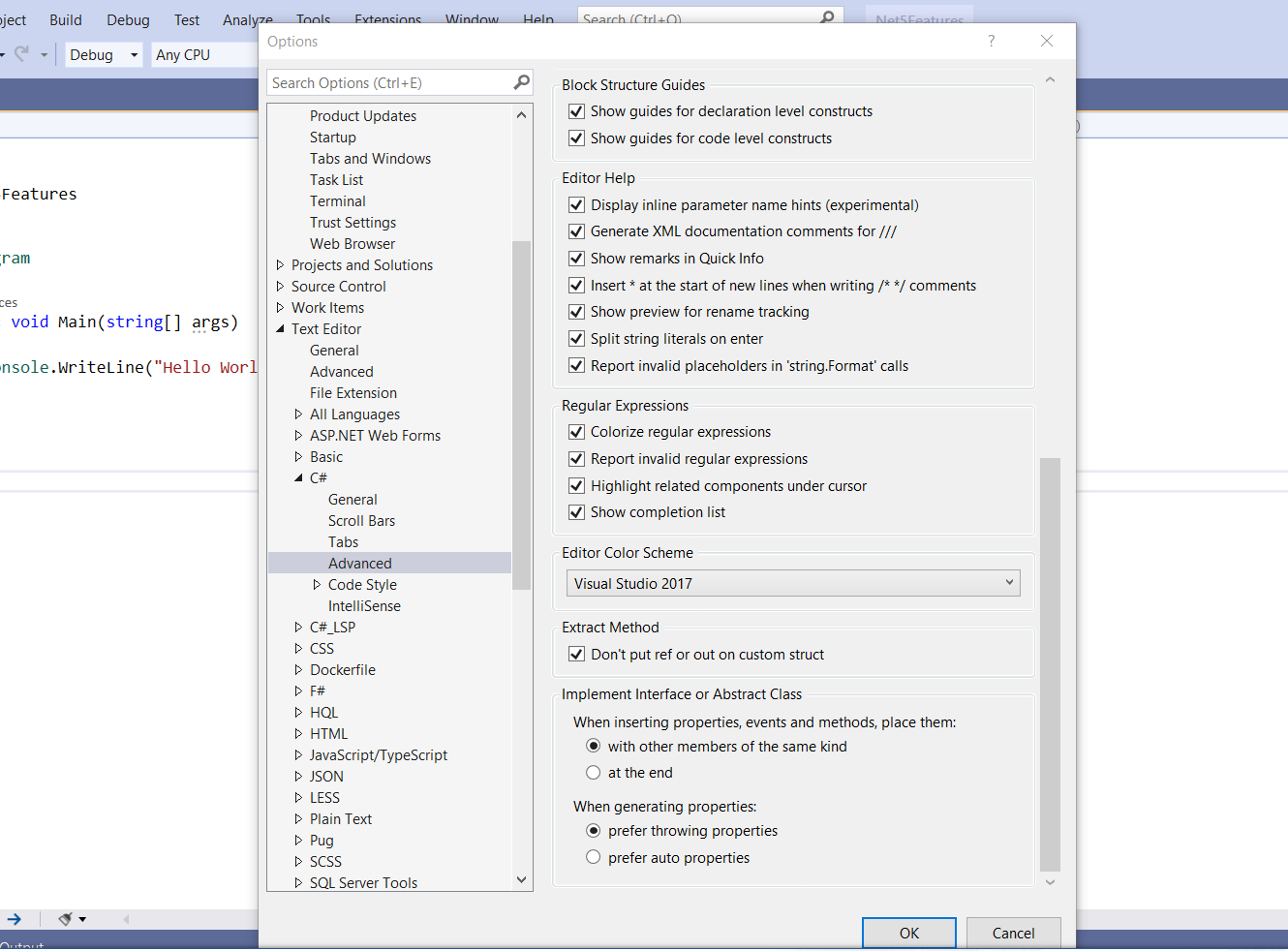
**.Net 5 features**

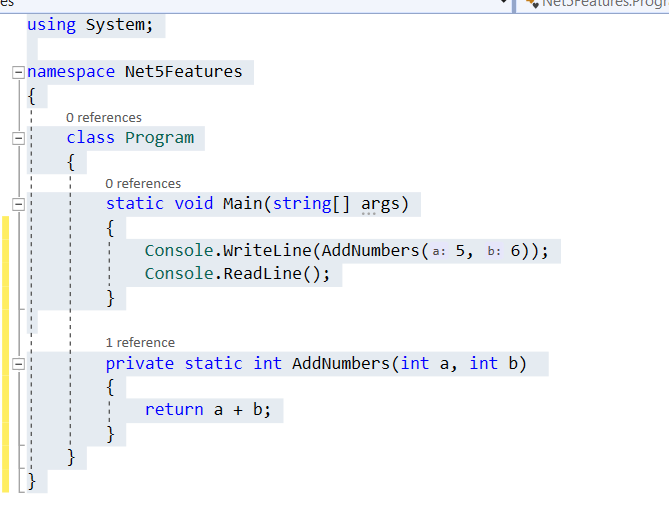
**Notes:-**

**1-to show hit parameter features in vs 2019 16.8.2 do the following steps:-**

**A-on the tools > options > Text editor > C# > select the option called**

**Display inline parameter name hints (experimental)**



**It will show code snippet on each parameter as below**

**Features:-**

**1-Top level program:-**

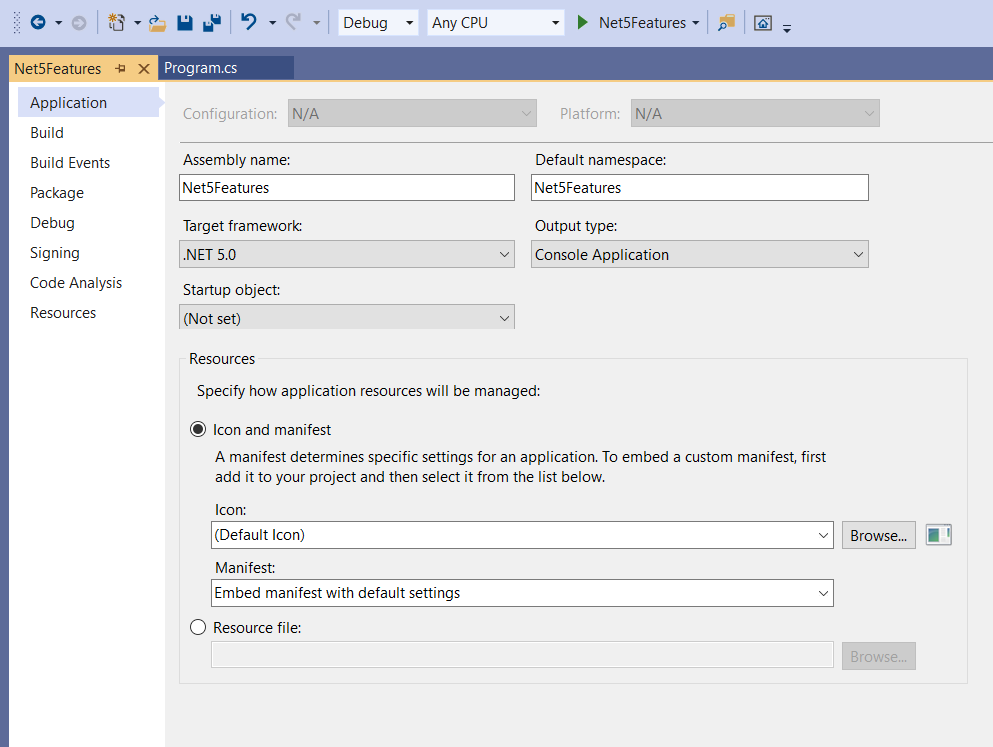
**By default we have namespace > type name like class, enum , etc .. , method name**

**And we have entry point to initiate execute our code**

**With Top level program, you can only create one class of this type and remove namespace and class name and set only the code directly as below**

**This is useful for the program.cs that represent the entry point for the applicaiton**

**2-make sure that the application convert from .net core 3.1 to .net 5 as below**



**3-we cannot add access modifier on the top level program only the method without it**

**4-if we add any type like enum , class , it will understood as end code for the top level program , so its prefer to put in separate file**

**Example:-**

**Code sample in Top level program**

**using System;**

**Console.WriteLine(AddNumbers(5, 6));**

**Console.ReadLine();**

**static int AddNumbers(int a, int b)**

**{**

**return a + b;**

**}**

**2-init setters:**

**Notes:-**

**1-if we want to prevent property from modified we have many options:**

**1-set property private set so cannot modified unless inside the class**

**2-set custom constructor with using the previous point**

**3-use the init property to only change the value in two scerio**

**A-on the constructor initialize**

**B-on the initialize instance without constructor**

**Example old way:-**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**namespace Net5Features{**

**public class PersonModel{**

**public int Id { get; private set; }**

**public string Fname { get; set; }**

**public string Lname { get; set; }**

**public PersonModel(string fname, string lname){**

**this.Fname = fname;**

**this.Lname = lname;}**

**public void changeId(int Id){this.Id = Id;}}}**

**//on the main program we can access to the Id by public method that internally change //the value**

**using Net5Features;**

**using System;**

**Console.WriteLine(AddNumbers(5, 6));**

**var person = new PersonModel("ali", "ahmad");**

**person.changeId(2);**

**Console.WriteLine($"{person.Fname} {person.Lname} ({person.Id})");**

**Console.ReadLine();**

**static int AddNumbers(int a, int b){return a + b;}**

**Example new way:-**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**namespace Net5Features{**

**public class PersonModel{**

**public int Id { get; init; }**

**public string Fname { get; set; }**

**public string Lname { get; set; }}}**

**using Net5Features;**

**using System;**

**Console.WriteLine(AddNumbers(5, 6));**

**var person = new PersonModel() {Id = 1, Fname = "ali", Lname = "ahmad" };**

**//we cannot after initialize change the Id only on the initialize on constructor or instance**

**person.Id = 2;**

**Console.WriteLine($"{person.Fname} {person.Lname} ({person.Id})");**

**Console.ReadLine();**

**static int AddNumbers(int a, int b){return a + b;}**

**3-Know object initializer**

**There is shortcut way to initialize variable in class instance**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Text;**

**using System.Threading.Tasks;**

**namespace Net5Features{**

**public class PersonModel{**

**public int Id { get; init; }**

**public string Fname { get; set; }**

**public string Lname { get; set; }**

**public PersonModel(){}**

**public PersonModel(int id, string fname, string lname){**

**this.Id = id;**

**this.Fname = fname;**

**this.Lname = lname;}}}**

**in the program.cs we set the following code**

**using Net5Features;**

**using System;**

**//by using var strongly type that can hold whatever type**

**var person = new PersonModel() {Id = 1, Fname = "ali", Lname = "ahmad" };**

**//or using declare type in variable and set to anonymous instance**

**PersonModel person1 = new () { Id = 1, Fname = "ali", Lname = "ahmad" };**

**PersonModel person2 = new(2, "Isam", "Ali");**

**4-Relational and logical pattern**

**Notes:-**

**1-with relation and pattern matching we can simplify our code in swtich statement which supported in C# 8.0 and above**

**Steps:-**

**1-create blazor server app and change the below code in WeatherForecastService class**

**public class WeatherForecastService{**

**private static readonly string[] Summaries = new[]{**

**"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"};**

**public Task<WeatherForecast[]> GetForecastAsync(DateTime startDate){**

**var rng = new Random();**

**var results = Enumerable.Range(1, 5).Select(index => new WeatherForecast{**

**Date = startDate.AddDays(index),**

**TemperatureC = rng.Next(-20, 55),**

**// Summary = Summaries[rng.Next(Summaries.Length)]}).ToArray();**

**//relation pattern matching is > , < , >= , <=**

**//logical pattern matching is and , or , not**

**foreach (var res in results){**

**res.Summary = res.TemperatureC switch{**

**< 0 => "Well below freezing",**

**>=0 and < 32 => "Freezing",**

**32 or 212 => "Exactly freezing or boiling",**

**> 32 and < 65 => "Cool",**

**>= 65 and < 85 => "Warm",**

**>= 85 and < 100 => "Hot",**

**\_ => "unknow"};}**

**return Task.FromResult(results);}}**

**5-Not null expression**

**Notes:-**

**1-we can use the is not null expression as well is null expression as below**

**PersonModel person1 = new () { Id = 1, Fname = "ali", Lname = "ahmad" };**

**if (person1 is not null) {**

**Console.WriteLine($"{person1.Fname} {person1.Lname} ({person1.Id})");}**

**6-Record**

**Notes:-**

**1-record act as value type, but actually as reference type, and has prebuild code to make life easier**

**2-record is immutable which when you initialize it cannot change like its class with init properties**

**3-record has built in methods that overwrite built in method like ToString() and clone()**

**4-you can write methods and properities inside records like class , if you want to show property in the ToString you have to set access modifier as public only**

**using Newtonsoft.Json;**

**using System;**

**namespace RecordTypePro{**

**/// <summary>**

**/// Benefits of Record Type:**

**/// 1-Simple to use**

**/// 2-Thread Safe (becouse if we have two thread working on the same class instance)**

**/// 3-easy to pass (share)**

**///**

**/// When to use records**

**/// 1-Capuring external data that doesnot change - weatherService,SWAPI,**

**/// 2-API Calls , and ViewModels**

**/// 3-Processing Data**

**/// 4-read only data**

**///**

**/// When not to use records**

**/// 1-when you want to update value (entity framework)**

**/// </summary>**

**class Program{**

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

**//the difference between record and struct**

**//struct is value type and cannot be inherited or initalize**

**//record can inherit from record and class inherit from class**

**//Record is class with extra stuff**

**RecordType rc1 = new("F1", "L1");**

**RecordType rc2 = new("F1", "L1");**

**RecordType rc3 = new("F3", "L3");**

**//internally its serialzie the object itself**

**Console.WriteLine(rc1);**

**//the records is reference but acting as value type it check to the value not to the address**

**Console.WriteLine($"Are Records Equal : {Equals(rc1, rc2)}");**

**//it will check to the reference address not value**

**Console.WriteLine($"Are Records Reference Equal : {ReferenceEquals(rc1, rc2)}");**

**//it will check to the value becouse its record type**

**Console.WriteLine($"Are Records Reference == : {rc1 == rc2}");**

**Console.WriteLine($"Are Class Reference == : {rc1 != rc2}");**

**//the method GetHashCode() its get the hash code that represent the object if changed it will change the hash code**

**//so in record type if the values are the same in two record instances the hash code will be the same**

**//while in the class it will be different becouse its act as reference type**

**Console.WriteLine($"Hash Code == : {rc1.GetHashCode()}");**

**Console.WriteLine($"Hash Code == : {rc2.GetHashCode()}");**

**Console.WriteLine($"Hash Code == : {rc3.GetHashCode()}");**

**//Deconstruct Record Type (it internally function that deconstruct the record type to the anonmous tuple )**

**var (fn, ln) = rc1;**

**Console.WriteLine($"The First Name is {fn} and Last Name is {ln}");**

**//Clonening object**

**//in record type we can clone the same object to another object but not refer to the same reference address**

**var rc4 = rc1 with{fname = "Testing Final"};**

**//so it will make copy of the rc1 and refer to its own instance**

**Console.WriteLine(rc4);**

**Console.WriteLine(rc1);**

**//RecordType**

**RecordType2 rec2 = new("Ali", "Ahmad");**

**Console.WriteLine(rec2);**

**Console.WriteLine(rec2.sayHello());**

**RecordType3 rec3 = new(1,"Ali","Ahmad");**

**rec3.sayHello();**

**Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");**

**RecordClass rct1 = new ("FC1", "LC1");**

**RecordClass rct2 = new ("FC1", "LC1");**

**RecordClass rct3 = new ("FC3", "LC3");**

**//in order to make the same result with record we have to override the ToString() with using SerializeObject**

**Console.WriteLine(rct1);**

**//becouse its point to different address the result is no**

**Console.WriteLine($"Are Class Equal : {Equals(rct1, rct2)}");**

**//it will check to the reference address not value**

**Console.WriteLine($"Are Class Reference Equal : {ReferenceEquals(rct1, rct2)}");**

**//it will check to the address becouse its class type**

**Console.WriteLine($"Are Class Reference == : {rct1 == rct2}");**

**Console.WriteLine($"Are Class Reference == : {rct1 != rct2}");**

**//so in record type if the values are the same in two record instances the hash code will be the same**

**//while in the class it will be different becouse its act as reference type**

**Console.WriteLine($"Hash Code == : {rct1.GetHashCode()}");**

**Console.WriteLine($"Hash Code == : {rct2.GetHashCode()}");**

**Console.WriteLine($"Hash Code == : {rct3.GetHashCode()}");**

**//In order to apply Deconstruct Record Type we have to make custom mehtod and send two variable as output parameters**

**string firsName;**

**string lastName;**

**rct1.Deconstruct(out firsName, out lastName);**

**Console.WriteLine($"The First Name is {firsName} and Last Name is {lastName}");**

**Console.ReadLine();}}**

**public record RecordType(string fname,string lname);**

**public record RecordType2(string FirstName, string LastName){**

**private string \_firstName = FirstName;**

**public string FirstName {**

**get { return \_firstName.Substring(0, 1); }**

**init { }}**

**//the properities must be in public access modifier to accept ToString()**

**//public string FirstName { get; init; } = FirstName;**

**public string FullName { get { return $"Full Name : {FirstName} {LastName}"; } }**

**public string sayHello(){return $"Hello {FullName}";}}**

**//Dont do this with the records**

**//1- use set; instead of init**

**//2- declare record without constructor paramters**

**//3- using clone of record as replacement of mutability of class becouse it will take large size of memory**

**public record InvalidRecord{**

**public string firstName { get; set; }**

**public string lastName { get; set; }}**

**public record RecordType3(int id,string fname,string lname) : RecordType2(fname,lname);**

**//equivalent to the RecordType**

**public class RecordClass{**

**public string Fname { get; init; }**

**public string Lname { get; init; }**

**public RecordClass(string fname, string lname){**

**this.Fname = fname;**

**this.Lname = lname;}**

**//equivalent to ToString in Record Type**

**public override string ToString(){return JsonConvert.SerializeObject(this);}**

**public void Deconstruct(out string firstName, out string lastName){**

**firstName = Fname;**

**lastName = Lname;}}**}