**118 L5 Polimorfizmas (8 var. 4 punktas)**

**Užduotis:** Turizmo informacijos centre perorganizuoti ir atskirai surašyti duomenys apie kiekviename mieste veikiančius muziejus. Keičiasi duomenų formatas. Pirmoje eilutėje – miestas, antroje – atsakingo asmens vardas ir pavardė. Turizmo informacijos centras teikia informaciją apie lankytinas vietas – muziejus, paminklus ir kita. Sukurkite klasę „LankytinaVieta“ (laukai - pavadinimas, adresas, įkūrimo ar pastatymo metai), kurią paveldės klasės “Muziejus” (papildomas laukas – tipas, 7 savaitės dienos (1 – darbo, 0 – nedarbo), požymis „turi gidą“, bilieto kaina) ir “Paminklas” (papildomas laukas – autorius, kam skirtas).

* Sudarykite ir surikiuokite naujų lankytinų vietų sąrašą, pateikdami pilną informaciją apie juos. Muziejus yra naujas, jei nuo įkūrimo prabėgo mažiau, nei 2 metai. Paminklas yra naujas, jei nuo pastatymo prabėgo mažiau nei metai. Muziejus rikiuokite pagal bilieto kainas, paminklus – pagal autorius.

Duomenys (3 failai viename kataloge):

**Places1.csv**

Kaunas

Atsakingas asmuo kaunas

M;MPavadinimas5;Adresas7;2017;Tipas1;1111100;True;5,5;

P;PPavadinimas6;Adresas8;2017;Cutorius1;KamSkirtas1;

M;MPavadinimas7;Adresas7;2016;Tipas2;1011100;False;7,5;

P;PPavadinimas8;Adresas8;2018;Autorius2;KamSkirtas;

M;MPavadinimas9;Adresas9;2017;Tipas3;2111100;True;9,5;

P;PPavadinimas6;Adresas8;2018;Butorius1;KamSkirtas1;

**Places2.csv**

Vilnius

Atsakingas asmuo vilnius

M;MPavadinimas0;Adresas1;2016;Tipas11;1001100;False;9,5;

P;PPavadinimas1;Adresas2;2015;Autorius11;KamSkirtas11;

M;MPavadinimas1;Adresas3;2015;Tipas22;1011100;False;11,5;

P;PPavadinimas2;Adresas4;2017;Autorius22;KamSkirtas22;

M;MPavadinimas3;Adresas5;2016;Tipas33;1110000;True;12,5;

P;PPavadinimas6;Adresas88;2014;Autorius111;KamSkirtas111;

M;MPavadinimas5;Adresas77;2015;Tipas111;1001100;True;5,5;

**Places3.csv**

Klaipėda

Atsakingas asmuo klaipėda

P;PPavadinimas66;Adresas660;2017;Autorius471;KamSkirtas1;

M;MPavadinimas700;Adresas700;2016;Tipas200;1011100;True;7,5;

P;PPavadinimas800;Adresas800;2017;Autorius511;KamSkirtas1;

Pradiniai duomenys ir rezultatai pritaikyti **Kauno šakai** (ekrane):

Kaunas muziejai

MPavadinimas5 Adresas7 2017 Tipas1 1111100 True 5,5

MPavadinimas7 Adresas7 2016 Tipas2 1011100 False 7,5

MPavadinimas9 Adresas9 2017 Tipas3 2111100 True 9,5

Kaunas paminklai

PPavadinimas6 Adresas8 2017 Cutorius1 KamSkirtas1

PPavadinimas8 Adresas8 2018 Autorius2 KamSkirtas

PPavadinimas6 Adresas8 2018 Butorius1 KamSkirtas1

Atrinkti muziejai (surikiuota pagal bilieto kaina)

MPavadinimas9 Adresas9 2017 Tipas3 2111100 True 9,5

MPavadinimas5 Adresas7 2017 Tipas1 1111100 True 5,5

Atrinkti paminklai (surikiuota pagal autoriu)

PPavadinimas8 Adresas8 2018 Autorius2 KamSkirtas

PPavadinimas6 Adresas8 2018 Butorius1 KamSkirtas1

**LankytinaVieta.cs**

abstract class LankytinaVieta

{

    public string Pavadinimas { get; set; }

    public string Adresas { get; set; }

    public int PastMetai { get; set; }

    public LankytinaVieta()

    {

        Pavadinimas = "";

        Adresas = "";

        DateTime dt = default(DateTime);

        PastMetai = dt.Year;

    }

    public LankytinaVieta(string Pavadinimas, string Adresas, int PastMetai)

    {

        this.Pavadinimas = Pavadinimas;

        this.Adresas = Adresas;

        this.PastMetai = PastMetai;

    }

    public LankytinaVieta(string data)

    {

        SetData(data);

    }

    public virtual void SetData(string line)

    {

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

        Pavadinimas = values[1];

        Adresas = values[2];

        PastMetai = int.Parse(values[3]);

    }

    public override string ToString()

    {

        return String.Format("{0, -15} {1,-10} {2,8}", Pavadinimas, Adresas, PastMetai);

    }

   public override bool Equals(object obj)

    {

        return this.Equals(obj as LankytinaVieta);

    }

    public bool Equals(LankytinaVieta vieta)

    {

        if (Object.ReferenceEquals(vieta, null))

        {

            return false;

        }

        if (this.GetType() != vieta.GetType())

        {

            return false;

        }

        return (PastMetai == vieta.PastMetai);

    }

    public override int GetHashCode() { return PastMetai.GetHashCode();     }

    public static bool operator ==(LankytinaVieta lhs, LankytinaVieta rhs)

    {

        if (Object.ReferenceEquals(lhs, null))

        {

            if (Object.ReferenceEquals(rhs, null))

            {

                return true;

            }

            return false;

        }

        return lhs.Equals(rhs);

    }

    public static bool operator !=(LankytinaVieta lhs, LankytinaVieta rhs)

    {

        return !(lhs == rhs);

    }

    // Rikiavimui

    abstract public bool Raktas(LankytinaVieta L);

    // Atrinkimui

    abstract public bool ArNaujas();

}

**Muziejus.cs**

class Muziejus : LankytinaVieta

{

    public string Tipas { get; set; }

    public string DarboDienos { get; set; }

    public bool Gidas { get; set; }

    public double BiletoKaina { get; set; }

    public Muziejus()

    {

        Tipas = "";

        DarboDienos = "";

        Gidas = default(Boolean);

        BiletoKaina = 0.0;

    }

    public Muziejus(string pavadinimas, string adresas, int metai,

          string tipas, string DarboDienos, bool Gidas, double BilietoKaina) : base(pavadinimas, adresas, metai)

    {

        this.Tipas = tipas;

        this.DarboDienos = DarboDienos;

        this.Gidas = Gidas;

        this.BiletoKaina = BiletoKaina;

    }

    public Muziejus(string data) : base(data)

    {

        SetData(data);

    }

    public override void SetData(string line)

    {

        base.SetData(line);

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

        Tipas = values[4];

        DarboDienos = values[5];

        Gidas = bool.Parse(values[6]);

        BiletoKaina = double.Parse(values[7]);

    }

    public override string ToString()

    {

        return base.ToString() + String.Format("{0, 15} {1,15} {2,8} {3,5}", Tipas, DarboDienos, Gidas, BiletoKaina);

    }

    public override bool Equals(object obj)

    {

       Muziejus m = obj as Muziejus;

       return this.Equals(obj as Muziejus) && BiletoKaina == m.BiletoKaina;

    }

    public bool Equals(Muziejus muz)

    {

        return base.Equals(muz);

    }

    public override int GetHashCode()

    {

        return  base.GetHashCode() ^ BiletoKaina.GetHashCode();

    }

    public static bool operator ==(Muziejus lhs, Muziejus rhs)

    {

        if (Object.ReferenceEquals(lhs, null))

        {

            if (Object.ReferenceEquals(rhs, null))

            {

                return true;

            }

            return false;

        }

        return lhs.Equals(rhs);

    }

    public static bool operator !=(Muziejus lhs, Muziejus rhs)

    {

        return !(lhs == rhs);

    }

    public override bool Raktas(LankytinaVieta L)

    {

        Muziejus M = L as Muziejus;

        return BiletoKaina <= M.BiletoKaina;

    }

    public override bool ArNaujas()

    {

        bool naujas = false;

        int einamieji = DateTime.Now.Year;

        int skirtumas = einamieji - PastMetai;

        if (skirtumas < 2) naujas = true;

        return naujas;

    }

}

**Paminklas.cs**

class Paminklas:LankytinaVieta

{

    public string Autorius { get; set; }

    public string Paskirtis { get; set; }

    public Paminklas()

    {

        Autorius = "";

        Paskirtis = "";

    }

    public Paminklas(string pavadinimas, string adresas, int metai, string autorius, string paskirtis)

        : base(pavadinimas, adresas, metai)

    {

        Autorius = autorius;

        Paskirtis = paskirtis;

    }

    public Paminklas(string data)

        : base(data)

    {

        SetData(data);

    }

    public override void SetData(string line)

    {

        base.SetData(line);

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

        Autorius = values[4];

        Paskirtis = values[5];

    }

    public override bool Raktas(LankytinaVieta klas)

    {

        Paminklas autor = klas as Paminklas;

        return (Autorius.CompareTo(autor.Autorius) >= 0);

    }

    public override bool Equals(object obj)

    {

       Paminklas p = obj as Paminklas;

       return this.Equals(obj as Paminklas) && Autorius == p.Autorius;

    }

    public bool Equals(Paminklas pam)

    {

        return base.Equals(pam);

    }

    public override int GetHashCode()

    {

        return base.GetHashCode() ^ Autorius.GetHashCode();

    }

    public override string ToString()

    {

        return base.ToString() + String.Format("{0, 20} {1,17}", Autorius, Paskirtis);

    }

    public static bool operator ==(Paminklas lhs, Paminklas rhs)

    {

        if (Object.ReferenceEquals(lhs, null))

        {

            if (Object.ReferenceEquals(rhs, null))

            {

                return true;

            }

            return false;

        }

        return lhs.Equals(rhs);

    }

    public static bool operator !=(Paminklas lhs, Paminklas rhs)

    {

        return !(lhs == rhs);

    }

    public override bool ArNaujas()

    {

        bool naujas = false;

        int einamieji = DateTime.Now.Year;

        int skirtumas = einamieji - PastMetai;

        if (skirtumas < 1) naujas = true;

        return naujas;

    }

}

**Saka.cs**

class Saka

{

    public string Miestas { get; set; }

    public string Vardas\_Pavarde { get; set; }

    private LankytinaVieta[] Vietos;

    public int KiekVietu { get; private set; }

   public Saka(string miestas = "", string vardas\_pavarde = "")

    {

        Miestas = miestas;

        Vardas\_Pavarde = vardas\_pavarde;

        Vietos = new LankytinaVieta[Program.CMax];

    }

    public Saka(int dydis)

    {

        Vietos = new LankytinaVieta[dydis];

        KiekVietu = 0;

    }

    public void PridetiVieta(LankytinaVieta vieta)

    {

        Vietos[KiekVietu++] = vieta;

    }

    public void PridetiVieta(LankytinaVieta vieta, int indeksas)

    {

        Vietos[KiekVietu++] = vieta;

    }

    public LankytinaVieta PaimtiVieta(int indeksas)

    {

        return Vietos[indeksas];

    }

    public void RikiuotiVietas()

    {

        LankytinaVieta laikinas;

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

        {

            for (int j = 0; j < KiekVietu; j++)

            {

                if (Vietos[j].Raktas(Vietos[i]))

                {

                    laikinas = Vietos[j];

                    Vietos[j] = Vietos[i];

                    Vietos[i] = laikinas;

                }

            }

        }

    }

    public static Saka operator +(Saka a, Saka b)

    {

        Saka c = new Saka(a.Miestas,a.Vardas\_Pavarde);

        for (int i = 0; i < a.KiekVietu; i++)

            c.PridetiVieta(a.Vietos[i]);

        for (int i = 0; i < b.KiekVietu; i++)

            c.PridetiVieta(b.Vietos[i]);

        return c;

    }

}

**Program.cs**

class Program

{

    public const int SakuKiekis = 10;   // Max. Saku Kiekis

    public const int CMax = 100;        // Max. vietu Kiekis

    private const string CFolder = "..//..//Duomenys";

    static void Main(string[] args)

    {

        Saka[] sakos = new Saka[SakuKiekis];

        int saku\_skaicius = 0;

        SkaitytiDuomenis(CFolder, sakos, ref saku\_skaicius);

        Console.WriteLine();

        // Demonstracija: 0 sakai t.y. Kaunui

        Console.WriteLine("{0} muziejai", sakos[0].Miestas);

        SpausdintiSakosTipa(sakos[0], typeof(Muziejus));

        Console.WriteLine();

        Console.WriteLine("{0} paminklai", sakos[0].Miestas);

        SpausdintiSakosTipa(sakos[0], typeof(Paminklas));

        // Atrinkimas (su rikiavimu)  0 sakai t.y. Kaunui

        Saka atrinktiM = new Saka(SakuKiekis \* CMax);

        AtrinktiIsSakosPagalTipa(sakos[0], typeof(Muziejus), ref atrinktiM);

        atrinktiM.RikiuotiVietas();

        Saka atrinktiP = new Saka(SakuKiekis \* CMax);

        AtrinktiIsSakosPagalTipa(sakos[0], typeof(Paminklas), ref atrinktiP);

        atrinktiP.RikiuotiVietas();

        Saka VisiAtrinkti = atrinktiM + atrinktiP;     // Naudojamas perdengtas operatorius

        Console.WriteLine();

        Console.WriteLine("Atrinkti muziejai (surikiuota pagal bilieto kaina)");

        SpausdintiSakosTipa(VisiAtrinkti, typeof(Muziejus));

        Console.WriteLine("Atrinkti paminklai (surikiuota pagal autoriu)");

        SpausdintiSakosTipa(VisiAtrinkti, typeof(Paminklas));

    }

    private static void SkaitytiDuomenis(string folder, Saka[] sakos, ref int saku\_skaicius)

    {

        string[] filePaths = Directory.GetFiles(folder, "\*.csv");

        int i = 0;

        foreach (string path in filePaths)

        {

            i++;

            SkaitytiVienaFaila(path, sakos, ref saku\_skaicius);

        }

    }

    private static void SkaitytiVienaFaila(string file, Saka[] sakos, ref int saku\_sk)

    {

        using (StreamReader reader = new StreamReader(@file, Encoding.GetEncoding(1257)))

        {

            string line = null;

            string miestas = reader.ReadLine();

            string vardas\_pavarde = reader.ReadLine();

            Saka saka = GautiSakaPagalMiesta(sakos, ref saku\_sk, miestas, vardas\_pavarde);

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

            {

                switch (line[0])

                {

                    case 'M':

                        saka.PridetiVieta(new Muziejus(line));

                        break;

                    case 'P':

                        saka.PridetiVieta(new Paminklas(line));

                        break;

                }

            }

        }

    }

    private static Saka GautiSakaPagalMiesta(Saka[] sakos, ref int saku\_sk, string miestas, string atsakingas)

    {

        for (int i = 0; i < saku\_sk; i++)

        {

            if (sakos[i].Miestas == miestas)

            {

                return sakos[i];

            }

        }

        sakos[saku\_sk++] = new Saka(miestas, atsakingas);

        return sakos[saku\_sk - 1];

    }

    private static void SpausdintiSakosTipa(Saka saka, Type t)

    {

        for (int i = 0; i < saka.KiekVietu; i++)

        {

            LankytinaVieta LV = saka.PaimtiVieta(i);

            if (LV.GetType() == t)

            {

               Console.WriteLine(LV.ToString());

            }

        }

    }

    private static void AtrinktiIsSakosPagalTipa(Saka saka, Type t, ref Saka nauja)

    {

        for (int i = 0; i < saka.KiekVietu; i++)

        {

            LankytinaVieta LV = saka.PaimtiVieta(i);

            if (LV.GetType() == t)

            {

                if (LV.ArNaujas()) nauja.PridetiVieta(LV);

            }

        }

    }

}