TURINYS

1. Duomenų klasė

1.1. Darbo užduotis

U1-23. Juvelyrikos parduotuvė. Turite UAB „Blizgučiai“ parduodamų žiedų sąrašą. Duomenų failepateikta informacija apie žiedus: gamintojas, pavadinimas, metalas, svoris, dydis, praba, kaina.
• Raskite sunkiausią žiedą, ekrane atspausdinkite jo pavadinimą, metalą, skersmenį, svorį irprabą. Jei yra keli, spausdinkite visus.
• Raskite, kiek žiedų yra aukščiausios prabos. Informaciją apie šiuos žiedus atspausdinkiteekrane. Informacija apie lietuviškas prabas: platinos – 950; aukso – 375, 585 ir 750; sidabro –800, 830 ir 925; paladžio – 500 ir 850.
• Sudarykite visų metalų, iš kurių pagaminti žiedai, sąrašą. Metalų pavadinimus surašykite įfailą „Metalai.csv“.

1.2. Programos tekstas

Class1.cs:

```
using System;

public class Class1
{
	public Class1()
	{
	}
}
```

Properties/AssemblyInfo.cs:

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("Lab1")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("Lab1")]
[assembly: AssemblyCopyright("Copyright ©  2021")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to
COM
[assembly: Guid("f36d8152-135f-442f-9dd1-3bee6718d344")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision
Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Rings.cs:

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

namespace U123
{
    /// <summary>
    /// Information about the ring
    /// </summary>
    class Ring
    {
        /// <summary>
        /// Ring's manufacturer
        /// </summary>
        public string Manufacturer { get; set; }
        /// <summary>
        /// Ring's name
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Metal from which ring is made
        /// </summary>
        public string Metal { get; set; }
        /// <summary>
        /// Ring's weight
        /// </summary>
        public double Weight { get; set; }
        /// <summary>
        /// Ring size
        /// </summary>
        public double Size { get; set; }
        /// <summary>
        /// Ring's` carat
        /// </summary>
        public double Carat { get; set; }
        /// <summary>
        /// Ring price
        /// </summary>
        public double Price { get; set; }
        /// <summary>
        /// Ring class
        /// </summary>
        /// <param name="manufacturer">Manufacturer</param>
        /// <param name="name">Name</param>
        /// <param name="metal">Metal</param>
        /// <param name="weight">Weight</param>
        /// <param name="size">Size</param>
        /// <param name="carat">Carat</param>
        /// <param name="price">Price</param>
        public Ring(string manufacturer, string name, string metal, double weight
, double size, double carat, double price)
        {
            this.Manufacturer = manufacturer;
            this.Name = name;
            this.Metal = metal;
            this.Weight = weight;
            this.Size = size;
            this.Carat = carat;
            this.Price = price;
        }
        /// <summary>
        /// Converts class object to string
        /// </summary>
        /// <returns>A string</returns>
        public override string ToString()
        {
            string temp = string.Format("{0,-14} {1,-16} {2,-13} {3,6} {4,7}
{5,7} {6,9}", Manufacturer, Name, Metal, Weight, Size, Carat, Price);
            return temp;
        }
    }
}
```

InOutUtils.cs:

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

namespace U123
{
    class InOutUtils
    {
        /// <summary>
        /// Reads the values of rings' properties
        /// </summary>
        /// <param name="f">Line</param>
        /// <returns>Information about rings</returns>
        public static List<Ring> Read(string f)
        {
            List<Ring> A = new List<Ring>();
            string[] Lines = File.ReadAllLines(f);
            foreach (string line in Lines)
            {
                string[] Values = line.Split(';');
                string manufacturer = Values[0];
                string name = Values[1];
                string metal = Values[2];
                double weight = Convert.ToDouble(Values[3]);
                double size = Convert.ToDouble(Values[4]);
                double carat = Convert.ToDouble(Values[5]);
                double price = Convert.ToDouble(Values[6]);
                Ring temp = new Ring(manufacturer, name, metal, weight, size,
carat, price);
                A.Add(temp);
            }
            return A;
        }
        /// <summary>
        /// Prints data
        /// </summary>
        /// <param name="rings">List of rings</param>
        /// <param name="Heading">The heading</param>
        public static void PrintData(List<Ring> rings, string Heading)
        {
            Console.WriteLine(Heading);
            Console.WriteLine(new string('-', 80));
            Console.WriteLine("Gamintojas     Pavadinimas      Metalas
Svoris  Dydis   Praba     Kaina ");
            Console.WriteLine(new string('-', 80));
            for (int i = 0; i < rings.Count; i++)
            {
                Console.WriteLine(rings[i].ToString());
            }
            Console.WriteLine(new string('-', 80));
            Console.WriteLine();
        }
        /// <summary>
        /// Prints data to a file
        /// </summary>
        /// <param name="rings">List of rings</param>
        /// <param name="Heading">The heading</param>
        /// <param name="file">File</param>
        public static void PrintDataToFile(List<Ring> rings, string Heading,
string file)
        {
            using (StreamWriter cin = new StreamWriter(file))
            {
                cin.WriteLine(Heading);
                cin.WriteLine(new string('-', 80));
                cin.WriteLine("Gamintojas     Pavadinimas      Metalas
Svoris  Dydis   Praba     Kaina ");
                cin.WriteLine(new string('-', 80));
                for (int i = 0; i < rings.Count; i++)
                {
                    cin.WriteLine(rings[i].ToString());
                }
                cin.WriteLine(new string('-', 80));
                cin.WriteLine();
            }
```

```
        }
        /// <summary>
        /// Prints the values
        /// </summary>
        /// <param name="values">Rings' properties</param>
        public static void PrintValues(double values)
        {
            Console.WriteLine($"Sunkiausias žiedas: {values,-12}");
            Console.WriteLine();
        }
        /// <summary>
        /// Deletes existing file and creates new one
        /// </summary>
        /// <param name="metals">List of metals</param>
        /// <param name="file">File</param>
        public static void PrintToFile(List<string> metals, string file, double[]
metalweight)
        {
            if (File.Exists(file))
            {
                File.Delete(file);
            }
            using (StreamWriter cin = new StreamWriter(file))
            {
                for (int i = 0; i < metals.Count; i++)
                {
                    cin.WriteLine("{0}, {1}",metals[i], metalweight[i]);

                }
            }
        }
    }
}
```

TaskUtils.cs:

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

namespace U123
{
    class TaskUtils
    {
        /// <summary>
        /// Finds the heaviest ring
        /// </summary>
        /// <param name="rings">list of rings</param>
        /// <returns>heviest ring's weight</returns>
        public static double FindHeaviest(List<Ring> rings)
        {
            double heaviest = -1;
            for (int i = 0; i < rings.Count; i++)
            {
                if (rings[i].Weight > heaviest)
                {
                    heaviest = rings[i].Weight;
                }
            }
            return heaviest;
        }
        /// <summary>
        /// If there are multiple heaviest rings, puts them in a list
        /// </summary>
        /// <param name="rings">List of the rings</param>
        /// <param name="heaviest">Heaviest rings</param>
        /// <returns>List of heaviest rings</returns>
        public static List<Ring> FindMultipleHeviest(List<Ring> rings, double
heaviest)
        {
            List<Ring> MultipleHeaviest = new List<Ring>();


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

```
            {
                if (rings[i].Weight == heaviest)
                {
                    MultipleHeaviest.Add(rings[i]);
                }
            }
            return MultipleHeaviest;
        }
        /// <summary>
        /// Finds highest carat ring
        /// </summary>
        /// <param name="rings">List of rings</param>
        /// <returns>List highest carat ring</returns>
        public static List<Ring> FindHighestCarat(List<Ring> rings)
        {
            List<Ring> HighestCarat = new List<Ring>();
            for (int i = 0; i < rings.Count; i++)
            {
                if (rings[i].Metal == "auksas")
                {
                    if (rings[i].Carat == 750)
                    {
                        HighestCarat.Add(rings[i]);
                    }
                }
                else if (rings[i].Metal == "platina")
                {
                    if (rings[i].Carat == 950)
                    {
                        HighestCarat.Add(rings[i]);
                    }
                }
                else if (rings[i].Metal == "sidabras")
                {
                    if (rings[i].Carat == 925)
                    {
                        HighestCarat.Add(rings[i]);
                    }
                }
                else if (rings[i].Metal == "paladis")
                {
                    if (rings[i].Carat == 850)
                    {
                        HighestCarat.Add(rings[i]);
                    }
                }
            }
            return HighestCarat;
        }
        /// <summary>
        /// Finds all of the metals
        /// </summary>
        /// <param name="rings">List of rings</param>
        /// <returns>List of the metals</returns>
        public static List<string> AllMetals(List<Ring> rings)
        {
            List<string> allmetals = new List<string>();
            for (int i = 0; i < rings.Count; i++)
            {
                if (!allmetals.Contains(rings[i].Metal))
                {
                    allmetals.Add(rings[i].Metal);
                }
            }
            return allmetals;
        }
        /// <summary>
        /// Counts total weight of each metal
        /// </summary>
        /// <param name="rings">List of rings</param>
        /// <param name="AllMetals">List of all metals</param>
        /// <returns></returns>
        public static double[] AllWeight(List<Ring> rings, List<string> AllMetals
)
        {
            double[] allweight = new double[AllMetals.Count];
            for (int i = 0; i < rings.Count; i++)
```

```
            {
                for (int j = 0; j < AllMetals.Count; j++)
                {
                    if (rings[i].Metal == AllMetals[j])
                    {
                        allweight[j] += rings[i].Weight;
                    }
                }

            }
            return allweight;
        }

    }
}
```

Program.cs:

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

namespace U123
{
    class Program
    {
        const string CFd = "RingData.txt";
        const string csv = "Metals.csv";
        const string Data = "Data.txt";
        static void Main(string[] args)
        {
            List<Ring> rings = InOutUtils.Read(CFd);
            InOutUtils.PrintData(rings, "Visi žiedai");
            InOutUtils.PrintDataToFile(rings, "Visi žiedai", Data);
            double heaviest = TaskUtils.FindHeaviest(rings);
            List<Ring> FindMultipleHeaviest = TaskUtils.FindMultipleHeviest(rings
, heaviest);
            InOutUtils.PrintData(FindMultipleHeaviest, "Sunkiausi(as) žiedas/i");
            List<Ring> HighestCarat = TaskUtils.FindHighestCarat(rings);
            InOutUtils.PrintData(HighestCarat, "Aukščiausios prabos žiedai");
            List<string> metals = TaskUtils.AllMetals(rings);
            double[] AllWeight = TaskUtils.AllWeight(rings, metals);
            InOutUtils.PrintToFile(metals, csv, AllWeight);
        }
    }
}
```

1.3. Pradiniai duomenys ir rezultatai

1.4. Dėstytojo pastabos

2. Skaičiavimų klasė

2.1. Darbo užduotis

U2-23. Juvelyrikos parduotuvė. Turite informaciją apie dvejose juvelyrikos parduotuvėse esančius žiedus.Keičiasi duomenų formatas. Pirmoje eilutėje pavadinimas, antroje – adresas, trečioje – telefonas. Toliauinformacija apie žiedus pateikta tokiu pačiu formatu kaip L1 užduotyje.
• Raskite, kiek aukščiausios prabos žiedų yra kiekvienoje parduotuvėje; rezultatąatspausdinkite ekrane. Informacija apie lietuviškas prabas: platinos – 950; aukso – 375, 585ir 750; sidabro – 800, 830 ir 925; paladžio – 500 ir 850.
• Raskite brangiausią platinos žiedą, ekrane atspausdinkite visus jo duomenis. Jei jų yrakeletas, spausdinkite visus.
• Sudarykite 12-13 dydžio žiedų, kurių kaina iki 300 eurų, sąrašą. Į failą „Žiedai.csv“ įrašykitežiedų dydžius, metalus iš kurio jie pagaminti, prabas, svorius, kainas ir parduotuviųpavadinimus

2.2. Programos tekstas

Properties/AssemblyInfo.cs:

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("U2-23")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("U2-23")]
[assembly: AssemblyCopyright("Copyright ©  2021")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to
COM
[assembly: Guid("43589e8a-cc5c-4c06-946c-624ee430a6eb")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision
Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Ring.cs:

```
namespace U2_23
{
    class Ring
    {
        /// <summary>
        /// Shop's name
        /// </summary>
        public string Shopname { get; set; }
        /// <summary>
        /// Ring's manufacturer
```

```
        /// </summary>
        public string Manufacturer { get; set; }
        /// <summary>
        /// Ring's name
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Metal from which ring is made
        /// </summary>
        public string Metal { get; set; }
        /// <summary>
        /// Ring's weight
        /// </summary>
        public double Weight { get; set; }
        /// <summary>
        /// Ring size
        /// </summary>
        public double Size { get; set; }
        /// <summary>
        /// Ring's` carat
        /// </summary>
        public double Carat { get; set; }
        /// <summary>
        /// Ring price
        /// </summary>
        public double Price { get; set; }
        /// <summary>
        /// Ring class
        /// </summary>
        /// <param name="shopname"></param>
        /// <param name="manufacturer"></param>
        /// <param name="name"></param>
        /// <param name="metal"></param>
        /// <param name="weight"></param>
        /// <param name="size"></param>
        /// <param name="carat"></param>
        /// <param name="price"></param>
        public Ring(string shopname, string manufacturer, string name, string
metal, double weight, double size, double carat, double price)
        {
            this.Shopname = shopname;
            this.Manufacturer = manufacturer;
            this.Name = name;
            this.Metal = metal;
            this.Weight = weight;
            this.Size = size;
            this.Carat = carat;
            this.Price = price;
        }
        /// <summary>
        /// Converts class object to string
        /// </summary>
        /// <returns>A string</returns>
        public override string ToString()
        {
            string temp = string.Format("{0, 23} {1,13} {2,15} {3,10} {4,8} {5,6}
{6,7} {7,9}",Shopname, Manufacturer, Name, Metal, Weight, Size, Carat, Price);
            return temp;
        }
    }
}
```

InOutUtils.cs:

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

namespace U2_23
{
    class InOutUtils
    {
        /// <summary>
        /// Reads informations about the rings
        /// </summary>
```

```
        /// <param name="f"></param>
        /// <returns></returns>
        public static RingRegister Read(string f)
        {
            string[] Lines = File.ReadAllLines(f);
            string shopname = Lines[0];
            string adress = Lines[1];
            string phone = Lines[3];

            RingRegister A = new RingRegister(shopname, adress, phone);
            for (int i = 3; i < Lines.Length; i++)
            {
                string[] Values = Lines[i].Split(';');
                string manufacturer = Values[0];
                string ringname = Values[1];
                string metal = Values[2];
                double weight = Convert.ToDouble(Values[3]);
                double size = Convert.ToDouble(Values[4]);
                double carat = Convert.ToDouble(Values[5]);
                double price = Convert.ToDouble(Values[6]);
                Ring temp = new Ring(shopname, manufacturer, ringname, metal,
weight, size, carat, price);
                A.Add(temp);
            }
            return A;
        }
        /// <summary>
        /// Prints data of a register
        /// </summary>
        /// <param name="register"></param>
        public static void PrintData(RingRegister register)
        {
            Console.WriteLine(new string('-', 110));
            Console.WriteLine("Parduotuvės pavadinimas Gamintojas     Pavadinimas
     Metalas        Svoris  Dydis   Praba     Kaina ");
            Console.WriteLine(new string('-', 110));
            for (int i = 0; i < register.Count(); i++)
            {
                Console.WriteLine(register.Get(i).ToString());
            }
            Console.WriteLine(new string('-', 110));
            Console.WriteLine();
        }
        /// <summary>
        /// Prints data of a list
        /// </summary>
        /// <param name="rings"></param>
        public static void PrintData(List<Ring> rings)
        {
            Console.WriteLine(new string('-', 110));
            Console.WriteLine("Parduotuvės pavadinimas    Gamintojas
Pavadinimas    Metalas   Svoris  Dydis   Praba     Kaina ");
            Console.WriteLine(new string('-', 110));
            for (int i = 0; i < rings.Count; i++)
            {
                Console.WriteLine(rings[i].ToString());
            }
            Console.WriteLine(new string('-', 110));
            Console.WriteLine();
        }
        /// <summary>
        /// Deletes existing file and creates new one
        /// </summary>
        /// <param name="file"></param>
        /// <param name="rings"></param>
        public static void PrintToFile(string file, List<Ring> rings)
        {
            if (File.Exists(file))
            {
                File.Delete(file);
            }
            using (StreamWriter cin = new StreamWriter(file))
            {
                for (int i = 0; i < rings.Count; i++)
                {
                    cin.WriteLine("{0}, {1}, {2}, {3}, {4}, {5}", rings[i].Size,
rings[i].Metal, rings[i].Carat, rings[i].Weight, rings[i].Price, rings[i].
```

```
Shopname);

                }
            }
        }
    }
}
```

RingRegester.cs:

```
using System.Collections.Generic;

namespace U2_23
{
    /// <summary>
    /// Stores informations about rings
    /// </summary>
    class RingRegister
    {
        private string name;
        private string adress;
        private string phone;

        private List<Ring> rings = new List<Ring>();
        /// <summary>
        /// Stores information about the shop and rings
        /// </summary>
        /// <param name="name">Shop's name</param>
        /// <param name="adress">Shop's adress</param>
        /// <param name="phone">Shop's phone number</param>
        public RingRegister(string name, string adress, string phone)
        {
            this.name = name;
            this.adress = adress;
            this.phone = phone;
        }
        /// <summary>
        /// Adds a single ring to the current register
        /// </summary>
        /// <param name="ring">ring</param>
        public void Add(Ring ring)
        {
            rings.Add(ring);
        }
        /// <summary>
        /// Finds the most expensive platinum ring
        /// </summary>
        /// <returns>A value of the most expensive platinum ring</returns>
        public List<Ring> FindMostExpensivePlatinum()
        {
            List<Ring> expensiveRings = new List<Ring>();
            Ring mostexpensive = null;
            for (int i = 0; i < this.rings.Count; i++)
            {
                if (rings[i].Metal == "platina")
                {
                    if (mostexpensive == null || this.rings[i].Price >
mostexpensive.Price)
                    {
                        mostexpensive = this.rings[i];
                    }
                }
            }
            for (int i = 0; i < this.rings.Count; i++)
            {
                if (rings[i].Metal == "platina" && rings[i].Price ==
mostexpensive.Price)
                {
                    expensiveRings.Add(rings[i]);
                }
            }
            return expensiveRings;
        }
        /// <summary>
        /// Counts rings
```

```
/// </summary>
/// <returns>number of rings</returns>
public int Count()
{
    return this.rings.Count;
}
/// <summary>
///
/// </summary>
/// <param name="i">index</param>
/// <returns>Ring's index</returns>
public Ring Get(int i)
{
    return rings[i];
}
/// <summary>
/// Merges 2 lists
/// </summary>
/// <param name="register1">First list</param>
/// <param name="register2">Second List</param>
public RingRegister(RingRegister register1, RingRegister register2)
{
    for (int i = 0; i < register1.Count(); i++)
    {
        this.Add(register1.Get(i));
    }
    for (int i = 0; i < register2.Count(); i++)
    {
        this.Add(register2.Get(i));
    }
}
/// <summary>
/// Finds highest carat ring
/// </summary>
/// <returns>Value of highest carat ring</returns>
public int FindHighestCarat()
{
    int count = 0;
    for (int i = 0; i < rings.Count; i++)
    {
        if (rings[i].Metal == "auksas")
        {
            if (rings[i].Carat == 750)
            {
                count++;
            }
        }
        else if (rings[i].Metal == "platina")
        {
            if (rings[i].Carat == 950)
            {
                count++;
            }
        }
        else if (rings[i].Metal == "sidabras")
        {
            if (rings[i].Carat == 925)
            {
                count++;
            }
        }
        else if (rings[i].Metal == "paladis")
        {
            if (rings[i].Carat == 850)
            {
                count++;
            }
        }
    }
    return count;
}
/// <summary>
/// Filters rings by size
/// </summary>
/// <returns>Returns size </returns>
public List<Ring> FilteredBySize()
{
```

```
            List<Ring> filtered = new List<Ring>();
            for (int i = 0; i < rings.Count; i++)
            {
                if (rings[i].Size >= 12 && rings[i].Size <= 13)
                {
                    filtered.Add(rings[i]);
                }
            }
            return filtered;
        }
    }
}
```

Program.cs:

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

namespace U2_23
{
    class Program
    {
        const string CFd1 = "RingData.txt";
        const string CFd2 = "RingData2.txt";
        const string csv = "Žiedai.csv";
        static void Main(string[] args)
        {
            RingRegister register1 = InOutUtils.Read(CFd1);
            RingRegister register2 = InOutUtils.Read(CFd2);

            RingRegister merged = new RingRegister(register1, register2);
            Console.WriteLine("Iš viso aukščiausios prabos žiedų kiekis:");
            Console.WriteLine(merged.FindHighestCarat());

            Console.WriteLine("Brangiausi(as) platinos žiedas/ai");
            InOutUtils.PrintData(merged.FindMostExpensivePlatinum());

            List<Ring> filtered = merged.FilteredBySize();
            InOutUtils.PrintToFile(csv, filtered);
        }
    }
}
```

2.3. Pradiniai duomenys ir rezultatai

2.4. Dėstytojo pastabos

3. Konteineris

3.1. Darbo užduotis

U3_23. Juvelyrikos parduotuvė. Turite informaciją apie dvejose juvelyrikos parduotuvėse esančius žiedus.Keičiasi duomenų formatas. Pirmoje eilutėje pavadinimas, antroje – adresas, trečioje – telefonas. Toliauinformacija apie žiedus pateikta tokiu pačiu formatu kaip L1 užduotyje.
• Raskite brangiausią aukso žiedą, ekrane atspausdinkite visus jo duomenis. Jei jų yra keletas,spausdinkite visus.
• Raskite, kiek aukščiausios prabos žiedų yra kiekvienoje parduotuvėje; rezultatą atspausdinkiteekrane. Informacija apie lietuviškas prabas: platinos – 950; aukso – 375, 585 ir 750; sidabro – 800,830 ir 925; paladžio – 500 ir 850.
• Ar yra tokių žiedų, kuriuos galima įsigyti abejose juvelyrinėse parduotuvėse? Atspausdinkite juos įfailą „Visur.csv“.
• Sudarykite 12-13 dydžio žiedų, kurių kaina iki 300 eurų, sąrašą. Surikiuokite žiedus pagal gamintojusir kainą bei įrašykite į failą „Žiedai.csv“.

3.2. Programos tekstas

Properties/AssemblyInfo.cs:

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("U3-23")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("U3-23")]
[assembly: AssemblyCopyright("Copyright ©  2021")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to
COM
[assembly: Guid("11f8d008-61ff-4528-9dcf-99aa6dffa4d1")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision
Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Ring.cs:

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

namespace U3_23
{
```

```
    class Ring
    {
        /// <summary>
        /// Shop's name
        /// </summary>
        public string Shopname { get; set; }
        /// <summary>
        /// Ring manufacturer's adress
        /// </summary>
        public string Address { get; set; }
        /// <summary>
        /// Ring manufacturer phone number
        /// </summary>
        public string Phone { get; set; }
        /// <summary>
        /// Ring's manufacturer
        /// </summary>
        public string Manufacturer { get; set; }
        /// <summary>
        /// Ring's name
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Metal from which ring is made
        /// </summary>
        public string Metal { get; set; }
        /// <summary>
        /// Ring's weight
        /// </summary>
        public double Weight { get; set; }
        /// <summary>
        /// Ring size
        /// </summary>
        public double Size { get; set; }
        /// <summary>
        /// Ring's` carat
        /// </summary>
        public double Carat { get; set; }
        /// <summary>
        /// Ring price
        /// </summary>
        public double Price { get; set; }
        /// <summary>
        /// Ring class
        /// </summary>
        /// <param name="shopname"></param>
        /// <param name="manufacturer"></param>
        /// <param name="name"></param>
        /// <param name="metal"></param>
        /// <param name="weight"></param>
        /// <param name="size"></param>
        /// <param name="carat"></param>
        /// <param name="price"></param>
        public Ring(string shopname,string address, string phone, string
manufacturer, string name, string metal, double weight, double size, double carat
, double price)
        {
            this.Shopname = shopname;
            this.Address = address;
            this.Phone = phone;
            this.Manufacturer = manufacturer;
            this.Name = name;
            this.Metal = metal;
            this.Weight = weight;
            this.Size = size;
            this.Carat = carat;
            this.Price = price;
        }
        /// <summary>
        /// Converts class object to string
        /// </summary>
        /// <returns>A string</returns>
        public override string ToString()
        {
            string temp = string.Format("{0, 23} {1,13} {2,15} {3,10} {4,8} {5,6}
{6,7} {7,9}", Shopname, Manufacturer, Name, Metal, Weight, Size, Carat, Price);
            return temp;
```

```
        }
        public int CompareTo(Ring other)
        {
            if (this.Manufacturer.CompareTo(other.Manufacturer) == 0)
            {
                return this.Price.CompareTo(other.Price);
            }
            return this.Manufacturer.CompareTo(other.Manufacturer);
        }

        public override bool Equals(object obj)
        {
            return obj is Ring ring &&
                   Name == ring.Name;
        }
    }
}
```

InOutUtils.cs:

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

namespace U3_23
{
    class InOutUtils
    {
        /// <summary>
        /// Reads informations about the rings
        /// </summary>
        /// <param name="f"></param>
        /// <returns></returns>
        public static RingsContainer Read(string f)
        {
            string[] Lines = File.ReadAllLines(f);
            string shopname = Lines[0];
            string address = Lines[1];
            string phone = Lines[2];

            RingsContainer A = new RingsContainer();
            for (int i = 3; i < Lines.Length; i++)
            {
                string[] Values = Lines[i].Split(';');
                string manufacturer = Values[0];
                string ringname = Values[1];
                string metal = Values[2];
                double weight = Convert.ToDouble(Values[3]);
                double size = Convert.ToDouble(Values[4]);
                double carat = Convert.ToDouble(Values[5]);
                double price = Convert.ToDouble(Values[6]);
                Ring temp = new Ring(shopname, address, phone, manufacturer,
ringname, metal, weight, size, carat, price);
                A.Add(temp);
            }
            return A;
        }
        /// <summary>
        /// Prints data of a register
        /// </summary>
        /// <param name="register"></param>
        public static void PrintData(RingRegister register)
        {
            Console.WriteLine(new string('-', 110));
            Console.WriteLine("Parduotuvės pavadinimas Gamintojas     Pavadinimas
     Metalas        Svoris  Dydis   Praba     Kaina ");
            Console.WriteLine(new string('-', 110));
            for (int i = 0; i < register.Count(); i++)
            {
                Console.WriteLine(register.Get(i).ToString());
            }
            Console.WriteLine(new string('-', 110));
            Console.WriteLine();
        }
        /// <summary>
```

```
        /// Prints data of a list
        /// </summary>
        /// <param name="rings"></param>
        public static void PrintData(RingsContainer rings)
        {
            Console.WriteLine(new string('-', 110));
            Console.WriteLine("Parduotuvės pavadinimas    Gamintojas
Pavadinimas    Metalas   Svoris  Dydis   Praba     Kaina ");
            Console.WriteLine(new string('-', 110));
            for (int i = 0; i < rings.Count; i++)
            {
                Console.WriteLine(rings.Get(i).ToString());
            }
            Console.WriteLine(new string('-', 110));
            Console.WriteLine();
        }
        /// <summary>
        /// Deletes existing file and creates new one
        /// </summary>
        /// <param name="file"></param>
        /// <param name="container"></param>
        public static void PrintToFile(string file, RingsContainer container)
        {
            if (File.Exists(file))
            {
                File.Delete(file);
            }
            using (StreamWriter cin = new StreamWriter(file))
            {
                for (int i = 0; i < container.Count; i++)
                {
                    cin.WriteLine("{0}, {1}, {2}, {3}, {4}, {5}", container.Get(i
).Size, container.Get(i).Metal, container.Get(i).Carat, container.Get(i).Weight,
container.Get(i).Price, container.Get(i).Shopname);

                }
            }
        }
    }
}
```

RingsContainer.cs:

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

namespace U3_23
{
    class RingsContainer
    {
        private int Capacity;
        private Ring[] rings;
        public int Count { get; private set; }
        public RingsContainer()
        {
            this.rings = new Ring[16];
            Capacity = 16;
        }
        public void Add(Ring ring)
        {
            if (this.Count == this.Capacity)
            {
                EnsureCapacity(this.Capacity * 2);
            }
            this.rings[this.Count++] = ring;
        }
        public Ring Get(int index)
        {
            return this.rings[index];
        }
        public bool Contains(Ring ring)
        {
```

```
    for (int i = 0; i < this.Count; i++)
    {
        if (this.rings[i].Equals(ring))
        {
            return true;
        }
    }
    return false;
}
public RingsContainer(int capacity = 16)
{
    this.Capacity = capacity;
    this.rings = new Ring[capacity];
}
private void EnsureCapacity(int minimumCapacity)
{
    if (minimumCapacity > this.Capacity)
    {
        Ring[] temp = new Ring[minimumCapacity];
        for (int i = 0; i < this.Count; i++)
        {
            temp[i] = this.rings[i];
        }
        this.Capacity = minimumCapacity;
        this.rings = temp;
    }
}
public void Put(int index, Ring ring)
{
    rings[index] = ring;
}
public void Insert(int index, Ring ring)
{
    if (this.Count == this.Capacity)
    {
        EnsureCapacity(this.Capacity * 2);
    }
    Count++;
    for (int i = Count; i > index; i--)
    {
        rings[i] = rings[i - 1];
    }
    rings[index] = ring;
}
public void Remove(Ring ring)
{
    for (int i = 0; i < Count; i++)
    {
        if (ring == rings[i])
        {
            RemoveAt(i);
            break;
        }
    }
}
public void RemoveAt(int index)
{
    for (int i = index; i < Count; i++)
    {
        rings[i] = rings[i + 1];
    }
    Count--;
}
public void Sort()
{
    bool flag = true;
    while (flag)
    {
        flag = false;
        for (int i = 0; i < this.Count - 1; i++)
        {
            Ring a = this.rings[i];
            Ring b = this.rings[i + 1];
            if (a.CompareTo(b) > 0)
            {
                this.rings[i] = b;
                this.rings[i + 1] = a;
```

```
                        flag = true;
                    }
                }
            }
        }
        public RingsContainer(RingsContainer container) : this()
        {
            for (int i = 0; i < container.Count; i++)
            {
                this.Add(container.Get(i));
            }
        }
    }
}
```

RingRegisters.cs:

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

namespace U3_23
{
    class RingRegister
    {
        private string name;
        private string adress;
        private string phone;

        private RingsContainer rings = new RingsContainer();
        /// <summary>
        /// Stores information about the shop and rings
        /// </summary>
        /// <param name="name">Shop's name</param>
        /// <param name="adress">Shop's adress</param>
        /// <param name="phone">Shop's phone number</param>
        public RingRegister(string name, string adress, string phone)
        {
            this.name = name;
            this.adress = adress;
            this.phone = phone;
        }
        /// <summary>
        /// Adds a single ring to the current register
        /// </summary>
        /// <param name="ring">ring</param>
        public void Add(Ring ring)
        {
            rings.Add(ring);
        }
        /// <summary>
        /// Finds the most expensive platinum ring
        /// </summary>
        /// <returns>A value of the most expensive platinum ring</returns>
        public RingsContainer FindMostExpensiveGold()
        {
            RingsContainer expensiveRings = new RingsContainer();
            Ring mostexpensive = null;
            for (int i = 0; i < this.rings.Count; i++)
            {
                if (rings.Get(i).Metal == "auksas")
                {
                    if (mostexpensive == null || this.rings.Get(i).Price >
mostexpensive.Price)
                    {
                        mostexpensive = this.rings.Get(i);
                    }
                }
            }
            for (int i = 0; i < this.rings.Count; i++)
            {
                if (rings.Get(i).Metal == "auksas" && rings.Get(i).Price ==
mostexpensive.Price)
```

```
                {
                    expensiveRings.Add(rings.Get(i));
                }
            }
            return expensiveRings;
        }
        /// <summary>
        /// Counts rings
        /// </summary>
        /// <returns>number of rings</returns>
        public int Count()
        {
            return this.rings.Count;
        }
        /// <summary>
        ///
        /// </summary>
        /// <param name="i">index</param>
        /// <returns>Ring's index</returns>
        public Ring Get(int i)
        {
            return rings.Get(i);
        }
        /// <summary>
        /// Merges 2 lists
        /// </summary>
        /// <param name="container1">First list</param>
        /// <param name="container2">Second List</param>
        public RingRegister(RingsContainer container1, RingsContainer container2)
        {
            for (int i = 0; i < container1.Count; i++)
            {
                this.Add(container1.Get(i));
            }
            for (int i = 0; i < container2.Count; i++)
            {
                this.Add(container2.Get(i));
            }
        }
        /// <summary>
        /// Finds highest carat ring
        /// </summary>
        /// <returns>Value of highest carat ring</returns>
        public int FindHighestCaratCount()
        {
            RingsContainer filtered = FindHighestCarat();

            return filtered.Count;
        }
        /// <summary>
        /// Filters rings by size
        /// </summary>
        /// <returns>Returns size </returns>
        public RingsContainer FilteredBySize()
        {
            RingsContainer filtered = new RingsContainer();
            for (int i = 0; i < rings.Count; i++)
            {
                if (rings.Get(i).Size >= 12 && rings.Get(i).Size <= 13 && rings.
Get(i).Price < 300)
                {
                    filtered.Add(rings.Get(i));
                }
            }
            return filtered;
        }
        public RingsContainer FilteredBySame()
        {
            RingsContainer filtered = FindHighestCarat();
            RingsContainer filteredSame = new RingsContainer();

            for (int i = 0; i < filtered.Count - 1; i++)
            {
                for (int j = i + 1; j < filtered.Count; j++)
                {
                    if (filtered.Get(i) == filtered.Get(j) && filtered.Get(i).
Shopname != filtered.Get(j).Shopname)
```

```
                    {
                        filteredSame.Add(filtered.Get(i));
                        filteredSame.Add(filtered.Get(j));
                    }
                }
            }
            return filteredSame;
        }
        public RingsContainer FindHighestCarat()
        {
            RingsContainer filtered = new RingsContainer();
            for (int i = 0; i < rings.Count; i++)
            {
                if (rings.Get(i).Metal == "auksas")
                {
                    if (rings.Get(i).Carat == 750)
                    {
                        filtered.Add(Get(i));
                    }
                }
                else if (rings.Get(i).Metal == "platina")
                {
                    if (rings.Get(i).Carat == 950)
                    {
                        filtered.Add(Get(i));
                    }
                }
                else if (rings.Get(i).Metal == "sidabras")
                {
                    if (rings.Get(i).Carat == 925)
                    {
                        filtered.Add(Get(i));
                    }
                }
                else if (rings.Get(i).Metal == "paladis")
                {
                    if (rings.Get(i).Carat == 850)
                    {
                        filtered.Add(Get(i));
                    }
                }
            }
            return filtered;
        }
    }
}
```

Program.cs:

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

namespace U3_23
{
    class Program
    {
        const string CFd1 = "RingData.txt";
        const string CFd2 = "RingData2.txt";
        const string csv1 = "Žiedai.csv";
        const string csv2 = "Visur.csv";
        static void Main(string[] args)
        {
            RingsContainer container1 = InOutUtils.Read(CFd1);
            RingsContainer container2 = InOutUtils.Read(CFd2);

            RingRegister merged = new RingRegister(container1, container2);
            Console.WriteLine("Brangiausi(as) auksinis/iai žiedas/ai");
            InOutUtils.PrintData(merged.FindMostExpensiveGold());

            Console.WriteLine("Iš viso aukščiausios prabos žiedų kiekis:");
            Console.WriteLine(merged.FindHighestCaratCount());
```

```
            RingsContainer filteredSame = merged.FilteredBySame();
            InOutUtils.PrintToFile(csv2, filteredSame);

            RingsContainer filtered = merged.FilteredBySize();
            InOutUtils.PrintToFile(csv1, filtered);
        }
    }
}
```

3.3. Pradiniai duomenys ir rezultatai

3.4. Dėstytojo pastabos

4. Teksto analizė ir redagavimas

4.1. Darbo užduotis

U4H-23. Trumpiausias sakinysDviejuose tekstiniuose failuose Knyga1.txt ir Knyga2.txt duotas tekstas sudarytas iš žodžių, atskirtųskyrikliais. Skyriklių aibė žinoma ir abejuose failuose yra ta pati.Raskite ir spausdinkite faile Rodikliai.txt:
• ilgiausių žodžių, surikiuotų ilgio mažėjimo tvarka, kurie yra tik faile Knyga1.txt, bet nėra faileKnyga2.txt, sąrašą (ne daugiau nei 10 žodžių) ir jų pasikartojimo skaičių;
• trumpiausią sakinį (mažiausias žodžių kiekis), bet ne trumpesnį, nei iš 3 žodžių, jo ilgį (simboliais iržodžiais) ir vietą (sakinio pradžios eilutės numerį) pirmame ir antrame faile.Spausdinkite faile ManoKnyga.txt apjungtą tekstą, sudarytą pagal tokias taisykles:
• kopijuojamas pirmojo failo tekstas tol, kol sutinkamas pirmasis nenukopijuotas antrojo failo žodisarba pasiekiama failo pabaiga;
• kopijuojamas antrojo failo tekstas tol, kol sutinkamas pirmasis nenukopijuotas pirmojo failo žodisarba pasiekiama failo pabaiga;
• kartojama tol, kol pasiekiama abiejų failų pabaiga.

4.2. Programos tekstas

Properties/AssemblyInfo.cs:

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("U4H-23")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("U4H-23")]
[assembly: AssemblyCopyright("Copyright ©  2021")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to
COM
[assembly: Guid("8dc90b02-1b77-4073-a3d9-d026466a0482")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision
Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Program.cs:

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

namespace U4H_23
```

```
{
    class Program
    {

        static void Main(string[] args)
        {
            const string CF1 = "Knyga1.txt";
            const string CF2 = "Knyga2.txt";
            const string CF3 = "Rodikliai.txt";
            char[] punctuation = { ',', '.', ',', '!', ' ', '?', ':', ';', '(',
')' };
            char[] sentenceEnd = { '.', '?', '!' };
            List<string> words = TaskUtils.CalculateNotMatchingWords(CF1, CF2,
punctuation);
            List<Tuple<string, int>> repetitions = TaskUtils.FindTopRepetitions(
words);
            Tuple<string, int> shortest = TaskUtils.FindShortestSentence(CF1, CF2
, sentenceEnd, punctuation, CF3);
            InOut.write(repetitions, CF3, shortest.Item1, shortest.Item1.Length,
TaskUtils.GetSentenceLenghtWords(shortest.Item1, punctuation), shortest.Item2);
        }
    }
    class InOut
    {
        public static List<string> Read(string f,char[] punctuation)
        {
            List<string> A = new List<string>();
            using (StreamReader reader = new StreamReader(f))
            {
                string line;
                while ((line=reader.ReadLine())!= null)
                {
                    string[] Words = line.Split(punctuation, StringSplitOptions.
RemoveEmptyEntries);
                    A.AddRange(Words);
                }
            }
            return A;
        }
        public static List<Tuple<string, int>> ReadSentence(string f, char[]
sentenceEnd)
        {
            List<Tuple<string, int>> B = new List<Tuple<string, int>>();
            using (StreamReader reader = new StreamReader(f))
            {
                string line;
                string partialSentence = "";
                int lineCounter = 0;
                while ((line=reader.ReadLine())!= null)
                {
                    lineCounter++;
                    string[] Sentences = line.Split(sentenceEnd,
StringSplitOptions.RemoveEmptyEntries);
                    Sentences[0] = partialSentence + Sentences[0];
                    partialSentence = "";
                    if (!Regex.IsMatch(line, @"[.!?]+$"))
                    {
                        partialSentence += Sentences[Sentences.Length - 1] + " ";
                        for (int i = 0; i < Sentences.Length - 1; i++)
                        {
                            B.Add(new Tuple<string, int>(Sentences[i],
lineCounter));
                        }
                    }
                    else
                    {
                        foreach (var item in Sentences)
                        {
                            B.Add(new Tuple<string, int>(item, lineCounter));
                        }
                    }
                }
            }
            return B;
        }
        public static void write (List<Tuple<string, int>> f, string rez, string
sentence, int charCount, int wordCount, int lineNumber)
```

```
        {
            using (StreamWriter writer = new StreamWriter(rez))
            {
                for (int i = 0; i < Math.Min(10, f.Count); i++)
                {
                    writer.WriteLine("{0, 15} | {1, 0}", f[i].Item1, f[i].Item2);
                }
                writer.WriteLine("{0} {1} {2} {3}", sentence, charCount,
wordCount, lineNumber);
            }
        }
    }
    class TaskUtils
    {
        public static List<string> CalculateNotMatchingWords(string f1, string f2
, char[] punctuation)
        {
            List<string> words1 = InOut.Read(f1, punctuation);
            List<string> words2 = InOut.Read(f2, punctuation);
            List<string> result = new List<string>();
            foreach (var word in words1)
            {
                if (!words2.Contains(word))
                {
                    result.Add(word);
                }
            }
            foreach (var word in words2)
            {
                if (!words1.Contains(word))
                {
                    result.Add(word);
                }
            }
            return result;
        }
        public static Tuple<string, int> FindShortestSentence(string f1, string
f2, char[] sentenceEnd,char[] punctuation, string rez)
        {
            Tuple<string, int> shortest = null;
            List<Tuple<string, int>> sentences1 = new List<Tuple<string, int>>(
InOut.ReadSentence(f1, sentenceEnd));
            List<Tuple<string, int>> sentences2 = new List<Tuple<string, int>>(
InOut.ReadSentence(f2, sentenceEnd));
            using (StreamWriter writer = new StreamWriter(rez))
            {
                for (int i = 0;i < sentences1.Count; i++)
                {
                    for (int j = 0; j < sentences2.Count; j++)
                    {
                        if (sentences1[i].Item1 == (sentences2[j].Item1) &&
GetSentenceLenghtWords(sentences1[i].Item1, punctuation) > 3)
                        {
                            if (shortest == null)
                            {
                                shortest = sentences1[i];
                            }
                            else if (GetSentenceLenghtWords(shortest.Item1,
punctuation) > GetSentenceLenghtWords(sentences1[i].Item1, punctuation))
                            {
                                shortest = sentences1[i];
                            }
                        }
                    }
                }
            }
            return shortest;
        }
        public static int GetSentenceLenghtWords(string sentence, char[]
punctuation)
        {
            return (new List<string>(sentence.Split(punctuation,
StringSplitOptions.RemoveEmptyEntries))).Count;
        }
        public static Dictionary<string, int> FindRepetitions(List<string> words)
        {
            Dictionary<string, int> repetitions = new Dictionary<string, int>();
```

```
            foreach (string word in words)
            {
                if (!repetitions.ContainsKey(word))
                {
                    repetitions.Add(word, 0);
                }
                repetitions[word]++;
            }
            return repetitions;
        }
        public static List<Tuple<string, int>> FindTopRepetitions(List<string>
words)
        {
            Dictionary<string, int> repetitions = FindRepetitions(words);
            var repetitionList = repetitions.ToList();
            repetitionList.Sort((pair1, pair2) => -pair1.Key.Length.CompareTo(
pair2.Key.Length));
            List<Tuple<string, int>> result = new List<Tuple<string, int>>();
            for (int i = 0; i < Math.Min(10, repetitionList.Count); i++)
            {
                result.Add(new Tuple<string, int>(repetitionList[i].Key,
repetitionList[i].Value));
            }
            return result;
        }
    }
}
```

4.3. Pradiniai duomenys ir rezultatai

4.4. Dėstytojo pastabos

5. Paveldėjimas

5.1. Darbo užduotis

U5_23. Juvelyrikos parduotuvė. Turite informaciją apie trijose juvelyrikos parduotuvėse esančius žiedus.Pirmoje eilutėje – pavadinimas, antroje – adresas, trečioje – telefonas. Parduotuvėje galima įsigyti žiedų,auskarų, grandinėlių. Sukurkite klasę „Juwelry“ (savybės - gamintojas, pavadinimas, metalas, svoris, praba,kaina), kurią paveldės „Ring” (savybė – dydis), „Earrings” (savybė – užsegimo tipas) ir „Collar” (savybė – ilgis).Raskite sunkiausią žiedą, auskarus ir grandinėlę. Ekrane atspausdinkite visą informaciją apiekiekvieną jų.

- Raskite, kiek aukščiausios prabos juvelyrinių gaminių yra kiekvienoje parduotuvėje, rezultatąatspausdinkite ekrane. Informacija apie lietuviškas prabas: platinos – 950; aukso – 375, 585 ir 750;sidabro – 800, 830 ir 925; paladžio – 500 ir 850.
- Ar yra tokių juvelyrinių dirbinių, kurių galima įsigyti visose juvelyrinėse parduotuvėse?Atspausdinkite visą informaciją apie juos faile „Visur.csv“.
- Sudarykite juvelyrinių dirbinių, pigesnių nei 300 eurų, sąrašą, išrikiuokite pagal gamintoją,pavadinimą ir kainą. Visus duomenis apie juos įrašykite į failą „300.csv“.

5.2. Programos tekstas

JuwleryComparator.cs:

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

namespace U5_23
{
    internal class JuwleryComparator
    {
        public virtual int Compare(Juwelry a, Juwelry b)
        {
            return a.CompareTo(b);
        }
    }
}
```

JuwlertyComparatorByManufacturer.cs:

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

namespace U5_23
{
    internal class JuwlertyComparatorByManufacturer :JuwleryComparator
    {
        public override int Compare(Juwelry a, Juwelry b)
        {
            //if (a.Manufacturer.CompareTo(b.Manufacturer) == 0)
            //{
            //    if (a.Name.CompareTo(b.Name) == 0)
            //    {
            //        return a.Price.CompareTo(b.Price);
            //    }
            //    return a.Name.CompareTo(b.Name);
            //}
            return a.Manufacturer.CompareTo(b.Manufacturer);
        }
    }
}
```

Ring.cs:

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

namespace U5_23
{
    class Ring : Juwelry
    {
        /// <summary>
        /// Ring size
        /// </summary>
        public double Size { get; set; }
        public Ring(string shopname, string address, string phone, string
manufacturer, string name, string metal, double weight, double carat, double
price, double size) : base(shopname, address, phone, manufacturer, name, metal,
weight, carat, price)
        {
            this.Size = size;
        }

        public override string ToString()
        {
            return string.Format("| {0,-23} | {1,-7} | {2,-12} | {3,-12} |
{4,-13} | {5,-12} | {6,6} | {7,5} | {8,4} | {9,6} | {10,5} | {11,14} |", Shopname
, Address, Phone, Manufacturer, Name, Metal, Weight, Carat, Price, Size, "-", "-"
);
        }
    }
}
```

Collar.cs:

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

namespace U5_23
{
    class Collar : Juwelry
    {
        /// <summary>
        /// Collar's length
        /// </summary>
        public double Length { get; set; }
        public Collar(string shopname, string address, string phone, string
manufacturer, string name, string metal, double weight, double carat, double
price, double length) : base(shopname, address, phone, manufacturer, name, metal,
weight, carat, price)
        {
            this.Length = length;
        }
        public override string ToString()
        {
            return string.Format("| {0,-23} | {1,-7} | {2,-12} | {3,-12} |
{4,-13} | {5,-12} | {6,6} | {7,5} | {8,4} | {9,6} | {10,5} | {11,14} |", Shopname
, Address, Phone, Manufacturer, Name, Metal, Weight, Carat, Price, "-", Length,
"-");
        }
    }
}
```

Earrings.cs:

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

```
namespace U5_23
{
    class Earrings : Juwelry
    {
        /// <summary>
        /// Ear ring's clasp type
        /// </summary>
        public string ClaspType { get; set; }
        public Earrings(string shopname, string address, string phone, string
manufacturer, string name, string metal, double weight, double carat, double
price, string ClaspType) : base(shopname, address, phone, manufacturer, name,
metal, weight, carat, price)
        {
            this.ClaspType = ClaspType;
        }
        public override string ToString()
        {
            return string.Format("| {0,-23} | {1,-7} | {2,-12} | {3,-12} |
{4,-13} | {5,-12} | {6,6} | {7,5} | {8,4} | {9,6} | {10,5} | {11,14} |", Shopname
, Address, Phone, Manufacturer, Name, Metal, Weight, Carat, Price, "-", "-",
ClaspType);
        }
    }
}
```

Properties/AssemblyInfo.cs:

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

// General Information about an assembly is controlled through the following
// set of attributes. Change these attribute values to modify the information
// associated with an assembly.
[assembly: AssemblyTitle("U5_23")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("U5_23")]
[assembly: AssemblyCopyright("Copyright ©  2021")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]

// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(false)]

// The following GUID is for the ID of the typelib if this project is exposed to
COM
[assembly: Guid("a84ca236-9d2c-495a-8880-1a4ec3400367")]

// Version information for an assembly consists of the following four values:
//
//      Major Version
//      Minor Version
//      Build Number
//      Revision
//
// You can specify all the values or you can default the Build and Revision
Numbers
// by using the '*' as shown below:
// [assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

Juwelry.cs:

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

```
namespace U5_23
{
    abstract class Juwelry
    {
        /// <summary>
        /// Shop's name
        /// </summary>
        public string Shopname { get; set; }
        /// <summary>
        /// Ring manufacturer's adress
        /// </summary>
        public string Address { get; set; }
        /// <summary>
        /// Ring manufacturer phone number
        /// </summary>
        public string Phone { get; set; }
        /// <summary>
        /// Ring's manufacturer
        /// </summary>
        public string Manufacturer { get; set; }
        /// <summary>
        /// Ring's name
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Metal from which ring is made
        /// </summary>
        public string Metal { get; set; }
        /// <summary>
        /// Ring's weight
        /// </summary>
        public double Weight { get; set; }
        /// <summary>
        /// Ring's` carat
        /// </summary>
        public double Carat { get; set; }
        /// <summary>
        /// Ring price
        /// </summary>
        public double Price { get; set; }

        /// <summary>
        /// Ring class
        /// </summary>
        /// <param name="shopname"></param>
        /// <param name="manufacturer"></param>
        /// <param name="name"></param>
        /// <param name="metal"></param>
        /// <param name="weight"></param>
        /// <param name="size"></param>
        /// <param name="carat"></param>
        /// <param name="price"></param>
        public Juwelry(string shopname, string address, string phone, string
manufacturer, string name, string metal, double weight, double carat, double
price)
        {
            this.Shopname = shopname;
            this.Address = address;
            this.Phone = phone;
            this.Manufacturer = manufacturer;
            this.Name = name;
            this.Metal = metal;
            this.Weight = weight;
            this.Carat = carat;
            this.Price = price;
        }
        public override bool Equals(object obj)
        {
            return obj is Juwelry juwelry &&
                   Manufacturer == juwelry.Manufacturer &&
                   Name == juwelry.Name &&
                   Metal == juwelry.Metal &&
                   Weight == juwelry.Weight &&
                   Carat == juwelry.Carat &&
                   Price == juwelry.Price;
        }
```

```
        public override int GetHashCode()
        {
            int hashCode = 99611837;
            hashCode = hashCode * -1521134295 + EqualityComparer<string>.Default.
GetHashCode(Manufacturer);
            hashCode = hashCode * -1521134295 + EqualityComparer<string>.Default.
GetHashCode(Name);
            hashCode = hashCode * -1521134295 + EqualityComparer<string>.Default.
GetHashCode(Metal);
            hashCode = hashCode * -1521134295 + Weight.GetHashCode();
            hashCode = hashCode * -1521134295 + Carat.GetHashCode();
            hashCode = hashCode * -1521134295 + Price.GetHashCode();
            return hashCode;
        }
        public int CompareTo(Juwelry other)
        {
            if (this.Manufacturer.CompareTo(other.Manufacturer) == 0)
            {
                return this.Price.CompareTo(other.Price);
            }
            return this.Manufacturer.CompareTo(other.Manufacturer);
        }
        public override string ToString()
        {
            return base.ToString();
        }
    }
}
```

InOutUtils.cs:

```
using System;
using System.IO;
using System.Linq;

namespace U5_23
{
    class InOutUtils
    {
        /// <summary>
        /// Reads informations about the juwelry
        /// </summary>
        /// <param name="f"></param>
        /// <returns></returns>
        public static JuwelryContainer Read(string f)
        {
            string[] Lines = File.ReadAllLines(f);
            string shopname = Lines[0];
            string address = Lines[1];
            string phone = Lines[2];

            JuwelryContainer A = new JuwelryContainer();
            for (int i = 3; i < Lines.Length; i++)
            {
                string[] Values = Lines[i].Split(';');
                string type = Values[0];
                string manufacturer = Values[1];
                string name = Values[2];
                string metal = Values[3];
                double weight = Convert.ToDouble(Values[4]);
                double carat = Convert.ToDouble(Values[5]);
                double price = Convert.ToDouble(Values[6]);
                if (type == "Ring")
                {
                    double size = Convert.ToDouble(Values[7]);
                    Ring ring = new Ring(shopname, address, phone, manufacturer,
name, metal, weight, carat, price, size);
                    A.Add(ring);
                }
                else if (type == "Earrings")
                {
                    string claspType = Values[7];
                    Earrings earrings = new Earrings(shopname, address, phone,
manufacturer, name, metal, weight, carat, price, claspType);
                    A.Add(earrings);
```

```
                }
                else if (type == "Collar")
                {
                    double lenght = Convert.ToDouble(Values[7]);
                    Collar collar = new Collar(shopname, address, phone,
manufacturer, name, metal, weight, carat, price, lenght);
                    A.Add(collar);
                }
            }
            return A;
        }
        /// <summary>
        /// Prints data of a register
        /// </summary>
        /// <param name="register"></param>
        public static void PrintData(JuwelryRegister register)
        {
            Console.WriteLine(new string('-', 157));
            Console.WriteLine("| {0,0} | {1,0} | {2,0} | {3,0} | {4,0} | {5,0}|
{6,0} | {7,0} |", "Parduotuvės pavadinimas", "Gamintojas", "Pavadinimas",
"Metalas", "Svoris", "Dydis", "Praba", "Kaina");
            Console.WriteLine(new string('-', 157));
            for (int i = 0; i < register.Count(); i++)
            {
                Console.WriteLine(register.Get(i).ToString());
            }
            Console.WriteLine(new string('-', 157));
            Console.WriteLine();
        }
        /// <summary>
        /// Prints data of a list
        /// </summary>
        /// <param name="rings"></param>
        public static void PrintData(JuwelryContainer juwelry)
        {
            Console.WriteLine(new string('-', 157));
            Console.WriteLine("| {0,0} | {1,-8} | {2,0} | {3,-12} | {4,-13} |
{5,-12} | {6,0} | {7,0} | {8,0} | {9,0} | {10,0} | {11,0} |", "Parduotuvės
pavadinimas", "Adresas", "Telefono Nr.", "Gamintojas", "Pavadinimas", "Metalas",
"Svoris", "Praba", "Kaina", "Dydis", "Ilgis", "Užsegimo tipas");
            Console.WriteLine(new string('-', 157));
            for (int i = 0; i < juwelry.Count; i++)
            {
                Console.WriteLine(juwelry.Get(i).ToString());
            }
            Console.WriteLine(new string('-', 157));
            Console.WriteLine();
        }
        /// <summary>
        /// Deletes existing file and creates new one
        /// </summary>
        /// <param name="file"></param>
        /// <param name="container"></param>
        public static void PrintToFile(string file, JuwelryContainer container)
        {
            using (StreamWriter writer = new StreamWriter(file))
            {
                for (int i = 0; i < container.Count; i++)
                {
                    Juwelry juwelry = container.Get(i);
                    if (juwelry is Ring)
                    {
                        Ring ring = juwelry as Ring;
                        writer.WriteLine(String.Join(";", "Ring", ring.
Manufacturer, ring.Name, ring.Metal, ring.Weight, ring.Carat, ring.Price, ring.
Size));
                    }
                    else if (juwelry is Earrings)
                    {
                        Earrings earrings = juwelry as Earrings;
                        writer.WriteLine(String.Join(";", "Earrings", earrings.
Manufacturer, earrings.Name, earrings.Metal, earrings.Weight, earrings.Carat,
earrings.Price, earrings.ClaspType));
                    }
                    else if (juwelry is Collar)
                    {
                        Collar collar = juwelry as Collar;
```

```
                        writer.WriteLine(String.Join(";", "Collar", collar.
Manufacturer, collar.Name, collar.Metal, collar.Weight, collar.Carat, collar.
Price, collar.Length));
                    }
                }
            }
        }
    }
}
```

JuwelryContainer.cs:

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

namespace U5_23
{
    class JuwelryContainer
    {
        private int Capacity;
        private Juwelry[] juwelries;
        public int Count { get; private set; }
        public JuwelryContainer()
        {
            this.juwelries = new Juwelry[16];
            Capacity = 16;
        }
        public void Add(Juwelry juwelry)
        {
            if (this.Count == this.Capacity)
            {
                EnsureCapacity(this.Capacity * 2);
            }
            this.juwelries[this.Count++] = juwelry;
        }
        public void Add(JuwelryContainer juwelryContainer)
        {
            for (int i = 0; i < juwelryContainer.Count; i++)
            {
                Add(juwelryContainer.Get(i));
            }
        }
        public Juwelry Get(int index)
        {
            return this.juwelries[index];
        }
        public bool Contains(Juwelry juwelry)
        {
            for (int i = 0; i < this.Count; i++)
            {
                if (this.juwelries[i].Equals(juwelry))
                {
                    return true;
                }
            }
            return false;
        }
        public JuwelryContainer(int capacity = 16)
        {
            this.Capacity = capacity;
            this.juwelries = new Juwelry[capacity];
        }
        private void EnsureCapacity(int minimumCapacity)
        {
            if (minimumCapacity > this.Capacity)
            {
                Juwelry[] temp = new Juwelry[minimumCapacity];
                for (int i = 0; i < this.Count; i++)
                {
                    temp[i] = this.juwelries[i];
                }
                this.Capacity = minimumCapacity;
```

```
                this.juwelries = temp;
            }
        }
        public void Put(int index, Juwelry juwelry)
        {
            juwelries[index] = juwelry;
        }
        public void Insert(int index, Juwelry juwelry)
        {
            if (this.Count == this.Capacity)
            {
                EnsureCapacity(this.Capacity * 2);
            }
            Count++;
            for (int i = Count; i > index; i--)
            {
                juwelries[i] = juwelries[i - 1];
            }
            juwelries[index] = juwelry;
        }
        public void Remove(Juwelry juwelry)
        {
            for (int i = 0; i < Count; i++)
            {
                if (juwelry == juwelries[i])
                {
                    RemoveAt(i);
                    break;
                }
            }
        }
        public void RemoveAt(int index)
        {
            for (int i = index; i < Count; i++)
            {
                juwelries[i] = juwelries[i + 1];
            }
            Count--;
        }
        public void Sort(JuwleryComparator comparator)
        {
            bool flag = true;
            while (flag)
            {
                flag = false;
                for (int i = 0; i < this.Count - 1; i++)
                {
                    Juwelry a = this.juwelries[i];
                    Juwelry b = this.juwelries[i + 1];
                    if (comparator.Compare(a,b) > 0)
                    {
                        this.juwelries[i] = b;
                        this.juwelries[i + 1] = a;
                        flag = true;
                    }
                }
            }
        }
        public void Sort()
        {
            Sort(new JuwleryComparator());
        }
        public JuwelryContainer(JuwelryContainer container) : this()
        {
            for (int i = 0; i < container.Count; i++)
            {
                this.Add(container.Get(i));
            }
        }
    }
}
```

JuwelryRegister.cs:

```
using System;
```

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

namespace U5_23
{
    class JuwelryRegister
    {
        private string name;
        private string adress;
        private string phone;

        private JuwelryContainer juwelries = new JuwelryContainer();
        /// <summary>
        /// Stores information about the shop and rings
        /// </summary>
        /// <param name="name">Shop's name</param>
        /// <param name="adress">Shop's adress</param>
        /// <param name="phone">Shop's phone number</param>
        public JuwelryRegister(string name, string adress, string phone)
        {
            this.name = name;
            this.adress = adress;
            this.phone = phone;
        }
        /// <summary>
        /// Adds a single juwelry to the current register
        /// </summary>
        /// <param name="juwelry">juwelry</param>
        public void Add(Juwelry juwelry)
        {
            juwelries.Add(juwelry);
        }
        public void Add(JuwelryRegister juwelryRegister)
        {
            for (int i = 0; i < juwelryRegister.Count(); i++)
            {
                this.Add(juwelryRegister.Get(i));
            }
        }
        /// <summary>
        /// Finds the most expensive platinum juwelry
        /// </summary>
        /// <returns>A value of the most expensive platinum juwelry</returns>
        public JuwelryContainer FindMostExpensiveGold()
        {
            JuwelryContainer expensiveRings = new JuwelryContainer();
            Juwelry mostexpensive = null;
            for (int i = 0; i < this.juwelries.Count; i++)
            {
                if (juwelries.Get(i).Metal == "auksas" && (mostexpensive == null
|| this.juwelries.Get(i).Price > mostexpensive.Price))
                {
                    mostexpensive = this.juwelries.Get(i);
                }
            }
            for (int i = 0; i < this.juwelries.Count; i++)
            {
                if (juwelries.Get(i).Metal == "auksas" && juwelries.Get(i).Price
== mostexpensive.Price)
                {
                    expensiveRings.Add(juwelries.Get(i));
                }
            }
            return expensiveRings;
        }
        /// <summary>
        /// Counts rings
        /// </summary>
        /// <returns>number of rings</returns>
        public int Count()
        {
            return this.juwelries.Count;
        }
        /// <summary>
        ///
```

```
        /// </summary>
        /// <param name="i">index</param>
        /// <returns>Juwelry's index</returns>
        public Juwelry Get(int i)
        {
            return juwelries.Get(i);
        }
        /// <summary>
        /// Merges 2 lists
        /// </summary>
        /// <param name="container1">First list</param>
        /// <param name="container2">Second List</param>
        public JuwelryRegister(JuwelryContainer container1, JuwelryContainer
container2, JuwelryContainer container3)
        {
            for (int i = 0; i < container1.Count; i++)
            {
                this.Add(container1.Get(i));
            }
            for (int i = 0; i < container2.Count; i++)
            {
                this.Add(container2.Get(i));
            }
            for (int i = 0; i < container3.Count; i++)
            {
                this.Add(container3.Get(i));
            }
        }
        /// <summary>
        /// Finds highest carat juwelry
        /// </summary>
        /// <returns>Value of highest carat juwelry</returns>
        public int FindHighestCaratCount()
        {
            JuwelryContainer filtered = FindHighestCarat();

            return filtered.Count;
        }
        /// <summary>
        /// Filters rings by size
        /// </summary>
        /// <returns>Returns size </returns>
        public JuwelryContainer FilteredBySize()
        {
            JuwelryContainer filtered = new JuwelryContainer();
            for (int i = 0; i < juwelries.Count; i++)
            {
                Ring ring = juwelries.Get(i) as Ring;
                if (ring != null)
                {
                    if (ring.Size >= 12 && ring.Size <= 13 && ring.Price < 300)
                    {
                        filtered.Add(ring);
                    }
                }
            }
            return filtered;
        }
        /// <summary>
        ///
        /// </summary>
        /// <returns></returns>
        public JuwelryContainer FilteredBySame()
        {
            JuwelryContainer Filtered = new JuwelryContainer();
            var stores = GroupJuwelsByStore();
            foreach (var store1 in stores)
            {
                for (int i = 0; i < store1.Value.Count; i++)
                {
                    bool good = true;
                    foreach (var store2 in stores)
                    {
                        if (!store2.Value.Contains(store1.Value.Get(i)))
                        {
                            good = false;
                            break;
```

```
                        }
                    }
                    if (good && !Filtered.Contains(store1.Value.Get(i)))
                    {
                        Filtered.Add(store1.Value.Get(i));
                    }
                }
            }
            return Filtered;
        }
        /// <summary>
        /// Groups Juwels by store
        /// </summary>
        /// <returns>A dictionary</returns>
        private Dictionary<string,JuwelryContainer> GroupJuwelsByStore()
        {
            Dictionary<string, JuwelryContainer> result = new Dictionary<string,
JuwelryContainer>();
            for (int i = 0; i < juwelries.Count; i++)
            {
                Juwelry juwelry = juwelries.Get(i);
                if (result.ContainsKey(juwelry.Shopname))
                {
                    result[juwelry.Shopname].Add(juwelry);
                }
                else
                {
                    result[juwelry.Shopname] = new JuwelryContainer();
                    result[juwelry.Shopname].Add(juwelry);
                }
            }
            return result;
        }
        /// <summary>
        ///
        /// </summary>
        /// <returns></returns>
        public JuwelryContainer FindHighestCarat()
        {
            JuwelryContainer filtered = new JuwelryContainer();
            for (int i = 0; i < juwelries.Count; i++)
            {
                Juwelry juwelry = juwelries.Get(i);
                if (juwelry.Metal == "auksas" && juwelry.Carat == 750)
                {
                    filtered.Add(juwelry);
                }
                else if (juwelry.Metal == "platina" && juwelry.Carat == 950)
                {
                    filtered.Add(juwelry);
                }
                else if (juwelry.Metal == "sidabras" && juwelry.Carat == 925)
                {
                    filtered.Add(juwelry);
                }
                else if (juwelry.Metal == "paladis" && juwelry.Carat == 850)
                {
                    filtered.Add(juwelry);
                }
            }
            return filtered;
        }
        /// <summary>
        /// Finds the highest carat
        /// </summary>
        /// <returns>a container</returns>
        public Dictionary<string, int> FindHighestCaratAtAll()
        {
            Dictionary<string, int> highestByShop = new Dictionary<string, int>()
;
            JuwelryContainer highestCarat = FindHighestCarat();
            for (int i = 0; i < highestCarat.Count; i++)
            {
                Juwelry juwelry = highestCarat.Get(i);
                if (highestByShop.ContainsKey(juwelry.Shopname))
                {
                    highestByShop[juwelry.Shopname]++;
```

```
                }
                else highestByShop.Add(juwelry.Shopname, 1);
            }
            return highestByShop;
        }
        /// <summary>
        /// Finds Heaviest of each Juwel
        /// </summary>
        /// <returns>a container</returns>
        public JuwelryContainer FindHeaviest()
        {
            JuwelryContainer heaviest = new JuwelryContainer();
            Ring heaviestRing = null;
            Earrings heaviestEarrings = null;
            Collar heaviestCollar = null;

            for (int i = 0; i < juwelries.Count; i++)
            {
                Juwelry juwelry = this.juwelries.Get(i);
                if (juwelry is Ring)
                {
                    Ring ring = juwelry as Ring;
                    if (heaviestRing == null || ring.Weight > heaviestRing.Weight
)
                    {
                        heaviestRing = ring;
                    }
                }
                else if (juwelry is Earrings)
                {
                    Earrings earrings = juwelry as Earrings;
                    if (heaviestEarrings == null || earrings.Weight >
heaviestEarrings.Weight)
                    {
                        heaviestEarrings = earrings;
                    }
                }
                else if (juwelry is Collar)
                {
                    Collar collar = juwelry as Collar;
                    if (heaviestCollar == null || collar.Weight > heaviestCollar.
Weight)
                    {
                        heaviestCollar = collar;
                    }
                }
            }
            if (heaviestRing != null)
            {
                heaviest.Add(heaviestRing);
            }
            if (heaviestCollar != null)
            {
                heaviest.Add(heaviestCollar);
            }
            if (heaviestEarrings != null)
            {
                heaviest.Add(heaviestEarrings);
            }
            return heaviest;
        }
        /// <summary>
        /// Filters by price
        /// </summary>
        /// <returns>A container</returns>
        public JuwelryContainer FilterByMaxPrice(double price)
        {
            JuwelryContainer Filtered = new JuwelryContainer();
            for (int i = 0; i < juwelries.Count; i++)
            {
                Juwelry juwelry = juwelries.Get(i);
                if (juwelry.Price < price)
                {
                    Filtered.Add(juwelry);
                }
            }
            return Filtered;
```

```
        }
    }
}
```

Program.cs:

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

namespace U5_23
{
    class Program
    {
        const string CFd1 = "RingData1.txt";
        const string CFd2 = "RingData2.txt";
        const string CFd3 = "RingData3.txt";
        const string csv1 = "Visur.csv";
        const string csv2 = "300.csv";
        static void Main(string[] args)
        {
            JuwelryContainer container = InOutUtils.Read(CFd1);
            JuwelryContainer container1 = InOutUtils.Read(CFd2);
            JuwelryContainer container2 = InOutUtils.Read(CFd3);
            JuwelryRegister register = new JuwelryRegister(container,container1,
container2);

            Console.WriteLine("Sunkiausi papuošalai iš kiekvienos parduotuvės: ")
;
            InOutUtils.PrintData(register.FindHeaviest());

            Console.WriteLine("Aukščiausios prabos žiedų kiekis kiekvienoje
parduotuvėje:");
            InOutUtils.PrintData(register.FindHighestCarat());

            InOutUtils.PrintToFile(csv1 ,register.FilteredBySame());

            JuwelryContainer filtered = register.FilterByMaxPrice(300);
            filtered.Sort(new JuwlertyComparatorByManufacturer());
            InOutUtils.PrintToFile(csv2, filtered);
        }
    }
}
```

5.3. Pradiniai duomenys ir rezultatai

5.4. Dėstytojo pastabos