Kauno technologijos universitetas
Informatikos fakultetas

Objektinis programavimas 2 (P175B123)

Laboratorinių darbų ataskaita

Laura Capaitė IF-0/1

Studentas

Doc. Svajūnas Sajavičius

Dėstytojas

Kaunas 2022

TURINYS

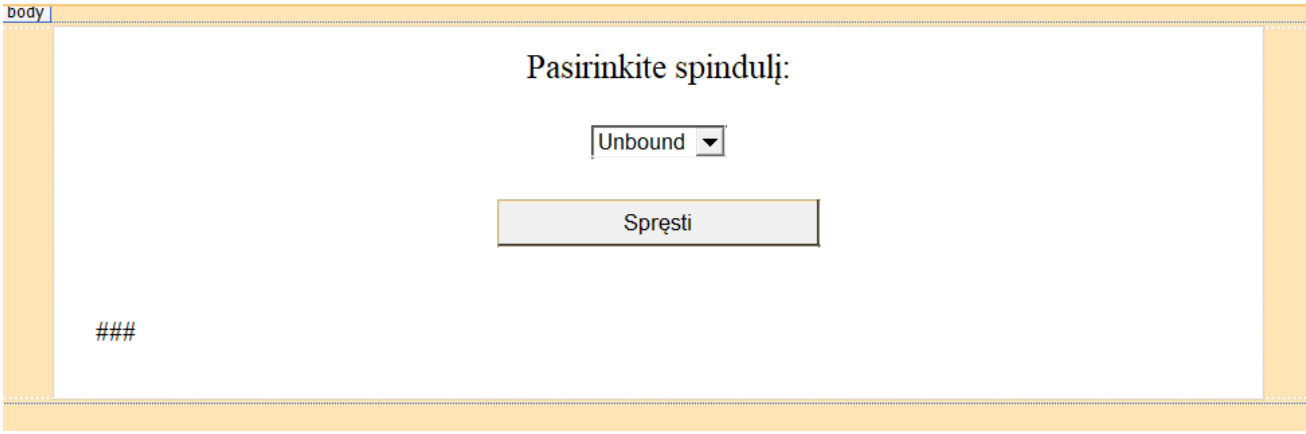
1. Rekursija (L1)

1.1. Darbo užduotis

Varianto numeris: 6

Užduotis: nubrėžiamas ant lapo apskritimas su pasirinkta spindulio reikšme (1<= r <= 16). Reikia sunumeruoti pilnai patenkančius langelius į apskritimą pradedant nuo 1. Rezultatas atvaizduojamas matrica.

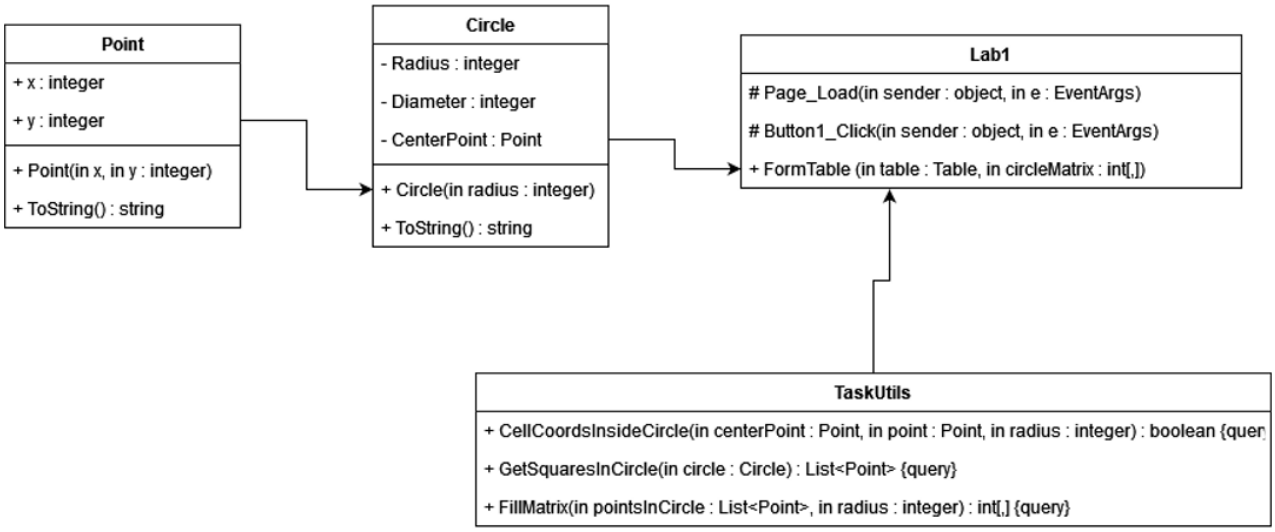
1.2. Grafinės vartotojo sąsajos schema



1.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
Label	Text	„Pasirinkite spindulį:“
	FontSize	25px
Button	Text	„Spręsti“
	BorderColor	Moccasin
	Height	35px
	Width	240px
Table		
DropDownList	Height	40px
	Width	100px

1.4. Klasių diagrama



1.5. Programos vartotojo vadovas

Programos valdymas yra nesudėtingas. Vartotojas turi atlikti tik 2 veikmus, kad pamatytų rezultatą. Pirmiausia vartotojas turi pasirinkti norimą apskritimo spindulį ir tada belieka paspausti mygtuką „Skaičiuoti“ ir matrica su rezultatu bus atspauzdinta žemiau.

Vartotojas gali pasirinkti spindulį nuo 1 iki 16. Bet kuriuo atveju matrica bus atspausdinta ir langeliai sunumeruoti.

1.6. Programos tekstas

```
/// <summary>
/// Circle object
/// </summary>
public class Circle
{
    public int Radius { get; private set; }
    public int Diameter { get; private set; }
    public Point CenterPoint { get; private set; }

    /// <summary>
    /// Constructor of circle
    /// </summary>
    /// <param name="radius">circle's radius</param>
    public Circle(int radius)
    {
        this.Radius = radius;
        this.Diameter = 2 * radius;
        this.CenterPoint = new Point(radius, radius);
    }

    /// <summary>
    /// Overrides ToString, to create formated line
    /// </summary>
    /// <returns>formated line</returns>
```

```
        public override string ToString()
        {
            return String.Format("Radius: {0}, center point: {1}", Radius,
CenterPoint);
        }
    }



    /// <summary>
    /// Point object with coordinates
    /// </summary>
    public class Point
    {
        public int x { get; set; }
        public int y { get; set; }

        /// <summary>
        /// Constructor for point object
        /// </summary>
        /// <param name="x">x coordinate</param>
        /// <param name="y">y coordinate</param>
        public Point(int x, int y)
        {
            this.x = x;
            this.y = y;
        }

        /// <summary>
        /// Overrides ToString, to create formated line
        /// </summary>
        /// <returns>formated line</returns>
        public override string ToString()
        {
            return String.Format("[{0},{1}]", x, y);
        }
    }



    /// <summary>
    /// Class for counting
    /// </summary>
    public static class TaskUtils
    {
        /// <summary>
        /// Checks if cell is inside the circle
        /// </summary>
        /// <param name="centerPoint">circle's center point</param>
        /// <param name="point">current point</param>
        /// <param name="radius">circle radius</param>
        /// <returns></returns>
        public static bool cellCoordsInsideCircle(Point centerPoint, Point point,
int radius)
        {
            /// <summary>
            /// Local function to check if specific point is in circle
            /// </summary>
            /// <param name="centerPoint1">circle's center point</param>
            /// <param name="point1">current point</param>
            /// <param name="radius1">circle radius</param>
            /// <returns>true if distance is equal or less than radius, otherwise
false</returns>
            bool pointIsInsideCircle(Point centerPoint1, Point point1, int
radius1)
```

```
            {
                int dx = centerPoint1.x - point1.x;
                int dy = centerPoint1.y - point1.y;
                double dist = Math.Sqrt(dx * dx + dy * dy);

                return dist <= radius1 ? true : false;
            }

            bool fullSquare = false;

            List<Point> coordsToCheck = new List<Point>();
            coordsToCheck.Add(point);
            coordsToCheck.Add(new Point(point.x, point.y + 1));
            coordsToCheck.Add(new Point(point.x + 1, point.y + 1));
            coordsToCheck.Add(new Point(point.x + 1, point.y));

            int pointToCheckIdx = coordsToCheck.Count - 1;

            recursiveCellCoordsInsideCircle(centerPoint, pointToCheckIdx, radius);

            /// <summary>
            /// Local recursive method to check if all 4 cell points are in circle
            /// </summary>
            /// <param name="centerPoint2">circle's center point</param>
            /// <param name="point2">current point</param>
            /// <param name="radius2">circle radius</param>
            void recursiveCellCoordsInsideCircle(Point centerPoint2, int
pointToCheckIdx2, int radius2)
            {

                if (!pointIsInsideCircle(centerPoint2,
coordsToCheck[pointToCheckIdx2], radius2))
                {
                    return;
                }

                if (pointToCheckIdx2 - 1 >= 0)
                {
                    recursiveCellCoordsInsideCircle(centerPoint2, pointToCheckIdx2
- 1, radius2);
                }
                else
                {
                    fullSquare = true;
                }

            }

            return fullSquare;
        }

        /// <summary>
        /// Gets list with point which are in circle
        /// </summary>
        /// <param name="circle">created object - circle</param>
        /// <returns>list with points</returns>
        public static List<Point> GetSquaresInCircle(Circle circle)
        {
            List<Point> pointsInCircle = new List<Point>();
            for (int i = 0; i < circle.Diameter; i++)
            {
                for (int j = 0; j < circle.Diameter; j++)
                {
                    Point point = new Point(i, j);
```

```
                    if(cellCoordsInsideCircle(circle.CenterPoint, point,
circle.Radius))
                    {
                        pointsInCircle.Add(point);
                    }
                }
            }
            return pointsInCircle;
        }

        /// <summary>
        /// Creates a matrix with points in circle (counts how many squares are in
the circle)
        /// </summary>
        /// <param name="pointsInCircle">list of points which are in
circle</param>
        /// <param name="radius">circle's radius</param>
        /// <returns>matrix with number of squares inside</returns>
        public static int[,] FillMatrix (List<Point> pointsInCircle, int radius)
        {
            int[,] circleMatrix = new int[radius * 2, radius * 2];
            for (int i = 0; i < pointsInCircle.Count; i++)
            {
                circleMatrix[pointsInCircle[i].x, pointsInCircle[i].y] = i+1;
            }

            return circleMatrix;
        }
    }
```

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="lab1.aspx.cs"
Inherits="OPlab1.lab1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="style.css" rel="stylesheet" />
</head>
<body>
    <form id="form1" runat="server">


        <div class="info">

            <div class="text">
            Pasirinkite spindulį:<br />
            <br />
            </div>

            <div class="DropDownList">
                <asp:DropDownList ID="DropDownList1" runat="server" Height="40px"
Width="100px">
                </asp:DropDownList>
            </div>

            <div class="button">
                <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
Text="Spręsti" Width="240px" BorderColor="Moccasin" Height="35px" />
            </div>

            <table class="table">
                <asp:Table ID="Table1" runat="server" Height="62px" Width="131px">
```

```
                </asp:Table>
            </table>
        </div>
    </form>
</body>
</html>
```

```
 /// <summary>
/// Main class
/// </summary>
public partial class lab1 : System.Web.UI.Page
    {
        /// <summary>
        /// Creates drop down list with numbers when page loads
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        protected void Page_Load(object sender, EventArgs e)
        {
            if (DropDownList1.Items.Count == 0)
            {
                DropDownList1.Items.Add("1");
                for (int i = 2; i <= 16; i++)
                {
                    DropDownList1.Items.Add(i.ToString());
                }
            }
        }

        /// <summary>
        /// Solves task with circles and prints matrix
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        protected void Button1_Click(object sender, EventArgs e)
        {
            int radius = int.Parse(DropDownList1.Text);

            Circle circle = new Circle(radius);
            List<Point> pointsInCircle = TaskUtils.GetSquaresInCircle(circle);

            int[,] circleMatrix = TaskUtils.FillMatrix(pointsInCircle, radius);
            FormTable(Table1, circleMatrix);

        }

        /// <summary>
        /// Forms the matrix
        /// </summary>
        /// <param name="table">table to print matrix</param>
        /// <param name="circleMatrix">matrix with squares in circle</param>
        public static void FormTable(Table table, int[,] circleMatrix)
        {
            int rowLength = circleMatrix.GetLength(0);
            int colLength = circleMatrix.GetLength(1);

            for (int i = 0; i < rowLength; i++)
            {
                TableRow row = new TableRow();
                for (int j = 0; j < colLength; j++)
                {
                    TableCell number = new TableCell();
```

```
                    number.Text = string.Format("{0, 2}", circleMatrix[i,
j].ToString());
                    row.Cells.Add(number);
                }
                table.Rows.Add(row);
            }
        }


    }
```

1.7. Pradiniai duomenys ir rezultatai

Kai r pasirenkamas 1, gaunama matrica:

```
0       0
0       0
```

Kai r pasirenkamas 5, gaunama matrica:

```
0 0  0  0  0  0  0  0  0  0
0 0  1  2  3  4  5  6  0  0
0 7  8  9  10 11 12 13 14 0
0 15 16 17 18 19 20 21 22 0
0 23 24 25 26 27 28 29 30 0
0 31 32 33 34 35 36 37 38 0
0 39 40 41 42 43 44 45 46 0
0 47 48 49 50 51 52 53 54 0
0 0  55 56 57 58 59 60 0  0
0 0  0  0  0  0  0  0  0  0
```

(Gaunasi taip pat kaip ir pavyzdį)

Kai r pasirenkamas 11, gaunama matrica:

```
0 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
0 0   0   0   0   0   0   1   2   3   4   5   6   7   8   0   0   0   0   0   0   0
0 0   0   0   0   9   10  11  12  13  14  15  16  17  18  19  20  0   0   0   0   0
0 0   0   0   21  22  23  24  25  26  27  28  29  30  31  32  33  34  0   0   0   0
0 0   0   35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  0   0   0
0 0   51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  0   0
0 0   69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  0   0
0 87  88  89  90  91  92  93  94  95  96  97  98  99  100 101 102 103 104 105 106 0
0 107 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 0
0 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 0
0 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 0
0 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 0
0 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 0
0 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 0
```

```
0 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 0
0 0   247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262 263 264 0   0
0 0   265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 0   0
0 0   0   283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 0   0   0
0 0   0   0   299 300 301 302 303 304 305 306 307 308 309 310 311 312 0   0   0   0
0 0   0   0   0   313 314 315 316 317 318 319 320 321 322 323 324 0   0   0   0   0
0 0   0   0   0   0   0   325 326 327 328 329 330 331 332 0   0   0   0   0   0   0
0 0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

1.8. Dėstytojo pastabos

P2, P5, P13, P15, P18

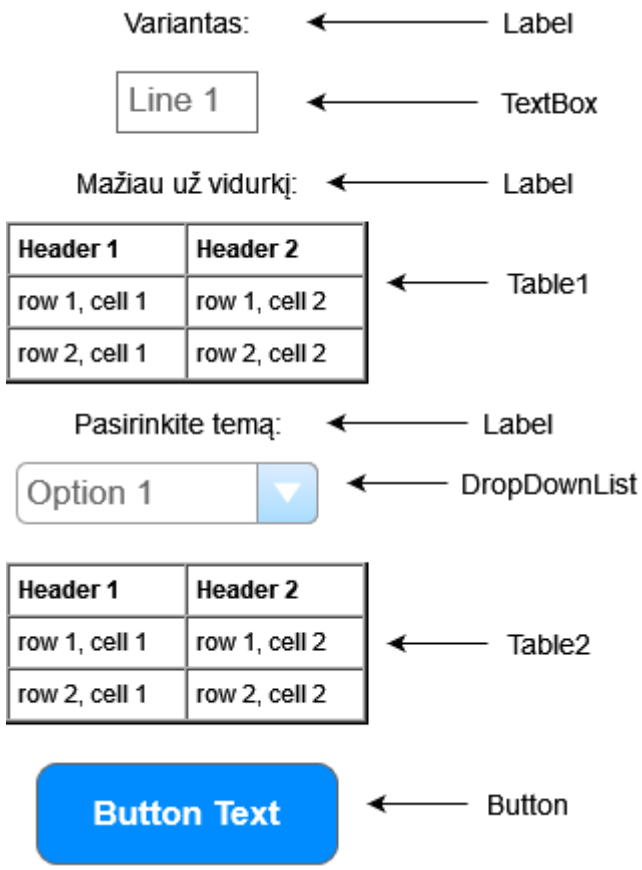
2. Dinaminis atminties valdymas (L2)

2.1. Darbo užduotis

Variantas: 6

Darbuotojai atliko 4 darbus ir jie visi gavo premijas. Remiantis darbuotojų indėliais, reikia paskaičiuoti kokia premijos suma priklauso kiekvienam bei bendrą premijų sumą. Svarbu neišdalinti daugiau pinigų nei turima. Reikia sudaryti sąrašą darbuotojų, kurie uždirbo mažiau už vidurkį. Sąrašas turi būti surikiuotas pagal darbuotojų pavardes, vardus abėcėlės tvarka ir bendrą premijų sumą. Taip pat sudaryti pasirinktos temos, kuri įvedama klaviatūra, darbuotojų sąrašą.

2.2. Grafinės vartotojo sąsajos schema

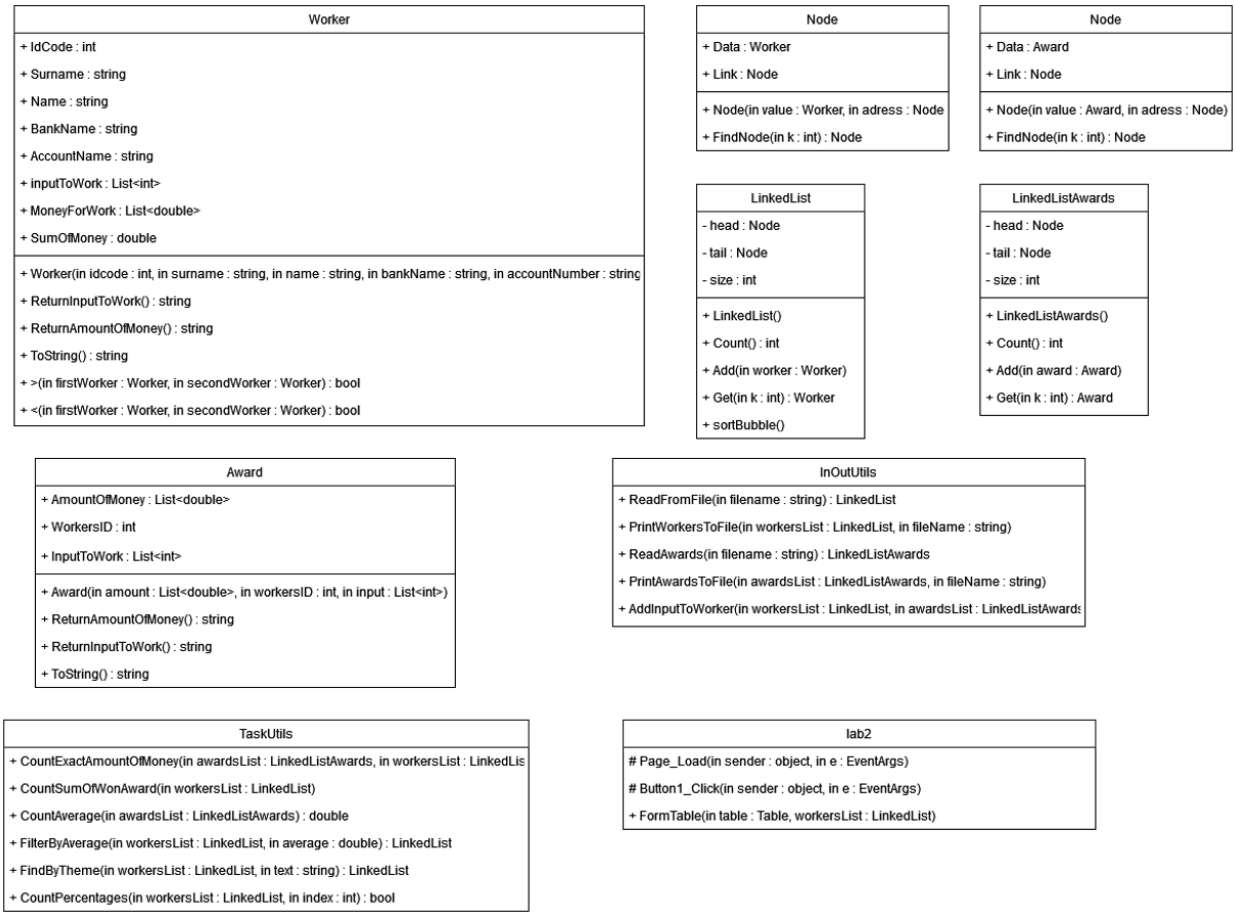


2.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
TextBox	Font	Larger
	ReadOnly	True
	Height	52px
	Width	72px
Table1	Height	193px
	Width	1048px
Table2	Height	268px

	Width	1030px
DropDownList	Height	55px
	Width	182px
	BackColor	White
	Font	Larger
Label	Text	Vidurkis:
Label	Text	Mažiau už vidurkį:
Label	Text	Pasirinkite norimą temą:
Button	Text	Spręsti
	Font	Larger
	Height	72px
	Width	234px

2.4. Klasių diagrama



2.5. Programos vartotojo vadovas

Į pirmą tekstinį failą įkeliama informacija apie darbuotojus(jų informacija), o į antrą – apie premijas informacija:

- Apie darbuotojus: asmens kodas, pavardė, vardas, banko pavadinims ir sąskaitos numeris
- Apie premijas: pirmoje eilutėje premijų sumos, kitose asmens kodas ir indėliai į darbus pagal temas (nurodoma procentais, negali indėlių suma vienai temai viršyti ar būti žemiau 100%)

Paleidus programą belieka paspausti mygtuką „Spręsti“ ir ekrane bus parodoma informacija apie tuos darbuotojus, kurie uždirbo mažiau nei vidurkis. Taip pat virš lentelės bus galima matyti ir vidurkį. Galima taip pat surasti darbuotojus pagal temą. Bus išvedama lentelė, kurie darbuotojai prisidėjo prie pasirinktos temos. Taip pat rašys visus duomenis apie juos.

Kartais prie gautų sumų už temas galima pastebėti 0. Tai gali reišti, kad darbuotojas neprisidėjo prie projekto arba buvo blogai nurodytas indėlis į darbą.

2.6. Programos tekstas

```
/// <summary>
    /// Award object
    /// </summary>
    public class Award
    {
        public List<double> AmountOfMoney { get; set; }
        public int WorkersID { get; set; }
        public List<int> InputToWork { get; set; }

        /// <summary>
        /// Object constructor
        /// </summary>
        /// <param name="amount">Amount of money for topics</param>
        /// <param name="workersID">worker's ID number</param>
        /// <param name="input">worker's input to work</param>
        public Award(List<double> amount,int workersID, List<int> input)
        {
            this.AmountOfMoney = amount;
            this.WorkersID = workersID;
            this.InputToWork = input;
        }

        /// <summary>
        /// Creates formated line of Amount of money
        /// </summary>
        /// <returns>Formated line</returns>
        public string ReturnAmountOfMoney()
        {
            string line = "";
            for (int i = 0; i < AmountOfMoney.Count; i++)
            {
                line += AmountOfMoney[i] + " ";
            }
            return line;
        }

        /// <summary>
        /// Creates formated line of Input to work
        /// </summary>
        /// <returns>Formated line</returns>
```

```
        public string ReturnInputToWork()
        {
            string line = "";
            for (int i = 0; i < InputToWork.Count; i++)
            {
                line += InputToWork[i] + " ";
            }
            return line;
        }

        /// <summary>
        /// Override ToString methos
        /// </summary>
        /// <returns>Formated line</returns>
        public override string ToString()
        {
            return string.Format("| {0, -12} | {1, -44} | ",
                this.WorkersID, ReturnInputToWork());
        }
    }


/// <summary>
    /// Linked list class
    /// </summary>
    public class LinkedListAwards
    {
        /// <summary>
        /// Private Node class
        /// </summary>
        private class Node
        {
            public Award Data { get; set; }
            public Node Link { get; set; }

            /// <summary>
            /// Node constructor
            /// </summary>
            /// <param name="value">Node's value</param>
            /// <param name="address">Shows to next node</param>
            public Node(Award value, Node address)
            {
                this.Data = value;
                this.Link = address;
            }

            /// <summary>
            /// Finds necessary node by index
            /// </summary>
            /// <param name="k">Index</param>
            /// <returns>Returns node</returns>
            public Node FindNode(int k)
            {
                Node data = this;
                for (int i = 0; i < k; i++)
                {
                    data = data.Link;
                }
                return data;
            }
        }

        private Node head;
        private Node tail;
        private int size;
```

```
        /// <summary>
        /// Linked list award constructor
        /// </summary>
        public LinkedListAwards()
        {
            this.head = null;
            this.tail = null;
            this.size = 0;
        }

        /// <summary>
        /// Counts size of linked list
        /// </summary>
        /// <returns>Size of linked list</returns>
        public int Count()
        {
            return this.size;
        }

        /// <summary>
        /// Add object to list
        /// </summary>
        /// <param name="award">An object</param>
        public void Add(Award award)
        {
            if (this.head == null)
            {
                this.head = new Node(award, this.head);
                this.tail = this.head;
            }
            else
            {
                Node newNode = new Node(award, null);
                this.tail.Link = newNode;
                this.tail = newNode;
            }
            size++;
        }

        /// <summary>
        /// Gets a specific object
        /// </summary>
        /// <param name="k">Index</param>
        /// <returns>Returns award object</returns>
        public Award Get(int k)
        {
            if (k < 0 || k >= size)
            {
                return null;
            }
             Node current = this.head.FindNode(k);
            return current.Data;
        }
    }


/// <summary>
    /// Workers object
    /// </summary>
    public class Worker
    {
        public int IdCode { get; set; }
        public string Surname { get; set; }
        public string Name { get; set; }
```

```
        public string BankName { get; set; }
        public string AccountNumber { get; set; }

        public List<int> inputToWork { get; set; }
        public List<double> MoneyForWork { get; set; }
        public double SumOfMoney { get; set; }

        /// <summary>
        /// Worker constructor
        /// </summary>
        /// <param name="idcode">Worker's id code</param>
        /// <param name="surname">Worker's surname</param>
        /// <param name="name">Worker's name</param>
        /// <param name="bankName">Worker's bank name</param>
        /// <param name="accountNumber">Worker's account number</param>
        public Worker(int idcode, string surname, string name, string bankName,
string accountNumber)
        {
            this.IdCode = idcode;
            this.Surname = surname;
            this.Name = name;
            this.BankName = bankName;
            this.AccountNumber = accountNumber;
            this.inputToWork = new List<int>();
            this.MoneyForWork = new List<double>();
            this.SumOfMoney = 0;
        }

        /// <summary>
        /// Overrides ToString method
        /// </summary>
        /// <returns>Formated line</returns>
        public override string ToString()
        {
            return string.Format("| {0, -12} | {1, -13} | {2, -13} | {3, -10} |
{4, -15} | {5, -44} | {6, -17} | {7, -20} |",
                this.IdCode, this.Surname, this.Name, this.BankName,
this.AccountNumber, ReturnInputToWork(), ReturnMoneyToWork(), this.SumOfMoney);
        }

        /// <summary>
        /// Formates line with input to work data
        /// </summary>
        /// <returns>Formated line</returns>
        public string ReturnInputToWork()
        {
            string line = "";
            for (int i = 0; i < inputToWork.Count; i++)
            {
                line += inputToWork[i] + " ";
            }
            return line;
        }

        /// <summary>
        /// Formates line with money to work
        /// </summary>
        /// <returns>Formated line</returns>
        public string ReturnMoneyToWork()
        {
            string line = "";
            for (int i = 0; i < MoneyForWork.Count; i++)
            {
                line += MoneyForWork[i] + " ";
            }
```

```
            return line;
        }

        /// <summary>
        /// Operator to compare
        /// </summary>
        /// <param name="firstWorker">First worker to compare</param>
        /// <param name="secondWorker">Second worker to compare</param>
        /// <returns>True if first worker is higher, otherwise false</returns>
        public static bool operator >(Worker firstWorker, Worker secondWorker)
        {
            if (firstWorker.Surname.CompareTo(secondWorker.Surname) == 0)
            {
                if (firstWorker.Name.CompareTo(secondWorker.Name) == 0)
                {
                    return firstWorker.SumOfMoney < secondWorker.SumOfMoney;
                }
                else
                {
                    return firstWorker.Name.CompareTo(secondWorker.Name) > 0;
                }
            }
            else
            {
                return firstWorker.Surname.CompareTo(secondWorker.Surname) > 0;
            }
        }


        /// <summary>
        /// Operator to compare
        /// </summary>
        /// <param name="firstWorker">First worker to compare</param>
        /// <param name="secondWorker">Second worker to compare</param>
        /// <returns>True if second worker is higher, otherwise false</returns>
        public static bool operator <(Worker firstWorker, Worker secondWorker)
        {
            if (firstWorker.Surname.CompareTo(secondWorker.Surname) == 0)
            {
                if (firstWorker.Name.CompareTo(secondWorker.Name) == 0)
                {
                    return firstWorker.SumOfMoney < secondWorker.SumOfMoney;
                }
                else
                {
                    return firstWorker.Name.CompareTo(secondWorker.Name) < 0;
                }
            }
            else
            {
                return firstWorker.Surname.CompareTo(secondWorker.Surname) < 0;
            }
        }



/// <summary>
    /// Linked list class
    /// </summary>
    public class LinkedList
    {
        /// <summary>
        /// Private node class
        /// </summary>
        private class Node
```

```
{
    public Worker Data { get; set; }
    public Node Link { get; set; }

    /// <summary>
    /// Node constructor
    /// </summary>
    /// <param name="value">Node's value</param>
    /// <param name="address">Shows to next node</param>
    public Node(Worker value, Node address)
    {
        this.Data = value;
        this.Link = address;
    }

    /// <summary>
    /// Finds necessary node by index
    /// </summary>
    /// <param name="k">Index</param>
    /// <returns>Returns node</returns>
    public Node FindNode(int k)
    {
        Node data = this;
        for (int i = 0; i < k; i++)
        {
            data = data.Link;
        }
        return data;
    }
}

private Node head;
private Node tail;
private int size;

/// <summary>
/// Linked list constructor
/// </summary>
public LinkedList()
{
    this.head = null;
    this.tail = null;
    this.size = 0;
}

/// <summary>
/// Counts size of linked list
/// </summary>
/// <returns>Size of linked list</returns>
public int Count()
{
    return this.size;
}

/// <summary>
/// Add object to list
/// </summary>
/// <param name="worker">An object</param>
public void Add(Worker worker)
{
    if (this.head == null)
    {
        this.head = new Node(worker, this.head);
        this.tail = this.head;
    }
```

```
            else
            {
                Node newNode = new Node(worker, null);
                this.tail.Link = newNode;
                this.tail = newNode;
            }
            size++;
        }

        /// <summary>
        /// Gets a specific object
        /// </summary>
        /// <param name="k">Index</param>
        /// <returns>Returns worker object</returns>
        public Worker Get(int k)
        {
            if (k < 0 || k >= size)
            {
                return null;
            }
             Node current = this.head.FindNode(k);
            return current.Data;
        }

        /// <summary>
        /// Method to sort list
        /// </summary>
        public void sortBubble()
        {
            if (this.head == null)
            {
                return;
            }
            for (Node d = head; d != null; d = d.Link)
            {
                bool sorted = true;
                Node element = this.head;
                for (Node d2 = this.head.Link; d2 != null; d2 = d2.Link)
                {
                    if (element.Data > d2.Data)
                    {
                        Worker worker = element.Data;
                        element.Data = d2.Data;
                        d2.Data = worker;
                        sorted = false;
                    }
                    element = d2;
                }
                if (sorted)
                {
                    return;
                }
            }
        }
    }



/// <summary>
    /// Class to read/print to file
    /// </summary>
    public static class InOutUtils
    {
        /// <summary>
        /// Reads info about workers from file
```

```
        /// </summary>
        /// <param name="filename">File name</param>
        /// <returns>Linked list with workers</returns>
        public static LinkedList ReadFromFile(string filename)
        {
            LinkedList workerList = new LinkedList();
            string[] lines = File.ReadAllLines(filename, Encoding.UTF8);
            for (int i = 0; i < lines.Length; i++)
            {
                string[] parts = lines[i].Split(',');
                int idCode = int.Parse(parts[0]);
                string surname = parts[1];
                string name = parts[2];
                string bankName = parts[3];
                string bankNumer = parts[4];
                Worker worker = new Worker(idCode, surname, name, bankName,
bankNumer);
                workerList.Add(worker);
            }

            return workerList;
        }

        /// <summary>
        /// Print workers info to file
        /// </summary>
        /// <param name="workersList">Linked list with workers</param>
        /// <param name="fileName">File name</param>
        /// <param name="header">Table's header</param>
        public static void PrintWorkersToFile(LinkedList workersList, string
fileName, string header)
        {
            string[] lines = new string[workersList.Count() + 10];
            lines[0] = header;
            lines[1] = new string('-', 169);
            lines[2] = String.Format("| {0, -12} | {1, -13} | {2, -13} | {3, -10}
| {4, -15} | {5, -44} | {6, -17} | {7, -20} |",
                "Asmens kodas", "Pavardė", "Vardas", "Bankas", "Sąskaitos nr.",
"Indėlis į darbą (gamta, fizika, chemija, IT)", "Atskiros sumos", "Gauta bendra
suma");
            lines[3] = new string('-', 169);
            for (int i = 0; i < workersList.Count(); i++)
            {
                lines[4 + i] = workersList.Get(i).ToString();
            }
            lines[workersList.Count() + 4] = new string('-', 169);
            File.AppendAllLines(fileName, lines, Encoding.UTF8);
        }

        /// <summary>
        /// Reads awards info from file
        /// </summary>
        /// <param name="filename">File name</param>
        /// <returns>Linked list with awards</returns>
        public static LinkedListAwards ReadAwards(string filename)
        {
            LinkedListAwards awardsList = new LinkedListAwards();
            List<double> amountOfMoney = new List<double>();
            string[] lines = File.ReadAllLines(filename, Encoding.UTF8);
            string[] firstLine = lines[0].Split(',');
            for (int i = 0; i < firstLine.Length; i++)
            {
                amountOfMoney.Add(int.Parse(firstLine[i]));
            }
            for (int i = 1; i < lines.Length; i++)
```

```
            {
                string[] parts = lines[i].Split(',');
                int idCode = int.Parse(parts[0]);

                List<int> amountOfInput = new List<int>();
                amountOfInput.Add(int.Parse(parts[1]));
                amountOfInput.Add(int.Parse(parts[2]));
                amountOfInput.Add(int.Parse(parts[3]));
                amountOfInput.Add(int.Parse(parts[4]));

                Award award = new Award(amountOfMoney, idCode, amountOfInput);
                awardsList.Add(award);
            }

            return awardsList;
        }

        /// <summary>
        /// Print awards info to file
        /// </summary>
        /// <param name="awardsList">Linked list with awards</param>
        /// <param name="fileName">File name</param>
        /// <param name="header">Table's header</param>
        public static void PrintAwardsToFile(LinkedListAwards awardsList, string
fileName, string header)
        {
            string[] lines = new string[awardsList.Count() + 10];
            lines[0] = header;
            lines[1] = new string('-', 63);
            lines[2] = "Premijų sumos(gamta, fizika, chemija, IT):";
            lines[3] = new string('-', 63);
            lines[4] = awardsList.Get(0).ReturnAmountOfMoney();
            lines[5] = new string('-', 63);
            lines[6] = String.Format("| {0, -12} | {1, -44} | ",
                "Asmens kodas", "Indėlis į darbą (gamta, fizika, chemija, IT)");
            lines[7] = new string('-', 63);
            for (int i = 0; i < awardsList.Count(); i++)
            {
                lines[7 + i] = awardsList.Get(i).ToString();
            }
            lines[awardsList.Count() + 7] = new string('-', 63);
            File.AppendAllLines(fileName, lines, Encoding.UTF8);
        }

        /// <summary>
        /// Add for worker information about their input to work
        /// </summary>
        /// <param name="workersList">Linked list with workers</param>
        /// <param name="awardsList">Linked list with awards</param>
        public static void AddInputToWorker(LinkedList workersList,
LinkedListAwards awardsList)
        {
            for (int i = 0; i < awardsList.Count(); i++)
            {
                for (int j = 0; j < workersList.Count(); j++)
                {
                    Award award = awardsList.Get(i);
                    Worker worker = workersList.Get(i);
                    if (award.WorkersID == worker.IdCode)
                    {
                        worker.inputToWork = award.InputToWork;
                    }
                }
            }
        }
```

```
    }


public static class TaskUtils
    {
        /// <summary>
        /// Method to count how much each worker got money for every topic
        /// </summary>
        /// <param name="awardsList">Linked list with awards</param>
        /// <param name="workersList">Linked list with workers</param>
        public static void CountExactAmountOfMoney(LinkedListAwards awardsList,
LinkedList workersList)
        {
            List<double> amountOfMoney = awardsList.Get(0).AmountOfMoney;
            for (int i = 0; i < amountOfMoney.Count(); i++)
            {
                List<double> money = new List<double>();
                for (int j = 0; j < workersList.Count(); j++)
                {
                    double countedMoney = 0;
                    List<int> inputOfWork = workersList.Get(j).inputToWork;
                    if (CountPercentages(workersList, i)) {
                        double first = amountOfMoney[i];
                        double second = inputOfWork[i] * 0.01;
                        countedMoney = first * second;
                    }
                    workersList.Get(j).MoneyForWork.Add(countedMoney);
                }
            }
        }

        /// <summary>
        /// Method to count overall rewards sum
        /// </summary>
        /// <param name="workersList">Linked list with worker</param>
        public static void CountSumOfWonAward(LinkedList workersList)
        {
            for (int i = 0; i < workersList.Count(); i++)
            {
                double sum = 0;
                List<double> awards = workersList.Get(i).MoneyForWork;
                for (int j = 0; j < awards.Count; j++)
                {
                    sum += awards[j];
                }
                workersList.Get(i).SumOfMoney = sum;
            }
        }

        /// <summary>
        /// Counts average amount of won money
        /// </summary>
        /// <param name="awardsList"></param>
        /// <returns></returns>
        public static double CountAverage(LinkedListAwards awardsList)
        {
            double average = 0;
            double sum = 0;
            List<double> awards = awardsList.Get(0).AmountOfMoney;
            for (int i = 0; i < awards.Count; i++)
            {
                sum += awards[i];
            }
```

```
            average = sum / awards.Count;
            return average;
        }

        /// <summary>
        /// Method which finds workers who got less than average
        /// </summary>
        /// <param name="workersList">Linked list with workers</param>
        /// <param name="average">Average amount of money</param>
        /// <returns>Linked list with filtered workers</returns>
        public static LinkedList FilterByAverage(LinkedList workersList, double
average)
        {
            LinkedList lessThanAverage = new LinkedList();
            for (int i = 0; i < workersList.Count(); i++)
            {
                double money = workersList.Get(i).SumOfMoney;
                if (money < average)
                {
                    lessThanAverage.Add(workersList.Get(i));
                }
            }
            return lessThanAverage;
        }

        /// <summary>
        /// Method which finds workers who worked on specific topic
        /// </summary>
        /// <param name="workersList">Linked list with workers</param>
        /// <param name="text">Name of theme</param>
        /// <returns>Linked list with workers who worked on specific
topic</returns>
        public static LinkedList FindByTheme (LinkedList workersList, string text)
        {
            Dictionary<string, int> themes = new Dictionary<string, int>();
            themes.Add("GAMTA", 0);
            themes.Add("FIZIKA", 1);
            themes.Add("CHEMIJA", 2);
            themes.Add("IT", 3);

            LinkedList searchedByTheme = new LinkedList();
            for (int i = 0; i < workersList.Count(); i++)
            {
                List<int> inputToWork = workersList.Get(i).inputToWork;
                if (inputToWork[themes[text]] > 0)
                {
                    searchedByTheme.Add(workersList.Get(i));
                }
            }
            return searchedByTheme;
        }

        /// <summary>
        /// Method to check if info about amount of input was given correctly
        /// </summary>
        /// <param name="workersList">Linked list with workers</param>
        /// <param name="index">Index of theme</param>
        /// <returns>True if 100%, otherwise false</returns>
        public static bool CountPercentages(LinkedList workersList, int index)
        {
            int percentage = 0;
            for (int i = 0; i < workersList.Count(); i++)
            {
                List<int> inputToWork = workersList.Get(i).inputToWork;
                if (inputToWork[index] > 0)
```

```
                {
                    percentage += inputToWork[index];
                }
            }
            if(percentage == 100)
            {
                return true;
            }
            return false;
        }

    }
```

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="lab2.aspx.cs"
Inherits="OPlab2.lab2" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <link href="style.css" rel="stylesheet" />
</head>
<body>
    <form id="form1" runat="server">
        <div class="info">
            <div class="text">
                Vidurkis:
            </div>
            <asp:TextBox ID="TextBox1" runat="server" Width="72px" Font-
Size="Larger" Height="52px" ReadOnly="True"></asp:TextBox>
            <br />
            <br />
            <div class="text">
                Mažiau už vidurkį:<br />
            </div>
            <asp:Table ID="Table1" runat="server" Height="193px" Width="1048px">
            </asp:Table>
            <br/>
            <div class="text">
                Pasirinkite norimą temą:<br />
            </div>
            <br />
            <asp:DropDownList ID="DropDownList1" runat="server" Height="55px"
Width="182px" Font-Size="Larger" BackColor="White">
            </asp:DropDownList>
            <br />
            <br />
            <asp:Table ID="Table2" runat="server" Height="268px" Width="1030px">
            </asp:Table>
             <div class="text2">
                (Jei ketvirtame stulpelyje matomi nuliai, tai reiškia, kad
darbuotojas neprisidėjo prie darbo arba buvo nekorektiškai nurodytas indėlis į
darbą)<br />
            </div>

            <br />
            <br />
            <asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
Text="Spręsti" Height="72px" Width="234px" Font-Size="Larger" />
            <br />
        </div>
    </form>
</body>
```

```
</html>


/// <summary>
    /// Main class
    /// </summary>
    public partial class lab2 : System.Web.UI.Page
    {
        /// <summary>
        /// Method which loads things when page is opened
        /// </summary>
        /// <param name="sender">sender</param>
        /// <param name="e">e event args</param>
        protected void Page_Load(object sender, EventArgs e)
        {
            if (DropDownList1.Items.Count == 0)
            {
                DropDownList1.Items.Add("GAMTA");
                DropDownList1.Items.Add("FIZIKA");
                DropDownList1.Items.Add("CHEMIJA");
                DropDownList1.Items.Add("IT");
            }
        }

        /// <summary>
        /// Main method to start all calculations
        /// </summary>
        /// <param name="sender">sender</param>
        /// <param name="e">e event args</param>
        protected void Button1_Click(object sender, EventArgs e)
        {
            if (File.Exists(Server.MapPath("Rezultatai.txt")))
            {
                File.Delete(Server.MapPath("Rezultatai.txt"));
            }
            LinkedList workersList =
InOutUtils.ReadFromFile(Server.MapPath("U6a.txt"));
            InOutUtils.PrintWorkersToFile(workersList,
Server.MapPath("Rezultatai.txt"), "Darbuotojų informacija");
            LinkedListAwards awardsList =
InOutUtils.ReadAwards(Server.MapPath("U6b.txt"));
            InOutUtils.PrintAwardsToFile(awardsList,
Server.MapPath("Rezultatai.txt"), "Premijų informacija");

            InOutUtils.AddInputToWorker(workersList, awardsList);
            TaskUtils.CountExactAmountOfMoney(awardsList, workersList);
            TaskUtils.CountSumOfWonAward(workersList);

            //InOutUtils.PrintWorkersToFile(workersList,
Server.MapPath("Rezultatai.txt"), "Darbuotojų informacija");

            double average = TaskUtils.CountAverage(awardsList);
            TextBox1.Text = average.ToString();

            LinkedList lessThanAverage = TaskUtils.FilterByAverage(workersList,
average);
            lessThanAverage.sortBubble();
            InOutUtils.PrintWorkersToFile(lessThanAverage,
Server.MapPath("Rezultatai.txt"), "Darbuotojai gavę mažiau nei vidurkis");

            FormTable(Table1, lessThanAverage);

            string text = DropDownList1.Text;
            LinkedList searchedByTheme = TaskUtils.FindByTheme(workersList, text);
            searchedByTheme.sortBubble();
```

```
            InOutUtils.PrintWorkersToFile(searchedByTheme,
Server.MapPath("Rezultatai.txt"), "Filtruoti pagal temą");

            FormTable(Table2, searchedByTheme);


        }

        /// <summary>
        /// Method to form table with data
        /// </summary>
        /// <param name="table">Table where info will be stored</param>
        /// <param name="workersList">Linked list with workers</param>
        public static void FormTable(Table table, LinkedList workersList)
        {
            TableRow row1 = new TableRow();
            TableCell surname1 = new TableCell();
            surname1.Text = "<b>Pavardė</b>";
            row1.Cells.Add(surname1);
            TableCell name1 = new TableCell();
            name1.Text = "<b>Vardas</b>";
            row1.Cells.Add(name1);
            TableCell input1 = new TableCell();
            input1.Text = "<b>Indėlis į darbą (gamta, fizika, chemija, IT)</b>";
            row1.Cells.Add(input1);
            TableCell moneyForWork1 = new TableCell();
            moneyForWork1.Text = "<b>Pinigai už darbą (gamta, fizika, chemija,
IT)</b>";
            row1.Cells.Add(moneyForWork1);
            TableCell money1 = new TableCell();
            money1.Text = "<b>Galutinė suma</b>";
            row1.Cells.Add(money1);

            table.Rows.Add(row1);
            for (int i = 0; i < workersList.Count(); i++)
            {
                TableRow row = new TableRow();
                TableCell surname = new TableCell();
                surname.Text = workersList.Get(i).Surname;
                TableCell name = new TableCell();
                name.Text = workersList.Get(i).Name;
                TableCell input = new TableCell();
                input.Text = workersList.Get(i).ReturnInputToWork();
                TableCell moneyForWork = new TableCell();
                moneyForWork.Text = workersList.Get(i).ReturnMoneyToWork();
                TableCell money = new TableCell();
                money.Text = workersList.Get(i).SumOfMoney.ToString();


                row.Cells.Add(surname);
                row.Cells.Add(name);
                row.Cells.Add(input);
                row.Cells.Add(moneyForWork);
                row.Cells.Add(money);

                table.Rows.Add(row);
            }
        }
    }


body {
    background-color: rgb(255, 238, 194);
    font-size: 18px;
}
```

```
.info {
    text-align: center;
    margin: 0 auto;
    background-color: white;
    border-radius: 10px;
    padding: 15px 25px;
    padding-bottom: 400px;
    width: 80%;
    background-color: rgb(255, 252, 245);
}

#Table1 {
    margin: 0 auto;
    border: 1px solid #ddd;
    background-color: rgb(255, 248, 232);
    border-radius: 10px;
    padding: 15px;
}

#Table2 {
    margin: 0 auto;
    border: 1px solid #ddd;
    background-color: rgb(255, 248, 232);
    border-radius: 10px;
    padding: 15px;
}

.text{
    font-size: 32px;
}

.text2 {
    font-size: 17px;
}

#Button1 {
    border-radius: 10px;
    border-color: rgb(255, 238, 194);
    background-color: white;
}

    #Button1:hover {
        background-color: rgb(255, 238, 194);
        box-shadow: 0 0.5em 0.5em -0.4em rgb(255, 248, 232);
        transform: translateY(-0.25em);
        cursor: pointer;
    }
```

2.7. Pradiniai duomenys ir rezultatai

Pirmas variantas:

U6a.txt

```
56485,Zetraitis,Petras,Swed,LT4521
75896,Simaitis,Simas,Swed,LT2456
75899,Zetraitis,Petras,Swed,LT4521
```

U6b.txt

```
1000,500,700,900
56485,9,0,0,10
75896,80,100,100,80
75899,11,0,0,10
```

Rezultatai.txt

```
Darbuotojų informacija
-------------------------------------------------------------------------------------------
| Asmens kodas | Pavardė       | Vardas        | Bankas     | Sąskaitos nr.   |
-------------------------------------------------------------------------------------------
| 56485        | Zetraitis     | Petras        | Swed       | LT4521          |
| 75896        | Simaitis      | Simas         | Swed       | LT2456          |
| 75899        | Zetraitis     | Petras        | Swed       | LT4521          |
-------------------------------------------------------------------------------------------

Premijų informacija
-----------------------------------------------------------------
Premijų sumos(gamta, fizika, chemija, IT):
-----------------------------------------------------------------
1000 500 700 900
-----------------------------------------------------------------
| Asmens kodas | Indėlis į darbą (gamta, fizika, chemija, IT) |
| 56485        | 9 0 0 10                                     |
| 75896        | 80 100 100 80                                |
| 75899        | 11 0 0 10                                    |
-----------------------------------------------------------------

Darbuotojai gavę mažiau nei vidurkis
-------------------------------------------------------------------------------------------
| Asmens kodas | Pavardė       | Vardas        | Bankas     | Sąskaitos nr.   |
-------------------------------------------------------------------------------------------
| 75899        | Zetraitis     | Petras        | Swed       | LT4521          |
| 56485        | Zetraitis     | Petras        | Swed       | LT4521          |
-------------------------------------------------------------------------------------------
Indėlis į darbą (gamta, fizika, chemija, IT) | Atskiros sumos    | Gauta bendra suma    |
| 11 0 0 10                                  | 110 0 0 90        | 200                  |
| 9 0 0 10                                   | 90 0 0 90         | 180                  |
-------------------------------------------------------------------------------------------


Filtruoti pagal temą
-------------------------------------------------------------------------------------------
| Asmens kodas | Pavardė       | Vardas        | Bankas     | Sąskaitos nr.   |
-------------------------------------------------------------------------------------------
| 75896        | Simaitis      | Simas         | Swed       | LT2456
Indėlis į darbą (gamta, fizika, chemija, IT) | Atskiros sumos    | Gauta bendra suma    |
| 80 100 100 80                              | 800 500 700 720   | 2720                 |
-------------------------------------------------------------------------------------------
```

Antras variantas:

U6a.txt

```
56485,Zetraitis,Petras,Swed,LT4521
75896,Simaitis,Simas,Swed,LT2456
```

U6b.txt

```
1000,500,700,900
56485,50,50,50,50
75896,50,50,50,50
```

Rezultatai.txt

```
Darbuotojų informacija
-------------------------------------------------------------------------------------------------
| Asmens kodas | Pavardė       | Vardas        | Bankas     | Sąskaitos nr.    |
-------------------------------------------------------------------------------------------------
| 56485        | Zetraitis     | Petras        | Swed       | LT4521           |
| 75896        | Simaitis      | Simas         | Swed       | LT2456           |
-------------------------------------------------------------------------------------------------


Premijų informacija
-------------------------------------------------------------
Premijų sumos(gamta, fizika, chemija, IT):
-------------------------------------------------------------
1000 500 700 900
-------------------------------------------------------------
| Asmens kodas | Indėlis į darbą (gamta, fizika, chemija, IT) |
| 56485        | 50 50 50 50                                  |
| 75896        | 50 50 50 50                                  |
-------------------------------------------------------------


Darbuotojai gavę mažiau nei vidurkis
-------------------------------------------------------------------------------------------------
```

```
| Asmens kodas | Pavardė       | Vardas        | Bankas     | Sąskaitos nr.
| Indėlis į darbą (gamta, fizika, chemija, IT) | Atskiros sumos    | Gauta bendra suma    |
-------------------------------------------------------------------------------------------


Filtruoti pagal temą
-------------------------------------------------------------------------------------------
| Asmens kodas | Pavardė       | Vardas        | Bankas     | Sąskaitos nr.
-------------------------------------------------------------------------------------------
| 75896        | Simaitis      | Simas         | Swed       | LT2456
| 56485        | Zetraitis     | Petras        | Swed       | LT4521
-------------------------------------------------------------------------------------------
| Indėlis į darbą (gamta, fizika, chemija, IT) | Atskiros sumos    | Gauta bendra suma    |
| 50 50 50 50                                  | 500 250 350 450   | 1550                 |
| 50 50 50 50                                  | 500 250 350 450   | 1550                 |
-------------------------------------------------------------------------------------------
```

2.8. Dėstytojo pastabos

P1, P2, P4

3. Bendrinės klasės ir testavimas (L3)

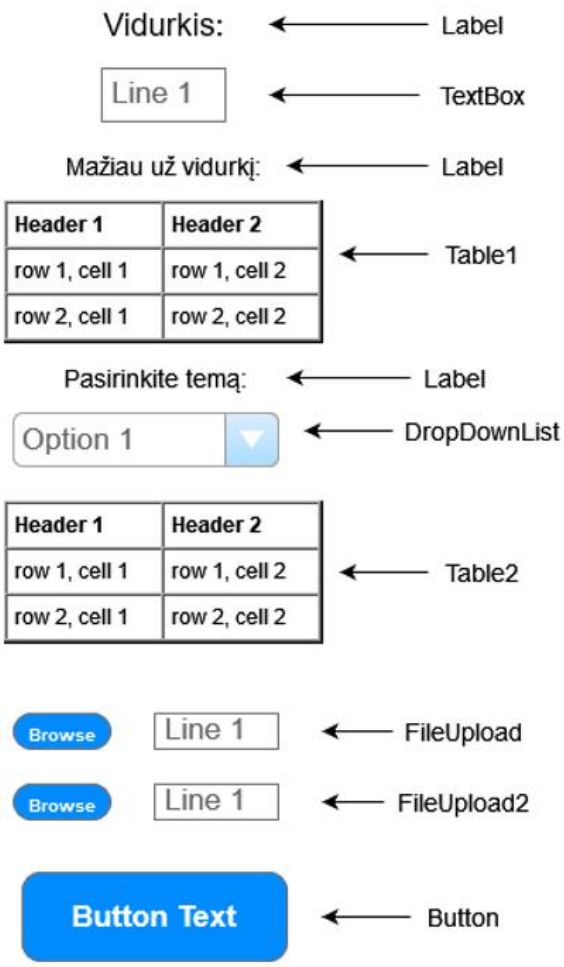
3.1. Darbo užduotis

Variantas: 6

Užduoties pav: Premijos

Darbuotojai atliko 4 darbus ir jie visi gavo premijas. Remiantis darbuotojų indėliais, reikia paskaičiuoti kokia premijos suma priklauso kiekvienam bei bendrą premijų sumą. Svarbu neišdalinti daugiau pinigų nei turima. Reikia sudaryti sąrašą darbuotojų, kurie uždirbo mažiau už vidurkį. Sąrašas turi būti surikiuotas pagal darbuotojų pavardes, vardus abėcėlės tvarka ir bendrą premijų sumą. Taip pat sudaryti pasirinktos temos, kuri įvedama klaviatūra, darbuotojų sąrašą.

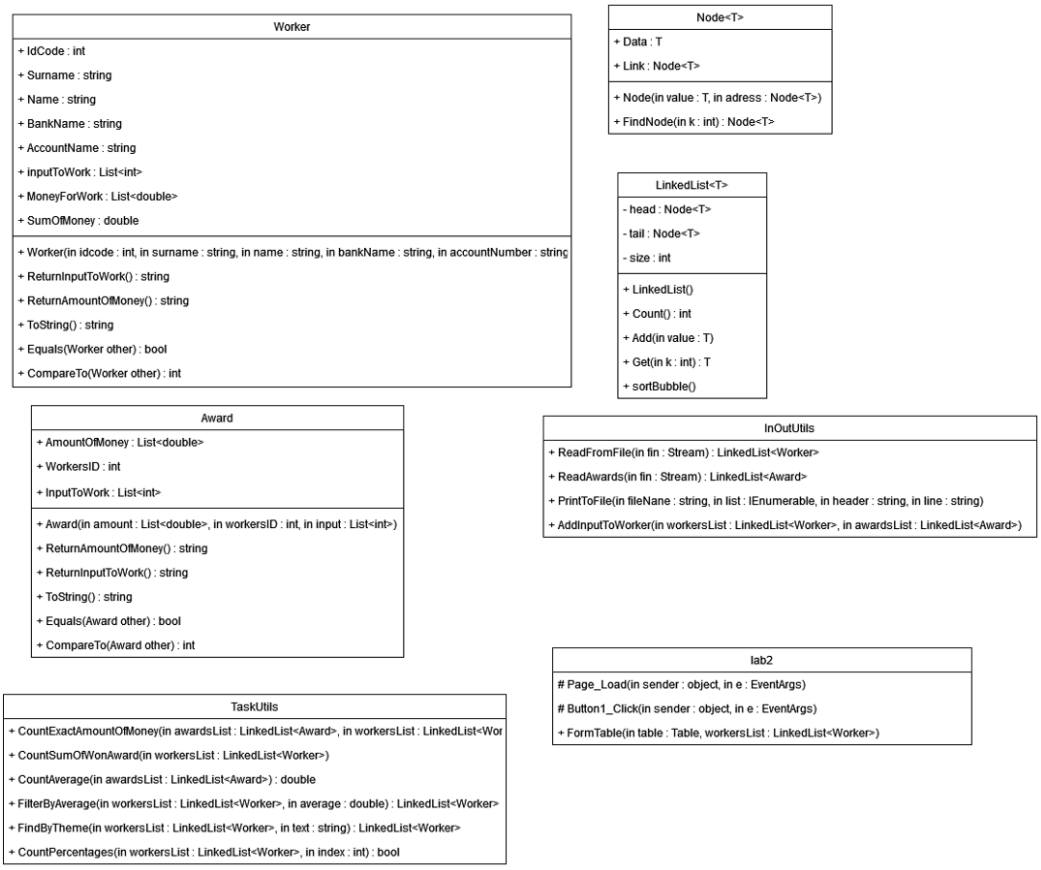
3.2. Grafinės vartotojo sąsajos schema



3.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
TextBox	Font	Larger
	ReadOnly	True
	Height	52px
	Width	72px
Table1	Height	193px
	Width	1048px
Table2	Height	268px
	Width	1030px
DropDownList	Height	55px
	Width	182px
	BackColor	White
	Font	Larger
Label	Text	Vidurkis:
Label	Text	Mažiau už vidurkį:
Label	Text	Pasirinkite norimą temą:
Button	Text	Spręsti
	Font	Larger
	Height	72px
	Width	234px
FileUpload	Height	50px
	Width	500px

3.4. Klasių diagrama



3.5. Programos vartotojo vadovas

Į pirmą lauką parenkamas tekstinis failas su informacija apie darbuotojus(jų informacija), o į antrą – apie premijas informacija:

- Apie darbuotojus: asmens kodas, pavardė, vardas, banko pavadinims ir sąskaitos numeris
- Apie premijas: pirmoje eilutėje premijų sumos, kitose asmens kodas ir indėliai į darbus pagal temas (nurodoma procentais, negali indėlių suma vienai temai viršyti ar būti žemiau 100%)

Parinkus tekstinius failus belieka paspausti mygtuką „Spręsti“ ir ekrane bus parodoma informacija apie tuos darbuotojus, kurie uždirbo mažiau nei vidurkis. Taip pat virš lentelės bus galima matyti ir vidurkį. Galima taip pat surasti darbuotojus pagal temą, tačiau failus reikia įkelti vėl iš naujo. Bus išvedama lentelė, kurie darbuotojai prisidėjo prie pasirinktos temos. Taip pat rašys visus duomenis apie juos.

3.6. Programos tekstas

```
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Web;

namespace OPlab2
{
    /// <summary>
    /// Workers object
    /// </summary>
    public class Worker : IComparable<Worker>, IEquatable<Worker>
    {
        public int IdCode { get; set; }
        public string Surname { get; set; }
        public string Name { get; set; }
        public string BankName { get; set; }
        public string AccountNumber { get; set; }

        public List<int> inputToWork { get; set; }
        public List<double> MoneyForWork { get; set; }
        public double SumOfMoney { get; set; }

        /// <summary>
        /// Worker constructor
        /// </summary>
        /// <param name="idcode">Worker's id code</param>
        /// <param name="surname">Worker's surname</param>
        /// <param name="name">Worker's name</param>
        /// <param name="bankName">Worker's bank name</param>
        /// <param name="accountNumber">Worker's account number</param>
        public Worker(int idcode, string surname, string name, string bankName,
string accountNumber)
        {
            this.IdCode = idcode;
            this.Surname = surname;
            this.Name = name;
            this.BankName = bankName;
            this.AccountNumber = accountNumber;
            this.inputToWork = new List<int>();
            this.MoneyForWork = new List<double>();
            this.SumOfMoney = 0;
        }

        /// <summary>
        /// Overrides ToString method
        /// </summary>
        /// <returns>Formated line</returns>
        public override string ToString()
        {
            return string.Format("| {0, -12} | {1, -13} | {2, -13} | {3, -10} |
{4, -15} | {5, -44} | {6, -17} | {7, -20} |",
                this.IdCode, this.Surname, this.Name, this.BankName,
this.AccountNumber, ReturnInputToWork(), ReturnMoneyToWork(), this.SumOfMoney);
        }

        /// <summary>
        /// Formates line with input to work data
        /// </summary>
        /// <returns>Formated line</returns>
        public string ReturnInputToWork()
        {
            string line = "";
            for (int i = 0; i < inputToWork.Count; i++)
            {
                line += inputToWork[i] + " ";
            }
            return line;
```

```
        }

        /// <summary>
        /// Formates line with money to work
        /// </summary>
        /// <returns>Formated line</returns>
        public string ReturnMoneyToWork()
        {
            string line = "";
            for (int i = 0; i < MoneyForWork.Count; i++)
            {
                line += MoneyForWork[i] + " ";
            }
            return line;
        }

        /// <summary>
        /// Checks if two objects are equal
        /// </summary>
        /// <param name="other">worker object</param>
        /// <returns>true if equal, otherwise false</returns>
        public bool Equals(Worker other)
        {
            if (other == null)
                return false;
            if (this.IdCode == other.IdCode &&
                this.Surname == other.Surname &&
                this.Name == other.Name &&
                this.BankName == other.BankName &&
                this.AccountNumber == other.AccountNumber)
                return true;
            else
                return false;
        }


        /// <summary>
        /// Compares two objects
        /// </summary>
        /// <param name="other">worker object</param>
        /// <returns>1 if first object is higher, -1 if lower, 0 if
equal</returns>
        public int CompareTo(Worker other)
        {
            if(other == null) return 1;
            if (this.Surname.CompareTo(other.Surname) == 0)
                if (this.Name.CompareTo(other.Name) == 0)
                    return this.SumOfMoney.CompareTo(other.SumOfMoney);
                else
                    return this.Name.CompareTo(other.Name);
            else
                return this.Surname.CompareTo(other.Surname);
        }

    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace OPlab2
{
```

```
/// <summary>
/// Award object
/// </summary>
public class Award : IComparable<Award>, IEquatable<Award>
{
    public List<double> AmountOfMoney { get; set; }
    public int WorkersID { get; set; }
    public List<int> InputToWork { get; set; }

    /// <summary>
    /// Object constructor
    /// </summary>
    /// <param name="amount">Amount of money for topics</param>
    /// <param name="workersID">worker's ID number</param>
    /// <param name="input">worker's input to work</param>
    public Award(List<double> amount,int workersID, List<int> input)
    {
        this.AmountOfMoney = amount;
        this.WorkersID = workersID;
        this.InputToWork = input;
    }

    /// <summary>
    /// Creates formated line of Amount of money
    /// </summary>
    /// <returns>Formated line</returns>
    public string ReturnAmountOfMoney()
    {
        string line = "";
        for (int i = 0; i < AmountOfMoney.Count; i++)
        {
            line += AmountOfMoney[i] + " ";
        }
        return line;
    }

    /// <summary>
    /// Creates formated line of Input to work
    /// </summary>
    /// <returns>Formated line</returns>
    public string ReturnInputToWork()
    {
        string line = "";
        for (int i = 0; i < InputToWork.Count; i++)
        {
            line += InputToWork[i] + " ";
        }
        return line;
    }

    /// <summary>
    /// Override ToString methos
    /// </summary>
    /// <returns>Formated line</returns>
    public override string ToString()
    {
        return string.Format("| {0, -12} | {1, -44} | ",
            this.WorkersID, ReturnInputToWork());
    }

    /// <summary>
    /// Checks if two objects are equal
    /// </summary>
    /// <param name="other">Award object</param>
    /// <returns>true if equal, otherwise false</returns>
```

```
        public bool Equals(Award other)
        {
            if (other == null) return false;
            if (this.WorkersID == other.WorkersID)
                return true;
            else
                return false;
        }

        /// <summary>
        /// Compares two objects
        /// </summary>
        /// <param name="other">Award object</param>
        /// <returns>1 if first object is higher, -1 if lower, 0 if
equal</returns>
        public int CompareTo(Award other)
        {
            if (other == null) return 1;
            return this.WorkersID.CompareTo(other.WorkersID);
        }
    }
}


using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace OPlab2
{
    /// <summary>
    /// Private node class
    /// </summary>
    public class Node<E> where E : IComparable<E>, IEquatable<E>
    {
        public E Data { get; set; }
        public Node<E> Link { get; set; }

        /// <summary>
        /// Node constructor
        /// </summary>
        /// <param name="value">Node's value</param>
        /// <param name="address">Shows to next node</param>
        public Node(E value, Node<E> address)
        {
            this.Data = value;
            this.Link = address;
        }

        /// <summary>
        /// Finds necessary node by index
        /// </summary>
        /// <param name="k">Index</param>
        /// <returns>Returns node</returns>
        public Node<E> FindNode(int k)
        {
            Node<E> data = this;
            for (int i = 0; i < k; i++)
            {
                data = data.Link;
            }
            return data;
        }
```

```
    }



    /// <summary>
    /// Linked list class
    /// </summary>
    public sealed class LinkedList<T> : IEnumerable<T> where T: IComparable<T>,
IEquatable<T>
    {
        private Node<T> head;
        private Node<T> tail;
        private int size;

        /// <summary>
        /// Linked list constructor
        /// </summary>
        public LinkedList()
        {
            this.head = null;
            this.tail = null;
            this.size = 0;
        }

        /// <summary>
        /// Counts size of linked list
        /// </summary>
        /// <returns>Size of linked list</returns>
        public int Count()
        {
            return this.size;
        }

        /// <summary>
        /// Add object to list
        /// </summary>
        /// <param name="worker">An object</param>
        public void Add(T value)
        {
            if (this.head == null)
            {
                this.head = new Node<T>(value, this.head);
                this.tail = this.head;
            }
            else
            {
                Node<T> newNode = new Node<T>(value, null);
                this.tail.Link = newNode;
                this.tail = newNode;
            }
            size++;
        }

        /// <summary>
        /// Gets a specific object
        /// </summary>
        /// <param name="k">Index</param>
        /// <returns>Returns worker object</returns>
        public T Get(int k)
        {
            if (k < 0 || k >= size)
            {
                return default;
            }
```

```
            Node<T> current = this.head.FindNode(k);
            return current.Data;
        }


        /// <summary>
        /// Method to sort list
        /// </summary>
        public void SortBubble()
        {
            if (this.head == null)
            {
                return;
            }
            for (Node<T> d = head; d != null; d = d.Link)
            {
                bool sorted = true;
                Node<T> element = this.head;
                for (Node<T> d2 = this.head.Link; d2 != null; d2 = d2.Link)
                {
                    if (element.Data.CompareTo(d2.Data) > 0)
                    {
                        T value = element.Data;
                        element.Data = d2.Data;
                        d2.Data = value;
                        sorted = false;
                    }
                    element = d2;
                }
                if (sorted)
                {
                    return;
                }
            }
        }

        /// <summary>
        /// Get enumerator
        /// </summary>
        /// <returns>data</returns>
        public IEnumerator<T> GetEnumerator()
        {
            for (Node<T> dd = head; dd != null; dd = dd.Link)
            {
                yield return dd.Data;
            }
        }

        /// <summary>
        /// get enumerator
        /// </summary>
        /// <returns>enumerator</returns>
        IEnumerator IEnumerable.GetEnumerator()
        {
            return (IEnumerator)GetEnumerator();
        }
    }
}


using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Linq;
```

```
using System.Text;
using System.Web;

namespace OPlab2
{
    /// <summary>
    /// Class to read/print to file
    /// </summary>
    public class InOutUtils
    {
        /// <summary>
        /// Reads workers from file
        /// </summary>
        /// <param name="fin">file</param>
        /// <returns>linked list with workers</returns>
        public static LinkedList<Worker> ReadFromFile(Stream fin)
        {
            LinkedList<Worker> workerList = new LinkedList<Worker>();
            string line = null;
            using (StreamReader reader = new StreamReader(fin))
            {
                while ((line = reader.ReadLine()) != null)
                {
                    string[] parts = line.Split(',');
                    int idCode = int.Parse(parts[0]);
                    string surname = parts[1];
                    string name = parts[2];
                    string bankName = parts[3];
                    string bankNumer = parts[4];
                    Worker worker = new Worker(idCode, surname, name, bankName,
bankNumer);
                    workerList.Add(worker);
                }
            }

            return workerList;
        }

        /// <summary>
        /// Prints list
        /// </summary>
        /// <param name="fileName">file name to print</param>
        /// <param name="list">linked list</param>
        /// <param name="header">text, file description</param>
        /// <param name="line">extra line</param>
        public static void PrintToFile(string fileName, IEnumerable list, string
header, string line)
        {
            using(var file = new StreamWriter(fileName, true))
            {
                file.WriteLine(header);
                file.WriteLine(new string('-', 169));
                file.WriteLine(line);
                file.WriteLine(new string('-', 169));
                foreach(var item in list)
                {
                    file.WriteLine(item.ToString());
                }
                file.WriteLine(new string('-', 169));
                file.WriteLine(" ");
            }
        }

        /// <summary>
        /// Reads awards from file
```

```
        /// </summary>
        /// <param name="fin">file</param>
        /// <returns>linked list with awards</returns>
        public static LinkedList<Award> ReadAwards(Stream fin)
        {
            LinkedList<Award> awardsList = new LinkedList<Award>();
            List<double> amountOfMoney = new List<double>();
            string line = null;
            using (StreamReader reader = new StreamReader(fin))
            {
                while ((line = reader.ReadLine()) != null)
                {
                    string[] parts = line.Split(',');
                    if (parts.Length == 4)
                    {
                        for (int i = 0; i < parts.Length; i++)
                        {
                            amountOfMoney.Add(int.Parse(parts[i]));
                        }
                    }
                    else
                    {
                        int idCode = int.Parse(parts[0]);

                        List<int> amountOfInput = new List<int>();
                        amountOfInput.Add(int.Parse(parts[1]));
                        amountOfInput.Add(int.Parse(parts[2]));
                        amountOfInput.Add(int.Parse(parts[3]));
                        amountOfInput.Add(int.Parse(parts[4]));

                        Award award = new Award(amountOfMoney, idCode,
amountOfInput);
                        awardsList.Add(award);
                    }

                }
            }
            return awardsList;
        }

        /// <summary>
        /// Add input to workers
        /// </summary>
        /// <param name="workersList">list with workers</param>
        /// <param name="awardsList">list with awrads</param>
        public static void AddInputToWorker(LinkedList<Worker> workersList,
LinkedList<Award> awardsList)
        {
            foreach (Award award in awardsList)
            {
                foreach (Worker worker in workersList)
                {
                    if (award.WorkersID == worker.IdCode)
                    {
                        worker.inputToWork = award.InputToWork;
                    }
                }
            }
        }

    }
}


using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace OPlab2
{
    public static class TaskUtils
    {
        /// <summary>
        /// Method to count how much each worker got money for every topic
        /// </summary>
        /// <param name="awardsList">Linked list with awards</param>
        /// <param name="workersList">Linked list with workers</param>
        public static void CountExactAmountOfMoney(LinkedList<Award> awardsList,
LinkedList<Worker> workersList)
        {
            List<double> amountOfMoney = awardsList.Get(0).AmountOfMoney;
            for (int i = 0; i < amountOfMoney.Count(); i++)
            {
                List<double> money = new List<double>();
                foreach (Worker worker in workersList)
                {
                    double countedMoney = 0;
                    List<int> inputOfWork = worker.inputToWork;
                    if (CountPercentages(workersList, i))
                    {
                        double first = amountOfMoney[i];
                        double second = inputOfWork[i] * 0.01;
                        countedMoney = first * second;
                    }
                    worker.MoneyForWork.Add(countedMoney);
                }
            }
        }



        /// <summary>
        /// Method to count overall rewards sum
        /// </summary>
        /// <param name="workersList">Linked list with worker</param>
        public static void CountSumOfWonAward(LinkedList<Worker> workersList)
        {
            foreach(Worker worker in workersList)
            {
                double sum = 0;
                List<double> awards = worker.MoneyForWork;
                for (int j = 0; j < awards.Count; j++)
                {
                    sum += awards[j];
                }
                worker.SumOfMoney = sum;
            }
        }

        /// <summary>
        /// Counts average amount of won money
        /// </summary>
        /// <param name="awardsList">List of awards</param>
        /// <returns>Returs average</returns>
        public static double CountAverage(LinkedList<Award> awardsList)
        {
            double average = 0;
            double sum = 0;
```

```
            List<double> awards = awardsList.Get(0).AmountOfMoney;
            for (int i = 0; i < awards.Count; i++)
            {
                sum += awards[i];
            }
            average = sum / awards.Count;
            return average;
        }

        /// <summary>
        /// Method which finds workers who got less than average
        /// </summary>
        /// <param name="workersList">Linked list with workers</param>
        /// <param name="average">Average amount of money</param>
        /// <returns>Linked list with filtered workers</returns>
        public static LinkedList<Worker> FilterByAverage(LinkedList<Worker>
workersList, double average)
        {
            LinkedList<Worker> lessThanAverage = new LinkedList<Worker>();
            foreach(Worker worker in workersList)
            {
                double money = worker.SumOfMoney;
                if (money < average)
                {
                    lessThanAverage.Add(worker);
                }
            }
            return lessThanAverage;
        }

        /// <summary>
        /// Method which finds workers who worked on specific topic
        /// </summary>
        /// <param name="workersList">Linked list with workers</param>
        /// <param name="text">Name of theme</param>
        /// <returns>Linked list with workers who worked on specific
topic</returns>
        public static LinkedList<Worker> FindByTheme(LinkedList<Worker>
workersList, string text)
        {
            Dictionary<string, int> themes = new Dictionary<string, int>();
            themes.Add("GAMTA", 0);
            themes.Add("FIZIKA", 1);
            themes.Add("CHEMIJA", 2);
            themes.Add("IT", 3);

            LinkedList<Worker> searchedByTheme = new LinkedList<Worker>();
            foreach (Worker worker in workersList)
            {
                List<int> inputToWork = worker.inputToWork;
                if (inputToWork[themes[text]] > 0)
                {
                    searchedByTheme.Add(worker);
                }
            }
            return searchedByTheme;
        }

        /// <summary>
        /// Method to check if info about amount of input was given correctly
        /// </summary>
        /// <param name="workersList">Linked list with workers</param>
        /// <param name="index">Index of theme</param>
        /// <returns>True if 100%, otherwise false</returns>
```

```
        public static bool CountPercentages(LinkedList<Worker> workersList, int
index)
        {
            int percentage = 0;
            foreach(Worker worker in workersList)
            {
                List<int> inputToWork = worker.inputToWork;
                if (inputToWork[index] > 0)
                {
                    percentage += inputToWork[index];
                }
            }
            if (percentage == 100)
            {
                return true;
            }
            return false;
        }

    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.IO;

namespace OPlab2
{
    /// <summary>
    /// Main class
    /// </summary>
    public partial class lab2 : System.Web.UI.Page
    {
        public LinkedList<Worker> workersList;
        public LinkedList<Award> awardsList;

        /// <summary>
        /// Method which loads things when page is opened
        /// </summary>
        /// <param name="sender">sender</param>
        /// <param name="e">e event args</param>
        protected void Page_Load(object sender, EventArgs e)
        {
            if (DropDownList1.Items.Count == 0)
            {
                DropDownList1.Items.Add("GAMTA");
                DropDownList1.Items.Add("FIZIKA");
                DropDownList1.Items.Add("CHEMIJA");
                DropDownList1.Items.Add("IT");
            }
        }

        /// <summary>
        /// Main method to start all calculations
        /// </summary>
        /// <param name="sender">sender</param>
        /// <param name="e">e event args</param>
        protected void Button1_Click(object sender, EventArgs e)
        {
```

```
            if (File.Exists(Server.MapPath("Rezultatai.txt")))
            {
                File.Delete(Server.MapPath("Rezultatai.txt"));
            }

            if (FileUpload1.HasFile &&
                FileUpload1.FileName.EndsWith(".txt"))
            {
                workersList = InOutUtils.ReadFromFile(FileUpload1.FileContent);
            }

            if (FileUpload2.HasFile &&
                FileUpload2.FileName.EndsWith(".txt"))
            {
                awardsList = InOutUtils.ReadAwards(FileUpload2.FileContent);
            }

            InOutUtils.PrintToFile(Server.MapPath("Rezultatai.txt"), workersList,
"Darbuotojų informacija", "Asmens kodas | Pavardė | Vardas | Bankas | Sąskaitos
nr. | c Atskiros sumos | Gauta bendra suma |");
            InOutUtils.PrintToFile(Server.MapPath("Rezultatai.txt"), awardsList,
"Premijų informacija", "Asmens kodas | Indėlis į darbą (gamta, fizika, chemija,
IT) |");

            InOutUtils.AddInputToWorker(workersList, awardsList);
            TaskUtils.CountExactAmountOfMoney(awardsList, workersList);
            TaskUtils.CountSumOfWonAward(workersList);

            InOutUtils.PrintToFile(Server.MapPath("Rezultatai.txt"), workersList,
"Darbuotojų informacija", "Asmens kodas | Pavardė | Vardas | Bankas | Sąskaitos
nr. | Indėlis į darbą (gamta, fizika, chemija, IT) | Atskiros sumos | Gauta bendra
suma |");

            double average = TaskUtils.CountAverage(awardsList);
            TextBox1.Text = average.ToString();

            LinkedList<Worker> lessThanAverage =
TaskUtils.FilterByAverage(workersList, average);
            lessThanAverage.SortBubble();
            InOutUtils.PrintToFile(Server.MapPath("Rezultatai.txt"),
lessThanAverage, "Darbuotojai gavę mažiau nei vidurkis", "Asmens kodas | Pavardė |
Vardas | Bankas | Sąskaitos nr. | Indėlis į darbą (gamta, fizika, chemija, IT) |
Atskiros sumos | Gauta bendra suma |");

            FormTable(Table1, lessThanAverage);

            string text = DropDownList1.Text;
            LinkedList<Worker> searchedByTheme =
TaskUtils.FindByTheme(workersList, text);
            searchedByTheme.SortBubble();
            InOutUtils.PrintToFile(Server.MapPath("Rezultatai.txt"),
searchedByTheme, "Filtruoti pagal temą", "Asmens kodas | Pavardė | Vardas | Bankas
| Sąskaitos nr. | Indėlis į darbą (gamta, fizika, chemija, IT) | Atskiros sumos |
Gauta bendra suma |");

            FormTable(Table2, searchedByTheme);


        }

        /// <summary>
        /// Method to form table with data
        /// </summary>
        /// <param name="table">Table where info will be stored</param>
```

```
        /// <param name="workersList">Linked list with workers</param>
        public static void FormTable(Table table, LinkedList<Worker> workersList)
        {
            TableRow row1 = new TableRow();
            TableCell surname1 = new TableCell();
            surname1.Text = "<b>Pavardė</b>";
            row1.Cells.Add(surname1);
            TableCell name1 = new TableCell();
            name1.Text = "<b>Vardas</b>";
            row1.Cells.Add(name1);
            TableCell input1 = new TableCell();
            input1.Text = "<b>Indėlis į darbą (gamta, fizika, chemija, IT)</b>";
            row1.Cells.Add(input1);
            TableCell moneyForWork1 = new TableCell();
            moneyForWork1.Text = "<b>Pinigai už darbą (gamta, fizika, chemija,
IT)</b>";
            row1.Cells.Add(moneyForWork1);
            TableCell money1 = new TableCell();
            money1.Text = "<b>Galutinė suma</b>";
            row1.Cells.Add(money1);

            table.Rows.Add(row1);
            foreach(Worker worker in workersList)
            {
                TableRow row = new TableRow();
                TableCell surname = new TableCell();
                surname.Text = worker.Surname;
                TableCell name = new TableCell();
                name.Text = worker.Name;
                TableCell input = new TableCell();
                input.Text = worker.ReturnInputToWork();
                TableCell moneyForWork = new TableCell();
                moneyForWork.Text = worker.ReturnMoneyToWork();
                TableCell money = new TableCell();
                money.Text = worker.SumOfMoney.ToString();


                row.Cells.Add(surname);
                row.Cells.Add(name);
                row.Cells.Add(input);
                row.Cells.Add(moneyForWork);
                row.Cells.Add(money);

                table.Rows.Add(row);
            }
        }
    }
}
```

```
body {
    background-color: rgb(255, 238, 194);
    font-size: 18px;
}
.info {
    text-align: center;
    margin: 0 auto;
    background-color: white;
    border-radius: 10px;
    padding: 15px 25px;
    padding-bottom: 400px;
    width: 80%;
    background-color: rgb(255, 252, 245);
}
```

```
#Table1 {
    margin: 0 auto;
    border: 1px solid #ddd;
    background-color: rgb(255, 248, 232);
    border-radius: 10px;
    padding: 15px;
}

#Table2 {
    margin: 0 auto;
    border: 1px solid #ddd;
    background-color: rgb(255, 248, 232);
    border-radius: 10px;
    padding: 15px;
}

.text{
    font-size: 32px;
}

.text2 {
    font-size: 17px;
}

#Button1 {
    border-radius: 10px;
    border-color: rgb(255, 238, 194);
    background-color: white;
}

    #Button1:hover {
        background-color: rgb(255, 238, 194);
        box-shadow: 0 0.5em 0.5em -0.4em rgb(255, 248, 232);
        transform: translateY(-0.25em);
        cursor: pointer;
    }
```

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OPlab2;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using FluentAssertions;


namespace OPlab2.Tests
{
    [TestClass()]
    public class LinkedListTests
    {
        [TestMethod()]
        public void EmptyContainerCountIsSetToZero()
        {
            LinkedList<Worker> list = new LinkedList<Worker>();
            Assert.AreEqual(0, list.Count(), "Naujo list dydis nelygus 0");
        }

        [TestMethod()]
        public void EmptyContainerIfHasSomeSpace()
```

```
        {
            LinkedList<Worker> list = new LinkedList<Worker>();
            Action act = () => list.Get(0);
            act.Should().NotThrow<NullReferenceException>();
        }

        [TestMethod()]
        public void ListWithOneObjectSizeEqualsToOne()
        {
            LinkedList<Worker> linkedlist = new LinkedList<Worker>();
            Worker worker = new Worker(2456, "Pavardenis", "Vardenis", "Swed",
"LT545");
            linkedlist.Add(worker);
            linkedlist.Count().Should().Be(1);
        }

        [TestMethod()]
        public void GetFifthElementReturnsNull()
        {
            LinkedList<Worker> linkedlist = new LinkedList<Worker>();
            linkedlist.Get(5).Should().Be(null);
        }

        [TestMethod()]
        public void GetFirstElementReturnsNull()
        {
            LinkedList<Worker> linkedlist = new LinkedList<Worker>();
            linkedlist.Get(1).Should().Be(null);
        }

        [TestMethod()]
        public void MethodSortReturnsSorted()
        {
            LinkedList<Worker> linkedlist = new LinkedList<Worker>();
            Worker worker = new Worker(2456, "Pavardenis", "Vardenis", "Swed",
"LT545");
            Worker worker2 = new Worker(2456, "Andriejus", "Vardenis", "Swed",
"LT545");
            linkedlist.Add(worker);
            linkedlist.Add(worker2);
            linkedlist.SortBubble();
            linkedlist.Get(0).Should().Be(worker2);
            linkedlist.Get(1).Should().Be(worker);
        }

        [TestMethod()]
        public void SortEmptyListReturnsNull()
        {
            LinkedList<Worker> linkedlist = new LinkedList<Worker>();
            linkedlist.SortBubble();
            linkedlist.Get(0).Should().Be(null);
        }

        [TestMethod()]
        public void MethodSortOneElementReturnsSorted()
        {
            LinkedList<Worker> linkedlist = new LinkedList<Worker>();
            Worker worker = new Worker(2456, "Pavardenis", "Vardenis", "Swed",
"LT545");
            linkedlist.Add(worker);
            linkedlist.SortBubble();
            linkedlist.Get(0).Should().Be(worker);
            linkedlist.Count().Should().Be(1);
        }
```

```
        [TestMethod()]
        public void MethodSortSameReturnsSorted()
        {
            LinkedList<Worker> linkedlist = new LinkedList<Worker>();
            Worker worker = new Worker(2456, "Pavardenis", "Vardenis", "Swed",
"LT545");
            Worker worker2 = new Worker(2456, "Pavardenis", "Vardenis", "Swed",
"LT545");
            linkedlist.Add(worker);
            linkedlist.Add(worker2);
            linkedlist.SortBubble();
            linkedlist.Get(0).Should().Be(worker);
            linkedlist.Get(1).Should().Be(worker2);
        }

        [TestMethod()]
        public void CheckIfAddedInCorrectOrder()
        {
            LinkedList<Worker> linkedlist = new LinkedList<Worker>();
            Worker worker = new Worker(2456, "Pavardenis", "Vardenis", "Swed",
"LT545");
            Worker worker2 = new Worker(2456, "Andriejus", "Vardenis", "Swed",
"LT545");
            Worker worker3 = new Worker(2456, "Andriejus", "Vardas", "Swed",
"LT545");
            linkedlist.Add(worker);
            linkedlist.Add(worker2);
            linkedlist.Add(worker3);
            linkedlist.Get(0).Should().Be(worker);
            linkedlist.Get(1).Should().Be(worker2);
            linkedlist.Get(2).Should().Be(worker3);
        }
    }
}
```

3.7. Pradiniai duomenys ir rezultatai

Pirmas variantas:

U6a.txt

```
56485,Zetraitis,Petras,Swed,LT4521
75896,Simaitis,Simas,Swed,LT2456
75899,Zetraitis,Petras,Swed,LT4521
```

U6b.txt

```
1000,500,700,900
56485,9,0,0,10
75896,80,100,100,80
75899,11,0,0,10
```

Rezultatai.txt

Darbuotojų informacija

Asmens kodas	Pavardė	Vardas	Bankas	Sąskaitos nr.
56485	Zetraitis	Petras	Swed	LT4521
75896	Simaitis	Simas	Swed	LT2456
75899	Zetraitis	Petras	Swed	LT4521

Premijų informacija

Premijų sumos(gamta, fizika, chemija, IT):

1000 500 700 900

Asmens kodas	Indėlis į darbą (gamta, fizika, chemija, IT)
56485	9 0 0 10
75896	80 100 100 80
75899	11 0 0 10

Darbuotojai gavę mažiau nei vidurkis

Asmens kodas	Pavardė	Vardas	Bankas	Sąskaitos nr.
75899	Zetraitis	Petras	Swed	LT4521
56485	Zetraitis	Petras	Swed	LT4521

Indėlis į darbą (gamta, fizika, chemija, IT)	Atskiros sumos	Gauta bendra suma
11 0 0 10	110 0 0 90	200
9 0 0 10	90 0 0 90	180

Filtruoti pagal temą

Asmens kodas	Pavardė	Vardas	Bankas	Sąskaitos nr.
75896	Simaitis	Simas	Swed	LT2456

Indėlis į darbą (gamta, fizika, chemija, IT)	Atskiros sumos	Gauta bendra suma
80 100 100 80	800 500 700 720	2720

Antras variantas:

U6a.txt

```
56485,Zetraitis,Petras,Swed,LT4521
75896,Simaitis,Simas,Swed,LT2456
```

U6b.txt

```
1000,500,700,900
56485,50,50,50,50
75896,50,50,50,50
```

Rezultatai.txt

```
Darbuotojų informacija
-----------------------------------------------------------------------------------------
| Asmens kodas | Pavardė       | Vardas        | Bankas     | Sąskaitos nr.   |
-----------------------------------------------------------------------------------------
| 56485        | Zetraitis     | Petras        | Swed       | LT4521          |
| 75896        | Simaitis      | Simas         | Swed       | LT2456          |
-----------------------------------------------------------------------------------------


Premijų informacija
-------------------------------------------------------------
Premijų sumos(gamta, fizika, chemija, IT):
-------------------------------------------------------------
1000 500 700 900
-------------------------------------------------------------
| Asmens kodas | Indėlis į darbą (gamta, fizika, chemija, IT) |
| 56485        | 50 50 50 50                                  |
| 75896        | 50 50 50 50                                  |
-------------------------------------------------------------


Darbuotojai gavę mažiau nei vidurkis
-----------------------------------------------------------------------------------------------------
| Asmens kodas | Pavardė       | Vardas        | Bankas     | Sąskaitos nr.
| Indėlis į darbą (gamta, fizika, chemija, IT) | Atskiros sumos    | Gauta bendra suma    |
-----------------------------------------------------------------------------------------------------


Filtruoti pagal temą
-----------------------------------------------------------------------------------------
| Asmens kodas | Pavardė       | Vardas        | Bankas     | Sąskaitos nr.
-----------------------------------------------------------------------------------------
| 75896        | Simaitis      | Simas         | Swed       | LT2456
| 56485        | Zetraitis     | Petras        | Swed       | LT4521
-----------------------------------------------------------------------------------------
| Indėlis į darbą (gamta, fizika, chemija, IT) | Atskiros sumos    | Gauta bendra suma    |
| 50 50 50 50                                  | 500 250 350 450   | 1550                 |
| 50 50 50 50                                  | 500 250 350 450   | 1550                 |
-----------------------------------------------------------------------------------------
```

3.8. Dėstytojo pastabos

P1, P2

4. Polimorfizmas ir išimčių valdymas (L4)

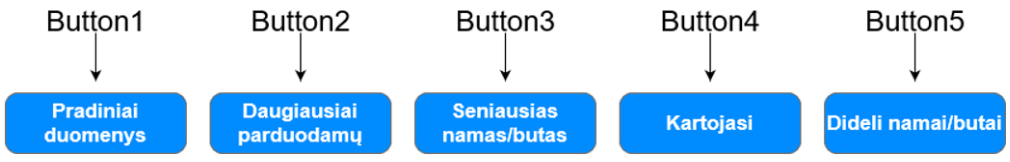
4.1. Darbo užduotis

Variantas 6: Nekilnojamojo turto agentūra

Turimi duomenys apie nekilnojamojo turto agentūras su informacija apie parduodamus objektus. Parduodami butai ir namai. Reikia:

- Rasti, kurioje gatvėje parduodama daugiausiai būtų ir namų. Atspausdinti duomenis ekrane;
- Rasti seniausią nekilnojamojo turto objektą, atspausdinti visą jo informaciją;
- Rasti namus ir butus, kurie buvo paskelbti ne vienoje agentūroje. Išrikiuoti pagal gatvės pavadinimą ir namo numerį. Atspausdinti į failą „Kartojasi.csv“;
- Sudaryti ir surikiuoti didelių NT objektų sąrašą (namai didesni už 200 kv.m, rikiuoti pagal plotą ir šildymo būdą. Butai didesni už 90 kv.m., rikiuoti pagal plotą ir aukštą). Rezultatus surašyti į failą „Dideli.csv“.

4.2. Grafinės vartotojo sąsajos schema



Header 1	Header 2
row 1, cell 1	row 1, cell 2
row 2, cell 1	row 2, cell 2

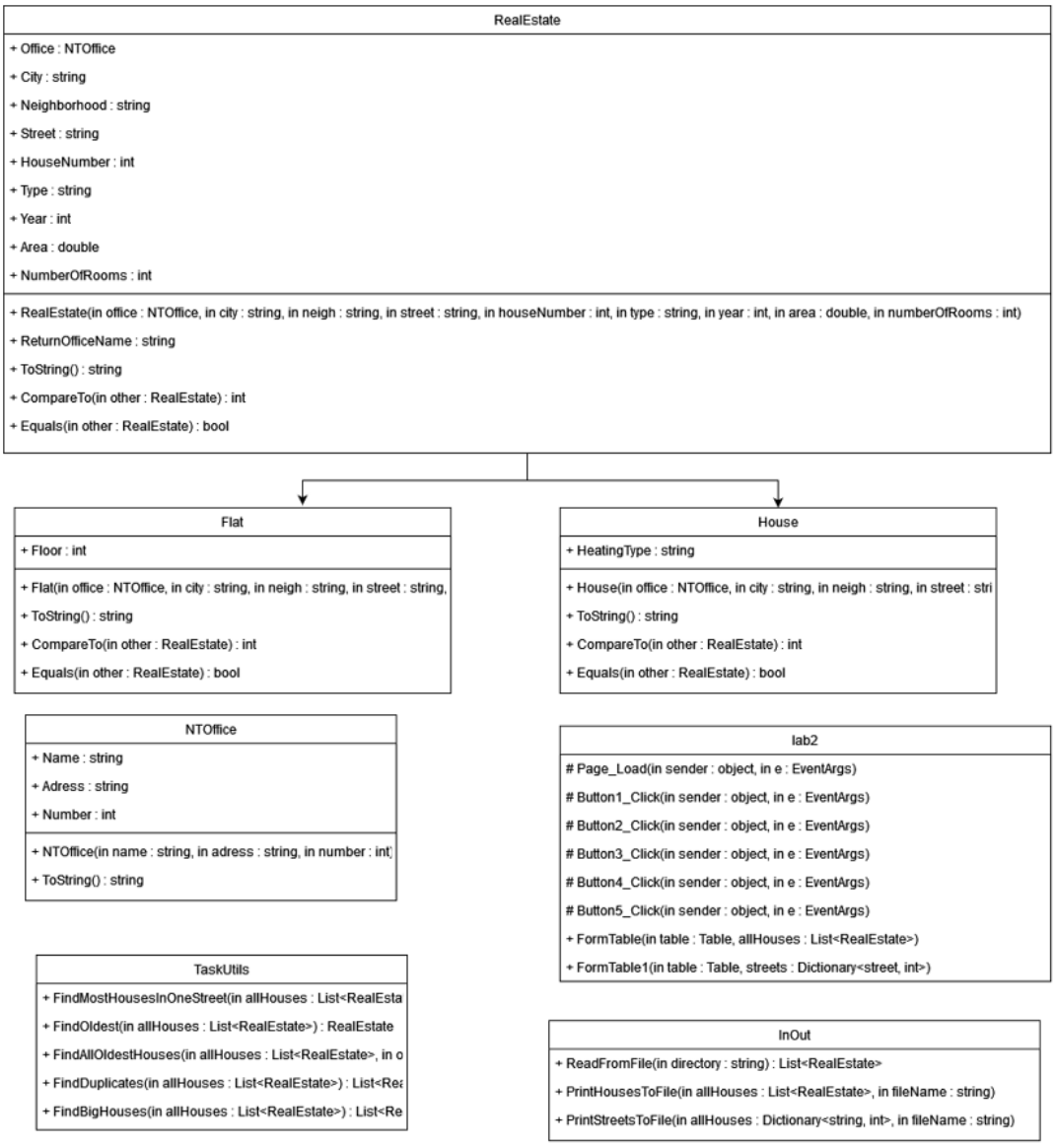
Table1

Miestas	Mikrorajonas	Gatvė	Namo nr.	Tipas	Pastatymo metai	Plotas	Kambarių sk.
Taurage	Taurai	Tauru g	6	butas	2000	50	2
Taurage	Taurai	Tauru g	7	butas	2000	91	2
Kaunas	Aleksotas	Aleksoto g	5	namas	1990	300	5
Vilnius	Antakalnis	Kalno g	6	butas	2010	60	3
Taurage	Taurai	Tauru g	6	butas	2000	50	2
Kaunas	Aleksotas	Aleksoto g	5	namas	1990	300	5

4.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
Button1	Text	Pradiniai duomenys
Button2	Text	Daugiausiai parduodamų
Button3	Text	Seniausias namas/butas
Button4	Text	Kartojasi
Button5	Text	Dideli namai/butai
Table1	Height	350px
	Width	779px

4.4. Klasių diagrama



4.5. Programos vartotojo vadovas

Programa veikia su dviem tekstiniais failais. Juos reikia įkelti į *duomenys* katalogą. Duomenų failas toks: pirma eilute – agentūros pavadinimas, antra – gatvė, trečia – telefono numeris, toliau eina

duomenys apie namus ir butus. Paleidus programą atsidaro langas su galimybę paspausti bet kurį mygtuką iš penkių. Visi mygtukai turi tam tikrą funkcionalumą.

Mygtukas „Pradiniai duomenys“ išves visus duomenis apie parduodamus butus ir namus, „Daugiausiai parduodamų“ išves dviejų stulpelių lentelę su duomenimis kiek namų/butų parduodama tam tikroje gatvėje, „Seniausias namas/butas“ išves lentelę apie seniausią namą/butą(jei yra keli, išves visus), „Kartojasi“ išves apie pasikartojančius parduodamus namus/butus dviejuose agentūrose, „Dideli namai/butai“ išves lentelę apie namus, kurie yra didesni už 200 kv.m, ir butus, kurie didesni už 90 kv.m.

„Rezultatai.txt“ failas bus sukurtas tik kai bus paspaustas bent vienas iš pirmų trijų mygtukų, „Kartojasi.csv“ kai bus paspaustas ketvirtas mygtukas, „Dideli.csv“ kai bus paspasustas penktas mygtukas.

4.6. Programos tekstas

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;

namespace OPlab4
{
    /// <summary>
    /// NT office class
    /// </summary>
    public class NTOffice
    {
        public string Name { get; set; }
        public string Adress { get; set; }
        public int PhoneNumber { get; set; }

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="name">office name</param>
        /// <param name="adress">adress</param>
        /// <param name="number">phone number</param>
        public NTOffice(string name, string adress, int number)
        {
            this.Name = name;
            this.Adress = Adress;
            this.PhoneNumber = number;
        }

        /// <summary>
        /// Forms line
        /// </summary>
        /// <returns>formated line</returns>
        public override string ToString()
        {
            return String.Format("{0},{1},{2}", this.Name, this.Adress,
this.PhoneNumber);
        }
    }
}
```

```
/// <summary>
    /// Abstract Real Estate class
    /// </summary>
    public abstract class RealEstate : IComparable<RealEstate>,
IEquatable<RealEstate>
    {
        public NTOffice Office { get; set; }
        public string City { get; set; }
        public string Neighborhood { get; set; }
        public string Street { get; set; }
        public int HouseNumber { get; set; }
        public string Type { get; set; }
        public int Year { get; set; }
        public double Area { get; set; }
        public int NumberOfRooms { get; set; }

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="office">office name</param>
        /// <param name="city">city</param>
        /// <param name="neigh">neighborhood</param>
        /// <param name="street">street</param>
        /// <param name="number">house number</param>
        /// <param name="type">type</param>
        /// <param name="year">built year</param>
        /// <param name="area">area</param>
        /// <param name="roomsNumber">number of rooms</param>
        public RealEstate(NTOffice office, string city, string neigh, string
street, int number, string type, int year, double area, int roomsNumber)
        {
            this.Office = office;
            this.City = city;
            this.Neighborhood = neigh;
            this.Street = street;
            this.HouseNumber = number;
            this.Type = type;
            this.Year = year;
            this.Area = area;
            this.NumberOfRooms = roomsNumber;
        }

        /// <summary>
        /// Get office name
        /// </summary>
        /// <returns>office name</returns>
        public string ReturnOfficeName()
        {
            return Office.Name;
        }

        /// <summary>
        /// Methos ToString
        /// </summary>
        /// <returns>formated line</returns>
        public override string ToString()
        {
            return string.Format("{0},{1},{2},{3},{4},{5},{6},{7},{8}",
                this.Office, this.City, this.Neighborhood, this.Street,
this.HouseNumber, this.Type, this.Year, this.Area, this.NumberOfRooms);
        }

        /// <summary>
        /// To compare two objects
        /// </summary>
```

```
        /// <param name="other">second object</param>
        /// <returns>0 if equal, otherwise 1 or -1</returns>
        public virtual int CompareTo(RealEstate other)
        {
            if (this.Street.CompareTo(other.Street) == 0)
                return this.HouseNumber.CompareTo(other.HouseNumber);
            return this.Street.CompareTo(other.Street);
        }

        /// <summary>
        /// Abstract equals method
        /// </summary>
        /// <param name="other">second object</param>
        /// <returns>true if equal or false</returns>
        public abstract bool Equals(RealEstate other);


    }



/// <summary>
    /// House class
    /// </summary>
    public class House : RealEstate, IComparable<RealEstate>,
IEquatable<RealEstate>
    {
        public string HeatingType { get; set; }

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="office">office name</param>
        /// <param name="city">city</param>
        /// <param name="neigh">neighborhood</param>
        /// <param name="street">street</param>
        /// <param name="number">house number</param>
        /// <param name="type">type</param>
        /// <param name="year">built year</param>
        /// <param name="area">area</param>
        /// <param name="roomsNumber">number of rooms</param>
        /// <param name="heatingType">heating type</param>
        public House(NTOffice office,string city, string neigh, string street, int
number, string type, int year, double area, int roomsNumber, string heatingType) :
base(office,city, neigh, street, number, type, year, area, roomsNumber)
        {
            this.HeatingType = heatingType;
        }

        /// <summary>
        /// Forms line
        /// </summary>
        /// <returns>formated line</returns>
        public override string ToString()
        {
            return string.Format("{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10}",
                this.Office, this.City, this.Neighborhood, this.Street,
this.HouseNumber, this.Type, this.Year, this.Area, this.NumberOfRooms,
this.HeatingType, "-");
        }

        /// <summary>
        /// To compare two objects
        /// </summary>
        /// <param name="other">second object</param>
```

```
        /// <returns>0 if equal, otherwise 1 or -1</returns>
        public override int CompareTo(RealEstate other)
        {
            House house = other as House;
            if (house == null) return 1;
            if (this.Area.CompareTo(house.Area) == 0)
                return this.HeatingType.CompareTo(house.HeatingType);
            return this.Area.CompareTo(house.Area);
        }

        /// <summary>
        /// Abstract equals method
        /// </summary>
        /// <param name="other">second object</param>
        /// <returns>true if equal or false</returns>
        public override bool Equals(RealEstate other)
        {
            House house = other as House;
            if (house == null) return false;
            if (this.City.Equals(house.City) &&
                this.Neighborhood.Equals(house.Neighborhood) &&
                this.Street.Equals(house.Street) &&
                this.HouseNumber == house.HouseNumber)
                return true;
            return false;
        }
    }


/// <summary>
    /// Flat class
    /// </summary>
    public class Flat : RealEstate, IComparable<RealEstate>,
IEquatable<RealEstate>
    {
        public int Floor { get; set; }

        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="office">office name</param>
        /// <param name="city">city</param>
        /// <param name="neigh">neighborhood</param>
        /// <param name="street">street</param>
        /// <param name="number">house number</param>
        /// <param name="type">type</param>
        /// <param name="year">built year</param>
        /// <param name="area">area</param>
        /// <param name="roomsNumber">number of rooms</param>
        /// <param name="floor">floor</param>
        public Flat(NTOffice office,string city, string neigh, string street, int
number, string type, int year, double area, int roomsNumber, int floor) :
base(office,city, neigh, street, number, type, year, area, roomsNumber)
        {
            this.Floor = floor;
        }

        /// <summary>
        /// Forms line
        /// </summary>
        /// <returns>Formated line</returns>
        public override string ToString()
        {
            return string.Format("{0},{1},{2},{3},{4},{5},{6},{7},{8},{9},{10}",
```

```
                this.Office, this.City, this.Neighborhood, this.Street,
this.HouseNumber, this.Type, this.Year, this.Area, this.NumberOfRooms, "-
",this.Floor);
        }

        /// <summary>
        /// To compare two objects
        /// </summary>
        /// <param name="other">second object</param>
        /// <returns>0 if equal, otherwise 1 or -1</returns>
        public override int CompareTo(RealEstate other)
        {
            Flat flat = other as Flat;
            if (flat == null) return 1;
            if (this.Area.CompareTo(flat.Area) == 0)
                return this.Floor.CompareTo(flat.Floor);
            return this.Area.CompareTo(flat.Area);
        }

        /// <summary>
        /// Abstract equals method
        /// </summary>
        /// <param name="other">second object</param>
        /// <returns>true if equal or false</returns>
        public override bool Equals(RealEstate other)
        {
            Flat flat = other as Flat;
            if (flat == null) return false;
            if (this.City.Equals(flat.City) &&
                this.Neighborhood.Equals(flat.Neighborhood) &&
                this.Street.Equals(flat.Street) &&
                this.HouseNumber == flat.HouseNumber)
                return true;
            return false;
        }



/// <summary>
    /// Class for reading and printing data
    /// </summary>
    public class InOut
    {
        /// <summary>
        /// Reads directory and creates list with house and flat objects
        /// </summary>
        /// <param name="directoryPath">path to directory</param>
        /// <returns>list of houses and flats</returns>
        public static List<RealEstate> ReadFromFile(string directoryPath)
        {
            List<RealEstate> AllHouses = new List<RealEstate>();
            foreach (string file in Directory.EnumerateFiles(directoryPath,
"*.txt"))
            {
                string[] contents = File.ReadAllLines(file);
                NTOffice office = new NTOffice(contents[0], contents[1],
int.Parse(contents[2]));
                for (int i = 3; i < contents.Length; i++)
                {
                    string[] values = contents[i].Split(',');
                    if (values[4] == "namas")
                    {

                        string city = values[0];
                        string neighborhood = values[1];
```

```
                        string street = values[2];
                        int houseNumber = int.Parse(values[3]);
                        string type = values[4];
                        int year = int.Parse(values[5]);
                        double area = int.Parse(values[6]);
                        int numberOfRooms = int.Parse(values[7]);
                        string heating = values[8];

                        House house = new House(office, city, neighborhood,
street, houseNumber, type, year, area, numberOfRooms, heating);

                        AllHouses.Add(house);
                    }
                    else if (values[4] == "butas")
                    {

                        string city = values[0];
                        string neighborhood = values[1];
                        string street = values[2];
                        int houseNumber = int.Parse(values[3]);
                        string type = values[4];
                        int year = int.Parse(values[5]);
                        double area = int.Parse(values[6]);
                        int numberOfRooms = int.Parse(values[7]);
                        int floor = int.Parse(values[8]);

                        Flat house = new Flat(office, city, neighborhood, street,
houseNumber, type, year, area, numberOfRooms, floor);

                        AllHouses.Add(house);

                    }
                    else
                    {
                        throw new Exception(string.Format("Klaida tekstiniame
faile"));
                    }
                }
            }
            return AllHouses;

        }

        /// <summary>
        /// Prints houses and flats to file
        /// </summary>
        /// <param name="allHouses">houses/flats list</param>
        /// <param name="fileName">file name</param>
        /// <param name="header">header line</param>
        public static void PrintHousesToFile(List<RealEstate> allHouses, string
fileName, string header)
        {
            string[] lines = new string[allHouses.Count() + 10];
            lines[0] = header;
            lines[1] = new string('-', 169);
            lines[2] = String.Format("| {0, -12} | {1, -13} | {2, -13} | {3, -10}
| {4, -15} | {5, -44} | {6, -17} | {7, -20} | {8, -20} | {9, -20} | {10, -20} |",
                "NT agentūra", "Miestas", "Mikrorajonas", "Gatvė", "Namo nr.",
"Tipas", "Pastatymo metai", "Plotas", "Kambarių sk.", "Šildymo tipas", "Aukštas");
            lines[3] = new string('-', 169);
            for (int i = 0; i < allHouses.Count(); i++)
            {
                lines[4 + i] = allHouses[i].ToString();
            }
            lines[allHouses.Count() + 4] = new string('-', 169);
```

```
            File.AppendAllLines(fileName, lines, Encoding.UTF8);
        }

        /// <summary>
        /// Prints streets to file
        /// </summary>
        /// <param name="allHouses">houses/flats list</param>
        /// <param name="fileName">file name</param>
        /// <param name="header">header line</param>
        public static void PrintStreetsToFile(Dictionary<string, int> allHouses,
string fileName, string header)
        {
            List<string> lines = new List<string>();
            lines.Add(header);
            lines.Add(new string('-', 30));
            lines.Add(string.Format("{0, 15} | {1, 20}", "Gatvės pav.",
"Parduodamų skaičius"));
            var str = allHouses.First();
            foreach(var street in allHouses)
            {
                if (street.Value == str.Value)
                    lines.Add(string.Format("{0, 15} | {1, 10}", street.Key,
street.Value));
                else
                    break;
            }
            lines.Add("");
            File.AppendAllLines(fileName, lines);
        }



/// <summary>
    /// Counting class
    /// </summary>
    public class TaskUtils
    {
        /// <summary>
        /// Finds streets where most houses/flats are for sell
        /// </summary>
        /// <param name="allHouses"></param>
        /// <returns></returns>
        public static Dictionary<string, int>
FindMostHousesInOneStreet(List<RealEstate> allHouses)
        {
            Dictionary<string, int> streets = new Dictionary<string, int>();
            for (int i = 0; i < allHouses.Count; i++)
            {
                if (!streets.ContainsKey(allHouses[i].Street))
                {
                    streets.Add(allHouses[i].Street, 1);
                }
                else
                {
                    streets[allHouses[i].Street]++;
                }
            }
            return streets;
        }

        /// <summary>
        /// Finds oldest house
        /// </summary>
        /// <param name="allHouses">list of houses/flats</param>
        /// <returns>oldest house/flat</returns>
```

```
        public static RealEstate FindOldest(List<RealEstate> allHouses)
        {
            int oldest = 3000;
            RealEstate oldestHouse = allHouses[0];
            for (int i = 1; i < allHouses.Count; i++)
            {
                if(allHouses[i].Year < oldest)
                {
                    oldest = allHouses[i].Year;
                    oldestHouse = allHouses[i];
                }
            }
            return oldestHouse;
        }

        /// <summary>
        /// finds all oldest houses/flats
        /// </summary>
        /// <param name="allHouses">list of houses/flats</param>
        /// <param name="oldest">oldest house</param>
        /// <returns>all oldest houses/flats</returns>
        public static List<RealEstate> FindAllOldestHouses(List<RealEstate>
allHouses, RealEstate oldest)
        {
            List<RealEstate> allOldests = new List<RealEstate>();
            for (int i = 0; i < allHouses.Count; i++)
            {
                if(allHouses[i].Year == oldest.Year)
                {
                    allOldests.Add(allHouses[i]);
                }
            }
            return allOldests;
        }

        /// <summary>
        /// Finds duplicates in 2 offices
        /// </summary>
        /// <param name="firstList">list of houses/flats</param>
        /// <returns>list of duplicates</returns>
        public static List<RealEstate> FindDuplicates(List<RealEstate> firstList)
        {
            List<RealEstate> duplicates = new List<RealEstate>();
            for (int i = 0; i < firstList.Count; i++)
            {
                for (int j = 1; j < firstList.Count; j++)
                {
                    if
(!firstList[i].ReturnOfficeName().Equals(firstList[j].ReturnOfficeName()))
                    {
                        if (firstList[i].Equals(firstList[j]))
                        {
                            if(!duplicates.Contains(firstList[j]))
                                duplicates.Add(firstList[j]);
                        }
                    }
                }
            }
            return duplicates;
        }

        /// <summary>
        /// Finds all big houses/flats
        /// </summary>
        /// <param name="allHouses">list of houses/flats</param>
```

```
        /// <returns>list of big houses/flats</returns>
        public static List<RealEstate> FindBigHouses(List<RealEstate> allHouses)
        {
            List<RealEstate> bigHouses = new List<RealEstate>();
            for (int i = 0; i < allHouses.Count; i++)
            {
                if (!bigHouses.Contains(allHouses[i]))
                {
                    if (allHouses[i] is Flat && allHouses[i].Area > 90)
                        bigHouses.Add(allHouses[i]);
                    else if (allHouses[i] is House && allHouses[i].Area > 200)
                        bigHouses.Add(allHouses[i]);
                    else
                        throw new Exception(string.Format("Netinkamas objektas"));
                }
            }
            return bigHouses;
        }
public partial class lab4 : System.Web.UI.Page
    {
        /// <summary>
        /// page load
        /// </summary>
        /// <param name="sender">sender</param>
        /// <param name="e">e</param>
        protected void Page_Load(object sender, EventArgs e)
        {


        }

        /// <summary>
        /// Prints data
        /// </summary>
        /// <param name="sender">sender</param>
        /// <param name="e">e</param>
        protected void Button1_Click(object sender, EventArgs e)
        {
            try
            {
                if (File.Exists(Server.MapPath("Rezultatai.txt")))
                {
                    File.Delete(Server.MapPath("Rezultatai.txt"));
                }
                List<RealEstate> allHouses =
InOut.ReadFromFile(Server.MapPath("duomenys"));
                InOut.PrintHousesToFile(allHouses,
Server.MapPath("Rezultatai.txt"), "Visi namai/butai");

                FormTable(Table1, allHouses);
            }catch(Exception ex)
            {

File.AppendAllText(Server.MapPath("Rezultatai.txt"),string.Format("{0} Exception
caught.", ex));
            }
        }

        /// <summary>
        /// Prints streets
        /// </summary>
        /// <param name="sender">sender</param>
        /// <param name="e">e</param>
        protected void Button2_Click(object sender, EventArgs e)
        {
```

```
            try
            {
                List<RealEstate> allHouses =
InOut.ReadFromFile(Server.MapPath("duomenys"));

                Dictionary<string, int> streets =
TaskUtils.FindMostHousesInOneStreet(allHouses);
                InOut.PrintStreetsToFile(streets,
Server.MapPath("Rezultatai.txt"), "Daugiausia vienoje gatvėje");
                FormTable1(Table1, streets);
            }
            catch (Exception ex)
            {
                File.AppendAllText(Server.MapPath("Rezultatai.txt"),
string.Format("{0} Exception caught.", ex));
            }
        }

        /// <summary>
        /// Prints oldests objects
        /// </summary>
        /// <param name="sender">sender</param>
        /// <param name="e"e></param>
        protected void Button3_Click(object sender, EventArgs e)
        {
            List<RealEstate> allHouses =
InOut.ReadFromFile(Server.MapPath("duomenys"));

            RealEstate oldest = TaskUtils.FindOldest(allHouses);
            List<RealEstate> allOldests = TaskUtils.FindAllOldestHouses(allHouses,
oldest);
            InOut.PrintHousesToFile(allOldests, Server.MapPath("Rezultatai.txt"),
"Seniausi namai/butai");
            //File.AppendAllText(Server.MapPath("Rezultatai.txt"),
oldest.ToString());

            FormTable(Table1, allOldests);
        }

        /// <summary>
        /// Forms table
        /// </summary>
        /// <param name="table">table</param>
        /// <param name="allHouses">list of objects</param>
        public static void FormTable(Table table, List<RealEstate> allHouses)
        {
            TableRow row1 = new TableRow();
            TableCell city = new TableCell();
            city.Text = "<b>Miestas</b>";
            row1.Cells.Add(city);
            TableCell neighborhood = new TableCell();
            neighborhood.Text = "<b>Mikrorajonas</b>";
            row1.Cells.Add(neighborhood);
            TableCell street = new TableCell();
            street.Text = "<b>Gatvė</b>";
            row1.Cells.Add(street);
            TableCell houseNumber = new TableCell();
            houseNumber.Text = "<b>Namo nr.</b>";
            row1.Cells.Add(houseNumber);
            TableCell type = new TableCell();
            type.Text = "<b>Tipas</b>";
            row1.Cells.Add(type);
            TableCell year = new TableCell();
            year.Text = "<b>Pastatymo metai</b>";
            row1.Cells.Add(year);
```

```
            TableCell area = new TableCell();
            area.Text = "<b>Plotas</b>";
            row1.Cells.Add(area);
            TableCell numberOfRooms = new TableCell();
            numberOfRooms.Text = "<b>Kambarių sk.</b>";
            row1.Cells.Add(numberOfRooms);


            table.Rows.Add(row1);
            for (int i = 0; i < allHouses.Count(); i++)
            {
                TableRow row = new TableRow();
                TableCell city1 = new TableCell();
                city1.Text = allHouses[i].City;
                TableCell neighborhood1 = new TableCell();
                neighborhood1.Text = allHouses[i].Neighborhood;
                TableCell street1 = new TableCell();
                street1.Text = allHouses[i].Street;
                TableCell houseNumber1 = new TableCell();
                houseNumber1.Text = allHouses[i].HouseNumber.ToString();
                TableCell type1 = new TableCell();
                type1.Text = allHouses[i].Type;
                TableCell year1 = new TableCell();
                year1.Text = allHouses[i].Year.ToString();
                TableCell area1 = new TableCell();
                area1.Text = allHouses[i].Area.ToString();
                TableCell numberOfRooms1 = new TableCell();
                numberOfRooms1.Text = allHouses[i].NumberOfRooms.ToString();

                row.Cells.Add(city1);
                row.Cells.Add(neighborhood1);
                row.Cells.Add(street1);
                row.Cells.Add(houseNumber1);
                row.Cells.Add(type1);
                row.Cells.Add(year1);
                row.Cells.Add(area1);
                row.Cells.Add(numberOfRooms1);

                table.Rows.Add(row);
            }
        }

        /// <summary>
        /// Forms streets table
        /// </summary>
        /// <param name="table">table</param>
        /// <param name="allHouses">list of streets</param>
        public static void FormTable1(Table table, Dictionary<string, int>
allHouses)
        {
            TableRow row1 = new TableRow();
            TableCell name = new TableCell();
            name.Text = "<b>Gatvė</b>";
            row1.Cells.Add(name);
            TableCell count = new TableCell();
            count.Text = "<b>Kiekis</b>";
            row1.Cells.Add(count);
            table.Rows.Add(row1);
             foreach(var street in allHouses)
            {
                TableRow row = new TableRow();
                TableCell name1 = new TableCell();
                name1.Text = street.Key;
                row.Cells.Add(name1);
                TableCell count1 = new TableCell();
```

```
                count1.Text = street.Value.ToString();
                row.Cells.Add(count1);
                table.Rows.Add(row);
            }
        }

        /// <summary>
        /// Prints duplicates
        /// </summary>
        /// <param name="sender">sender</param>
        /// <param name="e">e</param>
        protected void Button4_Click(object sender, EventArgs e)
        {
            List<RealEstate> allHouses =
InOut.ReadFromFile(Server.MapPath("duomenys"));
            List<RealEstate> duplicates = TaskUtils.FindDuplicates(allHouses);
            if (File.Exists(Server.MapPath("Kartojasi.csv")))
            {
                File.Delete(Server.MapPath("Kartojasi.csv"));
            }
            InOut.PrintHousesToFile(duplicates, Server.MapPath("Kartojasi.csv"),
"Kartojasi");
            FormTable(Table1, duplicates);
        }

        /// <summary>
        /// Prints big houses
        /// </summary>
        /// <param name="sender">sender</param>
        /// <param name="e">e</param>
        protected void Button5_Click(object sender, EventArgs e)
        {
            List<RealEstate> allHouses =
InOut.ReadFromFile(Server.MapPath("duomenys"));
            List<RealEstate> bigHouses = TaskUtils.FindBigHouses(allHouses);
            if (File.Exists(Server.MapPath("Dideli.csv")))
            {
                File.Delete(Server.MapPath("Dideli.csv"));
            }
            InOut.PrintHousesToFile(bigHouses, Server.MapPath("Dideli.csv"),
"Dideli namai ir butai");
            FormTable(Table1, bigHouses);
        }
    }
```

Style.css

```
body {
    background-color: #fffdf9;
    font-size: 18px;
}

#Table1 {
    margin: 0 auto;
    border: 1px solid #ddd;
    background-color: rgb(255, 248, 232);
    border-radius: 10px;
    padding: 15px;
}

#Button1, #Button2, #Button3, #Button4, #Button5 {
    border-radius: 10px;
    border-color: rgb(255, 238, 194);
    background-color: white;
```

```
    height: 30px;
    width: 300px;
}

    #Button1:hover, #Button2:hover, #Button3:hover, #Button4:hover, #Button1:hover
{
        background-color: rgb(255, 238, 194);
        box-shadow: 0 0.5em 0.5em -0.4em rgb(255, 248, 232);
        transform: translateY(-0.25em);
        cursor: pointer;
    }
```

4.7. Pradiniai duomenys ir rezultatai

Pirmas variantas:

data.txt

```
Namai
Veiveriu g.4
865412578
Taurage,Taurai,Tauru g,6,butas,2000,50,2,2
Taurage,Taurai,Tauru g,7,butas,2000,91,2,2
Kaunas,Aleksotas,Aleksoto g,5,namas,1990,300,5,malkos
```

Data2.txt

```
Nameliai
Tuopu g.10
864578210
Vilnius,Antakalnis,Kalno g,6,butas,2010,60,3,3
Taurage,Taurai,Tauru g,6,butas,2000,50,2,2
Kaunas,Aleksotas,Aleksoto g,5,namas,1990,300,5,malkos
```

Rezultatai.txt

```
Visi namai/butai
-------------------------------------------------------------------------------
| NT agentūra| Miestas| Mikrorajona| Gatvė Namo nr.| Tipas| Pastatymo metai |
Plotas | Kambarių sk. | Šildymo tipas| Aukštas|
-------------------------------------------------------------------------------
Namai,,865412578,Taurage,Taurai,Tauru g,6,butas,2000,50,2,-,2
Namai,,865412578,Taurage,Taurai,Tauru g,7,butas,2000,91,2,-,2
Namai,,865412578,Kaunas,Aleksotas,Aleksoto g,5,namas,1990,300,5,malkos,-
Nameliai,,864578210,Vilnius,Antakalnis,Kalno g,6,butas,2010,60,3,-,3
Nameliai,,864578210,Taurage,Taurai,Tauru g,6,butas,2000,50,2,-,2
Nameliai,,864578210,Kaunas,Aleksotas,Aleksoto g,5,namas,1990,300,5,malkos,-
-------------------------------------------------------------------------------
Daugiausia vienoje gatvėje
---------------------------------------
    Gatvės pav. |  Parduodamų skaičius
        Tauru g |           3

Seniausi namai/butai
-------------------------------------------------------------------------------
| NT agentūra| Miestas| Mikrorajona| Gatvė Namo nr.| Tipas| Pastatymo metai |
Plotas | Kambarių sk. | Šildymo tipas| Aukštas|
-------------------------------------------------------------------------------
```

```
Namai,,865412578,Kaunas,Aleksotas,Aleksoto g,5,namas,1990,300,5,malkos,-
Nameliai,,864578210,Kaunas,Aleksotas,Aleksoto g,5,namas,1990,300,5,malkos,-
--------------------------------------------------------------------------------
```

Kartojasi.csv

```
Kartojasi
--------------------------------------------------------------------------------
| NT agentūra| Miestas| Mikrorajona| Gatvė Namo nr.| Tipas| Pastatymo metai |
Plotas | Kambarių sk. | Šildymo tipas| Aukštas|
--------------------------------------------------------------------------------
Nameliai,,864578210,Taurage,Taurai,Tauru g,6,butas,2000,50,2,-,2
Nameliai,,864578210,Kaunas,Aleksotas,Aleksoto g,5,namas,1990,300,5,malkos,-
--------------------------------------------------------------------------------
```

Dideli.csv

```
Dideli namai/butai
--------------------------------------------------------------------------------
| NT agentūra| Miestas| Mikrorajona| Gatvė Namo nr.| Tipas| Pastatymo metai |
Plotas | Kambarių sk. | Šildymo tipas| Aukštas|
--------------------------------------------------------------------------------
Namai,,865412578,Taurage,Taurai,Tauru g,7,butas,2000,91,2,-,2
Namai,,865412578,Kaunas,Aleksotas,Aleksoto g,5,namas,1990,300,5,malkos,-
--------------------------------------------------------------------------------
```

Antras variantas:

data.txt

```
Namai
Veiveriu g.4
865412578
Taurage,Taurai,Tauru g,6,butas,2000,50,2,2
Taurage,Taurai,Tauru g,7,butas,2000,81,2,2
Kaunas,Aleksotas,Aleksoto g,5,namas,1990,120,5,malkos
```

Data2.txt

```
Nameliai
Tuopu g.10
864578210
Vilnius,Antakalnis,Kalno g,6,butas,2010,86,3,3
Druskininkai,Centras,Centro g,12,butas,2014,90,3,2
```

Rezultatai.txt

```
Visi namai/butai
--------------------------------------------------------------------------------
| NT agentūra| Miestas| Mikrorajona| Gatvė Namo nr.| Tipas| Pastatymo metai |
Plotas | Kambarių sk. | Šildymo tipas| Aukštas|
--------------------------------------------------------------------------------
Namai,,865412578,Taurage,Taurai,Tauru g,6,butas,2000,50,2,-,2
Namai,,865412578,Taurage,Taurai,Tauru g,7,butas,2000,81,2,-,2
Namai,,865412578,Kaunas,Aleksotas,Aleksoto g,5,namas,1990,120,5,malkos,-
Nameliai,,864578210,Vilnius,Antakalnis,Kalno g,6,butas,2010,86,3,-,3
Nameliai,,864578210,Druskininkai,Centras,Centro g,12,butas,2014,90,3,-,2
```

```
--------------------------------------------------------------------------------
Daugiausia vienoje gatvėje
------------------------------
    Gatvės pav. |  Parduodamų skaičius
        Tauru g |          2

Seniausi namai/butai
--------------------------------------------------------------------------------
| NT agentūra| Miestas| Mikrorajona| Gatvė Namo nr.| Tipas| Pastatymo metai |
Plotas | Kambarių sk. | Šildymo tipas| Aukštas|
--------------------------------------------------------------------------------
Namai,,865412578,Kaunas,Aleksotas,Aleksoto g,5,namas,1990,120,5,malkos,-
--------------------------------------------------------------------------------
```

Kartojasi.csv

```
Kartojasi
--------------------------------------------------------------------------------
| NT agentūra| Miestas| Mikrorajona| Gatvė Namo nr.| Tipas| Pastatymo metai |
Plotas | Kambarių sk. | Šildymo tipas| Aukštas|
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

Dideli.csv

```
Dideli namai/butai
--------------------------------------------------------------------------------
| NT agentūra| Miestas| Mikrorajona| Gatvė Namo nr.| Tipas| Pastatymo metai |
Plotas | Kambarių sk. | Šildymo tipas| Aukštas|
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
```

4.8. Dėstytojo pastabos

5. Deklaratyvusis programavimas (L5)

5.1. Darbo užduotis

5.2. Grafinės vartotojo sąsajos schema

5.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė

5.4. Klasių diagrama

5.5. Programos vartotojo vadovas

5.6. Programos tekstas

5.7. Pradiniai duomenys ir rezultatai

5.8. Dėstytojo pastabos