**Why Enums**

**Enums are strongly typed constants**. Let's understand enums with an example. I have a customer class with Name and Gender properties. Gender is an integer.   
**0 is an Unknown gender**  
**1 is Male**  
**2 is Female**  
  
  
This program is less readable and maintainable, as it operates on integrals instead of using **enums**.  
  
  
In the next session we will replace, these integral numbers with enums, which makes the program better readable and maintainable.

using System;  
public class Enums  
{  
    public static void Main()  
    {  
        Customer[] customers = new Customer[3];  
        customers[0] = new Customer()  
        {  
            Name = "Mark",  
            Gender = 1  
        };  
        customers[1] = new Customer()  
        {  
            Name = "Mary",  
            Gender = 2  
        };  
        customers[2] = new Customer()  
        {  
            Name = "Sam",  
            Gender = 0  
        };  
        foreach (Customer customer in customers)  
        {  
            Console.WriteLine("Name = {0} && Gender = {1}", customer.Name, GetGender(customer.Gender));  
        }  
    }  
  
  
    public static string GetGender(int gender)  
    {  
        // The swicth 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 Customer  
{  
    public string Name { get; set; }  
    public int Gender { get; set; }  
}

### Enums Example

using System;  
public class Enums  
{  
    public static void Main()  
    {  
        Customer[] customers = new Customer[3];  
        customers[0] = new Customer()  
        {  
            Name = "Mark",  
            Gender = Gender.Male  
        };  
        customers[1] = new Customer()  
        {  
            Name = "Mary",  
            Gender = Gender.Female  
        };  
        customers[2] = new Customer()  
        {  
            Name = "Sam",  
            Gender = Gender.Unknown  
        };  
        foreach (Customer customer in customers)  
        {  
            Console.WriteLine("Name = {0} && Gender = {1}", customer.Name, GetGender(customer.Gender));  
        }  
    }  
  
  
    public static string GetGender(Gender gender)  
    {  
        // The swicth here is now more readable and maintainable because   
        // of replacing the integral numbers with Gender enum  
        switch (gender)  
        {  
            case Gender.Unknown:  
                return "Unknown";  
            case Gender.Male:  
                return "Male";  
            case Gender.Female:  
                return "Female";  
            default:  
                return "Invalid Data for Gender";  
        }  
    }  
}  
  
  
public enum Gender  
{  
    Unknown = 0,  
    Male = 1,  
    Female = 2  
}  
  
  
public class Customer  
{  
    public string Name { get; set; }  
    public Gender Gender { get; set; }  
}

If a program uses set of integral numbers, consider replacing them with enums, which makes the program more  
    **Readable**  
    **Maintainable** .   
  
  
**1.** Enums are enumerations.  
**2.** Enums are strongly typed constants. Hence, an explicit cast is needed to convert from 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 are the same.  
**3.** The default underlying type of an enum is int.  
**4.** The default value for first element is ZERO and gets incremented by 1.  
**5.** It is possible to customize the underlying type and values.  
**6.** Enums are value types.  
**7.** Enum keyowrd (all small letteres) is used to create enumerations, where as 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  
public enum Gender  
{  
    Unknown,  
    Male,  
    Female  
}  
  
  
// Gender enum underlying type is now short and the value starts at ONE  
public enum Gender : short  
{  
    Unknown = 1,  
    Male = 2,  
    Female = 3  
}  
  
  
// Enum values need not be in sequential order. Any valid underlying type value is allowed   
public enum Gender : short  
{  
    Unknown = 10,  
    Male = 22,  
    Female = 35  
}  
  
  
// This enum will not compile, bcos the maximum value allowed for short data type is 32767.   
public enum Gender : short  
{  
    Unknown = 10,  
    Male = 32768,  
    Female = 35  
}   
  
  
**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 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  
Cannot implicitly convert type 'int' to 'Gender'. An explicit conversion exists (are you missing a cast?)  
  
**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. An explicit cast is required as shown below.**  
using System;  
public class Enums  
{  
    public static void Main()  
    {  
        // This line will not compile. Cannot implicitly convert type 'Season' to 'Gender'.   
        // An explicit conversion is required.  
        // Gender gender = Season.Winter;  
  
  
        // This 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  
}  
  
  
enum keyowrd (all small letteres) is used to create enumerations, where as Enum class, contains  static **GetValues**() and **GetNames**() methods which can be used to list Enum underlying type values and Names.  
  
  
**Sample Program listing all enum member values and Names**  
using System;  
public class Enums  
{  
    public static void Main()  
    {  
        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);  
        }  
    }  
}  
public enum Gender : int  
{  
    Unknown = 1,  
    Male = 2,  
    Female = 3  
}