ДОДАТОК А

Текст програми

alternative

```
public class Alternative {
int index;
float[] criteriaValues;
Alternative(int criteriaQuantity) {
index = 0;
criteriaValues = new float[criteriaQuantity];
for (int i = 0; i < criteriaQuantity; i++) {
criteriaValues[i] = 0;
        }
    }
Alternative(int index, float[] values) {
criteriaValues = new float[values.length];
criteriaValues = values;
        this.index = index;
    }
};
```

BestAndWorst

```
public class BestAndWorst {
public int indexOfBest, indexOfWorst;
```

```
public float best, worst;
BestAndWorst(){
indexOfBest = 0;
indexOfWorst = 0;
best = 0f;
worst = 0f;
}
};
```

MyMath

```
import java.util.Arrays;
import java.util.Collections;
import java.util.Random;
import java.util.Scanner;
import java.lang.Math;

public class MyMath {
static Random rng = new Random(11131);//11131

public static float getRandomNumber(int min, int max){
float randomNumber = 0f;
if ( (min == 0) && (max == 1) ) {
randomNumber = rng.nextFloat();
```

```
} else {
int temp = 0;
temp = min + rng.nextInt(max + 1 - min); // поумолчанию nextInt выдаётзначениеневключительно, поэтомудобавляем 1
randomNumber = (float)temp;
        }
return randomNumber;
}

public static float[][] getAlternativesArray(int quantity, int[][] criteriaRanges){
        /*
параметр quantity - кількістьальтернатив; criteriaRanges - діапазонзначеньпокожномуізкритеріїв;
isHigherBetter - показує чи краще якщо значення по критерію більше, чи навпаки
*/
float[][] alternatives = new float[criteriaRanges.length][quantity];
for (int i = 0; i < alternatives.length; i++) {
int min = criteriaRanges[i][0];
int max = criteriaRanges[i][1];
for (int j = 0; j < alternatives[i].length; j++) {
alternatives[i][j] = getRandomNumber(min, max);
}
        }
return alternatives;
    }
```

```
public static BestAndWorst[] getBestAndWorst(float[][] array, boolean[]
isHigherBetter) {

/*

    Дана функція видає масив найкращих та найгірших значень за всіма
критеріями

    та номери альтернатив, що мають ці значення. Для цього створили

    тип даних (клас) BestAndWorst

    перший параметр - масив альтернатив, другий - чи є більші значення по
критерію кращими

*/

BestAndWorst[] bestAndWorst = new BestAndWorst[array.length];

for (int i = 0; i < bestAndWorst.length; i++) { bestAndWorst[i] = new
BestAndWorst(); }


for (int i = 0; i < array.length; i++){

float min = 999999.0f, max = 0.0f;

for (int j = 0; j < array[i].length; j++){

if (array[i][j] < min){

            /* Знаходимомінімум */

min = array[i][j];

if (!isHigherBetter[i]){

/*

                Якщо для критерія чим менше значення тим краще, то

мінімальне значення - найкращий варіант

                */

bestAndWorst[i].indexOfBest = j;
```

```
bestAndWorst[i].best = min;
} else {
/*
                    Якщо для критерія чим більше значення тим краще, то
мінімальне значення - найгірший варіант
                    */
bestAndWorst[i].indexOfWorst = j;
bestAndWorst[i].worst = min;
}
              }
if (array[i][j] > max){
/* Знаходимо максимум */
                  max = array[i][j];
if (isHigherBetter[i]){
bestAndWorst[i].indexOfBest = j;
bestAndWorst[i].best = max;
                  } else {
bestAndWorst[i].indexOfWorst = j;
bestAndWorst[i].worst = max;
}
              }
           }
       }
return bestAndWorst;
   }
```

```
public static float[][] normalize(float[][] array, BestAndWorst[] arg2, int[][] ranges) {

/* Функцияполезности */

float[][] normalizedArray = new float[array.length][array[0].length];

for (int i = 0; i < array.length; i++) {

if ( !((ranges[i][0]) == 0 && (ranges[i][1] == 1))) { /*
перевіряємочипотрібнонормалізувати */

for (int j = 0; j < array[i].length; j++) {

                normalizedArray[i][j] = Math.abs( (array[i][j] -
arg2[i].worst)/(arg2[i].best - arg2[i].worst) );

            }

          } else {

normalizedArray[i] = array[i];

}

       }

return normalizedArray;

   }


public static byte compareArrays(float[] array1, float[] array2) {

byte result = 5;

if ( (array1[0] > array2[0]) && (array1[1] > array2[1]) && (array1[2] > array2[2])
&& (array1[3] > array2[3]) ) {

result = 0; /* функція повертає 0, якщо перша яльтернатива краща за другу за
всіма критеріями */

}

if ( (array1[0] < array2[0]) && (array1[1] < array2[1]) && (array1[2] < array2[2])
&& (array1[3] < array2[3]) ) {
```

```
result = 1; /* функція повертає 1, якщо перша альтернатива гірша за другу за всіма критеріями */
        }
        else { result = 2; } /* функція повертає 2, якщо перша альтернатива за якимись критеріями краще, за якимись менше*/
return result;
    }

public static float[] fillWithZeroes(float[] inputArray) {
float[] outputArray = new float[inputArray.length];
for (int i = 0; i < inputArray.length; i++) {
outputArray[i] = 0;
        }
return outputArray;
    }

public static boolean isZero(float[] arr) {
if ( (arr[0] == 0) && (arr[1] == 0) && (arr[2] == 0) && (arr[3] == 0) ) {
return true;
        } else {
return false;
}
}

    /* функция принимает массив альтернатив и возвращает меньший массив альтернатив, что нельзя уличшить ни по одному из крит. */
```

```
public static Alternative[] pairComparison(Alternative[] inputAlternatives) {

Alternative[] outputAlternatives;

byte temp;

int deletedAlternativesCounter = 0;

/* спочаткупотрібновизначитискількиальтернативвидаляється */

outerLoop:

for (int i = 0; i < inputAlternatives.length; i++) {

if ( isZero(inputAlternatives[i].criteriaValues) ) { continue; } /*
якщовсізначеннянульовізначитьальтернативувидалено */

for (int j = 0; j < inputAlternatives.length; j++) {

if (i == j) {continue;} /* немасенсупорівнюватиальтернативусамузсобою*/

if ( isZero(inputAlternatives[j].criteriaValues) ) { continue; } /*
якщовсізначеннянульовізначитьальтернативувидалено */

temp = compareArrays(inputAlternatives[i].criteriaValues,
inputAlternatives[j].criteriaValues);

if (temp == 0) {

deletedAlternativesCounter++;

inputAlternatives[j].criteriaValues =
fillWithZeroes(inputAlternatives[j].criteriaValues);

} else if (temp == 1) {

deletedAlternativesCounter++;

inputAlternatives[i].criteriaValues =
fillWithZeroes(inputAlternatives[i].criteriaValues);

continue outerLoop;

            } else if (temp == 2) { continue;}

        }

}
```

```
outputAlternatives = new Alternative[inputAlternatives.length -
deletedAlternativesCounter];

int counter = 0;

for (int i = 0; i < inputAlternatives.length; i++) {

if ( !(isZero(inputAlternatives[i].criteriaValues)) ) {

outputAlternatives[counter] = inputAlternatives[i];

counter++;

}

      }


return outputAlternatives;

   }


public static Alternative findBestAlternativeByCriterion(Alternative[] inputAlts, int
criterionNo) {

/* В дану функцію в якості параметрів подається масив альтернатив та номер
критерію

      Функція повертає найкращу альтернативу за заданим критерієм

*/

Alternative outputAlt = new Alternative(inputAlts[0].criteriaValues.length);

float max = 0f;

for (int i = 0; i < inputAlts.length; i++) {

if ( inputAlts[i].criteriaValues[criterionNo] > max) {

max = inputAlts[i].criteriaValues[criterionNo];

outputAlt = inputAlts[i];

         }
```

```
        }
return outputAlt;
}

public static Alternative[] expandAltArray(Alternative[] inputAlts, Alternative
bestAlt, int criterionNo, float plusMinus) {
/* В дану функцію в якості параметрів подається масив альтернатив,
        найкраща альтернатива за критерієм, номер цього критерія та діапазон
розширення
        Функція повертає розширений масив альтернатив
        */
Alternative[] outputAlts;
float rangeFrom = bestAlt.criteriaValues[criterionNo] - plusMinus;
float rangeTo = bestAlt.criteriaValues[criterionNo] + plusMinus;
int fittingAltsCount = 0; // кількістьальтернатив, щопідходятьпоумовам
/* Спочатку визначаємо скільки альтернатив задоволяняють умовам, що задати
розмір вихідного масиву */
for (int i = 0; i < inputAlts.length; i++) {
if ( (inputAlts[i].criteriaValues[criterionNo] > rangeFrom) &&
(inputAlts[i].criteriaValues[criterionNo] < rangeTo) ) {
fittingAltsCount++;
            }
        }
outputAlts = new Alternative[fittingAltsCount]; /* задаєморозмірвихідногомасиву
*/
/* Тепер вносимо значення в вихідний масив */
int counter = 0;
```

```
for (int i = 0; i < inputAlts.length; i++) {

if ( (inputAlts[i].criteriaValues[criterionNo] > rangeFrom) &&
(inputAlts[i].criteriaValues[criterionNo] < rangeTo) ) {

outputAlts[counter] = inputAlts[i];

counter++;

}

      }

return outputAlts;

   }


public static Alternative[] lexicograficOptimization(Alternative[] inputAlts, int[]
criteriaImportance) {

/* Функція приймає вхідний масив альтернатив та масив, в якому критерії
розташовані в порядку важливості

      Функція повертає двухмірний масив альтернатив

      Перший рядок - найкращі за найважливішим критерієм, другий - найкращі
за другим по важливості і так далі

      Останнійрядокрезультат

      */

Alternative[] outputAlts, currentAlts,previousAlts;

Alternative[] bestAtlsByCrit = new Alternative[criteriaImportance.length];

previousAlts = inputAlts;

for (int i = 0; i < criteriaImportance.length; i++) {

bestAtlsByCrit[i] = findBestAlternativeByCriterion(previousAlts,
criteriaImportance[i]);

currentAlts = expandAltArray(previousAlts, bestAtlsByCrit[i], criteriaImportance[i],
0.2f);
```

```
previousAlts = currentAlts;
        }
return previousAlts;
    }



// public static float[][] pairComparison(float[][] inputArray) {
    //     Alternative[] alt = new Alternative[inputArray[0].length];
    //     for (int i = 0; i < alt.length; i++) {
    //         alt[i].index = i;
    //         alt[i].criteriaValues = inputArray[]
    //     }
    // }

    // public static byte[] calculateCriteriaImportance() {
    //     /* Теперьопределяемстепениважностикритериев
//         0 - Производительность
    //         1 - Стоимость
    //         2 - Надежность
    //         3 - точность
    //     */
    //     byte[] criteria = new byte[4]; // Самому важному критерию будет присвоено значение 4, наименее важному - 1
//     int counter = 0;
//     Scanner userInput = new Scanner(System.in);
```

```
//     String[] criteriaNames = {"Производительность","Стоимость",
"Надежность","Точность"};


//     askUser:
   //     while (counter<4){
   //         int temp = 0;
//         System.out.printf("Выберите самый важный из этих критериев(введите
его номер)\n");
//         for (int i = 0; i<criteriaNames.length;i++){
//             if(criteria[i]==0 && counter!=3){ // Не выводим те критерии, важность
которых уже определена
//                 System.out.printf("%d - %s  ", i, criteriaNames[i]);
   //             } else if(counter == 3 && criteria[i]==0){
//                 // Последний критерий задаём автоматически
   //                 criteria[i]=1;
//                 counter++;
   //                 continue askUser;
   //             }
   //         }
   //         System.out.printf("\n>> ");
//         temp = userInput.nextByte();
//         if (temp>3 || temp<0){
//             // Проверяем ввёл ли пользователь правильное число, если нет -
повторяем цикл
   //             System.out.printf("Введён неправильный номер!\n\n");
//             temp = 0;
   //             continue;
```

```
    //         } elseif(criteria[temp] !=0){
//             System.out.printf("Важность введённого критерия уже задана!\n\n");
    //             temp = 0;
    //             continue;
    //         } else {
    //             criteria[temp] = (byte)(4-counter); // Добавляем единицу, потому что отсчёт начимнается с 0
//             counter++;
    //         }


    //     }
    //     userInput.close();
    //     return criteria;
// }


// public static int[] filterWorst(int leaveAmount,float[] inputArray,int[] inputIndexes) {
//     /* Удаляем худшие варианты по каждому критерию
    //      Для этого сортируем массив от большего к меньшему
    //      и заносим в новый массив только нужное кол-во */
//     float[] array = inputArray;
    //     float[] result = new float[leaveAmount]; // результирующиймассив
//     int[] outputIndexes = new int[leaveAmount]; // создаём массив индексов, чтобы потом снова расставить элементы по порядку


//     float temp = 0f;
```

```
//     int n = array.length;
    //     for(int i=0; i < n - 1; i++){
    //         for(int j=0; j < n-i-1; j++){
    //             if(array[j] < array[j+1]){
//                 // Меняем местами элементы массива и соответствующие индексы
//                 temp = array[j];
    //                 array[j] = array[j+1];
    //                 array[j+1] = temp;
    //                 temp = inputIndexes[j];
    //                 inputIndexes[j] = inputIndexes[j+1];
    //                 inputIndexes[j+1] = (int)temp;
//             }
    //         }
    //     }
    //     // Теперьубираемлишниеварианты
//     for(int i=0; i < leaveAmount; i++){
    //         result[i] = array[i];
    //         outputIndexes[i] = inputIndexes[i];
//     }
    //     // Расставляемэлементыпопорядку

//     for(int i=0; i < inputArray.length; i++){
    //         for(int j=1; j < (inputArray.length-i); j++){
    //             if(inputIndexes[j-1] > inputIndexes[j]){
    //                 temp = inputIndexes[j-1];
```

```
    //                inputIndexes[j-1] = inputIndexes[j];
    //                inputIndexes[j] = (int)temp;
    //                temp = array[j-1];
    //                array[j-1] = array[j];
    //                array[j] = temp;
//            }
    //        }
    //    }
    //    return outputIndexes;
    // }

    // критерий: имя, предел значений, больше-лучше или наоборот


// public static void main(String[] args) {
//    /**
    //     * Производительность (80-4000)
    //     * Стоимость (100-800000)
    //     * Надежность (100-2000)
    //     * точность ( 0-1)
    //     */
    //    final byte NUMBER_OF_VARIANTS = 100; // кол-во вариантов
//    final byte PRODUCTIVITY_MIN = 80;
    //    final short PRODUCTIVITY_MAX = 4000;
    //    final byte COST_MIN = 100;
```

```
    //     final int COST_MAX = 800000;
    //     final byte RELIABILITY_MIN = 100;
    //     final short RELIABILITY_MAX = 2000;
    //     final byte PRECISION_MIN = 0;
    //     final short PRECISION_MAX = 1000;


//     /* Создаём массив для каждого из критериев */
//     int[] productivity = randomArray(NUMBER_OF_VARIANTS,
PRODUCTIVITY_MIN, PRODUCTIVITY_MAX); //производительность
    //     int[] cost = randomArray(NUMBER_OF_VARIANTS, COST_MIN,
COST_MAX); //стоимость
    //     int[] reliability = randomArray(NUMBER_OF_VARIANTS,
RELIABILITY_MIN, RELIABILITY_MAX); //надёжность
    //     int[] precision = randomArray(NUMBER_OF_VARIANTS,
PRECISION_MIN, PRECISION_MAX); //точность


//     // Объединяемводиндвхмерныймассив
//     //int[][] variants = {productivity,cost,reliability,precision};
    //     /* Выводимполучившиесямассивы */


    //
System.out.printf("|\tНомервариантa\t|\tПроизводительность\t|\tСтоимость\t|\tНад
ёжность\t|\tТочность\t|\n\n\n");


    //     for (int i = 0; i < 100; i++){
    //         System.out.printf("|\t\t%d\t|\t\t%d\t\t|\t%d\t\t|\t%d\t\t|\t%d\t\t|\n",
i+1,productivity[i],cost[i],reliability[i],precision[i]);
//     }
```

```
    //     /* Находим лучшие и худшие по каждому критерию
    //     * Находим макс производительность
    //     */
//     int[] best_productivity = findMax(productivity); // Дляпроизводительности, больше - лучше
//     int[] worst_productivity = findMin(toFloat(productivity) );
    //     int[] best_cost = findMin(toFloat(cost)); // Длястоимости, меньше - лучше
    //     int[] worst_cost = findMax(cost);
    //     int[] best_reliability = findMax(reliability); // Длянадёжности, больше - лучше
    //     int[] worst_reliability = findMin(toFloat(reliability) );
    //     int[] best_precision = findMax(precision); // Дляточности, больше - лучше
    //     int[] worst_precision = findMin(toFloat(precision));
    //     System.out.printf("\nК1(производительность): Лучший - №%d: %d, Худший - №%d: %d\n",best_productivity[0],best_productivity[1], worst_productivity[0],worst_productivity[1]);
    //     System.out.printf("К2(стоимость): Лучший - №%d: %d, Худший - №%d: %d\n",best_cost[0],best_cost[1], worst_cost[0],worst_cost[1]);
    //     System.out.printf("К3(надёжность): Лучший - №%d: %d, Худший - №%d: %d\n",best_reliability[0],best_reliability[1], worst_reliability[0],worst_reliability[1]);
    //     System.out.printf("К4(точность): Лучший - №%d: %d, Худший - №%d: %d\n",best_precision[0],best_precision[1], worst_precision[0],worst_precision[1]);
//     /**Находим p i */
//     float[] normalizedArray_k1 = findnormalizedArray(productivity, best_productivity[1], worst_productivity[1]);
```

```
    //     float[] normalizedArray_k2 = findnormalizedArray(cost, best_cost[1],
worst_cost[1]);

    //     float[] normalizedArray_k3 = findnormalizedArray(reliability,
best_reliability[1], worst_reliability[1]);

    //     float[] normalizedArray_k4 = findnormalizedArray(precision,
best_precision[1], worst_precision[1]);



    //     /* Выводимполучившиесямассивы */



    //     System.out.printf("|\tНомерварианта\t|\tnormalizedArray_k1\t
|\tnormalizedArray_k2\t |\tnormalizedArray_k3\t |\tnormalizedArray_k4\t |\n\n\n");

    //     for (int i = 0; i < 100; i++){

    //         System.out.printf("|\t\t%d\t|\t%f |\t%f |\t%f |\t%f |\n",
i+1,normalizedArray_k1[i],normalizedArray_k2[i],normalizedArray_k3[i],normalize
dArray_k4[i]);

//     }



    //     /* Теперь определяем степени важности критериев

    //         0 - Производительность

    //         1 - Стоимость

    //         2 - Надежность

    //         3 - точность

    //     */



//     byte[] criteriaImportance = newbyte[4];

//     criteriaImportance = calculateCriteriaImportance();
```

```
    //     System.out.printf("\n\n");


    //     for (int i=0;i<criteriaImportance.length;i++){
//         int sum = 0;
//         for (int j=0;j<criteriaImportance.length;j++){
    //             if (i == j){
    //                 System.out.printf("\t ");
    //             } else if (criteriaImportance[i]>criteriaImportance[j]){
    //                 System.out.printf("\t1");
    //                 sum++;
    //             } else {
    //                 System.out.printf("\t0");
//             }
    //         }
//         System.out.printf(" | %d\n",sum);
//     }
    //     // Расставляем критерии в порядке значимости
    //     byte[] criteria = {0,1,2,3};
    //     String[] criteriaNames = {"Производительность", "Стоимость",
"Надежность", "Точность"};
//     for(int i=0; i < criteria.length; i++){
    //         for(int j=1; j < (criteria.length-i); j++){
    //             if(criteriaImportance[j-1] < criteriaImportance[j]){
    //                 byte temp = 0;
    //                 String tempString = "";
    //                 temp = criteria[j-1];
```

```
//                tempString = criteriaNames[j-1];
//                criteria[j-1] = criteria[j];
//                criteriaNames[j-1] = criteriaNames[j];
//                criteria[j] = temp;
//                criteriaNames[j] = tempString;
//            }
//        }
//    }
//    System.out.printf("\n\n");
//    for(int i=0; i < criteria.length; i++){
//        System.out.printf("K%d(%s)>",criteria[i]+1,criteriaNames[i]);
//    }
//    System.out.printf("\n\n");



//    // Объединяемводиндвухмерныймассив
//    float[][] variants =
{normalizedArray_k1,normalizedArray_k2,normalizedArray_k3,normalizedArray_k
4};
//    // Чтобы не потерять номера вариантов создадим массив индексов
//    int[] variantIndexes = new int[normalizedArray_k1.length];
//    for (int i = 0; i < variantIndexes.length; i++){
//        variantIndexes[i]=i+1;
//    }
//    int[] leaveAmount = {20,10,5,2}; // в этом массиве указываем сколько
вариантов оставлять по каждому критерию
```

```
    //     // Отсеиваем варианты по каждому критерию по найденой важности
    //     float[][] previousVariants = new float[4][100]; // массив с отфильтрованными значениями
    //     previousVariants = variants;
    //     int[] previousIndexes = new int[100]; // массив с индексами лучших вариантов по каждому критерию
    //     // Для начала просто заполним массив индексов числами от 0 до 99
//     for (int i = 0; i < 100; i++) { previousIndexes[i]=i; }
    //     int[] newIndexes;
    //     float[][] newVariants;
    //     for (int i = 0; i < criteria.length; i++){
    //         newIndexes = new int[ leaveAmount[i] ];
    //         newVariants = new float[4][ leaveAmount[i] ];// каждыйновыймассивменьшепредыдущего
    //         newIndexes = filterWorst( leaveAmount[i], previousVariants[ criteria[i] ],previousIndexes);
    //         previousIndexes = new int [ leaveAmount[i] ];
//         previousIndexes = newIndexes;
    //         // Теперь заполняем массив оптимальных с помощью индексов
//         for (int m=0;m<4;m++){
    //             for (int n=0;n<leaveAmount[i];n++){
    //                 newVariants[m][n] = variants[m][ newIndexes[n] ];
//             }
    //         }
//         previousVariants = new float[4][ leaveAmount[i] ];
//         previousVariants = newVariants;
    //         // выводимдляпроверки
```

```
//        System.out.printf("\n\n%dлучшихвариантовпокритериюК%d(%s):\n\n",
leaveAmount[i], criteria[i]+1, criteriaNames[i]);
   //        System.out.printf("#\t|\t\tK1|\t\tK2|\t\tK3|\t\tK4|\n\n");
   //        for (int m=0;m<leaveAmount[i];m++){

   //
System.out.printf("%d\t|\t%f|\t%f|\t%f|\t%f|\n",newIndexes[m]+1,newVariants[0][m],
newVariants[1][m],newVariants[2][m],newVariants[3][m]);

//        }

   //     }

   //     //filterWorst(20, variants[0]);
   // }
};
```

MyFrame

```
import javax.swing.JFrame;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.JPanel;
```

```
import javax.swing.JScrollPane;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.*;

public class MyFrame extends JFrame {
    JTable table1, table2, table3, table4;
    JTextArea textArea, currentStepText;
    JPanel centralPanel, buttonPanel;
    JButton nextStepButton,backButton;
String[] currentStep;
    JScrollPane scrollPane;
String[] columnNames;
int counter = 0;

    MyFrame(String[][] tableData1, String[][] tableData2, String[] criteriaNames,
String bestAndWorst, String[][] tableData3, String[][] tableData4) {
/*
        Функція MyFrame() - конструктор
        конструктор - функція, що виконується при створенні об'єкту даного типу
        в даному випадку цей об'єкт створюється в методі main класу Main
*/
```

```
currentStep = new String[5];
currentStep[0]="Отримання альтертатив";
currentStep[1]="Знаходження найкращих та найгірших значень по критеріям";
currentStep[2]="Нормалізація значень";
currentStep[3]="Метод парних порівнянь";
currentStep[4]="Метод лексикографічної оптимізації: результат";
centralPanel = new JPanel();
centralPanel.setLayout( new FlowLayout(FlowLayout.CENTER) );
buttonPanel = new JPanel();
nextStepButton = new JButton("Наступний етап >");
backButton = new JButton("< Назад");
currentStepText = new JTextArea("Поточний етап: "+currentStep[counter]);
currentStepText.setEditable(false);
textArea = new JTextArea();
      //buttonPanel.setBackground(Color.GREEN);
buttonPanel.add(currentStepText);
buttonPanel.add(backButton);
buttonPanel.add(nextStepButton);
buttonPanel.setLayout( new FlowLayout(FlowLayout.RIGHT) );
backButton.setEnabled(false);
      /*
       Створюємо массив з назвами стовбців таблиці
Перший стовбець - номер альтернативи
       Інші стовбці - назви критеріїв
*/
```

```
columnNames = new String[criteriaNames.length + 1];
columnNames[0] = "№ Альтернативи";
for (int i = 1; i < columnNames.length; i++){
columnNames[i] = criteriaNames[i - 1];
        }
        /* Початкові налаштування таблиці */
        table1 = new JTable(tableData1, columnNames);
table1.setEnabled(false);
        table2 = new JTable(tableData2, columnNames);
textArea.setText(bestAndWorst);
table2.setEnabled(false);
        table3 = new JTable(tableData3, columnNames);
table3.setEnabled(false);
        table4 = new JTable(tableData4, columnNames);
table4.setEnabled(false);
        //centralPanel.add(table1);
        //centralPanel.add(textArea);
        //centralPanel.add(table2);
scrollPane = new JScrollPane(table1);
        //scrollPane.setViewportView(table1);

        /* Обробники натискань кнопок */
backButton.addActionListener(new ActionListener(){
/* Функція, щовиконуєтьсяпринатисканнікнопки "Назад" */
@Override
```

```
public void actionPerformed(ActionEvent e) {
counter--;
switch (counter) {
case 0:
backButton.setEnabled(false);
setTitle("Початковий масив альтернатив");
scrollPane.setViewportView(table1);
break;
case 1:
scrollPane.setViewportView(textArea);
setTitle("Найкращі та найгірші");
break;
case 2:
scrollPane.setViewportView(table2);
setTitle("Нормалізовані значення");
break;
case 3:
scrollPane.setViewportView(table3);
setTitle("Метод парних порівнянь");
break;
case 4:
scrollPane.setViewportView(table4);
setTitle("Метод лексикографічної оптимізації: результат");
break;
            }
```

```
currentStepText.setText("Поточний етап: "+currentStep[counter]);
update(getGraphics());
            }
        });
nextStepButton.addActionListener(new ActionListener(){
/* Функція, щовиконуєтьсяпринатисканнікнопки "Наступнийетап" */
@Override
public void actionPerformed(ActionEvent e) {
if (counter < currentStep.length - 1) { counter++; backButton.setEnabled(true);}
switch (counter) {
case 1:
scrollPane.setViewportView(textArea);
setTitle("Найкращі та найгірші");
break;
case 2:
scrollPane.setViewportView(table2);
setTitle("Нормалізовані значення");
break;
case 3:
scrollPane.setViewportView(table3);
setTitle("Метод парних порівнянь");
break;
case 4:
scrollPane.setViewportView(table4);
setTitle("Метод лексикографічної оптимізації: результат");
```

```
break;
            }
currentStepText.setText("Поточний етап: "+currentStep[counter]);
update(getGraphics());
          }
       });

       /* Початкові налаштування вікна */

setLayout( new BorderLayout() );
       //add(textArea, BorderLayout.WEST);
add(scrollPane, BorderLayout.CENTER);
setTitle("Початковий масив альтернатив");
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

add(buttonPanel, BorderLayout.SOUTH);
setMinimumSize(new Dimension(800, 600));
setVisible(true);
   }
};
```

Main

```
public class Main {
/* Передтимяквиконуватикодзадаємонеобхібнізмінні; final - константа */
final static byte NUMBER_OF_VARIANTS = 100; // кількість альтернатив
/* Задаємо діапазон значень по критеріях */
final static int[][] CRITERIA_RANGES = {
{ 5, 100 }, // продуктивність
{ 100, 800000 }, // ціна
{ 0, 1 }, // надійність
{ 0, 1 } // точність
    };
    /* Задаємо чи є більші значення за критерієм кращими */
final static boolean[] isHigherBetter = {
true, // продуктивність
false, // ціна
true, // надійність
true // точність
    };
    /* Задаємо назви критеріїв */
final static String[] criteriaNames = {"Продуктивність", "Ціна", "Надійність",
"Точність"};
static BestAndWorst[] bestAndWorst = new BestAndWorst[criteriaNames.length];
final static int[] criteriaImportance = {1,0,2,3};/* Вказуємо критерії від
найважливішого до найменш важливого */


public static void main(String[] args) {
```

```
        /* Отримуємо масив альтернатив */

float[][] alternatives = new
float[CRITERIA_RANGES.length][NUMBER_OF_VARIANTS];

float[][] normalizedAlternatives = new
float[CRITERIA_RANGES.length][NUMBER_OF_VARIANTS];

Alternative[] normalizedAlternativesObj = new
Alternative[NUMBER_OF_VARIANTS];

alternatives = MyMath.getAlternativesArray(NUMBER_OF_VARIANTS,
CRITERIA_RANGES);

        //normalizedAlternatives = MyMath.normalize(array, best, worst,
isHigherBetter);




        /* Знаходимо найкращі та найгірші */

bestAndWorst = MyMath.getBestAndWorst(alternatives, isHigherBetter);

        String bestAndWorstString = "";

for (int i = 0; i < alternatives.length; i++) {

            bestAndWorstString += "Найкраще значення по критерію
К"+(i+1)+"("+criteriaNames[i]+") = "

              +bestAndWorst[i].best+", альтернатива
№"+(bestAndWorst[i].indexOfBest+1)+"\n";

            bestAndWorstString += "Найгірше значення по критерію
К"+(i+1)+"("+criteriaNames[i]+") = "

              +bestAndWorst[i].worst+", альтернатива
№"+(bestAndWorst[i].indexOfWorst+1)+"\n";

        }
```

```
        /* Нормалізуємо значення */
normalizedAlternatives = MyMath.normalize(alternatives,
bestAndWorst,CRITERIA_RANGES);
float[][] temp = new
float[normalizedAlternatives[0].length][normalizedAlternatives.length]; /* массив
100 x 4, вместо 4 x 100 */
for (int i = 0; i < normalizedAlternatives.length; i++) {
for (int j = 0; j < normalizedAlternatives[i].length; j++) {
temp[j][i] = normalizedAlternatives[i][j];
            }
        }




for (int i = 0; i < temp.length; i++) {
            //normalizedAlternativesObj[i].index = i + 1; // заполняем массив объектов
            //normalizedAlternativesObj[i].criteriaValues = temp[i];
normalizedAlternativesObj[i] = new Alternative(i + 1, temp[i]);
        }


        /* Метод парних порівнянь */
Alternative[] pairCompAlternatives =
MyMath.pairComparison(normalizedAlternativesObj);


        /* Метод лексикографічної оптимізації */
Alternative[] result = MyMath.lexicograficOptimization(pairCompAlternatives,
criteriaImportance);
```

```
/* Форматуємо отримані дані для показу в таблиці */

String[][] tableData1 = new String[alternatives[0].length][alternatives.length + 1]; // Додаємо 1 стовбець для номерів варіантів

for (int j = 0; j < tableData1.length; j++) {

tableData1[j][0] = String.valueOf(j + 1); // Вносимо номери варіантів

for (int i = 0; i < tableData1[0].length - 1; i++) {

tableData1[j][i + 1] = String.valueOf(alternatives[i][j]); // Вносимо значення по критеріям

        }

    }

String[][] tableData2 = new String[alternatives[0].length][alternatives.length + 1]; // Додаємо 1 стовбець для номерів варіантів

for (int j = 0; j < tableData2.length; j++) {

tableData2[j][0] = String.valueOf(j + 1); // Вносимо номери варіантів

for (int i = 0; i < tableData2[0].length - 1; i++) {

tableData2[j][i + 1] = String.valueOf(normalizedAlternatives[i][j]); // Вносимо значення по критеріям

        }

    }



String[][] tableDataPairComp = new String[pairCompAlternatives.length][criteriaNames.length + 1];

for (int i = 0; i < tableDataPairComp.length; i++) {

tableDataPairComp[i][0] = String.valueOf(pairCompAlternatives[i].index);
```

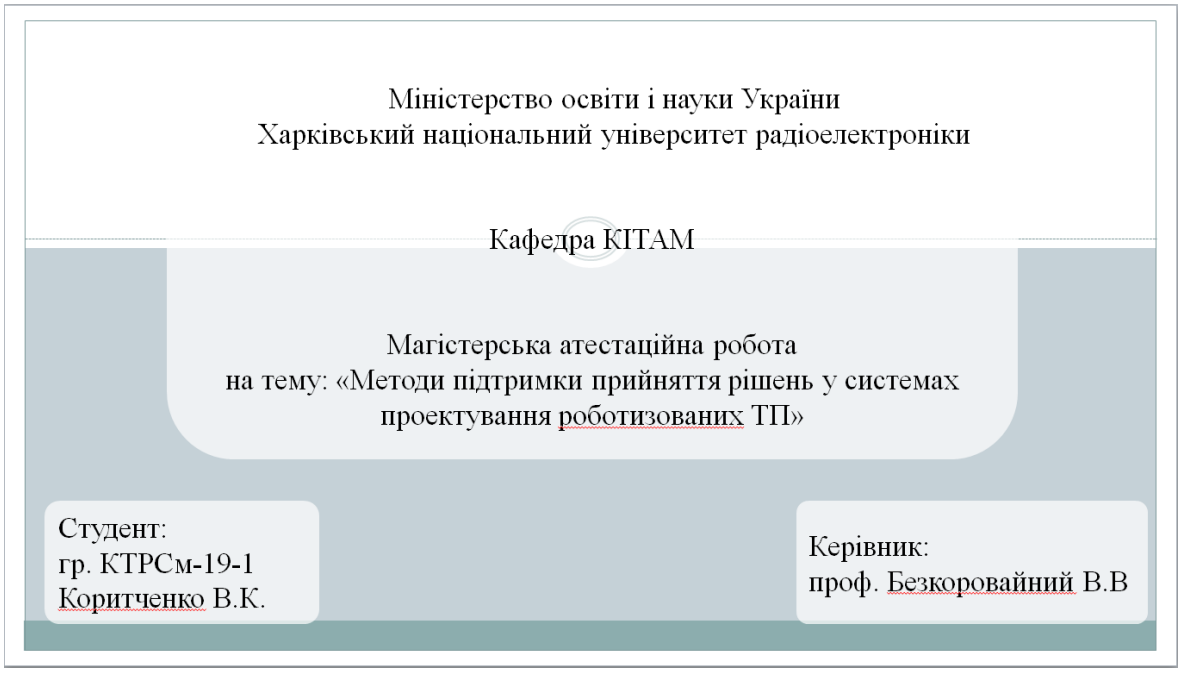
```
for (int j = 0; j < tableDataPairComp[j].length - 1; j++) {
tableDataPairComp[i][j+1] =
String.valueOf(pairCompAlternatives[i].criteriaValues[j]);
        }
    }
String[][] tableDataResult = new String[result.length][criteriaNames.length + 1];
for (int i = 0; i < tableDataResult.length; i++) {
tableDataResult[i][0] = String.valueOf(result[i].index);

for (int j = 0; j < tableDataResult[0].length - 1; j++) {
tableDataResult[i][j+1] = String.valueOf(result[i].criteriaValues[j]);
        }
    }

new MyFrame(tableData1,tableData2, criteriaNames,
bestAndWorstString,tableDataPairComp,tableDataResult);
}
};
```

ДОДАТОК Б

Демонстраційні матеріали

Слайд 1

ТИТУЛЬНИЙ ЛИСТ РОБОТИ



Слайд 2

ВСТУП

ВСТУП

Мета роботи – підвищення ефективності систем підтримки прийняття рішень в САПР за рахунок розробки методів виділення та скорочення підмножин ефективних варіантів побудови ТП.

Об'єкт дослідження – роботизовані технологічні процеси.

Предмет дослідження – методи багатокритеріального вибору у системах проектування роботизованих технологічних процесів.

Для досягнення мети необхідно виконати наступні завдання:

– провести огляд та аналіз сучасного стану проблеми підтримки прийняття рішень у системах проектування роботизованих ТП;

– проаналізувати особливості технологічних процесів виготовлення МЕМС-акселерометрів;

– проаналізувати математичне забезпечення системи підтримки прийняття проектних рішень;

– розробити діаграму прецедентів та діаграму класів для системи проектування роботизованого технологічного процесу;

– розробити програмне забезпечення для підвищення ефективності підтримки прийняття рішень з багатокритеріального вибору у системах проектування роботизованих технологічних процесів;

– провести аналіз, надати рекомендації щодо використання результатів роботи.

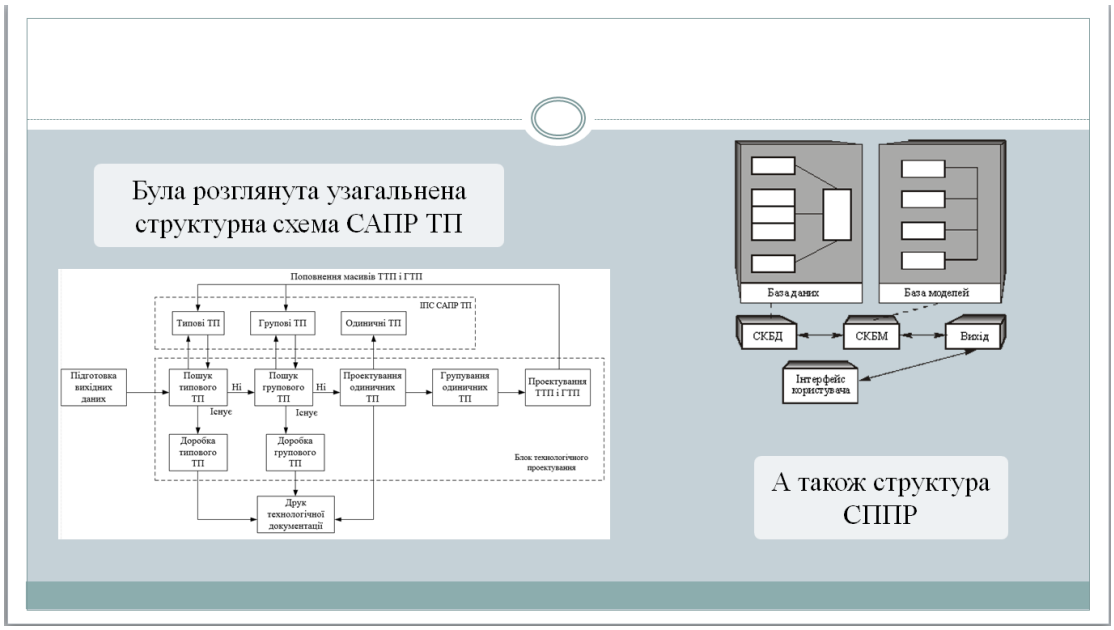
Слайд 3

ПРИКЛАДИ СУЧАСНИХ ВИРОБНИЧИХ РОБОТІВ

Приклади сучасних виробничих роботів

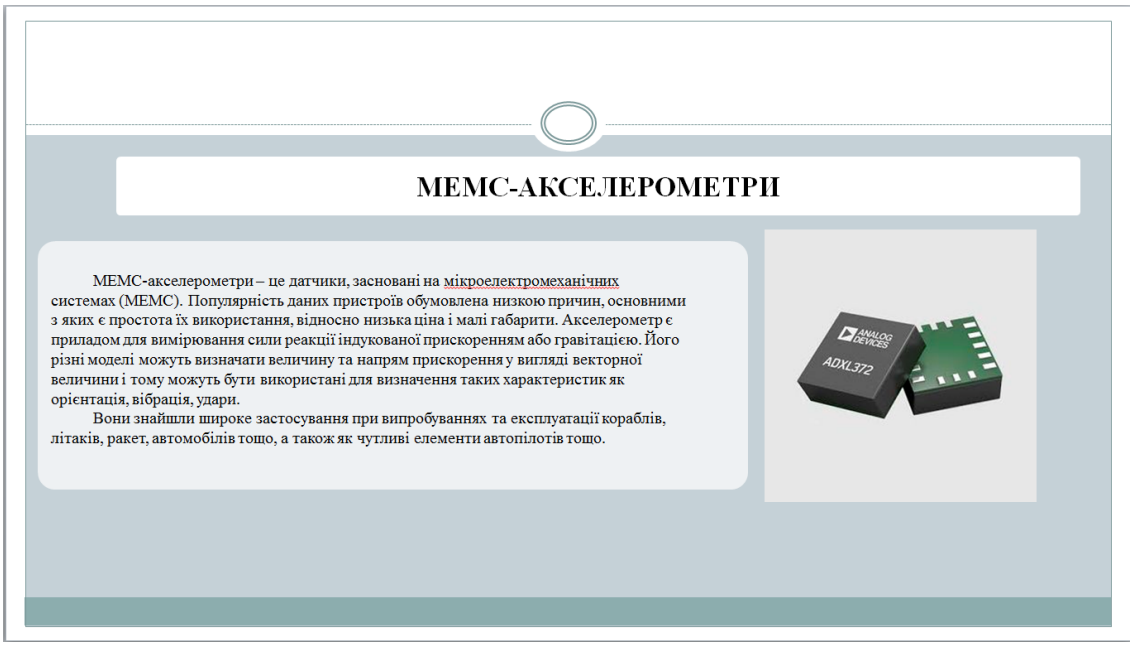
Слайд 4

СТРУКТУРИ САПР ТП І СППР



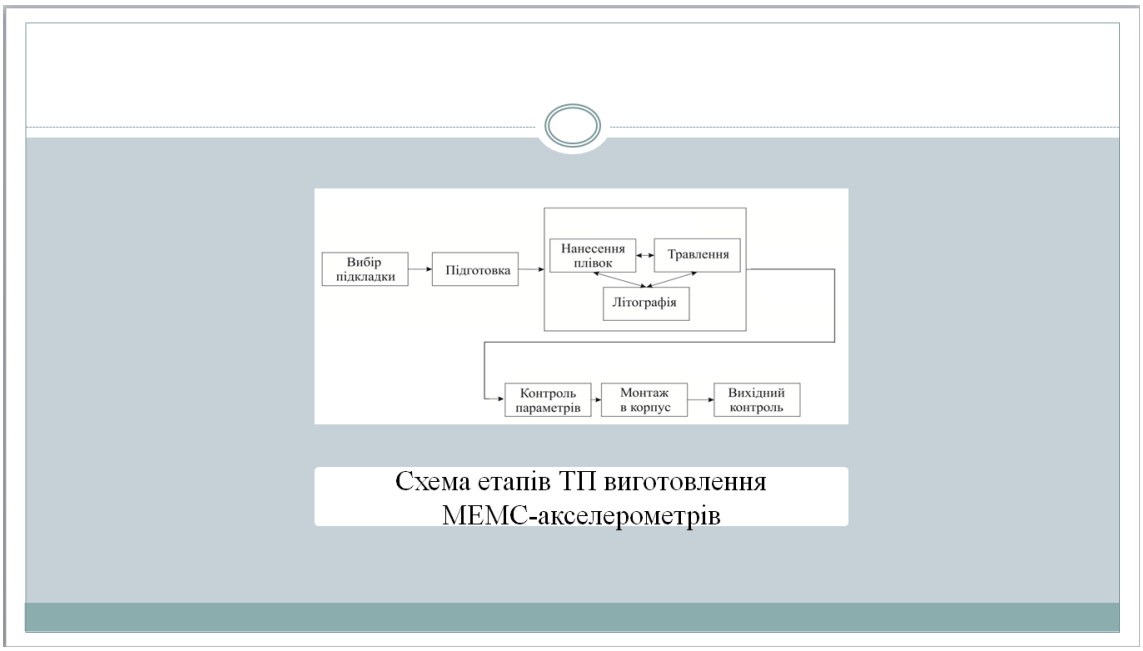
Слайд 5

МЕМС-АКСЕЛЕРОМЕТРИ



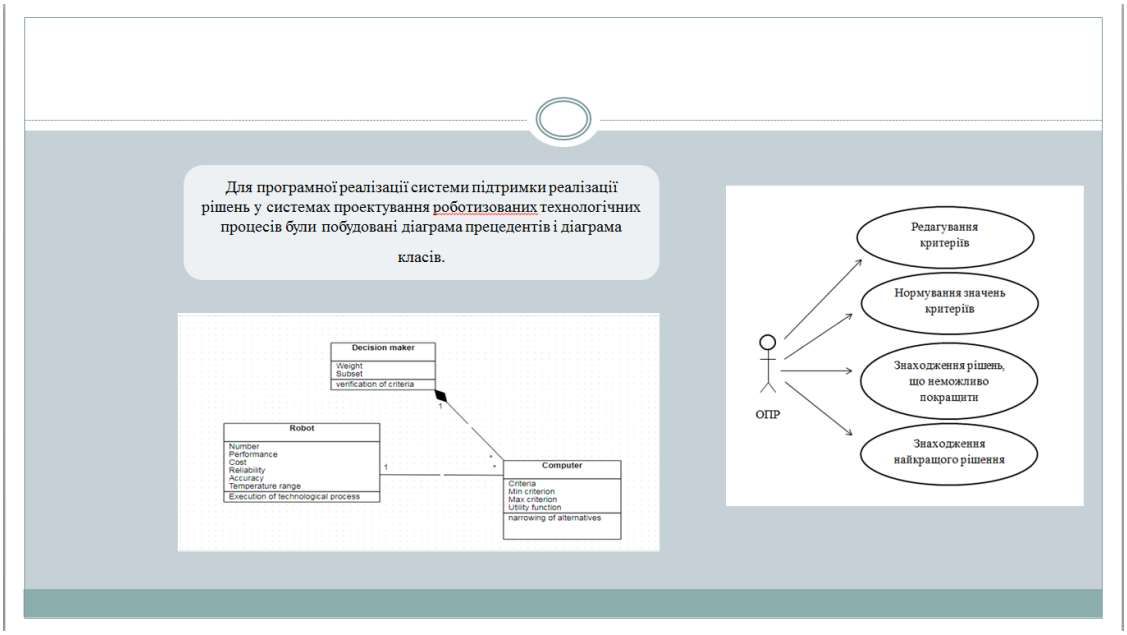
Слайд 6

СХЕМА ЕТАПІВ ТП ВИГОТОВЛЕННЯ МЕМС-АКСЕЛЕРОМЕТРІВ



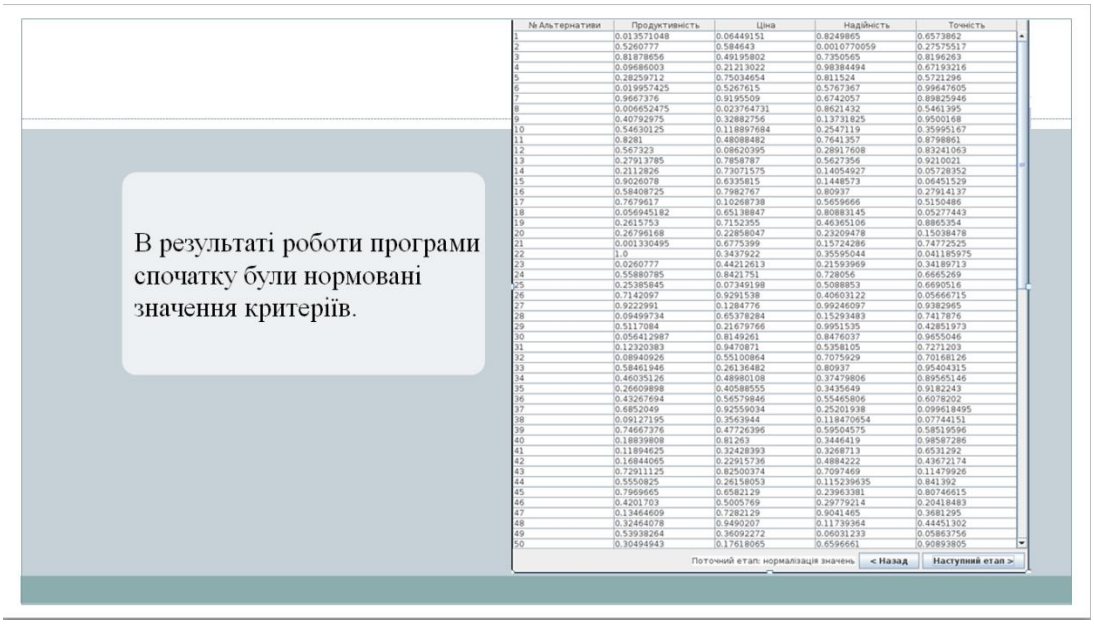
Слайд 7

ДІАГРАМА ПРЕЦЕДЕНТІВ І ДІАГРАМА КЛАСІВ



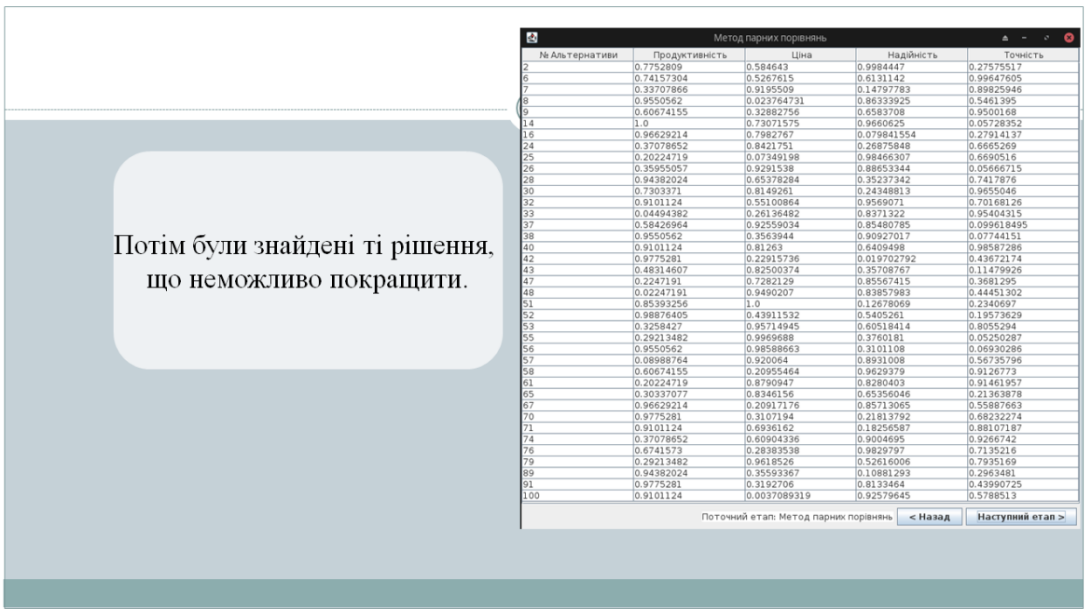
Слайд 8

РЕЗУЛЬТАТИ ПЕРШОГО ЕТАПУ РОБОТИ ПРОГРАМИ



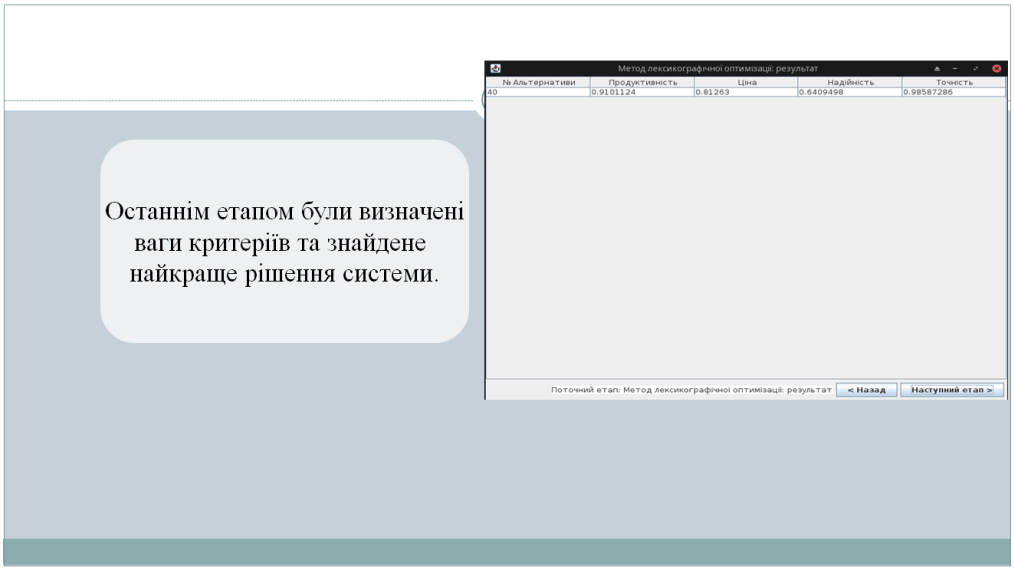
Слайд 9

РЕЗУЛЬТАТИ ДРУГОГО ЕТАПУ РОБОТИ ПРОГРАМИ



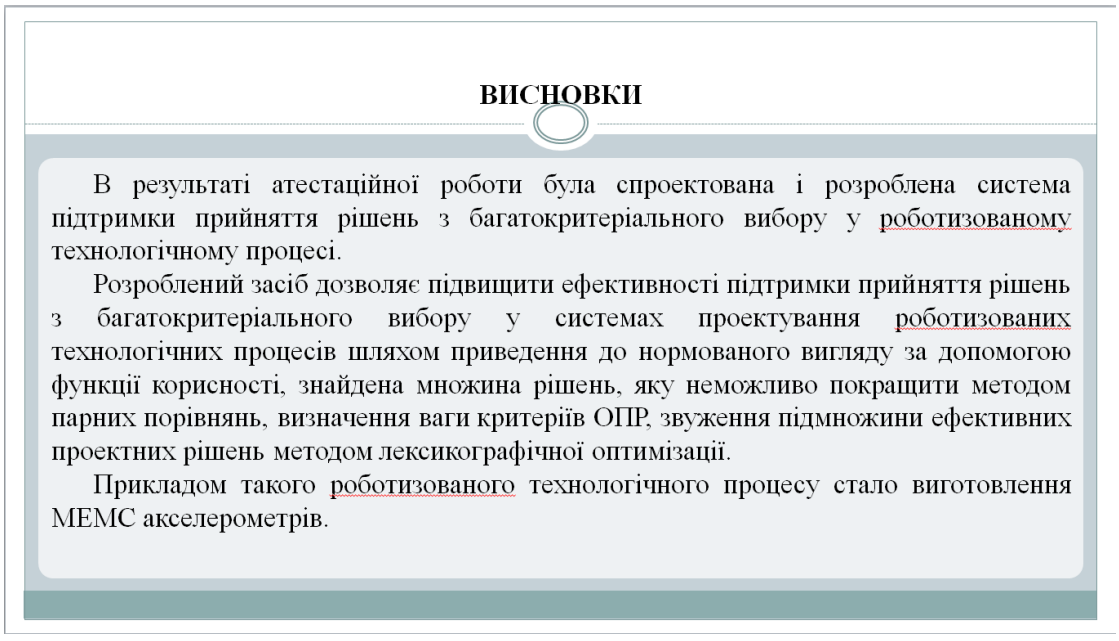
Слайд 10

РЕЗУЛЬТАТИ ТРЕТЬОГО ЕТАПУ РОБОТИ ПРОГРАМИ



Слайд 11

ВИСНОВКИ



ДОДАТОК В

Відомість атестаційної роботи

№	Позначення	Найменування	Дод. відомості
		Текстові документи	
1	ГЮИК.504300.004 ПЗ	Пояснювальна записка	116 с., ф. А4
		Додаткові матеріали	
2		Демонстраційні матеріали	7 с. у форматі *.ppt

Зм.	Арк.	№ докум.	Підпис	Дата	ГЮИК.504300.004 ВД			
Розробив		Коритченко В.К.			Методи підтримки прийняття рішень у системах проектування роботизованих ТП Відомість атестаційної роботи	Літера	Аркуш	Аркушів
Перевірив		Безкоровайний В.В.						1
						ХНУРЕ кафедра КІТАМ		
Н.контроль		Яшков І.О.						
Затвердив		Невлюдов І.Ш.						