DESIGNING JAVA COM FOR IMPLEMENTING MATRIX CLASS AND MATRIX-BASED OPERATIONS

ABUBAKAR MUHAMMAD 1, ZAINAB ALIYU MUSA 2

Computer Science Department, Federal Polytechnic, Bauchi (Nigeria)

ABSTRACT
COM, an acronym for Component Object Model is an object-oriented technology jointly developed by Microsoft and Digital Equipment Corporation that allows developers create distributed and reusable objects. An object model is a programming architecture in which developers define not only the type of data structure, but also the types of operations that can be applied on the data structure. This research looks into the common matrix operations which include multiplication, division, addition, subtraction, finding inverse, transpose and determinant of matrix. These operations are inbuilt into a component object model that can be reused in different environments to carry out these operations on any arbitrary given matrices. Main focus of the component object model is to provide reusable object for Java developers and programmers that they can use to carry out the defined matrix operation without the need of writing lengthy codes to do that. The component object model is designed using Java Netbeans and Matlab. Software reuse improves programmers’ efficiency, quality of software and reduces the time taken for software development project to complete. It is the main objective of the component object model that this research intends to design, to achieve improvement of programmers’ efficiency, quality of software and reduction of project completion time in Java programming involving matrix operations. Finally, this research attempts to reuse the component object model in Java to perform common matrix operations in order to justify the reusability of the component object model in Java programming.

Keywords: COM, matrix, determinant, transpose, inverse of Matrix

INTRODUCTION
Software reuse has long been one of the major issues in the world of software engineering. The reason is obvious. Software reuse can dramatically increase the productivity of software community, ease maintenance, and improve product reliability [6]. Although most people would agree upon the importance of reuse, it is only today that it has become a main goal in software engineering. Software reuse was first realized in the late 50’s or early 50’s, when the concept of libraries was developed which allowed collections of pre-compiled, reusable subroutines to be linked into a program for performing specialized tasks. The area in which libraries have succeeded best is probably numerical analysis, where a large number of FORTRAN libraries are available on various platforms and are used in many engineering projects. But there are not many successful stories in other areas. The difficulty of encapsulating high-level functionality in subroutines is responsible for the failure of library-based reuse [1].

The current trend in software engineering is towards component-based development. In building software with components, it promises more efficient and effective software reuse and higher productivity. A system can be designed and implemented by assembling components, customizing or extending them as needed; and publishing components in a form that can be applied to design and construct others, based purely on interfacing [6].

Background of the Research
The essence of COM is a language-neutral way of implementing objects that can be used in environments different from the one in which they were created, even across machine boundaries [1]. For well-authored components, COM allows reuse of objects with no knowledge of their internal implementation, as it forces component implementers to provide well-defined interfaces that are separate from the implementation. The different allocation semantics of languages are accommodated by making objects responsible for their own creation and destruction through reference-counting. Casting between different
interfaces of an object is achieved through the query interface () function. The preferred method of inheritance within COM is the creation of sub-objects to which method calls are delegated [3].

**History of Component Object Model**

Object Linking Embedding (OLE), Microsoft’s first object-based framework, was built on top of DDE and designed specifically for compound documents. It was introduced with word for Windows and Excel in 1991, and was later included with Windows, starting with version 3.1 in 1992 [10]. An example of compound document is a spreadsheet embedded in a word for windows document: as changes are made to the spreadsheet within excel, they appear automatically inside the word document. In 1991, Microsoft introduced Visual Basic Extensions (VBX) with Visual Basic 1.0 A VBX is a packaged extension in the form of a dynamic-link library (DDL) that allowed objects to be graphically placed in a form and manipulated by properties and methods. These were later adapted for use by other languages such as Visual C++ [10].

In 1992, when version 3.1 of windows was released, Microsoft released OLE 2 with its underlying object model. The COM Application binary interface (ABI) was the same as the MAPI ABI, which was released in 1992. While OLE 1 was focused on compound documents, COM and OLE 2 were designed to address software components in general. Text conversations and windows messages had proved not to be flexible enough to allow sharing application features in a robust and extensible way, so COM was created as a new foundation, and OLE changed to OLE 2 [4].

In 1994, OLE Custom Control (OCXs) was introduced as the successor to VBX controls. At the same time, Microsoft stated that OLE 2 would just be known as “OLE”, and that OLE was no longer an acronym, but a name for all the company’s component technologies. In early 1996, Microsoft found a new use for OLE Custom Controls, expanding their web browser’s capability to present content, renamed some part of OLE relating to the internet ActiveX, and gradually renamed all OLE technologies to active, except the compound document technology that was used in Microsoft Office [3].

**Interfaces**

All COM components must (at the very least) implement the standard IUnknown interface, and thus all COM interfaces are derived from IUnknown. The IUnknown interface consists of three methods; Addref() and Release(), which implement reference counting and controls the lifetime of interfaces; and QueryInterface(), which by specifying an IID allows a caller to retrieve references to the different interfaces the component implements. The effect of QueryInterface () is similar to dynamic_cast <> in C++ or casts in Java and C# (pronounced c sharp) [3]. COM component’s interfaces are required to exhibit the reflexive, symmetric, and transitive properties. The reflex property refers to the ability for the queryinterface () call on a given interface with the interface’s ID to return the same instance of the interface. The symmetric property requires that when interface B is retrieved from interface A via queryinterface (), interface A is retrieve from interface B as well. The Transitive property requires that if interface B is obtainable from interface A and interface C is obtainable from interface B, then interface C should be retrievable from interface A [3].

An interface consists of a pointer to a virtual function table that contains a list of pointers to the functions that implement the functions declared in the interface, in the same order that they are declared in the interface. This technique of passing structure of function pointers is very similar to the one used by OLE 1.0 to communicate with its system libraries.

COM specifies many other standard interfaces used to allow inter-component communication. For example, one such interface is IStream, which is exposed by components that have data stream semantics (e.g. a FileStream component used to read or write files). It has the expected Read and Write methods to perform stream reads and writes. Another standard interface is IOleObject, which is exposed by components that expect to be linked or embedded into a container. IOleObject contains methods that
allow callers to determine the size of the component’s bounding rectangle, whether the component supports operations like ‘Open’, ‘Save’ and so on.

**Class**
A class is COM’s language-independent way of defining a class in the object-oriented sense. A class can be a group of similar objects or a class is simply a representation of a type of object; it should be thought as a blueprint that describes the object.

A co-class supplies concrete implementation(s) of one or more interfaces. In COM, such concrete implementations can be written in any programming language that supports COM component development, e.g. Delphi, C++, Visual Basic, etc.

One of COM’s major contributions to the world of Windows development is the awareness of the concept of separation of interface from implementations. This means that at runtime, an application can choose to instantiate an interface from one of many different concrete implementations.

**Framework Object**
The fundamental principles of COM have their root in Object-Oriented philosophies. It is a platform for the realization of Object-Oriented Development and Deployment.

Because COM is a runtime framework, types have to be individually identifiable and specifiable at runtime. To achieve this, globally unique identifiers (GUIDs) are used. Each COM type is designated its own GUID for identification at runtime (versus compile time).

In order for information on COM types to be accessible at both compile time and runtime, COM uses type libraries. It is through the effective use of type libraries that COM achieves its capabilities as a dynamic framework for the interaction of objects.

**Object Oriented Programming**
Object-oriented programming is one of the current approaches to solve problems in the software industry. The object model helps us to realize principle like encapsulation, abstraction, modularity and hierarchical decomposition. The object-oriented paradigm can be described as the nineteen eighties successor of structured design and structured programming of the seventies. Object-oriented technology has several advantages. Firstly, the object model leads to reuse possibility of implementation and design. This property leads to reusable application frameworks. Frameworks provide not only reusable code, but more importantly, they carry a lot of information about design. Secondly, object reflects a natural concept in a human point of view. However, object-oriented frameworks often become large, complex class structures. This results is hard to learn and hard to understand class hierarchies and often makes it impossible to specialize a framework by less experienced framework users. The specialization of an object-oriented class hierarchy is often realized using class inheritance. Inheritance implies a direct implementation dependency between super- and subclasses in an “is a” relationship manner. If super- and subclasses are maintained by different group developers, this fact can lead to serious integration problems. An application framework can cover a well defined problem domain.

The combination of two or more different frameworks seems to be very hard, because the different designs often disturb each other.

**Review of Programming Concepts**
Computer programming or coding is the process of designing, writing, testing, debugging or troubleshooting and maintaining language. The code may be a modification of an existing source or something completely new. The purpose of programming is to create a program that exhibits a certain desire behavior (customization). The process of writing source code often requires expertise in many different subjects, including knowledge of the application domain, specialized algorithm and normal logic. Within software engineering, programming (the implementation) is regard as one phase in a software development process. A computer scientist, write the details to vary in different languages, but a few basic instructions appear in just about every language:

- Input: get data from the keyboard, a file, or some other device
- Output: Display data on the screen or send data to file or other device
- Arithmetic: Perform basic arithmetic operation, like addition and multiplication
• Condition execution: check for certain condition and execute the appropriate sequence of statement.
• Reputation: Perform some action repeatedly, usually with some variation.

Many computers provide a mechanism to call functions provided by libraries. Provided the functions in a function in a library follow the appropriate run time convention (e.g. method of passing arguments) then these functions may be written in any other language.

Matrix Operation

• **Multiplication of Matrices**

Matrices multiplication is achieved by conducting row by column operations of the two matrices. In our normal mathematics the two operands are simply multiply, but a matrix is having individual elements that are involved in the multiplication. For example

\[
\begin{bmatrix}
  a_{11} & a_{12} \\
  a_{21} & a_{22}
\end{bmatrix}
\]

are elements of matrix A and
\[
\begin{bmatrix}
  b_{11} & b_{12} \\
  b_{21} & b_{22}
\end{bmatrix}
\]

are elements of matrix B

\[
A \times B = a_{11} \times b_{11}, a_{12} \times b_{21},
\]

\[
a_{21} \times b_{12}, a_{22} \times b_{22}
\]

\[
\square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \square \s
PROBLEM IN FOCUS
The dominant nature of matrix operations in Computing and Mathematics prompted the design of many applications that needs matrix operations. Each of these applications has series of codes written to achieve this purpose making the design process of the application to be lengthy and sometimes with bugs in such codes. The matrix operations could have been designed and tested once, after success is recorded then programmers need not design such codes in their application again; they simply reuse the existing codes. The shareability of the matrix operation codes, removes redundancy by programmers to produce the same codes now and then. The ability to use the matrix operation codes in other environments other than the one they are designed is another target advantage.

OBJECTIVES
The objectives behind this research are:
• To create Java Beans(COM) covering basic operations on matrix, both unary and binary operations
• To implement and test the Java Beans so as to ensure its accuracy and reliability.
• To make the Java Beans(COM) reusable class in other applications

SIGNIFICANCE OF THE DESIGNED COM
The designed COM can be reused in Matlab and Microsoft products like the Visual Studio to implement Matrix operations, namely multiplication, addition, subtraction, inverse, transpose and determinant. It can also be reused in Java Development Kit environment as a matrix class for the same purpose. This improves Microsoft Programmers and Java programmers more productive and significantly reduces the effort needed in designing application.

IMPLEMENTATION
The COM is implemented using Java Netbeans and matrices multiplication, addition and subtraction are implemented in classes as binary operations. While transpose, inverse and determinant are designed in classes as well as unary operations. The complete source code of the program is shown in appendix A. Two matrices namely A and B, are used for binary operations. Both matrices can be used for unary operations.

Programming Language
Java is used to designed the Matrix COM, and the programming environment is Java Netbeans

Matrix Size
The pre-determined condition for all binary operations is that the two matrices must be square and of equal size. The unary operations require the matrix in question to be square. The matrix COM allows user to assign any arbitrary size to any of the two matrices.

Spaces (Matrices)
Matrix spaces are generated base on the matrix size, and each space is capable of taking any valid integer.

Process Design
The matrix COM can be operated base on the pseudo code shown below:
1. Start,
2. Generate size of the two matrices,
3. Generate spaces for the matrices,
4. Populate the matrices with element either automatically or manually,
5. Select the require operations by clicking its button,
6. The Resultant matrix displays result of such operations.
7. Exit.

Target Environment
The matrix COM can be reused in Matlab, Microsoft IDE like visual studio and Java Development Kit IDE.

Sample Run
After running the Matrix COM the following sample runs are obtained:

---

**Fig. 1 Addition of Matrices**

**Fig. 2 Multiplication of Matrices**
CONCLUSION
A matrix COM is successfully designed and implemented using Java Netbeans IDE. After testing it is found to produce accurate result and it has the reuse capability in Matlab, Microsoft IDE and Java Development Kit (JDK).

RECOMMENDATION
We hereby recommended the reuse of the designed Matrix COM in programming involving matrices operations

ACKNOWLEDGEMENTS
All duly consulted materials are hereby acknowledged.

REFERENCES

Appendix A
Code Listing
package matrixcalculator;
import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.text.NumberFormat;
import java.util.ArrayList;
import java.util.StringTokenizer;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
public class MatrixCalculator {
    private boolean DEBUG = true;
    private boolean INFO = true;
    private static int max = 100;
    private static int decimals = 3;
    private JLabel statusBar;
    private JTextArea taA, taB, taC;
    private int iDF = 0;
    private int n = 4;
    private static NumberFormat nf;
    public Component createComponents() {
        /* == MATRICES == */
        taA = new JTextArea();
        taB = new JTextArea();
        taC = new JTextArea();
        JPanel paneMs = new JPanel();
        paneMs.setLayout(new BoxLayout(paneMs, BoxLayout.X_AXIS));
        paneMs.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));
        paneMs.add(MatrixPane("Matrix A", taA));
        paneMs.add(Box.createRigidArea(new Dimension(10, 0)));
        paneMs.add(MatrixPane("Matrix B", taB));
        paneMs.add(Box.createRigidArea(new Dimension(10, 0)));
        paneMs.add(MatrixPane("Matrix C", taC));
        /* == OPERATION BUTTONS == */
        JPanel paneBtn = new JPanel();
        paneBtn.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
        paneBtn.setLayout(new GridLayout(3, 3));
        paneBtn.add(new JButton("Add"));
        paneBtn.add(new JButton("Subtract"));
        paneBtn.add(new JButton("Multiply"));
        paneBtn.add(new JButton("Divide"));
        return paneMs;
    }
}
JButton btnApB = new JButton("A + B = C");
JButton btnAmB = new JButton("A * B = C");
JButton btnBmA = new JButton("B * A = C");
JButton btnBmA = new JButton("A - B = C");
JButton btnAdjA = new JButton("adjoint(A) = C");
JButton btnInvA = new JButton("inverse(A) = C");
JButton btnInvB = new JButton("inverse(B) = C");
JButton btnTrnsA = new JButton("transpose(A) = C");
JButton btnDetA = new JButton("determ(A) = C");
JButton btnDetB = new JButton("determ(B) = C");
paneBtn.add(btnApB);
paneBtn.add(btnAmB);
paneBtn.add(btnBmA);
paneBtn.add(btnAdjA);
paneBtn.add(btnInvA);
paneBtn.add(btnInvB);
paneBtn.add(btnTrnsA);
paneBtn.add(btnDetA);
paneBtn.add(btnDetB);
/* == ADD BUTTON Listeners == */
btnApB.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent evt) {
    try {
      DisplayMatrix(AddMatrix(ReadInMatrix(taA),
                            ReadInMatrix(taB)), taC);
    } catch (Exception e) {
      System.err.println("Error: " + e);
    }
  }
});

btnAmB.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent evt) {
    try {
      DisplayMatrix(MultiplyMatrix(
                        ReadInMatrixNotSquare(taA),
                        ReadInMatrixNotSquare(taB)), taC);
    } catch (Exception e) {
      System.err.println("Error: " + e);
    }
  }
});

btnBmA.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent evt) {
    try {
      DisplayMatrix(SubtractMatrix(ReadInMatrixNotSquare(taB),
                                  ReadInMatrixNotSquare(taA)), taC);
    } catch (Exception e) {
      System.err.println("Error: " + e);
    }
  }
});
btnInvA.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        try {
            DisplayMatrix(Inverse(ReadInMatrix(taA)), taC);
        } catch (Exception e) {
            System.err.println("Error: " + e);
        }
    }
});

btnInvB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        try {
            DisplayMatrix(Inverse(ReadInMatrix(taB)), taC);
        } catch (Exception e) {
            System.err.println("Error: " + e);
        }
    }
});

btnAdjA.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        try {
            DisplayMatrix(Adjoint(ReadInMatrix(taA)), taC);
        } catch (Exception e) {
            System.err.println("Error: " + e);
        }
    }
});

btnTrnsA.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        try {
            DisplayMatrix(Transpose(ReadInMatrixNotSquare(taA)), taC);
        } catch (Exception e) {
            System.err.println("Error: " + e);
        }
    }
});

btnDetA.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
        try {
            taC.setText("Determinant A: 
            + nt.format(Determinant(ReadInMatrix(taA))));
        } catch (Exception e) {
            System.err.println("Error: " + e);
        }
    }
});

btnDetB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {
public void actionPerformed(ActionEvent evt) {
        try {
            DisplayMatrix(Transposed(ReadInMatrixNotSquare(taA)), taC);
        } catch (Exception e) {
            System.err.println("Error: " + e);
        }
    }
});

public void actionPerformed(ActionEvent evt) {
        try {
            DisplayMatrix(Transposed(ReadInMatrixNotSquare(taA)), taC);
        } catch (Exception e) {
            System.err.println("Error: " + e);
        }
    }
});

btnDetB.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent evt) {

try {
    taC.setText("Determinant B: "+ nf.format(Determinant(ReadInMatrix(taB))));
} catch (Exception e) {
    System.err.println("Error: " + e);
}

}:

/* == MAIN PANEL == */
JPanel pane = new JPanel();
pane.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
pane.setLayout(new BoxLayout(pane, BoxLayout.Y_AXIS));
pane.add(paneMs);
pane.add(paneBtn);
JPanel fpane = new JPanel();
fpane.setLayout(new BorderLayout());
fpane.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
fpane.add("Center", pane);
statusBar = new JLabel("Ready");
fpane.add("South", statusBar);
return fpane;

} /* == Setup Invidual Matrix Panes == */
private JPanel MatrixPane(String str, JTextArea ta) {
    JScrollPane scrollPane = new JScrollPane(ta);
    int size = 200;
    scrollPane.setPreferredSize(new Dimension(size, size));
    JLabel label = new JLabel(str);
    label.setLabelFor(scrollPane);
    JPanel pane = new JPanel();
    pane.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
    pane.setLayout(new BoxLayout(pane, BoxLayout.Y_AXIS));
    pane.add(label);
    pane.add(scrollPane);
    return pane;
}

public static void main(String[] args) {
    JFrame frame = new JFrame("Matrix Calculator");
    frame.setSize(new Dimension(725, 200));
    MatrixCalculator app = new MatrixCalculator();
    Component contents = app.createComponent();
    frame.getContentPane().add(contents, BorderLayout.CENTER);
    frame.addWindowListener(new WindowAdapter() {
        public void windowClosing(WindowEvent e) {
            System.exit(0);
        }
    });
    frame.pack();
    frame.setVisible(true);
    nf = NumberFormat.getInstance();
    nf.setMinimumFractionDigits(1);
```java
public float[][] ReadInMatrix(JTextArea ta) throws Exception {
    if (DEBUG) {
        System.out.println("Reading In Matrix");
    }
    /* == Parse Text Area == */
    String rawtext = ta.getText();
    String val = "", 
    int i = 0;
    int j = 0;
    int[] rsize = new int[max];
    /* == Determine Matrix Size/Valid == */
    StringTokenizer ts = new StringTokenizer(rawtext, "\n");
    while (ts.hasMoreTokens()) {
        StringTokenizer ts2 = new StringTokenizer(ts.nextToken());
        while (ts2.hasMoreTokens()) {
            ts2.nextToken();
            j++;
        }
        rsize[i] = j;
        i++;
        j = 0;
    }
    statusBar.setText("Matrix Size: " + i);
    if ((DEBUG) || (INFO)) {
        System.out.println("Matrix Size: " + i);
    }
    for (int c = 0; c < i; c++) {
        if (DEBUG) {
            System.out.println("i=" + i + "  j=" + rsize[c] + "  Column: "+ c);
        }
        if (rsize[c] != i) {
            statusBar.setText("Invalid Matrix Entered. Size Mismatch.");
            throw new Exception("Invalid Matrix Entered. Size Mismatch.");
        }
    }
    /* == Set matrix size == */
    n = i;
    float matrix[][] = new float[n][n];
    i = j = 0;
    val = "";
    /* == Do the actual parsing of the text now == */
    StringTokenizer st = new StringTokenizer(rawtext, "\n");
    while (st.hasMoreTokens()) {
        StringTokenizer st2 = new StringTokenizer(st.nextToken());
        /* == parse text == */
    }
    } // end of method

```
while (st2.hasMoreTokens()) {
    val = st2.nextToken();
    try {
        matrix[i][j] = Float.valueOf(val).floatValue();
    } catch (Exception exception) {
        statusBar.setText("Invalid Number Format");
    }
    j++;
}

   i++;  
j = 0;
}

if (DEBUG) {
    System.out.println("Matrix Read::");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            System.out.print("m[" + i + "]" + j + "] = "+ matrix[i][j] + " ");
        }
        System.out.println();
    }
    return matrix;
}

public float[][] ReadInMatrixNotSquare(JTextArea ta)
        throws Exception {
    if (DEBUG) {
        System.out.println("Reading In Matrix");
    }
    /* == Parse Text Area == */
    String rawtext = ta.getText();
    /* == Determne Matrix Size/Valid == */
    StringTokenizer ts = new StringTokenizer(rawtext, ",n");
    if (DEBUG)
        System.out.println("Rows: "+ ts.countTokens());
    float matrix[][] = new float[ts.countTokens()][];
    StringTokenizer st2;
    int row = 0;
    int col = 0;
    //making sure rows are same length
    int last = -5;
    int curr = -5;
    while (ts.hasMoreTokens()) {
        st2 = new StringTokenizer(ts.nextToken(), ", ");
        last = curr;
        curr = st2.countTokens();
        if (last != -5 && curr != last)
            throw new Exception("Rows not of equal length");
        if (DEBUG)
            System.out.println("Cols: "+ st2.countTokens());
        matrix[row] = new float[st2.countTokens()];
    }
while (st2.hasMoreElements()) {
    matrix[row][col++] = Float.parseFloat(st2.nextToken());
}
row++;
col = 0;
}
System.out.println();
return matrix;

// Display Matrix in TextArea
// --------------------------------------------------------------

public void DisplayMatrix(float[][] matrix, JTextArea ta) {
    /* == TODO: Better Formatting of Resultant Matrix == */
    if (DEBUG) {
        System.out.println("Displaying Matrix");
    }
    String rstr = "";
    String dv = "";
    for (int i = 0; i < matrix.length; i++) {
        for (int j = 0; j < matrix[i].length; j++) {
            dv = nf.format(matrix[i][j]);
            rstr = rstr.concat(dv + "  ");
        }
        rstr = rstr.concat("n");
    }
    ta.setText(rstr);
}
public float[][] AddMatrix(float[][] a, float[][] b) throws Exception {
    int tms = a.length;
    int tmsB = b.length;
    if (tms != tmsB) {
        statusBar.setText("Matrix Size Mismatch");
    }
    float matrix[][] = new float[tms][tms];
    for (int i = 0; i < tms; i++) {
        for (int j = 0; j < tms; j++) {
            matrix[i][j] = a[i][j] + b[i][j];
        }
    }
    return matrix;
}
public float[][] SubtractMatrix(float[][] a, float[][] b) throws Exception {
    int tms = a.length;
    int tmsB = b.length;
    if (tms != tmsB) {
        statusBar.setText("Matrix Size Mismatch");
    }
    float matrix[][] = new float[tms][tms];
    for (int i = 0; i < tms; i++) {
        for (int j = 0; j < tms; j++) {
            matrix[i][j] = a[i][j] - b[i][j];
        }
    }
    return matrix;
}
public float[][] MultiplyMatrix(float[][] a, float[][] b) throws Exception {
    if(a[0].length != b.length)
        throw new Exception("Matrices incompatible for multiplication");
    float matrix[][] = new float[a.length][b[0].length];
    for (int i = 0; i < a.length; i++)
        for (int j = 0; j < b[i].length; j++)
            matrix[i][j] = 0;
    //cycle through resultant matrix
    for(int i = 0; i < matrix.length; i++)
        for(int j = 0; j < matrix[i].length; j++){
            matrix[i][j] = calculateRowColumnProduct(a,i,b,j);
        }
    return matrix;
}
public float calculateRowColumnProduct(float[][] A, int row, float[][] B, int col){
    float product = 0;
    for(int i = 0; i < A[row].length; i++)
        product +=A[row][i]*B[i][col];
    return product;
}
public float[][] Transpose(float[][] a) {
    if (INFO) {
        System.out.println("Performing Transpose...");
    }
    float m[][] = new float[a[0].length][a.length];
    for (int i = 0; i < a.length; i++)
        for (int j = 0; j < a[i].length; j++)
            m[j][i] = a[i][j];
    return m;
}
public float[][] Inverse(float[][] a) throws Exception {
    // Formula used to Calculate Inverse:
    // inv(A) = 1/det(A) * adj(A)
    if (INFO) {
        System.out.println("Performing Inverse...");
    }
    int tms = a.length;
    float mm[][] = Adjoint(a);
    float det = Determinant(a);
    float dd = 0;
    if (det == 0) {
        statusBar.setText("Determinant Equals 0, Not Invertible.");
    if (INFO) {
System.out.println("Determinant Equals 0, Not Invertible.");
}
}
else {
    dd = 1 / det;
}
for (int i = 0; i < tms; i++)
    for (int j = 0; j < tms; j++) {
        m[i][j] = dd * mm[i][j];
    }
return m;

// ------------------------------------------------------------------------
public float[][] Adjoint(float[][] a) throws Exception {
    if (INFO) {
        System.out.println("Performing Adjoint...");
    }
    int tms = a.length;
    float m[][] = new float[tms][tms];
    int ii, jj, ia, ja;
    float det;
    for (int i = 0; i < tms; i++)
        for (int j = 0; j < tms; j++) {
            ia = ja = 0;
            float ap[][] = new float[tms - 1][tms - 1];
            for (ii = 0; ii < tms; ii++) {
                for (jj = 0; jj < tms; jj++) {
                    if ((ii != i) && (jj != j)) {
                        ap[ia][ja] = a[ii][jj];
                        ja++;
                    }
                }
                if ((ii != i) && (jj != j)) {
                    ia++;
                }
                ja = 0;
            }
            det = Determinant(ap);
            m[i][j] = (float) Math.pow(-1, i + j) * det;
        }
    m = Transpose(m);
    return m;
}
// ------------------------------------------------------------------------
public float[][] UpperTriangle(float[][] m) {
    if (INFO) {
        System.out.println("Converting to Upper Triangle...");
    }
    float f1 = 0;
    float temp = 0;
    int tms = m.length; // get This Matrix Size (could be smaller than
    // global)
public float Determinant(float[][] matrix) {
   if (INFO) {
      System.out.println("Getting Determinant...");
   }
   int tms = matrix.length;
   float det = 1;
   matrix = UpperTriangle(matrix);
   for (int i = 0; i < tms; i++) {
      det = det * matrix[i][i];
   } // multiply down diagonal
   return det;
} // ---------------------------------------------------------------------

public float[] UpperTriangle(float[][] matrix) {
   if (INFO) {
      System.out.println("Getting UpperTriangle...");
   }
   float[] result = new float[matrix.length];
   for (int i = 0; i < matrix.length; i++) {
      result[i] = 1;
   }
   for (int col = 0; col < matrix.length - 1; col++) {
      for (int row = col + 1; row < matrix.length; row++) {
         v = 1;
         outahere: while (m[col][col] == 0) // check if 0 in diagonal
            { // if so switch until not
               if (col + v >= tms) // check if switched all rows
                  {
                     iDF = 0;
                     break outahere;
                  }
               else {
                  for (int c = 0; c < tms; c++) {
                     temp = m[col][c];
                     m[col][c] = m[col + v][c]; // switch rows
                     m[col + v][c] = temp;
                  }
                  v++; // count row switches
                  iDF = iDF * -1; // each switch changes determinant
                                    // factor
               }
            }
         if (m[col][col] != 0) {
            if (DEBUG) {
               System.out.println("tms = " + tms + "   col = " + col
                                 + "   row = " + row);
            }
            try {
               f1 = (-1) * m[row][col] / m[col][col];
               for (int i = col; i < tms; i++) {
                  m[row][i] = f1 * m[col][i] + m[row][i];
               }
            } catch (Exception e) {
               System.out.println("Still Here!!!");
            }
         }
      }
   }
   return matrix;
} // ---------------------------------------------------------------------
det = det * iDF; // adjust w/ determinant factor
if (INFO) {
    System.out.println("Determinant: "+ det);
}
return det;