2. Skaičiavimo klasė

Siekiami studijų rezultatai:

- pagrindiniai objektinio programavimo principai;
- objektinio programavimo principų taikymas programų kūrimui;
- uždavinio sprendimo algoritmo sudarymas ar žinomo algoritmo pritaikymas;
- sudaryto algoritmo realizavimas C# programavimo kalba;
- programos derinimas ir testavimas.

Susipažinsite su:

- skaičiavimo klase;
- loginių operacijų užklojimu;
- metodų pakeitimu;

2.1. Klasė

Užduotis.

Gyvūnų registro projektas sulaukė pasisekimo, su pasisekimu atsirado ir naujų reikalavimų.
Papildykite **P1_0** programą galimybėmis:

- Saugoti informaciją apie paskutinę šunų paskiepijimo datą.
- Atrinkti sąrašą šunų, kurie neskiepyti, arba kurių skiepų galiojimo laikas pasibaigęs.

Duomenų faile įrašai gali kartotis. Patobulinkite programą — praleiskite pasikartojančius įrašus.
Papildomi reikalavimai:

- Pritaikykite savybes, kurios turi skaičiavimus

Pradiniai duomenys.

Pradiniai duomenys
Papildomas duomenų failas: **Vaccinations.csv**

`123;2018-07-24`
`124;2019-05-17`
`125;2019-01-01`
`320;2019-07-01`

Programos kūrimo eiga.

- Naudosime pirmame laboratoriniame darbe sukurtą projektą, kurį modifikuosime ir papildysime nauju funkcionalumu.
- Sukursime skaičiavimo klasę ***DogsRegister***, skirtą saugoti šunų registrą.
- Realizuosime metodus užduočių sprendimui.

Pasiruošimas.

- Sukurkite sprendimą (*solution*) pavadinimu **Lab2.Exercises**.
- Sukurkite jame projektą pavadinimu **Lab2. Exercises.Register**.
- Nukopijuokite ir pridėkite į projektą visus *.cs* failus iš ankstesnio projekto.
- Vardų sritį (namespace) pakeiskite į **Lab2. Exercises.Register**.

Pirmas žingsnis. Skaičiavimo klasės sukūrimas.

- Sukurkite skaičiavimams skirtą klasę ***DogsRegister*** (atskirame faile **DogsRegister.cs**).
- Klasėje paskelbkite lauką AllDogs, skirtą visų registro šunų informacijai saugoti.
- Sukurkite du konstruktorius (be parametrų ir su masyvo parametru) ir elemento papildymo į paskelbtą masyvą metodą.

```
class DogsRegister
{

    private List<Dog> AllDogs;

    public DogsRegister()
    {
        AllDogs = new List<Dog>();
    }

    public DogsRegister(List<Dog> Dogs)
    {
        AllDogs = new List<Dog>();
        foreach (Dog dog in Dogs)
        {
            this.AllDogs.Add(dog);
        }
    }

    public void Add(Dog dog)
    {
        AllDogs.Add(dog);
    }

}
```

- Klasėje `InOutUtils` parašykite skaitymo iš failo metodą, kuris perskaito duomenis į klasės `DogsRegister` objektą. Palyginkite ir pagalvokite, kuo skiriasi nuo LD1 skaitymo metodo.

```
    public static DogsRegister ReadDogs(string fileName)
    {
        DogsRegister Dogs = new DogsRegister ();
        string[] Lines = File.ReadAllLines(fileName, Encoding.UTF8);
        foreach (string line in Lines)
        {
            string[] Values = line.Split(';');
            int id = int.Parse(Values[0]);
            string name = Values[1];
            string breed = Values[2];
            DateTime birthDate = DateTime.Parse(Values[3]);

            Gender gender;
            Enum.TryParse(values[4], out gender); //tries to convert value to enum

            Dog dog = new Dog(id, name, breed, birthDate, gender);
```

```
            Dogs.Add(dog);
        }

        return Dogs;
    }
```

- Metodo Main pradžioje, paskelbkite registro objektą ir iškvieskite skaitymo metodą.

```
static void Main(string[] args)
{

    DogsRegister register = InOutUtils.ReadDogs(@"Dogs.csv");
    ...
}
```

- Klasėje ***DogsRegister*** sukurkite metodą, kuris grąžintų visų registre esančių šunų kiekį.

```
public int DogsCount()
{
    return this.AllDogs.Count;
}
```

- Panaudokite jį ***Main*** metode.

```
Console.WriteLine("Iš viso šunų: {0}", register.DogsCount());
```

1 savarankiško darbo užduotis

➢ Klasėje ***DogsRegister*** sukurkite metodą vieno masyvo elemento paėmimui pagal indeksą.

- Pertvarkyte metodą ***CountByGender***, esantį klasėje ***TaskUtils***. Perkelkite jį į klasę ***DogsRegister***, pašalinkite metodo parametrą *Dogs* (vietoje jo naudosite klasės kintamąjį), iš metodo aprašo pašalinkite *static*.

```
public int CountByGender(Gender gender)
{
    int count = 0;
    foreach (Dog dog in this.AllDogs)
    {
        if (dog.Gender.Equals(gender))
        {
            count++;
        }
    }
    return count;
}
```

- Atitinkamai paredaguokite ***Main*** metodą.

```
Console.WriteLine("Patinų: {0}", register.CountByGender(Gender.Male));
Console.WriteLine("Patelių: {0}", register.CountByGender(Gender.Female));
```

- Tokiu pat principu pertvarkykite likusius klasės ***TaskUtils*** metodus. Kai ši klasė taps nebereikalinga — pašalinkite.

Antras žingsnis. Metodų užklojimas.

- Pirmųjų pratybų savarankiška užduotis prašė surasti seniausią *pasirinktos* veislės šunį. Realizuosime šį reikalavimą panaudodami metodų užklojimą. Klasė ***DogsRegister*** turės 3 metodus ***FindOldestDog***:
    1. Atvirą, be parametrų, kuris bus skirtas surasti seniausią šunį registre.
    2. Atvirą, su string parametru, kuris bus skirtas surasti pasirinktos veislės seniausią šunį.
    3. Privatų, su šunų sąrašo parametru, kuris bus skirtas ieškoti seniausio šuns duotame sąraše.

```
public Dog FindOldestDog()
{
    return this.FindOldestDog(this.AllDogs);
}

public Dog FindOldestDog(string breed)
{
    List<Dog> Filtered = this.FilterByBreed(breed);
    return this.FindOldestDog(Filtered);
}

private Dog FindOldestDog(List<Dog> Dogs)
{
    Dog oldest = Dogs[0];
    for (int i = 1; i < Dogs.Count; i++) //starts on index value 1
    {
        if (DateTime.Compare(oldest.BirthDate, Dogs[i].BirthDate) > 0)
        {
            oldest = Dogs[i];
        }
    }
    return oldest;
}
```

Atkreipkite dėmesį, jog pirmieji du metodai kviečia trečiąjį. Trečiasis metodas padėjo išvengti to paties kodo kartojimo.

Trečias žingsnis. Papildoma duomenų klasė.

- Sukurkite duomenų klasę ***Vaccination***, skirtą skiepijimo informacijai saugoti.

```
class Vaccination
{
    public int DogID { get; set; }
    public DateTime Date { get; set; }

    public Vaccination(int dogID, DateTime date)
    {
        this.DogID = dogID;
        this.Date = date;
    }
}
```

- Klasę ***InOutUtils*** papildykite metodu, skirtu skiepijimo informacijai perskaityti.

```
public static List<Vaccination> ReadVaccinations(string fileName)
{
    List<Vaccination> Vaccinations = new List<Vaccination>();
    string[] Lines = File.ReadAllLines(fileName);
    foreach (string line in Lines)
    {
        string[] Values = line.Split(';');
        int id = int.Parse(Values[0]);
        DateTime vaccinationDate = DateTime.Parse(Values[1]);

        Vaccination v = new Vaccination(id, vaccinationDate);
        Vaccinations.Add(v);
    }

    return Vaccinations;
}
```

- Klasę ***Dog*** papildykite konstanta, kuri nurodytų skiepų galiojimo laiką.

```
private const int VaccinationDuration = 1;
```

- Klasę ***Dog*** papildykite sąvybe, skirta saugoti paskutinio skiepijimo datai.

```
public DateTime LastVaccinationDate { get; set; }
```

- Taip pat pridėkite metodą, skirtą patikrinti ar skiepas vis dar galioja.

```
public bool RequiresVaccination()
{
    if (LastVaccinationDate.Equals(DateTime.MinValue))
    {
        return true;
    }
    return LastVaccinationDate.AddYears(VaccinationDuration).CompareTo(DateTime.Now)<0;
}
```

- Klasę ***DogsRegister*** papildykite privačiu metodu, kuris ieškos šuns objekto registre pagal duotą ID.

```
private Dog FindDogByID(int ID)
{
    foreach (Dog dog in this.AllDogs)
    {
        if (dog.ID == ID)
        {
            return dog;
        }
    }
    return null;
}
```

- Klasėje ***DogsRegister*** papildykite metodu, kuris papildys registre esančius *Dog* tipo objektus paskutinio skiepijimo data.

```
public void UpdateVaccinationsInfo(List<Vaccination> Vaccinations)
{
    foreach (Vaccination vaccination in Vaccinations)
    {
        Dog dog = this.FindDogByID(vaccination.DogID);
        dog.LastVaccinationDate = vaccination.Date;
    }
}
```

- Galiausiai, papildykite ***Main*** metodą šiomis eilutėmis.

```
List<Vaccination> VaccinationsData = InOutUtils.ReadVaccinations(@"Vaccinations.csv");
register.UpdateVaccinationsInfo(VaccinationsData);
```

2 savarankiško darbo užduotis

- Parašykite metodą ***FilterByVaccinationExpired***, kuris sudarytų neskiepytų ar su nebegaliojančiais skiepais šunų sąrašą ir grąžintų šunų registro objektą.
- Įvykdykite programą ir pasitikrinkite gautus rezultatus.

Ketvirtas žingsnis. Metodų pakeitimas.

Pradinių duomenų faile įrašai apie šunis gali kartotis.

- Patobulinkite skaitymo metodą ***ReadDogs*** — neįtraukite pasikartojančių įrašų. Papildykite klasę `DogsRegister`.

```
public bool Contains(Dog dog)
{
        return AllDogs.Contains(dog);
}
```

- Tuomet metodas ***ReadDogs*** atrodys taip:

```
public static DogsRegister ReadDogs(string fileName)
{
    DogsRegister Dogs = new DogsRegister ();
    string[] Lines = File.ReadAllLines(fileName, Encoding.UTF8);
    foreach (string line in Lines)
    {
        string[] Values = line.Split(';');
        int id = int.Parse(Values[0]);
        string name = Values[1];
        string breed = Values[2];
        DateTime birthDate = DateTime.Parse(Values[3]);

        Gender gender;
        Enum.TryParse(Values[4], out gender); //tries to convert value to enum

        Dog dog = new Dog(id, name, breed, birthDate, gender);
        if (!Dogs.Contains(dog))
        {
            Dogs.Add(dog);
        }
    }

    return Dogs;
}
```

- Pakoreguokite pradinių duomenų failą — įterpkite pasikartojančių įrašų.
- Įvykdykite programą ir patikrinkite gautus rezultatus.

- Pasikartojantys įrašai vis dar yra. Objektų palyginimui klasės ***List*** metodas ***Contains*** naudoja klasės ***Object*** metodą ***Equals***. Pagal numatytąją reikšmę šis metodas tikrina, ar tai **tas pats** objektas. Klasėje ***Dog*** pakeiskite ***Equals*** metodą. Tikrinkite, ar tai **toks pat šuo**. Tikrinkite pagal registravimo ID.

```
public override bool Equals(object other)
{
    return this.ID == ((Dog)other).ID;
}
```

- Jei pakeičiamas metodas ***Equals***, būtina pakeisti ir ***GetHashCode*** metodą.

```
public override int GetHashCode()
{
    return this.ID.GetHashCode();
}
```

- Įvykdykite programą ir patikrinkite gautus rezultatus.

Penktas žingsnis. Operatorių užklojimas.

Metodas ***UpdateVaccinationsInfo*** turi trūkumą. Jei yra keli įrašai apie to paties šuns skiepus, galiausiai bus panaudotas paskutinis esantis faile. Teisingiau būtų atsižvelgti į skiepijimo datą, ir naudoti paskutinę. Ištaisykime šį trūkumą.

- Klasėje ***Vaccination*** realizuokite operatorius < ir >, kurie lygins skiepijimo įrašą su duota data.

```
public static bool operator <(Vaccination vaccination, DateTime date)
{
    return vaccination.Date.CompareTo(date) < 0;
}

public static bool operator >(Vaccination vaccination, DateTime date)
{
    return vaccination.Date.CompareTo(date) > 0;
}
```

- Panaudokime sukurtą operatorių ***UpdateVaccinationsInfo*** metode.

```
public void UpdateVaccinationsInfo(List<Vaccination> Vaccinations)
{
    foreach (Vaccination vaccination in Vaccinations)
    {
        Dog dog = this.FindDogByID(vaccination.DogID);
        if (vaccination > dog.LastVaccinationDate)
        {
            dog.LastVaccinationDate = vaccination.Date;
        }
    }
}
```

3 savarankiško darbo užduotis

- Programa neveiks, jei Vaccinations.csv turės informacijos apie šunis, kurių neturime registre. Ištaisykite šį trūkumą.

Šeštas žingsnis. Savybės, turinčios skaičiavimus.

- Klasės ***Dog*** metodus ***CalculateAge*** ir ***RequiresVaccination*** pakeiskite į sąvybes.

```
public int Age
{
    get
    {
        DateTime today = DateTime.Today;
        int age = today.Year - this.BirthDate.Year;
        if (this.BirthDate.Date > today.AddYears(-age))
        {
            age--;
        }
        return age;
    }
}

public bool RequiresVaccination
{
    get
    {
        if (LastVaccinationDate.Equals(DateTime.MinValue))
        {
            return true;
        }
        return LastVaccinationDate.AddYears(VaccinationDuration)
            .CompareTo(DateTime.Now) < 0;
    }
}
```

- Kitose klasėse pakeiskite šių metodų iškvietimus savybių reikšmių priskyrimais.

4 savarankiško darbo užduotis

Pateikiamas 9 aukštų namo parduodamų butų sąrašas. Kiekviename laiptinės aukšte yra po 3 butus. Namo laiptinių skaičius yra mažesnis, nei 20. Duomenys tokie: buto numeris, bendras plotas, kambarių skaičius, pardavimo kaina, telefono Nr. Suraskite butus, kurie turi nurodytą kambarių skaičių, yra nurodytame aukšte ir kurių kaina neviršija nurodytos kainos, ir juos surašykite į tinkamų butų klasės objektą. Aukšto numeris nurodomas intervalu [nuo, iki]. Aukšto numeris išskaičiuojamas iš buto numerio. Butų numeriai yra iš aibės [1, laiptinių skaičius daugintas iš 27].