Przykład:

$$A = \begin{matrix} 0 & 1 & 2 \\ 1 & 0 & 6 \\ -8 & 1 & 0 \end{matrix} \qquad P = \begin{matrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{matrix} \qquad L = \begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0{,}5 & 0{,}5 & 1 \end{matrix} \qquad U = \begin{matrix} -8 & 1 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 5 \end{matrix}$$

Please get size matrix: 3

Please get value matrix 9 yet: 0

Please get value matrix 8 yet: 1

Please get value matrix 7 yet: 2

Please get value matrix 6 yet: 4

Please get value matrix 5 yet: 0

Please get value matrix 4 yet: 6

Please get value matrix 3 yet: -8

Please get value matrix 2 yet: 1

Please get value matrix 1 yet: 0

Matrix P

0 0 1

1 0 0

0 1 0

Matrix L

1.00 0.00 0.00

-0.00 1.00 0.00

-0.50 0.50 1.00

Matrix U

-8.00 1.00 0.00

0.00 1.00 2.00

0.00 0.00 5.00

```
public class Exercise2 {
    public static void main(String[] args) {
        LU.LUDecomposition(LU.preaperDate());
    }
}
```

```
import java.util.InputMismatchException;
import java.util.Scanner;

import static java.lang.Math.*;

public class LU {

    public double[][] matrix;

    public int row;
    public int column;

    public int pivoting;
    public int[] pivot;

    public static int indexK;
    public static int indexJ;

    public LU(Matrix paramMatrix) {

        this.matrix = paramMatrix.getArrayCopy();
        this.row = paramMatrix.getRowDimension();
        this.column = paramMatrix.getColumnDimension();
        this.pivot = new int[row];
```

```
    repleacePivot(pivot, row);
    this.pivoting = 1;

    double[] vector = new double[row];
    initialization(vector);
}


public void initialization(double[] vector) {

    for (int j = 0; j < column; j++) {
        for (int k = 0; k < row; k++)
            vector[k] = matrix[k][j];

        double d = determination(vector, j);

        indexK = j;
        for (int i1 = j + 1; i1 < this.row; i1++)
            if (abs(vector[i1]) > abs(vector[indexK])) indexK = i1;



        if (indexK != j) {
            for (indexJ = 0; indexJ < this.column; indexJ++) {
                d = this.matrix[indexK][indexJ];
                this.matrix[indexK][indexJ] = this.matrix[j][indexJ];
                this.matrix[j][indexJ] = d;
            }

            indexJ = this.pivot[indexK];
            this.pivot[indexK] = this.pivot[j];
            this.pivot[j] = indexJ;
            this.pivoting = (-this.pivoting);
        }
```

```
        if (((j < this.row ? 1 : 0) & (this.matrix[j][j] != 0.0D ? 1 : 0)) != 0)
          for (indexJ = j + 1; indexJ < this.row; indexJ++)
            this.matrix[indexJ][j] /= this.matrix[j][j];
      }
    }

    public double determination(double[] vector, int j) {

      double result = 0;
      for (indexK = 0; indexK < row; indexK++) {
        double[] arrayOfDouble1 = matrix[indexK];



        indexJ = min(indexK, j);
        result = 0.0D;
        for (int i2 = 0; i2 < indexJ; i2++)
          result += arrayOfDouble1[i2] * vector[i2];

        int tmp = indexK;
        double[] tmp_array = vector;
        double tmp2 = (tmp_array[tmp] - result);
        tmp_array[tmp] = tmp2;
        arrayOfDouble1[j] = tmp2;
      }

      return result;
    }

    public static void repleacePivot(int[] pivot, int row) {

      for (int i = 0; i < row; i++) pivot[i] = i;
    }

    public Matrix getL() {
```

```
    Matrix result = new Matrix(row, column);
    double[][] arrayOfDouble = result.getArray();

    for (int i = 0; i < row; i++) {
        for (int j = 0; j < column; j++) {
            if (i > j) {
                arrayOfDouble[i][j] = matrix[i][j];
            } else if (i == j) {
                arrayOfDouble[i][j] = 1.0D;
            } else {
                arrayOfDouble[i][j] = 0.0D;
            }
        }
    }
    return result;
}

public Matrix getU() {

    Matrix result = new Matrix(column, column);
    double[][] arrayOfDouble = result.getArray();

    for (int i = 0; i < column; i++) {
        for (int j = 0; j < column; j++) {
            if (i <= j) {
                arrayOfDouble[i][j] = matrix[i][j];
            } else {
                arrayOfDouble[i][j] = 0.0D;
            }
        }
    }
    return result;
}

public int[] getPivot() {
```

```
    int[] result = new int[row];

    for (int i = 0; i < row; i++) {
        result[i] = pivot[i];
    }
    return result;
}

public static double multiplicationIndex(double[][] matrix1, double[][] matrix2, int x, int y) {

    int size = matrix1.length;
    double result = 0;
    for (int i = 0; i < size; i++) {

        result += matrix1[x][i] * matrix2[i][y];
    }

    return result;
}

public static double[][] multiplication(double[][] matrix1, double[][] matrix2) {

    int size = matrix1.length;
    double[][] result = new double[size][size];

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {

            result[i][j] = multiplicationIndex(matrix1, matrix2, i, j);
        }
    }
    return result;
}
```

```
public static void correctPUL(double[][] matrixP, double[][] matrixL, double[][] matrixU) {
    Matrix.showMatrix((multiplication(
            multiplication(matrixP, matrixL), matrixU)), "Check question:");
}

public static double[][] preaperDate() {

    boolean correct = true;
    double[][] result;
    int size = 0;
    Scanner input;

    do {
        input = new Scanner(System.in);
        try {
            System.out.print("Please get size matrix: ");
            size = input.nextInt();
        } catch (InputMismatchException ex) {
            correct = false;
        }
    } while (!correct);

    result = new double[size][size];


    correct = false;
    while (!correct) {

        input = new Scanner(System.in);
        correct = true;
        int how = (int) Math.pow(size, 2);
        for (int i = 0; i < size && correct == true; i++) {
            for (int j = 0; j < size && correct == true; j++) {

                try {
```

```
                System.out.print("Please get value matrix " + how + " yet: ");
                result[i][j] = input.nextDouble();
            } catch (InputMismatchException ex) {
                correct = false;
            }

            how--;
        }
    }
  }

  return result;
}

public static String LUDecomposition(double[][] array) {

  Matrix A = new Matrix(array);
  LU lu = new LU(A);

  Matrix L = lu.getL();
  Matrix U = lu.getU();
  int[] p = lu.getPivot();

  double[][] printL = L.getArray();
  double[][] printU = U.getArray();

  Matrix.showPivot(p);
  L.showMatrix("Matrix L");
  U.showMatrix("Matrix U");

  //correctPUL(Matrix.getPivot(p, U.determinationU()), printL, printL);

  return "END";
```

```
    }
}

public class Matrix {

    private double[][] matrix;

    private int row;
    private int column;

    public Matrix(double[][] array) {

        this.row = array.length;
        this.column = array[0].length;
        this.matrix = array;
    }

    public Matrix(int paramInt1, int paramInt2) {

        this.row = paramInt1;
        this.column = paramInt2;
        this.matrix = new double[paramInt1][paramInt2];
    }

    public double[][] getArray() {
        return this.matrix;
    }

    public double[][] getArrayCopy() {

        double[][] arrayOfDouble = new double[this.row][this.column];
        for (int i = 0; i < this.row; i++) {
            for (int j = 0; j < this.column; j++) {
                arrayOfDouble[i][j] = this.matrix[i][j];
            }
```

```
    }
    return arrayOfDouble;
}

public double determinationU(){

    double result = 1;
    int size = matrix.length;
    for(int i =0;i < size;i++){
        result *= matrix[i][i];
    }
    return result;
}

public int getRowDimension() {

    return this.row;
}

public int getColumnDimension() {

    return this.column;
}

public void showMatrix(String communicat) {

    System.out.println("\n" + communicat);
    for (double[] va : matrix) {
        for (double var : va) {
            System.out.printf("%-7.2f ", var);
        }
        System.out.println();
    }
    System.out.println();
}
```

```
public static void showMatrix(double[][] matrix, String communicat) {

    new Matrix(matrix).showMatrix(communicat);
}

public static void showPivot(int[] pivot) {

    int size = pivot.length;

    System.out.println("\nMatrix P");
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {

            if (pivot[i] == j)
                System.out.print(" 1 ");
            else
                System.out.print(" 0 ");
        }
        System.out.println();
    }
}

public static double[][] getPivot(int[] pivot, double determination) {

    int size = pivot.length;

    double[][] result = new double[size][size];

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {

            if (pivot[i] == j)
                result[i][j] = 1/ determination;
            else
```

```
                result[i][j] = 0;
            }
        }
        return result;
    }
}
```

Piwot jest potrzebny do obliczenia rozkładu LU dla macierzy które mają 0 na diagonali. Lecz pełny piwot jest nie opłacalny. Dzięki faktoryzacji LU możemy szybko i w łatwy sposób obliczyć wyznacznik macierzy, jest ona równa iloczynowi liczb na diagonali w macierzy U.