3. Konteinerinė 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;
- programų derinimas ir testavimas.

3.1 Konteinerinė klasė, naudojant fiksuoto dydžio masyvą

Užduotis.
Pertvarkykite P2 programą: vietoj sisteminės bibliotekos klasės ***List*** naudokite savo sukurtą konteinerinę klasę.

Konteinerinė klasė privalo realizuoti standartinius ***Add***, ***Put***, ***Insert***, ***Get***, ***Remove***, ***RemoveAt***, ***Contains***, ***Sort*** metodus. Konteinerio talpa turi kisti dinamiškai.

Papildykite programą:

- Išrikiuokite šunis pagal veislę ir lytį;
- Atspausdinkite sąrašą šunų, kuriuos reikia skiepyti;
- Atspausdinkite sąrašą pasirinktos veislės šunų, kuriuos reikia skiepyti.

Programos kūrimo eiga.

- Naudosime antrame laboratoriniame darbe sukurtą projektą, kurį modifikuosime ir papildysime nauju funkcionalumu.
- Sukursime konteinerinę klasę ***DogsContainer***, kurią naudosime vietoj sisteminės bibliotekos klasės ***List***.
- Realizuosime metodus užduočių sprendimui.

Pasiruošimas.

- Sukurkite sprendimą (*solution*) pavadinimu **Lab3.Exercises**.
- Sukurkite jame projektą pavadinimu **Lab3. Exercises.Register**.
- Nukopijuokite ir pridėkite į projektą visus *.cs* failus iš ankstesnio projekto.
- Vardų sritį (namespace) pakeiskite į **Lab3. Exercises.Register**.

Pirmas žingsnis. Konteinerinės klasės sukūrimas.

- Sukurkite konteinerio funkcijas atliekančią klasę ***DogsContainer*** (atskirame faile **DogsContainer.cs**).
- Klasėje paskelbkite šunų masyvą ***dogs***.
- Klasėje paskelbkite savybę ***Count*** skirtą šunų kiekiui saugoti.
- Sukurkite konstruktorių bei sąsajos metodus ***Add*** ir ***Get***.

```
namespace Lab3.Exercises.Register
{
    class DogsContainer
    {
        private Dog[] dogs;
        public int Count { get; private set; }

        public DogsContainer()
        {
            this.dogs = new Dog[16];//default capacity
        }

        public void Add(Dog dog)
        {
            this.dogs[this.Count++] = dog;
        }

        public Dog Get(int index)
        {
            return this.dogs[index];
        }
    }
}
```

- Realizuokite metodą ***Contains***, kuris tikrins, ar konteineryje yra duotas šuo.

```
public bool Contains(Dog dog)
{
    for (int i = 0; i < this.Count; i++)
    {
        if (this.dogs[i].Equals(dog))
        {
            return true;
        }
    }
    return false;
}
```

Antras žingsnis. Konteinerio panaudojimas.

- Klasę ***List*** pakeiskite naujai sukurta klase ***DogsContainer***.
- Pertvarkykite klasės ***InOutUtils*** metodą ***ReadDogs***.

```
public static DogsContainer ReadDogs(string fileName)
{
    DogsContainer dogs = new DogsContainer();
    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;
}
```

- Pertvarkykite klasės ***InOutUtils*** metodą ***PrintDogs***.

```
public static void PrintDogs(string label, DogsContainer dogs)
{
    Console.WriteLine(new string('-', 74));
    Console.WriteLine("| {0,-70} |", label);
    Console.WriteLine(new string('-', 74));
    Console.WriteLine("| {0,8} | {1,-15} | {2,-15} | {3,-12} | {4,-8} |",
        "Reg.Nr.", "Vardas", "Veislė", "Gimimo data", "Lytis");
    Console.WriteLine(new string('-', 74));
    for (int i = 0; i < dogs.Count; i++)
    {
        Dog dog = dogs.Get(i);
        Console.WriteLine("| {0,8} | {1,-15} | {2,-15} | {3,-12:yyyy-MM-dd} | {4,-
8} |",
            dog.ID, dog.Name, dog.Breed, dog.BirthDate, dog.Gender);
    }
    Console.WriteLine(new string('-', 74));
}
```

- Atitinkamai pertvarkykite kitas klases ir metodus.
- Atkreipkite dėmesį, jog skaitymui iš konteinerinės klasės objekto naudojamas metodas **Get**.
- Atkreipkite dėmesį, jog (kol kas) negalime naudoti **foreach** ciklo, pakeičiame jį **for** ciklu.
- Įvykdykite programą ir pasitikrinkite gautus rezultatus.

Trečias žingsnis. Dinamiškas konteinerio dydžio keitimas.

- Dabar konteinerio dydis fiksuotas — jame visada tilps tik 16 šunų. Papildykime klasės ***DogsContainer*** konstruktorių parametru ***capacity***, kuris leis konteinerio iniciavimo metu nurodyti jo talpą.

```
public DogsContainer(int capacity = 16) //parameter with default value
{
    this.dogs = new Dog[capacity];
}
```

- Atkreipkite dėmesį, jog šį konstruktorių galima naudoti dvejopai. Kviečiant konstruktorių su parametru pvz. `new DogsContainer(100)`, bus inicijuojamas nurodyto dydžio konteineris. Kviečiant konstruktorių be parametro, bus naudojama numatytoji reikšmė.

- Papildykite klasę ***DogsContainer*** kintamuoju ***Capacity***, kuris saugos dabartinę konteinerio talpą.

```
private int Capacity;

public DogsContainer(int capacity = 16)
{
    this.Capacity = capacity;
    this.dogs = new Dog[capacity];
}
```

- Papildykite klasę ***DogsContainer*** metodu ***EnsureCapacity***. Šis metodas patikrins, ar dabartinė konteinerio talpa pakankama, ir jei reikia — padidins konteinerio talpą.

```
private void EnsureCapacity(int minimumCapacity)
{
    if (minimumCapacity > this.Capacity)
    {
        Dog[] temp = new Dog[minimumCapacity];
        for (int i = 0; i < this.Count; i++)
        {
            temp[i] = this.dogs[i];
        }
        this.Capacity = minimumCapacity;
        this.dogs = temp;
    }
}
```

- Patobulinkite klasės ***DogsContainer*** metodą ***Add***. Jei bandant pridėti naują elementą nebėra vietos — padidinkite konteinerio talpą.

```
public void Add(Dog dog)
{
    if (this.Count == this.Capacity) //container is full
    {
        EnsureCapacity(this.Capacity * 2);
    }
    this.dogs[this.Count++] = dog;
}
```

1 savarankiško darbo užduotis (0,4 taško).

- Papildykite klasę metodu ***Put***, kuris skirtas padėti elementą į nurodytą vietą konteineryje.
- Papildykite klasę metodu ***Insert***, kuris skirtas įterpti elementą į nurodytą vietą konteineryje.
- Papildykite klasę metodais ***Remove*** ir ***RemoveAt***, kurie skirti elementų šalinimui iš konteinerio (pagal objekto vieną iš savybių (parametras yra objektas) ir pagal indeksą). Visuomet šalinamas tik vienas objektas.

Ketvirtas žingsnis. Rikiavimas.

- Papildykite klasę ***Dog*** metodu ***CompareTo***, kuris lygins šunis pagal veislę.

```
public int CompareTo(Dog other)
{
    return this.Breed.CompareTo(other.Breed);
}
```

- Papildykite klasę ***DogsContainer*** metodu ***Sort***, skirtu šunims rikiuoti. Naudokite burbuliuko algoritmą.

```
public void Sort()
{
    bool flag = true;
    while (flag)
    {
        flag = false;
        for (int i = 0; i < this.Count - 1; i++)
        {
            Dog a = this.dogs[i];
            Dog b = this.dogs[i + 1];
            if (a.CompareTo(b) > 0)
            {
                this.dogs[i] = b;
                this.dogs[i + 1] = a;
                flag = true;
            }
        }
    }
}
```

- Panaudokite jį ***Main*** metode.

```
...
DogsContainer allDogs = InOutUtils.ReadDogs(@"Dogs.csv");
allDogs.Sort();
...
```

- Įvykdykite programą ir pasitikrinkite gautus rezultatus.

2 savarankiško darbo užduotis (0,1 taško).

- Rikiuokite šunis pagal lytį ir veislę, pakeisdami esamą rikiavimo metodą.

Penktas žingsnis. Kopijos konstruktorius.

- Kartais prireikia konteinerio kopijos. Pavyzdžiui, jei norime turėti tuos pačius duomenis surikiuotus ir nesurikiuotus. Tokias atvejais patogu klasėje turėti kopijos konstruktorių.
- Papildykite klasę ***DogsContainer*** kopijos konstruktoriumi.

```
public DogsContainer(DogsContainer container): this() //calls another constructor
{
    for (int i = 0; i < container.Count; i++)
    {
        this.Add(container.Get(i));
    }
}
```

Šeštas žingsnis. Sudėtingesni konteinerio metodai.

- Ankstesnėse pratybose turėjote papildyti **Main** metodą šunų, kuriuos reikia skiepyti, spausdinimu.

```
DogsContainer requiresVaccination = register.FilterByVaccination();
InOutUtils.PrintDogs("Reikia skiepyti (Visos veislės)", requiresVaccination);
```

- **Main** metode jau turite konteinerį su šunimis, kuriuos reikia skiepyti, bei konteinerį su pasirinktos veislės šunimis. Reikia rasti, kuriuos pasirinktos veislės šunis reikia skiepyti. Vadinasi, reikia rasti elementus, kuries pasikartoja abejuose konteineriuose.
- Papildykite klasę ***DogsContainer*** metodu ***Intersect***, kuris kaip parametrą priims kitą konteinerį, ir gražins naują konteinerį, sudarytą iš abejuose konteineriuose pasikartojančių elementų.

```
public DogsContainer Intersect(DogsContainer other)
{
    DogsContainer result = new DogsContainer();
    for (int i = 0; i < this.Count; i++)
    {
        Dog current = this.dogs[i];
        if(other.Contains(current))
        {
            result.Add(current);
        }
    }
    return result;
}
```

- Panaudokite sukurtą metodą uždaviniu išspręsti.

```
InOutUtils.PrintDogs("Reikia skiepyti (" + selectedBreed + ")",
        requiresVaccination.Intersect(filteredByBreed));
```

- Įvykdykite programą ir pasitikrinkite rezultatus.

4 savarankiško darbo užduotis (0,5 taško).

Pirmoje failo eilutėje nurodytas fakulteto pavadinimas. Tekstiniame faile yra fakulteto studentų žiemos sesijos dalykų įvertinimai. Eilutėje apie studentą yra tokie duomenys: pavardė, vardas, grupė, pažymių kiekis, pažymiai. Nustatykite kiekvienos grupės bendrą mokymosi vidurkį. Atspausdinkite pradinius duomenis. Rezultatus atspausdinkite surikiuotus mažėjančiai pagal vidurkį ir pagal grupes abėcėlės tvarka. Užduotį realizuokite naudodami techniką, išmoktą šiose pratybose.