# Base Keyword

In this lesson, you'll get to know the uses of the 'base' keyword in C#.

WE'LL COVER THE FOLLOWING

## What Is the `base` Keyword? #

As you already know, the `this` keyword in C# is used to refer to the *instance* of the current class.

In a similar fashion, the `base` keyword in C# is used to refer to the *base class* members from inside the immediate *derived class*. The use of `base` comes into play when we implement inheritance.

## Use Cases of the `base` Keyword #

The base keyword is used in *three* major contexts:

### Accessing Base Class Fields #

The `base` keyword can be used to access any *non-private* fields of the base class directly from the derived class. The `base` keyword followed by the dot `.` operator is used in the derived class to access a field of the base class.

Consider a field named `price` defined inside a `Product` class to keep track of the *price* of a product. Another class named as `Beverage` inherits from this `Product` class. We declare a field inside the `Beverage` class with the same name, `price`, but different value. Now if we want to access the `price` field of name, `price`, but different value. Now if we want to access the `price` field of

the *base class* from the *derived class*, we will then have to use the `base` keyword.

Let’s understand this using a bit of code.

```
class Product { //Base class Product

  public double price = 1.7; //price field inside base class

}


class Beverage : Product { // sub class Beverage extending from Product

  new double price = 1; //price field inside derived class

  public void Display() {
    //accessing the field of parent class using base*/
    Console.WriteLine("Price from the Product class: " + base.price);
    //without using base the field of current class shadows the field of parant class*/
    Console.WriteLine("Price from the Beverage class: " + this.price);

  }

}

class Demo {

  public static void Main(String[] args) {
    Beverage cola = new Beverage();
    cola.Display();
  }

}
```

> The `new` keyword in the above code at **line 10** is used to let the compiler know that we are intentionally re-declaring the `price` field in the derived class.

## Calling a Base Class Method

As with fields, `base` is also used with methods. Whenever a *base class* and the immediate *derived class* have methods with the **same name**, we use `base` to access the methods from the *base class* inside the *derived class*.

Let’s go through an example:

```
class Product {          // Base class Product

  public void Display() {   // Display method inside base class
    Console.WriteLine("I am from the Product Class!");
  }

}

class Beverage : Product { // Sub class Beverage inheriting from Product

  new public void Display() { // Display method inside derived class
    Console.WriteLine("I am from the Beverage Class!");
  }

  public void PrintOut(){
    Console.WriteLine("The Display() call with base:");
    base.Display();  // Calling the Display() of Product(base class)
    Console.WriteLine("The Display() call without base:");
    Display();       // Calling the Display() of the Beverage(derived class)
  }

}

class Demo {

  public static void Main(string[] args) {
    Beverage cola = new Beverage(); // An object of derived class
    cola.PrintOut();
  }

}
```

> The `new` keyword in the above code at **line 11** is used to let the compiler know that we are intentionally hiding the inherited `Display()` method in the derived class. This is known as *method hiding or shadowing*.

## Using with Constructors #

Another very important use of the keyword `base` is to call the *constructor* of the *base class* from inside of the *constructor* of the *derived class*.

> **Important Note:** If the base class has a default or parameter-less constructor, the compiler implicitly calls the base class constructor before the derived class constructor when an object of the derived class

> is created.

The syntax of the constructor call is as follows:

```
: base(parameters); // Calls the parameterized constructor of the base cla
ss
                    // with matching parameters from the SubClass construc
tor
```

> **Very Important:** The call to the base class constructor using `base()` should always be the first line of code after the constructor signature of the derived class.

Let's look at an example of a constructor call using `base()`.

**Note:** The below code will give an error, ❌, as there is no call to the base class constructor from inside of the derived class constructor.

```
// Base Class Product
class Product {

  // Private Fields
  private string _name;
  private double _price;
  private string _expiryDate;


  // Parameterized Constructor
  public Product(string name, double price, string expiryDate) {
    this._name = name;
    this._price = price;
    this._expiryDate = expiryDate;

  }

  // Public method to print details
  public void PrintDetails() {
    Console.WriteLine("Name: " + this._name);
    Console.WriteLine("Price: " + this._price);
    Console.WriteLine("Expiry Date: " + this._expiryDate);
  }

}

// Derived Class Beverage
class Beverage : Product {
  // Private fields
  private double _litres;
  private string _flavor;


  // Parameterized Constructor
  public Beverage(string name, double price, string expiryDate, double litres, string flavor)
   // : base(name, price, expiryDate) //calling parent class constructor
  {
      this._litres = litres;
      this._flavor = flavor;
  }

  public void BeverageDetails() {  //details of Beverage
    PrintDetails();         //calling inherited method from parent class
    // Printing fields of this class
    Console.WriteLine("Litres: " + this._litres);
    Console.WriteLine("Flavor: " + this._flavor);
  }

}

class Demo {

  public static void Main(string[] args) {
    Beverage cola = new Beverage("RC Cola", 0.7, "8/12/2019", 0.35, "Cola"); //creation of Be
    cola.BeverageDetails(); //calling method to print details
  }

}
```

Now let’s uncomment the above highlighted line in the code widget and try running the code again. It will execute this time.

```
// Base Class Product
class Product {

  // Private Fields: Common attributes of all type of products
  private string _name;
  private double _price;
  private string _expiryDate;


  // Parameterized Constructor
  public Product(string name, double price, string expiryDate) {
    this._name = name;
    this._price = price;
    this._expiryDate = expiryDate;

  }

  // public method to print details
  public void PrintDetails() {
    Console.WriteLine("Name: " + this._name);
    Console.WriteLine("Price: " + this._price);
    Console.WriteLine("Expiry Date: " + this._expiryDate);
  }

}

// Derived Class Beverage
class Beverage : Product {

  // Private fields : Fields specific to the derived class
  private double _litres;
  private string _flavor;

  // Parameterized Constructor
  public Beverage(string name, double price, string expiryDate, double litres, string flavor)
    : base(name, price, expiryDate) //calling parent class constructor
  {
      this._litres = litres;
      this._flavor = flavor;
  }

  public void BeverageDetails() {  //details of Beverage
    PrintDetails();         //calling inherited method from parent class
    // Printing fields of this class
    Console.WriteLine("Litres: " + this._litres);
    Console.WriteLine("Flavor: " + this._flavor);
  }

}

class Demo {

  public static void Main(string[] args) {
    Beverage cola = new Beverage("RC Cola", 0.7, "8/12/2019", 0.35, "Cola"); //creation of Be
    cola.BeverageDetails(); //calling method to print details
  }

}
```

This time the execution is successful.

> **Note:** In a constructor we can include a call to `base()` **or** `this()` but not both. Also, these calls can only be made after a constructor's signature and before the start of curly brackets.

---

This covers everything about the `base` keyword. In the next lesson, we will discuss the different types of inheritance.