRows.set(row, matrixCols);
}

/**
 * Method to get the value of a specific cell in the Cayley Table.
 * @param row integer containing row of the cell to retrieve in the
Cayley Table
 * @param col integer containing column of the cell to retrieve in the
Cayley Table
 * @return integer containing value of the cell retrieved from the
Cayley Table
 */
public int getEntry(int row, int col)
{
    ArrayList<String> matrixCols = matrixRows.get(row);
    return Integer.decode((String) matrixCols.get(col)).intValue();
}

/**
 * Method to determine if all of the cells in the Cayley Table have a
 * value. Does not determine if the values are legal (0 to order-1).
 * @return Boolean with result of the check
 */
public boolean checkComplete()
{
    ArrayList<String> matrixCols;

```

```

        int temp;
        for (int row=0; row < order; row++)
        {
            for (int col=0; col < order; col++)
            {
                matrixCols = matrixRows.get(row);
                if (matrixCols.get(col) == null) // ||
matrixCols.get(col) == "-1")
                    return false;
            }
        }
        return true;
    }

/**
 * Method to return value of <CODE>order</CODE> in this instance of
<CODE>groupMatrix</CODE>
 * @return integer value containing <CODE>order</CODE> of the Cayley
Table.
 */
public int getOrder()
{
    return order;
}

/**
 * Method to determine the identity element of the group stored in the
 * Cayley Table.
 * @return integer value of the identity element
 */
public int findIdentity()
{
    if (!checkForIdentity()) return -1;

    for (int row=0; row < order; row++)
    {
        int col = 0;
        while (col < order && getEntry(row, col) == col) col++;
        if (col >= order)
            return row;
    }

    return -1;
}

/**
 * Method to return inverse of an element in the group
 * @param element integer element in group of which to find inverse
 * @return integer value of the inverse
 */
public int findInverse(int element)
{
    int identity = findIdentity();
    for (int row=0; row < order; row++)
    {
        if (getEntry(row, element) == identity)

```

```

        return row;
    }
    return order+1;
}

/**
 * Method to determine if the Cayley Table contains an identity
element
 * @return Boolean result of the check
 */
public boolean checkForIdentity()
{
    int columnID = -1;
    int rowID = -1;

    // Check for the row that is the Identity;
    for (int row=0; row < order; row++)
    {
        // Check if the row returns it's relative column position
        int col = 0;
        while (col < order && getEntry(row, col) == col) col++;
        if (col >= order)
        {
            rowID = row;
            break;
        }
    }

    // Check if row Identity element was found
    if (rowID < 0) return false;

    // Find the column in the zeroth row that returns the value of the
zeroth row position
    columnID = 0;
    while (columnID < order && getEntry(0, columnID) != 0)
        columnID++;

    // If no column in the zeroth row is 0, then there is no identity
column in the table
    if (columnID >= order)
        return false;

    // Check the column on the remaining rows to ensure it is the
identity
    for (int row=1; row < order; row++)
    {
        // If this column continues to return the current row position
for all rows
        // continue, else return that there is no true identity column
        if (getEntry(row, columnID) != row) return false;
    }

    // The identity row and column should be the same
    if (rowID == columnID) return true;
    else return false;
}

```

```

}

/**
 * Method to determine if all elements in the Cayley Table have an
 * inverse element.
 * @return Boolean result of the check
 */
public boolean checkForInverse()
{
    int identity = findIdentity();
    if (identity == -1) return false;

    // Check for the row that is the Identity;
    for (int row=0; row < order; row++)
    {
        int inverse = findInverse(row);
        // No inverse found for this row value so return false
        if (inverse >= order)
        {
            return false;
        }
        // The inverse should work both ways so ensure it does, else
return false
        else if (getEntry(inverse,row) != identity)
            return false;
    }
    return true;
}

/**
 * Method to determine if all operations in the Cayley Table are
 * associative.
 * @return Boolean result of the check
 */
public boolean checkIfAssociative()
{
    for (int x=0; x<order; x++)
    {
        for (int y=0; y<order; y++)
        {
            for (int z=0; z<order; z++)
            {
                // For associative property check that (x * (y *
z)) = ((x * y) * z)
                if (getEntry(x, getEntry(y,z)) !=
getEntry(getEntry(x,y), z))
                    return false;
            }
        }
    }
    return true;
}

/**
 * Method to determine if all operations in the Cayley Table are
 * commutative.

```

```

    * @return Boolean result of the check
    */
public boolean checkIfCommutative()
{
    for (int x=0; x<order; x++)
    {
        for (int y=0; y<order; y++)
        {
            // For commutative property check that (x * y) = (y *
x)
            if (getEntry(x, y) != getEntry(y, x))
                return false;
        }
    }
    return true;
}

/**
 * Method to determine if all rows and columns in the Cayley Table
have
 * unique elements for each cell in the row or column.
 * @return Boolean result of the check
 */
public boolean checkForUniquenessInRowAndCol()
{
    // Checks that each row never repeats a value within the row
    for (int row=0; row<order; row++)
    {
        // Get the row and place it into an array
        int rowArray[] = new int[order];
        for (int col=0; col < order; col++)
            rowArray[col] = getEntry(row, col);

        // Check that the array does not repeat a value
        if (!checkArray(rowArray, order)) return false;
    }

    // Checks that each col never repeats a value within the col
    for (int col=0; col<order; col++)
    {
        // For each column, get the value from each row
        // for that column and place it into an array
        int colArray[] = new int[order];
        for (int row=0; row<order; row++)
        {
            colArray[row] = getEntry(row, col);
        }

        // Check that the array does not repeat a value
        if (!checkArray(colArray, order)) return false;
    }

    return true;
}

```

```

/**
 * Method to check if a given array has a unique set of elements that
 * never repeat.
 * @param values Array of integers that is being checked.
 * @param size integer size of array being checked
 * @return Boolean result of the check
 */
private boolean checkArray(int[] values, int size)
{
    // Sort the array
    java.util.Arrays.sort(values);

    // Determine if the sorted array repeats any values
    for (int i=0; i<size-1; i++)
        {
            if (values[i] == values[i+1]) return false;
        }
    return true;
}
}

```

GROUPCREATOR.JAVA

```

/*
 * groupCreator.java
 *
 * Created on January 17, 2005, 12:57 PM
 */

package cayleytable;

import java.util.*;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/**
 * Class used to create and store a <CODE>group</CODE> in
 * <CODE>groupMatrix</CODE> object.
 * @author Jeffrey Barr
 */
public class groupCreator {

    /**
     * <CODE>groupMatrix</CODE> object that is created and stored by this
     * class.
     */
    private cayleytable.groupMatrix group;

    /**
     * Constructor that creates a new instance of
     * <CODE>groupCreator</CODE> with an empty

```

```

    * <CODE>group</CODE> <CODE>groupMatrix</CODE> object.
    */
    public groupCreator() {
        group = new cayleytable.groupMatrix(0);
    }

    /**
     * Constructor that creates a new instance of
     <CODE>groupCreator</CODE> with
     * a <CODE>group</CODE> <CODE>groupMatrix</CODE> object of specific
     order.
     * @param size Order of empty <CODE>groupMatrix</CODE> object to
     create
     */
    public groupCreator(int size)
    {
        group = new cayleytable.groupMatrix(size);
    }

    /**
     * Returns <CODE>group</CODE> <CODE>groupMatrix</CODE> object created
     by <CODE>groupCreator</CODE>.
     * @return Current <CODE>group</CODE> <CODE>groupMatrix</CODE> object
     created and stored in the class.
     */
    public cayleytable.groupMatrix getGroup()
    {
        return group;
    }

    /**
     * Resets the <CODE>group</CODE> <CODE>groupMatrix</CODE> object in
     the class <CODE>groupCreator</CODE>
     * with a new <CODE>groupMatrix</CODE> object passed to the method.
     * @param newGroup <CODE>groupMatrix</CODE> object that is to replace
     the <CODE>group</CODE> <CODE>groupMatrix</CODE> object.
     * @return Boolean value representing success of updating the
     <CODE>group</CODE> <CODE>groupMatrix</CODE> object.
     */
    public boolean resetGroup(cayleytable.groupMatrix newGroup)
    {
        if (group.getOrder() != newGroup.getOrder())
            group.resetSize(newGroup.getOrder());

        for (int row=0; row<group.getOrder(); row++)
        {
            for (int col=0; col < group.getOrder(); col++)
            {
                group.setEntry(row, col, newGroup.getEntry(row,col));
            }
        }
        return group.checkComplete();
    }
}

/**

```

```

    * Method to update <CODE>group</CODE> <CODE>groupMatrix</CODE> object
with a new empty group of
    * given order.
    * @param size Order of empty <CODE>group</CODE>
<CODE>groupMatrix</CODE> object to create and store.
    * @return Boolean value representing success of updating the
<CODE>group</CODE> <CODE>groupMatrix</CODE> object.
    */
    public boolean createEmptyGroup(int size)
    {
        int order = group.getOrder();
        if (order != size) group.resetSize(size);
        return group.checkComplete();
    }

    /**
    * Method to update <CODE>group</CODE> <CODE>groupMatrix</CODE> object
with a
    * new cyclic group of given order.
    * @param size Order of cyclic <CODE>group</CODE>
<CODE>groupMatrix</CODE> object to create and store.
    * @return Boolean value representing success of updating the
<CODE>group</CODE> <CODE>groupMatrix</CODE> object.
    */
    public boolean createCyclicGroup(int size)
    {
        int order = group.getOrder();
        if (order != size) group.resetSize(size);
        for (int row=0; row < size; row++)
        {
            for (int col=0; col < size; col++)
            {
                group.setEntry(row, col, (row+col)%size);
            }
        }
        return group.checkComplete();
    }

    /**
    * Method to update <CODE>group</CODE> <CODE>groupMatrix</CODE> object
with a
    * new cross product group that is a combination of two groups that
are passed
    * to the method.
    * @param g1 First <CODE>groupMatrix</CODE> object passed to the
method for cross product calculation.
    * @param g2 Second <CODE>groupMatrix</CODE> object passed to the
method for cross product calculation.
    * @return Boolean value representing success of updating the
<CODE>group</CODE> <CODE>groupMatrix</CODE> object.
    */
    public boolean createXProdGroup(cayleytable.groupMatrix g1,
cayleytable.groupMatrix g2)
    {
        int x1, x2;           // positions in g1 table
        int y1, y2;           // positions in g2 table

```

```

int x,y;                // results of g1 and g2 tables

int size = g1.getOrder() * g2.getOrder();
if (size != group.getOrder()) group.resetSize(size);

int[][] cross_prod_set = new int[size][2];
int count = 0;
for (int i=0; i < g1.getOrder(); i++)
    {
        for (int j=0; j < g2.getOrder(); j++)
            {
                cross_prod_set[count][0] = i;
                cross_prod_set[count][1] = j;
                count++;
            }
    }

if (count != size) return false;

for (int row=0; row < size; row++)
    {
        for (int col=0; col < size; col++)
            {
                x1 = cross_prod_set[row][0];
                x2 = cross_prod_set[col][0];
                x = g1.getEntry(x1, x2);
                y1 = cross_prod_set[row][1];
                y2 = cross_prod_set[col][1];
                y = g2.getEntry(y1, y2);

                boolean found = false;
                for (int k=0; k<size && found==false; k++)
                    {
                        if (x == cross_prod_set[k][0] &&
                            y == cross_prod_set[k][1])
                            {
                                found = true;
                                group.setEntry(row, col, k);
                            }
                    }

                if (found != true) return false;
            }
    }

return group.checkComplete();
}

/**
 * Method to update <CODE>group</CODE> <CODE>groupMatrix</CODE> object
with a
 * new defined relationship group that is defined by relationships and
generators
 * that are passed to the method.

```

```

    * @param generatorList ArrayList of String objects that are the order
of each of the generators of the group.
    * @param relationships ArrayList of <CODE>groupRelation</CODE>
objects that contain the relationships between the generators.
    * @return Boolean value representing success of updating the
<CODE>group</CODE> <CODE>groupMatrix</CODE> object.
    */
    public boolean createDefineRelationGroup(ArrayList<String>
generatorList, ArrayList<groupRelation> relationships)
    {
        int size = 1;

        for (int i=0; i<generatorList.size(); i++)
            {
                size *= Integer.parseInt(generatorList.get(i));
            }

        if (size != group.getOrder()) group.resetSize(size);

        ArrayList finalElements = createFinalElementList(generatorList);
        //      System.out.println("Final Elements created");

        for (int row=0; row<finalElements.size(); row++)
            {
                for (int col=0; col < finalElements.size(); col++)
                    {
                        String tempWord = new String((String)
finalElements.get(row) + (String) finalElements.get(col));
                        //System.out.println("tempWord = " + tempWord + "\trow =
" + row + "\tcol = " + col);
                        boolean stillReducing = true;
                        while (stillReducing)
                            {
                                int k = 0;
                                boolean substMade = false;
                                while (k < relationships.size() && (!substMade))
                                    {
                                        groupRelation curRelation = (groupRelation)
relationships.get(k);

                                        k = k+1;
                                        int loc =
tempWord.indexOf(curRelation.getLeft());
                                        StringBuffer tempBuffer = new
StringBuffer(tempWord);

                                        if (loc >= 0)
                                            {
                                                tempBuffer.replace(loc,
loc+curRelation.getLeft().length(), curRelation.getRight());
                                                tempWord = tempBuffer.toString();
                                                substMade = true;
                                            }
                                    }
                                if (!substMade) stillReducing = false;
                            }

                        int count = 0;

```

```

        boolean elementFound = false;
        //System.out.println("Reduced tempWord = " + tempWord +
"\trow = " + row + "\tcol = " + col);
        while ((!elementFound) && count < finalElements.size())
        {
            String tempFinal = (String)
finalElements.get(count);
            if (tempWord.trim().compareTo(tempFinal.trim()) ==
0)
                {
                    group.setEntry(row, col, count);
                    elementFound = true;
                }
            if (!elementFound) count = count + 1;
        }

        if (!elementFound) return false;
    }
}
return group.checkComplete();
}

/**
 * Method to determine all the elements of a group given the orders of
 * a list of generators of the group being created in
<CODE>createDefineRelationGroup</CODE>
 * @param generatorList ArrayList of String objects that are the order
of each of the generators of the group.
 * @return The final list of elements that comprise the group being
created in <CODE>createDefineRelationGroup</CODE>
 */
private ArrayList createFinalElementList(ArrayList<String>
generatorList)
{
    ArrayList<String> finalElements = new ArrayList<String>();
    int size = generatorList.size();
    String str = new String("");
    int gens[] = new int[size];
    ArrayList<ArrayList<String>> elementMatrix = new
ArrayList<ArrayList<String>>();

    for (int i=0; i<size; i++)
    {
        str = "";
        ArrayList<String> elementRow = new ArrayList<String>();
        for (int j=0; j < Integer.parseInt(generatorList.get(i)); j++)
        {
            elementRow.add(str);
            str = Integer.toString(i);
        }
        elementMatrix.add(elementRow);
    }

    int count = 0;
    String lastElement = new String("");
    String nextElement = new String("");

```

```

        for (int k=0; k < Integer.parseInt(generatorList.get(count)); k++)
        {
            ArrayList<String> elementRow = elementMatrix.get(count);
            nextElement = nextElement + elementRow.get(k);
            finalElements = fillList(finalElements, elementMatrix,
generatorList, nextElement, count+1);
        }

        // for (int x=0; x < finalElements.size(); x++)
System.out.println(x + ":\t" + (String)finalElements.get(x));

        return finalElements;
    }

    /**
     * Recursive method used by <CODE>createFinalElementList</CODE> to
determine
     * the final list of elements that are to be
     * @param finalElements Current list of all elements that have thus
far been derived for the group.
     * @param elementMatrix ArrayList of ArrayList of String objects where
     *     each String is one of the possible elements derived from one
generator
     *     each ArrayList of Strings is the list of all possible elements
derived from one generator
     *     and each ArrayList of ArrayList of Strings is the list of all
possible elements derived from one generator for all the generators
     * @param generatorList ArrayList of String objects that are the order
of each of the generators of the group.
     * @param nextElement Current element being built recursively for the
final element list
     * @param count Number of recursive steps that has been run of
<CODE>fillList</CODE>
     * @return The final list of elements thus far derived for the group
that comprise the group being created in
<CODE>createDefineRelationGroup</CODE>
     */
    private ArrayList<String> fillList(ArrayList<String> finalElements,
ArrayList<ArrayList<String>> elementMatrix,
                                ArrayList<String> generatorList, String
nextElement, int count)
    {
        if (count >= generatorList.size())
            finalElements.add(nextElement);
        else
        {
            for (int i=0; i < Integer.parseInt(generatorList.get(count));
i++)
                {
                    ArrayList<String> elementRow = elementMatrix.get(count);
                    nextElement = nextElement + elementRow.get(i);
                    finalElements = fillList(finalElements, elementMatrix,
generatorList, nextElement, count+1);
                }
        }
        return finalElements;
    }

```

```

}

/**
 * Method to calculate and store the inner automorphism of the
<CODE>group</CODE>
 * <CODE>groupMatrix</CODE> object.
 * @return Boolean value representing success of updating the
<CODE>group</CODE> <CODE>groupMatrix</CODE> object.
 */
public boolean createInnerAutGroup()
{
    int order = group.getOrder();

    ArrayList<groupMatrix> innerGroups = new ArrayList<groupMatrix>();
    int innerGroupCount = 0;
    int[] groupPosn = new int[order];
    for (int g=0; g<order; g++)
    {
        boolean inList = false;
        cayleytable.groupMatrix tempGroup =
createInnerAutFromElement(g);
        if (g!=0)
        {
            for (int i=0; i<innerGroups.size(); i++)
            {
                cayleytable.groupMatrix listGroup =
innerGroups.get(i);
                if (listGroup.isEqual(tempGroup)) inList = true;
            }
            if (!inList)
            {
                groupPosn[innerGroupCount++] = g;
                innerGroups.add(tempGroup);
            }
        }

        cayleytable.groupMatrix nextGroup = new
cayleytable.groupMatrix(innerGroups.size());

        for (int row=0; row<innerGroups.size(); row++)
        {
            for (int col=0; col<innerGroups.size(); col++)
            {
                cayleytable.groupMatrix tempGroup =
createInnerAutFromTwoElements(groupPosn[row], groupPosn[col]);
                for (int i=0; i<innerGroups.size(); i++)
                {
                    cayleytable.groupMatrix listGroup =
innerGroups.get(i);
                    if (listGroup.isEqual(tempGroup))
                    {
                        nextGroup.setEntry(row, col, i);
                    }
                }
            }
        }
    }
}

```

```

    }

    if (nextGroup.checkComplete())
    {
        group.resetGroup(nextGroup);
        return true;
    }

    return false;
}

/**
 * Method to calculate the inner automorphism of for two elements of
 * the <CODE>group</CODE> <CODE>groupMatrix</CODE> object. Each cell
 * in the group now equals the value of  $xygy^{-1}x^{-1}$  where x and y are
 * the elements and g is current element in the cell of the group for
 * which the inner automorphism is calculated.
 * @param elmtA First integer representing element in group for which
the inner automorphism is calculated
 * @param elmtB Second integer representing element in group for which
the inner automorphism is calculated
 * @return <CODE>groupMatrix</CODE> object that holds the inner
automorphism for the <CODE>group</CODE> <CODE>groupMatrix</CODE> object.
 */
public cayleytable.groupMatrix createInnerAutFromTwoElements(int
elmtA, int elmtB)
{
    int order = group.getOrder();
    cayleytable.groupMatrix tempGrp = new
cayleytable.groupMatrix(order);

    // Determine the inverse of the element if it exists,
    // else return the empty group
    int invA = -1;
    int invB = -1;
    int identity = group.findIdentity();
    if (identity == -1) return tempGrp;
    for (int i=0; i<order; i++)
    {
        if (group.getEntry(elmtA, i) == identity)
            invA = i;
        if (group.getEntry(elmtB, i) == identity)
            invB = i;
    }
    if (invA == -1 || invB == -1) return tempGrp;

    // Create the Inner Automorphism for the element
    for (int row=0; row < order; row++)
    {
        for (int col=0; col < order; col++)
        {
            tempGrp.setEntry(row, col,
group.getEntry(group.getEntry(elmtA, group.getEntry(group.getEntry(elmtB,
group.getEntry(row, col))), invB)), invA));
        }
    }
}

```

```

    return tempGrp;
}

/**
 * Method to calculate the inner automorphism of a single element of
 * the <CODE>group</CODE> <CODE>groupMatrix</CODE> object. Each cell
 * in the group now equals the value of  $xgx^{-1}$  where x is the single
 * element and g is current element in the cell of the group for which
 * the inner automorphism is calculated.
 * @param element Integer representing element in group for which the
inner automorphism is calculated
 * @return <CODE>groupMatrix</CODE> object that holds the inner
automorphism group for an element.
 */
public cayleytable.groupMatrix createInnerAutFromElement(int element)
{
    int order = group.getOrder();
    cayleytable.groupMatrix tempGrp = new
cayleytable.groupMatrix(order);

    // Determine the inverse of the element if it exists,
    // else return the empty group
    int inverse = -1;
    int identity = group.findIdentity();
    if (identity == -1) return tempGrp;
    for (int i=0; i<order; i++)
    {
        if (group.getEntry(element, i) == identity)
        {
            inverse = i;
            break;
        }
    }
    if (inverse == -1) return tempGrp;

    // Create the Inner Automorphism for the element
    for (int row=0; row < order; row++)
    {
        for (int col=0; col < order; col++)
        {
            tempGrp.setEntry(row, col,
group.getEntry(group.getEntry(element, group.getEntry(row, col)),
inverse));
        }
    }
    return tempGrp;
}
}

```

GROUPIDENTIFY.JAVA

```

/*
 * groupIdentify.java
 *
 * Created on January 16, 2005, 8:00 PM
 */

package cayleytable;

import java.util.*;
/**
 * Class used to identify by name the current <CODE>groupMatrix</CODE>
object that
 * it contains.
 * @author Jeffrey Barr
 */
public class groupIdentify {

    /**
     * <CODE>groupMatrix</CODE> object that is to be identified.
     */
    private cayleytable.groupMatrix identifyMatrix;
    /**
     * String object containing name of current <CODE>groupMatrix</CODE>
object stored
     * in the class.
     */
    private String name;
    /**
     * Boolean result of the identification routine
     */
    private boolean identified;

    /**
     * Creates a new instance of <CODE>groupIdentify</CODE> with a
<CODE>groupMatrix</CODE> object
     * of order 0 and no name.
     */
    public groupIdentify() {
        identifyMatrix = new cayleytable.groupMatrix(0);
        String name = new String("emptyGroup");
        identified = false;
    }

    /**
     * Method to update the <CODE>identifyMatrix</CODE>
<CODE>groupMatrix</CODE> object stored in the class as well
     * as the method to actually identify the group stored in the
<CODE>identifyMatrix</CODE> <CODE>groupMatrix</CODE>
     * object.
     * @param myGroup <CODE>groupMatrix</CODE> object to be identified
     */
    public void resetGroupIdentify (cayleytable.groupMatrix myGroup)
    {
        identifyMatrix.resetGroup(myGroup);
    }
}

```

```

int order[] = build_order_array();
ArrayList<num_order> orderList = build_num_order_arrayList(order);

    showCenterOrders(computeCenter(), order);
    if (isCyclic(order))
        {
            name = "Z" + Integer.toString(identifyMatrix.getOrder());
            identified = true;
        }
    else
        {
            if (isTwoPrime(identifyMatrix.getOrder()))
                {
                    if (identifyMatrix.getOrder() != 4)
                        name = "D" +
Integer.toString(identifyMatrix.getOrder() / 2);
                    else
                        name = "Klein 4";
                    identified = true;
                }
            else if (isPrimeSqr(identifyMatrix.getOrder()))
                {
                    int root = (int)
java.lang.Math.round(java.lang.Math.sqrt(identifyMatrix.getOrder()));
                    name = "Z" + Integer.toString(root) + " x Z" +
Integer.toString(root);
                    identified = true;
                }
            else
                {
                    if (identifyMatrix.checkIfCommutative())
                        name = identifyAbelianXGroup(order, orderList);
                    else
                        name = identifyNonAbelianGroup(order, orderList);
                }
        }
    System.out.print("Orders: ");
    for (int j=0; j<orderList.size(); j++)
        {
            num_order myOrders = orderList.get(j);
            System.out.print("\t" + myOrders.getOrderOfElement() + ": " +
myOrders.getNumOfElements());
        }
    System.out.println();
}

/**
 * Method to return name of group stored in
<CODE>identifyMatrix</CODE> <CODE>groupMatrix</CODE> object
 * @return String containing name of the group
 */
public String getName()
{
    return name;
}

```

```

/**
 * Method to return result of whether the group in
<CODE>identifyMatrix</CODE>
 * <CODE>groupMatrix</CODE> object could be identified.
 * @return boolean result of identification routine.
 */
public boolean isIdentified()
{
    return identified;
}

/**
 * Method to determine the order of each of the elements in the group.
 * @return Array of integers containing order of all elements in the
group
 */
private int[] build_order_array()
{
    int[] order = new int[identifyMatrix.getOrder()];
    int j;

    int i = 0;
    while (i < identifyMatrix.getOrder())
    {
        j = i;
        order[i] = 1;
        do
        {
            if (j != identifyMatrix.findIdentity()) order[i] =
order[i] + 1;
            j = identifyMatrix.getEntry(i, j);
        }
        while (j != i);
        i = i + 1;
    }

    return order;
}

/**
 * Method to determine the number of elements in the group of each
order.
 * @param order Array of integers containing order of all elements in
the group
 * @return ArrayList of <CODE>groupIdentify.numOrder</CODE> that
contains the number of elements of a specific order of the group.
 */
private ArrayList<num_order> build_num_order_arrayList(int[] order)
{
    ArrayList<num_order> orderList = new ArrayList<num_order>();
    for (int x = 1; x <= identifyMatrix.getOrder(); x++)
    {
        int count = 0;
        for (int y = 0; y < identifyMatrix.getOrder(); y++)
            if (order[y] == x) count = count + 1;
    }
}

```

```

        if (count != 0)
        {
            cayleytable.groupIdentify.num_order next = new
cayleytable.groupIdentify.num_order(x, count);
            orderList.add(next);
        }
    }

    return orderList;
}

/**
 * Method to determine if current group is a cyclic group.
 * @param order Array of integers containing order of all elements in
the group
 * @return Boolean result of check to determine if the group is
cyclic.
 */
private boolean isCyclic(int[] order)
{
    for (int i=0; i<identifyMatrix.getOrder(); i++)
    {
        if (order[i] == identifyMatrix.getOrder())
            return true;
    }
    return false;
}

/**
 * Method to determine if a given value is prime
 * @param n integer value to check
 * @return Boolean result of check for prime
 */
private boolean isPrime(int n)
{
    int divisor = 2;
    double max = java.lang.Math.sqrt(n);

    while ((double) divisor <= max) {
        if (n % divisor == 0) return false;
        divisor = divisor + 1;
    }
    return true;
}

/**
 * Method to determine if a given value is 2*prime
 * @param n integer value to check
 * @return Boolean result of check for 2*prime
 */
private boolean isTwoPrime(int n)
{
    if (n % 2 == 0) return isPrime(n / 2);
    return false;
}

```

```

/**
 * Method to determine if a given value is prime^2
 * @param n integer value to check
 * @return Boolean result of check for prime^2
 */
private boolean isPrimeSqrD(int n)
{
    int root = (int) java.lang.Math.round(java.lang.Math.sqrt(n));

    if (n == (root*root)) return isPrime(root);

    return false;
}

/**
 * Method to determine remaining Abelian groups that have not been
identified
 * by the <CODE>isCyclic</CODE>, <CODE>isTwoPrime</CODE>, and
<CODE>isPrimeSqrD</CODE> methods.
 * All remaining Abelian groups are of the cross product form.
 * @param order Array of integers containing order of all elements in
the group
 * @param orderList ArrayList of <CODE>groupIdentify.numOrder</CODE>
that contains the number of elements of a specific order of the group.
 * @return String object containing name of the Abelian group
 */
private String identifyAbelianXGroup(int[] order, ArrayList<num_order>
orderList)
{
    int max_order = 0;
    int temp;
    ArrayList<factorType> factors, tempFactors;
    cayleytable.groupIdentify.factorType nextFactor, nextTempFactor;
    String name;

    for (int i=0; i<identifyMatrix.getOrder(); i++)
        if (order[i] > max_order) max_order = order[i];

    temp = (int) identifyMatrix.getOrder() / max_order;

    factors = primeFactor(temp);
    name = new String("");

    for (int j=0; j < factors.size(); j++)
    {
        nextFactor = factors.get(j);
        if (nextFactor.getNumOfPrimes() == 1)
            name = name + "Z" +
Integer.toString(nextFactor.getPrime()) + " x ";
        else
        {
            temp = numElementsOfOrder(orderList,
nextFactor.getPrime()) + 1;
            tempFactors = primeFactor(temp);
            nextTempFactor = tempFactors.get(j);

```

```

        temp = nextTempFactor.getNumOfPrimes() - 1;
        name = name + generateUniqueFactor(nextFactor, temp);
    }
}

name = name + "Z" + Integer.toString(max_order);
identified = true;
return name;
}

/**
 * Method to determine the prime factorization of a given value
 * @param n integer value to factor
 * @return ArrayList of <CODE>groupIdentify.factorType</CODE>
containing prime factors and the number of those primes used in the
factorization
 */
private ArrayList<factorType> primeFactor(int n)
{
    ArrayList<factorType> factors = new ArrayList<factorType>();

    int i = 2;
    while (n > 1)
    {
        cayleytable.groupIdentify.factorType nextFactor = new
cayleytable.groupIdentify.factorType(i, 0);
//        if ((n % i) == 0)
//        {
            while (n % i == 0)
            {
                nextFactor.incrementNumOfPrimes();
                n = n / i;
            }
//        }
        if (nextFactor.getNumOfPrimes() > 0)
            factors.add(nextFactor);
        i = i + 1;
        while (!isPrime(i)) i = i + 1;
    }

    return factors;
}

/**
 * Method to determine the number of elements in a group of a given
order.
 * @param orderList ArrayList of <CODE>groupIdentify.numOrder</CODE>
that contains the number of elements of a specific order of the group.
 * @param k integer of order to check
 * @return integer of number of elements of the given order
 */
private int numElementsOfOrder(ArrayList<num_order> orderList, int k)
{
    cayleytable.groupIdentify.num_order nextOrder;

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

```

```

        {
            nextOrder = orderList.get(i);
            if (nextOrder.getOrderOfElement() == k)
                return nextOrder.getNumOfElements();
        }

    return 0;
}

/**
 * Method to generate remaining factors based upon a specific number
of a prime order
 * and a number or terms those primes should be combined into.
 *
 * @param nextFactor groupIdentify.factorType containing next prime
and the number of primes that are to be used
 * @param terms integer providing number of terms in the cross product
to be returned
 * @return String object containing elements of the cross product in
the name of the group
 */
private String
generateUniqueFactor(cayleytable.groupIdentify.factorType nextFactor, int
terms)
{
    int[] xTermArray = new int[terms];
    int index = terms ;
    String name = new String("");

    for (int i=0; i < terms; i++) xTermArray[i] = 1;

    for (int j=0; j < nextFactor.getNumOfPrimes(); j++)
    {
        index = index - 1;
        xTermArray[index] = xTermArray[index] * nextFactor.getPrime();
        if (index == 0) index = terms;
    }

    for (int k=0; k < terms; k++)
        name = name + "Z" + Integer.toString(xTermArray[k]) + " x ";

    return name;
}

/**
 * Method to determine remaining non-Abelian group names for all
groups of order less
 * than or equal to 32. Will return error message in String and set
<CODE>identified</CODE>
 * to false if name cannot be determined or order is greater than 32.
 * @param order Array of integers containing order of all elements in
the group
 * @param orderList ArrayList of <CODE>groupIdentify.numOrder</CODE>
that contains the number of elements of a specific order of the group.
 * @return String object containing name of the non-Abelian group
 */

```

```

private String identifyNonAbelianGroup(int[] order,
ArrayList<num_order> orderList)
{
String name = new String("");
identified = true;
switch(identifyMatrix.getOrder())
{
case 8:
if (numElementsOfOrder(orderList, 4) == 6)
name = "Quaternion";
else if (numElementsOfOrder(orderList, 4) == 2)
name = "D4";
else {
identified = false;
name = "ERROR - order 8";
}
break;
case 12:
if (numElementsOfOrder(orderList, 2) == 7)
name = "D6";
else if (numElementsOfOrder(orderList, 3) == 8)
name = "A4";
else if (numElementsOfOrder(orderList, 3) == 2)
name = "<2,2,3>";
else {
identified = false;
name = "ERROR - order 12";
}
break;
case 16:
if (numElementsOfOrder(orderList, 2) == 9)
name = "D8";
else if (numElementsOfOrder(orderList, 2) == 11)
name = "Z2 x D4";
else if (numElementsOfOrder(orderList, 2) == 1)
name = "Q4";
else if (numElementsOfOrder(orderList, 8) == 8)
name = "Z2 xo Z8";
else if (numElementsOfOrder(orderList, 2) == 5)
name = "Z2 xi Z8";
else if (numElementsOfOrder(orderList, 2) == 7)
{
if (centerIsKlein4(computeCenter(), order))
name = "Weird2";
else
name = "Weird1";
}
else if (numElementsOfOrder(orderList, 4) == 12)
{
if (subGroupsAllNormal())
name = "Z2 x Quaternion";
else
name = "Z4 xo Z4";
}
else {
identified = false;
}
}
}

```

```

        name = "ERROR - order 16";
    }
    break;
case 18:
    if ((numElementsOfOrder(orderList, 2) == 9) &&
(numElementsOfOrder(orderList, 3) == 2))
        name = "D9";
    else if ((numElementsOfOrder(orderList, 2) == 9) &&
(numElementsOfOrder(orderList, 3) == 8))
        name = "((3,3,3;2))";
    else if (numElementsOfOrder(orderList, 2) == 3)
        name = "Z3 x D3";
    else {
        identified = false;
        name = "ERROR - order 18";
    }
    break;
case 20:
    if (numElementsOfOrder(orderList, 2) == 11)
        name = "D10";
    else if (numElementsOfOrder(orderList, 2) == 1)
        name = "<2,2,5>";
    else if (numElementsOfOrder(orderList, 2) == 5)
        name = "K-Metacyclic (20)";
    else {
        identified = false;
        name = "ERROR - order 20";
    }
    break;
case 21:
    if (numElementsOfOrder(orderList, 3) == 14)
        name = "Z3 xo Z7";
    else {
        identified = false;
        name = "ERROR - order 21";
    }
    break;
case 24:
    if (numElementsOfOrder(orderList, 2) == 13)
        name = "D12";
    else if ((numElementsOfOrder(orderList, 6) == 8) &&
(numElementsOfOrder(orderList, 2) == 7))
        name = "Z2 x A4";
    else if (numElementsOfOrder(orderList, 2) == 15)
        name = "Z2 x D6";
    else if (numElementsOfOrder(orderList, 2) == 5)
        name = "Z3 x D4";
    else if (numElementsOfOrder(orderList, 12) == 12)
        name = "Z3 x Quaternion";
    else if (numElementsOfOrder(orderList, 4) == 8)
        name = "Z4 x D3";
    else if (numElementsOfOrder(orderList, 4) == 12)
        name = "Z2 x <2,2,3>";
    else if ((numElementsOfOrder(orderList, 2) == 9) &&
(numElementsOfOrder(orderList, 3) == 2))
        name = "(4,6|2,2)";

```

```

        else if ((numElementsOfOrder(orderList, 2) == 9) &&
(numElementsOfOrder(orderList, 3) == 8))
            name = "S4";
        else if ((numElementsOfOrder(orderList, 2) == 1) &&
(numElementsOfOrder(orderList, 6) == 8))
            name = "<2,3,3>";
        else if (numElementsOfOrder(orderList, 4) == 14)
            name = "<2,2,6>";
        else if (numElementsOfOrder(orderList, 8) == 12)
            name = "<-2,2,3>";
        else {
            identified = false;
            name = "ERROR - order 24";
        }
        break;
case 27:
    if (numElementsOfOrder(orderList, 3) == 26)
        name = "(3,3|3,3)";
    else if (numElementsOfOrder(orderList, 3) == 8)
        name = "Weird 27";
    else {
        identified = false;
        name = "ERROR - order 27";
    }
    break;
case 28:
    if (numElementsOfOrder(orderList, 2) == 15)
        name = "D14";
    else if (numElementsOfOrder(orderList, 2) == 1)
        name = "<2,2,7>";
    else {
        identified = false;
        name = "ERROR - order 28";
    }
    break;
case 30:
    if (numElementsOfOrder(orderList, 2) == 15)
        name = "D15";
    else if (numElementsOfOrder(orderList, 2) == 5)
        name = "Z3 x D5";
    else if (numElementsOfOrder(orderList, 2) == 3)
        name = "Z5 x D3";
    else {
        identified = false;
        name = "ERROR - order 30";
    }
    break;
case 32:
    showCenterOrders(computeCenter(), order);
    if (subGroupsAllNormal()) System.out.println("All
Normal");
        else System.out.println("Not All Normal:");
    if (numElementsOfOrder(orderList, 2) == 23)
        name = "Z2 x Z2 x D4 - Hall Senior Number 8 for Order 32";
    else if (numElementsOfOrder(orderList, 16) == 16)
        name = "Z16 xo Z2 - Hall Senior Number 22 for Order 32";

```

```

        else if (numElementsOfOrder(orderList, 2) == 19 &&
numElementsOfOrder(orderList, 8) == 8)
            name = "Z2 x D8 - Hall Senior Number 23 for Order 32";
        else if (numElementsOfOrder(orderList, 8) == 24)
            name = "Hall Senior Number 32 for Order 32";
        else if (numElementsOfOrder(orderList, 2) == 15 &&
determineInnerAutomorphism().trim().equalsIgnoreCase("Z2 x Z2 x Z2"))
            name = "Hall Senior Number 36 for Order 32";
        else if (numElementsOfOrder(orderList, 2) == 14)
            name = "Hall Senior Number 42 for Order 32";
        else if (numElementsOfOrder(orderList, 2) == 15 &&
numElementsOfOrder(orderList, 8) == 8)
            name = "Hall Senior Number 44 for Order 32";
        else if (numElementsOfOrder(orderList, 2) == 11 &&
numElementsOfOrder(orderList, 4) == 4)
            name = "Hall Senior Number 47 for Order 32";
        else if (numElementsOfOrder(orderList, 2) == 17)
            name = "D16 - Hall Senior Number 49 for Order 32";
        else if (numElementsOfOrder(orderList, 2) == 9)
            name = "Z16 xi Z2 - Hall Senior Number 50 for Order 32";
        else if (numElementsOfOrder(orderList, 4) == 18)
            name = "<2, 2, 8> - Hall Senior Number 51 for Order 32";

        else {
            identified = false;
            name = "ERROR Unknown group name - order 32";
        }
        break;
    default:
        name = "ERROR - order " +
Integer.toString(identifyMatrix.getOrder());
        identified = false;
    }
    return name;
}

/**
 * Method to determine the elements in the group that are the center
of the group. The
 * center of the group consists of all elements in the group that
commutes with all of the
 * other elements.
 *
 * @return ArrayList of Integers representing elements that make up
the center of the group
 */
private ArrayList<Integer> computeCenter()
{
    ArrayList<Integer> center_list = new ArrayList<Integer>();

    center_list.add(identifyMatrix.findIdentity());

    for (int row = 0; row < identifyMatrix.getOrder(); row++)
    {
        if (row == identifyMatrix.findIdentity()) continue;
        boolean commutes = true;

```

```

        int col = 1;
        while (commutes && col < identifyMatrix.getOrder())
        {
            if (identifyMatrix.getEntry(row, col) !=
identifyMatrix.getEntry(col, row))
                commutes = false;
                col++;
        }
        if (commutes)
        {
            center_list.add(new Integer(row));
            System.out.println("Center contains: " + row);
        }
    }
    return center_list;
}

/**
 * Method to determine if the center elements in the group correspond
to the Klein-4 group.
 *
 * @param center_list ArrayList of Integers representing elements that
make up the center of the group
 * @param order Array of integers containing order of all elements in
the group
 * @return Boolean result of check if center is Klein-4 group
 */
private boolean centerIsKlein4(ArrayList<Integer> center_list, int[]
order)
{
    // Klein-4 group is of order four so must be only four elements in
the center
    if (center_list.size() != 4) return false;
    Integer element;
    int count = 1;
    while (count <= 3)
    {
        element = (Integer) center_list.get(count);
        if (order[element.intValue()] != 2)
            return false;
        count = count + 1;
    }
    return true;
}

/**
 * Method to determine center elements in the group and their order.
 *
 * @param center_list ArrayList of Integers representing elements that
make up the center of the group
 * @param order Array of integers containing order of all elements in
the group
 */
private void showCenterOrders(ArrayList<Integer> center_list, int[]
order)

```

```

{
    //if (center_list.size() != 4) return false;
    Integer element;
    System.out.print("Center: ");
    for (int count = 0; count < center_list.size(); count++)
    {
        element = (Integer) center_list.get(count);
        System.out.print("\telmt " + element + ": " +
order[element.intValue()]);
    }
    System.out.println();
}

/**
 * Method to determine if all subgroups of the group are normal.
 *
 * @return Boolean result of check if all subgroups are normal
 */
private boolean subGroupsAllNormal()
{
    for (int i=0; i < identifyMatrix.getOrder(); i++)
    {
        ArrayList<Integer> sgList = generateSubGroupList(i);
        if (!normal(sgList)) return false;
    }
    return true;
}

/**
 * Method to determine the elements in a subgroup that include a
specific element.
 *
 * @param element integer of element contained in subgroup
 * @return ArrayList of Integers representing elements that make up
the center of the group
 */
public ArrayList<Integer> generateSubGroupList(int element)
{
    ArrayList<Integer> subGroupList = new ArrayList<Integer>();
    subGroupList.add(identifyMatrix.findIdentity());

    int temp = element;
    while (temp != identifyMatrix.findIdentity())
    {
        subGroupList.add(new Integer(temp));
        temp = identifyMatrix.getEntry(temp, element);
    }
    return subGroupList;
}

/**
 * Method to determine if the elements in a subgroup make up a normal
subgroup. A normal
 * subgroup is a group where the subgroup is commutative with all of
the elements in the
 * original group.

```

```

*
* @param subGroupList ArrayList of Integers representing elements
that make up the center of the group
* @return Boolean result of check if subgroup are normal
*/
private boolean normal(ArrayList<Integer> subGroupList)
{
    for (int i=0; i < identifyMatrix.getOrder(); i++)
    {
        ArrayList<Integer> tempSet = new ArrayList<Integer>();
        int inverse = identifyMatrix.findInverse(i);
        for (int j=0; j < subGroupList.size(); j++)
        {
            Integer tempCol = subGroupList.get(j);
            int tempRow = identifyMatrix.getEntry(i,
tempCol.intValue());
            tempSet.add(new Integer(identifyMatrix.getEntry(tempRow,
inverse)));
        }

        for (int k=0; k < subGroupList.size(); k++)
        {
            boolean inSet = false;
            for (int t=0; t<tempSet.size(); t++)
                if (tempSet.get(t).intValue() ==
subGroupList.get(k).intValue()) inSet = true;
            if (!inSet) return false;
        }
    }

    return true;
}

/**
* Method to determine the inner automorphism of the group being
identified.
*
* @return String Name of inner automorphism
*/
private String determineInnerAutomorphism()
{
    groupCreator localCreator = new groupCreator();
    localCreator.resetGroup(identifyMatrix);
    if (localCreator.createInnerAutGroup())
    {
        groupIdentify innerAutName = new groupIdentify();
        innerAutName.resetGroupIdentify(localCreator.getGroup());
        if (innerAutName.isIdentified())
            return innerAutName.getName();
    }
    return null;
}

/**
* Class used to store element orders and the number of elements of
that order

```

```

*/
private class num_order
{
    /**
     * integer order of elements to be stored
     */
    private int orderOfElement;
    /**
     * integer number of elements that have the associated order
     */
    private int numOfElements;

    /**
     * Constructor for the class that saves an order value and number of
values of that order
     * @param order integer order to be stored
     * @param num integer number of elements of order to be stored
     */
    public num_order(int order, int num)
    {
        orderOfElement = order;
        numOfElements = num;
    }

    /**
     * Returns the order
     * @return integer value of the order
     */
    public int getOrderOfElement()
    {
        return orderOfElement;
    }

    /**
     * Returns the number of elements
     * @return integer value of the number of elements
     */
    public int getNumOfElements()
    {
        return numOfElements;
    }
}

/**
 * Class to store a prime and number used of a prime factorization
 */
private class factorType
{
    /**
     * Prime from the prime factorization
     */
    private int prime;
    /**
     * Number of primes used in the prime factorization
     */

```

```

private int numOfPrimes;

/**
 * Constructor for the class that saves a prime and number of values
of that prime used in the prime factorization
 * @param p integer value of the prime
 * @param num integer value of the number of primes in the
factorization
 */
public factorType(int p, int num)
{
    prime = p;
    numOfPrimes = num;
}

/**
 * Returns the prime
 * @return integer value of the prime
 */
public int getPrime()
{
    return prime;
}

/**
 * Returns the number of times the prime is used in the prime
factorization
 * @return integer value of the number of times the prime is used
 */
public int getNumOfPrimes()
{
    return numOfPrimes;
}

/**
 * Increments the value of <CODE>numOfPrimes</CODE> by 1
 */
public void incrementNumOfPrimes()
{
    numOfPrimes = numOfPrimes + 1;
}
}
}

```

GROUPPERMUTATION.JAVA

```

/*
 * groupPermutation.java
 *
 * Created on February 6, 2005, 10:29 PM
 */

```

```

package cayleytable;

import java.util.*;
/**
 * Class used to determine the number of permutations of a particular
order
 * that form a legal group where the identity element is always 0.
 *
 * @author Jeffrey Barr
 */
public class groupPermutation {

    int groupCount;
    private cayleytable.groupIdentify groupNamer;
    private cayleytable.groupMatrix myGroup;

    /** Creates a new instance of groupPermutation */
    public groupPermutation() {

        System.out.println("Determine all of the permutations that are
groups for order n");
        System.out.println("There are a total of  $n!$  permutations");
        int n = 7;
        myGroup = new cayleytable.groupMatrix(n);
        groupNamer = new cayleytable.groupIdentify();
        groupCount = 0;

        ArrayList<PermutationGenerator> myPermGens = new
ArrayList<PermutationGenerator>();
        for (int i=0; i<n; i++)
        {
            PermutationGenerator myPermGen = new PermutationGenerator(n);
            myPermGens.add(myPermGen);
        }

        long total = (long) java.lang.Math.pow(factorial(n-1), n-1);
        System.out.println("Total = " + total + " possibilities");
        determinePermutationAndSolve(myPermGens, n);

        System.out.println("Group Count = " + groupCount + " out of " +
total + " possibilities");
    }

    private void
determinePermutationAndSolve(ArrayList<PermutationGenerator> myPermGens,
int n)
    {

        if (n==myGroup.getOrder())
        {
            for (int i=0; i<myGroup.getOrder(); i++)
                myGroup.setEntry(0, i, i);
            determinePermutationAndSolve(myPermGens, n-1);
            return;
        }
    }
}

```

```

PermutationGenerator myPG = myPermGens.get(n-1);
while (myPG.hasMore())
{
    int[] indices = myPG.getNext();
    if (indices[0] < myGroup.getOrder()-n)
        continue;
    if (indices[0] > myGroup.getOrder()-n)
        break;

    for (int i=0; i<indices.length; i++)
        myGroup.setEntry(myGroup.getOrder()-n, i, indices[i]);
    if (repetitionCheck(myGroup.getOrder()-n)) continue;
    if (n-1 > 0) determinePermutationAndSolve(myPermGens, n-1);
    else
    {
        if (myGroup.checkForIdentity() &&
            myGroup.checkForInverse() &&
            myGroup.checkIfAssociative())
        {
            groupCount++;
            groupNamer.resetGroupIdentify(myGroup);
            System.out.println("Found Group " + groupCount + ": "
+ groupNamer.getName());
            for (int row=0; row<myGroup.getOrder(); row++)
            {
                for (int col=0; col<myGroup.getOrder(); col++)
                {
                    System.out.print("\t" + myGroup.getEntry(row,
col));
                }
                System.out.println();
            }
        }
    }
    myPG.reset();
}

private boolean repetitionCheck(int currentRow)
{
    int totalRows = myGroup.getOrder();
    if (currentRow == 0) return false;
    //if (myGroup.getEntry(currentRow, 0) != currentRow) return true;
    for (int col=0; col<totalRows; col++)
    {
        for(int row=0; row<currentRow-1; row++)
            if (myGroup.getEntry(row, col) ==
myGroup.getEntry(currentRow,col)) return true;
    }
    return false;
}

private int factorial(int n)
{

```

```

        if (n <= 0) return 1;
        else return n * factorial(n-1);
    }

    public static void main(String[] args) {
        // TODO code application logic here
        groupPermutation gp = new groupPermutation();
    }
}

```

PERMUTATIONGENERATOR.JAVA

```

/*
 * PermutationGenerator.java
 *
 * Source Code taken from http://www.merriampark.com/perm.htm
 * Created on February 4, 2005, 4:15 PM
 */

package cayleytable;
import java.math.BigInteger;
/**
 * Class used to create permutations of a set of numbers from 0 to n-1
 * @author {@link http://www.merriampark.com/perm.htm}
 */
public class PermutationGenerator {

    /**
     * array of integers representing current permutation that is being sent
     back to requestor
     */
    private int[] a;
    /**
     * BigInteger of the number of permutations that have not been retrieved
     */
    private BigInteger numLeft;
    /**
     * BigInteger of the total number of permutations
     */
    private BigInteger total;

    /**
     * Constructor to create the initial permutation of the list of integers
     from
     * 0 to a given number.
     * WARNING: Don't make n too large.
     * Recall that the number of permutations is n!
     * which can be very large, even when n is as small as 20 --
     * 20! = 2,432,902,008,176,640,000 and
     * 21! is too big to fit into a Java long, which is
     * why we use BigInteger instead.
     *
     * @param n integer size of the array to permute
     */
}

```

```

*/
public PermutationGenerator (int n) {
    if (n < 1) {
        throw new IllegalArgumentException ("Min 1");
    }
    a = new int[n];
    total = getFactorial (n);
    reset ();
}

/**
 * Method to reset the permutation list to the original array
 *
 */
public void reset () {
    for (int i = 0; i < a.length; i++) {
        a[i] = i;
    }
    numLeft = new BigInteger (total.toString ());
}

/**
 * Method to return number of permutations not yet generated
 *
 * @return BigInteger value of the number of permutations left
 */
public BigInteger getNumLeft () {
    return numLeft;
}

/**
 * Method to return total number of permutations
 *
 * @return BigInteger value of the total number of permutations
 */
public BigInteger getTotal () {
    return total;
}

/**
 * Method to return whether any more permutations are left to use
 * @return Boolean value determining if <CODE>numLeft</CODE> is greater
than zero
 */
public boolean hasMore () {
    return numLeft.compareTo (BigInteger.ZERO) == 1;
}

/**
 * Method to compute the factorial of a given integer
 *
 * @param n integer to which compute the factorial
 * @return BigInteger factorial of the given value
 */
private static BigInteger getFactorial (int n) {

```

```

BigInteger fact = BigInteger.ONE;
for (int i = n; i > 1; i--) {
    fact = fact.multiply (new BigInteger (Integer.toString (i)));
}
return fact;
}

/**
 * Method to generate next permutation (algorithm from Rosen p. 284)
 *
 * @return integer array containing next permutation of list of integers
from 0 to n-1
 */

public int[] getNext () {

    if (numLeft.equals (total)) {
        numLeft = numLeft.subtract (BigInteger.ONE);
        return a;
    }

    int temp;

    // Find largest index j with a[j] < a[j+1]

    int j = a.length - 2;
    while (a[j] > a[j+1]) {
        j--;
    }

    // Find index k such that a[k] is smallest integer
// greater than a[j] to the right of a[j]

    int k = a.length - 1;
    while (a[j] > a[k]) {
        k--;
    }

    // Interchange a[j] and a[k]

    temp = a[k];
    a[k] = a[j];
    a[j] = temp;

    // Put tail end of permutation after jth position in increasing order

    int r = a.length - 1;
    int s = j + 1;

    while (r > s) {
        temp = a[s];
        a[s] = a[r];
        a[r] = temp;
        r--;
        s++;
    }
}

```

```
    numLeft = numLeft.subtract (BigInteger.ONE);  
    return a;  
  }  
}
```

ABSTRACT OF THE THESIS

Computational Tools for Group Theory
by
Jeffrey H. Barr
Master of Science in Computer Science
San Diego State University, 2005

This thesis describes the refinement and extension of code that was originally developed as part of a 1987 Math 797 project by David Gibbs, "Computer Generation and Identification of Groups of Order 2 to 31." The purpose of the code was to generate, identify, and analyze groups presented in the form of a Cayley Table. Gibbs' code was transferred from Pascal to Java. Objects were created to improve the code design and allow for better interaction between the generation, identification, analysis, and visualization sections of code.

The code for this thesis allows cyclic groups to easily be generated, along with groups created via defined relationships and the cross product of multiple groups. A user interface was added to the system to assist the user when utilizing the code as well as visualizing the groups that are generated. Functionality to allow a user to manually enter a Cayley Table for analysis was also added to the system.

The generation code is no longer limited to groups of order less than 31. Improvements were made to the identification code so that the system can identify all Abelian groups including those created via the cross product of groups. Additionally, many non-Abelian groups of order 32 were added to the list of groups which could be identified.

Analysis functionality was added including the identification of whether a table actually represents a group as well as if the group is Abelian. Also, functionality was added to calculate the inner automorphism group of the group being displayed. The analysis functionality will provide users the ability to analyze groups that the code cannot yet identify.