Code Academy

Dėstytojas

Vilmantas Neviera

Interfaces ir IComparer

Data

Šiandien išmoksite

01 Kas yra interface’ai?

02 Kaip panaudoti IComparable interface’ą?

Interface, kas tai?

- Interface’as yra kontraktas, kurį klasė “pasirašo” įvykdyti norėdama jį implementuoti.

  Tarkim turim interface’ą ICanRun (interface’ų vadinimo konvencija yra su raide “I” priekyje)

  ICanRun deklaruoja metodą void Run()

  Kaip matome metodas neturi kūno(body), interface’ai nurodo metodus, kuriuos klasė paveldinti šį interface’ą **privalo** implementuoti**.**

```
public interface ICanRun
{
    void Run();
}
```

  - Susikūrę klasę Human ir paveldėję iš ICanRun gauname klaidą

    Klaida aiškiai pasakanti, kad Human klasė **privalo** implementuoti metodą Run()

    Paspaudę “Show potential fixes” implementuojame metodą. Kaip matome, pagal nutylėjimą, metodas sukuriamas, bet viduje throw’inamas erroras, čia yra apsauga nuo neimplementuotų iki galo metodų.

```
public class Human : ICanRun
{
}
```

interface Sandbox.ICanRun

CS0535: 'Human' does not implement interface member 'ICanRun.Run()'

Show potential fixes (Ctrl+.)

```
public class Human : ICanRun
{
    public void Run()
    {
        throw new NotImplementedException();
    }
}
```

Interface, kas tai?

- Įsivaizduokim, kad kuriam aplikaciją picų kepimui, kuriant switch/case būdu, aplikacijai didėjant labai greitai susidursime su “bloated” kodu. Tikrinant kiekvieną tipą atskirai, metodas Prepare taps šimtų eilučių ilgio.

```
public class Pizza
{
    public void Prepare(string type)
    {
        switch (type)
        {
            case PizzaType.StuffedCrust:
                // prepare stuffed crust ingredients in system
                break;

            case PizzaType.DeepDish:
                // prepare deep dish ingredients in system
                break;

                //.... etc.
        }
    }
}
```

Interface, kas tai?

- Vienas iš sprendimų tam būtų naudoti interface’us.

  Susikuriam interface pavadinimu IPizza.

  Kontraktas IPizza įpareigos visas picos klases implementuoti metodą Prepare()

```
public interface IPizza
{
    public void Prepare();
}
```

```
public class NewYorkPizza : IPizza
{
    public void Prepare()
    {
        Console.WriteLine("Preparing New York pizza");
    }
}
```

```
public class CheesePizza : IPizza
{
    public void Prepare()
    {
        Console.WriteLine("Preparing Cheese pizza");
    }
}
```

Interface, kas tai?

- Tokiu būdu, turėdami picų sąrašą mes lengvai galime jas visas “paruošti” ;)

```
class Program
{
    public static void Main(string[] args)
    {
        var pizzas = new List<IPizza> { new NewYorkPizza(), new CheesePizza() };

        pizzas.ForEach(pizza => pizza.Prepare());
    }
}
```

```
C:\Users\ITWORK\Documents\CodeAcademy\Sandbox\Sandbox\bin\Debug\net5.0\Sandbox.e
Preparing New York pizza
Preparing Cheese pizza
```

Picų pavyzdys iš: https://stackoverflow.com/questions/6802573/interfaces-whats-the-point

Interface, kas tai?

3 I would add that you can't expect to understand interfaces in a couple of minutes. I think it is not reasonable to understand interfaces if you don't have years of object oriented programming experience. You may add some links to books. I would suggest: Dependency Injection in .NET which is really how deep the rabit hole goes, not just a gentle introduction. – knut Jul 26, 2011 at 10:15

Užduotis nr. 1

- Sukurkite interface’ą pavadinimų IVehicle su metodais Drive ir Refuel (grąžinimo tipą galit pasirinkti patys ir tai kaip jį implementuosit, galite daryti ir su void)
- Tada sukurkite klasę Car, kuri implementuos šį interface’ą ir taip pat turės properties string Model ir int Fuel.
- Implementuokite reikalingus metodus, Drive() metodas turi patikrinti ar degalai nėra pasibaigę, jeigu ne, važiuoti galima. Refuel metodas turi patikrinti ar kiekis, kuriuo norima papildyti nėra minusinis(galite padaryti ir maksimalų degalų bako dydį, bet pagalvokite, kaip tą funkcionalumą pernaudosite)

Užduotis nr. 2

- Pamodifikuokite pirmos užduoties programą.
- Paverskite Car klasę **abstrakčia** ir pridėkite property string Model.
- Sukurkite klasę BmwCar, kuri paveldės iš klasės Car ir turės savo property bool IsXDrive.
- Sukurkite klasę AudiCar, kuri paveldės iš klasės Car ir turės savo property bool IsQuattro.

```
public static void Main(string[] args)
{
    var bmw = new BmwCar(true, "Bmw", 50);
    var audi = new AudiCar(false, "Audi", 100);

    bmw.Drive();
    bmw.Refuel(50);

    audi.Drive();
    audi.Refuel(100);

    Console.WriteLine(audi.Model);
    Console.WriteLine(bmw.Model);
}
```

```
C:\Users\ITWORK\Documents\CodeAcademy\Sar
Driving
Driving
Audi
Bmw
```

IComparer<>

“Exposes a method that compares two objects.”

IComparer<> Pavyzdys

Turint sąrašą stringų, metodas list.Sort() yra labai akivaizdus - surūšiuoja sąrašą abėcėlės tvarka.

Jeigu mes turime sąrašą labiau komplikuotų objektų pvz. kaip Car, rūšiavimas sąrašo tampa nebe toks akivaizdus. Pagal, kurį property mes norime, kad programa sąrašą rūšiuotų?

Tokiam tikslui IComparer<> interface’as ir yra.

IComparer<> Pavyzdys

Pamodifikuojame jau turimą automobilių programą.

Esamo kodo judinti nebereikės, tik pridėsime naujo.

Susikuriame klasę BmwCarComparer ir paveldime IComparer interface'ą ir kaip duomenų tipą nurodydami BmwCar.

```
public class BmwCarComparer : IComparer<BmwCar>
```

Iškarto turėsim implementuoti metodą Compare.

Implementavus gauname tuščią metodą su throw'inamu exceptionu.

Tarkim mes norime, kad mūsų sąrašas būtų rūšiuojamas pagal automobilio modelį abėcėlės tvarka.

```
public int Compare(BmwCar x, BmwCar y)
{
    throw new System.NotImplementedException();
}
```

Compare metodo grąžinimą parašome taip:

```
return string.Compare(x.Model, y.Model);
```

IComparer<> Pavyzdys

IComparer Compare() metodo principas yra grąžinti -1 0 arba 1 pagal gautą sąlygą, kurį naudojant interface'as išsiaiškina ar reikšmė yra mažesnė lygi ar didesnė.

```
public class BmwCarComparer : IComparer<BmwCar>
{
    public int Compare(BmwCar x, BmwCar y)
    {
        return string.Compare(x.Model, y.Model);
    }
}
```

IComparer<> Pavyzdys

Susikurkime sąrašą bmwCar objektų.

```
var bmwCars = new List<BmwCar> { new BmwCar(true, "model2", 50), new BmwCar(true, "model3", 20), new BmwCar(true, "model1", 60), };
```

Tada susikurkime naujai sukurtą comparer ir panaudokime jį rūšiuojant sąrašą.

Paleiskite programą ir įsitikinkite, kad sąrašas
yra surūšiuotas pagal mašinų modelius abėcėlės tvarka.

```
public static void Main(string[] args)
{
    var bmwCars = new List<BmwCar> { new BmwCar(true, "model2", 50), new Bmw
    var carComparer = new BmwCarComparer();

    bmwCars.Sort(carComparer);
}
```

Užduotis nr. 3

- Implementuokite IComparer interface’ą ir kitiems automobilių tipams.

Naudinga informacija

Interfaces ir IComparable

https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/interface

https://docs.microsoft.com/en-us/dotnet/api/system.collections.icomparer?view=net-6.0