ToString Method in C#

Representing data as strings is an inbuilt feature of the C# language and is a common operation in applications. In this article, we are going to learn how we can use the ToString() method in C# through an example.

Let's start.

What Is ToString() Method in C#?

This is a method that returns the string values of an object in C#. We may also need to format data in a specific way, and this method helps us accomplish that.

To demonstrate its usage, let's create a console application first.

After that, we create a new Person class:

```
public class Person
{
   public string Name { get; set; }
   public int Age { get; set; }
   public string Profession { get; set; }
   public Person(string name, int age, string profession)
   {
      Name = name;
      Age = age;
      Profession = profession;
   }
}
```

Here, we define the class and then initialize the three class properties in the constructor.

Let's now instantiate the class and call the ToString() method:

```
var person = new Person("John Doe", 30, "Software Developer");
person.ToString();
```

By default, ToString() does not take any parameters. When we print the results of the method call to the console, we get the namespace of the Person class:

```
ToStringMethod.Person
```

What we have covered is only the basic usage of the ToString() method. But this method can do much more than that.

So how can we customize the ToString() method to suit our needs?

Overriding the ToString() Method

When working with objects, we may want to explicitly define how the object data should appear. In this case, we define our own implementation of the ToString() method.

Working with our Person class, for instance, we may want to view the values of the class properties as well.

To accomplish this, we override the ToString() method:

```
public override string ToString()
{
   return $"{Name} is {Age} years old, and is a {Profession}";
}
```

We interpolate the properties of the class and return a string representing the object values.

If we print the results now, we get a completely different output:

```
John Doe is 30 years old, and is a Software Developer
```

This is just a simple demonstration of what the ToString() method can do.

Overloading the ToString() Method

The default method does not take any parameters. However, depending on the data, we may want to pass additional parameters that dictate the format.

To demonstrate overloading, let's create a new Car class:

```
public class Car
{
   public string Make { get; set; }
   public string Model { get; set; }
   public decimal Price { get; set; }
```

```
    public DateTime SoldAt { get; set; }

    public Car(string make, string model, decimal price, DateTime soldAt)
    {
        Model = model;
        Make = make;
        Price = price;
        SoldAt = soldAt;
    }
}
```

After that, let’s create an instance of the class:

```
var car = new Car("Tesla", "Model S", 100000M, DateTime.Now);
```

Here, we pass the values of the object during instantiation.

Formatting Currency

The Car class has Price property whose value is 10000M. We would like to format this value as a currency.

To do this, we can overload ToString() by passing an additional culture parameter:

```
public string ToString(string culture)
{
    var price = Price.ToString("C", new CultureInfo(culture));

    return $"{Make} {Model} costs {price} and was sold on {SoldAt}";
}
```

In this case, we use culture to format the value of Price as currency.

Calling the overloaded method:

```
car.ToString("en-US");
```

We use the "en-US" culture because we want to format the price in US Dollars.

Printing the result, we get:

```
Tesla Model S costs $100,000.00 and was sold on 12/2/2022 5:10:56 PM
```

If we pass a different culture argument, the price will be formatted using the price of that country. Also, please note that the value SoldAt is the current time and will vary depending on when you're running the application.

Formatting Dates

In the same way, we may want to use custom DateTime formatting for the SoldAt property of the class. For this, we pass one more additional parameter to the ToString() method:

```
public string ToString(string culture, string dateFormat)
{
   var price = Price.ToString("C", new CultureInfo(culture));
   var saleDate = SoldAt.ToString(dateFormat);

   return $"{Make} {Model} costs {price} and was sold on {saleDate}";
}
```

When calling this method, we also specify the format in which we want the date to be:

```
car.ToString("en-US", "dd-MMM-yyyy");
```

Printing the results, we get:

```
Tesla Model S costs $100,000.00 and was sold on 02-Dec-2022
```

Also, depending on the format we pass as the argument, we will get a different date format.

Conclusion

In this article, we have learned how to use the ToString() method in C#. Beyond the basic usage, we have demonstrated overloading and overriding the method.