

# C# .NET

Laborator 7

# Assembly

# Assembly

- O colectie de entitati
  - Clase, interfete, enumerari, etc
- Grupate sub forma
  - Executabil
    - Cod binar de sine statator care poate rula
    - Reprezinta “punctul de pornire al ulnei aplicatii
  - Dll – dynamically linked library

## Translations of assembly

noun

asamblare  
assembly, assembling, assemblage

montare  
mounting, assembling, assembly, erection, setting, assemblage

adunare  
assembly, meeting, gathering, congregation, addition, rally

întreunire  
meet, meeting, assembly, rally, congregation, bevy

reuniune  
reunion, gathering, assembly, party, social, sociable

ședință  
meeting, sitting, session, conference, assembly

consiliu  
council, assembly, concilium, senate, corporation

sobor  
synod, assembly, thousand, service, group of priests

# Assembly - executabil

- Aplicatii de sine statatoare, rulabile
- Code perspective : detin functia *Main*
- Tipuri de aplicatii in conxtul windows
  - Aplicatii desktop
    - Dezvoltate pe baza mai multor framework-uri
      - [Windows forms](#)
      - [WPF](#)
      - Unity
    - Programe, Jocuri, medii de dezvoltare, aplicatii de system
  - Aplicatii web/cloud
    - ASP MVC Core app / Azure webapp
    - Worker service
    - Azure function
    - Etc
  - [Windows services](#)
    - Aplicatii windows ce pornesc la pornirea calculatorului inainte de login
    - De sine statatoare
    - Gestionate de catre svchost.exe

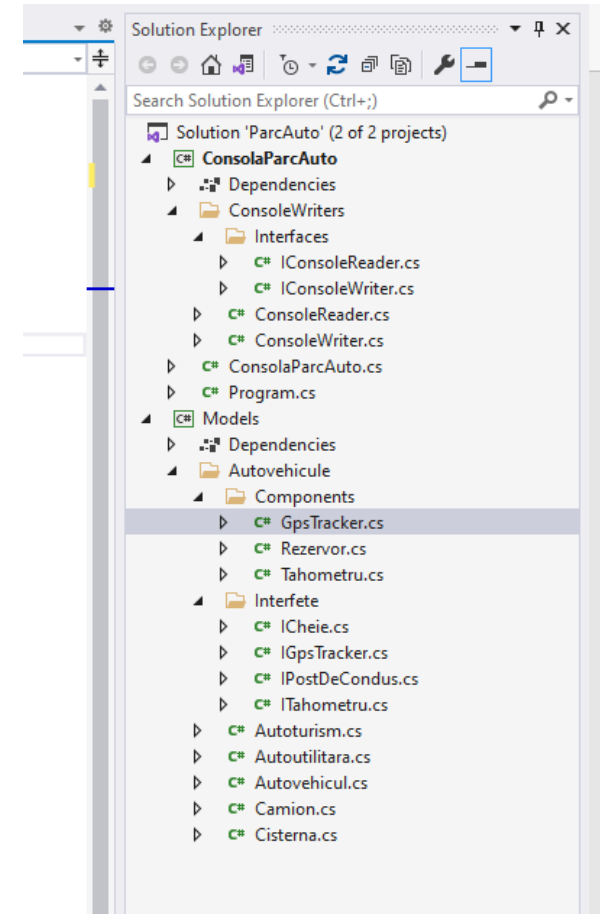
Exemple : visual studio templates

# Assembly – DLL

- Dinamic-Link Library
  - Microsoft shared libraries – biblioteci care pot fi partajate intre aplicatii
- Colectii de functii, clase, interfete
- In functie de modul de compilare –
  - codul este sau nu obfuscate
  - Pdb – existente sau nu
- Importate de catre aplicatii
- Versionate
- Sharing
  - Ca fisier – in cadrul proiectelor mai vechi
  - Ca pachet nuget – via internet/intranet

# Solution structure

- Solutia
  - Contine mai multe proiecte
    - Dll
    - Console Apps
    - Windows service
    - WebApp
    - Etc...
  - Contine setari care gestioneaza interactiunea dintre proiecte
    - Startup project
    - Configuratii pt build
    - Dependente, nu dependinte
      - Dependentele - chestii de care depinzi
      - Dependinte – ce sunt dependintele?

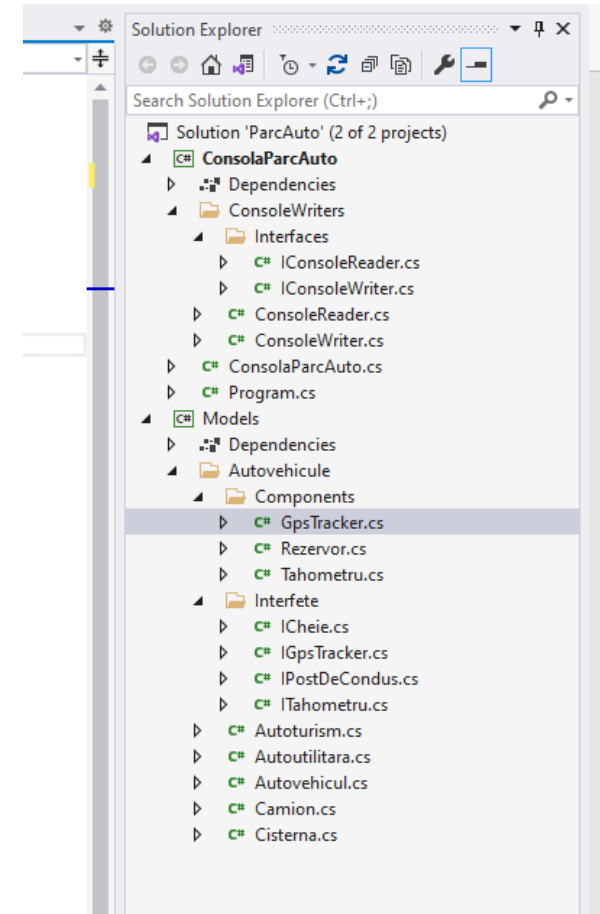


# Solution structure – depdeninte anexe pe langa casa



# Solution structure

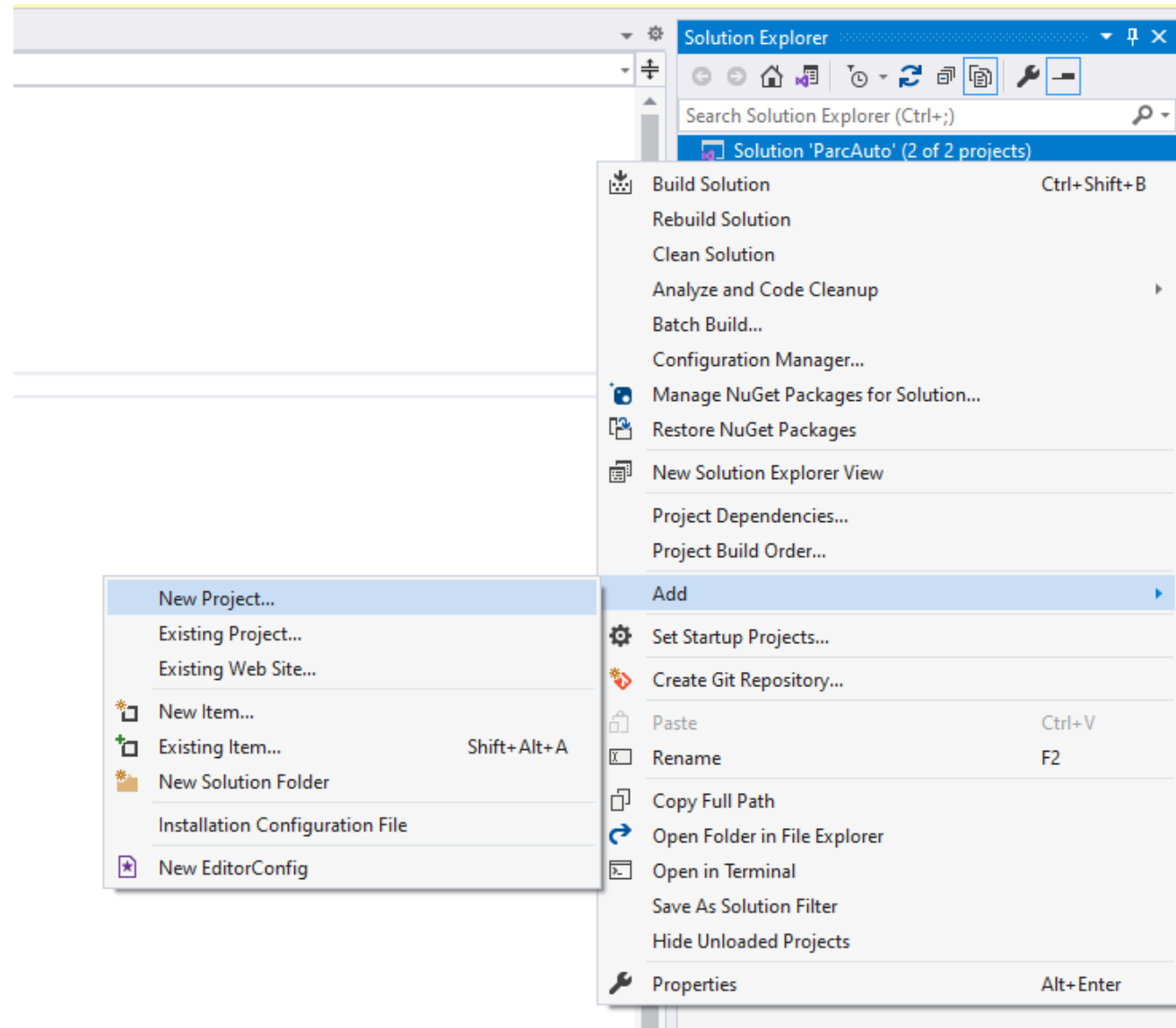
- Solutia
  - Contine mai multe proiecte
    - Dll
    - Console Apps
    - Windows service
    - WebApp
    - Etc...
  - Contine setari care gestioneaza interactiunea dintre proiecte
    - Startup project
    - Configuratii pt build
    - Dependente, nu dependinte
      - Dependentele - chestii de care depinzi
      - Dependinte – in ele cresti gaini



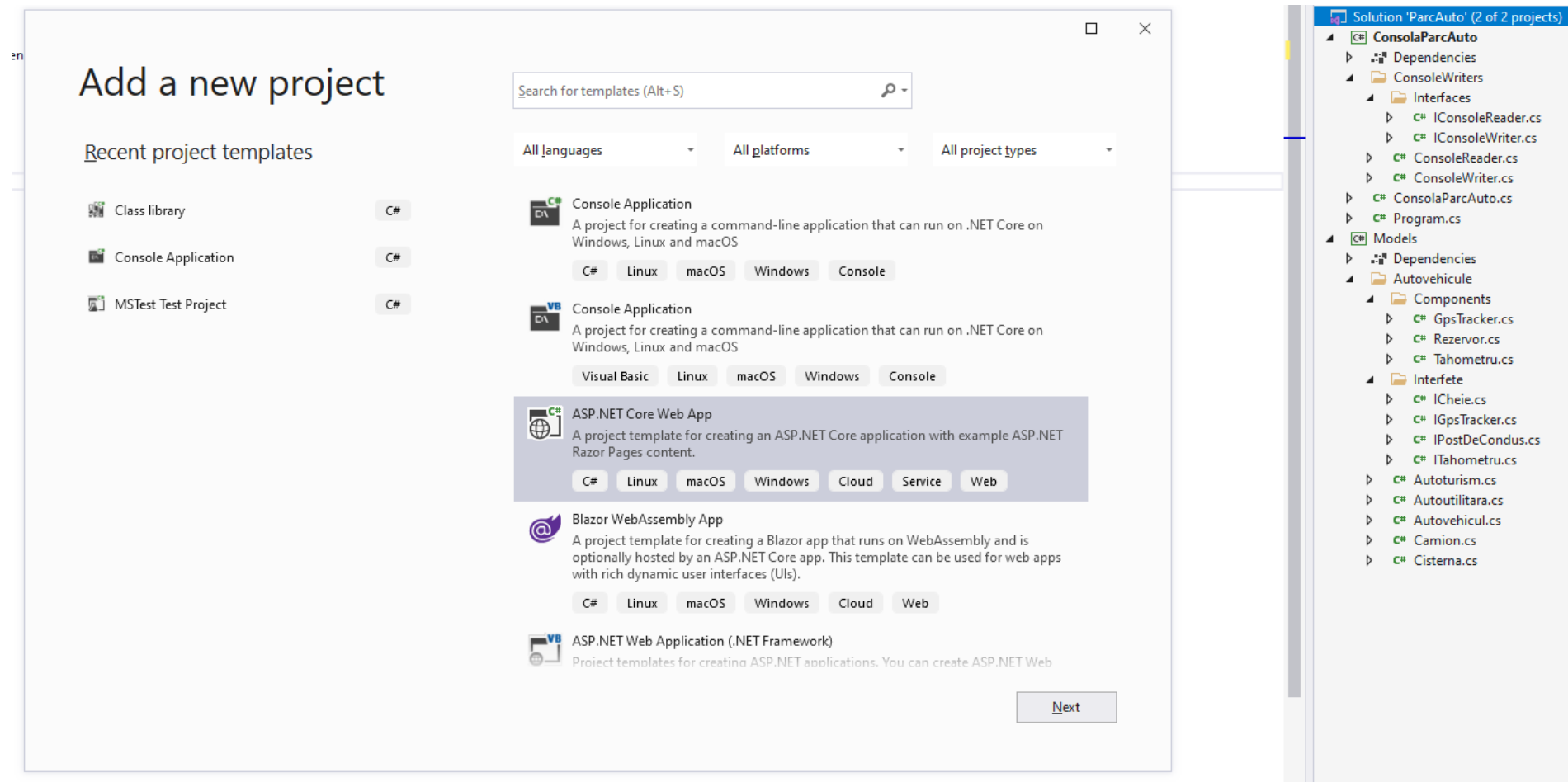


# Add project to solution

- Right click pe solutie
  - New project

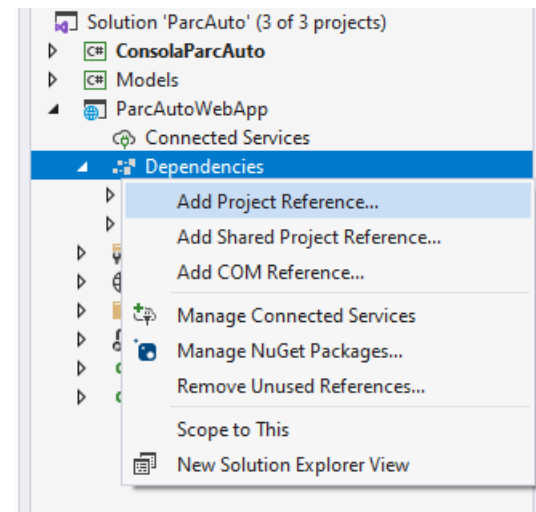
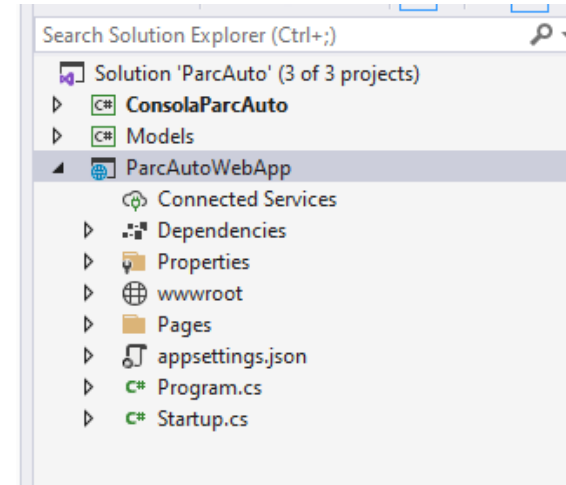


# Tipul proiectului



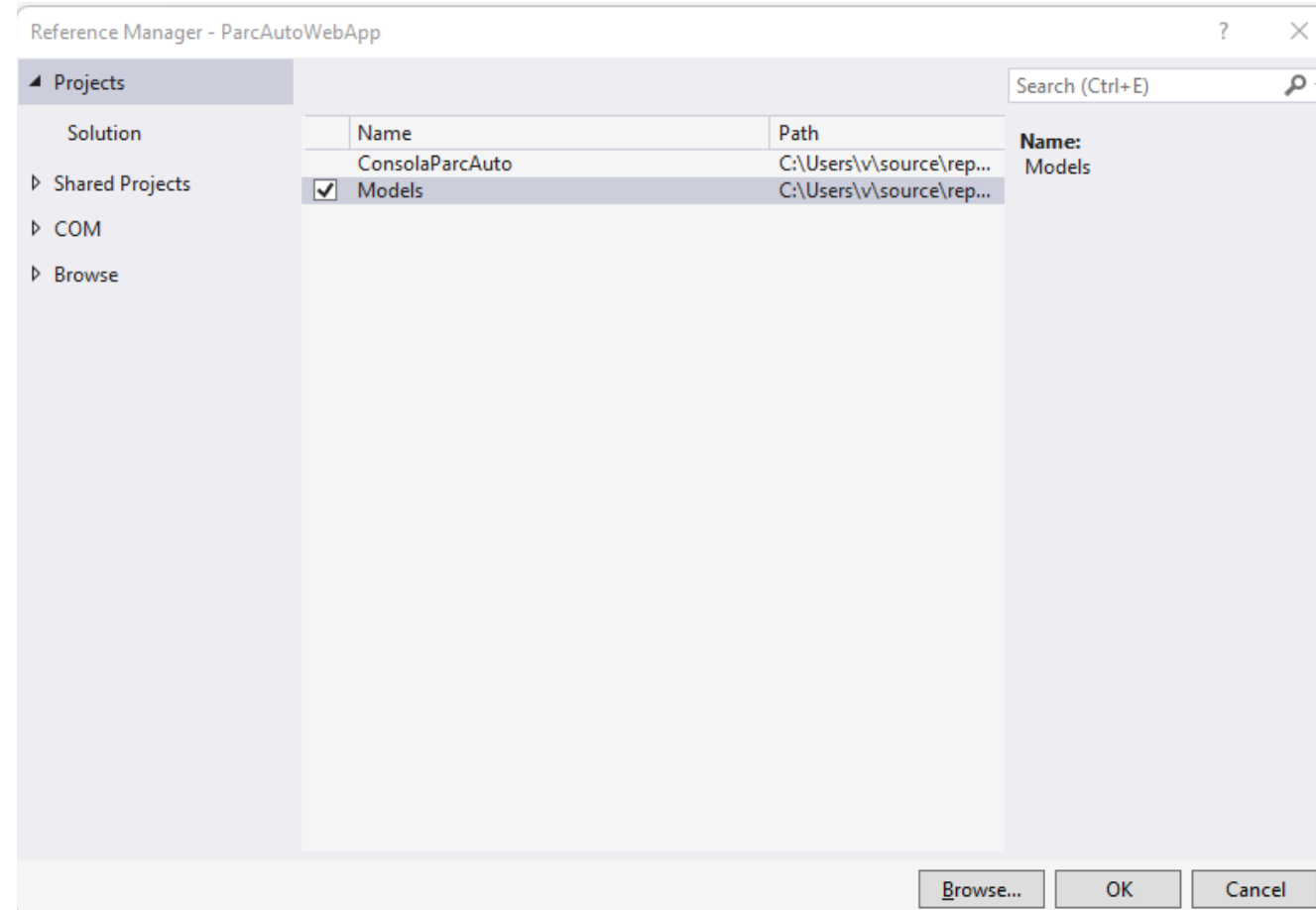
# Add project reference

- Referinta spre un proiect
  - Acces la toate entitatile **PUBLICE**
    - clase, interfete, enums, etc..
  - Partajare a claselor/entitatilor intre proiecte



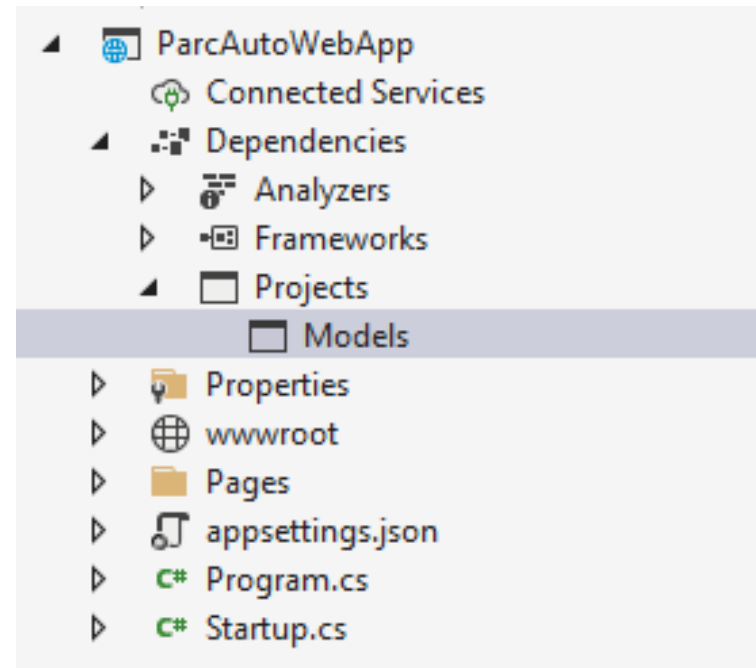
# Add reference

- Selectam proiectul spre care adaugam o referinta
- OK



# Add reference

- Proiectul este adaugat ca referinta
- ParcAutoWebapp
  - Va depinde de “Models”
  - Va avea acces la toate entitatile **PUBLIC** definite in “Models”
- Dublu click pe proiectul caruia ii facem referinta
  - Toate entitatile publice vor fi descrites
  - XML comments – modul de a interactiona cu 3<sup>rd</sup> party-uri



```

/// <summary>
/// Functionality that offers GPS Tracking of an entity
/// </summary>
1 reference
public interface IGpsTracker
{
    /// <summary>
    /// Gets the current location of the traced entity
    /// </summary>
    /// <returns>Tuple with two double values, latitude and longitude</returns>
    1 reference
    public Tuple<double,double> GetCurrentLocation();
}

```

```

/// <summary>
/// Defineste un autoturism
/// </summary>
0 references
public class Autoturism : Autovehicul, ICheie, IGpsTracker
{
    1 reference
    public Tuple<double, double> GetCurrentLocation()
    {
        throw new NotImplementedException();
    }

    1 reference
    public void Lock()
    {
        throw new NotImplementedException();
    }

    1 reference
    public void Unlock()
    {
        throw new NotImplementedException();
    }
}

```

Search

- ▷ Microsoft.Extensions.FileSystemGlobbing
- ▷ Microsoft.Extensions.Hosting
- ▷ Microsoft.Extensions.Hosting.Abstractions
- ▷ Microsoft.Extensions.Http
- ▷ Microsoft.Extensions.Identity.Core
- ▷ Microsoft.Extensions.Identity.Stores
- ▷ Microsoft.Extensions.Localization
- ▷ Microsoft.Extensions.Localization.Abstractions
- ▷ Microsoft.Extensions.Logging
- ▷ Microsoft.Extensions.Logging.Abstractions
- ▷ Microsoft.Extensions.Logging.Configuration
- ▷ Microsoft.Extensions.Logging.Console
- ▷ Microsoft.Extensions.Logging.Debug
- ▷ Microsoft.Extensions.Logging.EventLog
- ▷ Microsoft.Extensions.Logging.EventSource
- ▷ Microsoft.Extensions.Logging.TraceSource
- ▷ Microsoft.Extensions.ObjectPool
- ▷ Microsoft.Extensions.Options
- ▷ Microsoft.Extensions.Options.ConfigurationExtensions
- ▷ Microsoft.Extensions.Options.DataAnnotations
- ▷ Microsoft.Extensions.Primitives
- ▷ Microsoft.Extensions.WebEncoders
- ▷ Microsoft.JSInterop
- ▷ Microsoft.Net.Http.Headers
- ▷ Microsoft.VisualBasic
- ▷ Microsoft.VisualBasic.Core
- ▷ Microsoft.Win32.Primitives
- ▷ Microsoft.Win32.Registry
- ▾ Models
  - ▾ { } ParcAuto.Autovehicule
    - ▷ Autoturism
    - ▷ Autoutilitara
    - ▷ Autovehicul
    - ▷ Camion
    - ▷ Cisterna
  - ▷ { } ParcAuto.Autovehicule.Components
  - ▷ { } ParcAuto.Autovehicule.Interfete
  - ▷ { } ParcAuto.Models.Autovehicule.Components
- ▷ mscorlib
- ▷ netstandard
- ▷ ParcAutoWebApp

- GetCurrentLocation()
- Lock()
- Unlock()

public class **Autoturism** : [ParcAuto.Autovehicule.Autovehicul](#)  
Member of [ParcAuto.Autovehicule](#)

**Summary:**  
Defineste un autoturism

```

/// <summary>
/// Functionality that offers GPS Tracking of an entity
/// </summary>
1 reference
public interface IGpsTracker
{
    /// <summary>
    /// Gets the current location of the traced entity
    /// </summary>
    /// <returns>Tuple with two double values, latitude and longitude</returns>
    1 reference
    public Tuple<double,double> GetCurrentLocation();
}

```

```

/// <summary>
/// Defineste un autoturism
/// </summary>
0 references
public class Autoturism : Autovehicul, ICheie, IGpsTracker
{
    1 reference
    public Tuple<double, double> GetCurrentLocation()
    {
        throw new NotImplementedException();
    }

    1 reference
    public void Lock()
    {
        throw new NotImplementedException();
    }

    1 reference
    public void Unlock()
    {
        throw new NotImplementedException();
    }
}

```



- Microsoft.Extensions.FileSystemGlobbing
- Microsoft.Extensions.Hosting
- Microsoft.Extensions.Hosting.Abstractions
- Microsoft.Extensions.Http
- Microsoft.Extensions.Identity.Core
- Microsoft.Extensions.Identity.Stores
- Microsoft.Extensions.Localization
- Microsoft.Extensions.Localization.Abstractions
- Microsoft.Extensions.Logging
- Microsoft.Extensions.Logging.Abstractions
- Microsoft.Extensions.Logging.Configuration
- Microsoft.Extensions.Logging.Console
- Microsoft.Extensions.Logging.Debug
- Microsoft.Extensions.Logging.EventLog
- Microsoft.Extensions.Logging.EventSource
- Microsoft.Extensions.Logging.TraceSource
- Microsoft.Extensions.ObjectPool
- Microsoft.Extensions.Options
- Microsoft.Extensions.Options.ConfigurationExtensions
- Microsoft.Extensions.Options.DataAnnotations
- Microsoft.Extensions.Primitives
- Microsoft.Extensions.WebEncoders
- Microsoft.JSInterop
- Microsoft.Net.Http.Headers
- Microsoft.VisualBasic
- Microsoft.VisualBasic.Core
- Microsoft.Win32.Primitives
- Microsoft.Win32.Registry

#### Models

- └─ { } ParcAuto.Autovehicule
  - └─ Autoturism
    - └─ Base Types
      - └─ Autovehicul
        - └─ Object
          - └─ ICheie
          - └─ IGpsTracker
  - └─ Autoutilitara
  - └─ Autovehicul
  - └─ Camion
  - └─ Cisterna
  - └─ { } ParcAuto.Autovehicule.Components
  - └─ { } ParcAuto.Autovehicule.Interfete
  - └─ { } ParcAuto.Models.Autovehicule.Components
- mscorlib

- GetCurrentLocation()
- Lock()
- Unlock()

public [System.Tuple<double, double>](#) GetCurrentLocation()  
 Member of [ParcAuto.Autovehicule.Autoturism](#)

#### Summary:

Gets the current location of the traced entity

#### Returns:

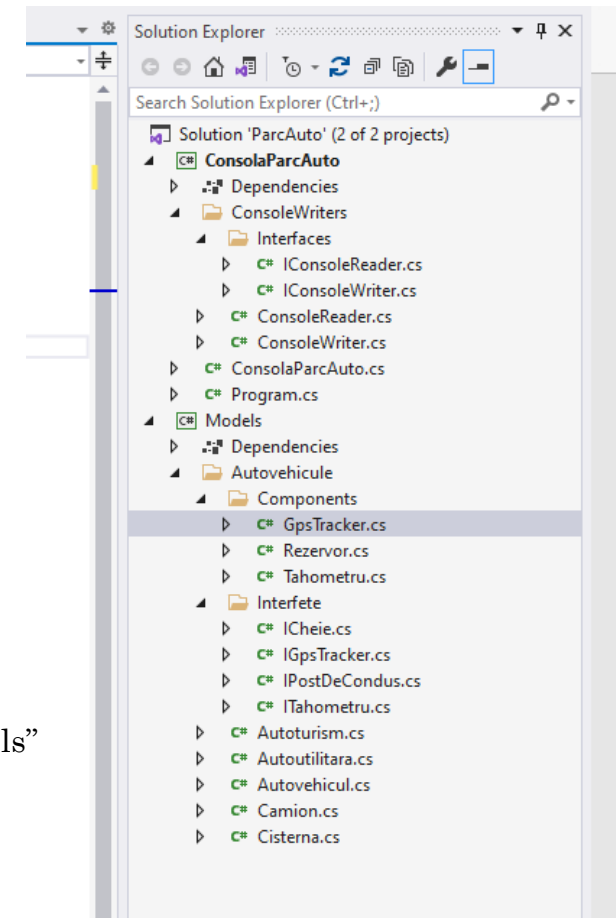
Tuple with two double values, latitude and longitude

# Assemblies - internal

- Internal access modifier
  - Clase, interfete, entitati marcate ca internal
    - accesibile doar in interiorul assembly-ului in care au fost definite
  - Valoarea default ca access modifier
- Ajuta la ascunderea implementarii
  - Vom expune altor proiecte care vor folosi librariile noastre doar ceea ce consideram necesar
  - Asemănător “private” la nivelul claselor

Cisterna – accesibil doar in “Models”

```
namespace ParcAuto.Autovehicule
{
    /// <summary>
    /// Clasa care modeleaza o cisterna
    /// </summary>
    0 references
    internal class Cisterna : IGpsTracker
    {
        1 reference
        public Tuple<double, double> GetCurrentLocation()
        {
            throw new NotImplementedException();
        }
    }
}
```



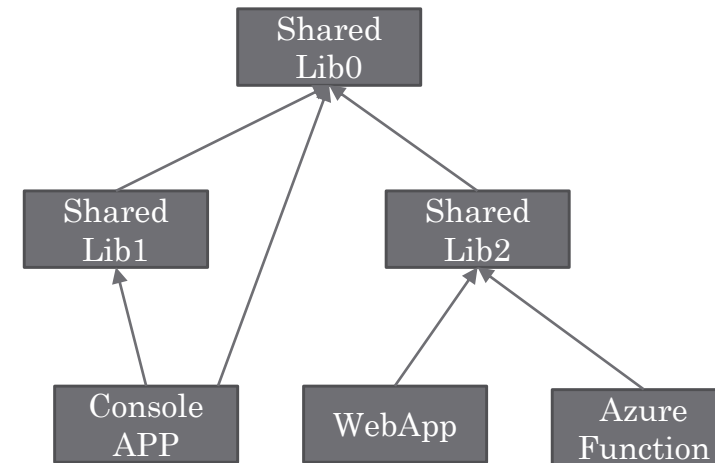
# internal

- Pentru a putea fi utilizata intr-un alt assembly
  - Clasa/entitatea trebuie facuta *public*
- Clasa *public*
  - Toate dependentele sale trebuie facute publice
    - Clase pe care le mosteneste
    - Interfete pe care le implementeaza
    - Tipuri de date/obiecte pe care le foloseste
    - Tipuri de date/obiecte pe care metodele sale le returneaza

# Good practices

- Proiectele executabile nu ar trebui sa depinda intre ele
  - Console apps, WebApps, Azure Functions, Windows services
- Proiectele ar trebui sa depinda de librarii comune
  - Graph-ul dependentelor ar trebui sa nu fie foarte complex
- Referinte ciclice- nu sunt suportate

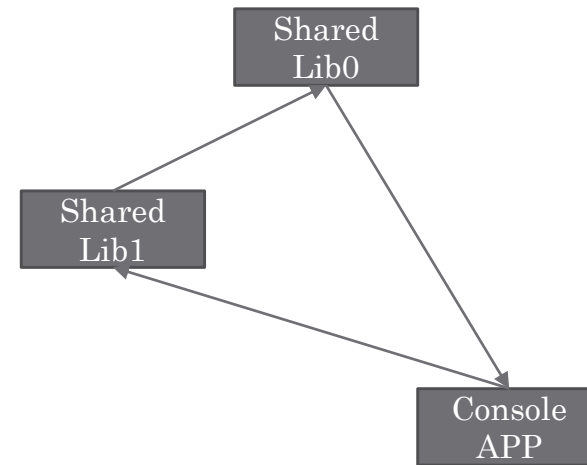
Graph al dependentelor



# Good practices

- Proiectele executabile nu ar trebui sa depinda intre ele
  - Console apps, WebApps, Azure Functions, Windows services
- Proiectele ar trebui sa depinda de librarii comune
  - Graph-ul dependentelor ar trebui sa nu fie foarte complex
- Referinte ciclice- nu sunt suportate

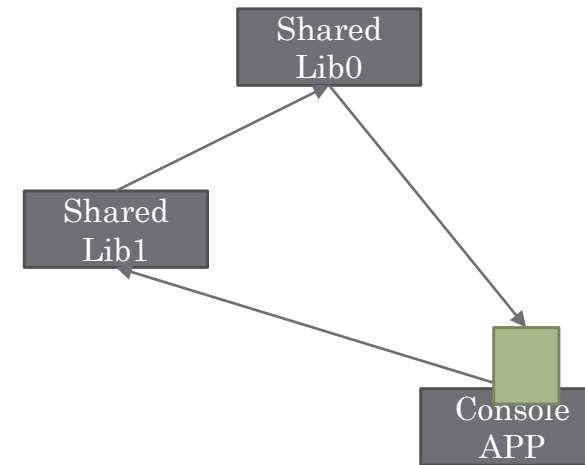
## Referinte circulare



# Good practices

- Proiectele executabile nu ar trebui sa depinda intre ele
  - Console apps, WebApps, Azure Functions, Windows services
- Proiectele ar trebui sa depinda de librarii comune
  - Graph-ul dependentelor ar trebui sa nu fie foarte complex
- Referinte ciclice- nu sunt suportate
  - Solutia: extragem codul care cauzeaza circularitatea intr-un proiect shared

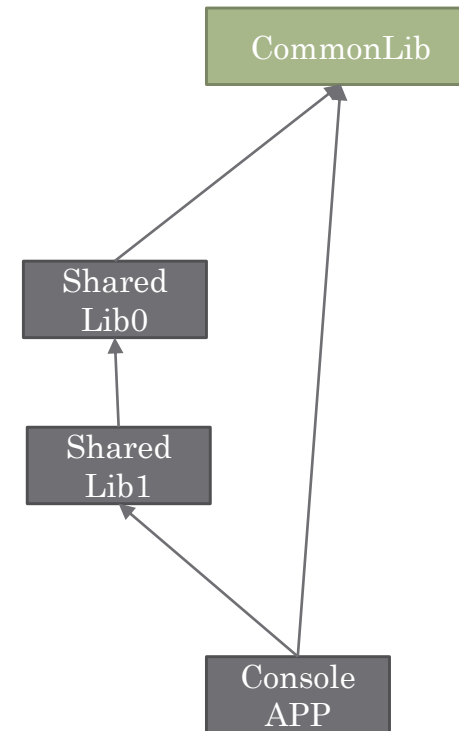
## Referinte circulare



# Good practices

- Proiectele executabile nu ar trebui sa depinda intre ele
  - Console apps, WebApps, Azure Functions, Windows services
- Proiectele ar trebui sa depinda de librarii comune
  - Graph-ul dependentelor ar trebui sa nu fie foarte complex
- Referinte ciclice- nu sunt suportate
  - Solutia: extragem codul care cauzeaza circularitatea intr-un proiect shared

## Referinte circulare



# Assembly – DLL

- Se comporta precum “project reference”
  - De fapt project reference se comporta precum DLL ☺
  - Aceiasi modificatory de acces – internal, public
- **Dinamic-Link Library**
  - Microsoft shared libraries – biblioteci care pot fi partajate intre aplicatii
- Colectii de functii, clase, interfete
- In functie de modul de compilare –
  - codul este sau nu obfuscate
  - Pdb – existente sau nu
- Importate de catre aplicatii
- **DLL-urile**
  - Versionate
  - Un proiect poate cere strict o anumita versiune a unui DLL
  - Daca Libraria evolueaza, proiectul poate in continuare functiona folosind versiunea specificata
  - Trecerea la o versiune mai noua – la latitudinea proiectului-consumer al dll-ului
- **Sharing**
  - Ca fisier – in cadrul proiectelor mai vechi
  - Ca pachet NuGet – via internet/intranet



Comments

# Comments

- Codul comentat nu va fi executat
- Putem scrie text non-instructiune, explicatii pt algoritmi complecsi

```
// Console.WriteLine("hello world"); this is a line coment. the code does nothing.  
// another line
```

```
/* here starts a multiline comment
```

```
this code does nothing
```

```
Console.WriteLine($"varsta studentului este {student.varsta} de ani");
```

```
here the multiline comment ends */
```


```
// we commented only a part of the line
```

```
Student student = new Student("a"/*, "b", 22*/);
```

# XML Comments

- Informatii pentru consumatori claselor **publice**
  - Comentati cu preponderenta entitatile **publice** ale claselor **publice**
- Scrieti “///” inaintea entitatii pentru care doriti adaugarea xml comment-ului

```
Student student = new Student(20);  
}  
}
```

 class Lab6.Student  
Aceasta clasa modeleaza un student

```
/// <summary>  
/// Aceasta clasa modeleaza un student  
/// </summary>  
4 references  
class Student  
{
```

# XML Comments

```
Student student = new Student(20);
student.
}
}

/// <summary>
/// Aceasta clasa reprezinta un student
/// </summary>
4 references
class Student
{
    /// <summary>
    /// Calculeaza media generala tinand cont de nota din teza
    /// </summary>
    /// <param name="notaTezei"></param>
    /// <param name="ponderaNoteiTezei"></param>
    /// <returns>Media generala</returns>
    0 references
    public double CalculeazaMediaCuTeza(int notaTezei, double ponderaNoteiTezei)
    {
        double media = 0.0;
        //.....
        return media;
    }
}
```

CalculeazaMediaCuTeza double Student.CalculeazaMediaCuTeza(int notaTezei, double ponderaNoteiTezei)  
Calculeaza media generala tinand cont de nota din teza

Equals  
GetHashCode  
GetType  
note  
nume  
prenume  
ToString  
varsta

# Generics

Methods and classes

# Generics

- [Generics](#) - documentatie
- Functionalitate a limbajului C# care ne permite scrierea unui cod care va opera asupra unor tipuri de date specificate ulterior

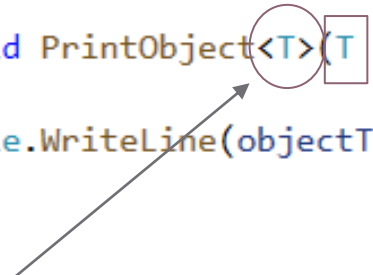
Sau

- Tipuri de date ca parametrii

# Generic methods

- Un tip de date ca si argument

```
static void PrintObject<T>(T objectToPrint)
{
    Console.WriteLine(objectToPrint.ToString());
}
```




Declararea tipului de date

- In semnatura metodei si interiorul ei T va reprezenta tipul de date dat ca parametru


# Generic methods

- Putem declara variabile de tipul generic



```
static void PrintObject<T>(T objectToPrint)
{
    Console.WriteLine(objectToPrint.ToString());
    T secondObject = objectToPrint;
    Console.WriteLine(secondObject.ToString());
}
```

- Putem declara tipul generic ca return type



```
static T PrintObject<T>(T objectToPrint)
{
    Console.WriteLine(objectToPrint.ToString());
    T secondObject = objectToPrint;
    Console.WriteLine(secondObject.ToString());
    return objectToPrint;
}
```



# Generic methods


- T se comporta ca si orice alt tip iar type checking-ul functioneaza

```
1 reference
static T PrintObject<T>(T objectToPrint)
{
    Console.WriteLine(objectToPrint.ToString());

    T secondObject = 3;

    Console.WriteLine(secondObject);

    return objectToPrint;
}
```

 **readonly struct System.Int32**  
Represents a 32-bit signed integer.

CS0029: Cannot implicitly convert type 'int' to 'T'


- Pot fi declarate mai multe tipuri generice
  - Tipurile vor fi considerate diferite
    - Variabile incompatibile

```
1 reference
static T PrintObject<T, G>(T objectToPrint, G secondType)
{
    Console.WriteLine(objectToPrint.ToString());

    T secondObject = secondType;

    Console.WriteLine(secondObject);

    return objectToPrint;
}
```

 (parameter) **G secondType**

CS0029: Cannot implicitly convert type 'G' to 'T'

# Exercitiu

- Scrieti o metoda care va interschimba doua valori generice

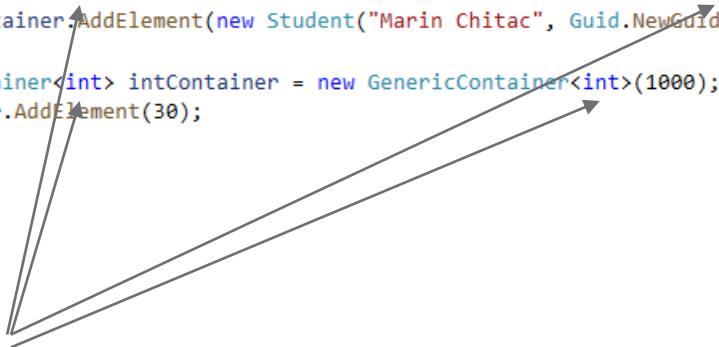
# Generic Classes

- Tipul parametru – la nivelul clasei
- In interiorul tuturor membrilor clasei, T va reprezenta acelasi tip
- Instantierea:

```
GenericContainer<Student> studentsContainer = new GenericContainer<Student>(10);
studentsContainer.AddElement(new Student("Marin Chitac", Guid.NewGuid()));

GenericContainer<int> intContainer = new GenericContainer<int>(1000);
intContainer.AddElement(30);
```

Tipul ca parametru



```
class GenericContainer<T>
{
    private T[] elements;
    private int currentCount = 0;
    private readonly int capacity;
    0 references
    public GenericContainer(int capacity)
    {
        this.capacity = capacity;
        elements = new T[capacity];
        currentCount = 0;
    }

    0 references
    public void AddElement(T elementToAdd)
    {
        if (currentCount == capacity)
            return;

        this.elements[currentCount] = elementToAdd;
        currentCount++;
    }

    0 references
    public T GetElementAt(int index)
    {
        if (index < currentCount && index > 0)
        {
            return elements[index];
        }
        else
        {
            throw new Exception();
        }
    }
}
```

# Generic classes - inheritance

- La mostenirea claselor generice
  - Subclasa trebuie sa rezolve tipul generic
  - O poate face prin declararea acestuia ca tip generic

```
class GenericSwapper<T>
{
    2 references
    public virtual void Swap(ref T object1, ref T object2)
    {
        T aux = object1;
        object1 = object2;
        object2 = aux;
    }
}

1 reference
class GenericLogSwapper<T> : GenericSwapper<T>
{
    2 references
    public override void Swap(ref T object1, ref T object2)
    {
        Console.WriteLine("Swapping");
        base.Swap(ref object1, ref object2);
        Console.WriteLine("Swapped");
    }
}
```

# Generic classes - inheritance

- Subclasa trebuie sa resolve tipul generic
  - Prin declararea acestuia
  - Toate obiectele de tipul *subclasei* vor fi nevoite sa lucreze
    - cu tipul declarant
    - cu subtipuri ale tipului declarat

```
class GenericSwapper<T>
{
    2 references
    public virtual void Swap(ref T object1, ref T object2)
    {
        T aux = object1;
        object1 = object2;
        object2 = aux;
    }
}
2 references
class IntSwapper : GenericLogSwapper<int>
{
}
```

```
IntSwapper intSwapper = new IntSwapper();
```

```
intSwapper.
```



The image shows an IntelliSense dropdown menu for the `intSwapper` variable. The menu lists several methods: `Equals`, `GetHashCode`, `GetType`, `Swap`, and `ToString`. The `Swap` method is highlighted in blue. To the right of the `Swap` method, the signature `void GenericLogSwapper<int>.Swap(ref int object1, ref int object2)` is displayed.

# Generics – constraints

- Codul generic este limitat
  - Operatiile obiectelor/tipurilor generice sunt foarte limitate
  - Obiectele generice – value types sau reference types?
  - Nu avem informatii despre tipuri, metode disponibile, constructori
- Restrictii
  - Ne permit sa avem mai multe informatii despre tipurile generice

# Generics – constraints

- Restrictiile – folosind keyword-ul “where: dupa declararea clasei/structurii sau dupa semnatura metodei
- Tipuri de restrictii
  - Tipul generic trebuie sa fie un value-type
  - Tipul generic trebuie sa fie un reference type
  - Tipul generic trebuie sa aibe un constructor public fara parametrii
  - Tipul generic trebuie sa derive dintr-o anumita clasa de baza
  - Tipul generic trebuie sa implementeze o anumita interfata
  - Tipul generic trebuie sa derive dintr-alt tip generic
  - Tipul generic trebuie sa fie un delegat
  - Tipul generic trebuie sa fie o enumeratie

# Generics – constraints

T – instanta a unei clase

```
public static class VarHelper
{
    2 references
    public static void Swap<T>(ref T left, ref T right)
        where T: class
    {
        T aux = left;
        left = right;
        right = aux;
    }
}
```

T - instanta a unei clase

Trebuie sa aiba un constructor no-arg

Poate fi instantiate in interiorul clasei/metodei generice

```
public class Factory<T>
    where T: class, new()
{
    0 references
    public T Create()
    {
        return new T();
    }
}
```



# Generics – constraints

Putem specifica un tip

- Tipul : clasa de baza
- Clasa generica va accepta orice obiect de tipul sau mostenit din T

```
public static class ExceptionLogger
{
    2 references
    public static void Log<T>(T exception)
        where T: Exception
    {
        Console.WriteLine(exception.Message);
        Console.WriteLine(exception.StackTrace);
        if (exception.InnerException != null)
        {
            ExceptionLogger.Log(exception.InnerException);
        }
    }
}
```

Putem specifica o interfata

- -Clasa generica va accepta obiecte ce implementeaza interfata respective
- Vom avea acces la toti membrii interfetei

```
public static class EqualityChecker
{
    3 references
    public static bool Equals<T>(T first, T second)
        where T: IComparable<T>
    {
        if ((first is null) || (second is null))
        {
            return false;
        }

        return ((IEquatable<T>)first).Equals(second);
    }
}
```

# Generics – constraints

- Constaint-uri pentru multiple tipuri

```
public static class ExceptionLogger
{
    2 references
    public static void Log<TLogLevel, TException>(TLogLevel level, TException exception)
        where TLogLevel: Enum
        where TException : Exception
    {
        string labelName = Enum.GetName(typeof(TLogLevel), level);
        Console.WriteLine($"[{labelName}]: {exception.Message}");
        Console.WriteLine(exception.StackTrace);
        if (exception.InnerException != null)
        {
            ExceptionLogger.Log(level, exception.InnerException);
        }
    }
}
```

# Generics –default

- Exista situatii in care, in codul clasei/structurii/metodei noastre generice avem nevoie sa declaram si sa instantiem obiecte de tipul generic (T) cu o valoare implicita. In aceste cazuri putem folosi cuvantul cheie “[default](#)” care va returna urmatoarele:
  - Daca la utilizarea efectiva, pentru tipul generic T se foloseste un tip valoare, atunci default(T) va returna 0, sau valoarea echivalenta a lui 0
  - Daca la utilizarea efectiva, pentru tipul generic T se foloseste un tip referinta, atunci default(T) va returna null;

```
public class ElementFinder<T>
{
    private readonly T[] array;

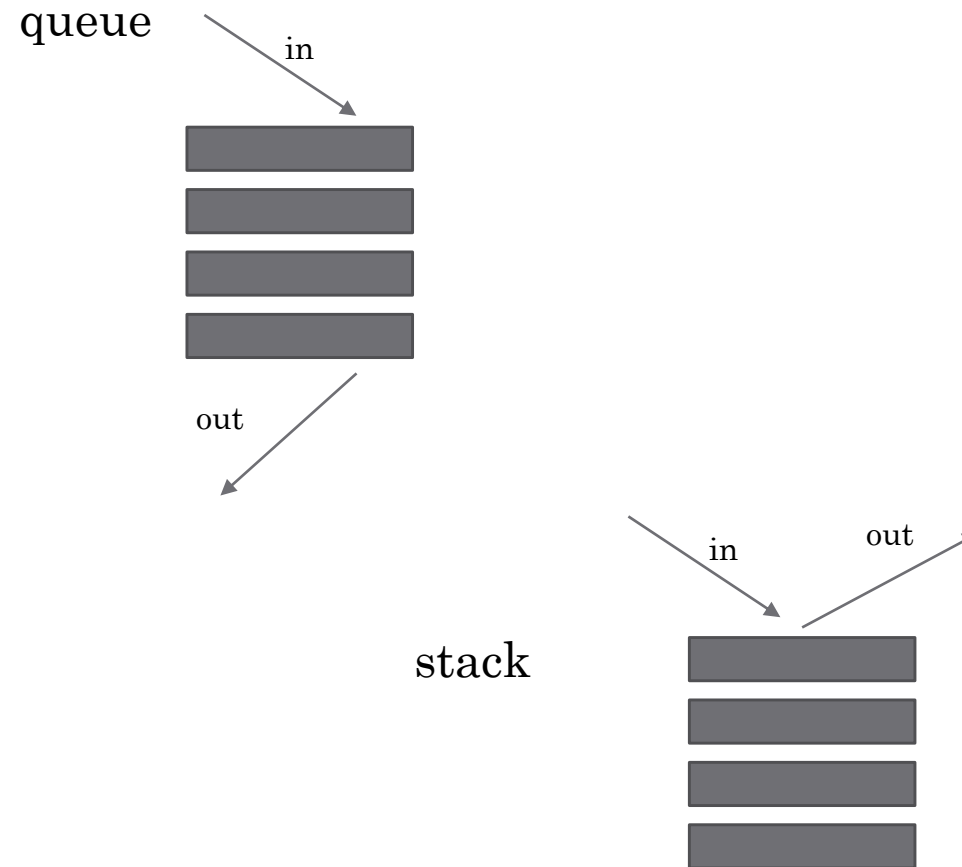
    0 references
    public ElementFinder(params T[] array)
    {
        this.array = array ?? new T[0];
    }

    0 references
    public bool TryGetElement(int index, out T element)
    {
        if ((index >= 0) && (index < this.array.Length))
        {
            element = this.array[index];
            return true;
        }

        element = default(T);
        return false;
    }
}
```

# Exemple

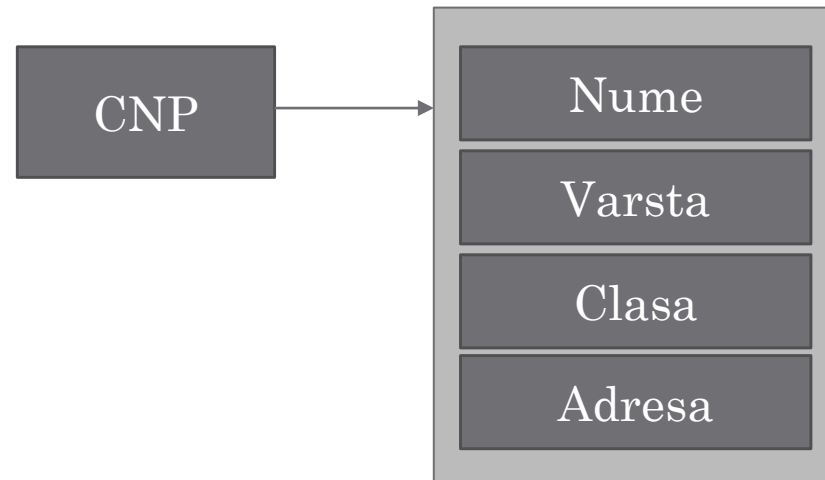
- System.Collections.Generic
  - List
  - Queue - coada
    - FIFO
      - First In First Out
  - Stack - stiva
    - LIFO
      - Last In Frist Out



# Exemple - Dictionary

- Dictionary

- o lista generica de perechi cheie-valoare
- Cheile sunt **unice**
- Fiecarei chei ii corespunde o valoare
- Cheile si valorile
  - tipuri generice
  - Key, value – tipuri diferite
- Analogie : DEX
  - Fiecarui cuvant ii corespunde o definitie
  - Cuvintele sun unice
- Cautarea –  $O(1)$
- Cum indexam o lista de personae dupa CNP?
- CNP – cheia, obiectul persoana - valoarea



```
public class Dictionary<TKey,TValue>
```

# Dictionary

- Metode
  - Add
    - Adauga un element la dictionar
      - Daca exista un element cu aceeaasi cheie va arunca o exceptie
  - ContainsKey
    - Returneaza True daca cheia cautata exista deja
    - Get- [] operatorul vectorial
  - Remove
    - Sterge o pereche cheie- continut din dictionar
  - Parcurgerea
    - Foreach -> KeyValuePair

## Exercitiu - inventar

- Creati un dictionar care va gestiona produse, indexandu-le dupa id-ul acestora. Un produs va avea un Id de tip guid, un nume, un pret, precum si o cantitate disponibila.

# Interface



# Interface

- [Interface in C# \(c-sharpcorner.com\)](http://c-sharpcorner.com)
- [interface - C# Reference | Microsoft Docs](#)
- este un "contract" care specifica un set de membrii (proprietati / metode)
  - Contine doar semnaturile membrilor
  - Nu contine implementari
- Mostenirea – vorbește despre **CE ESTE** o clasa
- Implementarea unei interfete – vorbește despre **CE FACE** o clasa
- Clasele **implementeaza** interfete
  - Clasa care implementeaza o interfata – trebuie sa ofere implementari pentru toti membrii definiti in interfata
  - Clasele pot implementa mai multe interfete

# Interface

- interface INumeleInterfetei
  - “interface” keyword
  - I + PascalCase
  - Definire – precum o metoda abstracta

- Membrii

- Default public
- Public
- Protected
- Private

NO.  
Just because we could  
Doesn't mean we should

- Interfetele

- Teoretic: doar membri fara implementare
- Practic: pot contine implementari

NO.  
Just because we could  
Doesn't mean we should

```
interface IAlarmSetter
{
    2 references
    public void SetAlarm(int ms, string alarmMessage);
}
```

```
interface IAlarmSetter
{
    3 references
    void SetAlarm(int ms, string alarmMessage);
}
```

<<Interface>> IAlarmSetter	
+ field1: Type	
+ field2: Type	
+SetAlarm(int ms, string message)	
+ method2(Type, Type): Type	

# Interface

- Clasele care nu sunt in relatie de mostenire – pot fi “grupate” sub variabile de tipul interfetei
  - Ceas/oven nu sunt in relatii de mostenire
  - Acces doar la metodele interfetei
  - Acces la celelalte metode – prin cast / as

```
interface IAlarmSetter
{
    2 references
    public void SetAlarm(int ms, string alarmMessage);
}

class Clock : IAlarmSetter
{
    0 references
    public void ArataOra()
    {
        Console.WriteLine("este ora 14:00");
    }
    2 references
    public void SetAlarm(int miliSecunde, string alarmMessage)
    {
        Thread.Sleep(miliSecunde);
        Console.WriteLine($"TRRRRR: {alarmMessage}");
    }
}

class Oven : IAlarmSetter
{
    0 references
    public void TurnOnHeating()
    {
        Console.WriteLine($"Incalzim cuptorul...");
    }
    2 references
    public void SetAlarm(int ms, string alarmMessage)
    {
        Console.WriteLine($"Friptura va fi gata peste {ms} ms. {alarmMessage}");
    }
}

static void Main(string[] args)
{
    IAlarmSetter[] alarms = new IAlarmSetter[2];
    IAlarmSetter alarm1 = new Oven();
    IAlarmSetter alarm2 = new Clock();

    alarms[0] = alarm1;
    alarms[1] = alarm2;

    foreach(var alarm in alarms)
    {
        alarm.SetAlarm(1000, "Alarma de 1 secunda");
    }
}
```

# Interface

- O clasa poate implementa mai multe interfete
- Interface segregation principle
  - no code should be forced to depend on methods it does not use.
  - “large” interfaces should be split into smaller interfaces with encapsulated functionality
- Nu avem IOven ci IAlarmSetter, IHeater
- Mai multe interfete mici – mai util
  - Pot fi aplicate pe clase de natura diferita din domenii diferite.
- Fiecare interfata – un anumit **aspect** al obiectului

```
interface IAlarmSetter
{
    2 references
    public void SetAlarm(int ms, string alarmMessage);
}

interface IHeater
{
    1 reference
    void TurnOnHeating();
    1 reference
    void TurnOffHeating();
}

class Oven : IAlarmSetter, IHeater
{
    2 references
    public void SetAlarm(int ms, string alarmMessage)
    {
        Console.WriteLine($"Friptura va fi gata peste {ms} ms. {alarmMessage}");
    }
    1 reference
    public void TurnOnHeating()
    {
        Console.WriteLine($"Incalzim cuptorul...");
    }
    1 reference
    public void TurnOffHeating()
    {
        Console.WriteLine($"Racim cuptorul...");
    }
}
```

IAlarmSetter

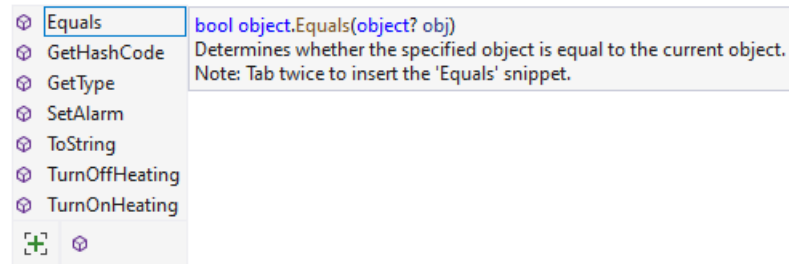
IHeater

# Interface

- O clasa poate implementa mai multe interfete
- In functie de tipul variabilei
  - Acces la membri obiectului
  - Acces la membri anumitei interfete

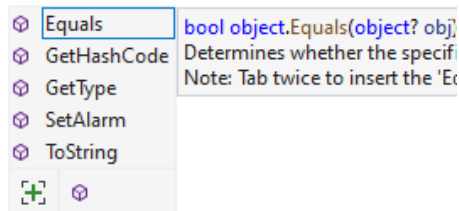
```
var cuptor = new Oven();
```

```
cuptor.
```



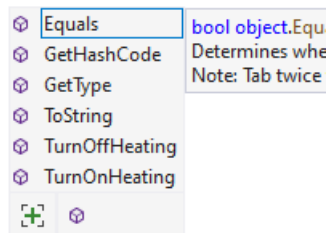
```
IAlarmSetter heater = cuptor as IAlarmSetter;
```

```
heater.
```



```
IHeater heater = cuptor as IHeater;
```

```
heater.
```



# Interface – mostenirea interfetelor

- Mostenire multipla
- IOven
  - contine toti membrii definiti de interfetele mostenite
  - poate adauga functionalitati noi

```
interface IAlarmSetter
{
    3 references
    void SetAlarm(int ms, string alarmMessage);
}

2 references
interface IHeater
{
    1 reference
    void TurnOnHeating();
    1 reference
    void TurnOffHeating();
}

0 references
interface IOven : IHeater, IAlarmSetter
{
    0 references
    void CookRoast(Roast roastToCook);
}
```

# Interface – mostenirea interfetelor

- In functie de interfata definita ca tip al variabilei, firul de executie va avea acces la membrii specifici.

```
class Oven : IOven
{
    2 references
    public void SetAlarm(int ms, string alarmMessage)
    {
        Console.WriteLine($"Friptura va fi gata peste {ms} ms. {alarmMessage}");
    }
    1 reference
    public void TurnOnHeating()
    {
        Console.WriteLine($"Incalzim cuptorul...");
    }
    1 reference
    public void TurnOffHeating()
    {
        Console.WriteLine($"Racim cuptorul...");
    }
    1 reference
    public void CookRoast(Roast roastToCook)
    {
        Console.WriteLine($"Prepar friptura {roastToCook}...");
    }
}
```

```
Roast friptura = new Roast();
Oven cuptor = new Oven();
```

```
IOven asOven = cuptor as IOven;
asOven.CookRoast(friptura);
asOven.TurnOnHeating();
asOven.SetAlarm(friptura.CookingTime, "roast is cooked");
asOven.TurnOffHeating();
```

```
IAlarmSetter asAlarmSetter = cuptor as IAlarmSetter;
asAlarmSetter.SetAlarm(friptura.CookingTime, "roast is cooked");
```

```
IHeater asHeater = cuptor as IHeater;
asHeater.TurnOnHeating();
asHeater.TurnOffHeating();
```

# Interfete si mostenirea claselor

- O clasa care mosteneste o alta clasa
  - Va implementa implicit interfetele superclasei

```
class Oven : IOven
{
    4 references
    public void SetAlarm(int ms, string alarmMessage)
    {
        Console.WriteLine($"Friptura va fi gata peste {ms} ms. {alarmMessage}");
    }
    5 references
    public virtual void TurnOnHeating()
    {
        Console.WriteLine($"Incalzim cuptorul...");
    }
    3 references
    public void TurnOffHeating()
    {
        Console.WriteLine($"Racim cuptorul...");
    }
    2 references
    public void CookRoast(Roast roastToCook)
    {
        Console.WriteLine($"Prepar friptura {roastToCook}...");
    }
}
1 reference
class ElectricOven : Oven
{
    5 references
    public override void TurnOnHeating()
    {
        Console.WriteLine("Incalzzzzzzzzzzzzzzzzzzzzzzzzim cuptorul");
    }
}

var cuptorElectric = new ElectricOven();
IHeater asElectricHeater = cuptorElectric as IHeater;
asElectricHeater.TurnOnHeating();
```



# Interfete si mostenirea claselor

- Clasele abstracte
  - Nu sunt obligate sa implemetneze toti membrii interfetelor
- Subclasele non-abstracte
  - Vor fi obligati sa implementeze toti membrii neimplementati ai interfetelor
  - Vor putea suprascrie membri interfetei implementati in superclasa
    - Override keyword.
    - Interfata este **reimplementata**

```
interface IAlarmSetter
{
    7 references
    void SetAlarm(int ms, string alarmMessage);
}
1 reference
abstract class WristWatch : IAlarmSetter
{
    5 references
    public abstract void SetAlarm(int ms, string alarmMessage);

    0 references
    public void OpenWristband()
    {
        Console.WriteLine("wristband has been opened");
    }
}
1 reference
class Smartwatch : WristWatch
{
    private Display display;

    0 references
    public Smartwatch(Display display)
    {
        this.display = display;
    }
    5 references
    public override void SetAlarm(int ms, string alarmMessage)
    {
        Console.WriteLine($"alarm was set to {ms} with message {alarmMessage}");
    }
}
```

# Interfete vs clase abstracte - asemanari

- Sunt abstractizari
- Forteaza implementarea membrilor specificati
- Membrii specificati nu au cod
- Nu pot fi instantiate direct
- Pot specifica atat membrii de instanta cat si membrii statici

# Interfete vs clase abstracte - deosebiri

## Clasa abstracte

- Formeaza relatii “is-a”
- Mostenire simpla
- Nivel de access specificat explicit
- Poate prezenta cod (constructori, membrii concreti)
- Membrii concreti nu trebuie re-implementati in clasele derivate (doar cei abstracti)
- Poate specifica campuri, constante, constructori, destructori, tipuri imbricate, proprietati, indexatori, evenimente, metode

## Interfeta

- Reprezinta o specificatie “can-do”
- O clasa poate implementa mai multe interfete
- Interfetele – mostenire multipla
- Nivel de access public implicit\*
- Nu ar trebui sa contina cod (post C#8 poate contine)
- Toti membrii specificati trebuie implementati in clasele non abstracte care realizeaza implementarea
- Poate specifica doar Proprietati / Indexatori / Evenimente / Metode

# Interfete vs clase abstracte – deosebiri – rezumat

## Clase abstracte

- Formeaza relatii “is-a”
- Pot contine behaviour
- Mostenire simpla

## Interfete

- Reprezinta o specificatie “can-do”
- Nu ar trebui sa contina implementari concrete
- Mostenire multipla
- Clasele – implementari multiple
- **Interface segregation principle**

# Interfete generice

- Interfetele pot fi generice
- Clasele care le implementeaza trebuie sa handle-ue si tipul de date generic
  - Fie ca parametru

```
J  
1 reference  
interface IGenericSwapper<T>  
{  
    1 reference  
    void Swap(ref T object1,ref T object2);  
}  
  
0 references  
class GenericSwapper<T> : IGenericSwapper<T>  
{  
    1 reference  
    public void Swap(ref T object1,ref T object2)  
    {  
        T aux = object1;  
        object1 = object2;  
        object2 = aux;  
    }  
}
```

# Interfete generice

- Interfetele pot fi generice
- Clasele care le implementeaza trebuie sa handle-ue si tipul de date generic
  - Fie prin declararea explicita a tipului
  - In cazul de clasa subclasa va putea lucra
    - doar cu obiecte de tipul declarat explicit
    - **cu obiecte derivate din tipul explicit**

```
interface IGenericSwapper<T>
{
    4 references
    void Swap(ref T object1, ref T object2);
}
2 references
class IntSwapper : IGenericSwapper<int>
{
    1 reference
    public void Swap(ref int object1, ref int object2)
    {
        int aux = object1;
        object1 = object2;
        object2 = aux;
    }
}
```

```
IntSwapper intSwapper = new IntSwapper();
```

intSwapper.W

Equals	
GetHashCode	
GetType	
Swap	void IntSwapper.Swap(ref int object1, ref int object2)
ToString	
 	

# Generic types inference

- Atunci cand utilizam metode generice, compilatorul C# poate deduce tipul generic folosit pentru acea metoda pe baza tipului parametrilor transmisi, astfel incat sa nu mai fie necesare specificarea lui intre paranteze ascutite.
- “generic type inference”
  - Ca si determinarea automata a tipului la “var”.
  - Tipul generic este dedus pe baza parametrilor folositi
- Obs: la initializarea claselor, structurilor generice inferenta nu poate functiona

Va multumesc!