KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Objektinio programavimo pagrindai II (P175B502)

Laboratorinio darbo ataskaita

Atliko:

MGDMI42 gr. studentas

Mykolas Logminas

2025 m. gegužės 19 d.

Priėmė:

Gintarė Paškauskaitė

KAUNAS 2025

TURINYS

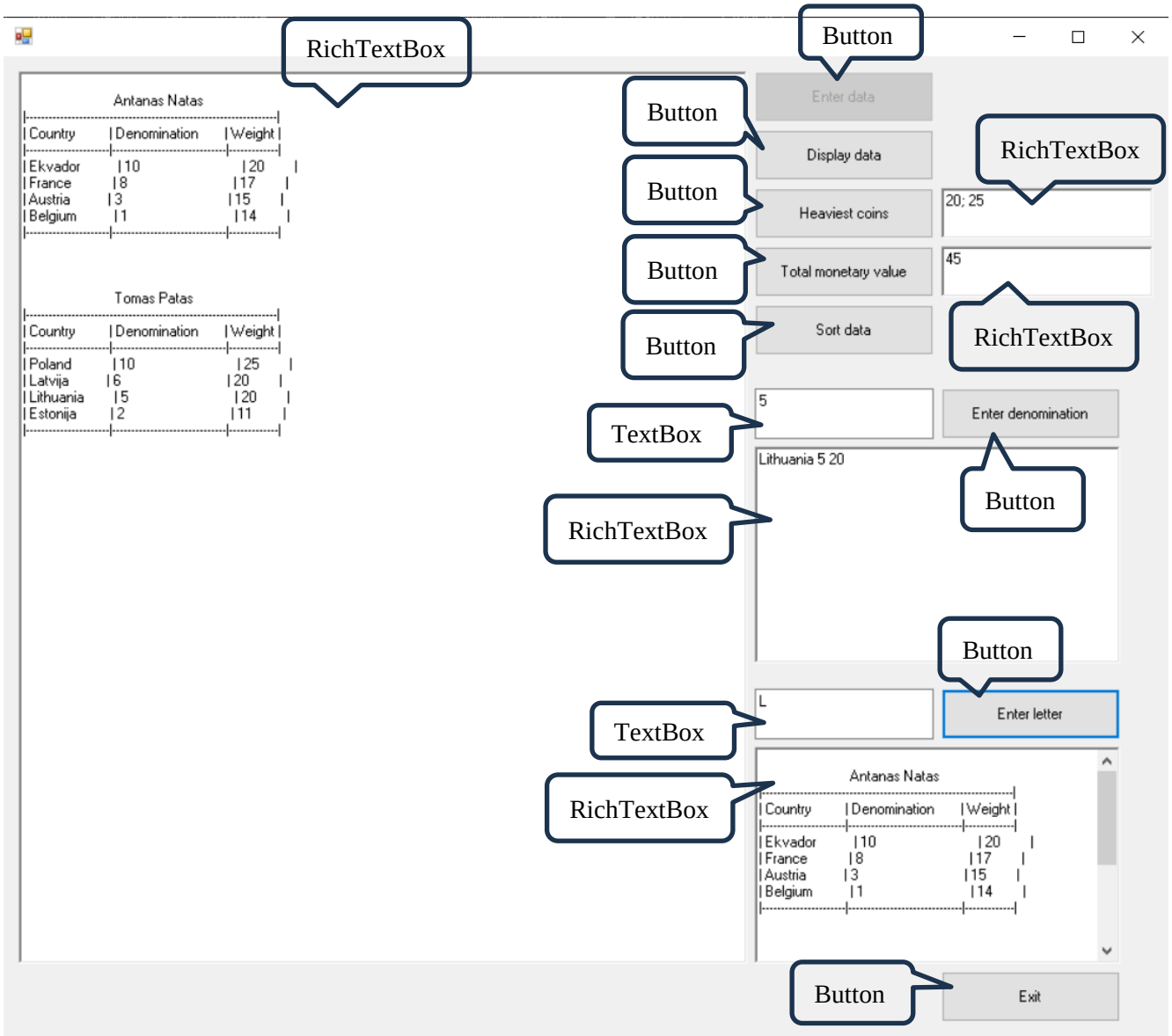
1. Grafinė vartotojo sąsaja ir algoritmų taikymas (L1)

1.1. Darbo užduotis

U1-4. **Monetos**. Monetų kolekcininko turimų monetų duomenys surašyti faile: monetos kilmės šalis, nominalas ir svoris. Pirmoje eilutėje yra kolekcininko vardas ir pavardė. Turime dviejų kolekcininkų duomenis. L1+L2+L4.

• Raskite kiekvieno kolekcininko sunkiausią monetą ir kokia visų kolekcininko monetų bendra piniginė vertė?
• Surašykite į atskirą rinkinį visus abiejų kolekcijų monetų, kurių nominalas yra N, duomenis. Nominalas nurodomas klaviatūra.
• Surikiuokite rezultatų sąrašą monetų svorio mažėjimo tvarka ir nominalą (L2).
• Pašalinkite iš rezultatų sąrašo monetas, kurių šalies pavadinimas prasideda nurodyta klaviatūra raide.

1.2. Grafinės vartotojo sąsajos schema ir paveikslas



1.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
RrichTextBox	Font	Courier New, 10.2pt
	Name	Data
RrichTextBox	Font	Courier New, 10.2pt
	Name	totalMonetaryValueDisplay
RrichTextBox	Font	Courier New, 10.2pt
	Name	denominationDisplay
RrichTextBox	Font	Courier New, 10.2pt
	Name	letterDisplay
RrichTextBox	Font	Courier New, 10.2pt
	Name	hevyestCoinsDisplay
TextBox	Name	denomination
TextBox	Name	letter
Button	Text	Display data
Button	Text	Heaviest coins
Button	Text	Enter data
Button	Text	Total monetary value
Button	Text	Sort sata
Button	Text	Enter denomination
Button	Text	Exit
Button	Text	Enter letter

1.4. Programos vartotojo vadovas

Mygtukas „Enter data“ – Įveda duomenis
Mygtukas „Display data“ – Išveda duomenis lentele į failą rezults.txt ir RichTextBox(data)
Mygtukas „Heaviest coins“ – Suranda iš visų kolekcionierių sunkiausias monetas ir atvaizduoa jas RichTextBox(hevyestCoinsDisplay)
Mygtukas „Totasl monetary value” – Suskaičiuoja visų kolekcionierių, visų monetų bendrą piniginę vertę ir atvaizduoja ją RichTextBox(totalMonetaryValueDisplay)
Mygtukas „Sort data” – Surikiuoja turimus duomenis monetų svorio mažėjimo tvarka ir nominalą ir juos atvaizduoja RichTextBox(data)
Įvedimo laukelis „denomination” – Reikia įvesti nominalą(skaičių)
Mygtukas „Enter denomination” – RichTextBox(denominationDisplay) laukelyje išveda monetas, kurios turi nurodytą nominalą
Įvedimo laukelis „letter” – Reikia įvesti reidę,
Mygtukas „Enter letter” – Pakeičia orginalų sąraša pašalindamas monetas, kurios yra iš šalių prasidedančių nurodyta raide

Duomenų failo pavyzdys:

```
Antanas Natas
Austria 3 15
Belgium 1 14
France 8 17
Ekvador 10 20
```

1.5. Programos tekstas

```
using System;

namespace Laboras_1
{
    /// <summary>
    /// class Coin
    /// </summary>
    internal class Coin
    {
        /// <summary>
        /// Country
        /// </summary>
        public string Country { get; set; }
        /// <summary>
        /// Denomination
        /// </summary>
        public int Denomination { get; set; }
        /// <summary>
        /// Weight
        /// </summary>
        public int Weight { get; set; }
        /// <summary>
        /// Coin's constructor
        /// </summary>
        /// <param name="country">country</param>
        /// <param name="denomination">denomination</param>
        /// <param name="weight">weight</param>
        public Coin(string country, int denomination, int weight)
        {
            Country = country;
            Denomination = denomination;
            Weight = weight;
        }
        /// <summary>
        /// Frmats the line
        /// </summary>
        /// <returns>Formated line</returns>
        public override string ToString()
        {
            return String.Format("| {0,-16} | {1,-28} | {2,-10}|", Country, Denomination,
                                 Weight);
        }
        /// <summary>
        /// Checks if the objects features are equal
        /// </summary>
        /// <param name="obj">object</param>
        /// <returns>TRUE; FALSE</returns>
        public override bool Equals(object obj)
        {
            if (obj is Coin other)
            {
                return this.Country == other.Country &&
                       this.Denomination == other.Denomination &&
                       this.Weight == other.Weight;
            }
            return false;
        }
        /// <summary>
        /// Gets object's HashCode
        /// </summary>
        /// <returns>HashCode</returns>
        public override int GetHashCode()
        {
            return base.GetHashCode();
        }
        /// <summary>
        /// Covered operator "<"
```

```
        /// </summary>
        /// <param name="coin1">coin1</param>
        /// <param name="coin2">coin2</param>
        /// <returns>TRUE; FALSE</returns>
        public static bool operator <(Coin coin1, Coin coin2)
        {
            if (coin1.Weight == coin2.Weight)
            {
                return coin1.Denomination < coin2.Denomination;
            }
            return coin1.Weight < coin2.Weight;
        }
        /// <summary>
        /// Covered operator ">"
        /// </summary>
        /// <param name="coin1">coin1</param>
        /// <param name="coin2">coin2</param>
        /// <returns>TRUE; FALSE</returns>
        public static bool operator >(Coin coin1, Coin coin2)
        {
            if (coin1.Weight == coin2.Weight)
            {
                return coin1.Denomination > coin2.Denomination;
            }
            return coin1.Weight > coin2.Weight;
        }
    }
}


namespace Laboras_1
{
    /// <summary>
    /// Class Collector
    /// </summary>
    internal class Collector
    {
        /// <summary>
        /// Class Collector's fetures
        /// </summary>
        public string Name { get; set; }
        private Coin[] Coins;
        private int CoinCount;
        /// <summary>
        /// Collector's constructor
        /// </summary>
        /// <param name="name">name</param>
        /// <param name="MAX">max copacity</param>
        public Collector(string name, int MAX)
        {
            Name = name;
            Coins = new Coin[MAX];
            CoinCount = 0;
        }
        /// <summary>
        /// Adds coin to the array
        /// </summary>
        /// <param name="coin">coin</param>
        public void AddCoin(Coin coin)
        {
            if (CoinCount < Coins.Length)
            {
                Coins[CoinCount] = coin;
                CoinCount++;
            }
        }
        /// <summary>
        /// Returns coin of the given index
```

```
/// </summary>
/// <param name="i">index</param>
/// <returns>coin of the given index</returns>
public Coin GetCoin(int i) { return Coins[i]; }
/// <summary>
/// Expands array's capacity
/// </summary>
/// <param name="biggerCapacity">bigger capacity</param>
public void ExpandCapacity(int biggerCapacity)
{
    Coin[] newCoins = new Coin[biggerCapacity];
    for (int i = 0; i < CoinCount; i++)
    {
        newCoins[i] = Coins[i];//copying data to the new array with bigger
                               copacity
    }
    Coins = newCoins;
}
/// <summary>
/// Returns arrays capacity
/// </summary>
/// <returns>CoinCount</returns>
public int Quantity() { return CoinCount; }
/// <summary>
/// Finds the heaviest coin
/// </summary>
/// <returns>Heaviest coin's fetures</returns>
public Coin GetHeaviestCoin()
{
    Coin heaviest = Coins[0];
    for (int i = 1; i < Quantity(); i++)
    {
        if (Coins[i] > heaviest)
        {
            heaviest = Coins[i];
        }
    }
    return heaviest;
}
/// <summary>
/// Coulculates monetary value of the coins
/// </summary>
/// <returns>Total monetary value</returns>
public int MonetaryValue()
{
    int sum = 0;
    for (int i = 0; i < Quantity(); i++)
    {
        sum += Coins[i].Denomination;
    }
    return sum;
}
/// <summary>
/// Sorts the array
/// </summary>
public void Sort()
{
    int i = 0;
    bool swicth = true;
    while (swicth)
    {
        swicth = false;
        for (int j = Quantity() - 1; j > i; j--)
        {
            if (Coins[j] > Coins[j - 1])
            {
                swicth = true;
                Coin coin = Coins[j];
```

```
                    Coins[j] = Coins[j - 1];
                    Coins[j - 1] = coin;
                }
            }
            i++;
        }
    }
    /// <summary>
    /// Finds coins indeces which have the give denomination
    /// </summary>
    /// <param name="Denomination">denomination</param>
    /// <returns>indeces</returns>
    public int[] HasDenomination(int Denomination)
    {
        int count = 0;
        for (int i = 0; i < CoinCount; i++)
        {
            if (Coins[i].Denomination == Denomination)
            {
                count++;
            }
        }
        int[] approvedIndices = new int[count];
        int index = 0;
        for (int i = 0; i < CoinCount; i++)
        {
            if (Coins[i].Denomination == Denomination)
            {
                approvedIndices[index] = i;
                index++;
            }
        }
        return approvedIndices;
    }
    /// <summary>
    /// Removes coins that are from the country which starts by the give letter
    /// </summary>
    /// <param name="letter">letter</param>
    public void RemoveCoinsByCountry(string letter)
    {
        int removeCount = 0;
        for (int i = 0; i < CoinCount; i++)
        {
            if (Coins[i].Country.ToUpper().StartsWith(letter))
            {
                removeCount++;
            }
            else if (removeCount > 0)
            {
                Coins[i - removeCount] = Coins[i];
            }
        }
        CoinCount -= removeCount;
    }
}
}
```

```
using System;
using System.IO;
using System.Windows.Forms;
//Monetų kolekcininko turimų monetų duomenys surašyti faile: monetos kilmės šalis,
nominalas ir svoris. Pirmoje eilutėje
//yra kolekcininko vardas ir pavardė. Turime dviejų kolekcininkų duomenis.
//• Raskite kiekvieno kolekcininko sunkiausią monetą ir kokia visų kolekcininko monetų
bendra piniginė vertė?
```

```
//• Surašykite į atskirą rinkinį visus abiejų kolekcijų monetų, kurių nominalas yra N,
duomenis. Nominalas
//nurodomas klaviatūra.
//• Surikiuokite rezultatų sąrašą monetų svorio mažėjimo tvarka ir nominalą (L2).
//• Pašalinkite iš rezultatų sąrašo monetas, kurių šalies pavadinimas prasideda nurodyta
klaviatūra raide.
namespace Laboras_1
{
    public partial class Form1 : Form
    {
        const string DF1 = "..\\..\\data1.txt";
        const string DF2 = "..\\..\\data2.txt";
        const string REZ = "..\\..\\rez.txt";
        const string TABLE = "|---------------------------------------------------------
                              --------|\r\n| Country         | Denomination       | Weight
                              |\r\n|-----------------------|----------------------------
                              |-------------|";
        Collector collector1, collector2;
        public Form1()
        {
            InitializeComponent();
            displayData.Enabled = false;
            HevyestCoins.Enabled = false;
            totalMonetaryValue.Enabled = false;
            sort.Enabled = false;
            enterDenomination.Enabled = false;
            enterLetter.Enabled = false;
        }
        /// <summary>
        /// Scans data
        /// </summary>
        /// <param name="DF">data file</param>
        /// <returns></returns>
        static Collector DataScanning(string DF)
        {
            using (StreamReader df = new StreamReader(DF))
            {
                string nameSurn = df.ReadLine();
                int capacity = 10;
                Collector collector = new Collector(nameSurn, capacity);
                int maxCoins = 0;
                string line;
                while ((line = df.ReadLine()) != null)
                {
                    string[] parts = line.Split(' ');
                    string country = parts[0];
                    int denomination = int.Parse(parts[1]);
                    int weight = int.Parse(parts[2]);
                    Coin coin = new Coin(country, denomination, weight);
                    if (maxCoins >= capacity)
                    {
                        capacity *= 2;
                        collector.ExpandCapacity(capacity);
                    }
                    collector.AddCoin(coin);
                    maxCoins++;
                }
                return collector;
            }
        }
        private void Form1_Load(object sender, EventArgs e)
        {

        }
        private void textBox2_TextChanged(object sender, EventArgs e)
        {

        }
```

```
/// <summary>
/// Exit button
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Exit_Click(object sender, EventArgs e)
{
    Close();
}
/// <summary>
/// Enter data button
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void EnterData_Click(object sender, EventArgs e)
{
    string df1 = File.ReadAllText(DF1), df2 = File.ReadAllText(DF2);
    data.Clear();
    data.Text = df1 + "\n\n" + df2 + "\n\n";
    collector1 = DataScanning(DF1);
    collector2 = DataScanning(DF2);
    displayData.Enabled = true;
    HevyestCoins.Enabled = true;
    totalMonetaryValue.Enabled = true;
    sort.Enabled = true;
    enterDenomination.Enabled = true;
    enterLetter.Enabled = true;
    EnterData.Enabled = false;
}
/// <summary>
/// Data display button
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void displayData_Click(object sender, EventArgs e)
{
    File.WriteAllText(REZ, string.Empty);
    DisplayData(REZ, collector1, collector1.Name);
    DisplayData(REZ, collector2, collector2.Name);
    string output = File.ReadAllText(REZ);
    data.Clear();
    data.Text = output;
}
/// <summary>
/// Hevyest coin button
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void HevyestCoins_Click(object sender, EventArgs e)
{
    Coin heaviestCoin1 = collector1.GetHeaviestCoin();
    Coin heaviestCoin2 = collector2.GetHeaviestCoin();
    hevyestCoinsDisplay.Clear();
    if (heaviestCoin1 != null && heaviestCoin2 != null)
    {
        hevyestCoinsDisplay.Text = heaviestCoin1.Weight + "; " +
                                   heaviestCoin2.Weight;
    }
    if (heaviestCoin1 != null && heaviestCoin1 == null)
    {
        hevyestCoinsDisplay.Text = heaviestCoin1.Weight + "; " + "No data";
    }
    if (heaviestCoin1 == null && heaviestCoin1 != null)
    {
        hevyestCoinsDisplay.Text = "No data" + "; " + heaviestCoin2.Weight;
    }
    if (heaviestCoin1 == null && heaviestCoin1 == null)
    {
```

```
            hevyestCoinsDisplay.Text = "No data";
        }
    }
    /// <summary>
    /// Monetary value button
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void totalMonetaryValue_Click(object sender, EventArgs e)
    {
        int sum = collector1.MonetaryValue() + collector2.MonetaryValue();
        totalMonetaryValueDisplay.Clear();
        totalMonetaryValueDisplay.Text = $"{sum}";
    }
    /// <summary>
    /// Sort button
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void sort_Click(object sender, EventArgs e)
    {
        collector1.Sort();
        collector2.Sort();
        File.WriteAllText(REZ, string.Empty);
        DisplayData(REZ, collector1, collector1.Name);
        DisplayData(REZ, collector2, collector2.Name);
        string output = File.ReadAllText(REZ);
        data.Clear();
        data.Text = output;
    }
    /// <summary>
    /// Enter denomination button
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void enterDenomination_Click(object sender, EventArgs e)
    {
        string denomination = this.denomination.Text;
        int[] indices1 = collector1.HasDenomination(int.Parse(denomination));
        int[] indices2 = collector2.HasDenomination(int.Parse(denomination));
        denominationDisplay.Clear();
        bool found = false;
        if (indices1.Length > 0)
        {
            for (int i = 0; i < indices1.Length; i++)
            {
                Coin coin = collector1.GetCoin(indices1[i]);
                denominationDisplay.AppendText($"{coin.Country} {coin.Denomination}
                {coin.Weight}\n");
                found = true;
            }
        }
        if (indices2.Length > 0)
        {
            for (int i = 0; i < indices2.Length; i++)
            {
                Coin coin = collector2.GetCoin(indices2[i]);
                denominationDisplay.AppendText($"{coin.Country} {coin.Denomination}
               {coin.Weight}\n");
                found = true;
            }
        }
        if (!found)
        {
            denominationDisplay.Text = "Not found";
        }
    }
    /// <summary>
```

```
        /// Enter letter button
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        private void enterLetter_Click(object sender, EventArgs e)
        {
            string letter = this.letter.Text.ToUpper();
            collector1.RemoveCoinsByCountry(letter);
            collector2.RemoveCoinsByCountry(letter);
            File.WriteAllText(REZ, string.Empty);
            DisplayData(REZ, collector1, collector1.Name);
            DisplayData(REZ, collector2, collector2.Name);
            string output = File.ReadAllText(REZ);
            letterDisplay.Clear();
            letterDisplay.Text = output;
        }
        /// <summary>
        /// Displays the data
        /// </summary>
        /// <param name="REZ">rezults file</param>
        /// <param name="collector">collector</param>
        /// <param name="title">title</param>
        static void DisplayData(string REZ, Collector collector, string title)
        {
            using (var rez = new StreamWriter(File.Open(REZ, FileMode.Append)))
            {
                rez.WriteLine("\n " + "                        " + title);
                rez.WriteLine(TABLE);
                if (collector != null && collector.Quantity() > 0)
                {
                    for (int i = 0; i < collector.Quantity(); i++)
                    {
                        Coin coin = collector.GetCoin(i);
                        rez.WriteLine(coin.ToString());
                    }
                }
                else
                {
                    rez.WriteLine("|                              No data
                                  |");
                }
                rez.WriteLine("|---------------------|-----------------------------|---
                              ----------|\n\n");
            }
        }
    }
}


using Laboras_1;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Laboras1_testai_
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void Test_AddCoin()
        {
            Collector collector = new Collector("TestCollector", 5);
            Coin coin = new Coin("USA", 1, 5);

            collector.AddCoin(coin);

            Assert.AreEqual(1, collector.Quantity());
            Assert.AreEqual(coin, collector.GetCoin(0));
        }
```

```
[TestMethod]
public void Test_GetCoin()
{
    Collector collector = new Collector("TestCollector", 5);
    Coin coin = new Coin("USA", 1, 5);
    collector.AddCoin(coin);

    Coin result = collector.GetCoin(0);

    Assert.AreEqual(coin, result);
}

[TestMethod]
public void Test_ExpandCapacity()
{
    Collector collector = new Collector("TestCollector", 2);
    collector.AddCoin(new Coin("USA", 1, 5));
    collector.AddCoin(new Coin("Canada", 2, 10));

    collector.ExpandCapacity(5);
    collector.AddCoin(new Coin("Germany", 5, 15));

    Assert.AreEqual(3, collector.Quantity());
}

[TestMethod]
public void Test_Quantity()
{
    Collector collector = new Collector("TestCollector", 10);
    collector.AddCoin(new Coin("France", 2, 6));
    collector.AddCoin(new Coin("Japan", 5, 8));

    Assert.AreEqual(2, collector.Quantity());
}

[TestMethod]
public void Test_GetHeaviestCoin()
{
    Collector collector = new Collector("TestCollector", 5);
    Coin coin1 = new Coin("USA", 1, 5);
    Coin coin2 = new Coin("Canada", 2, 15);
    Coin coin3 = new Coin("UK", 5, 10);
    collector.AddCoin(coin1);
    collector.AddCoin(coin2);
    collector.AddCoin(coin3);

    Coin heaviest = collector.GetHeaviestCoin();

    Assert.AreEqual(coin2, heaviest);
}

[TestMethod]
public void Test_MonetaryValue()
{
    Collector collector = new Collector("TestCollector", 10);
    collector.AddCoin(new Coin("USA", 1, 5));
    collector.AddCoin(new Coin("Canada", 2, 10));

    int totalValue = collector.MonetaryValue();

    Assert.AreEqual(3, totalValue);
}

[TestMethod]
public void Test_Sort()
{
    Collector collector = new Collector("TestCollector", 5);
```

```
            Coin coin1 = new Coin("USA", 1, 5);
            Coin coin2 = new Coin("Canada", 2, 15);
            Coin coin3 = new Coin("UK", 5, 10);
            collector.AddCoin(coin2);
            collector.AddCoin(coin1);
            collector.AddCoin(coin3);

            collector.Sort();

            Assert.AreEqual(coin2, collector.GetCoin(0));
            Assert.AreEqual(coin3, collector.GetCoin(1));
            Assert.AreEqual(coin1, collector.GetCoin(2));
        }

        [TestMethod]
        public void Test_HasDenomination()
        {
            Collector collector = new Collector("TestCollector", 10);
            collector.AddCoin(new Coin("USA", 1, 5));
            collector.AddCoin(new Coin("Canada", 2, 10));
            collector.AddCoin(new Coin("Mexico", 1, 8));

            int[] indices = collector.HasDenomination(1);

            Assert.AreEqual(2, indices.Length);
            Assert.AreEqual(0, indices[0]);
            Assert.AreEqual(2, indices[1]);
        }

        [TestMethod]
        public void Test_RemoveCoinsByCountry()
        {
            Collector collector = new Collector("TestCollector", 10);
            collector.AddCoin(new Coin("Spain", 1, 5));
            collector.AddCoin(new Coin("Sweden", 2, 10));
            collector.AddCoin(new Coin("Germany", 5, 15));

            collector.RemoveCoinsByCountry("S");

            Assert.AreEqual(1, collector.Quantity());
            Assert.AreEqual("Germany", collector.GetCoin(0).Country);
        }
    }

    [TestClass]
    public class UnitTest2
    {
        [TestMethod]
        public void Test_CoinComparison_Weight()
        {
            Coin coin1 = new Coin("USA", 1, 5);
            Coin coin2 = new Coin("Canada", 2, 10);

            Assert.IsTrue(coin1 < coin2);
            Assert.IsTrue(coin2 > coin1);
        }

        [TestMethod]
        public void Test_CoinComparison_SameWeightDifferentDenomination()
        {
            Coin coin1 = new Coin("USA", 1, 5);
            Coin coin2 = new Coin("Canada", 2, 5);

            Assert.IsTrue(coin1 < coin2);
            Assert.IsTrue(coin2 > coin1);
        }

        [TestMethod]
```

```
        public void Test_CoinEquality()
        {
            Coin coin1 = new Coin("USA", 1, 5);
            Coin coin2 = new Coin("USA", 1, 5);
            Coin coin3 = new Coin("Canada", 2, 10);

            Assert.IsTrue(coin1.Equals(coin2));
            Assert.IsFalse(coin1.Equals(coin3));
        }

        [TestMethod]
        public void Test_CoinToString()
        {
            Coin coin = new Coin("USA", 1, 5);
            string expectedString = "| USA                | 1                                  |
                                    5          |";

            Assert.AreEqual(expectedString, coin.ToString());
        }
    }
}
```

1.6. Pradiniai duomenys ir rezultatai

data1.txt – normal data

```
Antanas Natas
Austria 3 15
Belgium 1 14
France 8 17
Ekvador 10 20
```

data2.txt – normal data

```
Tomas Patas
Lithuania 5 20
Latvija 6 20
Estonija 2 11
Poland 10 25
```

rezults file

```
                      Antanas Natas
|-------------------------------------------------------------|
| Country         | Denomination       | Weight |
|-----------------------|-----------------------------|-------------|
| Austria           | 3                              | 15         |
| Belgium           | 1                              | 14         |
| France            | 8                              | 17         |
| Ekvador           | 10                             | 20         |
|-----------------------|-----------------------------|-------------|



                      Tomas Patas
|-------------------------------------------------------------|
| Country         | Denomination       | Weight |
|-----------------------|-----------------------------|-------------|
| Lithuania         | 5                              | 20         |
| Latvija           | 6                              | 20         |
| Estonija          | 2                              | 11         |
| Poland            | 10                             | 25         |
|-----------------------|-----------------------------|-------------|
```

data1.txt – no data

```
Antanas Natas
```

data2.txt – no data

```
Tomas Patas
```

rezults file

```
                        Antanas Natas
|-------------------------------------------------------------------|
| Country         | Denomination       | Weight |
|-----------------------|------------------------------|-------------|
|                            No data                          |
|-----------------------|------------------------------|-------------|



                        Tomas Patas
|-------------------------------------------------------------------|
| Country         | Denomination       | Weight |
|-----------------------|------------------------------|-------------|
|                            No data                          |
|-----------------------|------------------------------|-------------|
```

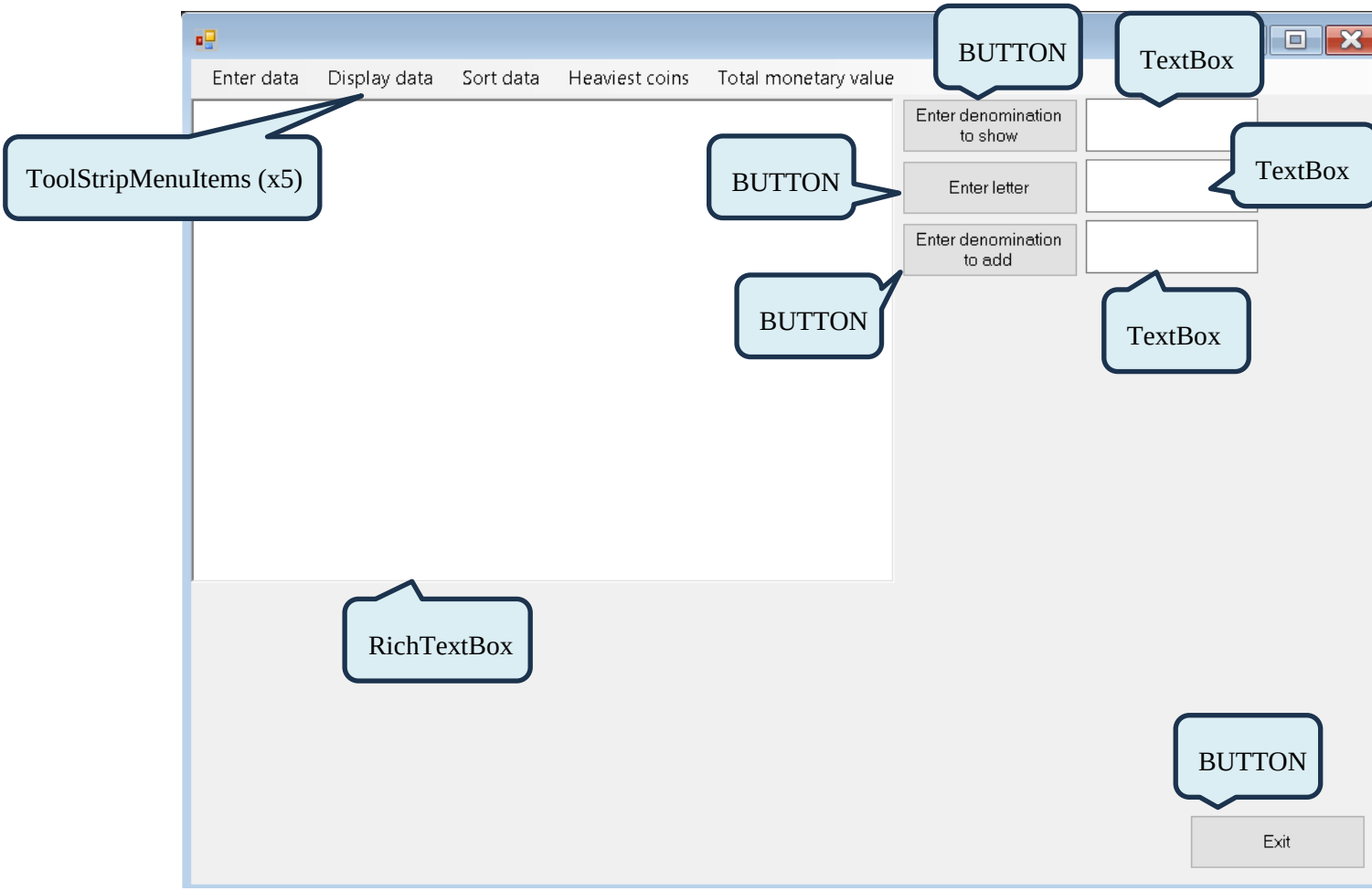
1.7. Dėstytojo pastabos

2. Dinaminis masyvas (L2)

2.1. Darbo užduotis

U1-4. Monetos. Monetų kolekcininko turimų monetų duomenys surašyti faile: monetos kilmės šalis, nominalas ir svoris. Pirmoje eilutėje yra kolekcininko vardas ir pavardė. Turime dviejų kolekcininkų duomenis. L1+L2+L4. • Raskite kiekvieno kolekcininko sunkiausią monetą ir kokia visų kolekcininko monetų bendra piniginė vertė? • Surašykite į atskirą rinkinį visus abiejų kolekcijų monetų, kurių nominalas yra N, duomenis. Nominalas nurodomas klaviatūra. • Surikiuokite rezultatų sąrašą monetų svorio mažėjimo tvarka ir nominalą (L2). • Pašalinkite iš rezultatų sąrašo monetas, kurių šalies pavadinimas prasideda nurodyta klaviatūra raide. L2 papildymas. • Papildykite surikiuotą rezultatų sąrašą naujo kolekcininko monetomis, kurių nominalas ne didesnis už nurodytą klaviatūra. Duomenys yra atskirame faile. Pirmoje eilutėje yra kolekcininko vardas ir pavardė.

2.2. Grafinės vartotojo sąsajos schema ir paveikslas



2.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
Button	Name	Exit
Button	Name	Enter letter
Button	Name	Enter denomination to show
Button	Name	Enter denomination to add
RichTextBox	Font	Courier New, 12pt
	Name	display

TextBox	Name	textDenominationToAdd
TextBox	Name	textEnterLetter
TextBox	Name	textDenominationToShow
ToolStripMenuItem	Name	enterDataToolStripMenuItem
ToolStripMenuItem	Name	displayDataToolStripMenuItem
ToolStripMenuItem	Name	sortDataToolStripMenuItem
ToolStripMenuItem	Name	heaviestCoinsToolStripMenuItem
ToolStripMenuItem	Name	totalMonetaryValueToolStripMenuItem

2.4. Programos vartotojo vadovas

Enter data – įveda
Display data – pavaizduoja duomenis
Sort data – surikiuoja duomenis
Heaviest coins – randa sunkiausias monetas
Total monetary value – suskaičiuoja bendrą monetų vertę kiekvieno kolekcionieriaus
Enter denomination to show – randa visas monetas, su nurodytu nominalu
Enter letter – pašalina monetas, kurios yra iš šalių, kurios prasideda nurodyta raide
Enter denomination to add – prideda į pirmojo kolekcionieriaus surikiuotą lentelę naujo kolekcionieriaus monetas, kurios turi ne didesnį už nurodytą nominąlą
Exit – uždaro programą

2.5. Programos tekstas

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Laboras_2
{
    /// <summary>
    /// Coin class
    /// </summary>
    class Coin
    {
        /// <summary>
        /// Country
        /// </summary>
        public string Country { get; set; }
        /// <summary>
        /// Weight
        /// </summary>
        public double Weight { get; set; }
        /// <summary>
        /// Denomination
        /// </summary>
        public double Denomination { get; set; }
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="country">country</param>
        /// <param name="weight">weight</param>
        /// <param name="denomination">denomination</param>
        public Coin(string country, double weight, double denomination)
        {
            Country = country;
            Weight = weight;
```

```
            Denomination = denomination;
        }
        /// <summary>
        /// Method ToString
        /// </summary>
        /// <returns>Formated line</returns>
        public override string ToString()
        {
            return string.Format("| {0,-11} | {1,-12} | {2,-6} |", Country, Denomination,
                                 Weight);
        }
        public string ToStringCSV()
        {
            return string.Format("{0,-11}, {1,-12}, {2,-6}", Country, Denomination,
                                 Weight);
        }
    }
}
```

```
using System.Collections.Generic;

namespace Laboras_2
{
    /// <summary>
    /// Collector class
    /// </summary>
    internal class Collector
    {
        /// <summary>
        /// Name and Surname
        /// </summary>
        public string NameSur { get; set; }
        /// <summary>
        /// Coins list
        /// </summary>
        public List<Coin> Coins { get; set; }
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="nameSur">Name and Surname</param>
        /// <param name="coins">coins list</param>
        public Collector(string nameSur, List<Coin> coins)
        {
            NameSur = nameSur;
            Coins = coins;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Windows.Forms;
namespace Laboras_2
{
    public partial class Form1 : Form
    {
        const string TABLE1 = "|-----------------------------------|\n" +
                              "| Country     | Denomination | Weight |\n" +
                              "|-----------------------------------|\n";
        const string TABLE2 = "Country, Denomination, Weight";

         private List<Collector> collector1;
         private List<Collector> collector2;
```

```
 private List<Collector> sortedCollector1;
 private string rez;
 public Form1()
{
    InitializeComponent();
    displayDataToolStripMenuItem.Enabled = false;
    sortDataToolStripMenuItem.Enabled = false;
    heaviestCoinsToolStripMenuItem.Enabled = false;
    totalMonetaryValueToolStripMenuItem.Enabled = false;
    enterDenominationToShow.Enabled = false;
    enterLetter.Enabled = false;
    enterDenominationToAdd.Enabled = false;
}
/// <summary>
/// Enters data
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void enterDataToolStripMenuItem_Click(object sender, System.EventArgs e)
{
    string dataFile1 = string.Empty, dataFile2 = string.Empty;
    OpenFileDialog openFileDialog1 = new OpenFileDialog
    {
        Filter = "All files (*.*)|*.*",
        Title = "Choose the first data file"
    };
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        string df1 = openFileDialog1.FileName;
        dataFile1 = File.ReadAllText(df1);
        collector1 = ReadData(df1);
    }
    OpenFileDialog openFileDialog2 = new OpenFileDialog
    {
        Filter = "All files (*.*)|*.*",
        Title = "Choose the second data file"
    };
    if (openFileDialog2.ShowDialog() == DialogResult.OK)
    {
        string df2 = openFileDialog2.FileName;
        dataFile2 = File.ReadAllText(df2);
        collector2 = ReadData(df2);
    }
    if (!string.IsNullOrEmpty(dataFile1) && !string.IsNullOrEmpty(dataFile2))
    {
        display.Text = dataFile1 + "\n\n" + dataFile2;
        displayDataToolStripMenuItem.Enabled = true;
        sortDataToolStripMenuItem.Enabled = true;
        heaviestCoinsToolStripMenuItem.Enabled = true;
        totalMonetaryValueToolStripMenuItem.Enabled = true;
        enterDenominationToShow.Enabled = true;
        enterLetter.Enabled = true;
        enterDenominationToAdd.Enabled = true;
        enterDataToolStripMenuItem.Enabled = false;
    }
}
/// <summary>
/// Displays data
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void displayDataToolStripMenuItem_Click(object sender, EventArgs e)
{
    SaveFileDialog saveFileDialog1 = new SaveFileDialog
    {
        Filter = "All files (*.*)|*.*",
        Title = "Choose results data file"
    };
```

```
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        rez = saveFileDialog1.FileName;
        if (File.Exists(rez))
            File.Delete(rez);
        string output = Print(rez, collector1, collector1[0].NameSur) +
                        Print(rez, collector2, collector2[0].NameSur);
        display.Text = output;
    }
}
/// <summary>
/// Sorts data
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void sortDataToolStripMenuItem_Click(object sender, EventArgs e)
{
    using (StreamWriter rf = new StreamWriter(rez, true))
    {
        rf.WriteLine("\nSORTED DATA:");
    }
    Sort(collector1);
    Sort(collector2);
    sortedCollector1 = collector1;
    display.Text += "\nSORTED DATA:\n";
    display.Text += Print(rez, collector1, collector1[0].NameSur);
    display.Text += Print(rez, collector2, collector2[0].NameSur);
}
/// <summary>
/// Finds the heaviest coins
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void heaviestCoinsToolStripMenuItem_Click(object sender, EventArgs e)
{
    string heaviestCoinsInfo = HeaviestCoins(collector1) + "\n\n" +
                HeaviestCoins(collector2);
    string output = $"\nHeaviet coins:\n{heaviestCoinsInfo}";
    display.Text += output;
    using (StreamWriter rf = new StreamWriter(rez, true))
    {
        rf.WriteLine(output);
    }
    heaviestCoinsToolStripMenuItem.Enabled = false;
}
/// <summary>
/// Coulculates total monetary value
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void totalMonetaryValueToolStripMenuItem_Click(object sender, EventArgs
                    e)
{
    string output = $"\n\nTotal monetary value:\n{collector1[0].Coins.Sum(coin =>
            coin.Denomination)}\n{collector2[0].Coins.Sum(coin =>
            coin.Denomination)}";
    display.Text += output;
    using (StreamWriter rf = new StreamWriter(rez, true))
    {
        rf.WriteLine(output);
    }
    totalMonetaryValueToolStripMenuItem.Enabled = false;
}
/// <summary>
/// Finds coins with specific denomination
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
```

```
private void enterDenominationToShow_Click(object sender, EventArgs e)
{
    int denomination = int.Parse(textDenominationToShow.Text);
    string data = CoinsWithDenomination(denomination, collector1) +
           CoinsWithDenomination(denomination, collector2);
    string output;
    if (string.IsNullOrEmpty(data))
    {
        output = $"\n\nCoins with denomination {denomination}:\nNot Found\n";
    }
    else
    {
        output = $"\n\nCoins with denomination
       {denomination}:\n{CoinsWithDenomination(denomination,
       collector1)}\n{CoinsWithDenomination(denomination, collector2)}\n";
    }
    display.Text += output;
    using (StreamWriter rf = new StreamWriter(rez, true))
    {
        rf.WriteLine(output);
    }
}
/// <summary>
/// Removes coins thats are from a country starting with the specific letter
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void enterLetter_Click(object sender, EventArgs e)
{
    char letter = char.ToUpper(textEnterLetter.Text[0]);
    RemoveCoinsStartingWithLetter(collector1, letter);
    RemoveCoinsStartingWithLetter(collector2, letter);
    using (StreamWriter rf = new StreamWriter(rez, true))
    {
        rf.WriteLine("Data with removed coins:\n\n");
    }
    string output = Print(rez, collector1, collector1[0].NameSur) + Print(rez,
           collector2, collector2[0].NameSur);
    display.Text += "Data with removed coins:\n\n" + output;
}
/// <summary>
/// Adds coins from a new collector that are not higher than noted denomination
           to the first collector's sorted table
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void enterDenominationToAdd_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog = new OpenFileDialog
    {
         Filter = "All files (*.*)|*.*",
         Title = "Choose a collector's data file"
    };
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
         if (sortedCollector1 == null || sortedCollector1.Count() == 0)
         {
             using (StreamWriter rf = new StreamWriter(rez, true))
             {
                 rf.WriteLine("\n\nSort the current data first!!!\n");
             }
             display.Text += "\n\nSort the current data first!!!\n";
         }
         else
         {
             int maxDenomination = int.Parse(textDenominationToAdd.Text);
             List<Collector> newCollector = ReadData(openFileDialog.FileName);
```

```
                List<Coin> filteredCoins = FilterByDenomination(newCollector,
                                                maxDenomination);
                InsertCoinsIntoSortedList(sortedCollector1[0].Coins, filteredCoins);
                using (StreamWriter rf = new StreamWriter(rez, true))
                {
                    rf.WriteLine("\n\nData with added coins:\n");
                }
                string updatedData = Print(rez, sortedCollector1,
                                    sortedCollector1[0].NameSur + " su papildytom
                                    monetomis");
                display.Text += "\n\nData with added coins:\n" + updatedData;
            }
        }
    }
    /// <summary>
    /// Exits the program
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void exit_Click(object sender, EventArgs e)
    {
        Close();
    }
    /// <summary>
    /// Reads data
    /// </summary>
    /// <param name="DF">data file</param>
    /// <returns></returns>
    static List<Collector> ReadData(string DF)
    {
        List<Collector> collectors = new List<Collector>();
        using (StreamReader df = new StreamReader(DF))
        {
            string line;
            line = df.ReadLine();
            string name = line;
            List<Coin> coins = new List<Coin>();
            while ((line = df.ReadLine()) != null)
            {
                string[] parts = line.Split(' ');
                string country = parts[0];
                double weight = double.Parse(parts[1]);
                double denomination = double.Parse(parts[2]);
                Coin coin = new Coin(country, denomination, weight);
                coins.Add(coin);
            }
            Collector collector = new Collector(name, coins);
            collectors.Add(collector);
        }
        return collectors;
    }
    /// <summary>
    /// Prints data
    /// </summary>
    /// <param name="REZ">rezults file</param>
    /// <param name="collectors">collectors list</param>
    /// <param name="title">title</param>
    /// <returns>data</returns>
    static string Print(string REZ, List<Collector> collectors, string title)
     {
         int tableWidth1 = TABLE1.Split('\n')[0].Length;
         int tableWidth2 = TABLE2.Length;
         int titlePadding1 = Math.Max(0, (tableWidth1 - title.Length) / 2);
         int titlePadding2 = Math.Max(0, (tableWidth2 - title.Length) / 2);
         string centeredTitle1 = new string(' ', titlePadding1) + title;
         string centeredTitle2 = new string(' ', titlePadding2) + title;
         string output = $"\n{centeredTitle1}\n{TABLE1}";
         using (StreamWriter rez = new StreamWriter(File.Open(REZ, FileMode.Append)))
```

```
        {
            rez.WriteLine($"\n{centeredTitle2}");
            rez.WriteLine(TABLE2);
            foreach (Collector collector in collectors)
            {
                foreach (Coin coin in collector.Coins)
                {
                    rez.WriteLine(coin.ToStringCSV());
                    output += coin.ToString() + "\n";
                }
            }
            output += "|-----------------------------------|\n";
        }
        return output;
    }
/// <summary>
/// Finds heavyiest coins
/// </summary>
/// <param name="collectors"></param>
/// <returns></returns>
private string HeaviestCoins(List<Collector> collectors)
{
    double maxWeight = 0;
    string result = "";
    for (int i = 0; i < collectors.Count; i++)
    {
        for (int j = 0; j < collectors[i].Coins.Count; j++)
        {
            double weight = collectors[i].Coins[j].Weight;
            if (weight > maxWeight)
            {
                maxWeight = weight;
                result = "";
            }
            if (weight == maxWeight)
            {
                result += collectors[i].Coins[j].Country + " " +
               collectors[i].Coins[j].Denomination + " " +
               collectors[i].Coins[j].Weight + "\n";
            }
        }
    }
    return result.Trim();
}
/// <summary>
/// Sorts the list
/// </summary>
/// <param name="collectors">collectors list</param>
private void Sort(List<Collector> collectors)
{
    foreach (Collector collector in collectors)
    {
        collector.Coins.Sort((a, b) => b.Weight.CompareTo(a.Weight) != 0 ?
            b.Weight.CompareTo(a.Weight) :
            b.Denomination.CompareTo(a.Denomination));
    }
}
/// <summary>
/// Finds coins with a specific denomination
/// </summary>
/// <param name="denomination">denomination</param>
/// <param name="collectors">collectors list</param>
/// <returns>coins info</returns>
private string CoinsWithDenomination(double denomination, List<Collector>
            collectors)
{
    string result = "";
    for (int i = 0; i < collectors.Count; i++)
```

```
    {
        for (int j = 0; j < collectors[i].Coins.Count; j++)
        {
            if (collectors[i].Coins[j].Denomination == denomination)
            {
                result += collectors[i].Coins[j].Country + " " +
               collectors[i].Coins[j].Denomination + " " +
               collectors[i].Coins[j].Weight + "\n";
            }
        }
    }
    return result.Trim();
}
/// <summary>
/// Removes coins that are from a country thats starts with a noted letter
/// </summary>
/// <param name="collectors">collectors list</param>
/// <param name="letter">letter</param>
private void RemoveCoinsStartingWithLetter(List<Collector> collectors, char
           letter)
{
    for (int i = 0; i < collectors.Count; i++)
    {
        collectors[i].Coins.RemoveAll(coin => coin.Country[0] == letter);
    }
}
/// <summary>
/// Filters coins that has denomination not higher than max denomination
/// </summary>
/// <param name="collectors"></param>
/// <param name="maxDenomination"></param>
/// <returns></returns>
private List<Coin> FilterByDenomination(List<Collector> collectors, int
           maxDenomination)
{
    List<Coin> filteredCoins = new List<Coin>();
    foreach (Collector collector in collectors)
    {
        foreach (Coin coin in collector.Coins)
        {
            if (coin.Denomination <= maxDenomination)
            {
                filteredCoins.Add(coin);
            }
        }
    }
    return filteredCoins;
}
 /// <summary>
 /// Inserts new coins into the sorted List
 /// </summary>
 /// <param name="sortedCoins">sorted list</param>
 /// <param name="newCoins">new coins list</param>
 private void InsertCoinsIntoSortedList(List<Coin> sortedCoins, List<Coin>
 newCoins)
 {
     foreach (Coin newCoin in newCoins)
     {
         int index = sortedCoins.Count;
         for (int i = 0; i < sortedCoins.Count; i++)
         {
             Coin existingCoin = sortedCoins[i];
             if (newCoin.Weight > existingCoin.Weight || (newCoin.Weight ==
                existingCoin.Weight && newCoin.Denomination >
                existingCoin.Denomination))
             {
                 index = i;
                 break;
```

```
                    }
                }
                sortedCoins.Insert(index, newCoin);
            }
        }
    }
}


using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;
using Laboras_2;
using System.Collections.Generic;
using System.Globalization;

namespace Laboras2testai
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void CoinConstructors()
        {
            string country = "Lithuania";
            double weight = 5.0;
            double denomination = 2.0;
            Coin coin = new Coin(country, weight, denomination);
            Assert.AreEqual(country, coin.Country);
            Assert.AreEqual(weight, coin.Weight);
            Assert.AreEqual(denomination, coin.Denomination);
        }
        [TestMethod]
        public void CoinToString()
        {
            Coin coin = new Coin("USA", 10.5, 1.0);
            string output = string.Format("| {0,-11} | {1,-12} | {2,-6} |", "USA", 1.0,
                   10.5);
            string realOutput = coin.ToString();
            Assert.AreEqual(output, realOutput);
        }
        public void CoinToStringCSV()
        {
            Coin coin = new Coin("USA", 10.5, 1.0);
            string output = string.Format("{0,-11}, {1,-12}, {2,-6}", "USA", 1.0, 10.5);
            string realOutput = coin.ToStringCSV();
            Assert.AreEqual(output, realOutput);
        }
    }
    [TestClass]
    public class UnitTest2
    {
        [TestMethod]
        public void CollectorConstructor()
        {
            string nameSur = "John Doe";
            List<Coin> coins = new List<Coin>
            {
                new Coin("Lithuania", 5.0, 2.0),
                new Coin("USA", 10.0, 1.0)
            };
            Collector collector = new Collector(nameSur, coins);
            Assert.AreEqual(nameSur, collector.NameSur);
            Assert.AreEqual(coins.Count, collector.Coins.Count);
            CollectionAssert.AreEqual(coins, collector.Coins);
        }
        [TestMethod]
        public void CollectorCoinsList()
        {
```

```
            Collector collector = new Collector("Jane Doe", new List<Coin>());
            Coin coin = new Coin("Germany", 7.5, 5.0);
            collector.Coins.Add(coin);
            Assert.AreEqual(1, collector.Coins.Count);
            Assert.AreEqual(coin, collector.Coins[0]);
        }
    }
}
```

2.6. Pradiniai duomenys ir rezultatai

Duomenys

```
Antanas Antanaitis
?veicarija 5 8
Lietuva 1 8
```

```
Tomas Tomaitis
Bulgarija 4 1
Belgija 5 6
```

```
Aidas Taidas
Vengrija 7 8
Portugalija 5 9
```

Rezultatai

```
            Antanas Antanaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| ?veicarija  | 5            | 8      |
| Lietuva     | 1            | 8      |
|-------------------------------------|

             Tomas Tomaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Bulgarija   | 4            | 1      |
| Belgija     | 5            | 6      |
|-------------------------------------|

SORTED DATA:

          Antanas Antanaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| ?veicarija  | 5            | 8      |
| Lietuva     | 1            | 8      |
|-------------------------------------|

             Tomas Tomaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Belgija     | 5            | 6      |
| Bulgarija   | 4            | 1      |
|-------------------------------------|
```

```
Heaviet coins:
?veicarija 5 8
Lietuva 1 8

Belgija 5 6


Total monetary value:
6
9

Coins with denomination 5:
?veicarija 5 8
Belgija 5 6


Data with removed coins:

          Antanas Antanaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| ?veicarija  | 5            | 8      |
|-------------------------------------|

            Tomas Tomaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Belgija     | 5            | 6      |
| Bulgarija   | 4            | 1      |
|-------------------------------------|


Data with added coins:

Antanas Antanaitis su papildytom monetomis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Portugalija | 5            | 9      |
| ?veicarija  | 5            | 8      |
|-------------------------------------|
```

Duomenys

```
Antanas Antanaitis
?veicarija 0 0
Lietuva 0 0
```

```
Tomas Tomaitis
Bulgarija 4 1
Belgija 5 6
```

```
Aidas Taidas
```

```
Vengrija 7 8
Portugalija 5 9
```

Rezultatai

```
            Antanas Antanaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| ?veicarija  | 0            | 0      |
| Lietuva     | 0            | 0      |
|-------------------------------------|

            Tomas Tomaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Bulgarija   | 4            | 1      |
| Belgija     | 5            | 6      |
|-------------------------------------|

SORTED DATA:

          Antanas Antanaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| ?veicarija  | 0            | 0      |
| Lietuva     | 0            | 0      |
|-------------------------------------|

            Tomas Tomaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Belgija     | 5            | 6      |
| Bulgarija   | 4            | 1      |
|-------------------------------------|

Heaviet coins:
?veicarija 0 0
Lietuva 0 0

Belgija 5 6


Total monetary value:
0
9

Coins with denomination 4:

Bulgarija 4 1


Data with removed coins:

          Antanas Antanaitis
```

```
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| ?veicarija  | 0            | 0      |
| Lietuva     | 0            | 0      |
|-------------------------------------|

            Tomas Tomaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
|-------------------------------------|


Data with added coins:

Antanas Antanaitis su papildytom monetomis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Portugalija | 5            | 9      |
| ?veicarija  | 0            | 0      |
| Lietuva     | 0            | 0      |
|-------------------------------------|
```

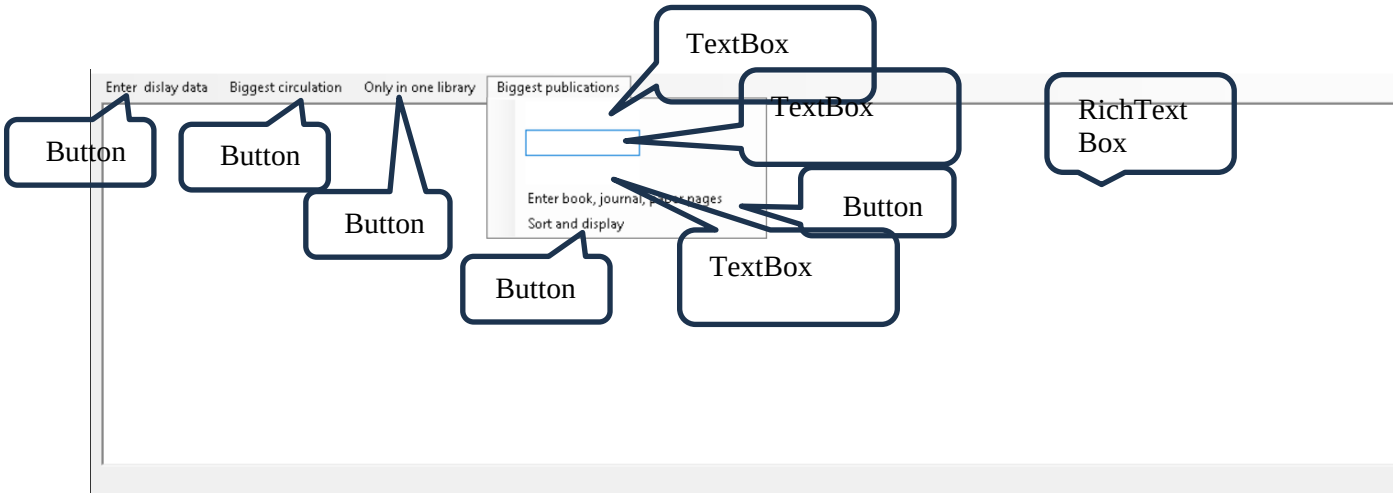
2.7. Dėstytojo pastabos

3. Paveldėjimas (L3)

3.1. Darbo užduotis

Turite dviejų KTU fakultetų bibliotekų duomenis. Pirmoje eilutėje – bibliotekos pavadinimas, antroje – adresas. Toliau informacija apie leidinius (knygas, žurnalus ir laikraščius): [Požymis,] [ISBN,] pavadinimas[, autorius(-iai)], leidykla, išleidimo metai[, išleidimo data][, numeris], puslapių skaičius, tiražas. Sukurkite klasę „Publication“ (savybės – pavadinimas, leidykla, išleidimo metai, puslapių skaičius, tiražas), kurią paveldės klasės „Book” (savybės – ISBN, autorius), “Journal” (savybės – ISBN, numeris) ir „Paper” (savybės – data, numeris). Problemos: • Raskite didžiausiu tiražu išleistą knygą,-as, žurnalą,-us ir laikraštį,-ius kiekvienoje bibliotekoje. Atspausdinkite jų pavadinimus, tiražą. • Sudarykite visų leidinių, kuriuos galima rasti tik vienoje bibliotekoje, bendrą sąrašą. Pateikite pilną informaciją apie juos. • Sudarykite nurodytos leidyklos storiausių leidinių bendrą sąrašą. Knyga yra stora jeigu jos puslapių skaičius yra > už P dydį, žurnalas – jeigu puslapių skaičius yra > už K dydį, laikraštis- puslapių skaičius yra > už N dydį. Pateikite pilną informaciją apie juos. Dydžiai P,K ir N nurodomi klaviatūra. • Išrikiuokite sudarytą sąrašą pagal puslapių skaičių ir tiražą.

3.2. Grafinės vartotojo sąsajos schema ir paveikslas



3.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
Button	Name	enterDisplayDataToolStripMenuItem
Button	Name	biggestCirculationToolStripMenuItem
Button	Name	onlyInOneLibraryToolStripMenuItem
Button	Name	sortToolStripMenuItem
Button	Name	enterBookJurnalPaperPagesToolStripMenuItem
TextBox	Name	writeBookPages
TextBox	Name	writeJournalPages
TextBox	Name	writePaperPages
RichTextBox	Name	display
	Font	Courier New, 9pt

3.4. Programos vartotojo vadovas

enterDisplayDataToolStripMenuItem – įveda ir parodo duomenis
biggestCirculationToolStripMenuItem – parodo didžiausio tiražo leidinius
onlyInOneLibraryToolStripMenuItem – parodo leidinius, kurie yra tik vienoje bibliotekoje
sortToolStripMenuItem – surikiuoja ir parodo atrinktus leidinius
enterBookJurnalPaperPagesToolStripMenuItem – atrenka leidinius
writeBookPages – storos knygos puslapiai
writeJournalPages – storo žurnalo puslapiai
writePaperPages – storo laikraščio puslapiai
display – rodo veiksmų rezultatus

3.5. Programos tekstas

```
using System;
using System.Collections.Generic;
namespace Laboras_3
{
    public abstract class Publication : IComparable<Publication>, IEquatable<Publication>
    {
        /// <summary>
        /// title
        /// </summary>
        public string Title { get; set; }
        /// <summary>
        /// publisher
        /// </summary>
        public string Publisher { get; set; }
        /// <summary>
        /// year
        /// </summary>
        public int Year { get; set; }
        /// <summary>
        /// pages
        /// </summary>
        public int Pages { get; set; }
        /// <summary>
        /// circulation
        /// </summary>
        public int Circulation { get; set; }
        /// <summary>
        /// constructor
        /// </summary>
        /// <param name="title">title</param>
        /// <param name="publisher">publisher</param>
        /// <param name="year">year</param>
        /// <param name="pages">pages</param>
        /// <param name="circulation">circulation</param>
        public Publication(string title, string publisher, int year, int pages, int
          circulation)
        {
            Title = title;
            Publisher = publisher;
            Year = year;
            Pages = pages;
            Circulation = circulation;
        }
        /// <summary>
        /// ToString
        /// </summary>
        /// <returns>Formated lines</returns>
        public abstract override string ToString();
        /// <summary>
```

```
        /// CompareTo
        /// </summary>
        /// <param name="publication">publication</param>
        /// <returns>true/false</returns>
        public int CompareTo(Publication publication)
        {
            int value = Pages.CompareTo(publication.Pages);
            if (value == 0)
                return Circulation.CompareTo(publication.Circulation);
            return value;
        }
        /// <summary>
        /// Equals
        /// </summary>
        /// <param name="publication">publication</param>
        /// <returns>true/false</returns>
        public bool Equals(Publication publication)
        {
            return Title == publication.Title && Year == publication.Year;
        }
        /// <summary>
        /// GetHashCode
        /// </summary>
        /// <returns>hashCode</returns>
        public override int GetHashCode()
        {
            int hashCode = 1080638607;
            hashCode = hashCode * -1521134295 +
            EqualityComparer<string>.Default.GetHashCode(Title);
            hashCode = hashCode * -1521134295 +
            EqualityComparer<string>.Default.GetHashCode(Publisher);
            hashCode = hashCode * -1521134295 + Year.GetHashCode();
            hashCode = hashCode * -1521134295 + Pages.GetHashCode();
            hashCode = hashCode * -1521134295 + Circulation.GetHashCode();
            return hashCode;
        }
        /// <summary>
        /// Equals
        /// </summary>
        /// <param name="obj">object</param>
        /// <returns>true/false</returns>
        public override bool Equals(object obj)
        {
            if (obj is Publication other)
                return Equals(other);
            return false;
        }
    }
}


using System;

namespace Laboras_3
{
    public class Book : Publication
    {
        /// <summary>
        /// isbn
        /// </summary>
        public int ISBN { get; set; }
        /// <summary>
        /// author
        /// </summary>
        public string Author { get; set; }
        /// <summary>
        /// constructor
        /// </summary>
```

```
        /// <param name="title">title</param>
        /// <param name="publisher">publisher</param>
        /// <param name="year">year</param>
        /// <param name="pages">pages</param>
        /// <param name="circulation">circulation</param>
        /// <param name="isbn">isbn</param>
        /// <param name="author">author</param>
        public Book(string title, string publisher, int year, int pages, int circulation,
               int isbn, string author)
            : base(title, publisher, year, pages, circulation)
        {
            ISBN = isbn;
            Author = author;
        }
        /// <summary>
        /// ToString
        /// </summary>
        /// <returns>Formated line</returns>
        public override string ToString()
        {
            return $"| b     | {Title,-11} | {Publisher,-9} | {Year,-16} | {Pages,-9} |
                   {Circulation,-7} | {ISBN,-4} | {Author,-15} |                 |
                   |";
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Laboras_3
{
    public class Journal : Publication
    {
        /// <summary>
        /// isbn
        /// </summary>
        public int ISBN { get; set; }
        /// <summary>
        /// number
        /// </summary>
        public int Number { get; set; }
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="title">title</param>
        /// <param name="publisher">publisher</param>
        /// <param name="year">year</param>
        /// <param name="pages">pages</param>
        /// <param name="circulation">circulation</param>
        /// <param name="isbn">isbn</param>
        /// <param name="number">number</param>
        public Journal(string title, string publisher, int year, int pages, int
             circulation, int isbn, int number)
            : base(title, publisher, year, pages, circulation)
        {
            ISBN = isbn;
            Number = number;
        }
        /// <summary>
        /// ToString
        /// </summary>
        /// <returns>Formated line</returns>
        public override string ToString()
```

```
        {
            return $"| j     | {Title,-11} | {Publisher,-9} | {Year,-16} | {Pages,-9} |
             {Circulation,-7} | {ISBN,-4} |                  | {"",-14} | {Number,-7} |";
        }
    }
}


using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Laboras_3
{
    public class Paper : Publication
    {
        /// <summary>
        /// Date
        /// </summary>
        public string Date { get; set; }
        /// <summary>
        /// Number
        /// </summary>
        public int Number { get; set; }
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="title">title</param>
        /// <param name="publisher">publisher</param>
        /// <param name="year">year</param>
        /// <param name="pages">pages</param>
        /// <param name="circulation">circulation</param>
        /// <param name="date">date</param>
        /// <param name="number">number</param>
        public Paper(string title, string publisher, int year, int pages, int
             circulation, string date, int number)
            : base(title, publisher, year, pages, circulation)
        {
            Date = date;
            Number = number;
        }
        /// <summary>
        /// ToString
        /// </summary>
        /// <returns>Formated line</returns>
        public override string ToString()
        {
            return $"| p     | {Title,-11} | {Publisher,-9} | {Year,-16} | {Pages,-9} |
             {Circulation,-7} |      |                  | {Date,-14} | {Number,-7} |";
        }
    }
}


using System.Collections.Generic;

namespace Laboras_3
{
    public class Library
    {
        /// <summary>
        /// Name
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Address
```

```
        /// </summary>
        public string Address { get; set; }
        /// <summary>
        /// Publications
        /// </summary>
        public List<Publication> Publications { get; set; }
        const string TABLE = "|-------------------------------------------------------
             ----------------------------------------------------------------|\r\n" +
                             "| Tipas | Pavadinimas | Leidykla  | Išleidimo metai  |
             Puslapiai | Tiražas | ISBN | Autorius        | Išleidimo data | Numeris
             |\r\n" +
                             "|-------------------------------------------------------
             ----------------------------------------------------------------|\r\n";
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="name">name</param>
        /// <param name="address"><address/param>
        public Library(string name, string address)
        {
            Name = name;
            Address = address;
            Publications = new List<Publication>();
        }
        /// <summary>
        /// ToString
        /// </summary>
        /// <returns>Formated output</returns>
        public override string ToString()
        {
            string result = $"\n {Name}\n {Address}\n";
            result += TABLE;
            foreach (var publication in Publications)
                result += publication.ToString() + "\n";
            result += "|-------------------------------------------------------------
             ----------------------------------------------------------|\r\n";
            return result;
        }
        /// <summary>
        /// Sort
        /// </summary>
        public void Sort()
        {
            this.Publications.Sort((publication1, publication2) =>
            {
                int pageCompare = publication2.Pages.CompareTo(publication1.Pages);
                if (pageCompare != 0)
                    return pageCompare;
                return publication2.Circulation.CompareTo(publication1.Circulation);
            });
        }
    }
}



using System;
using System.Collections.Generic;
using System.IO;
using System.Windows.Forms;

namespace Laboras_3
{
    public partial class Form1 : Form
    {
        Library library1, library2, filtered;
        public Form1()
        {
```

```
    InitializeComponent();
}
/// <summary>
/// Enters and displays data
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void enterDisplayDataToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog dialog1 = new OpenFileDialog();
    if (dialog1.ShowDialog() == DialogResult.OK)
    {
        string filePath = dialog1.FileName;
        library1 = ReadData(filePath);
    }
    OpenFileDialog dialog2 = new OpenFileDialog();
    if (dialog2.ShowDialog() == DialogResult.OK)
    {
        string filePath = dialog2.FileName;
        library2 = ReadData(filePath);
    }
    display.Text = library1.ToString() + "\n" + library2.ToString() + "\n";
}
/// <summary>
/// Displays publications with biggest circulations
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void biggestCirculationToolStripMenuItem_Click(object sender, EventArgs
     e)
{
    display.Text += "\n\nPublications with biggest circulation:\n\n";
    display.Text += FormatBiggestCirculation(library1);
    display.Text += FormatBiggestCirculation(library2);
}
/// <summary>
/// Displays publications wich are only in one library
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void onlyInOneLibraryToolStripMenuItem_Click(object sender, EventArgs e)
{
    Library unique = new Library("Publications found only in one library",
     "Library1 and Library2");
    unique = GetPublicationsInOnlyOneLibrary(library1, library2);
    if (unique != null && unique.Publications.Count > 0)
    {
        display.Text += unique.ToString();
    }
    else
    {
        display.Text += "\nPublications found only in one library:\nNot found\n";
    }
}
/// <summary>
/// filters publications wich have more pages than writen
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void enterBookJournalPaperPagesToolStripMenuItem_Click(object sender,
     EventArgs e)
{
    int thickBookPages = int.Parse(writeBookPages.Text);
    int thickJournalPages = int.Parse(writeJournalPages.Text);
    int thickPaperPages = int.Parse(writePaperPages.Text);
    filtered = GetThickPublications(library1, library2, thickBookPages,
     thickJournalPages, thickPaperPages);
}
```

```
/// <summary>
/// Sorts and displays filtered publications
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void sortDisplayToolStripMenuItem_Click(object sender, EventArgs e)
{
    filtered.Sort();
    if (filtered != null && filtered.Publications.Count > 0)
    {
        display.Text += filtered.ToString();
    }
    else
    {
        display.Text += "\nFiltered and sorted publications:\nNot found\n";
    }
}
/// <summary>
/// Reads data
/// </summary>
/// <param name="path"></param>
/// <returns>library</returns>
private Library ReadData(string path)
{
    using (StreamReader df = new StreamReader(path))
    {
        string line = df.ReadLine();
        string libraryName = line;
        line = df.ReadLine();
        string libraryAddress = line;
        Library library = new Library(libraryName, libraryAddress);
        string title, publisher, author, date, type;
        int year, pages, circulation, isbn, number;
        while ((line = df.ReadLine()) != null)
        {
            string[] parts = line.Split(';');
            type = parts[0];
            title = parts[1];
            publisher = parts[2];
            year = int.Parse(parts[3]);
            pages = int.Parse(parts[4]);
            circulation = int.Parse(parts[5]);
            if (type == "b")
            {
                isbn = int.Parse(parts[6]);
                author = parts[7];
                library.Publications.Add(new Book(title, publisher, year, pages,
     circulation, isbn, author));
            }
            else if (type == "j")
            {
                isbn = int.Parse(parts[6]);
                number = int.Parse(parts[7]);
                library.Publications.Add(new Journal(title, publisher, year,
     pages, circulation, isbn, number));
            }
            else
            {
                date = parts[6];
                number = int.Parse(parts[7]);
                library.Publications.Add(new Paper(title, publisher, year, pages,
     circulation, date, number));
            }
        }
        return library;
    }
}
/// <summary>
```

```
/// Formats publications with biggest circulations
/// </summary>
/// <param name="library"></param>
/// <returns>biggest circulations</returns>
private string FormatBiggestCirculation(Library library)
{
    string result = $"{library.Name}:\n";
    result += "Books:\n";
    result += FormatBiggestCirculations(BiggestCirculationByType(library,
     typeof(Book)));
    result += "Journals:\n";
    result += FormatBiggestCirculations(BiggestCirculationByType(library,
     typeof(Journal)));
    result += "Papers:\n";
    result += FormatBiggestCirculations(BiggestCirculationByType(library,
     typeof(Paper)));
    return result + "\n";
}
/// <summary>
/// Finds publicatios with biggest circulations by type
/// </summary>
/// <param name="library"></param>
/// <param name="type"></param>
/// <returns>biggest circulations by type</returns>
private List<Publication> BiggestCirculationByType(Library library, Type type)
{
    int maxCirculation = int.MinValue;
    List<Publication> result = new List<Publication>();
    foreach (var pub in library.Publications)
    {
        if (pub.GetType() == type)
        {
            if (pub.Circulation > maxCirculation)
            {
                maxCirculation = pub.Circulation;
                result.Clear();
                result.Add(pub);
            }
            else if (pub.Circulation == maxCirculation)
            {
                result.Add(pub);
            }
        }
    }
    return result;
}
/// <summary>
/// Formats publications with biggest circulations
/// </summary>
/// <param name="publications"></param>
/// <returns>Formated lines</returns>
private string FormatBiggestCirculations(List<Publication> publications)
{
    string result = "";
    foreach (var publication in publications)
    {
        result += $"{publication.Title}, {publication.Circulation}\n";
    }
    return result;
}
/// <summary>
/// Finds publications that are only in one library
/// </summary>
/// <param name="library1"></param>
/// <param name="library2"></param>
/// <returns>unique publications</returns>
private Library GetPublicationsInOnlyOneLibrary(Library library1, Library
     library2)
```

```
        {
            Library unique = new Library("Publications found only in one library",
             "Library1 and Library2");
            int count1 = library1.Publications.Count;
            int count2 = library2.Publications.Count;
            int max = Math.Max(count1, count2);
            for (int i = 0; i < max; i++)
            {
                if (i < count1)
                {
                    if (!library2.Publications.Contains(library1.Publications[i]))
                    {
                        unique.Publications.Add(library1.Publications[i]);
                    }
                }
                if (i < count2)
                {
                    if (!library1.Publications.Contains(library2.Publications[i]))
                    {
                        unique.Publications.Add(library2.Publications[i]);
                    }
                }
            }
            return unique;
        }
        /// <summary>
        /// Filters publications that are considered thick
        /// </summary>
        /// <param name="library1">library1</param>
        /// <param name="library2">library2</param>
        /// <param name="bookPages">thick book pages</param>
        /// <param name="journalPages">thick journal pages</param>
        /// <param name="paperPages">thick paper pages</param>
        /// <returns>filtered publications</returns>
        private Library GetThickPublications(Library library1, Library library2, int
             bookPages, int journalPages, int paperPages)
        {
            Library filtered = new Library("Filtered and sorted publications", "Library1
             and Library2");
            List<Publication> publications = new List<Publication>();
            publications.AddRange(library1.Publications);
            publications.AddRange(library2.Publications);
            for (int i = 0; i < publications.Count; i++)
            {
                Publication publication = publications[i];
                if (publication is Book book && book.Pages > bookPages)
                {
                    filtered.Publications.Add(book);
                }
                else if (publication is Journal journal && journal.Pages > journalPages)
                {
                    filtered.Publications.Add(journal);
                }
                else if (publication is Paper paper && paper.Pages > paperPages)
                {
                    filtered.Publications.Add(paper);
                }
            }
            return filtered;
        }
    }
}


using Microsoft.VisualStudio.TestTools.UnitTesting;
using Laboras_3;
using System;
```

```
namespace Laboras3_Testai
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void Book_ToString_ReturnsCorrectFormat()
        {
            Book book = new Book("TestBook", "TestPub", 2020, 300, 1000, 123456, "Author
             Name");
            string expectedStart = "| b     | TestBook    | TestPub   | 2020
             | 300       | 1000    | 123456";
            StringAssert.StartsWith(book.ToString(), expectedStart);
        }

        [TestMethod]
        public void Journal_ToString_ReturnsCorrectFormat()
        {
            Journal journal = new Journal("Science", "Nature", 2022, 100, 5000, 112233,
             4);
            string expectedStart = "| j     | Science     | Nature    | 2022
             | 100       | 5000    | 112233";
            StringAssert.StartsWith(journal.ToString(), expectedStart);
        }
        [TestMethod]
        public void Paper_ToString_ReturnsCorrectFormat()
        {
            Paper paper = new Paper("Daily News", "NewsPub", 2021, 50, 20000, "2021-08-
             01", 15);
            string expectedStart = "| p     | Daily News  | NewsPub   | 2021
             | 50        | 20000   |";
            StringAssert.StartsWith(paper.ToString(), expectedStart);
        }
        [TestMethod]
        public void Publication_Equals_ReturnsTrueForEqualTitleAndYear()
        {
            Book book1 = new Book("Title", "Pub1", 2020, 100, 1000, 123, "Author");
            Book book2 = new Book("Title", "Pub2", 2020, 150, 2000, 456, "Another");
            Assert.IsTrue(book1.Equals(book2));
        }
        [TestMethod]
        public void Publication_CompareTo_PrioritizesPagesThenCirculation()
        {
            Book b1 = new Book("Book1", "Pub", 2020, 200, 1000, 1, "Author1");
            Book b2 = new Book("Book2", "Pub", 2020, 200, 500, 2, "Author2");
            Assert.IsTrue(b1.CompareTo(b2) > 0);
        }
        [TestMethod]
        public void Publication_GetHashCode_ConsistentWithEquals()
        {
            Book book1 = new Book("HashTitle", "HPub", 2022, 120, 3000, 555, "Hash");
            Book book2 = new Book("HashTitle", "HPub", 2022, 120, 3000, 555, "Hash");
            Assert.AreEqual(book1.GetHashCode(), book2.GetHashCode());
        }
    }
    [TestClass]
    public class LibraryTests
    {
        [TestMethod]
        public void Library_ToString_IncludesAllPublications()
        {
            Library library = new Library("Main Library", "Vilnius");
            library.Publications.Add(new Book("B1", "P1", 2020, 300, 1000, 111,
             "Author1"));
            library.Publications.Add(new Paper("P2", "P2Pub", 2021, 50, 2000, "2021-01-
             01", 1));
            string output = library.ToString();
            StringAssert.Contains(output, "B1");
```

```
            StringAssert.Contains(output, "P2");
        }
        [TestMethod]
        public void Library_Sort_SortsByPagesDescendingThenCirculationDescending()
        {
            Library library = new Library("SortLib", "Kaunas");
            var b1 = new Book("B1", "P1", 2020, 100, 500, 1, "A");
            var b2 = new Book("B2", "P2", 2020, 150, 600, 2, "B");
            var b3 = new Book("B3", "P3", 2020, 150, 700, 3, "C");
            library.Publications.AddRange(new[] { b1, b2, b3 });
            library.Sort();
            Assert.AreEqual("B3", library.Publications[0].Title);
            Assert.AreEqual("B2", library.Publications[1].Title);
            Assert.AreEqual("B1", library.Publications[2].Title);
        }
    }
}
```

3.6. Pradiniai duomenys ir rezultatai

Duomenys 1:

Biblioteka1 Laisvės al. 53, Kaunas, Lietuva b;pav1;leidykla1;2000;200;3;123;autorius6261616 j;pav2;leidykla2;2001;25;5;500;4 p;pav7;leidykla3;2002;10;5;2002-05-03;10

Duomenys 2:

Biblioteka2 Studentų g. 50, Kaunas, Lietuva b;pav1;leidykla1;2000;200;3;123;autorius1 j;pav3;leidykla4;2005;27;4;502;7 p;pav6;leidykla5;2004;11;8;2004-05-03;17

Rezultatai:

```
Biblioteka1
 Laisvės al. 53, Kaunas, Lietuva
|-----------------------------------------------------------------------------------------------------------------------|
| Tipas | Pavadinimas | Leidykla  | Išleidimo metai  | Puslapiai | Tiražas | ISBN | Autorius        | Išleidimo data | Numeris |
|-----------------------------------------------------------------------------------------------------------------------|
| b     | pav1        | leidykla1 | 2000             | 200       | 3       | 123  | autorius6261616 |                |         |
| j     | pav2        | leidykla2 | 2001             | 25        | 5       | 500  |                 |                | 4       |
| p     | pav7        | leidykla3 | 2002             | 10        | 5       |      |                 | 2002-05-03     | 10      |
|-----------------------------------------------------------------------------------------------------------------------|


 Biblioteka2
 Studentų g. 50, Kaunas, Lietuva
|-----------------------------------------------------------------------------------------------------------------------|
| Tipas | Pavadinimas | Leidykla  | Išleidimo metai  | Puslapiai | Tiražas | ISBN | Autorius        | Išleidimo data | Numeris |
|-----------------------------------------------------------------------------------------------------------------------|
| b     | pav1        | leidykla1 | 2000             | 200       | 3       | 123  | autorius1       |                |         |
| j     | pav3        | leidykla4 | 2005             | 27        | 4       | 502  |                 |                | 7       |
| p     | pav6        | leidykla5 | 2004             | 11        | 8       |      |                 | 2004-05-03     | 17      |
|-----------------------------------------------------------------------------------------------------------------------|



Publications with biggest circulation:

Biblioteka1:
Books:
pav1, 3
Journals:
pav2, 5
Papers:
pav7, 5

Biblioteka2:
Books:
pav1, 3
Journals:
pav3, 4
Papers:
pav6, 8
```

```
 Publications found only in one library
 Library1 and Library2
|-----------------------------------------------------------------------------------------------------------------------|
| Tipas | Pavadinimas | Leidykla  | Išleidimo metai  | Puslapiai | Tiražas | ISBN | Autorius        | Išleidimo data | Numeris |
|-----------------------------------------------------------------------------------------------------------------------|
| j     | pav2        | leidykla2 | 2001             | 25        | 5       | 500  |                 |                | 4       |
| j     | pav3        | leidykla4 | 2005             | 27        | 4       | 502  |                 |                | 7       |
| p     | pav7        | leidykla3 | 2002             | 10        | 5       |      |                 | 2002-05-03     | 10      |
| p     | pav6        | leidykla5 | 2004             | 11        | 8       |      |                 | 2004-05-03     | 17      |
|-----------------------------------------------------------------------------------------------------------------------|

 Filtered and sorted publications
 Library1 and Library2
|-----------------------------------------------------------------------------------------------------------------------|
| Tipas | Pavadinimas | Leidykla  | Išleidimo metai  | Puslapiai | Tiražas | ISBN | Autorius        | Išleidimo data | Numeris |
|-----------------------------------------------------------------------------------------------------------------------|
| j     | pav3        | leidykla4 | 2005             | 27        | 4       | 502  |                 |                | 7       |
| j     | pav2        | leidykla2 | 2001             | 25        | 5       | 500  |                 |                | 4       |
| p     | pav6        | leidykla5 | 2004             | 11        | 8       |      |                 | 2004-05-03     | 17      |
|-----------------------------------------------------------------------------------------------------------------------|
```

Duomenys 1:

```
Biblioteka1
Laisvės al. 53, Kaunas, Lietuva
b;pav1;leidykla1;2000;200;3;123;autorius6261616
j;pav2;leidykla2;2001;25;5;500;4
p;pav7;leidykla3;2002;10;5;2002-05-03;10
```

Duomenys 2:

```
Biblioteka2
Studentų g. 50, Kaunas, Lietuva
b;pav1;leidykla1;2000;200;3;123;autorius1
b;pav8;leidykla6;2050;253;3;123;autorius6261616
j;pav3;leidykla4;2005;27;4;502;7
p;pav6;leidykla5;2004;11;8;2004-05-03;17
```

Rezultatai

Biblioteka1
Laisvės al. 53, Kaunas, Lietuva

Tipas	Pavadinimas	Leidykla	Išleidimo metai	Puslapiai	Tiražas	ISBN	Autorius	Išleidimo data	Numeris
b	pav1	leidykla1	2000	200	3	123	autorius6261616		
j	pav2	leidykla2	2001	25	5	500			4
p	pav7	leidykla3	2002	10	5			2002-05-03	10

Biblioteka2
Studentų g. 50, Kaunas, Lietuva

Tipas	Pavadinimas	Leidykla	Išleidimo metai	Puslapiai	Tiražas	ISBN	Autorius	Išleidimo data	Numeris
b	pav1	leidykla1	2000	200	3	123	autorius1		
b	pav8	leidykla6	2050	253	3	123	autorius6261616		
j	pav3	leidykla4	2005	27	4	502			7
p	pav6	leidykla5	2004	11	8			2004-05-03	17

Publications with biggest circulation:

Biblioteka1:
Books:
pav1, 3
Journals:
pav2, 5
Papers:
pav7, 5

Biblioteka2:
Books:
pav1, 3
pav8, 3
Journals:
pav3, 4
Papers:
pav6, 8

```
 Publications found only in one library
 Library1 and Library2
|-----------------------------------------------------------------------------------------------------------------------|
| Tipas | Pavadinimas | Leidykla  | Išleidimo metai  | Puslapiai | Tiražas | ISBN | Autorius        | Išleidimo data | Numeris |
|-----------------------------------------------------------------------------------------------------------------------|
| j     | pav2        | leidykla2 | 2001             | 25        | 5       | 500  |                 |                | 4       |
| b     | pav8        | leidykla6 | 2050             | 253       | 3       | 123  | autorius6261616 |                |         |
| p     | pav7        | leidykla3 | 2002             | 10        | 5       |      |                 | 2002-05-03     | 10      |
| j     | pav3        | leidykla4 | 2005             | 27        | 4       | 502  |                 |                | 7       |
| p     | pav6        | leidykla5 | 2004             | 11        | 8       |      |                 | 2004-05-03     | 17      |
|-----------------------------------------------------------------------------------------------------------------------|

 Filtered and sorted publications
 Library1 and Library2
|-----------------------------------------------------------------------------------------------------------------------|
| Tipas | Pavadinimas | Leidykla  | Išleidimo metai  | Puslapiai | Tiražas | ISBN | Autorius        | Išleidimo data | Numeris |
|-----------------------------------------------------------------------------------------------------------------------|
| b     | pav8        | leidykla6 | 2050             | 253       | 3       | 123  | autorius6261616 |                |         |
| j     | pav3        | leidykla4 | 2005             | 27        | 4       | 502  |                 |                | 7       |
| j     | pav2        | leidykla2 | 2001             | 25        | 5       | 500  |                 |                | 4       |
|-----------------------------------------------------------------------------------------------------------------------|
```

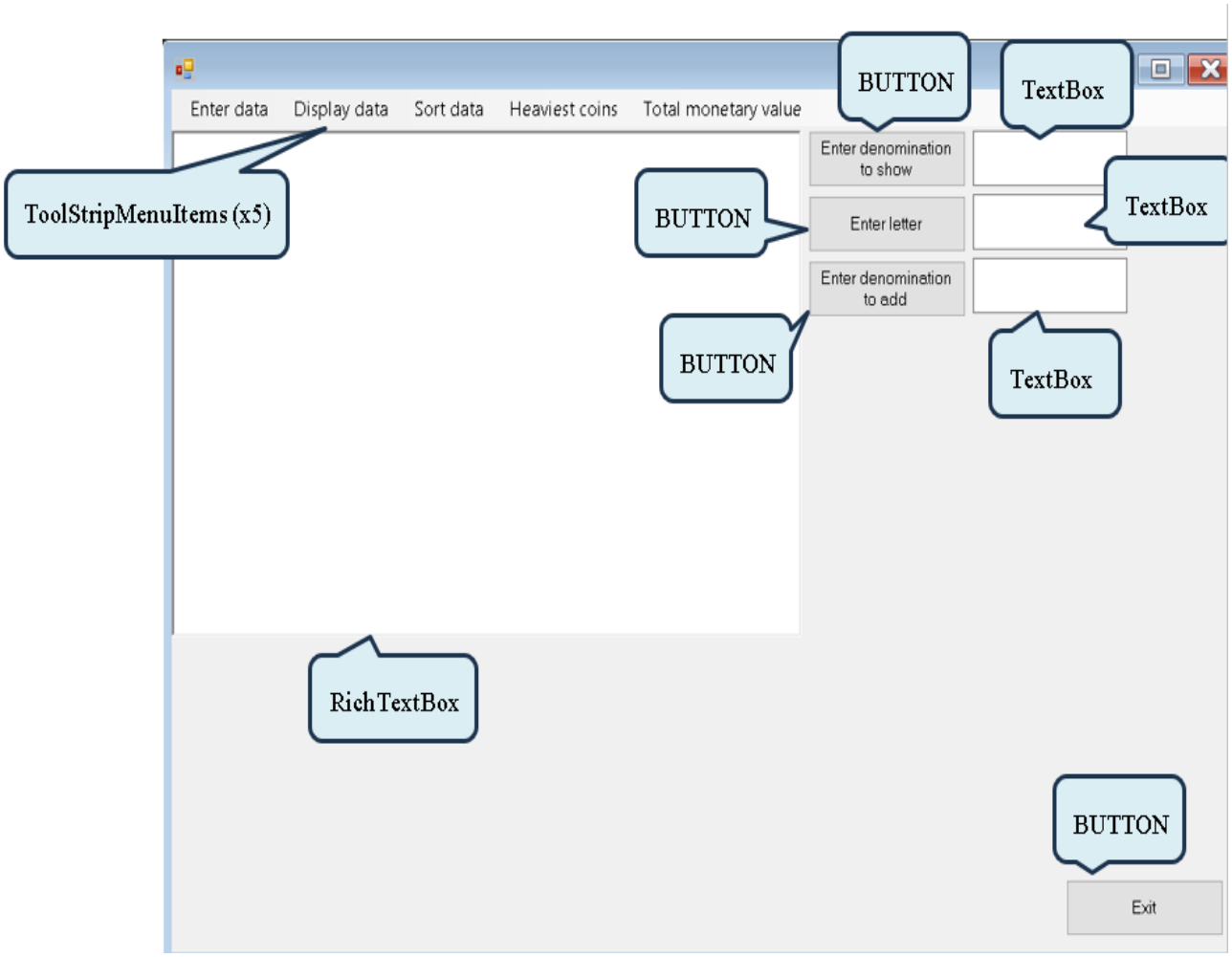
3.7. Dėstytojo pastabos

5. Susietasis sąrašas (L4)

5.1. Darbo užduotis

U1-4. Monetos. Monetų kolekcininko turimų monetų duomenys surašyti faile: monetos kilmės šalis, nominalas ir svoris. Pirmoje eilutėje yra kolekcininko vardas ir pavardė. Turime dviejų kolekcininkų duomenis. L1+L2+L4. • Raskite kiekvieno kolekcininko sunkiausią monetą ir kokia visų kolekcininko monetų bendra piniginė vertė? • Surašykite į atskirą rinkinį visus abiejų kolekcijų monetų, kurių nominalas yra N, duomenis. Nominalas nurodomas klaviatūra. • Surikiuokite rezultatų sąrašą monetų svorio mažėjimo tvarka ir nominalą (L2). • Pašalinkite iš rezultatų sąrašo monetas, kurių šalies pavadinimas prasideda nurodyta klaviatūra raide. L2 papildymas. • Papildykite surikiuotą rezultatų sąrašą naujo kolekcininko monetomis, kurių nominalas ne didesnis už nurodytą klaviatūra. Duomenys yra atskirame faile. Pirmoje eilutėje yra kolekcininko vardas ir pavardė.

5.2. Grafinės vartotojo sąsajos schema ir paveikslas



5.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
Button	Name	Exit
Button	Name	Enter letter
Button	Name	Enter denomination to show
Button	Name	Enter denomination to add

RichTextBox	Font	Courier New, 12pt
	Name	display
TextBox	Name	textDenominationToAdd
TextBox	Name	textEnterLetter
TextBox	Name	textDenominationToShow
ToolStripMenuItem	Name	enterDataToolStripMenuItem
ToolStripMenuItem	Name	displayDataToolStripMenuItem
ToolStripMenuItem	Name	sortDataToolStripMenuItem
ToolStripMenuItem	Name	heaviestCoinsToolStripMenuItem
ToolStripMenuItem	Name	totalMonetaryValueToolStripMenuItem

5.4. Programos vartotojo vadovas

Enter data – įveda
Display data – pavaizduoja duomenis
Sort data – surikiuoja duomenis
Heaviest coins – randa sunkiausias monetas
Total monetary value – suskaičiuoja bendrą monetų vertę kiekvieno kolekcionieriaus
Enter denomination to show – randa visas monetas, su nurodytu nominalu
Enter letter – pašalina monetas, kurios yra iš šalių, kurios prasideda nurodyta raide
Enter denomination to add – prideda į pirmojo kolekcionieriaus surikiuotą lentelę naujo kolekcionieriaus monetas, kurios turi ne didesnį už nurodytą nominąlą
Exit – uždaro programą

5.5. Programos tekstas

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Laboras_4
{
    /// <summary>
    /// Represents a coin with country, denomination, and weight properties
    /// Provides comparison operators based on weight and denomination
    /// </summary>
    internal class Coin
    {
        /// <summary>
        /// Gets or sets the country where the coin was issued
        /// </summary>
        public string Country { get; set; }
        /// <summary>
        /// Gets or sets the denomination (value) of the coin
        /// </summary>
        public int Denomination { get; set; }
        /// <summary>
        /// Gets or sets the weight of the coin
        /// </summary>
        public double Weight { get; set; }
        /// <summary>
        /// Initializes a new instance of the Coin class with specified country,
        denomination, and weight
```

```
        /// </summary>
        /// <param name="country">The country of origin</param>
        /// <param name="denomination">The value of the coin</param>
        /// <param name="weight">The weight of the coin</param>
        public Coin(string country, int denomination, double weight)
        {
            Country = country;
            Denomination = denomination;
            Weight = weight;
        }
        /// <summary>
        /// Determines whether one coin is considered less than another
        /// First compares weight; if equal, compares denomination
        /// </summary>
        /// <param name="coin1">The first coin</param>
        /// <param name="coin2">The second coin</param>
        /// <returns>True if coin1 is less than coin2, otherwise false</returns>
        public static bool operator <(Coin coin1, Coin coin2)
        {
            if (coin1.Weight == coin2.Weight)
            {
                return coin1.Denomination < coin2.Denomination;
            }
            return coin1.Weight < coin2.Weight;
        }
        /// <summary>
        /// Determines whether one coin is considered greater than another
        /// First compares weight; if equal, compares denomination
        /// </summary>
        /// <param name="coin1">The first coin</param>
        /// <param name="coin2">The second coin</param>
        /// <returns>True if coin1 is greater than coin2, otherwise false</returns>
        public static bool operator >(Coin coin1, Coin coin2)
        {
            if (coin1.Weight == coin2.Weight)
            {
                return coin1.Denomination > coin2.Denomination;
            }
            return coin1.Weight > coin2.Weight;
        }
        /// <summary>
        /// Returns a string representation of the coin in a formatted table row
        /// </summary>
        /// <returns>Formatted string representing the coin</returns>
        public override string ToString()
        {
            return string.Format("| {0,-11} | {1,-12} | {2,-6} |\n", Country,
              Denomination, Weight);
        }
    }
}


namespace Laboras_4
{
    /// <summary>
    /// Represents a single node in a linked list, containing a Coin and reference to the
        next node
    /// </summary>
    internal sealed class Node
    {
        /// <summary>
        /// Gets or sets the Coin data stored in this node
        /// </summary>
        public Coin Data { get; set; }
        /// <summary>
        /// Gets or sets the reference to the next node in the linked list
        /// </summary>
```

```
        public Node Next { get; set; }
        /// <summary>
        /// Initializes a new instance of the Node class with default values
        /// </summary>
        public Node() { }
        /// <summary>
        /// Initializes a new instance of the Node class with specified data and next
         node reference
        /// </summary>
        /// <param name="data">The Coin to store in this node</param>
        /// <param name="next">Reference to the next node in the list</param>
        public Node(Coin data, Node next)
        {
            Data = data;
            Next = next;
        }
    }
}


using System.Collections;

namespace Laboras_4
{
    /// <summary>
    /// Represents a collection of Coin objects stored in a linked list
    /// Implements IEnumerable for iteration
    /// </summary>
    internal sealed class Collector : IEnumerable
    {
        /// <summary>
        /// Gets or sets the name of the collector
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Reference to the first node in the linked list
        /// </summary>
        public Node Start;
        /// <summary>
        /// Reference used for iteration through the list
        /// </summary>
        public Node Current;
        /// <summary>
        /// Initializes a new instance of the Collector class with an empty list
        /// </summary>
        public Collector()
        {
            Start = null;
            Current = null;
        }
        /// <summary>
        /// Returns an enumerator that iterates through the linked list
        /// </summary>
        public IEnumerator GetEnumerator()
        {
            for (Node ss = Start; ss != null; ss = ss.Next)
            {
                yield return ss.Data;
            }
        }
        /// <summary>
        /// Returns the Coin data from the current node
        /// </summary>
        public Coin GetData()
        {
            return Current.Data;
        }
        /// <summary>
```

```
/// Resets the Current pointer to the start of the list
/// </summary>
public void First()
{
    Current = Start;
}
/// <summary>
/// Moves the Current pointer to the next node
/// </summary>
public void Next()
{
    Current = Current.Next;
}
/// <summary>
/// Checks if the Current node exists (is not null)
/// </summary>
public bool Exist()
{
    return Current != null;
}
/// <summary>
/// Adds a new Coin to the end of the linked list
/// </summary>
/// <param name="add">The Coin to add</param>
public void AddToTheEnd(Coin add)
{
    Node node = new Node(add, null);
    if (Start == null)
    {
        Start = node;
    }
    else
    {
        Node d = Start;
        while (d.Next != null)
        {
            d = d.Next;
        }
        d.Next = node;
    }
}
/// <summary>
/// Sorts the linked list in descending order using bubble sort logic
/// Sorting is based on Coin comparison operators
/// </summary>
public void Sort()
{
    if (Start == null || Start.Next == null)
    {
        return;
    }
    bool swap = true;
    while (swap)
    {
        swap = false;
        Node node = Start;
        while (node.Next != null)
        {
            if (node.Data < node.Next.Data)
            {
                Coin temp = node.Data;
                node.Data = node.Next.Data;
                node.Next.Data = temp;
                swap = true;
            }
            node = node.Next;
        }
    }
```

```
        }
        /// <summary>
        /// Combines two collectors by appending all coins from both into a new collector
        /// </summary>
        /// <param name="collector1">The first collector</param>
        /// <param name="collector2">The second collector</param>
        /// <returns>A new collector containing all coins from both collectors</returns>
        public static Collector operator +(Collector collector1, Collector collector2)
        {
            Collector joint = new Collector();
            Collector[] collectors = { collector1, collector2 };
            foreach (Collector collector in collectors)
            {
                collector.First();
                while (collector.Exist())
                {
                    joint.AddToTheEnd(collector.GetData());
                    collector.Next();
                }
            }
            return joint;
        }
        /// <summary>
        /// Inserts a Coin into the linked list while maintaining descending order
        /// Ordering is determined by Coin's comparison operators
        /// </summary>
        /// <param name="coin">The Coin to insert</param>
        public void InsertInSorted(Coin coin)
        {
            Node newNode = new Node(coin, null);
            if (Start == null || coin > Start.Data)
            {
                newNode.Next = Start;
                Start = newNode;
            }
            else
            {
                Current = Start;
                while (Current.Next != null && Current.Next.Data > coin)
                {
                    Current = Current.Next;
                }
                newNode.Next = Current.Next;
                Current.Next = newNode;
            }
        }
    }
}


using System;
using System.IO;
using System.Windows.Forms;

namespace Laboras_4
{
    public partial class Form1 : Form
    {
        const string TABLE = "|-----------------------------------|\n" +
                             "| Country     | Denomination | Weight |\n" +
                             "|-----------------------------------|\n";
        string REZ;
        Collector collector1, collector2, collector3, sorted, joint;
        public Form1()
        {
            InitializeComponent();
        }
        /// <summary>
```

```
/// Handles reading data from two selected files and initializes collectors
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void enterDataToolStripMenuItem_Click(object sender, EventArgs e)
{
    string dataFile1 = string.Empty, dataFile2 = string.Empty;
    OpenFileDialog openFileDialog1 = new OpenFileDialog
    {
        Filter = "All files (*.*)|*.*",
        Title = "Choose the first data file"
    };
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        string df1 = openFileDialog1.FileName;
        dataFile1 = File.ReadAllText(df1);
        collector1 = ReadData(df1);
    }
    OpenFileDialog openFileDialog2 = new OpenFileDialog
    {
        Filter = "All files (*.*)|*.*",
        Title = "Choose the second data file"
    };
    if (openFileDialog2.ShowDialog() == DialogResult.OK)
    {
        string df2 = openFileDialog2.FileName;
        dataFile2 = File.ReadAllText(df2);
        collector2 = ReadData(df2);
    }
    if (!string.IsNullOrEmpty(dataFile1) && !string.IsNullOrEmpty(dataFile2))
    {
        display.Text = dataFile1 + "\n\n" + dataFile2;
        displayDataToolStripMenuItem.Enabled = true;
        sortDataToolStripMenuItem.Enabled = true;
        heaviestCoinsToolStripMenuItem.Enabled = true;
        totalMonetaryValueToolStripMenuItem.Enabled = true;
        enterDenominationToShow.Enabled = true;
        enterLetter.Enabled = true;
        enterDenominationToAdd.Enabled = true;
        enterDataToolStripMenuItem.Enabled = false;
        joint = collector1 + collector2;
    }
}
/// <summary>
/// Saves and displays the contents of both collectors
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void displayDataToolStripMenuItem_Click(object sender, EventArgs e)
{
    SaveFileDialog saveFileDialog1 = new SaveFileDialog
    {
        Filter = "All files (*.*)|*.*",
        Title = "Choose results data file"
    };
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        REZ = saveFileDialog1.FileName;
        if (File.Exists(REZ)) File.Delete(REZ);
        string output = Print(REZ, collector1, collector1.Name) + Print(REZ,
     collector2, collector2.Name);
        display.Text = output;
    }
}
/// <summary>
/// Sorts the combined collection and displays it
/// </summary>
/// <param name="sender"></param>
```

```
/// <param name="e"></param>
private void sortDataToolStripMenuItem_Click(object sender, EventArgs e)
{
    sorted = joint;
    sorted.Sort();
    display.Text += Print(REZ, sorted, "Sorted coins");
}
/// <summary>
/// Displays the heaviest coins from both collectors
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void heaviestCoinsToolStripMenuItem_Click(object sender, EventArgs e)
{
    display.Text += "\nHeaviest coins:\n";
    Coin coin1 = HeaviestCoin(collector1);
    Collector collector1HeaviestCoins = HeaviestCoins(coin1, collector1);
    Coin coin2 = HeaviestCoin(collector2);
    Collector collector2HeaviestCoins = HeaviestCoins(coin2, collector2);
    display.Text += Print(REZ, collector1HeaviestCoins, "First collector's
     heaviest coins");
    display.Text += Print(REZ, collector2HeaviestCoins, "Second collector's
     heaviest coins");
}
/// <summary>
/// Calculates and displays the total monetary value for each collector
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void totalMonetaryValueToolStripMenuItem_Click(object sender, EventArgs
     e)
{
    display.Text += "\nTotal monetary value:\n";
    display.Text += $"Collector 1 - {TotalMonetaryValue(collector1)}\n";
    display.Text += $"Collector 2 - {TotalMonetaryValue(collector2)}\n";
    using (StreamWriter rez = new StreamWriter(File.Open(REZ, FileMode.Append)))
    {
        rez.WriteLine("\nTotal monetary value:\n");
        rez.WriteLine($"Collector 1 - {TotalMonetaryValue(collector1)}");
        rez.WriteLine($"Collector 2 - {TotalMonetaryValue(collector2)}");
    }
}
/// <summary>
/// Filters and displays coins with the specified denomination
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void enterDenominationToShow_Click(object sender, EventArgs e)
{
    int denomination = int.Parse(textDenominationToShow.Text);
    display.Text += DenominationToShow(joint, denomination, REZ);
}
/// <summary>
/// Removes and displays coins from countries starting with a given letter
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void enterLetter_Click(object sender, EventArgs e)
{
    string letter = textEnterLetter.Text;
    display.Text += Letter(joint, letter, REZ);
}
/// <summary>
/// Adds coins of a specific denomination from a third file to the sorted
     collection
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
```

```
private void enterDenominationToAdd_Click(object sender, EventArgs e)
{
    int denomination = int.Parse(textDenominationToAdd.Text);
    OpenFileDialog openFileDialog = new OpenFileDialog
    {
        Filter = "All files (*.*)|*.*",
        Title = "Choose the third data file"
    };
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        string df3 = openFileDialog.FileName;
        string dataFile3 = File.ReadAllText(df3);
        collector3 = ReadData(df3);
    }
    display.Text += Add(sorted, collector3, denomination, REZ);
}
/// <summary>
/// Closes the form
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void exit_Click(object sender, EventArgs e)
{
    Close();
}
/// <summary>
/// Reads coin data from a file and returns a Collector object
/// </summary>
/// <param name="DF">Path to the data file</param>
/// <returns>Collector with loaded coin data</returns>
static Collector ReadData(string DF)
{
    Collector collector = new Collector();
    using (StreamReader df = new StreamReader(DF))
    {
        string line = df.ReadLine();
        collector.Name = line;
        while ((line = df.ReadLine()) != null)
        {
            string[] parts = line.Split(' ');
            string country = parts[0];
            int denomination = int.Parse(parts[1]);
            double weight = double.Parse(parts[2]);
            Coin coin = new Coin(country, denomination, weight);
            collector.AddToTheEnd(coin);
        }
    }
    return collector;
}
/// <summary>
/// Formats and writes collector data to the result file
/// </summary>
/// <param name="REZ">Path to the result file</param>
/// <param name="collector">Collector to print</param>
/// <param name="title">Title of the section</param>
/// <returns>Formatted string output</returns>
static string Print(string REZ, Collector collector, string title)
{
    int tableWidth = TABLE.Split('\n')[0].Length;
    int titlePadding = Math.Max(0, (tableWidth - title.Length) / 2);
    string centeredTitle1 = new string(' ', titlePadding) + title;
    string output = $"\n{centeredTitle1}\n{TABLE}";
    using (StreamWriter rez = new StreamWriter(File.Open(REZ, FileMode.Append)))
    {
        foreach (Coin coin in collector)
        {
            output += coin.ToString();
        }
```

```
            output += "|-----------------------------------|\n";
            rez.WriteLine(output);
        }
        return output;
    }
    /// <summary>
    /// Finds and returns the heaviest coin from the collector
    /// </summary>
    /// <param name="collector">Collector with the coins</param>
    /// <returns>The heaviest coin</returns>
    static Coin HeaviestCoin(Collector collector)
    {
        collector.First();
        double heaviest = collector.GetData().Weight;
        Coin heavy = collector.GetData();
        for (collector.First(); collector.Exist(); collector.Next())
        {
            if (collector.GetData().Weight > heaviest)
            {
                heaviest = collector.GetData().Weight;
                heavy = collector.GetData();
            }
        }
        return heavy;
    }
    /// <summary>
    /// Filters and returns all coins from the collector that are as heavy as the
     given coin
    /// </summary>
    /// <param name="heavy">The heaviest coin</param>
    /// <param name="collector">Collector containing the coins</param>
    /// <returns>Collector with only the heaviest coins</returns>
    static Collector HeaviestCoins(Coin heavy, Collector collector)
    {
        Collector heaviestCoins = new Collector();
        foreach (Coin coin in collector)
        {
            if (coin.Weight == heavy.Weight)
            {
                heaviestCoins.AddToTheEnd(coin);
            }
        }
        return heaviestCoins;
    }
    /// <summary>
    /// Calculates and returns the total monetary value of all coins in the collector
    /// </summary>
    /// <param name="collector">Collector containing the coins</param>
    /// <returns>Total monetary value of coins</returns>
    static int TotalMonetaryValue(Collector collector)
    {
        int value = 0;
        foreach (Coin coin in collector)
        {
            value += coin.Denomination;
        }
        return value;
    }
    /// <summary>
    /// Filters coins based on denomination and returns them as a formatted string
    /// </summary>
    /// <param name="collector">Collector containing the coins</param>
    /// <param name="denomination">Denomination to filter by</param>
    /// <param name="REZ">Path to the result file</param>
    /// <returns>Formatted string of filtered coins</returns>
    static string DenominationToShow(Collector collector, int denomination, string
     REZ)
    {
```

```
        Collector filtered = new Collector();
        foreach (Coin coin in collector)
        {
            if (coin.Denomination == denomination)
            {
                filtered.AddToTheEnd(coin);
            }
        }
        if (filtered.Start == null)
        {
            using (StreamWriter rez = new StreamWriter(File.Open(REZ,
         FileMode.Append)))
            {
                rez.WriteLine($"\nNo coins with denomination {denomination}
         found.\n");
            }
            return $"\nNo coins with denomination {denomination} found\n";
        }
        return Print(REZ, filtered, $"Coins with denomination {denomination}");
    }
    /// <summary>
    /// Filters coins from countries starting with a specific letter and returns the
         result as a string
    /// </summary>
    /// <param name="collector">Collector containing the coins</param>
    /// <param name="letter">Letter to filter by</param>
    /// <param name="REZ">Path to the result file</param>
    /// <returns>Formatted string of filtered coins</returns>
    static string Letter(Collector collector, string letter, string REZ)
    {
        Node current = collector.Start;
        Node previous = null;
        while (current != null)
        {
            if (current.Data.Country.StartsWith(letter.ToUpper()))
            {
                if (previous == null)
                {
                    collector.Start = current.Next;
                    current = collector.Start;
                }
                else
                {
                    previous.Next = current.Next;
                    current = previous.Next;
                }
            }
            else
            {
                previous = current;
                current = current.Next;
            }
        }
        return Print(REZ, collector, $"Coins not from a country starting with letter
         '{letter}'");
    }
    /// <summary>
    /// Adds coins with a specific denomination from one collector to another sorted
         collector
    /// </summary>
    /// <param name="sorted">Collector with sorted coins</param>
    /// <param name="collector">Collector to take coins from</param>
    /// <param name="denomination">Denomination to filter by</param>
    /// <param name="REZ">Path to the result file</param>
    /// <returns>Formatted string of updated sorted coins</returns>
    static string Add(Collector sorted, Collector collector, int denomination, string
         REZ)
    {
```

```
            for (collector.First(); collector.Exist(); collector.Next())
            {
                if (collector.GetData().Denomination == denomination)
                {
                    sorted.InsertInSorted(collector.GetData());
                }
            }
            return Print(REZ, sorted, $"Sorted list with added coins with denomination
              {denomination}");
        }
    }
}


using Laboras_4;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Laboras4_testai_
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void Coin_Constructor_Creates_Coin()
        {
            string expectedCountry = "Lithuania";
            int expectedDenomination = 1;
            double expectedWeight = 5.6;
            Coin coin = new Coin(expectedCountry, expectedDenomination, expectedWeight);
            Assert.AreEqual(expectedCountry, coin.Country);
            Assert.AreEqual(expectedDenomination, coin.Denomination);
            Assert.AreEqual(expectedWeight, coin.Weight);
        }

        [TestMethod]
        public void Coin_ToString_ShouldFormatCorrectly()
        {
            Coin coin = new Coin("Lithuania", 1, 5.6);
            string result = coin.ToString();
            StringAssert.Contains(result, "Lithuania");
            StringAssert.Contains(result, "1");
            StringAssert.Contains(result, "5.6");
        }
        [TestMethod]
        public void Node_Constructor_Creates_Node()
        {
            Coin coin = new Coin("Lithuania", 1, 5.6);
            Node node = new Node(coin, null);
            Assert.AreEqual(coin, node.Data);
            Assert.IsNull(node.Next);
        }

        [TestMethod]
        public void Node_SetNext_Sets_Next_Node()
        {
            Coin coin1 = new Coin("Lithuania", 1, 5.6);
            Coin coin2 = new Coin("Latvia", 2, 7.8);
            Node node1 = new Node(coin1, null);
            Node node2 = new Node(coin2, null);
            node1.Next = node2;
            Assert.AreEqual(node2, node1.Next);
        }
        [TestMethod]
        public void Collector_AddToTheEnd_ShouldIncreaseCount()
        {
            Collector collector = new Collector();
            Coin coin = new Coin("Lithuania", 1, 5.6);
            collector.AddToTheEnd(coin);
```

```
    int count = 0;
    Node current = collector.Start;
    while (current != null)
    {
        count++;
        current = current.Next;
    }
    Assert.AreEqual(1, count);
}

[TestMethod]
public void Collector_AddToTheEnd_Adds_Coin_To_Collector()
{
    Collector collector = new Collector();
    Coin coin = new Coin("Lithuania", 1, 5.6);
    collector.AddToTheEnd(coin);
    int actualCount = 0;
    Node current = collector.Start;
    while (current != null)
    {
        actualCount++;
        current = current.Next;
    }
    Assert.AreEqual(1, actualCount);
}
[TestMethod]
public void Collector_First_Returns_First_Coin()
{
    Collector collector = new Collector();
    Coin coin1 = new Coin("Lithuania", 1, 5.6);
    Coin coin2 = new Coin("Latvia", 2, 7.8);
    collector.AddToTheEnd(coin1);
    collector.AddToTheEnd(coin2);
    collector.First();
    Assert.AreEqual(coin1, collector.GetData());
}

[TestMethod]
public void Collector_Exist_Returns_True_If_There_Are_Coins()
{
    Collector collector = new Collector();
    Coin coin = new Coin("Lithuania", 1, 5.6);
    collector.AddToTheEnd(coin);
    collector.First();
    Assert.IsTrue(collector.Exist());
}
[TestMethod]
public void Collector_Exist_Returns_False_If_No_Coins()
{
    Collector collector = new Collector();
    collector.First();
    Assert.IsFalse(collector.Exist());
}

[TestMethod]
public void Collector_Next_Moves_To_Next_Coin()
{
    Collector collector = new Collector();
    Coin coin1 = new Coin("Lithuania", 1, 5.6);
    Coin coin2 = new Coin("Latvia", 2, 7.8);
    collector.AddToTheEnd(coin1);
    collector.AddToTheEnd(coin2);
    collector.First();
    collector.Next();
    Assert.AreEqual(coin2, collector.GetData());
}
[TestMethod]
public void Collector_ShouldStoreCoinCorrectly()
```

```
        {
            Collector collector = new Collector();
            Coin expectedCoin = new Coin("Lithuania", 1, 5.6);
            collector.AddToTheEnd(expectedCoin); // Pirma turi būti Lithuania
            collector.AddToTheEnd(new Coin("Latvia", 2, 7.8));
            Coin actualCoin = collector.Start.Data; // Tikrinam pirmą monetą
            Assert.AreEqual(expectedCoin.Country, actualCoin.Country);
            Assert.AreEqual(expectedCoin.Denomination, actualCoin.Denomination);
            Assert.AreEqual(expectedCoin.Weight, actualCoin.Weight);
        }
    }
}
```

5.6. Pradiniai duomenys ir rezultatai

Testas 1:

Pirmas failas:

```
Antanas Antanaitis
Šveicarija 3 10
Lietuva 5 14
```

Antras failas:

```
Petras Petraitis
Latvija 1 12
Bulgarija 10 20
```

Rezultatai:

```
          Antanas Antanaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Šveicarija  | 3            | 10     |
| Lietuva     | 5            | 14     |
|-------------------------------------|


           Petras Petraitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Latvija     | 1            | 12     |
| Bulgarija   | 10           | 20     |
|-------------------------------------|


             Sorted coins
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Bulgarija   | 10           | 20     |
| Lietuva     | 5            | 14     |
| Latvija     | 1            | 12     |
| Šveicarija  | 3            | 10     |
|-------------------------------------|


   First collector's heaviest coins
|-------------------------------------|
```

```
| Country     | Denomination | Weight |
|-------------------------------------|
| Lietuva     | 5            | 14     |
|-------------------------------------|


   Second collector's heaviest coins
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Bulgarija   | 10           | 20     |
|-------------------------------------|


Total monetary value:

Collector 1 - 8
Collector 2 - 11

       Coins with denomination 5
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Lietuva     | 5            | 14     |
|-------------------------------------|


Sorted list with added coins with denomination 6
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Bulgarija   | 10           | 20     |
| Ispanija    | 6            | 20     |
| Lietuva     | 5            | 14     |
| Latvija     | 1            | 12     |
| Šveicarija  | 3            | 10     |
|-------------------------------------|


Coins not from a country starting with letter 'L'
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Bulgarija   | 10           | 20     |
| Ispanija    | 6            | 20     |
| Šveicarija  | 3            | 10     |
|-------------------------------------|
```

Testas 2:

Pirmas failas:

```
Antanas Antanaitis
Šveicarija 3 10
Lietuva 3 11
```

Antras failas:

```
Petras Petraitis
Belgija 1 12
Bulgarija 10 20
```

Rezultatai

```
          Antanas Antanaitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Šveicarija  | 3            | 10     |
| Lietuva     | 3            | 11     |
|-------------------------------------|


           Petras Petraitis
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Belgija     | 1            | 12     |
| Bulgarija   | 10           | 20     |
|-------------------------------------|


             Sorted coins
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Bulgarija   | 10           | 20     |
| Belgija     | 1            | 12     |
| Lietuva     | 3            | 11     |
| Šveicarija  | 3            | 10     |
|-------------------------------------|


   First collector's heaviest coins
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Lietuva     | 3            | 11     |
|-------------------------------------|


   Second collector's heaviest coins
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Bulgarija   | 10           | 20     |
|-------------------------------------|


Total monetary value:

Collector 1 - 6
Collector 2 - 11

       Coins with denomination 3
|-------------------------------------|
| Country     | Denomination | Weight |
```

```
|-------------------------------------|
| Lietuva     | 3            | 11     |
| Šveicarija  | 3            | 10     |
|-------------------------------------|


Sorted list with added coins with denomination 6
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Bulgarija   | 10           | 20     |
| Ispanija    | 6            | 20     |
| Belgija     | 1            | 12     |
| Lietuva     | 3            | 11     |
| Šveicarija  | 3            | 10     |
|-------------------------------------|


Coins not from a country starting with letter 'B'
|-------------------------------------|
| Country     | Denomination | Weight |
|-------------------------------------|
| Ispanija    | 6            | 20     |
| Lietuva     | 3            | 11     |
| Šveicarija  | 3            | 10     |
|-------------------------------------|
```

5.7. Dėstytojo pastabos

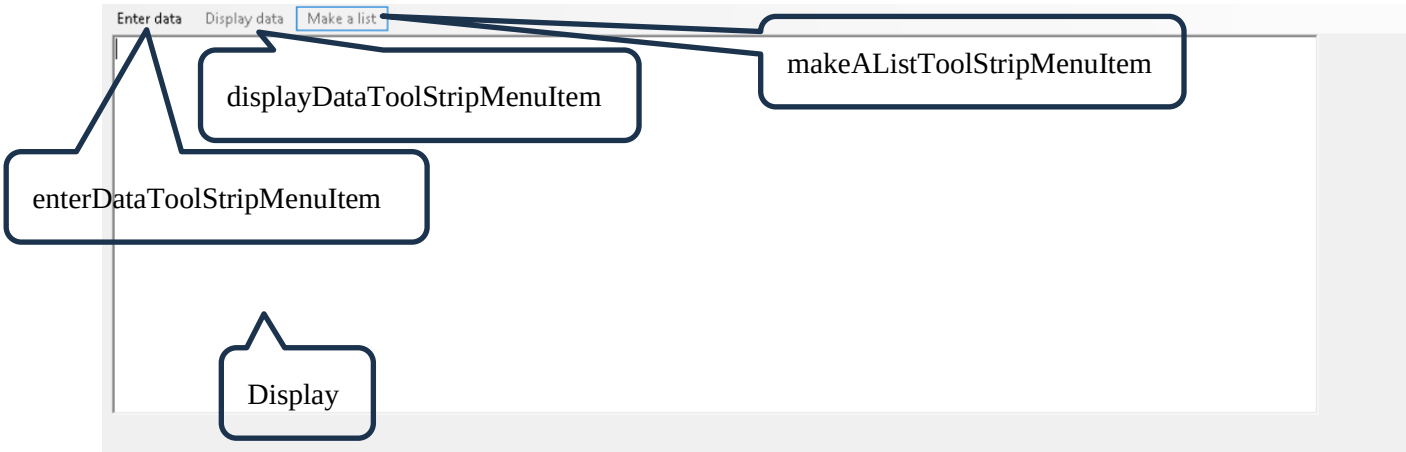
6. Bendrinės klasės (L5)

6.1. Darbo užduotis

Trigonometrija. Yra duomenys apie stačiakampius ir trikampius. Sudarykite sąrašą stačiakampių, kurių viduje yra viena trikampio viršūnė: stačiakampio vardas, jo koordinatės, trikampio vardas ir kampo koordinatės. Duomenys:

- Tekstiniame faile U4a.txt yra duomenys apie stačiakampius: vardas, viršutinio kairiojo ir apatinio dešiniojo kampų koordinatės.
- Tekstiniame faile U4b.txt yra duomenys apie trikampius: vardas ir visų trijų kampų koordinatės.

6.2. Grafinės vartotojo sąsajos schema ir paveikslas



6.3. Sąsajoje panaudotų komponentų keičiamos savybės

Komponentas	Savybė	Reikšmė
Display	Font	Courier New, 10.2pt

6.4. Programos vartotojo vadovas

enterDataToolStripMenuItem – įveda duomenis
displayDataToolStripMenuItem – parodo duomenis
makeAListToolStripMenuItem – sudaro sąrašą stačiakampių, kurių viduje yra vienas trikampio kampas

6.5. Programos tekstas

```
abstract class Angle
{
    /// <summary>
    /// X coordinate.
    /// </summary>
    public double x { get; set; }

    /// <summary>
    /// Y coordinate.
```

```
    /// </summary>
    public double y { get; set; }

    /// <summary>
    /// Initializes a new instance of the Angle class with default values.
    /// </summary>
    public Angle() { }

    /// <summary>
    /// Initializes a new instance of the Angle class with specified coordinates.
    /// </summary>
    /// <param name="x">The X coordinate.</param>
    /// <param name="y">The Y coordinate.</param>
    public Angle(double x, double y)
    {
        this.x = x;
        this.y = y;
    }
}

class Rectangle : Angle
{
    /// <summary>
    /// Name of the rectangle.
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// Upper-right corner of the rectangle.
    /// </summary>
    public Angle UpRight { get; set; }

    /// <summary>
    /// Lower-left corner of the rectangle.
    /// </summary>
    public Angle DownLeft { get; set; }

    /// <summary>
    /// Initializes a new instance of the Rectangle class with default values.
    /// </summary>
    public Rectangle() { }

    /// <summary>
    /// Initializes a new instance of the Rectangle class with specific values.
    /// </summary>
    /// <param name="upRight">Upper-right corner coordinates.</param>
    /// <param name="downLeft">Lower-left corner coordinates.</param>
    /// <param name="x">Inherited X coordinate.</param>
    /// <param name="y">Inherited Y coordinate.</param>
    /// <param name="name">Name of the rectangle.</param>
    public Rectangle(Angle upRight, Angle downLeft, double x, double y, string
        name) : base(x, y)
    {
        UpRight = upRight;
        DownLeft = downLeft;
        Name = name;
    }

    /// <summary>
    /// Returns a formatted string representation of the rectangle.
    /// </summary>
    /// <returns>A string with rectangle name and corner coordinates.</returns>
    public override string ToString()
    {
```

```
        return $"| {Name} | {$"({UpRight.x}, {UpRight.y})",-22} |
        {$"({DownLeft.x}, {DownLeft.y})",-23} |\n|------------------------------
        ---------------------------------------|\n";
    }
}

class Triangle : Angle
{
    /// <summary>
    /// Name of the triangle.
    /// </summary>
    public string Name { get; set; }

    /// <summary>
    /// First vertex of the triangle.
    /// </summary>
    public Angle A { get; set; }

    /// <summary>
    /// Second vertex of the triangle.
    /// </summary>
    public Angle B { get; set; }

    /// <summary>
    /// Third vertex of the triangle.
    /// </summary>
    public Angle C { get; set; }

    /// <summary>
    /// Initializes a new instance of the Triangle class with default values.
    /// </summary>
    public Triangle() { }

    /// <summary>
    /// Initializes a new instance of the Triangle class with specified points and
        name.
    /// </summary>
    /// <param name="a">First vertex.</param>
    /// <param name="b">Second vertex.</param>
    /// <param name="c">Third vertex.</param>
    /// <param name="x">Inherited X coordinate.</param>
    /// <param name="y">Inherited Y coordinate.</param>
    /// <param name="name">Name of the triangle.</param>
    public Triangle(Angle a, Angle b, Angle c, double x, double y, string name) :
            base(x, y)
    {
        A = a;
        B = b;
        C = c;
        Name = name;
    }

    /// <summary>
    /// Returns a formatted string representation of the triangle.
    /// </summary>
    /// <returns>A string with triangle name and vertex coordinates.</returns>
    public override string ToString()
    {
        return $"| {Name} | {$"({A.x}, {A.y})",-19} | {$"({B.x}, {B.y})",-20} |
        {$"({C.x}, {C.y})",-19} |\n|------------------------------------------
        ---------------------------------------|\n";
    }
}

class ConcreteAngle : Angle
{
```

```
    /// <summary>
    /// Initializes a new ConcreteAngle with specified coordinates.
    /// </summary>
    /// <param name="x">The X coordinate.</param>
    /// <param name="y">The Y coordinate.</param>
    public ConcreteAngle(double x, double y) : base(x, y) { }
}

/// <summary>
/// A generic container for storing a list of elements, such as geometric shapes.
/// </summary>
/// <typeparam name="T">The type of elements to store. Can be any type, but
    typically used with types derived from <see cref="Angle"/> (e.g., <see
    cref="Rectangle"/>, <see cref="Triangle"/>).</typeparam>
class Figures<T>
{
    /// <summary>
    /// Gets the linked list of elements stored in this container.
    /// </summary>
    public LinkedList<T> FigureList { get; private set; }

    /// <summary>
    /// Initializes a new instance of the <see cref="Figures{T}"/> class with an
        empty list.
    /// </summary>
    public Figures()
    {
        FigureList = new LinkedList<T>();
    }
}


public partial class Form1 : Form
{
    private Dictionary<string, Figures<Triangle>> triangles = new
    Dictionary<string, Figures<Triangle>>();
    private Dictionary<string, Figures<Rectangle>> rectangles = new
    Dictionary<string, Figures<Rectangle>>();
    const string RECTANGLETABLE = "|-----------------------------------------------
    --------------------------|\n" +
                                  "| Name                | Up right corner (x, y)
    | Down left corner (x, y) |\n" +
                                  "|-----------------------------------------------
    --------------------------|\n";
    const string TRIANGLETABLE = "|-----------------------------------------------
    -------------------------------------|\n" +
                                 "| Name              | First corner (x, y) |
    Second corner (x, y) | Third corner (x, y) |\n" +
                                 "|-----------------------------------------------
    -------------------------------------|\n";
    string REZ;
    public Form1()
    {
        InitializeComponent();
        enterDataToolStripMenuItem.Enabled = true;
        displayDataToolStripMenuItem.Enabled = false;
        makeAListToolStripMenuItem.Enabled = false;
    }

    /// <summary>
    /// Opens file dialogs to select and read rectangle and triangle data files.
    /// Then displays the contents in the output area if both files are
        successfully loaded.
    /// Enables options to display data and make a filtered list.
    /// </summary>
    private void enterDataToolStripMenuItem_Click(object sender, EventArgs e)
```

```
{
    string dataFile1 = string.Empty, dataFile2 = string.Empty;
    OpenFileDialog openFileDialog1 = new OpenFileDialog
    {
        Filter = "All files (*.*)|*.*",
        Title = "Choose the rectangle file"
    };
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        dataFile1 = openFileDialog1.FileName;
        ReadRectangles(dataFile1, rectangles);
    }
    OpenFileDialog openFileDialog2 = new OpenFileDialog
    {
        Filter = "All files (*.*)|*.*",
        Title = "Choose the triangle file"
    };
    if (openFileDialog2.ShowDialog() == DialogResult.OK)
    {
        dataFile2 = openFileDialog2.FileName;
        ReadTriangles(dataFile2, triangles);
    }
    if (!string.IsNullOrEmpty(dataFile1) && !string.IsNullOrEmpty(dataFile2))
    {
        enterDataToolStripMenuItem.Enabled = false;
        displayDataToolStripMenuItem.Enabled = true;
        makeAListToolStripMenuItem.Enabled = true;
        display.Text = "Rectangles:\n" + File.ReadAllText(dataFile1) +
        "\n\nTriangles:\n" + File.ReadAllText(dataFile2);
    }
}

/// <summary>
/// Opens a file dialog to select a result file and displays the rectangle and
    triangle data in formatted output.
/// </summary>
private void displayDataToolStripMenuItem_Click(object sender, EventArgs e)
{
    OpenFileDialog openFileDialog1 = new OpenFileDialog
    {
        Filter = "All files (*.*)|*.*",
        Title = "Choose the rezults file"
    };
    if (openFileDialog1.ShowDialog() == DialogResult.OK)
    {
        REZ = openFileDialog1.FileName;
        string output = PrintRectangles(REZ, rectangles, "Rectangles");
        output += PrintTriangles(REZ, triangles, "Triangles");
        display.Text = output;
    }
}

/// <summary>
/// Filters rectangles that contain exactly one point from any triangle.
/// Displays either the filtered rectangles or "Not found" if none match.
/// Writes results to the specified output file.
/// </summary>
private void makeAListToolStripMenuItem_Click(object sender, EventArgs e)
{
    FilterRectangles(triangles, rectangles);
    string output;
    if (rectangles.Count == 0)
    {
        int tableWidth = RECTANGLETABLE.Split('\n')[0].Length;
        int titlePadding = Math.Max(0, (tableWidth - "Not found".Length) / 2);
```

```
            string centeredTitle1 = new string(' ', titlePadding) + "Not found" +
            new string(' ', titlePadding - 1);
            output = $"\n{RECTANGLETABLE}|{centeredTitle1}|\n|--------------------
                        ------------------------------------------------|";
            using (StreamWriter rez = new StreamWriter(REZ))
            {
                rez.WriteLine(output);
            }
            display.Text += output;
            return;
        }

        output = PrintRectangles(REZ, rectangles, "Filtered rectangles");
        display.Text += output;
    }

    /// <summary>
    /// Reads rectangles from a given file and populates the provided dictionary
        by name.
    /// </summary>
    /// <param name="file">Path to the file containing rectangle data.</param>
    /// <param name="rectangles">Dictionary to store parsed rectangles grouped by
        name.</param>
    static void ReadRectangles(string file, Dictionary<string, Figures<Rectangle>>
                       rectangles)
    {
        using (StreamReader df = new StreamReader(file))
        {
            string line;
            while ((line = df.ReadLine()) != null)
            {
                string[] parts = line.Split(';');
                string name = parts[0].Trim();
                string upRight = parts[1].Trim();
                string downLeft = parts[2].Trim();
                double upRightX = double.Parse(upRight.Split(',')[0]);
                double upRightY = double.Parse(upRight.Split(',')[1]);
                double downLeftX = double.Parse(downLeft.Split(',')[0]);
                double downLeftY = double.Parse(downLeft.Split(',')[1]);
                Angle A = new ConcreteAngle(upRightX, upRightY);
                Angle B = new ConcreteAngle(downLeftX, downLeftY);
                Rectangle rectangle = new Rectangle(A, B, B.x, B.y, name);
                if (!rectangles.ContainsKey(name))
                {
                    rectangles[name] = new Figures<Rectangle>();
                }
                rectangles[name].FigureList.AddLast(rectangle);
            }
        }
    }

    /// <summary>
    /// Reads triangles from a given file and populates the provided dictionary by
        name.
    /// </summary>
    /// <param name="file">Path to the file containing triangle data.</param>
    /// <param name="triangles">Dictionary to store parsed triangles grouped by
        name.</param>
    static void ReadTriangles(string file, Dictionary<string, Figures<Triangle>>
            triangles)
    {
        using (StreamReader df = new StreamReader(file))
        {
            string line;
            while ((line = df.ReadLine()) != null)
            {
```

```
                string[] parts = line.Split(';');
                string name = parts[0].Trim();
                string angle1 = parts[1].Trim();
                string angle2 = parts[2].Trim();
                string angle3 = parts[3].Trim();
                double angle1X = double.Parse(angle1.Split(',')[0]);
                double angle1Y = double.Parse(angle1.Split(',')[1]);
                double angle2X = double.Parse(angle2.Split(',')[0]);
                double angle2Y = double.Parse(angle2.Split(',')[1]);
                double angle3X = double.Parse(angle3.Split(',')[0]);
                double angle3Y = double.Parse(angle3.Split(',')[1]);
                Angle A = new ConcreteAngle(angle1X, angle1Y);
                Angle B = new ConcreteAngle(angle2X, angle2Y);
                Angle C = new ConcreteAngle(angle3X, angle3Y);
                Triangle triangle = new Triangle(A, B, C, A.x, A.y, name);
                if (!triangles.ContainsKey(name))
                {
                    triangles[name] = new Figures<Triangle>();
                }
                triangles[name].FigureList.AddLast(triangle);
            }
        }
    }

/// <summary>
/// Writes and formats rectangles to a result file with a centered title and table
format.
/// </summary>
/// <param name="file">Path to the output result file.</param>
/// <param name="rectangles">Dictionary of rectangles to be written.</param>
/// <param name="title">Title displayed above the rectangle table.</param>
/// <returns>Formatted output string containing rectangle data.</returns>
static string PrintRectangles(string file, Dictionary<string, Figures<Rectangle>>
rectangles, string title)
{
    int tableWidth = RECTANGLETABLE.Split('\n')[0].Length;
    int titlePadding = Math.Max(0, (tableWidth - title.Length) / 2);
    string centeredTitle1 = new string(' ', titlePadding) + title;
    string output = $"\n{centeredTitle1}\n{RECTANGLETABLE}";
    using (StreamWriter rez = new StreamWriter(file, append: true))
    {
        foreach (var entry in rectangles)
        {
            var node = entry.Value.FigureList.Start;
            while (node != null)
            {
                output += node.Data.ToString();
                node = node.Next;
            }
        }
        rez.WriteLine(output);
    }
    return output;
}

    /// <summary>
/// Writes and formats triangles to a result file with a centered title and table
format.
/// </summary>
/// <param name="file">Path to the output result file.</param>
/// <param name="triangles">Dictionary of triangles to be written.</param>
/// <param name="title">Title displayed above the triangle table.</param>
/// <returns>Formatted output string containing triangle data.</returns>
static string PrintTriangles(string file, Dictionary<string, Figures<Triangle>>
triangles, string title)
{
```

```
    int tableWidth = TRIANGLETABLE.Split('\n')[0].Length;
    int titlePadding = Math.Max(0, (tableWidth - title.Length) / 2);
    string centeredTitle1 = new string(' ', titlePadding) + title;
    string output = $"\n{centeredTitle1}\n{TRIANGLETABLE}";
    using (StreamWriter rez = new StreamWriter(file, append: true))
    {
        foreach (var entry in triangles)
        {
            var node = entry.Value.FigureList.Start;
            while (node != null)
            {
                output += node.Data.ToString();
                node = node.Next;
            }
        }
        rez.WriteLine(output);
    }
    return output;
}
    /// <summary>
    /// Determines whether a given point (x, y) lies inside a specified rectangle.
    /// </summary>
    /// <param name="x">X-coordinate of the point.</param>
    /// <param name="y">Y-coordinate of the point.</param>
    /// <param name="rectangle">The rectangle to test against.</param>
    /// <returns>True if the point is inside the rectangle; otherwise,
       false.</returns>
    static bool IsPointInsideRectangle(double x, double y, Rectangle rectangle)
    {
        return x >= rectangle.DownLeft.x && x <= rectangle.UpRight.x &&
               y >= rectangle.DownLeft.y && y <= rectangle.UpRight.y;
    }

    /// <summary>
/// Filters the rectangles, keeping only those that contain exactly one point of
any triangle.
/// Removes non-matching rectangles from the dictionary.
/// </summary>
/// <param name="triangles">Dictionary containing triangle figures.</param>
/// <param name="rectangles">Dictionary containing rectangle figures. This will be
modified.</param>
static void FilterRectangles(Dictionary<string, Figures<Triangle>> triangles,
    Dictionary<string, Figures<Rectangle>> rectangles)
{
    foreach (var rectangleEntry in rectangles.ToList())
    {
        var rectangle = rectangleEntry.Value.FigureList.Start;
        bool matchFound = false;
        foreach (var triangleEntry in triangles)
        {
            var triangle = triangleEntry.Value.FigureList.Start;
            int insideCount = 0;
            if (IsPointInsideRectangle(triangle.Data.A.x, triangle.Data.A.y,
                rectangle.Data)) insideCount++;
            if (IsPointInsideRectangle(triangle.Data.B.x, triangle.Data.B.y,
                rectangle.Data)) insideCount++;
            if (IsPointInsideRectangle(triangle.Data.C.x, triangle.Data.C.y,
                rectangle.Data)) insideCount++;
            if (insideCount == 1)
            {
                matchFound = true;
                break;
            }
        }
        if (!matchFound)
        {
```

```
                rectangles.Remove(rectangleEntry.Key);
            }
        }
    }

using Microsoft.VisualStudio.TestTools.UnitTesting;
using _5_laboras;

namespace _5_laboras_testai
{
    [TestClass]
    public class NodeTests
    {
        [TestMethod]
        public void Node_Constructor_SetsDataCorrectly()
        {
            var node = new Node<int>(5);
            Assert.AreEqual(5, node.Data);
            Assert.IsNull(node.Next);
        }
    }

    [TestClass]
    public class LinkedListTests
    {
        [TestMethod]
        public void AddLast_AddsFirstElementCorrectly()
        {
            var list = new LinkedList<int>();
            list.AddLast(10);
            Assert.IsNotNull(list.Start);
            Assert.AreEqual(10, list.Start.Data);
        }

        [TestMethod]
        public void AddLast_AddsMultipleElementsCorrectly()
        {
            var list = new LinkedList<string>();
            list.AddLast("A");
            list.AddLast("B");
            list.AddLast("C");

            var node = list.Start;
            Assert.AreEqual("A", node.Data);
            node = node.Next;
            Assert.AreEqual("B", node.Data);
            node = node.Next;
            Assert.AreEqual("C", node.Data);
            Assert.IsNull(node.Next);
        }
    }

    [TestClass]
    public class FiguresTests
    {
        [TestMethod]
        public void Figures_Constructor_InitializesEmptyList()
        {
            var figures = new Figures<int>();
            Assert.IsNotNull(figures.FigureList);
            Assert.IsNull(figures.FigureList.Start);
        }
    }

    [TestClass]
    public class RectangleTests
```

```
{
    [TestMethod]
    public void Rectangle_Constructor_SetsPropertiesCorrectly()
    {
        var upRight = new ConcreteAngle(10, 20);
        var downLeft = new ConcreteAngle(0, 0);
        var rect = new Rectangle(upRight, downLeft, "Rect1");

        Assert.AreEqual("Rect1", rect.Name);
        Assert.AreEqual(10, rect.UpRight.x);
        Assert.AreEqual(0, rect.DownLeft.x);
    }

    [TestMethod]
    public void Rectangle_ToString_ReturnsFormattedString()
    {
        var upRight = new ConcreteAngle(10, 20);
        var downLeft = new ConcreteAngle(0, 0);
        var rect = new Rectangle(upRight, downLeft, "R");

        string result = rect.ToString();
        StringAssert.Contains(result, "(10, 20)");
        StringAssert.Contains(result, "(0, 0)");
        StringAssert.Contains(result, "R");
    }
}

[TestClass]
public class TriangleTests
{
    [TestMethod]
    public void Triangle_Constructor_SetsPropertiesCorrectly()
    {
        var a = new ConcreteAngle(1, 1);
        var b = new ConcreteAngle(2, 2);
        var c = new ConcreteAngle(3, 3);
        var tri = new Triangle(a, b, c, "T1");

        Assert.AreEqual("T1", tri.Name);
        Assert.AreEqual(1, tri.A.x);
        Assert.AreEqual(2, tri.B.x);
        Assert.AreEqual(3, tri.C.x);
    }

    [TestMethod]
    public void Triangle_ToString_ReturnsFormattedString()
    {
        var a = new ConcreteAngle(1, 2);
        var b = new ConcreteAngle(3, 4);
        var c = new ConcreteAngle(5, 6);
        var tri = new Triangle(a, b, c, "TR");

        string result = tri.ToString();
        StringAssert.Contains(result, "(1, 2)");
        StringAssert.Contains(result, "(3, 4)");
        StringAssert.Contains(result, "(5, 6)");
        StringAssert.Contains(result, "TR");
    }
}

[TestClass]
public class ConcreteAngleTests
{
    [TestMethod]
    public void ConcreteAngle_SetsCoordinatesCorrectly()
    {
```

```
            var angle = new ConcreteAngle(5.5, 6.6);
            Assert.AreEqual(5.5, angle.x);
            Assert.AreEqual(6.6, angle.y);
        }
    }

    [TestClass]
    public class AngleTests
    {
        [TestMethod]
        public void Angle_DefaultConstructor_SetsZeroValues()
        {
            var angle = new TestableAngle();
            angle.x = 1.1;
            angle.y = 2.2;

            Assert.AreEqual(1.1, angle.x);
            Assert.AreEqual(2.2, angle.y);
        }

        [TestMethod]
        public void Angle_Constructor_SetsValuesCorrectly()
        {
            var angle = new TestableAngle(3.3, 4.4);
            Assert.AreEqual(3.3, angle.x);
            Assert.AreEqual(4.4, angle.y);
        }

        private class TestableAngle : Angle
        {
            public TestableAngle() : base() { }
            public TestableAngle(double x, double y) : base(x, y) { }
        }
    }
}
```

6.6. Pradiniai duomenys ir rezultatai

Pradiniai duomenys:
Pirmas failas:

```
Pirmas staciakampis; 6,6; 2,2
Antras staciakampis; 60,60; 20,20
```

Antras failas:

```
Pirmas trikampis; 40,30; 70,80; 90,19
Antras trikampis; 49,39; 79,89; 99,19
```

Rezultatų failas:

```
                                                                  Rectangles
|-------------------------------------------------------------------------|
| Name                | Up right corner (x, y) | Down left corner (x, y)  |
|-------------------------------------------------------------------------|
| Pirmas staciakampis | (6, 6)                 | (2, 2)                   |
|-------------------------------------------------------------------------|
| Antras staciakampis | (60, 60)               | (20, 20)                 |
|-------------------------------------------------------------------------|

                                    Triangles
|-------------------------------------------------------------------------------------|
| Name             | First corner (x, y) | Second corner (x, y) | Third corner (x, y)  |
|-------------------------------------------------------------------------------------|
| Pirmas trikampis | (40, 30)            | (70, 80)             | (90, 19)             |
|-------------------------------------------------------------------------------------|
| Antras trikampis | (49, 39)            | (79, 89)             | (99, 19)             |
|-------------------------------------------------------------------------------------|

                           Filtered rectangles
|-------------------------------------------------------------------------|
| Name                | Up right corner (x, y) | Down left corner (x, y)  |
|-------------------------------------------------------------------------|
| Antras staciakampis | (60, 60)               | (20, 20)                 |
|-------------------------------------------------------------------------|
```

Pradiniai duomenys:
Pirmas failas:

```
Pirmas staciakampis; 6,6; 2,2
Antras staciakampis; 60,60; 20,20
```

Antras failas:

```
Pirmas trikampis; 400,300; 700,800; 900,190
Antras trikampis; 499,399; 799,899; 999,199
```

Rezultatų failas:

```
                              Rectangles
|-------------------------------------------------------------------|
| Name                | Up right corner (x, y) | Down left corner (x, y) |
|-------------------------------------------------------------------|
| Pirmas staciakampis | (6, 6)                 | (2, 2)                  |
|-------------------------------------------------------------------|
| Antras staciakampis | (60, 60)               | (20, 20)                |
|-------------------------------------------------------------------|

                                   Triangles
|-----------------------------------------------------------------------------------|
| Name              | First corner (x, y) | Second corner (x, y) | Third corner (x, y) |
|-----------------------------------------------------------------------------------|
| Pirmas trikampis  | (400, 300)          | (700, 800)           | (900, 190)          |
|-----------------------------------------------------------------------------------|
| Antras trikampis  | (499, 399)          | (799, 899)           | (999, 199)          |
|-----------------------------------------------------------------------------------|

|-------------------------------------------------------------------|
| Name                | Up right corner (x, y) | Down left corner (x, y) |
|-------------------------------------------------------------------|
```

Not found

6.7. Dėstytojo pastabos