**How to make Optional Parameters in C#**

**How to make Optional Parameters in C# with Examples**

In this article, I am going to discuss **How to make Optional Parameters in C#**with Examples. Please read our previous article, where we discussed **[Automappers in C#](https://dotnettutorials.net/lesson/automapper-in-c-sharp/)**. This is also one of the most frequently asked interview questions. So here we will discuss the different options that are available in C# to make the method parameters optional.

**How to make Optional Parameters in C#?**

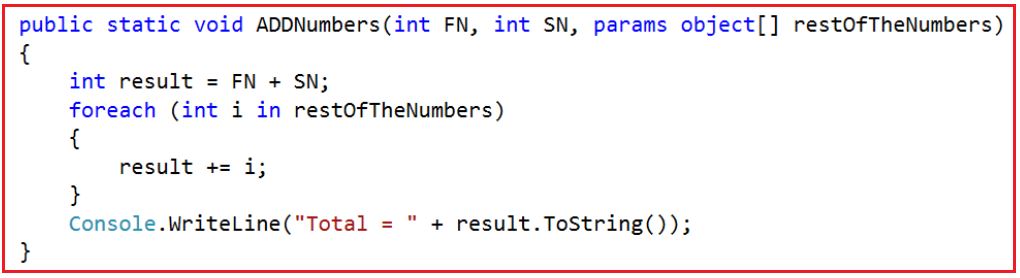
We can make the method parameters optional in C# in many different ways as follows.

1. **Using Parameter Array**
2. **Method Overloading**
3. **Specify Parameter Defaults**
4. **Using OptionalAttribute**

So, let us discuss all these options one by one with examples.

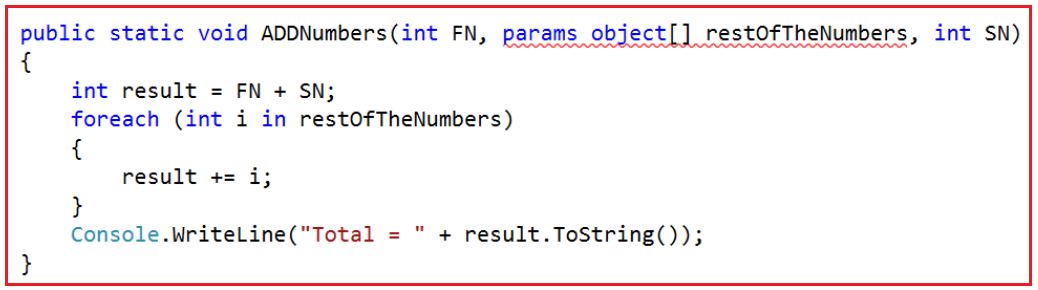
**Using Parameter Array to Make Optional Parameter in C#:**

Let us understand how to make method parameters optional using parameter array in C# with an example. Consider the following ADDNumbers method. Here, we are declaring the first and second parameters as integers and the third parameter is a parameter array i.e. params object[].



The above ADDNumbers method allows the user to add 2 or more numbers. The FN and SN parameters are mandatory parameters whereas the restOfTheNumbers parameter is optional. If the user wants to add just two numbers, then he can invoke the method as shown below  
**ADDNumbers(10, 20);**  
On the other hand, if the user wants to add 5 numbers, then he can invoke the method in the below two ways.  
**ADDNumbers(10, 20, 30, 40, 50);**  
**or**  
**ADDNumbers(10, 20, new object[]{30, 40, 50});**

The parameter array must be the last parameter in the formal parameter list. The following function will not be compiled as the parameter array not the last parameter of the parameter list.



**The complete example is given below.**

**using** *System;*

**namespace** *OptionalParameter*

**{**

**class** Program

**{**

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

**{**

ADDNumbers**(**10, 20**)**;

ADDNumbers**(**10, 20, 30, 40**)**;

ADDNumbers**(**10, 20, new **object[]** **{** 30, 40, 50 **})**;

Console.ReadLine**()**;

**}**

**public** **static** **void** ADDNumbers**(int** FN, **int** SN, **params** **object[]** restOfTheNumbers**)**

**{**

**int** result = FN + SN;

**foreach** **(int** i in restOfTheNumbers**)**

**{**

result += i;

**}**

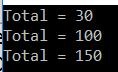
Console.WriteLine**(**"Total = " + result.ToString**())**;

**}**

**}**

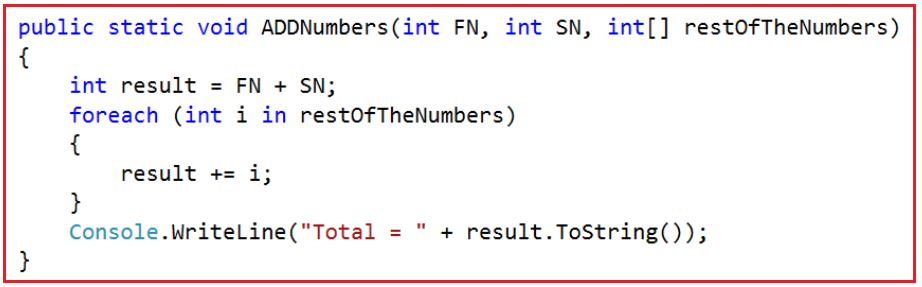
**}**

**Output:**

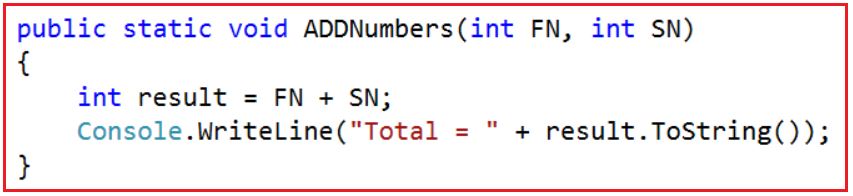


**Using Method Overloading to Make Optional Parameter in C#:**

Let us understand how to make method parameters optional using method overloading in C#. Let create the following method which will add any number of integers. Here, we created, the first two parameters as integers and the third parameter is an integer array. The first two parameters are mandatory and in the case of the third parameter, if you don’t want to pass any value, then you simply need to pass null.



If you want to add 5 integers, let’s say 10, 20, 30, 40, and 50 then you need to call the method as shown below.  
**ADDNumbers(10, 20, new int[]{30, 40, 50});**  
At the moment all the 3 method parameters are mandatory. Now, if I want to add just 2 numbers, then I can invoke the method as shown below.  
**ADDNumbers(10, 20, null);**  
Notice that, here I am passing null as the argument for the 3rd parameter. We can make the 3rd parameter optional by overloading ADDNumbers() function which takes two parameters as shown below.



Now, we have 2 overloaded versions of the ADDNumbers() function. If we want to add just 2 numbers, then we can use the overloaded version of ADDNumbers() function which takes 2 parameters as shown below.  
**ADDNumbers(10, 20);**  
Similarly, if we want to add 3 or more numbers, then we can use the overloaded version of ADDNumbers() function which takes 3 parameters as shown below.  
**ADDNumbers(10, 20, new int[] { 30, 40 });**

**The complete example is given below**

**using** *System;*

**namespace** *OptionalParameter*

**{**

**class** Program

**{**

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

**{**

ADDNumbers**(**10, 20**)**;

ADDNumbers**(**10, 20, new **int[]** **{** 30, 40, 50 **})**;

Console.ReadLine**()**;

**}**

**public** **static** **void** ADDNumbers**(int** FN, **int** SN, **int[]** restOfTheNumbers**)**

**{**

**int** result = FN + SN;

**foreach** **(int** i in restOfTheNumbers**)**

**{**

result += i;

**}**

Console.WriteLine**(**"Total = " + result.ToString**())**;

**}**

**public** **static** **void** ADDNumbers**(int** FN, **int** SN**)**

**{**

**int** result = FN + SN;

Console.WriteLine**(**"Total = " + result.ToString**())**;

**}**

**}**

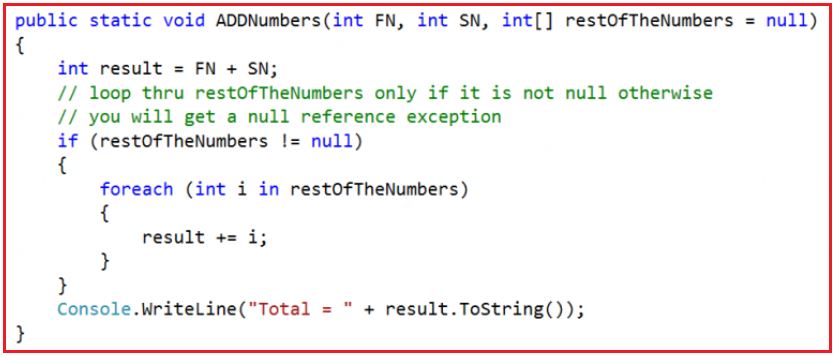
**}**

**Output:**

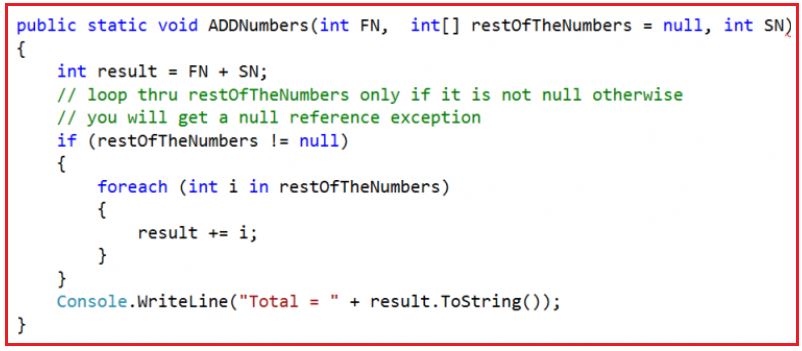
How to make method parameters optional in C#

**Making Method Parameters Optional by Specifying Parameter Defaults in C#**

Let us understand how to specify the parameter defaults to make the make method parameters optional in C#. We can make the method parameter optional by specifying a default value of null as shown below. As you can see in the below image, we have made the third parameter optional by specifying a default value null. Here, the first and second parameters are mandatory parameters.

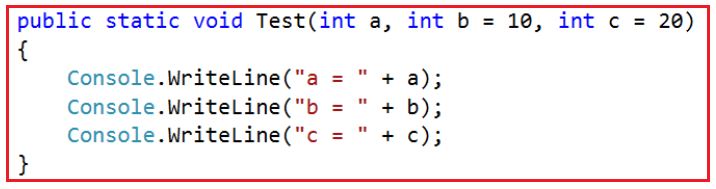


As we have specified a default value for the 3rd parameter now it becomes optional. So, if we want to add just 2 numbers, we can invoke the method as shown below.  
**ADDNumbers(10, 20);**  
On the other hand, if we want to add 3 or more numbers, then we can invoke the method ADDNumbers() as shown below.  
**ADDNumbers(10, 20, new int[] { 30, 40 });**  
The Optional parameters in C# must appear after all the required parameters. The following method will not be compiled. This is because, we are making the parameter “restOfTheNumbers” optional, but it appears before the required parameters “SN”.



**Named Parameters in C#:**

In the following method, the parameters “b” & “c” are optional.



When we invoke the above method as shown below “1” is passed as the argument for parameter “a” and “2” is passed as the argument for parameter “b” by default.  
**Test(1, 2);**

My intention is to pass “2” as the argument for parameter “c”. To achieve this we can make use of named parameters, as shown below. Notice that, I have specified the name of the parameter for which value “2” is being passed.  
**Test(1, c: 2);**

**The complete example is given below.**

**using** *System;*

**namespace** *OptionalParameter*

**{**

**class** Program

**{**

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

**{**

ADDNumbers**(**10, 20**)**;

ADDNumbers**(**10, 20, new **int[]** **{** 30, 40, 50 **})**;

Test**(**1, 2**)**;

Test**(**1, c:2**)**;

Console.ReadLine**()**;

**}**

**public** **static** **void** ADDNumbers**(int** FN, **int** SN, **int[]** restOfTheNumbers = **null)**

**{**

**int** result = FN + SN;

// loop thru restOfTheNumbers only if it is not null otherwise

// you will get a null reference exception

**if** **(**restOfTheNumbers != **null)**

**{**

**foreach** **(int** i in restOfTheNumbers**)**

**{**

result += i;

**}**

**}**

Console.WriteLine**(**"Total = " + result.ToString**())**;

**}**

**public** **static** **void** Test**(int** a, **int** b = 10, **int** c = 20**)**

**{**

Console.WriteLine**(**"a = " + a**)**;

Console.WriteLine**(**"b = " + b**)**;

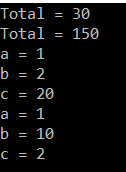
Console.WriteLine**(**"c = " + c**)**;

**}**

**}**

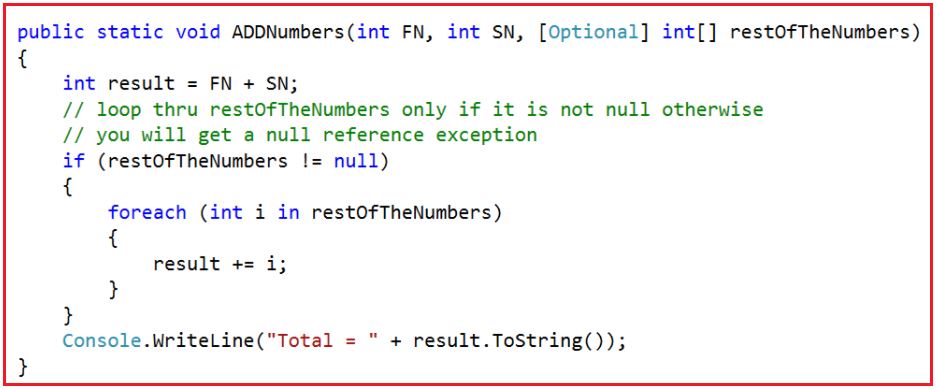
**}**

**Output:**



**How to make Parameter Optional Using OptionalAttribute in C#**

Let us understand how to make method parameters optional by using the OptionalAttribute in C# that is present in **System.Runtime.InteropServices** namespace. Please have a look at the following function. Here, we decorate the third parameter with the Optional attribute which makes this parameter optional.



Here we are making the restOfTheNumbers optional by using the [Optional] attribute. Now, if we want to add just 2 numbers, then we can call the ADDNumbers method as shown below.  
**ADDNumbers(10, 20);**  
On the other hand, if you want to add 3 or more numbers, then you can invoke the method ADDNumbers() as shown below.  
**ADDNumbers(10, 20, new int[] { 30, 40 });**

**The complete example is given below.**

**using** *System;*

**using** *System.Runtime.InteropServices;*

**namespace** *OptionalParameter*

**{**

**class** Program

**{**

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

**{**

ADDNumbers**(**10, 20**)**;

ADDNumbers**(**10, 20, new **int[]** **{** 30, 40, 50 **})**;

Console.ReadLine**()**;

**}**

**public** **static** **void** ADDNumbers**(int** FN, **int** SN, **[**Optional**]** **int[]** restOfTheNumbers**)**

**{**

**int** result = FN + SN;

// loop thru restOfTheNumbers only if it is not null otherwise

// you will get a null reference exception

**if** **(**restOfTheNumbers != **null)**

**{**

**foreach** **(int** i in restOfTheNumbers**)**

**{**

result += i;

**}**

**}**

Console.WriteLine**(**"Total = " + result.ToString**())**;

**}**

**}**

**}**

**Output:**

How to make Parameter Optional Using OptionalAttribute in C#

**Indexers in C#**

**Indexers in C# with Examples**

In this article, I am going to discuss **Indexers in C#** with Examples. Please read our previous article where we discussed [**How to make Optional Parameters in C#**](https://dotnettutorials.net/lesson/how-to-make-method-parameters-optional-csharp/)**.**As part of this article, we will discuss what indexers are and how to create and use indexers in C#.

**What are Indexers in C#?**

The Indexers in C# are the members of a class and if we define indexers in a class then the class behaves like a virtual array. So it’s a member of a class that gives access to the values of a class just like an array.

**Example: Indexers in C#**

Let us understand Indexers in C# with an example. Create a new console application. Create a new class file with the name **Employee.cs** and then copy and paste the following code into it. The following class is very straightforward; we are just declaring some properties and initializing them through the class constructor.

**namespace** *IndexersDemo*

**{**

**public** **class** Employee

**{**

//Declare the properties

**public** **int** ID **{** **get**; **set**; **}**

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

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

**public** **double** Salary **{** **get**; **set**; **}**

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

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

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

//Initialize the properties through constructor

**public** Employee**(int** ID, string Name, string Job, **int** Salary, string Location,

string Department, string Gender**)**

**{**

this.ID = ID;

this.Name = Name;

this.Job = Job;

this.Salary = Salary;

this.Location = Location;

this.Department = Department;

this.Gender = Gender;

**}**

**}**

**}**

Let’s try to create an instance of the Employee class and let’s try to consume the employee object like an array. Let’s create another class with the main method as shown below where I try to access the Employee data using index positions.

**using** *System;*

**namespace** *IndexersDemo*

**{**

**class** Program

**{**

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

**{**

//Creating the Employee instance

Employee emp = new Employee**(**101, "Pranaya", "SSE", 10000, "Mumbai", "IT", "Male"**)**;

//Accessing Employee Properties using Indexers i.e. using Index positions

Console.WriteLine**(**"EID = " + emp**[**0**])**;

Console.WriteLine**(**"Name = " + emp**[**1**])**;

Console.WriteLine**(**"Job = " + emp**[**2**])**;

Console.WriteLine**(**"Salary = " + emp**[**3**])**;

Console.WriteLine**(**"Location = " + emp**[**4**])**;

Console.WriteLine**(**"Department = " + emp**[**5**])**;

Console.WriteLine**(**"Gender = " + emp**[**6**])**;

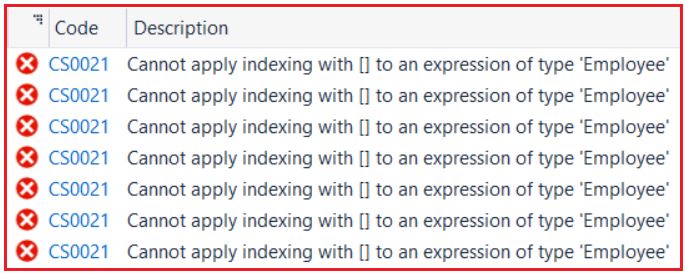
Console.ReadLine**()**;

**}**

**}**

**}**

**Now when you try to build the application, you will get the following error.**



This is because we cannot apply indexing directly to a class. We can do indexing on an array but we cannot do the same thing with a user-defined class like Employee. An array is a predefined class and all the logic’s are implemented in that class for indexing so that we can access them using indexes. But Employee is a user-defined class and we have not implemented any logic to access the class like an array.

If you want to access the class like an array then first you need to define an indexer in the class. Once you define an indexer in the class then you can start accessing the values of the class by using the index position.

**How to define an Indexer in C#?**

You need to use the following syntax to define an Indexer in a Class.



Let us understand the above syntax:

1. **Modifiers**: The “modifiers” are nothing but the access specifiers such as public, private, protected, etc.
2. **Type**: As we are dealing with the integer (ID), string (Name, Job, Department, Location, and Gender), and double (Salary) type of values, so here we need to use the “type” as Object because Object type can return any type of values.
3. **This**: The “this” keyword telling that we are defining an indexer on the current class, in this case, the current class is Employee.
4. **Int index or String name:** The int index or string name is used to specify whether you want to access the values by using its integer index position or by using the string name
5. **Get and Set:** The get accessor is used for returning the value and the set accessor is used for assigning the value.

**Example: Indexers in C#**

Let’s create an index on the Employee class for both get and set accessor. The complete code of the Employee class is given below. Here, we creating an index by using the int index position so that we can access the elements by using its integer index postion. In the case of the set accessor, the parameter “value” is implicit holds the assigned value.

**using** *System;*

**namespace** *IndexersDemo*

**{**

**public** **class** Employee

**{**

//Declare the properties

**public** **int** ID **{** **get**; **set**; **}**

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

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

**public** **double** Salary **{** **get**; **set**; **}**

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

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

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

//Initialize the properties through constructor

**public** Employee**(int** ID, string Name, string Job, **int** Salary, string Location,

string Department, string Gender**)**

**{**

this.ID = ID;

this.Name = Name;

this.Job = Job;

this.Salary = Salary;

this.Location = Location;

this.Department = Department;

this.Gender = Gender;

**}**

**public** **object** this **[int** index**]**

**{**

//The get accessor is used for returning a value

**get**

**{**

**if** **(**index == 0**)**

**return** ID;

**else** **if** **(**index == 1**)**

**return** Name;

**else** **if** **(**index == 2**)**

**return** Job;

**else** **if** **(**index == 3**)**

**return** Salary;

**else** **if** **(**index == 4**)**

**return** Location;

**else** **if** **(**index == 5**)**

**return** Department;

**else** **if** **(**index == 6**)**

**return** Gender;

**else**

**return** **null**;

**}**

// The set accessor is used to assigning a value

**set**

**{**

**if** **(**index == 0**)**

ID = Convert.ToInt32**(value)**;

**else** **if** **(**index == 1**)**

Name = **value**.ToString**()**;

**else** **if** **(**index == 2**)**

Job = **value**.ToString**()**;

**else** **if** **(**index == 3**)**

Salary = Convert.ToDouble**(value)**;

**else** **if** **(**index == 4**)**

Location = **value**.ToString**()**;

**else** **if** **(**index == 5**)**

Department = **value**.ToString**()**;

**else** **if** **(**index == 6**)**

Gender = **value**.ToString**()**;

**}**

**}**

**}**

**}**

Now, let’s try to access the values like an array, and also let’s try to modify the values like an array. So, please modify modifying the Program class Main method as shown below.

**using** *System;*

**namespace** *IndexersDemo*

**{**

**class** Program

**{**

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

**{**

Employee emp = new Employee**(**101, "Pranaya", "SSE", 10000, "Mumbai", "IT", "Male"**)**;

Console.WriteLine**(**"EID = " + emp**[**0**])**;

Console.WriteLine**(**"Name = " + emp**[**1**])**;

Console.WriteLine**(**"Job = " + emp**[**2**])**;

Console.WriteLine**(**"Salary = " + emp**[**3**])**;

Console.WriteLine**(**"Location = " + emp**[**4**])**;

Console.WriteLine**(**"Department = " + emp**[**5**])**;

Console.WriteLine**(**"Gender = " + emp**[**6**])**;

emp**[**1**]** = "Kumar";

emp**[**3**]** = 65000;

emp**[**5**]** = "BBSR";

Console.WriteLine**(**"=======Afrer Modification========="**)**;

Console.WriteLine**(**"EID = " + emp**[**0**])**;

Console.WriteLine**(**"Name = " + emp**[**1**])**;

Console.WriteLine**(**"Job = " + emp**[**2**])**;

Console.WriteLine**(**"Salary = " + emp**[**3**])**;

Console.WriteLine**(**"Location = " + emp**[**4**])**;

Console.WriteLine**(**"Department = " + emp**[**5**])**;

Console.WriteLine**(**"Gender = " + emp**[**6**])**;

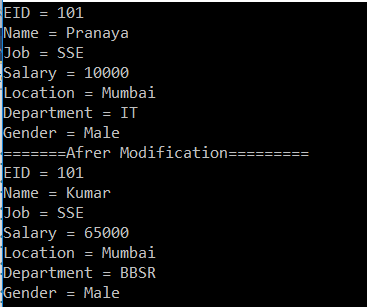
Console.ReadLine**()**;

**}**

**}**

**}**

**When we run the application, it gives us the below output.**



But in real-time, we may have more number of properties and it’s very difficult to access the values by using the integer index position. So in such cases, most of the time we need to access the values by using the property name. To do so we need to use a string name instead of an int indexer. So let’s modify the Employee class to use string name as indexer as shown below.

**using** *System;*

**namespace** *IndexersDemo*

**{**

**public** **class** Employee

**{**

//Declare the properties

**public** **int** ID **{** **get**; **set**; **}**

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

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

**public** **double** Salary **{** **get**; **set**; **}**

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

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

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

//Initialize the properties through constructor

**public** Employee**(int** ID, string Name, string Job, **int** Salary, string Location,

string Department, string Gender**)**

**{**

this.ID = ID;

this.Name = Name;

this.Job = Job;

this.Salary = Salary;

this.Location = Location;

this.Department = Department;

this.Gender = Gender;

**}**

**public** **object** this **[**string Name**]**

**{**

//The get accessor is used for returning a value

**get**

**{**

**if** **(**Name == "ID"**)**

**return** ID;

**else** **if** **(**Name == "Name"**)**

**return** Name;

**else** **if** **(**Name == "Job"**)**

**return** Job;

**else** **if** **(**Name == "Salary"**)**

**return** Salary;

**else** **if** **(**Name == "Location"**)**

**return** Location;

**else** **if** **(**Name == "Department"**)**

**return** Department;

**else** **if** **(**Name == "Gender"**)**

**return** Gender;

**else**

**return** **null**;

**}**

// The set accessor is used to assigning a value

**set**

**{**

**if** **(**Name == "ID"**)**

ID = Convert.ToInt32**(value)**;

**else** **if** **(**Name == "Name"**)**

Name = **value**.ToString**()**;

**else** **if** **(**Name == "Job"**)**

Job = **value**.ToString**()**;

**else** **if** **(**Name == "Salary"**)**

Salary = Convert.ToDouble**(value)**;

**else** **if** **(**Name == "Location"**)**

Location = **value**.ToString**()**;

**else** **if** **(**Name == "Department"**)**

Department = **value**.ToString**()**;

**else** **if** **(**Name == "Gender"**)**

Gender = **value**.ToString**()**;

**}**

**}**

**}**

**}**

**Let’s test this by modify the Main method of the Program class as shown below.**

**using** *System;*

**namespace** *IndexersDemo*

**{**

**class** Program

**{**

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

**{**

Employee emp = new Employee**(**101, "Pranaya", "SSE", 10000, "Mumbai", "IT", "Male"**)**;

Console.WriteLine**(**"EID = " + emp**[**"ID"**])**;

Console.WriteLine**(**"Name = " + emp**[**"Name"**])**;

Console.WriteLine**(**"Job = " + emp**[**"job"**])**;

Console.WriteLine**(**"Salary = " + emp**[**"salary"**])**;

Console.WriteLine**(**"Location = " + emp**[**"Location"**])**;

Console.WriteLine**(**"Department = " + emp**[**"department"**])**;

Console.WriteLine**(**"Gender = " + emp**[**"Gender"**])**;

emp**[**"Name"**]** = "Kumar";

emp**[**"salary"**]** = 65000;

emp**[**"Location"**]** = "BBSR";

Console.WriteLine**(**"=======Afrer Modification========="**)**;

Console.WriteLine**(**"EID = " + emp**[**"ID"**])**;

Console.WriteLine**(**"Name = " + emp**[**"Name"**])**;

Console.WriteLine**(**"Job = " + emp**[**"job"**])**;

Console.WriteLine**(**"Salary = " + emp**[**"salary"**])**;

Console.WriteLine**(**"Location = " + emp**[**"Location"**])**;

Console.WriteLine**(**"Department = " + emp**[**"department"**])**;

Console.WriteLine**(**"Gender = " + emp**[**"Gender"**])**;

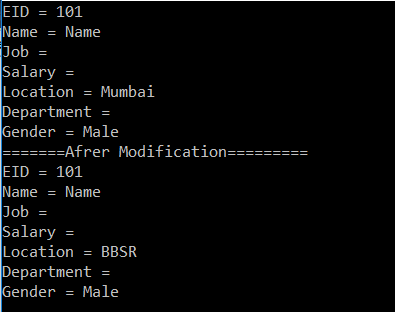
Console.ReadLine**()**;

**}**

**}**

**}**

**When we run the application, we get the following output**



As you can see, we are not getting the data for Job, Salary, and Department. This is because the indexers are case-sensitive. To get or set the data properly, either we need to convert the indexer name to upper or lower case as shown below.

**public** **object** this**[**string Name**]**

**{**

//The get accessor is used for returning a value

**get**

**{**

**if** **(**Name.ToUpper**()** == "ID"**)**

**return** ID;

**else** **if** **(**Name.ToUpper**()** == "NAME"**)**

**return** Name;

**else** **if** **(**Name.ToUpper**()** == "JOB"**)**

**return** Job;

**else** **if** **(**Name.ToUpper**()** == "SALARY"**)**

**return** Salary;

**else** **if** **(**Name.ToUpper**()** == "LOCATION"**)**

**return** Location;

**else** **if** **(**Name.ToUpper**()** == "DEPARTMENT"**)**

**return** Department;

**else** **if** **(**Name.ToUpper**()** == "GENDER"**)**

**return** Gender;

**else**

**return** **null**;

**}**

// The set accessor is used to assigning a value

**set**

**{**

**if** **(**Name.ToUpper**()** == "ID"**)**

ID = Convert.ToInt32**(value)**;

**else** **if** **(**Name.ToUpper**()** == "NAME"**)**

Name = **value**.ToString**()**;

**else** **if** **(**Name.ToUpper**()** == "JOB"**)**

Job = **value**.ToString**()**;

**else** **if** **(**Name.ToUpper**()** == "SALARY"**)**

Salary = Convert.ToDouble**(value)**;

**else** **if** **(**Name.ToUpper**()** == "LOCATION"**)**

Location = **value**.ToString**()**;

**else** **if** **(**Name.ToUpper**()** == "DEPARTMENT"**)**

Department = **value**.ToString**()**;

**else** **if** **(**Name.ToUpper**()** == "GENDER"**)**

Gender = **value**.ToString**()**;

**}**

**}**

Now if you run the application, you will get the output as expected.

**Indexers Real-Time Example in C#**

**Indexers Real-Time Example in C#**

In this article, I am going to discuss **Indexers Real-Time Example in C#**. Please read our [previous article](https://dotnettutorials.net/lesson/indexers-csharp/) before proceeding to this article where we discussed [**what indexers are and how to create and use indexers in C#**](https://dotnettutorials.net/lesson/indexers-csharp/)with examples. As we already discussed in our previous article that an indexer is a member of a class that enables an object (i.e. instance) to be indexed like an array.

**Let us understand indexers with one real-time example.**

Create a console application. Add a class file with the name **Employee.cs** and then copy and paste the following code.

**namespace** *IndexersDemo*

**{**

**public** **class** Employee

**{**

**public** **int** EmployeeId **{** **get**; **set**; **}**

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

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

**public** **double** Salary **{** **get**; **set**; **}**

**}**

**}**

**Now create another class file with the name Company.cs and then copy and paste the following code into the class.**

**using** *System.Collections.Generic;*

**using** *System.Linq;*

**namespace** *IndexersDemo*

**{**

**public** **class** Company

**{**

//Create a varibale to hold a list of employees

**private** List**<**Employee**>** listEmployees;

//Through the constructor initialize the listEmployees variable

**public** Company**()**

**{**

listEmployees = new List**<**Employee**>()**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 101, Name = "Pranaya", Gender = "Male", Salary = 1000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 102, Name = "Preety", Gender = "Female", Salary = 2000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 103, Name = "Anurag", Gender = "Male", Salary = 5000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 104, Name = "Priyanka", Gender = "Female", Salary = 4000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 105, Name = "Hina", Gender = "Female", Salary = 3000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 106, Name = "Sambit", Gender = "Male", Salary = 6000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 107, Name = "Tarun", Gender = "Male", Salary = 8000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 108, Name = "Santosh", Gender = "Male", Salary = 7000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 109, Name = "Trupti", Gender = "Female", Salary = 5000 **})**;

**}**

// The indexer takes an employeeId as parameter

// and returns the employee name

**public** string this**[int** employeeId**]**

**{**

**get**

**{**

**return** listEmployees.

FirstOrDefault**(**x =**>** x.EmployeeId == employeeId**)**.Name;

**}**

**set**

**{**

listEmployees.

FirstOrDefault**(**x =**>** x.EmployeeId == employeeId**)**.Name = **value**;

**}**

**}**

**}**

**}**

**Explanation of the above code:**

In the Company class constructor, we are initializing the variablelistEmployees by adding employees to the list. We then created an indexer by using the“this”keyword**.**The above indexer takes theemployeeIdas the parameter and returnsthat employee name. Just like properties indexers have thegetandsetaccessors. The get accessor is used to return the value whereas the set accessor is used to assign a value.

**Let see how to use the above indexer. Create another class with the Main method and then copy and paste the below code.**

**using** *System;*

**namespace** *IndexersDemo*

**{**

**class** Program

**{**

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

**{**

Company company = new Company**()**;

Console.WriteLine**(**"Name of Employee with Id = 101: " + company**[**101**])**;

Console.WriteLine**()**;

Console.WriteLine**(**"Name of Employee with Id = 105: " + company**[**105**])**;

Console.WriteLine**()**;

Console.WriteLine**(**"Name of Employee with Id = 107: " + company**[**107**])**;

Console.WriteLine**()**;

Console.WriteLine**()**;

Console.WriteLine**(**"Changing the names of employees with Id = 101,105,107"**)**;

Console.WriteLine**()**;

company**[**101**]** = "Employee 101 Name Changed";

company**[**105**]** = "Employee 105 Name Changed";

company**[**107**]** = "Employee 107 Name Changed";

Console.WriteLine**(**"Name of Employee with Id = 101: " + company**[**101**])**;

Console.WriteLine**()**;

Console.WriteLine**(**"Name of Employee with Id = 105: " + company**[**105**])**;

Console.WriteLine**()**;

Console.WriteLine**(**"Name of Employee with Id = 107: " + company**[**107**])**;

Console.ReadLine**()**;

**}**

**}**

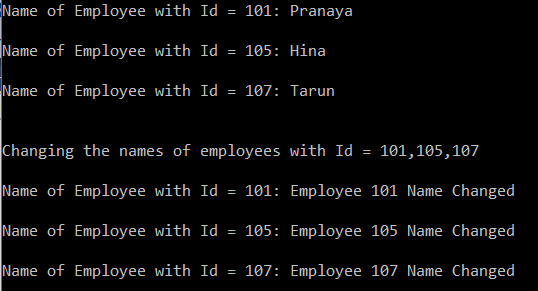
**}**

**Points to remember:**

TheEmployeeId’s101,105and107 are passed into the company object to retrieve the respective employee names. To retrieve the names of the employees, here the“get”accessor of the indexer is used. SimilarlyTo change the names of the employees, here the set accessor of the integral indexer defined on the Company class is used.

**company[101] = “Employee 101 Name Changed”;**  
**company[105] = “Employee 105 Name Changed”;**  
**company[107] = “Employee 107 Name Changed”;**

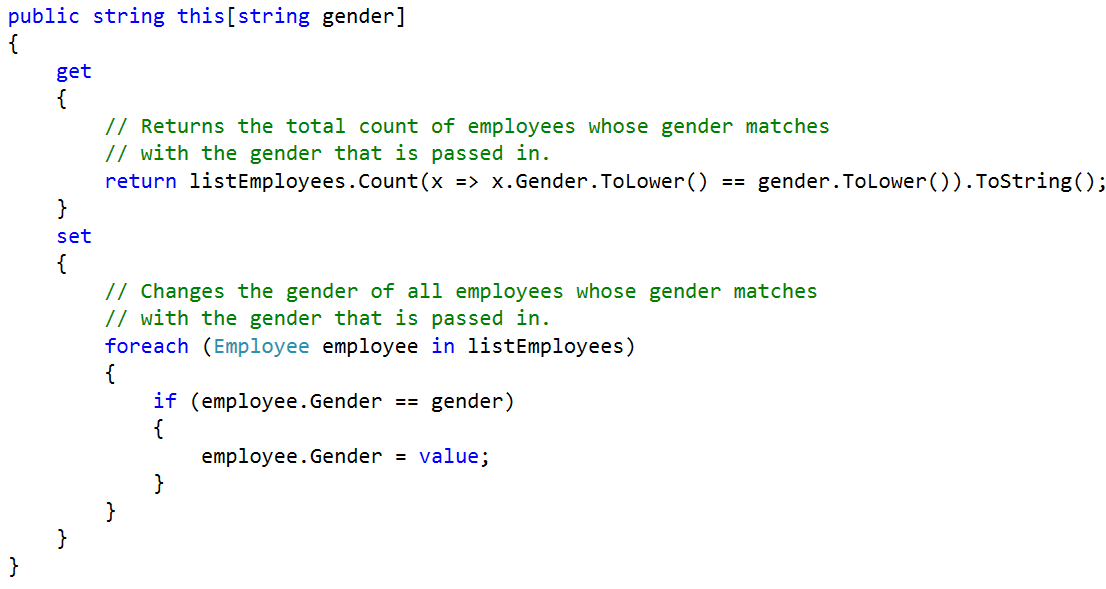
**So when we run the application, it gives us the following output.**



Notice that because of the“employeeId”indexer, now we are able to use the company object like an array.

**Overloading the Indexer in C#**

We can also overload the indexers in C#. Let us understand this with an example. As of now, we have an integer indexer in the Company class. Now let us create another indexer based on the string parameter in the company class.



The important point to keep in mind is that the indexers are overloaded based on the number and type of parameters.

**The complete code of the Company class is given below.**

**using** *System.Collections.Generic;*

**using** *System.Linq;*

**namespace** *IndexersDemo*

**{**

**public** **class** Company

**{**

//Create a varibale to hold a list of employees

**private** List**<**Employee**>** listEmployees;

//Through the constructor initialize the listEmployees variable

**public** Company**()**

**{**

listEmployees = new List**<**Employee**>()**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 101, Name = "Pranaya", Gender = "Male", Salary = 1000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 102, Name = "Preety", Gender = "Female", Salary = 2000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 103, Name = "Anurag", Gender = "Male", Salary = 5000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 104, Name = "Priyanka", Gender = "Female", Salary = 4000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 105, Name = "Hina", Gender = "Female", Salary = 3000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 106, Name = "Sambit", Gender = "Male", Salary = 6000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 107, Name = "Tarun", Gender = "Male", Salary = 8000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 108, Name = "Santosh", Gender = "Male", Salary = 7000 **})**;

listEmployees.Add**(**new Employee

**{** EmployeeId = 109, Name = "Trupti", Gender = "Female", Salary = 5000 **})**;

**}**

// The indexer takes an employeeId as parameter

// and returns the employee name

**public** string this**[int** employeeId**]**

**{**

**get**

**{**

**return** listEmployees.

FirstOrDefault**(**x =**>** x.EmployeeId == employeeId**)**.Name;

**}**

**set**

**{**

listEmployees.

FirstOrDefault**(**x =**>** x.EmployeeId == employeeId**)**.Name = **value**;

**}**

**}**

**public** string this**[**string gender**]**

**{**

**get**

**{**

// Returns the total count of employees whose gender matches

// with the gender that is passed in.

**return** listEmployees.Count**(**x =**>** x.Gender.ToLower**()** == gender.ToLower**())**.ToString**()**;

**}**

**set**

**{**

// Changes the gender of all employees whose gender matches

// with the gender that is passed in.

**foreach** **(**Employee employee in listEmployees**)**

**{**

**if** **(**employee.Gender == gender**)**

**{**

employee.Gender = **value**;

**}**

**}**

**}**

**}**

**}**

**}**

Notice that, now the Company class has2 indexers. The first indexer has aninteger (employeeId) parameter and the second indexer has got astring(gender) parameter.

To test the string indexer, that we have just created, copy and paste the following code in the Main method of the Program class as shown below.

**using** *System;*

**namespace** *IndexersDemo*

**{**

**class** Program

**{**

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

**{**

Company company = new Company**()**;

Console.WriteLine**(**"Before changing the Gender of all the male employees to Female"**)**;

Console.WriteLine**()**;

// Get accessor of string indexer is invoked to return the total count of male employees

Console.WriteLine**(**"Total Number Employees with Gender = Male:" + company**[**"Male"**])**;

Console.WriteLine**()**;

Console.WriteLine**(**"Total Number Employees with Gender = Female:" + company**[**"Female"**])**;

Console.WriteLine**()**;

// Set accessor of string indexer is invoked to change the gender all "Male" employees to "Female"

company**[**"Male"**]** = "Female";

Console.WriteLine**(**"After changing the Gender of all male employees to Female"**)**;

Console.WriteLine**()**;

Console.WriteLine**(**"Total Employees with Gender = Male:" + company**[**"Male"**])**;

Console.WriteLine**()**;

Console.WriteLine**(**"Total Employees with Gender = Female:" + company**[**"Female"**])**;

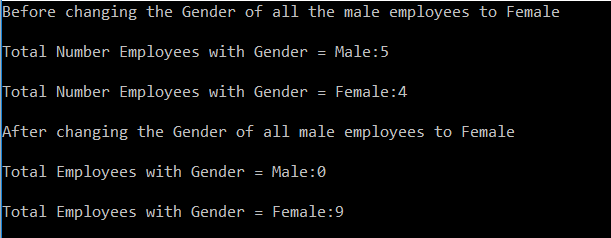
Console.ReadLine**()**;

**}**

**}**

**}**

**When we run the application, it gives us the below output.**



**Enums in C#**

**Enums in C# with Examples**

In this article, I am going to discuss **Enums in C#** with examples. Please read our previous article where we discussed [**Indexers in C#**](https://dotnettutorials.net/lesson/indexers-csharp/) in detail. At the end of this article, you will understand what are Enums in C# and when and how to use Enums in C# with some examples.

**Why do we need Enums in C#?**

The Enums are strongly typed names constants. Let’s understand enums with an example. I have an Employee class with the Name and Gender properties. Gender is an integer.

1. 0 is an Unknown gender
2. 1 is Male
3. 2 is Female

**The complete example is given below**

**using** *System;*

**using** *System.Collections.Generic;*

**namespace** *EnumsDemo*

**{**

**class** Program

**{**

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

**{**

//Create a collection to store employees

List**<**Employee**>** empList = new List**<**Employee**>()**;

empList.Add**(**new Employee**()** **{** Name = "Anurag", Gender = 0**})**;

empList.Add**(**new Employee**()** **{** Name = "Pranaya", Gender = 1 **})**;

empList.Add**(**new Employee**()** **{** Name = "Priyanka", Gender = 2 **})**;

empList.Add**(**new Employee**()** **{** Name = "Sambit", Gender = 3 **})**;

//Loop through each employees and print the Name and Gender

**foreach** **(**var emp in empList**)**

**{**

Console.WriteLine**(**"Name = {0} && Gender = {1}", emp.Name, GetGender**(**emp.Gender**))**;

**}**

Console.ReadLine**()**;

**}**

//This method is used to return the Gender

**public** **static** string GetGender**(int** gender**)**

**{**

// The switch here is less readable because of these integral numbers

**switch** **(**gender**)**

**{**

**case** 0:

**return** "Unknown";

**case** 1:

**return** "Male";

**case** 2:

**return** "Female";

**default**:

**return** "Invalid Data for Gender";

**}**

**}**

**}**

// 0 - Unknown

// 1 - Male

// 2 - Female

**public** **class** Employee

**{**

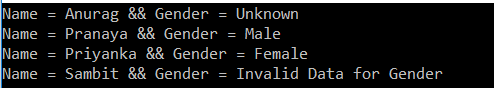
**public** string Name **{** **get**; **set**; **}**

**public** **int** Gender **{** **get**; **set**; **}**

**}**

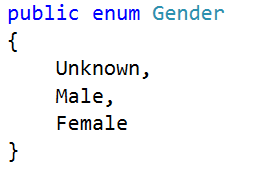
**}**

**When we run the program we get the output as expected as shown below.**

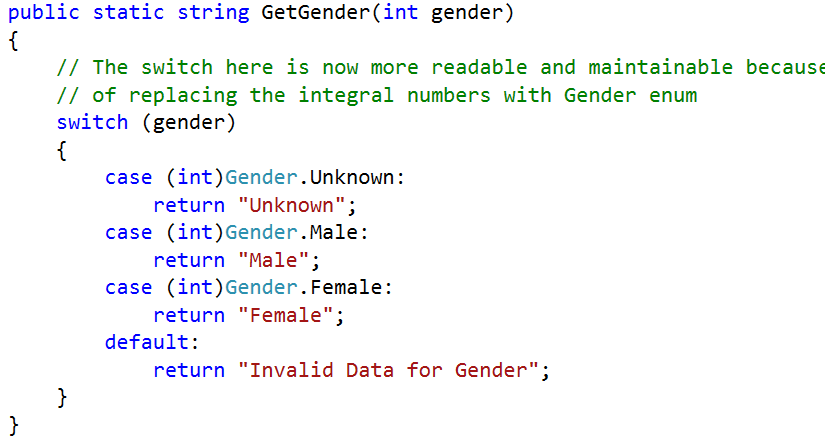


The downside of the above program is less readable as well as less maintainable. This is because it operates on integrals instead of using enums to get the gender. Now let’s see how to replace these integral numbers with enums to makes the program more readable and maintainable.

**First, create an enum for the Gender as shown below.**



**Then modify the GetGender method as shown below to enums.**



Here in the above, we are using Enums instead of integer integrals which make the code more readable and maintainable.

**The complete example is given below**

**using** *System;*

**using** *System.Collections.Generic;*

**namespace** *EnumsDemo*

**{**

**class** Program

**{**

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

**{**

//Create a collection to store employees

List**<**Employee**>** empList = new List**<**Employee**>()**;

empList.Add**(**new Employee**()** **{** Name = "Anurag", Gender = 0**})**;

empList.Add**(**new Employee**()** **{** Name = "Pranaya", Gender = 1 **})**;

empList.Add**(**new Employee**()** **{** Name = "Priyanka", Gender = 2 **})**;

empList.Add**(**new Employee**()** **{** Name = "Sambit", Gender = 3 **})**;

//Loop through each employees and print the Name and Gender

**foreach** **(**var emp in empList**)**

**{**

Console.WriteLine**(**"Name = {0} && Gender = {1}", emp.Name, GetGender**(**emp.Gender**))**;

**}**

Console.ReadLine**()**;

**}**

//This method is used to return the Gender

**public** **static** string GetGender**(int** gender**)**

**{**

// The switch here is now more readable and maintainable because

// of replacing the integral numbers with Gender enum

**switch** **(**gender**)**

**{**

**case** **(int)**Gender.Unknown:

**return** "Unknown";

**case** **(int)**Gender.Male:

**return** "Male";

**case** **(int)**Gender.Female:

**return** "Female";

**default**:

**return** "Invalid Data for Gender";

**}**

**}**

**}**

// 0 - Unknown

// 1 - Male

// 2 - Female

**public** **class** Employee

**{**

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

**public** **int** Gender **{** **get**; **set**; **}**

**}**

**public** enum Gender

**{**

Unknown,

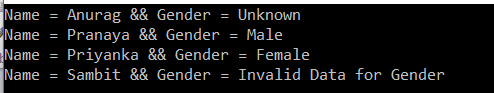
Male,

Female

**}**

**}**

**Now when you run the application you will get the output as expected as shown below.**

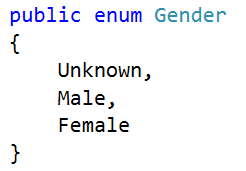


So, if a program uses a set of integral numbers then consider them replacing with enums which makes the program more Readable and Maintainable.

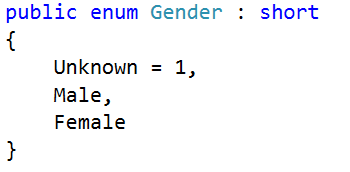
**Points to Remember about C# Enums:**

1. The Enums are enumerations.
2. Enums are **strongly typed named** constants. Hence, an explicit cast is needed to convert from the enum type to an integral type and vice versa. Also, an enum of one type cannot be implicitly assigned to an enum of another type even though the underlying value of their members is the same.
3. The default underlying type of an enum is int.
4. The default value for the first element of the enum is ZERO and gets incremented by 1.
5. It is also possible to customize the underlying type and values of enums.
6. The Enums are value types.
7. Enum keyword (all small letters) is used to create the enumerations, whereas the Enum class, contains static GetValues() and GetNames() methods which can be used to list Enum underlying type values and Names.

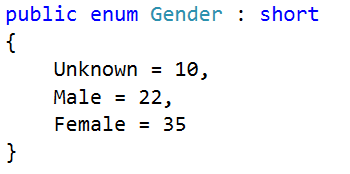
**Default underlying type is int and the value starts at ZERO**



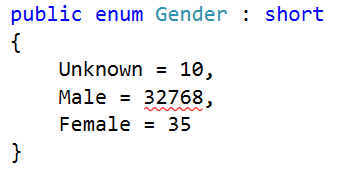
**The Gender enum underlying type is now short and the value starts from 1 and is incremented by 1**



So, in this case, the value for Male is 2 and for Female the value is 3. The Enum values need not be in sequential order. Any valid underlying type value is allowed



**The following enum will not be compiled, because the maximum value allowed for the short data type is 32767.**



**Note:** Use**short.MaxValue** to find out the maximum value that a short data type can hold

An explicit cast is needed to convert from an enum type to an integral type and vice versa.

**int i = Gender.Male;**

The above line will not compile. A compiler error will be generated stating: **Cannot implicitly convert type ‘Gender’ to ‘int’. An explicit conversion exists (are you missing a cast?)**

**Gender female = 2;**

The above line will also not compile. A slightly different compiler error will be generated stating: **The left-hand side of an assignment must be a variable, property or indexer**

**Enum of one type cannot be implicitly assigned to an enum of another type**

The Enum of one type cannot be implicitly assigned to an enum of another type even though the underlying value of their members are the same. In such cases, an explicit cast is required as shown in the below example

**namespace** *EnumsDemo*

**{**

**class** Program

**{**

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

**{**

// This following line will not compile.

// Cannot implicitly convert type 'Season' to 'Gender'.

// An explicit conversion is required.

// Gender gender = Season.Winter;

// The following line comiples as we have an explicit cast

Gender gender = **(**Gender**)**Season.Winter;

**}**

**}**

**public** enum Gender : **int**

**{**

Unknown = 1,

Male = 2,

Female = 3

**}**

**public** enum Season : **int**

**{**

Winter = 1,

Spring = 2,

Summer = 3

**}**

**}**

**Understanding GetValues() and GetNames() methods**

The enum keyword (all small letters) in C# is used to create enumerations whereas the Enum class in C# contains the static **GetValues**() and **GetNames**() methods which can be used to list the Enum underlying type values and Names. Let us understand this with an example.

**namespace** *EnumsDemo*

**{**

**class** Program

**{**

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

**{**

**int[]** Values = **(int[])**Enum.GetValues**(**typeof**(**Gender**))**;

Console.WriteLine**(**"Gender Enum Values"**)**;

**foreach** **(int** **value** in Values**)**

**{**

Console.WriteLine**(value)**;

**}**

Console.WriteLine**()**;

string**[]** Names = Enum.GetNames**(**typeof**(**Gender**))**;

Console.WriteLine**(**"Gender Enum Names"**)**;

**foreach** **(**string Name in Names**)**

**{**

Console.WriteLine**(**Name**)**;

**}**

Console.ReadKey**()**;

**}**

**}**

**public** enum Gender : **int**

**{**

Unknown = 1,

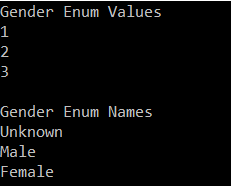
Male = 2,

Female = 3

**}**

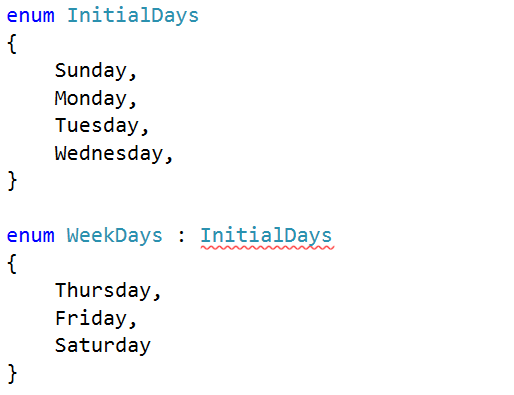
**}**

**When we run the application, it gives us the following output.**



**The Enums cannot be derived from other enums.**

Let us understand this with an example. Here we have two enums InitialDays and Weekdays and we are trying to inherit the WeekDays enums from the InitialDays enum as shown in the below image.



**When we compile the application, it gives us the below error.**

The Enums cannot be derived from other enums

So, we cannot derive enums from another enum. Well, if enums cannot be derived from enums then can be a class derived from enums? Let’s find it out with an example.

**using** *System;*

**namespace** *EnumsDemo*

**{**

**class** Program : WeekDays

**{**

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

**{**

Console.WriteLine**((int)**WeekDays.Monday**)**;

Console.WriteLine**((int)**WeekDays.Sunday**)**;

Console.WriteLine**((int)**WeekDays.Tuesday**)**;

Console.WriteLine**((int)**WeekDays.Wednesday**)**;

Console.WriteLine**((int)**WeekDays.Thursday**)**;

Console.WriteLine**((int)**WeekDays.Friday**)**;

Console.WriteLine**((int)**WeekDays.Saturday**)**;

Console.ReadKey**()**;

**}**

**}**

**public** enum WeekDays

**{**

Sunday,

Monday,

Tuesday,

Wednesday,

Thursday,

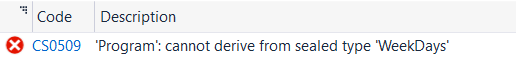
Friday,

Saturday

**}**

**}**

**When we compile the application, it gives us the following error.**



The above error states that classes cannot be derived from enums. This is because Enums are treated as sealed classes and hence all rules that are applicable to sealed classes also apply to enums.

**Duplicate Enum Members:**

Let us try to create two enum members with the same name as shown in the below example.

**using** *System;*

**namespace** *EnumsDemo*

**{**

**class** Program

**{**

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

**{**

Console.WriteLine**((int)**Gender.Unknown**)**;

Console.WriteLine**((int)**Gender.Male**)**;

Console.WriteLine**((int)**Gender.Female**)**;

Console.ReadKey**()**;

**}**

**}**

**public** enum Gender

**{**

Unknown,

Male,

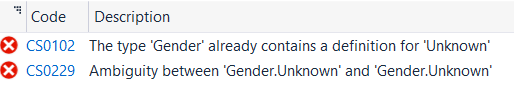
Female,

Unknown

**}**

**}**

**When we compile the program, it gives us the following error.**



The Enums are like classes, so we cannot define two members with the same name.