You will learn about enums, static members and default arguments.

## Enums

An enum is a user-define type like a class. However, all the members are constants with implied values. It provides an efficient way of defining a set of named integral constants such as the days of the week or months of the year or even the set of pre-defined colors.

It supports a more robust programming style e.g. If you have a method that expects a month as an argument of type to a string, you will have do lots of defensive programming, instead define an enum type that has the 12 months. During development, the compiler will do the necessary checks for the require type hence you will do less programming.

## Static members

The static decorator when used on a member indicates that member belongs to the type rather than an object. You will access that member using the type rather than an object reference. Static methods or properties may only access other static members. The only instance member that can be accessed is the constructor.

You can recognize an instance member by the absence of the static modifier. Instance methods are able to access other instance members as well as class members.

# The TimeFormat Enum

Sometimes you don’t even care about the underlining values as in this case or, (Maritial status: Single, Married, Widowed etc.), (Sale status: Confirmed, Shipped, Paid etc.)

|  |
| --- |
| **TimeFormat**  Enum |
| **Values** |
| Mil  Hour12  Hour24 |

You may define/code this type in the Time.cs file, but it should be external to the Time class.

# The Time Class

**UML Class Diagram**

$ 🡪 static

- 🡪 private

+ 🡪 public

# 🡪 protected

### Code the Time class below:

This class comprise of six members: a field, two properties and two methods and a constructor a SetTimeFormat() and a ToString() method. All the fields are private and all the methods are public

|  |
| --- |
| **Time**  Class |
| **Fields** |
|  |
| **Properties** |
| -$ TIME\_FORMAT = TimeFormat.Hour12 : **TimeFormat**  + «setter absent» Hour : **int**  + «setter absent» Minute : **int** |
| **Methods** |
| + «constructor» Time(hour = 0 : **int**, minute = 0: **int**)  + ToString() : **string**  $+ SetFormat(timeFormat : **TimeFormat**) : **void** |

#### Description of class members

A class member that is decorated with the **static** keyword implies that only one copy of this member exists and it is shared by all objects of the class. i.e. If it is changed then all object will see the new value immediately.

A class method can only access other class member (fields/properties/methods)

##### Fields:

There are no fields for this class.

##### Properties:

**TIME\_FORMAT** – this class variable is of type TimeFormat and it represents the intended format of the ToString() method. This field has private accessibility and it initializes to TimeFormat.Hour12 at declaration.

**Hour** – this is an integer representing the hour value this object. This property has public read access and the setter is missing.

**Minute** – this is an integer representing the minute value this object. This property has public read access and the setter is missing.

##### Constructor:

**public Time(int hours = 0, int minutes = 0)** – This is a public constructor that takes two integer arguments with default values of **0**. If checks if the first argument is between 0 and 24 it assigns it to the Hour property otherwise a value of 0 is assign to the hour property. Similarly the second argument is check if it is between 0 and 60 then it is assigned to the Minute property otherwise a value of 0 is assigned to the Minute property.

##### Methods:

**public override string ToString()** – This public method overrides the ToString of the object class. It does not take any argument and returns a string representation of the object. The return value will depend on the field TIME\_FORMAT value of the class. So it is a good plan if you use a switch statement to handle the three possibilities. Your switch statement should check the value of the TIME\_FORMAT field.

If you have a Time object with the hours value as **18** and the minutes value as **5** then the print out of the object will be

A **switch** statement is probably the simplest way to implement this functionality

“1805” if TIME\_FORMAT is set to TimeFormat.MIL. Use the d2 format specifier to get the leading zero.

“18:05” if TIME\_FORMAT is set to TimeFormat.HOUR24. Again, use the d2 format specifier to get the leading zero.

“6:05 PM” if TIME\_FORMAT is set to TimeFormat.HOUR12. Again, use the d2 format specifier to get the leading zero.

**public static void SetTimeFormat(**TimeFormat **time\_format)** – This a class method that is public. It takes a single TimeFormat argument and assigns it to the static TIME\_FORMAT field.

This method must be static because it needs to access a static field.

### Test Harness

Insert the following code statements in your Program.cs file:

//create a list to store the objects

List<Time> times = new List<Time>()

{

new Time(9, 35),

new Time(18, 5),

new Time(20, 500),

new Time(10),

new Time()

};

//display all the objects

TimeFormat format = TimeFormat.Hour12;

Console.WriteLine($"\n\nTime format is {format}");

foreach (Time t in times)

{

Console.WriteLine(t);

}

//change the format of the output

format = TimeFormat.Mil;

Console.WriteLine($"\n\nSetting time format to {format}");

//SetFormat(TimeFormat) is a class member, so you need the type to access it

Time.SetFormat(format);

//again display all the objects

foreach (Time t in times)

{

Console.WriteLine(t);

}

//change the format of the output

format = TimeFormat.Hour24;

Console.WriteLine($"\n\nSetting time format to {format}");

//SetFormat(TimeFormat) is a class member, so you need the type to access it

Time.SetFormat(format);

foreach (Time t in times)

{

Console.WriteLine(t);

}

Your output will look like below:

Time format is Hour12

9:35 AM

6:05 PM

8:00 PM

10:00 AM

0:00 AM

Setting time format to Mil

0935

1805

2000

1000

0000

Setting time format to Hour24

09:35

18:05

20:00

10:00

00:00