

Kauno technologijos universitetas

Informatikos fakultetas

Objektinis programavimas 2 (P175B123)

Laboratorinių darbų ataskaita

Normantas Stankevičius IFF-1/4

Studentas

Prof. Vacius Jusas

Kaunas 2022

TURINYS

1. Rekursija (L1) 4

1.1. Darbo užduotis 4

1.2. Grafinės vartotojo sąsajos schema 5

1.3. Sąsajoje panaudotų komponentų keičiamos savybės 5

1.4. Klasių diagrama 6

1.5. Programos vartotojo vadovas 6

1.6. Programos tekstas 6

1.7. Pradiniai duomenys ir rezultatai 14

1.8. Dėstytojo pastabos 18

2. Dinaminis atminties valdymas (L2) 19

2.1. Darbo užduotis 19

2.2. Grafinės vartotojo sąsajos schema 19

2.3. Sąsajoje panaudotų komponentų keičiamos savybės 19

2.4. Klasių diagrama 19

2.5. Programos vartotojo vadovas 19

2.6. Programos tekstas 19

2.7. Pradiniai duomenys ir rezultatai 19

2.8. Dėstytojo pastabos 20

3. Bendrinės klasės ir testavimas (L3) 21

3.1. Darbo užduotis 21

3.2. Grafinės vartotojo sąsajos schema 21

3.3. Sąsajoje panaudotų komponentų keičiamos savybės 21

3.4. Klasių diagrama 21

3.5. Programos vartotojo vadovas 21

3.6. Programos tekstas 21

3.7. Pradiniai duomenys ir rezultatai 21

3.8. Dėstytojo pastabos 22

4. Polimorfizmas ir išimčių valdymas (L4) 23

4.1. Darbo užduotis 23

4.2. Grafinės vartotojo sąsajos schema 23

4.3. Sąsajoje panaudotų komponentų keičiamos savybės 23

4.4. Klasių diagrama 23

4.5. Programos vartotojo vadovas 23

4.6. Programos tekstas 23

4.7. Pradiniai duomenys ir rezultatai 23

4.8. Dėstytojo pastabos 24

5. Deklaratyvusis programavimas (L5) 25

5.1. Darbo užduotis 25

5.2. Grafinės vartotojo sąsajos schema 25

5.3. Sąsajoje panaudotų komponentų keičiamos savybės 25

5.4. Klasių diagrama 25

5.5. Programos vartotojo vadovas 25

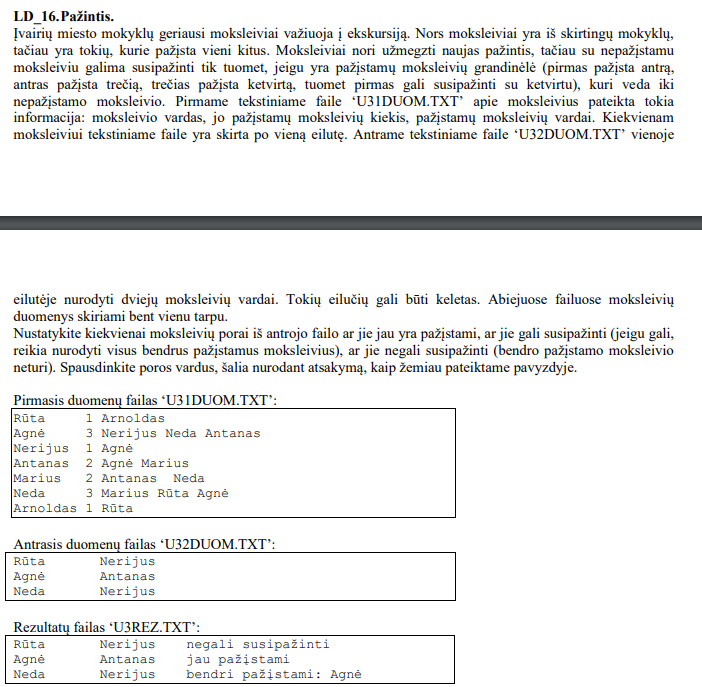
5.6. Programos tekstas 25

5.7. Pradiniai duomenys ir rezultatai 25

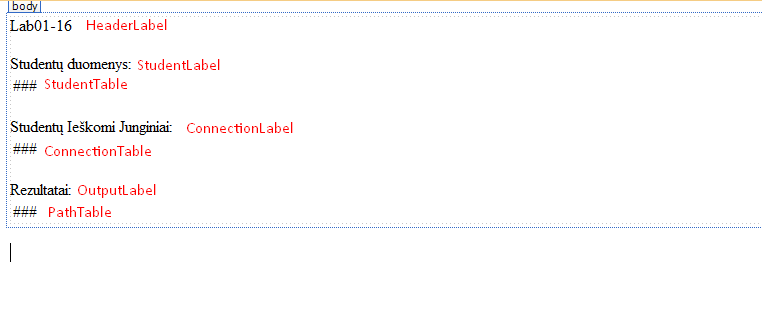
5.8. Dėstytojo pastabos 26

# Rekursija (L1)

## Darbo užduotis



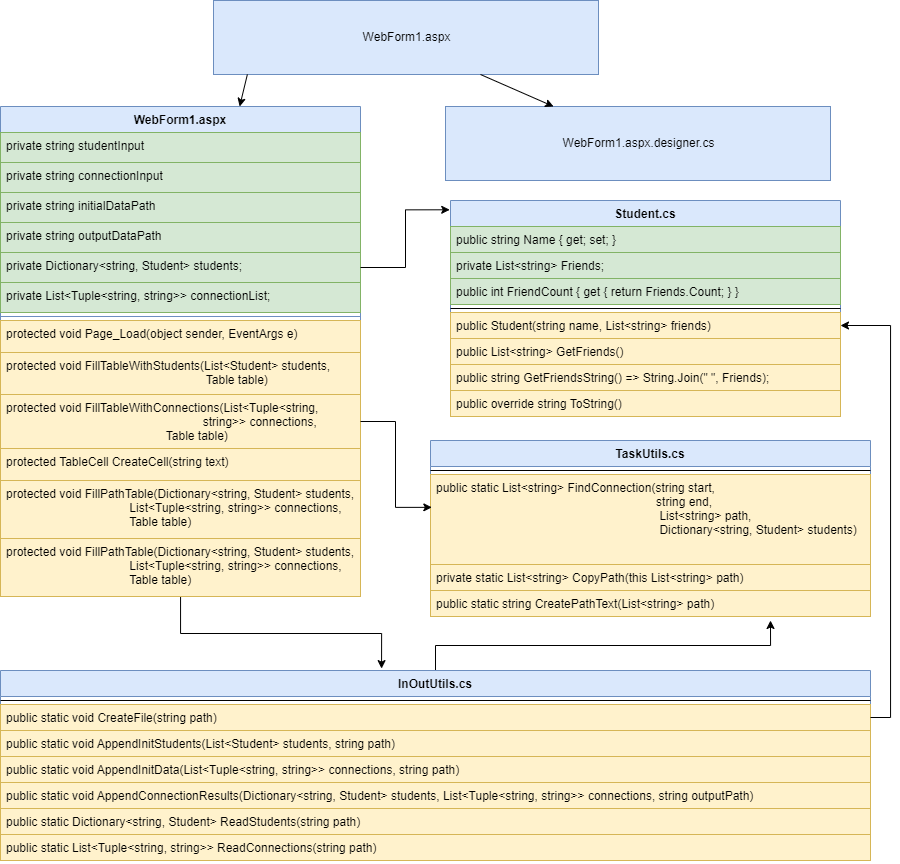
## Grafinės vartotojo sąsajos schema



## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
| HeaderLabel | Text | "Lab01-16" |
| StudentLabel | Text | "Studentų duomenys:" |
| ConnectionLabel | Text | "Studentų Ieškomi Junginiai:" |
| OutputLabel | Text | "Rezultatai:" |

## Klasių diagrama



## Programos vartotojo vadovas

Atsidarius programą, programa nuskaito App\_Data/students.txt ir App\_Data/connections.txt. Naudojant tą informaciją, parašo visą informaciją į StudentTable, ConnectionTable, PathTable su duota ir apskaičiuota informacija.

## Programos tekstas

InOutUtils.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Web;

namespace Lab01

{

/// <summary>

/// InOutUtils class for reading and writing data from/to a file

/// </summary>

public static class InOutUtils

{

/// <summary>

/// Creates a new empty file, ready for appending data

/// </summary>

/// <param name="path">path to the file</param>

public static void CreateFile(string path)

{

using (FileStream fs = new FileStream(path, FileMode.Create))

new StreamWriter(fs, encoding: System.Text.Encoding.UTF8).Close();

}

/// <summary>

/// appends initial student data to TXT file

/// </summary>

/// <param name="students">List of all students (Student object)</param>

/// <param name="path">path to the file where information will be appended</param>

public static void AppendInitStudents(List<Student> students, string path)

{

using (StreamWriter sr = new StreamWriter(path, append: true))

{

sr.WriteLine("Studentai ir jų draugai");

sr.WriteLine($"{"Studentas",-20}|{"Draugų kiekis",-20}|{"Draugai:"}");

foreach (Student student in students)

sr.WriteLine(student);

sr.WriteLine();

}

}

/// <summary>

/// Appends initial connection data to output file

/// </summary>

/// <param name="connections">List of Tuples(string, string) that work as nodes from student a to student b while using DFS</param>

/// <param name="path">path to the file where to append initial data</param>

public static void AppendInitData(List<Tuple<string, string>> connections, string path)

{

using (StreamWriter sr = new StreamWriter(path, append: true))

{

sr.WriteLine("Studentai ir jų ieškomi draugai:");

sr.WriteLine($"{"Studentas",-20} {"Ieškomas draugas",-20}");

foreach (Tuple<string, string> connection in connections)

sr.WriteLine($"{connection.Item1,-20} {connection.Item2,-20}");

sr.WriteLine();

}

}

/// <summary>

/// Appends output connection data to output file

/// </summary>

/// <param name="students">Dictionary, key -> string, name of the student, value -> Student class object of the student</param>

/// <param name="connections">List of tuples(string, string) that is compromised of student names that work as nodes that are used for DFS</param>

/// <param name="outputPath">output path to the txt file where data will be APPENDED</param>

public static void AppendConnectionResults(Dictionary<string, Student> students, List<Tuple<string, string>> connections, string outputPath)

{

using (StreamWriter sr = new StreamWriter(outputPath))

{

sr.WriteLine("Draugai ir jų junginiai, bei keliai:");

sr.WriteLine($"{"Draugas",-20}|{"Ieškomas draugas:",-20}|{"Kelias:"}");

foreach (Tuple<string, string> connection in connections)

{

List<string> studentPath = new List<string>();

studentPath.Add(connection.Item1);

studentPath = TaskUtils.FindConnection(connection.Item1, connection.Item2, studentPath, students);

string pathText = TaskUtils.CreatePathText(studentPath);

sr.WriteLine($"{connection.Item1,-20}|{connection.Item2,-20}|{pathText}");

}

}

}

/// <summary>

/// Creates a name to Student class object relation dictionary

/// </summary>

/// <param name="path">Path to the the text file containing the data</param>

/// <returns>Dictionary(key -> string, value -> Student class object) </returns>

public static Dictionary<string, Student> ReadStudents(string path)

{

Dictionary<string, Student> students = new Dictionary<string, Student>();

using (StreamReader sr = new StreamReader(path))

{

string line;

while ((line = sr.ReadLine()) != null)

{

string[] elements = line.Split(' ');

string name = elements[0];

List<string> friends = new List<string>();

for (int i = 2; i < elements.Length; i++)

friends.Add(elements[i]);

students.Add(name, new Student(name, friends));

}

}

return students;

}

/// <summary>

/// Gets the connections of students

/// </summary>

/// <param name="path">.txt file to the input</param>

/// <returns>List of Tupples(string, string)</returns>

public static List<Tuple<string, string>> ReadConnections(string path)

{

List<Tuple<string, string>> conncetions = new List<Tuple<string, string>>();

using (StreamReader sr = new StreamReader(path))

{

string line;

while ((line = sr.ReadLine()) != null)

{

string[] elements = line.Split(' ');

conncetions.Add(new Tuple<string, string>(elements[0], elements[1]));

}

}

return conncetions;

}

}

}

TaskUtils.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Web;

namespace Lab01

{

/// <summary>

/// TaskUtils class for extra (backend) computation functions

/// </summary>

public static class TaskUtils

{

/// <summary>

/// Recursive implementation of DFS

/// </summary>

/// <param name="start">Start of the person</param>

/// <param name="end">End of the person</param>

/// <param name="path">path to current position from initial start</param>

/// <param name="students">Dictionary, key: string (name of the student), value Student class object</param>

/// <returns>List of strings, that create a path from student a to b</returns>

public static List<string> FindConnection(string start, string end, List<string> path, Dictionary<string, Student> students)

{

Student curr = students[start];

List<string> outputPath = null;

foreach(string next in curr.GetFriends())

{

if (next == end)

return path;

else if (path.Contains(next)) // Checks if the current node has been visited, so it does not loop

continue;

Student nextStudent = students[next];

List<string> pathCopy = path.CopyPath();

pathCopy.Add(next);

List<String> pathToEnd = FindConnection(next, end, pathCopy, students); // Recursion Call

if(outputPath == null || (pathToEnd != null && pathToEnd.Count < outputPath.Count))

outputPath = pathToEnd;

}

return outputPath; // Did not found the path

}

/// <summary>

/// Deep copies a string list

/// </summary>

/// <param name="path">string list</param>

/// <returns>string list</returns>

private static List<string> CopyPath(this List<string> path)

{

List<string> copy = new List<string>();

foreach (string s in path)

copy.Add(s);

return copy;

}

/// <summary>

/// Creates connection depending on the path

/// </summary>

/// <param name="path"> List of strings that the path is compromised of </param>

/// <returns>a string form of the path from student a to student b</returns>

public static string CreatePathText(List<string> path)

{

if (path == null)

return "negali susipažinti";

else if (path.Count == 1)

return "jau pažįstami";

else

{

path.RemoveAt(0);

return $"bendri pažįstami: {String.Join(" ", path)}";

}

}

}

}

Student.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Lab01

{

/// <summary>

/// Student Class Data Object that stores the name and connection

/// </summary>

public class Student

{

public string Name { get; set; }

private List<string> Friends;

public int FriendCount { get { return Friends.Count; } }

/// <summary>

/// Constructor

/// </summary>

public Student(string name, List<string> friends)

{

Name = name;

Friends = new List<string>();

foreach (string friend in friends)

Friends.Add(friend);

}

/// <summary>

/// Copies friends

/// </summary>

/// <returns>Deep copy of Friends List</returns>

public List<string> GetFriends()

{

List<string> friendList = new List<string>();

foreach (string friend in Friends)

friendList.Add(friend);

return friendList;

}

/// <summary>

/// Transforms Friends list into a string seperated by spaces

/// </summary>

/// <returns>string of all friends</returns>

public string GetFriendsString() => String.Join(" ", Friends);

/// <summary>

/// ToString Override

/// </summary>

/// <returns>string version of the object: Name, Friend Count, Friends</returns>

public override string ToString()

{

return $"{Name,-20}|{Friends.Count,20}|{GetFriendsString()}";

}

}

}

WebForm1.aspx:

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="Lab01.WebForm1" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title></title>

</head>

<body>

<form id="form1" runat="server">

<div>

<asp:Label ID="HeaderLabel" runat="server" Text="Lab01-16"></asp:Label>

<br />

<br />

<asp:Label ID="StudentLabel" runat="server" Text="Studentų duomenys:"></asp:Label>

<br />

<asp:Table ID="StudentTable" runat="server">

</asp:Table>

<br />

<asp:Label ID="ConnectionLabel" runat="server" Text="Studentų Ieškomi Junginiai:"></asp:Label>

<br />

<asp:Table ID="ConnectionTable" runat="server">

</asp:Table>

<br />

<asp:Label ID="OutputLabel" runat="server" Text="Rezultatai:"></asp:Label>

<br />

<asp:Table ID="PathTable" runat="server">

</asp:Table>

</div>

</form>

</body>

</html>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

namespace Lab01

{

public partial class WebForm1 : System.Web.UI.Page

{

private string studentInput = @"App\_Data/students.txt";

private string connectionInput = @"App\_Data/connections.txt";

private string initialDataPath = @"App\_Data/initial\_data.txt";

private string outputDataPath = @"App\_Data/result.txt";

private Dictionary<string, Student> students;

private List<Tuple<string, string>> connectionList;

protected void Page\_Load(object sender, EventArgs e)

{

// Initial Data

InOutUtils.CreateFile(Server.MapPath(initialDataPath));

students = InOutUtils.ReadStudents(Server.MapPath(studentInput));

FillTableWithStudents(new List<Student>(students.Values),

StudentTable);

InOutUtils.AppendInitStudents(new List<Student>(students.Values),

Server.MapPath(initialDataPath));

connectionList = InOutUtils.ReadConnections(Server.MapPath(connectionInput));

FillTableWithConnections(connectionList,

ConnectionTable);

InOutUtils.AppendInitData(connectionList,

Server.MapPath(initialDataPath));

FillPathTable(students, connectionList, PathTable);

InOutUtils.CreateFile(Server.MapPath(outputDataPath));

InOutUtils.AppendConnectionResults(students,

connectionList,

Server.MapPath(outputDataPath));

}

/// <summary>

/// Used to show initial Student Data

/// </summary>

/// <param name="students">List Student data type</param>

/// <param name="table">Table Object data type</param>

protected void FillTableWithStudents(List<Student> students, Table table)

{

TableRow row = new TableRow();

row.Cells.Add(CreateCell("Studentas"));

row.Cells.Add(CreateCell("Draugų Kiekis"));

row.Cells.Add(CreateCell("Studentų Draugai:"));

table.Rows.Add(row);

foreach (Student student in students)

{

row = new TableRow();

row.Cells.Add(CreateCell(student.Name));

row.Cells.Add(CreateCell(student.FriendCount.ToString()));

row.Cells.Add(CreateCell(student.GetFriendsString()));

table.Rows.Add(row);

}

}

/// <summary>

/// Used to show initial connection data

/// </summary>

/// <param name="connections">List of Tuples compromised of string, string containing the initial node and end node to use for DFS</param>

/// <param name="table">Table object data type</param>

protected void FillTableWithConnections(List<Tuple<string,

string>> connections,

Table table)

{

TableRow row = new TableRow();

row.Cells.Add(CreateCell("Draugas"));

row.Cells.Add(CreateCell("Ieškomas Draugas"));

table.Rows.Add(row);

foreach (Tuple<string, string> connection in connections)

{

row = new TableRow();

row.Cells.Add(CreateCell(connection.Item1));

row.Cells.Add(CreateCell(connection.Item2));

table.Rows.Add(row);

}

}

/// <summary>

/// Creates A cell with provided Text

/// </summary>

/// <param name="text">text to be added to the Cell.text param</param>

/// <returns>TableCell object</returns>

protected TableCell CreateCell(string text)

{

TableCell cell = new TableCell();

cell.Style.Add("padding", "5px");

cell.Text = text;

return cell;

}

/// <summary>

/// Fills the table with paths from student a to b

/// </summary>

/// <param name="students"> Dictionary, key -> string of the student, value -> student object</param>

/// <param name="connections">List of Tuples compromised of string, string containing the initial node and end node to use for DFS</param>

/// <param name="table">Table object where the data will be added</param>

protected void FillPathTable(Dictionary<string, Student> students,

List<Tuple<string, string>> connections,

Table table)

{

TableRow row = new TableRow();

row.Cells.Add(CreateCell("Draugas"));

row.Cells.Add(CreateCell("Ieškomas Draugas"));

row.Cells.Add(CreateCell("Kelias: "));

table.Rows.Add(row);

foreach (Tuple<string, string> connection in connections)

{

List<string> path = new List<string>();

path.Add(connection.Item1);

path = TaskUtils.FindConnection(connection.Item1,

connection.Item2,

path, students);

string pathText = TaskUtils.CreatePathText(path);

row = new TableRow();

row.Cells.Add(CreateCell(connection.Item1));

row.Cells.Add(CreateCell(connection.Item2));

row.Cells.Add(CreateCell(pathText));

table.Rows.Add(row);

}

}

}

}

## Pradiniai duomenys ir rezultatai

Pradiniai Duomenys 1:

students.txt:

Rūta 1 Arnoldas

Agnė 3 Nerijus Neda Antanas

Nerijus 1 Agnė

Antanas 2 Agnė Marius

Marius 2 Antanas Neda

Neda 3 Marius Rūta Agnė

Arnoldas 1 Rūta

connections.txt:

Rūta Nerijus

Agnė Antanas

Neda Nerijus

Rezultatai 1:

Vartotojo sąsaja:



initial\_data.txt:

Studentai ir jų draugai

Studentas |Draugų kiekis |Draugai:

Rūta | 1|Arnoldas

Agnė | 3|Nerijus Neda Antanas

Nerijus | 1|Agnė

Antanas | 2|Agnė Marius

Marius | 2|Antanas Neda

Neda | 3|Marius Rūta Agnė

Arnoldas | 1|Rūta

Studentai ir jų ieškomi draugai:

Studentas Ieškomas draugas

Rūta Nerijus

Agnė Antanas

Neda Nerijus

Result.txt:

Draugai ir jų junginiai, bei keliai:

Draugas |Ieškomas draugas: |Kelias:

Rūta |Nerijus |negali susipažinti

Agnė |Antanas |jau pažįstami

Neda |Nerijus |bendri pažįstami: Agnė

Pradiniai Duomenys 2:

students.txt:

a 2 g b

b 2 a c

c 2 b f

d 1 e

e 1 d

f 2 h c

g 2 a h

h 2 g f

connections.txt:

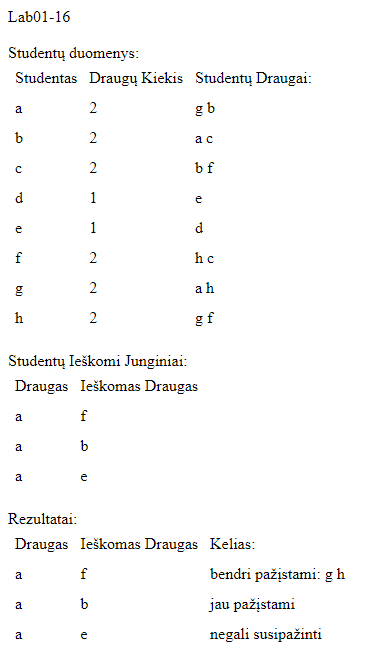
a f

a b

a e

Rezultatai 2:

Vartotojo Sąsaja:



Initial\_data.txt:

Studentai ir jų draugai

Studentas |Draugų kiekis |Draugai:

a | 2|g b

b | 2|a c

c | 2|b f

d | 1|e

e | 1|d

f | 2|h c

g | 2|a h

h | 2|g f

Studentai ir jų ieškomi draugai:

Studentas Ieškomas draugas

a f

a b

a e

result.txt:

Draugai ir jų junginiai, bei keliai:

Draugas |Ieškomas draugas: |Kelias:

a |f |bendri pažįstami: g h

a |b |jau pažįstami

a |e |negali susipažinti

## Dėstytojo pastabos

1. Reiktų šiek tiek pakeisti ataskaitos įvardinimą. Jūsų grupė nėra IFF14.

2. Klasių diagramai vien tik Visual Studio įrankio neužtenka. Jis ne neatskleidžia pilnai klasės vidaus.

3. Garmatinės klaidos "su duotą ir apskaičiuotą "

4. • Įvedimo ir išvedimo metodus, veikiančius su tekstiniu failu, talpinkite į public static class InOutUtils.

5. Parametrus reikia komentuoti visiems metodams // /// appends students to TXT file  
/// public static void AppendInitialStudentData(List students, string path)

6. Čia tik rodyklės perrašymas:

public Student(string name, List friends)

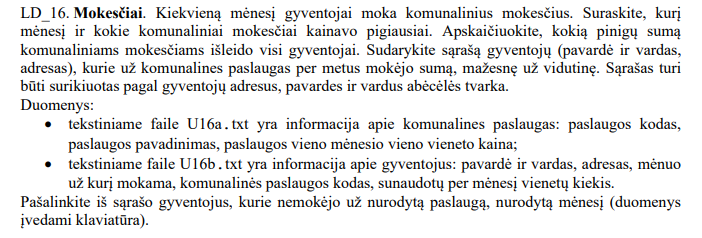
{

Name = name;

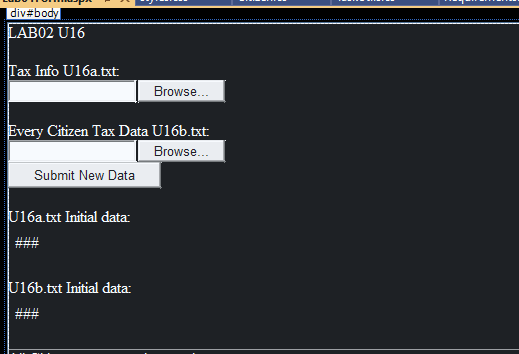
Friends = friends;

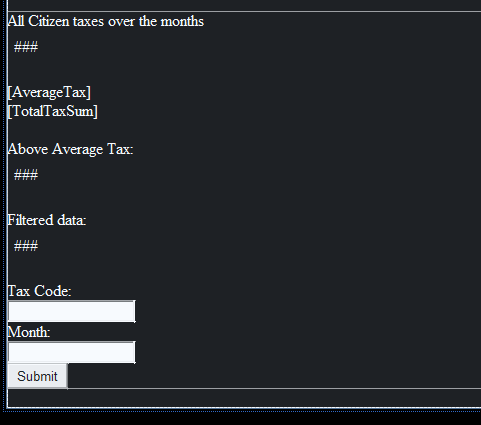
# Dinaminis atminties valdymas (L2)

## Darbo užduotis



## Grafinės vartotojo sąsajos schema

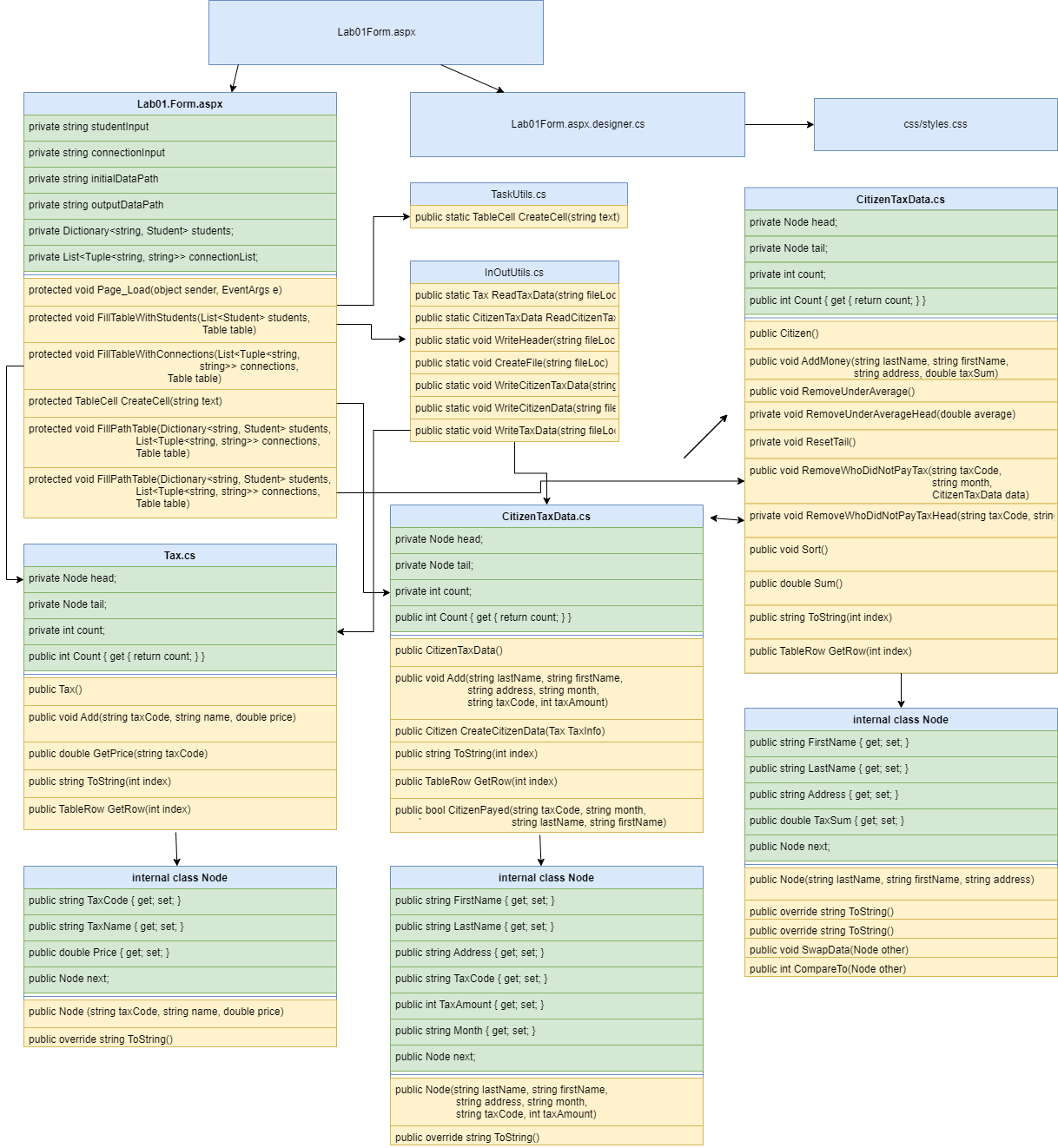




## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
| HeaderLabel | Text | LAB02 U16 |
| Label1 | Text | Tax InfoU16a.txt: |
| Label2 | Text | Every Citizen Tax Data U16b: |
| InitTaxLabel | Text | U16a.txt Initial data: |
| InitCitizenLabel | Text | U16b.txt Initial data: |
| CitizenTaxLabel | Text | All Citizen taxes over the months |
| AverageTax | Text | “” |
| TotalTaxSum | Text | “” |
| CitizenTaxLabel0 | Text | Above Average Tax: |
| FilterData | Text | Filtered data: |
| ButtonFilter | Text | Tax Code: |
| DataButton | Text | Month: |

## Klasių diagrama



## Programos vartotojo vadovas

Jeigu neranda failų visų duombazėje, programa paprašo failų. Jeigu randa tik vieną pradinį failą, rodo tik jį ir prašo likusių failų. Kai abu failai atsiranda duombazėje, užkrauna skaičiavimus. Apskaičiuoja vidutinę mokesčių kainą, sumą visų ir individualių žmonių. Tekstas yra rikiuojamas A-Z pagal: adresą, pavardę, vardą. Kodas leidžia filtruoti žmones, kurie mokėjo nurodytą mėnesį (mėnuo yra string) už nurodytus mokesčius naudojant „Tax Code“ (string). Prie filtered lentelės prideda tik filtruotus duomenis.

## Programos tekstas

Tax.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI.WebControls;

namespace Lab02

{

public class Tax

{

private Node head;

private Node tail;

private int count;

public int Count { get { return count; } }

public Tax()

{

head = null;

tail = null;

count = 0;

}

/// <summary>

/// Adds Node to the tail of the LinkedList

/// </summary>

/// <param name="taxCode">Code of the tax</param>

/// <param name="name"> name of the company</param>

/// <param name="price">price of a single use</param>

public void Add(string taxCode, string name, double price)

{

if (head == null)

{

head = new Node(taxCode, name, price);

tail = head;

count++;

}

else

{

tail.next = new Node(taxCode, name, price);

tail = tail.next;

count++;

}

}

/// <summary>

/// Returns the price of the tax of a single use

/// </summary>

/// <param name="taxCode">Code to identify the type of tax</param>

/// <returns>Double, price of a single use tax item</returns>

public double GetPrice(string taxCode)

{

Node curr = head;

while (curr != null)

{

if (curr.TaxCode == taxCode)

return curr.Price;

curr = curr.next;

}

return 0;

}

/// <summary>

/// Returns string format of Tax Node

/// </summary>

/// <param name="index">index of the node in Tax Linked List</param>

/// <returns>string format of the specified Node</returns>

public string ToString(int index)

{

int i = 0;

for(Node curr = head; curr != null; curr = curr.next)

{

if (i == index)

return curr.ToString();

i++;

}

return "";

}

/// <summary>

/// Returns Tax Node in TableRow format

/// </summary>

/// <param name="index">Index of the Linked List node</param>

/// <returns>Returns Node with specified index in TableRow format</returns>

public TableRow GetRow(int index)

{

int i = 0;

TableRow row = new TableRow();

for (Node curr = head; curr != null; curr = curr.next)

{

if (i == index)

{

row.Cells.Add(TaskUtils.CreateCell(curr.TaxCode));

row.Cells.Add(TaskUtils.CreateCell(curr.TaxName));

row.Cells.Add(TaskUtils.CreateCell(curr.Price.ToString()));

return row;

}

i++;

}

return row;

}

/// <summary>

/// Tax Node

/// </summary>

internal class Node

{

public string TaxCode { get; set; }

public string TaxName { get; set; }

public double Price { get; set; }

public Node next;

/// <summary>

/// Constructor

/// </summary>

/// <param name="taxCode">Code of the tax</param>

/// <param name="name">name of the compnay</param>

/// <param name="price">price of a single use</param>

public Node (string taxCode, string name, double price)

{

TaxCode = taxCode;

TaxName = name;

Price = price;

next = null;

}

/// <summary>

/// Returns Node in string format

/// </summary>

/// <returns>Node in string format</returns>

public override string ToString()

{

return $"{TaxCode,-20}|{TaxName,-20}|{Price,10:f}|";

}

}

}

}

CitizenTaxData.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI.WebControls;

namespace Lab02

{

/// <summary>

/// Citizen class object meant to store name and how much the individual payed for taxe

/// </summary>

public class CitizenTaxData

{

private Node head;

private Node tail;

private int count;

public int Count { get { return count; } }

/// <summary>

/// Constructor

/// </summary>

public CitizenTaxData()

{

head = null;

tail = null;

count = 0;

}

/// <summary>

/// Adds element to Linked List

/// </summary>

/// <param name="lastName">Last Name</param>

/// <param name="firstName">First Name</param>

/// <param name="address">Address</param>

/// <param name="month">Month</param>

/// <param name="taxCode">Tax Code</param>

/// <param name="taxAmount">Tax Amount</param>

public void Add(string lastName, string firstName, string address, string month, string taxCode, int taxAmount)

{

count++;

if (head == null)

{

head = new Node(lastName, firstName, address, month, taxCode, taxAmount);

tail = head;

}

else

{

tail.next = new Node(lastName, firstName, address, month, taxCode, taxAmount);

tail = tail.next;

}

}

/// <summary>

/// Creates Citizen class object using Tax object

/// </summary>

/// <param name="TaxInfo">Tax class object</param>

/// <returns>Citizen class object</returns>

public Citizen CreateCitizenData(Tax TaxInfo)

{

Citizen citizens = new Citizen();

for (Node curr = head; curr != null; curr = curr.next)

{

citizens.AddMoney(curr.LastName, curr.FirstName, curr.Address, (double)TaxInfo.GetPrice(curr.TaxCode) \* curr.TaxAmount);

}

return citizens;

}

/// <summary>

/// Returns string format using index

/// </summary>

/// <param name="index">index of CitizenTaxData Node element</param>

/// <returns>string format of CitizenTaxData</returns>

public string ToString(int index)

{

int i = 0;

for (Node curr = head; curr != null; curr = curr.next)

{

if (i == index)

return curr.ToString();

i++;

}

return "";

}

/// <summary>

/// Returns CitizenTaxData specified element in TableRow format

/// </summary>

/// <param name="index">index of the Node element</param>

/// <returns>TableRow of the specified Node</returns>

public TableRow GetRow(int index)

{

int i = 0;

TableRow row = new TableRow();

for (Node curr = head; curr != null; curr = curr.next)

{

if (i == index)

{

row.Cells.Add(TaskUtils.CreateCell(curr.LastName));

row.Cells.Add(TaskUtils.CreateCell(curr.FirstName));

row.Cells.Add(TaskUtils.CreateCell(curr.Address));

row.Cells.Add(TaskUtils.CreateCell(curr.Month));

row.Cells.Add(TaskUtils.CreateCell(curr.TaxCode));

row.Cells.Add(TaskUtils.CreateCell(curr.TaxAmount.ToString()));

return row;

}

i++;

}

return row;

}

/// <summary>

/// Checks of the specified citizen has payed

/// </summary>

/// <param name="taxCode">Tax Code of the Tax Company</param>

/// <param name="month">Month</param>

/// <param name="lastName">Last name of the citizen</param>

/// <param name="firstName"> First Name of the citizen</param>

/// <returns>true if citizen has payed for specified tax on specified month, false if the citizen did not</returns>

public bool CitizenPayed(string taxCode, string month, string lastName, string firstName)

{

Node curr = head;

while(curr != null)

{

if (curr.LastName == lastName && curr.FirstName == firstName && curr.Month == month && curr.TaxCode == taxCode)

return true; // The Person paid for the month

curr = curr.next;

}

return false;

}

/// <summary>

/// Node class object for CitizenTaxData

/// </summary>

internal class Node

{

public string FirstName { get; set; }

public string LastName { get; set; }

public string Address { get; set; }

public string TaxCode { get; set; }

public int TaxAmount { get; set; }

public string Month { get; set; }

public Node next;

/// <summary>

/// Node of CitizenTaxData LinkedList

/// </summary>

/// <param name="lastName">Last Name</param>

/// <param name="firstName">First Name</param>

/// <param name="address">Address of the specified Citizen</param>

/// <param name="month">Time of Month when the specified tax was payed</param>

/// <param name="taxCode">TaxCode of the tax</param>

/// <param name="taxAmount">how much tax units did the citizen use</param>

public Node(string lastName, string firstName, string address, string month, string taxCode, int taxAmount)

{

LastName = lastName;

FirstName = firstName;

Address = address;

Month = month;

TaxCode = taxCode;

TaxAmount = taxAmount;

}

/// <summary>

/// ToString format of the node

/// </summary>

/// <returns>string format of CitizenTaxData Node</returns>

public override string ToString()

{

return $"{LastName,-20} {FirstName,-20}|{Address, -20}|{Month, -15}|{TaxCode, -20}|{TaxAmount,10}|";

}

}

}

}

Citizen.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI.WebControls;

namespace Lab02

{

/// <summary>

/// Citizen class object

/// </summary>

public class Citizen

{

private Node head;

private Node tail;

private int count;

public int Count { get { return count; } }

/// <summary>

/// Constructor

/// </summary>

public Citizen()

{

head = null;

tail = null;

count = 0;

}

/// <summary>

/// Adds money to the specified citizen using his FirstName and Last name.

/// If the citizen does not exists, adds him to the LinkedList

/// </summary>

/// <param name="lastName">Last name of the citizen</param>

/// <param name="firstName">FIrst name of the citizn</param>

/// <param name="address">address of the citizen</param>

/// <param name="taxSum">Tax sum to add to his TOTAL</param>

public void AddMoney(string lastName, string firstName, string address, double taxSum)

{

// If Citizen exists, adds sum to his current balance

for (Node curr = head; curr != null; curr = curr.next)

{

if (curr.LastName == lastName && curr.FirstName == firstName)

{

curr.TaxSum += taxSum;

return;

}

}

// If No citizen was found, adds the citizen to Linked List

count++;

if (head == null)

{

head = new Node(lastName, firstName, address);

head.TaxSum = taxSum;

tail = head;

}

else

{

tail.next = new Node(lastName, firstName, address);

tail = tail.next;

tail.TaxSum = taxSum;

}

}

/// <summary>

/// Removes citiznens from linked list who payed belove average taxes

/// </summary>

public void RemoveUnderAverage()

{

if (count == 0)

return;

Node prev = head;

Node curr = head.next;

double average = GetAverage();

while(curr != null)

{

if(curr.TaxSum < average)

{

prev.next = curr.next;

curr = curr.next;

count--;

}

else

{

curr = curr.next;

prev = prev.next;

}

}

RemoveUnderAverageHead(average);

ResetTail();

}

/// <summary>

/// Checks if head/start of linked list is below average. If true removes

/// </summary>

/// <param name="average">Average tax sum of a citizen</param>

private void RemoveUnderAverageHead(double average)

{

Node curr = head;

while(curr.TaxSum < average)

{

curr = curr.next;

count--;

}

head = curr;

}

/// <summary>

/// Resets tail after removing elements

/// </summary>

private void ResetTail()

{

Node curr = head;

if (curr == null)

{

tail = null;

count = 0;

return;

}

while(curr.next != null)

{

curr = curr.next;

}

tail = curr;

}

/// <summary>

/// Removes citizens who did not pay taxes specified month

/// </summary>

/// <param name="taxCode"> Tax Code of the tax</param>

/// <param name="month">Specified Month </param>

/// <param name="data">CitizenTaxData to see what citizen payed what tax at the specified month</param>

public void RemoveWhoDidNotPayTax(string taxCode, string month, CitizenTaxData data)

{

{

if (count == 0)

return;

Node prev = head;

Node curr = head.next;

while (curr != null)

{

// Checks if the citizen has payed Taxes in CitizenTaxData on specified Month

if (curr != null && data.CitizenPayed(taxCode, month, curr.LastName, curr.FirstName) == false)

{

prev.next = curr.next;

curr = curr.next;

count--;

}

else

{

curr = curr.next;

prev = prev.next;

}

}

RemoveWhoDidNotPayTaxHead(taxCode, month, data);

ResetTail();

}

}

/// <summary>

/// Checks first/start/head element of the linked list if the tax was paid

/// </summary>

/// <param name="taxCode">Tax code of the specified tax</param>

/// <param name="month">specified month to check</param>

/// <param name="data">CitizenTaxData to check if the first element of the linked list payed for taxes</param>

private void RemoveWhoDidNotPayTaxHead(string taxCode, string month, CitizenTaxData data)

{

Node curr = head;

// Checks if the citizen has payed Taxes in CitizenTaxData on specified Month

while (curr != null && data.CitizenPayed(taxCode, month, curr.LastName, curr.FirstName) == false)

{

curr = curr.next;

count--;

}

head = curr;

}

public double GetAverage()

{

return count > 0 ? (double)Sum() / count : 0;

}

/// <summary>

/// Sorts LinkedList A-Z using keys: address, last name, first name. Does data swap instead of pointers.

/// </summary>

public void Sort()

{

if (count > 1)

{

for (int i = 0; i < count; i++)

{

Node curr = head;

Node next = head.next;

for (int j = 0; j < count - 1 - i; j++)

{

if (curr.CompareTo(next) > 0)

{

curr.SwapData(next);

}

curr = next;

next = next.next;

}

}

}

}

/// <summary>

/// Returns the total amount citizens payed for taxes

/// </summary>

/// <returns></returns>

public double Sum()

{

Node curr = head;

double sum = 0;

while (curr != null)

{

sum += curr.TaxSum;

curr = curr.next;

}

return sum;

}

/// <summary>

/// Returns citizen in string format

/// </summary>

/// <param name="index"> specified citizen</param>

/// <returns>string format of the citizen for .txt output</returns>

public string ToString(int index)

{

int i = 0;

for (Node curr = head; curr != null; curr = curr.next)

{

if (i == index)

return curr.ToString();

i++;

}

return "";

}

/// <summary>

/// Returns cictizen in TableRow format for the specified citizen

/// </summary>

/// <param name="index">Index of the citizen</param>

/// <returns>TableRow format of the specified citizen</returns>

public TableRow GetRow(int index)

{

int i = 0;

TableRow row = new TableRow();

for (Node curr = head; curr != null; curr = curr.next)

{

if (i == index)

{

row.Cells.Add(TaskUtils.CreateCell(curr.LastName));

row.Cells.Add(TaskUtils.CreateCell(curr.FirstName));

row.Cells.Add(TaskUtils.CreateCell(curr.Address));

row.Cells.Add(TaskUtils.CreateCell(curr.TaxSum.ToString()));

return row;

}

i++;

}

return row;

}

/// <summary>

/// Node class to be used to save every citizen seperately

/// </summary>

internal class Node

{

public string FirstName { get; set; }

public string LastName { get; set; }

public string Address { get; set; }

public double TaxSum { get; set; }

public Node next;

/// <summary>

/// Constructor

/// </summary>

/// <param name="lastName">Last name of the citizen</param>

/// <param name="firstName">First Name of the citizen</param>

/// <param name="address">Address of the citizen</param>

public Node(string lastName, string firstName, string address)

{

LastName = lastName;

FirstName = firstName;

Address = address;

TaxSum = 0;

}

/// <summary>

/// To String override

/// </summary>

/// <returns>stringg format of the citizen</returns>

public override string ToString()

{

return $"{LastName,-20} {FirstName,-20}|{Address,-20}|{TaxSum,10:f}|";

}

/// <summary>

/// Swaps the DATA, keeps the pointers

/// </summary>

/// <param name="other">Other node to be swapped with</param>

public void SwapData(Node other)

{

string lastName = LastName;

string firstName = FirstName;

string address = Address;

double taxSum = TaxSum;

LastName = other.LastName;

FirstName = other.FirstName;

Address = other.Address;

TaxSum = other.TaxSum;

other.LastName = lastName;

other.FirstName = firstName;

other.Address = address;

other.TaxSum = taxSum;

}

/// <summary>

/// Compares to other Node of citizen type

/// </summary>

/// <param name="other"></param>

/// <returns></returns>

public int CompareTo(Node other)

{

int comparison = other.Address.CompareTo(Address);

if (comparison == 0 )

{

comparison = other.LastName.CompareTo(LastName);

if (comparison == 0)

{

comparison = other.FirstName.CompareTo(FirstName);

}

}

return comparison;

}

}

}

}

InOutUtils.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Web;

namespace Lab02

{

/// <summary>

/// Static InOutUtils helper class for Input/Output with files

/// </summary>

public static class InOutUtils

{

/// <summary>

/// Reads Tax Data from txt to Tax class object+

/// </summary>

/// <param name="fileLoc">Location of the data in .txt format</param>

/// <returns>Tax class object</returns>

public static Tax ReadTaxData(string fileLoc)

{

Tax TaxData = new Tax();

string[] lines = File.ReadAllLines(fileLoc);

foreach (string line in lines)

{

string[] elements = line.Split(';');

TaxData.Add(elements[0], elements[1], double.Parse(elements[2]));

}

return TaxData;

}

/// <summary>

/// Creates CitizenTaxData from .txt file

/// </summary>

/// <param name="fileLoc">Location of .txt file</param>

/// <returns>CitizenTaxData class object</returns>

public static CitizenTaxData ReadCitizenTaxData(string fileLoc)

{

CitizenTaxData data = new CitizenTaxData();

string[] lines = File.ReadAllLines(fileLoc);

foreach (string line in lines)

{

string[] elements = line.Split(';');

data.Add(elements[0], elements[1], elements[2], elements[3], elements[4], int.Parse(elements[5]));

}

return data;

}

/// <summary>

/// Appends a header to a file

/// </summary>

/// <param name="fileLoc">Name/location of the file</param>

/// <param name="header">text to be appended</param>

public static void WriteHeader(string fileLoc, string header)

{

using (StreamWriter writer = new StreamWriter(fileLoc, append: true))

{

writer.WriteLine(header);

writer.WriteLine();

}

}

/// <summary>

/// Creates a new or wipes a file

/// </summary>

/// <param name="fileLoc">Location of the file</param>

public static void CreateFile(string fileLoc)

{

using (FileStream fs = new FileStream(fileLoc, FileMode.Create))

new StreamWriter(fs, encoding: System.Text.Encoding.UTF8).Close();

}

/// <summary>

/// Appends CitizenTaxData to a file

/// </summary>

/// <param name="fileLoc">Location/name of the file</param>

/// <param name="data">data to append to the .txt file</param>

/// <param name="header">Header text of the data file</param>

public static void WriteCitizenTaxData(string fileLoc, CitizenTaxData data, string header)

{

using (StreamWriter writer = new StreamWriter(fileLoc, append:true))

{

writer.WriteLine(header);

writer.WriteLine();

writer.WriteLine($"{"LastName",-20} {"FirstName",-20}|{"Address",-20}|{"Month",-15}|{"TaxCode",-20}|{"TaxAmount",10}|");

for (int i = 0; i < data.Count; i++)

{

writer.WriteLine(data.ToString(i));

}

writer.WriteLine();

}

}

/// <summary>

/// appends Citizen class object data to text file

/// </summary>

/// <param name="fileLoc">location/name of the file</param>

/// <param name="data">data to append to the file</param>

/// <param name="header">Header of the file</param>

public static void WriteCitizenData(string fileLoc, Citizen data, string header)

{

using (StreamWriter writer = new StreamWriter(fileLoc, append: true))

{

writer.WriteLine(header);

writer.WriteLine();

writer.WriteLine($"{"LastName",-20} {"FirstName",-20}|{"Address",-20}|{"TaxSum",-10}|");

for (int i = 0; i < data.Count; i++)

{

writer.WriteLine(data.ToString(i));

}

writer.WriteLine();

}

}

/// <summary>

/// Appends Tax data to a .txt file

/// </summary>

/// <param name="fileLoc">Location/name of the file</param>

/// <param name="data">data to append to the .txt file</param>

/// <param name="header">header to be added to the file</param>

public static void WriteTaxData(string fileLoc, Tax data, string header)

{

using (StreamWriter writer = new StreamWriter(fileLoc, append: true))

{

writer.WriteLine(header);

writer.WriteLine();

writer.WriteLine($"{"TaxCode",-20}|{"TaxName",-20}|{"Price",10:2f}|");

for (int i = 0; i < data.Count; i++)

{

writer.WriteLine(data.ToString(i));

}

writer.WriteLine();

}

}

}

}

TaskUtils.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI.WebControls;

namespace Lab02

{

/// <summary>

/// TaskUtils static class for helper functions

/// </summary>

public static class TaskUtils

{

/// <summary>

/// Creates TableCell from text to speed up TableCell creation

/// </summary>

/// <param name="text">string text to add to the table cell</param>

/// <returns>TableCell class object</returns>

public static TableCell CreateCell(string text)

{

TableCell cell = new TableCell();

cell.Text = text;

return cell;

}

}

}

css/styles.css:

body {

color:white;

background:black;

}

td

{

padding:5px;

}

Lab01Form.aspx:

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Lab01Form.aspx.cs" Inherits="Lab02.Lab01Form" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<link rel="stylesheet" runat="server" media="screen" href="~/css/styles.css" />

<title>Lab02 U16</title>

</head>

<body>

<form id="form1" runat="server">

<div id="body">

<asp:Label ID="HeaderLabel" runat="server" Text="LAB02 U16"></asp:Label>

<br />

<br />

<asp:Label ID="Label1" runat="server" Text="Tax Info U16a.txt:"></asp:Label>

<br />

<asp:FileUpload ID="FileUpload1" runat="server" />

<br />

<br />

<asp:Label ID="Label2" runat="server" Text="Every Citizen Tax Data U16b.txt: "></asp:Label>

<br />

<asp:FileUpload ID="FileUpload2" runat="server" />

<br />

<asp:Button ID="DataButton" runat="server" Text="Submit New Data" OnClick="DataButton\_Click" />

<br />

<br />

<asp:Label ID="InitTaxLabel" runat="server" Text="U16a.txt Initial data:"></asp:Label>

<asp:Table ID="InitTaxTable" runat="server">

</asp:Table>

<br />

<asp:Label ID="InitCitizenLabel" runat="server" Text="U16b.txt Initial data:"></asp:Label>

<asp:Table ID="InitCitizenTable" runat="server">

</asp:Table>

<br />

<asp:Panel ID="CalculationsPanel" runat="server">

<asp:Label ID="CitizenTaxLabel" runat="server" Text="All Citizen taxes over the months"></asp:Label>

<asp:Table ID="CitizenTaxTable" runat="server">

</asp:Table>

<br />

<asp:Label ID="AverageTax" runat="server"></asp:Label>

<br />

<asp:Label ID="TotalTaxSum" runat="server"></asp:Label>

<br />

<br />

<asp:Label ID="CitizenTaxLabel0" runat="server" Text="Above Average Tax:"></asp:Label>

<asp:Table ID="AboveAverageTable" runat="server">

</asp:Table>

<br />

<asp:Label ID="FilterData" runat="server" Text="Filtered data:"></asp:Label>

<asp:Table ID="FilterTable" runat="server">

</asp:Table>

<br />

Tax Code:<br />

<asp:TextBox ID="TaxCodeTextBox" runat="server"></asp:TextBox>

<br />

Month:<br />

<asp:TextBox ID="TaxMonthTextBox" runat="server"></asp:TextBox>

<br />

<asp:Button ID="ButtonFilter" runat="server" Text="Submit" OnClick="ButtonFilter\_Click" />

</asp:Panel>

<br />

</div>

</form>

</body>

</html>

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

namespace Lab02

{

public partial class Lab01Form : System.Web.UI.Page

{

private string taxDataInput = @"App\_Data/U16a.txt";

private string citizenDataInput = @"App\_Data/U16b.txt";

private string outputDataPath = @"App\_Data/U16result.txt";

protected void Page\_Load(object sender, EventArgs e)

{

CitizenTaxData citizenTaxData = null;

Tax taxInfo = null;

if (File.Exists(Server.MapPath(taxDataInput)))

{

taxInfo = InOutUtils.ReadTaxData(Server.MapPath(taxDataInput));

InOutUtils.WriteTaxData(Server.MapPath(outputDataPath), taxInfo, "Initial Tax Company Data:");

FillTaxDataTable(taxInfo, InitTaxTable);

}

else

{

InitTaxLabel.Text = "";

}

if (File.Exists(Server.MapPath(citizenDataInput)))

{

citizenTaxData = InOutUtils.ReadCitizenTaxData(Server.MapPath(citizenDataInput));

InOutUtils.WriteCitizenTaxData(Server.MapPath(outputDataPath), citizenTaxData, "Initial Citizen Tax Data:");

FillCitizenTaxDataTable(citizenTaxData, InitCitizenTable);

}

else

{

InitCitizenLabel.Text = "";

}

if (citizenTaxData != null && taxInfo != null)

{

// Reads Initial Data and Outputs the Initial Data To WebForm and to text

InOutUtils.CreateFile(Server.MapPath(outputDataPath));

CitizenCalculations(taxInfo, citizenTaxData);

CheckFiltered(taxInfo, citizenTaxData);

}

else

{

HeaderLabel.Text = "Plaese Upload remaining data files";

CalculationsPanel.Visible = false;

}

}

protected void CitizenCalculations(Tax taxInfo, CitizenTaxData citizenTaxData)

{

Citizen citizensAverage = citizenTaxData.CreateCitizenData(taxInfo); // For Above Average

InOutUtils.WriteCitizenData(Server.MapPath(outputDataPath), citizensAverage, "Tax Sum of all citizens:");

citizensAverage.Sort();

InOutUtils.WriteCitizenData(Server.MapPath(outputDataPath), citizensAverage, "Tax Sum of all citizens SORTED A-Z:");

FillCitizenTable(citizensAverage, CitizenTaxTable);

double sum = citizensAverage.Sum();

double average = citizensAverage.GetAverage();

InOutUtils.WriteHeader(Server.MapPath(outputDataPath), $"All Citizen TOTAL Tax Sum: {sum:f}");

InOutUtils.WriteHeader(Server.MapPath(outputDataPath), $"Average Tax Sum: {average:f}");

AverageTax.Text = $"Average tax per citizen: {average}";

TotalTaxSum.Text = $"Total tax sum: {sum}";

citizensAverage.RemoveUnderAverage();

InOutUtils.WriteCitizenData(Server.MapPath(outputDataPath), citizensAverage, "Citizens who paid above average:");

FillCitizenTable(citizensAverage, AboveAverageTable);

}

protected void CheckFiltered(Tax taxInfo, CitizenTaxData citizenTaxData)

{

if (Session["TaxCode"] != null && Session["Month"] != null)

{

Citizen citizensFiltered = citizenTaxData.CreateCitizenData(taxInfo); // For Filter

citizensFiltered.Sort();

citizensFiltered.RemoveWhoDidNotPayTax(Session["TaxCode"].ToString(), Session["Month"].ToString(), citizenTaxData);

InOutUtils.WriteCitizenData(Server.MapPath(outputDataPath), citizensFiltered, $" Citizens who paid TaxCode: \"{Session["TaxCode"]}\" on Month: \"{Session["Month"]}\"");

FillCitizenTable(citizensFiltered, FilterTable);

}

else

{

FilterData.Text = "No Filter provided";

}

Session["TaxCode"] = null;

Session["Month"] = null;

}

protected void FillCitizenTaxDataTable(CitizenTaxData data, Table table)

{

TableRow headerRow = new TableRow();

headerRow.Cells.Add(TaskUtils.CreateCell("Last Name"));

headerRow.Cells.Add(TaskUtils.CreateCell("First Name"));

headerRow.Cells.Add(TaskUtils.CreateCell("Address"));

headerRow.Cells.Add(TaskUtils.CreateCell("Month"));

headerRow.Cells.Add(TaskUtils.CreateCell("Tax Code"));

headerRow.Cells.Add(TaskUtils.CreateCell("Amount"));

table.Rows.Add(headerRow);

for (int i = 0; i < data.Count; i++)

{

table.Rows.Add(data.GetRow(i));

}

}

protected void FillTaxDataTable(Tax data, Table table)

{

TableRow headerRow = new TableRow();

headerRow.Cells.Add(TaskUtils.CreateCell("Tax Code"));

headerRow.Cells.Add(TaskUtils.CreateCell("Tax Company Name:"));

headerRow.Cells.Add(TaskUtils.CreateCell("Price:"));;

table.Rows.Add(headerRow);

for (int i = 0; i < data.Count; i++)

{

table.Rows.Add(data.GetRow(i));

}

}

protected void FillCitizenTable(Citizen data, Table table)

{

TableRow headerRow = new TableRow();

headerRow.Cells.Add(TaskUtils.CreateCell("Last Name"));

headerRow.Cells.Add(TaskUtils.CreateCell("First Name"));

headerRow.Cells.Add(TaskUtils.CreateCell("Address"));

headerRow.Cells.Add(TaskUtils.CreateCell("Tax Sum"));

table.Rows.Add(headerRow);

for (int i = 0; i < data.Count; i++)

{

table.Rows.Add(data.GetRow(i));

}

}

protected void ButtonFilter\_Click(object sender, EventArgs e)

{

string taxCode = TaxCodeTextBox.Text;

string month = TaxMonthTextBox.Text;

if (month != "" && taxCode != null)

{

Session["TaxCode"] = TaxCodeTextBox.Text;

Session["Month"] = TaxMonthTextBox.Text;

}

Response.Redirect("Lab01Form.aspx");

}

protected void DataButton\_Click(object sender, EventArgs e)

{

if(FileUpload1.HasFile)

{

FileUpload1.SaveAs(Server.MapPath(taxDataInput));

}

if (FileUpload2.HasFile)

{

FileUpload2.SaveAs(Server.MapPath(citizenDataInput));

}

Response.Redirect("Lab01Form.aspx");

}

}

}

## Pradiniai duomenys ir rezultatai

Pradiniai duomenys 1

U16a.txt:

11; Elektra; 0.12

21; Dujos; 0.58

31; Benzinas; 1.78

32; Diezelis; 1.90

U16b.txt:

pavardė1; vardas1; adresas1;1;22;28;

pavardė1; vardas1; adresas1;5;22;20;

pavardė1; vardas1; adresas1;1;32;100;

pavardė1; vardas1; adresas1;2;32;97;

pavardė1; vardas1; adresas1;3;32;63;

pavardė1; vardas1; adresas1;2;22;25;

pavardė1; vardas1; adresas1;3;22;29;

pavardėAA; vardasAA; adresasAA;1;21;13;

pavardėAA; vardasAA; adresasAA;2;21;84;

pavardėAA; vardasAA; adresasAA;3;21;76;

pavardė1; vardas1; adresas1;4;22;39;

pavardė2; vardas2; adresas2;3;31;67;

pavardė2; vardas2; adresas2;4;31;98;

pavardė0; vardas2; adresas2;5;31;125;

pavardė0; vardas0; adresas3;1;11;31;

pavardė1; vardas1; adresas1;4;32;39;

pavardė1; vardas1; adresas1;5;32;20;

pavardė1; vardas1; adresas1;3;11;80;

pavardė1; vardas1; adresas1;4;11;39;

pavardė1; vardas1; adresas1;1;11;120;

pavardė1; vardas1; adresas1;2;11;100;

pavardė1; vardas1; adresas1;5;11;139;

pavardė2; vardas2; adresas2;1;31;31;

pavardė2; vardas2; adresas2;2;31;48;

pavardė0; vardas0; adresas3;2;11;48;

pavardė0; vardas0; adresas3;3;11;67;

pavardė0; vardas0; adresas3;4;11;98;

pavardė0; vardas0; adresas3;5;11;125;

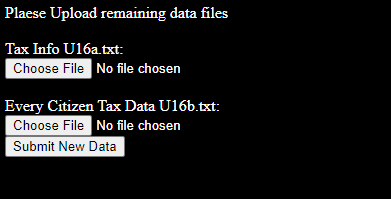
pavardėAA; vardasAA; adresasAA;4;21;8;

pavardėAA; vardasAA; adresasAA;5;21;25;

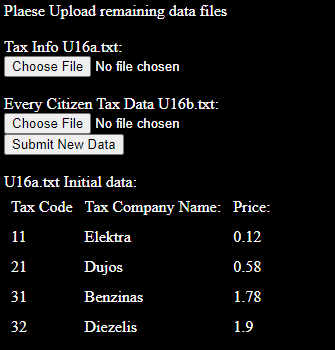
Rezultatai 1:

Vartotojo sąsaja:

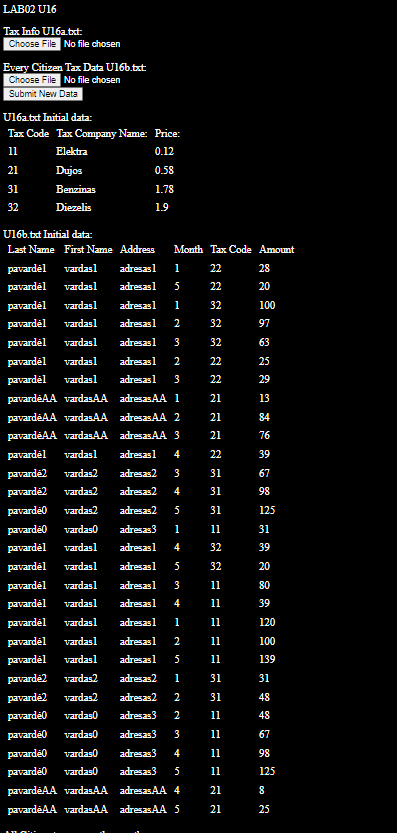
Duombazėje nerado jokio failo:

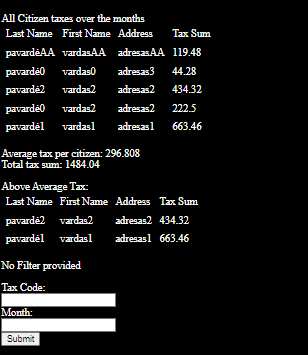


Prikabinamo U16a.txt failą:

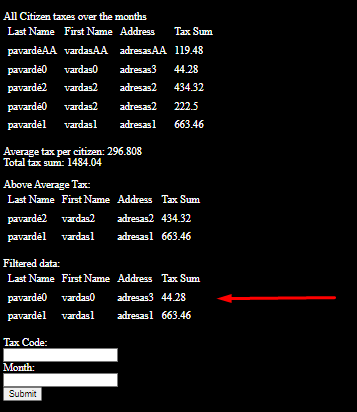


Prikabinome U16b.txt:





Prafiltravome duomenis pagal kodą: „11“, mėnesį: „2“. Prisidėjo lentelė papildoma.



U16result.txt:

Tax Sum of all citizens:

LastName FirstName |Address |TaxSum |

pavardė1 vardas1 | adresas1 | 663.46|

pavardėAA vardasAA | adresasAA | 119.48|

pavardė2 vardas2 | adresas2 | 434.32|

pavardė0 vardas2 | adresas2 | 222.50|

pavardė0 vardas0 | adresas3 | 44.28|

Tax Sum of all citizens SORTED A-Z:

LastName FirstName |Address |TaxSum |

pavardėAA vardasAA | adresasAA | 119.48|

pavardė0 vardas0 | adresas3 | 44.28|

pavardė2 vardas2 | adresas2 | 434.32|

pavardė0 vardas2 | adresas2 | 222.50|

pavardė1 vardas1 | adresas1 | 663.46|

All Citizen TOTAL Tax Sum: 1484.04

Average Tax Sum: 296.81

Citizens who paid above average:

LastName FirstName |Address |TaxSum |

pavardė2 vardas2 | adresas2 | 434.32|

pavardė1 vardas1 | adresas1 | 663.46|

Citizens who paid TaxCode: "11" on Month: "2"

LastName FirstName |Address |TaxSum |

pavardė0 vardas0 | adresas3 | 44.28|

pavardė1 vardas1 | adresas1 | 663.46|

Duomenys 2:

U16a.txt:

VAND; Vanduo; 0.07

KVND; Karštas vanduo; 0.20

LH20; Ledinis Vanduo; 0.10

U16b.txt:

Pavardauskis; Vardenis; Adresatas;Vasaris;Benzinas;28;

Pavardauskis; Vardenis; Adresatas;Vasaris;VAND;14;

Pavardauskis; Vardenis; Adresatas;Kovas;KVND;20;

Pavardauskis; Vardenis; Adresatas;Kovas;LH20;30;

Pavardauskis; Vardenis; Adresatas;Kovas;VAND;15;

Pavardauskis; Vardenis; Adresatas;Balandis;VAND;99;

Tomas; Tomukas; Tomo namas 1;Kovas;VAND;97;

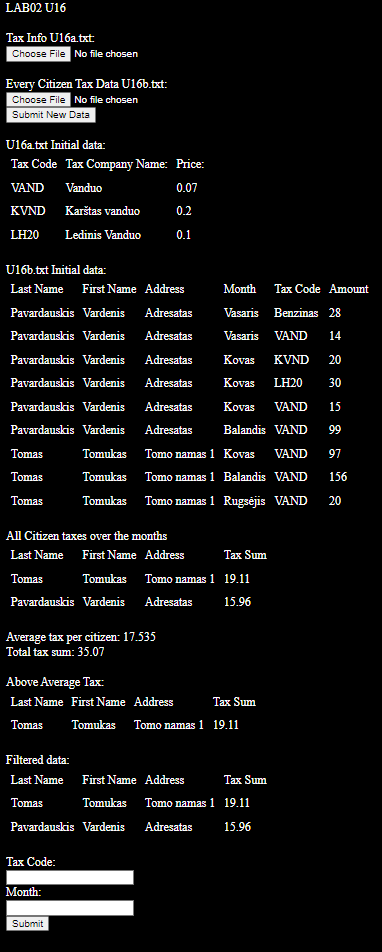
Tomas; Tomukas; Tomo namas 1;Balandis;VAND;156;

Tomas; Tomukas; Tomo namas 1;Rugsėjis;VAND;20;

Rezultatai:

Vartotojo sąsaja

Naudojant mėnesį: Balandis ir mokesčių kodą: VAND:



U16result.txt:

Tax Sum of all citizens:

LastName FirstName |Address |TaxSum |

Pavardauskis Vardenis | Adresatas | 15.96|

Tomas Tomukas | Tomo namas 1 | 19.11|

Tax Sum of all citizens SORTED A-Z:

LastName FirstName |Address |TaxSum |

Tomas Tomukas | Tomo namas 1 | 19.11|

Pavardauskis Vardenis | Adresatas | 15.96|

All Citizen TOTAL Tax Sum: 35.07

Average Tax Sum: 17.54

Citizens who paid above average:

LastName FirstName |Address |TaxSum |

Tomas Tomukas | Tomo namas 1 | 19.11|

Citizens who paid TaxCode: "VAND" on Month: "Balandis"

LastName FirstName |Address |TaxSum |

Tomas Tomukas | Tomo namas 1 | 19.11|

Pavardauskis Vardenis | Adresatas | 15.96|

## Dėstytojo pastabos

# Bendrinės klasės ir testavimas (L3)

## Darbo užduotis

## Grafinės vartotojo sąsajos schema

## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
|  |  |  |
|  |  |  |
|  |  |  |

## Klasių diagrama

## Programos vartotojo vadovas

## Programos tekstas

## Pradiniai duomenys ir rezultatai

## Dėstytojo pastabos

# Polimorfizmas ir išimčių valdymas (L4)

## Darbo užduotis

## Grafinės vartotojo sąsajos schema

## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
|  |  |  |
|  |  |  |
|  |  |  |

## Klasių diagrama

## Programos vartotojo vadovas

## Programos tekstas

## Pradiniai duomenys ir rezultatai

## Dėstytojo pastabos

# Deklaratyvusis programavimas (L5)

## Darbo užduotis

## Grafinės vartotojo sąsajos schema

## Sąsajoje panaudotų komponentų keičiamos savybės

|  |  |  |
| --- | --- | --- |
| Komponentas | Savybė | Reikšmė |
|  |  |  |
|  |  |  |
|  |  |  |

## Klasių diagrama

## Programos vartotojo vadovas

## Programos tekstas

## Pradiniai duomenys ir rezultatai

## Dėstytojo pastabos