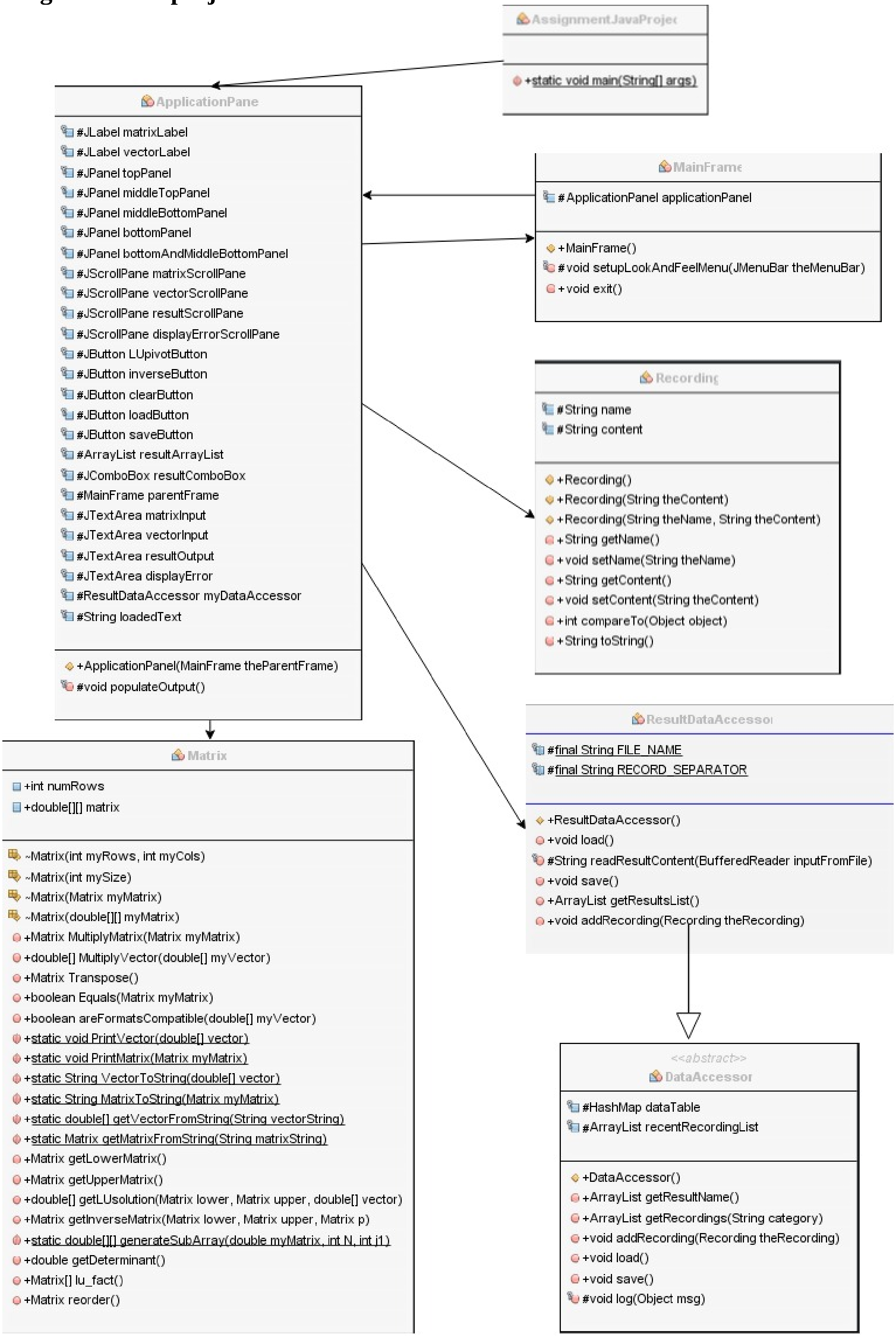
Description of the Java project with class diagram

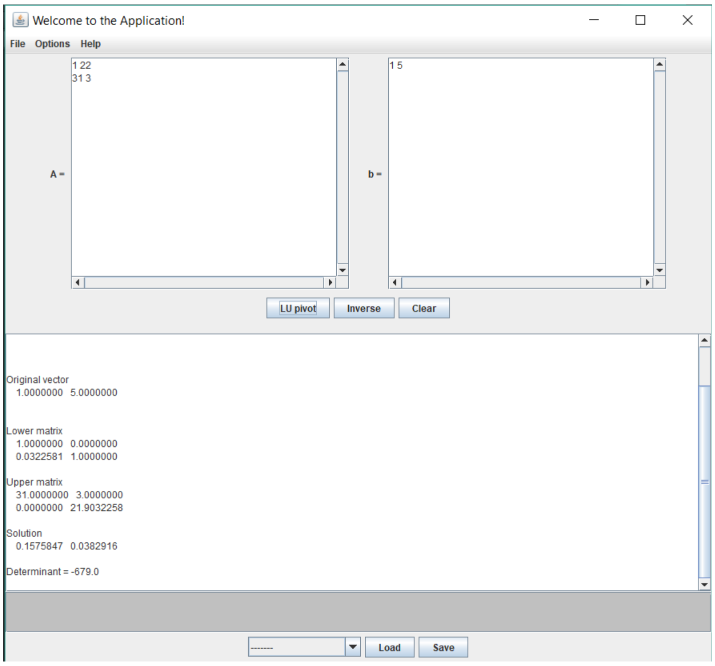
Class diagram of the project:



AssignmentJavaProje
+static void main(String[] args)
ApplicationPane
#JLabel matrixLabel
#JLabel vectorLabel
#JPanel topPanel
#JPanel middleTopPanel
#JPanel middleBottomPanel
#JPanel bottomPanel
#JPanel bottomAndMiddleBottomPanel
#JScrollPane matrixScrollPane
#JScrollPane vectorScrollPane
#JScrollPane resultScrollPane
#JScrollPane displayErrorScrollPane
#JButton LUpivotButton
#JButton inverseButton
#JButton clearButton
#JButton loadButton
#JButton saveButton
#ArrayList resultArrayList
#JComboBox resultComboBox
#MainFrame parentFrame
#JTextArea matrixInput
#JTextArea vectorInput
#JTextArea resultOutput
#JTextArea displayError
#ResultDataAccessor myDataAccessor
#String loadedText
+ApplicationPanel(MainFrame theParentFrame)
#void populateOutput()
MainFrame
#ApplicationPanel applicationPanel
+MainFrame()
#void setupLookAndFeelMenu(JMenuBar theMenuBar)
+void exit()
Recording
#String name
#String content
+Recording()
+Recording(String theContent)
+Recording(String theName, String theContent)
+String getName()
+void setName(String theName)
+String getContent()
+void setContent(String theContent)
+int compareTo(Object object)
+String toString()
Matrix
+int numRows
+double[][] matrix
~Matrix(int myRows, int myCols)
~Matrix(int mySize)
~Matrix(Matrix myMatrix)
~Matrix(double[][] myMatrix)
+Matrix MultiplyMatrix(Matrix myMatrix)
+double[] MultiplyVector(double[] myVector)
+Matrix Transpose()
+boolean Equals(Matrix myMatrix)
+boolean areFormatsCompatible(double[] myVector)
+static void PrintVector(double[] vector)
+static void PrintMatrix(Matrix myMatrix)
+static String VectorToString(double[] vector)
+static String MatrixToString(Matrix myMatrix)
+static double[] getVectorFromString(String vectorString)
+static Matrix getMatrixFromString(String matrixString)
+Matrix getLowerMatrix()
+Matrix getUpperMatrix()
+double[] getLUsolution(Matrix lower, Matrix upper, double[] vector)
+Matrix getInverseMatrix(Matrix lower, Matrix upper, Matrix p)
+static double[][] generateSubArray(double myMatrix, int N, int j1)
+double getDeterminant()
+Matrix[] lu_fact()
+Matrix reorder()
ResultDataAccesso
#final String FILE_NAME
#final String RECORD_SEPARATOR
+ResultDataAccessor()
+void load()
#String readResultContent(BufferedReader inputFromFile)
+void save()
+ArrayList getResultsList()
+void addRecording(Recording theRecording)
<<abstract>>
DataAccessor
#HashMap dataTable
#ArrayList recentRecordingList
+DataAccessor()
+ArrayList getResultName()
+ArrayList getRecordings(String category)
+void addRecording(Recording theRecording)
+void load()
+void save()
#void log(Object msg)

Description of the structure of the project:

Our projects consists in seven classes in a java package. In the following figure is an example of use of our project.



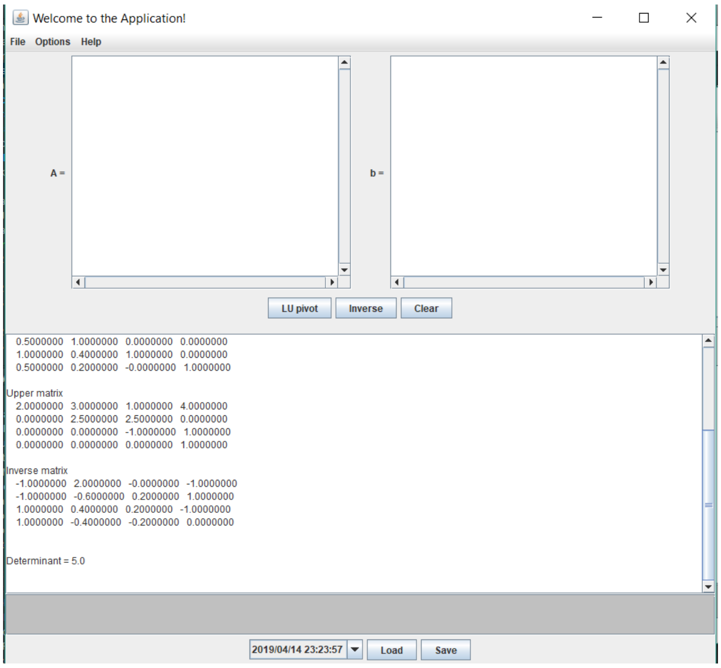
```
1 22
31 3
```

```
1 5
```

```
Original vector
  1.0000000  5.0000000


Lower matrix
  1.0000000  0.0000000
  0.0322581  1.0000000

Upper matrix
  31.0000000  3.0000000
  0.0000000  21.9032258

Solution
  0.1575847  0.0382916

Determinant = -679.0
```

Figure 1: Usage of our application to calculate a LU pivot of a matrix 2x2

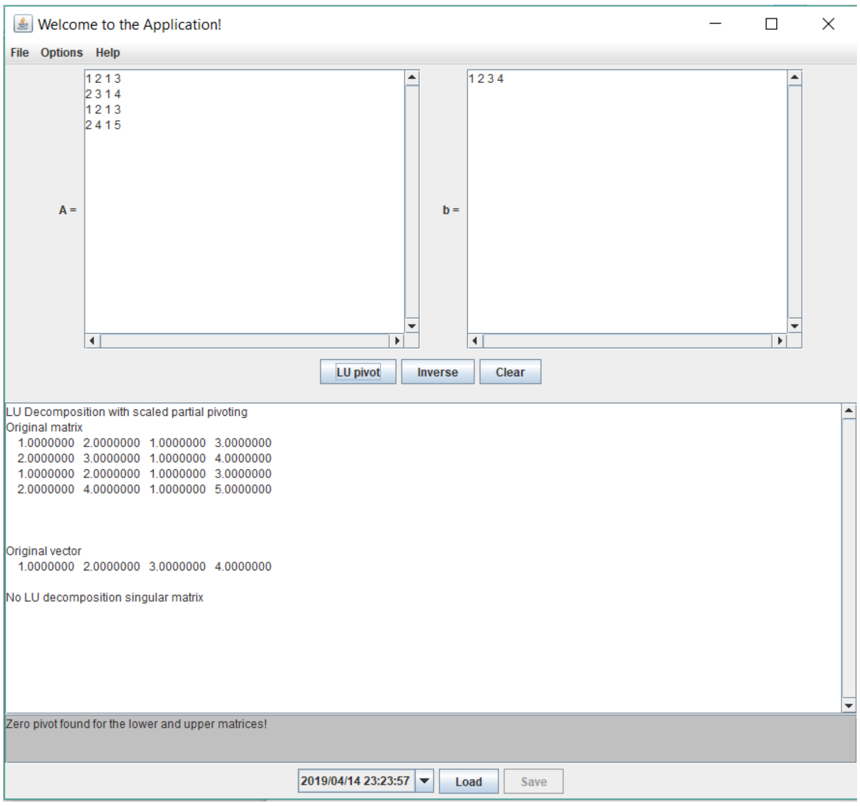
As can be seen, a matrix A can be input in the top left part of our application. At the top right part a vector b can also be input. Once that is done, the user can click on the “LU pivot” button to generate a lu pivot in the output at the bottom. He can also generate the inverse of the matrix by clicking on the “Inverse” button. If the bottom text is not empty, the user can clear it using the clear button. He can then save his result by clicking on the “Save” button. All the saved results are put inside of the result.db file in the project. Every result saved this way can be loaded by choosing one of the results in the combo box and clicking on the load button, as shown in the next figure.



```
  0.5000000  1.0000000  0.0000000  0.0000000
  1.0000000  0.4000000  1.0000000  0.0000000
  0.5000000  0.2000000  -0.0000000  1.0000000

Upper matrix
  2.0000000  3.0000000  1.0000000  4.0000000
  0.0000000  2.5000000  2.5000000  0.0000000
  0.0000000  0.0000000  -1.0000000  1.0000000
  0.0000000  0.0000000  0.0000000  1.0000000

Inverse matrix
  -1.0000000  2.0000000  -0.0000000  -1.0000000
  -1.0000000  -0.6000000  0.2000000  1.0000000
  1.0000000  0.4000000  0.2000000  -1.0000000
  1.0000000  -0.4000000  -0.2000000  0.0000000


Determinant = 5.0
```

Figure 2: Loading a previous result

As can be seen, every result is saved with the current date in the format yyyy/mm/dd hh:mm:ss as its name and as its key in the datatable. This is to make the saving of results really easy, just by clicking on the save button. We have also decided to keep the bottom part possible to edit if the user wants to change the result in some way. Moreover, if the matrix cannot be pivoted, an appropriate error message is shown as can be seen in the next figure. A similar message is shown if the matrix cannot be inverted.



```
1 2 1 3
2 3 1 4
1 2 1 3
2 4 1 5
```

```
1 2 3 4
```

```
LU Decomposition with scaled partial pivoting
Original matrix
   1.0000000  2.0000000  1.0000000  3.0000000
   2.0000000  3.0000000  1.0000000  4.0000000
   1.0000000  2.0000000  1.0000000  3.0000000
   2.0000000  4.0000000  1.0000000  5.0000000



Original vector
   1.0000000  2.0000000  3.0000000  4.0000000

No LU decomposition singular matrix
```

Zero pivot found for the lower and upper matrices!

2019/04/14 23:23:57

Figure 3: error message shown when the matrix cannot be pivoted

This interface was coded by using swing. We save the results as recordings, objects of the Recording class that have a name (string) and content (string), where the name is the current date as previously described, and the content is what is shown in the output at the bottom of the page. We have used for the database a similar approach to that seen during the exercises with data accessors and result.db that contains all our saved results. Additionnaly, we have implemented a Matrix class that handles all the calculations related to the matrices and the format of the matrices themselves. This is how we can find the LU pivot and the inverse for the input of the user. The output at the very bottom of the application contains the errors and cannot be edited. It has a length of three lines because it is the size of our longest error message. With this structure, we were able to implement the interface for the user to find LU pivots and inverses, and save and load his results.