**C#**

**C# Features**

* **.NET** consists of **CLR** (common language runtime).
* **CLR** manages **execution** & **memory** etc.

**Facts**

* Both ***char*** and ***string*** are data types in **C#**.
* ***char*** uses **single quotes** & ***string*** uses **double quotes**.

**Using statements**



**Numerical Input**



**switch-cases**

* ***default*** block works like ***else*** in **switch-cases**.

**while’s Increment**



* Increments value of ***x*** in **each** iteration.

**Optional & Named Argument**



* The **default/optional argument** must be written after **non-default** ones.



* We can call a method with different order of argument as above.

**Type Casting**



* Conversion to **int** type.
* **f** in **0.8f** is used to allow a float’s operation with an integer, and make final result a float.
* Its **compulsory** to add ***f*** in ***float***, unlike ***double***.
* It **can’t** be done with ***chars*** & ***strings***.

**Calling By Reference**



* Also, the **argument** must be passed with ***ref***.

**Outting Argument**



* ***out*** is used for **assigning** variable & **operation** with it, **even if** it is assigned outside.

**Method Overloading**

* **Method overloading:** Different methods with ***same******name***.
* Difference is indicated by **data type** of input.
* Must be declared **outside** ***Main*** method & must be ***static***.

**Facts**

* Preventing a recursion to execute at a certain point is called **exit condition**.
* The process of creating an object is called **instantiation**.
* **object** is a keyword, be careful with it.
* **Members** of class are by **default *private***.

**Facts**

* A **constructor** must be **compulsorily *public***, with **no data type**.
* **Constructor** and **destructor** are **methods**.
* No need to declare destructors as ***public***.
* Properties are used to **read & write** private members.

**Properties**



* We write ***get* & *set*** this way for **read & write** mode respectively.
* ***value*** is a **keyword** referring to **value** we have assigned to the **property**.



**Custom Logic For Setting Value**



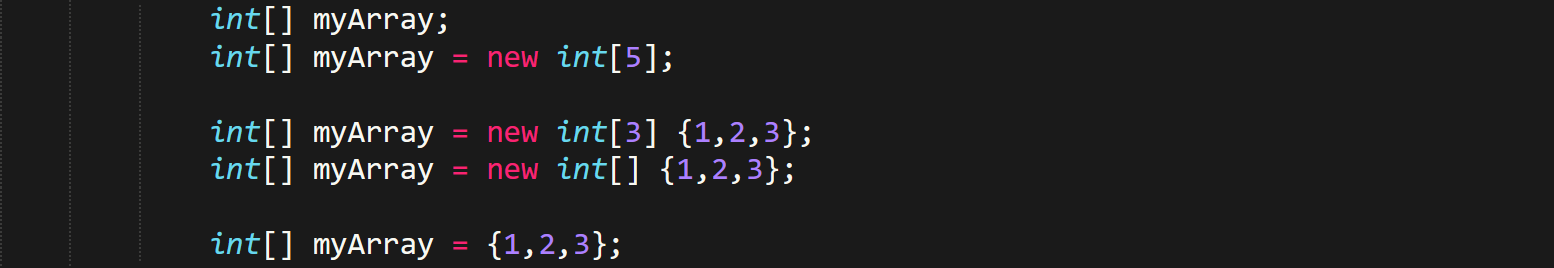
**Property Without Reference Value**



* Now member Name can be assigned a value.

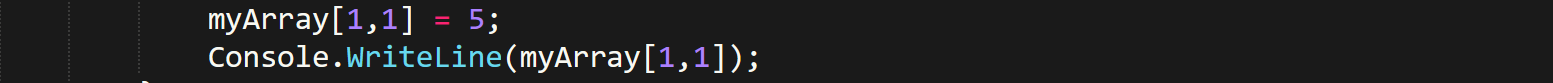
**Array**

* Array is a class.



**Facts**

* ***int[,] m = new int [3,4];*** (3\*4 elements = 4 elements in each 3 array)
* , = 2 dimension
* ,, = 3 dimension and so on.



* Thus the way to describe them is very **untraditional**.

**Jagged Arrays**

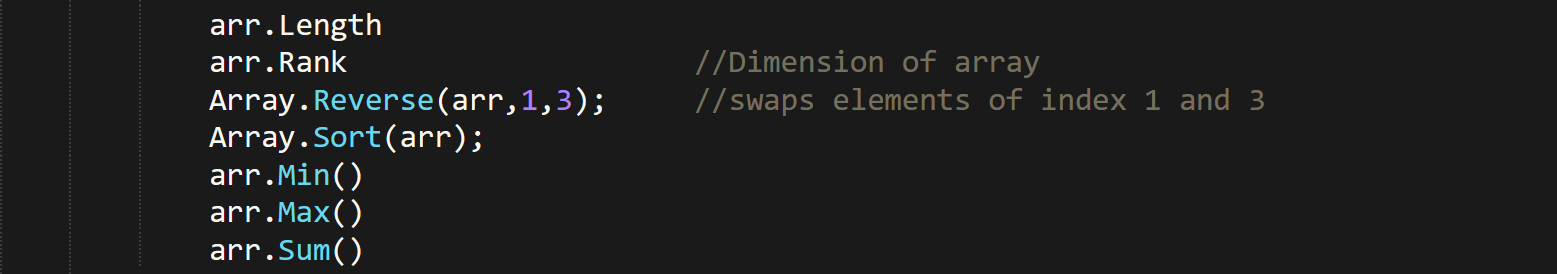


**OR**



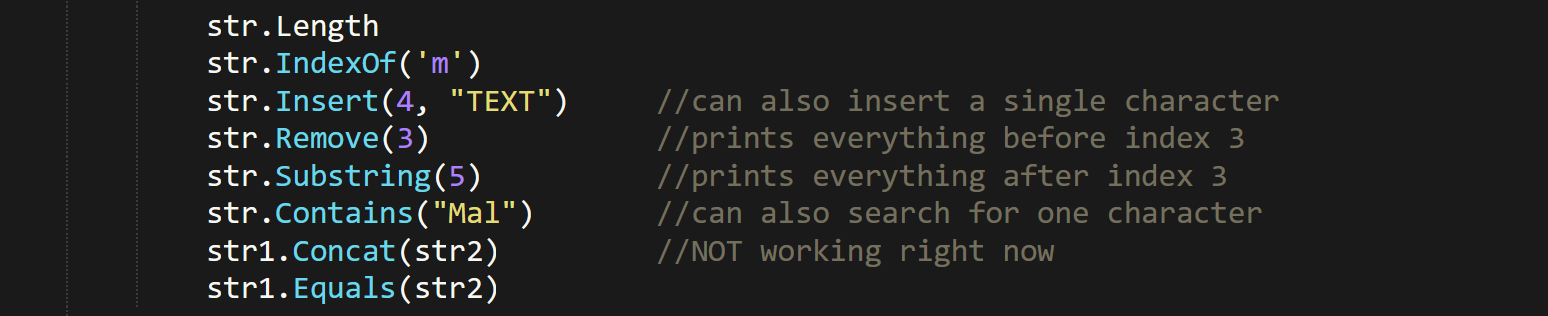
* It’s the **traditional** way to program **multi-dimensional arrays**.

**Array Properties & Methods**

****

**Working With String**

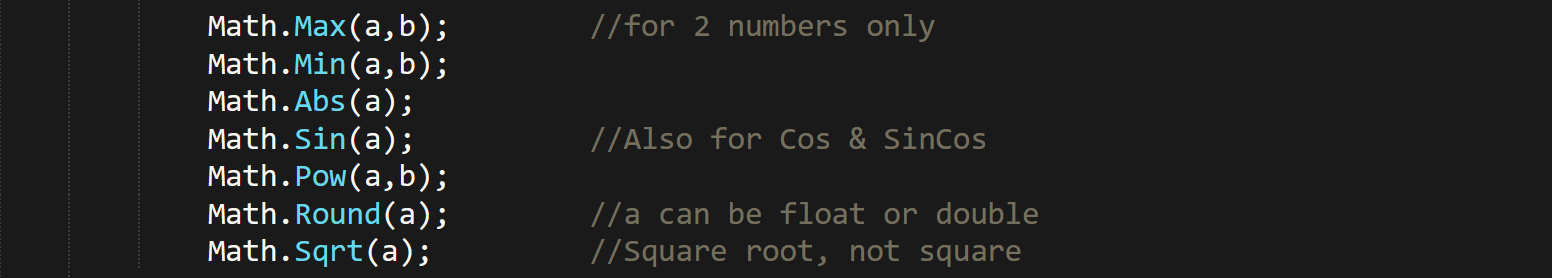
* **Strings are objects, not array of characters in C#.**

****

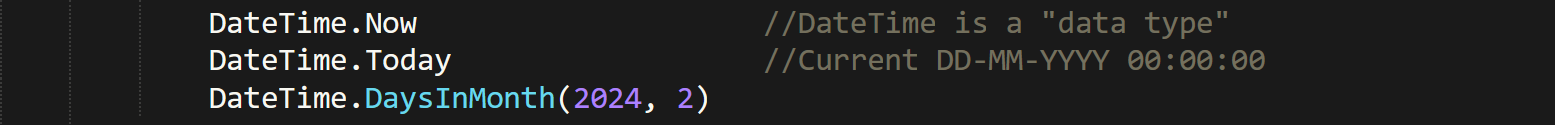
**Facts**

* We **can’t** pass any argument in **destructor** paranthesis, unlike **constructor**.
* We can have only one destructor, unlike constructor.
* **static** member/ method of a class can be accessed & modified without creating an object separately.
* **static class** contains only static members & methods.
* **Console**, **Math**, **Array**, **String**, **Date Time** etc, are **static** class.

**Math Methods:-**



**DateTime Methods:-**

****

**Facts:-**

* **this.name** kind of keywords used in classes refer to current instance/ object.
* **this** can also be passed as method arguments.

**Readonly Modifier:-**

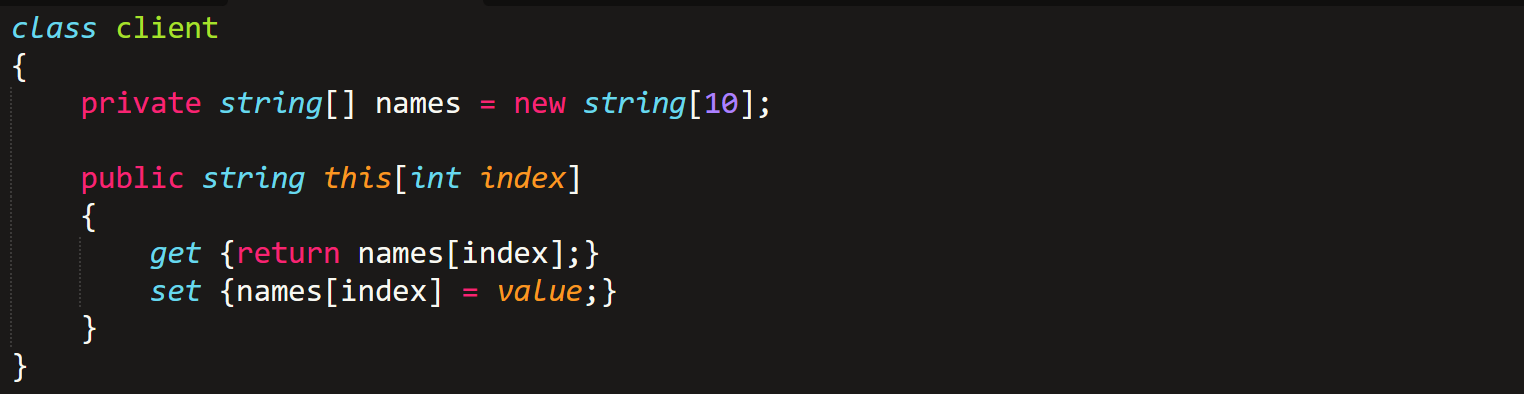


* **readonly** members can be only modified during assignation, or from a constructor.

**Facts:-**

* **const** variables has to be declared and assigned in the same line.
* Strings can be accessed and modified the same way as arrays.
* Like the for loop, accessing by index number etc.

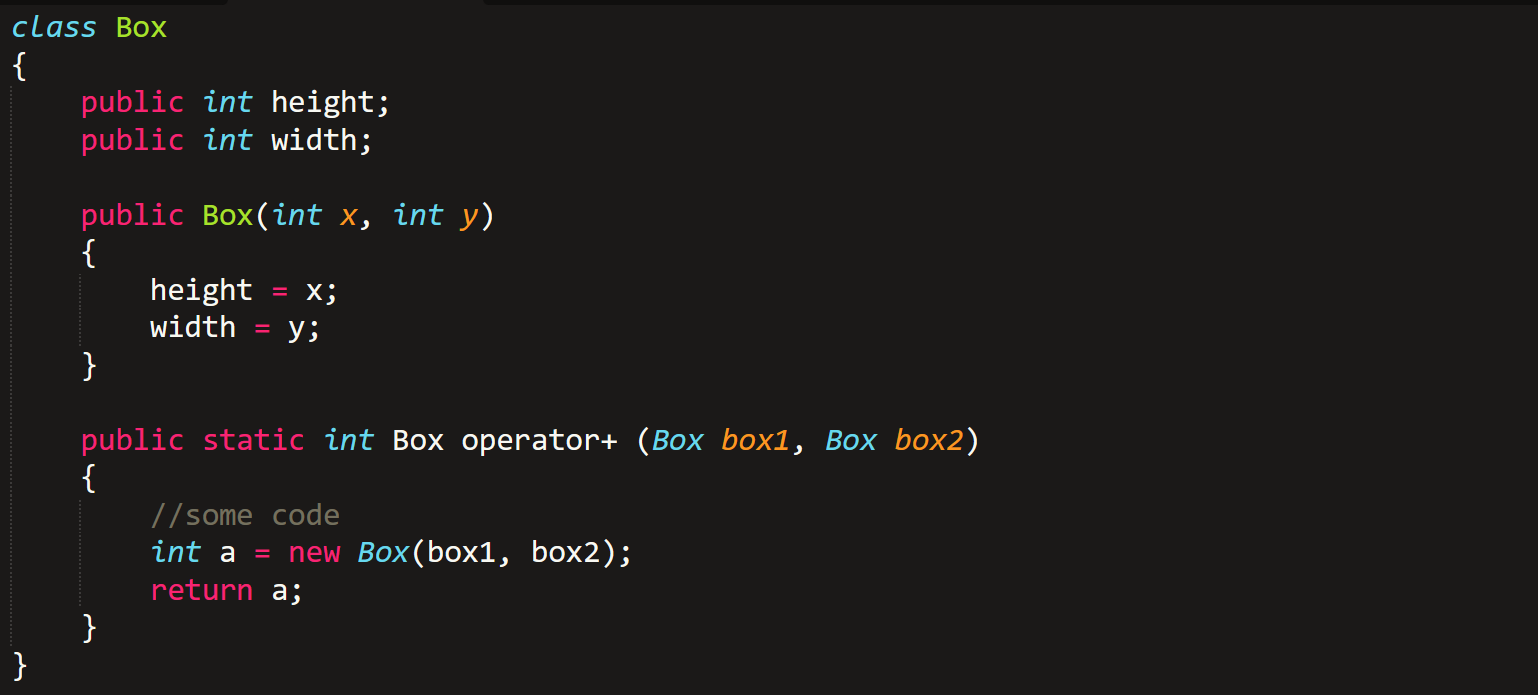
**get set Index:-**



* Basically returns character at index **“index”**.
* Same approach can be **done arrays** too.

**Operator Overloading:-**

* **+ , - , \* , / , % , ! , ++ ,-- , = , != , ==** can be overloaded only.
* Must be compulsorily static.



**Facts:-**

* Private members can be accessed & modified only from within base class.
* Protected members can be accessed & modified from within both base and inherited classes.
* Inheritence prevention – **“sealed class A { }”**.
* Prevent + Inherit = **sealed class B:A { }**
* Constructor & destructor are not inherited, but are called while code execution.

**Polymorphism:-**

* In base class, **public virtual void Draw();**
* In inherited class, **public override void Draw();**

**Abstract:-**

* To define an **abstract method**, it is **compulsory** to make the **class abstract**.
* And the same **override** keyword is used in derived classes.

**Interface**

* In **interface classes**, all members are abstract **without defining** them abstract.



**Fact:-**

* Nested classes are possible, like **dream inside dream in Nolan’s Inception**.
* Not using **“using” = System.Console.WriteLine();**

**Structure:-**

* All data types are structures.



To access method **“c”**,

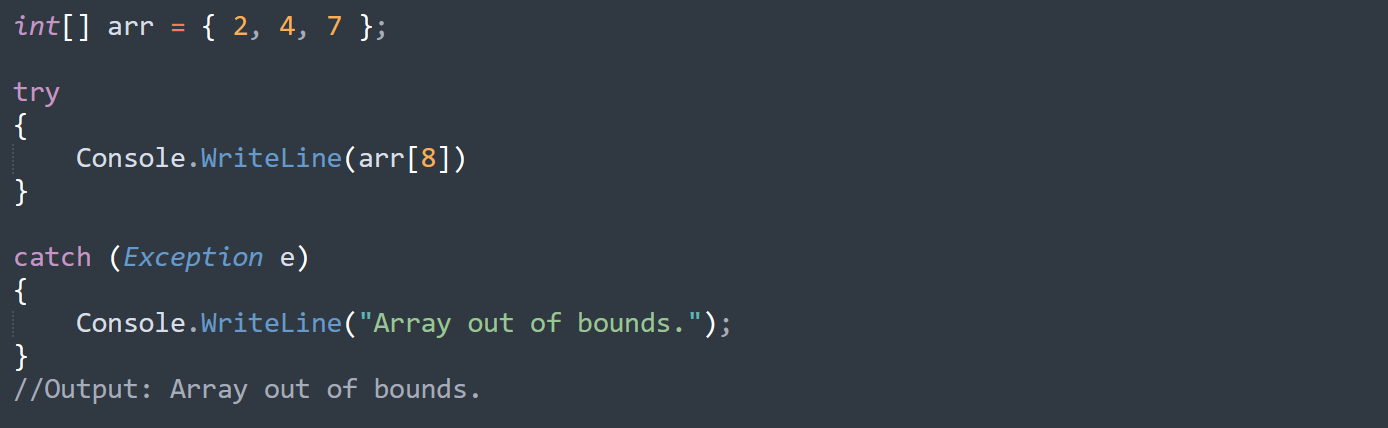


**Enums:-**

* Uses **enum** keyword.



**Exception Handling**



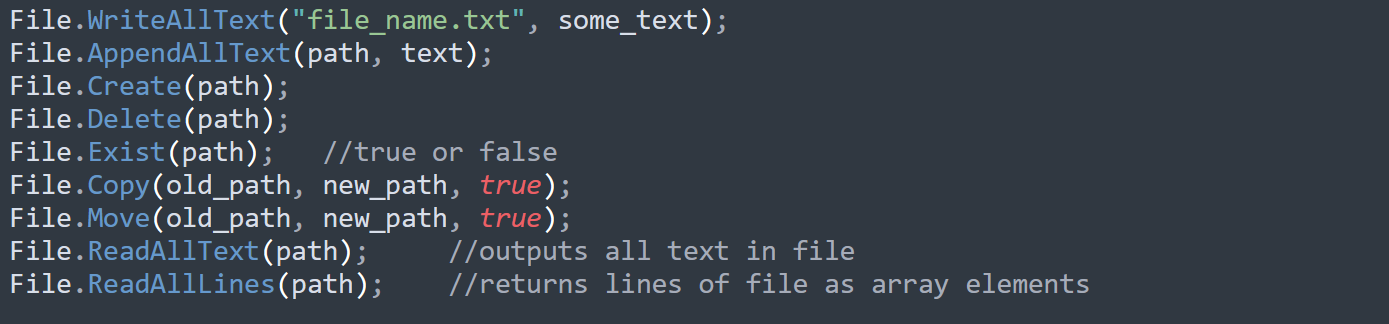
Types of exception keywords:-

* ***Exception* (All)**
* ***DivideByZeroException***
* ***FileNotFoundException***
* ***FormatException***
* ***IndexOutOfRangeException***
* ***InvalidOperationException***
* ***OutOfMemoryException***

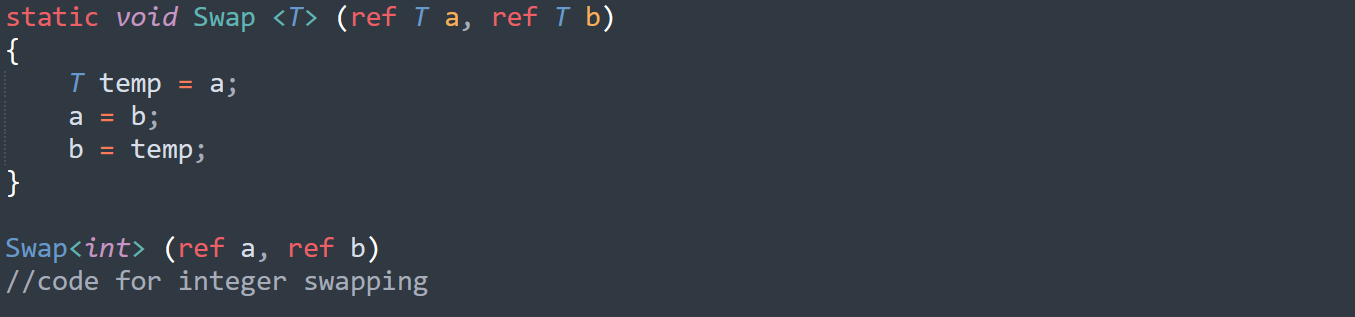
**e.Message** is the code to access the exception message in the catch block.

**File Handling**

using System.IO;

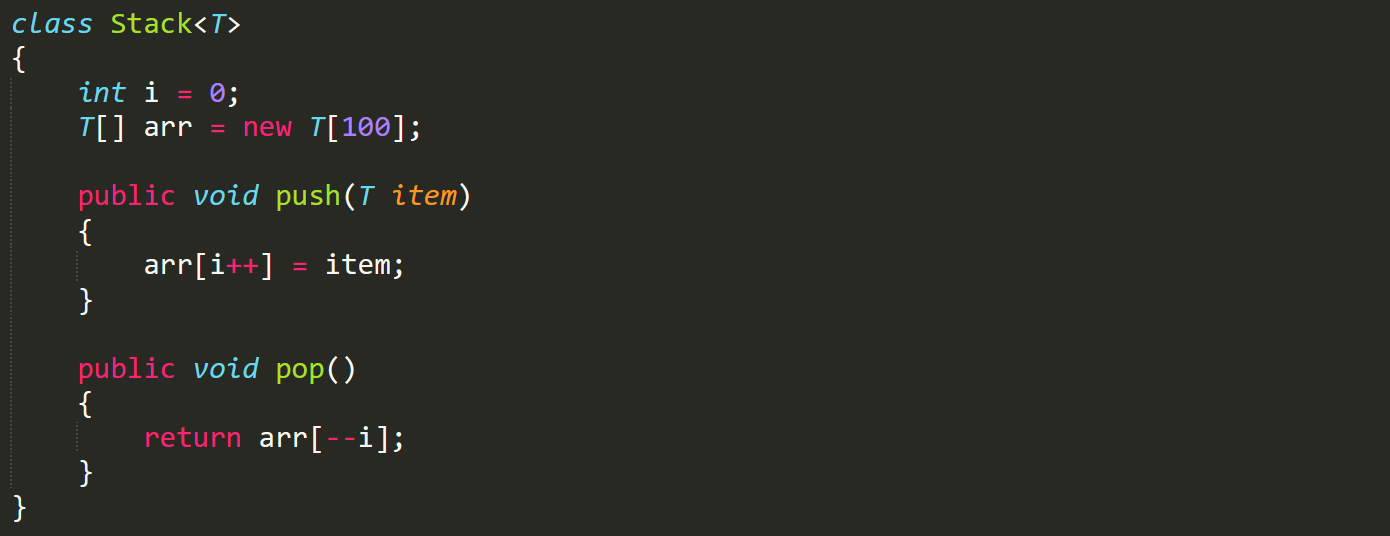


**Generics**



* It is necessary to use **static** and **ref**.
* Omitting writing type and continuing with **T** will make compiler decide on its own as per the type of data passed.

**Generic Classes**



* Look at this example of **Stack** above.

We access the elements in such array using get method, **“name.get(index)”**.

**Collections**

using System.Collections.Generic;

1. List <T>
2. Dictionary <T Key, T Value>
3. StoredList <T Key, T Value>
4. Stack <T>
5. Queue <T>
6. Hashset <T>

* **Non-generic** collections are error prone and are thus not recommended.

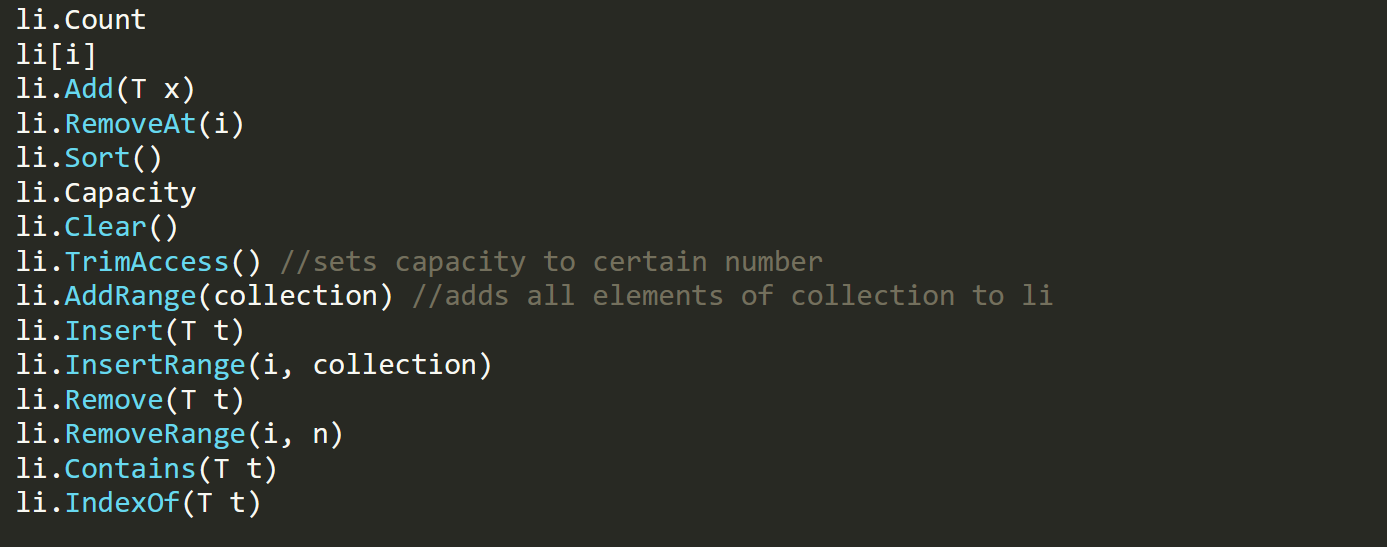
Some **non-generic** collections:-

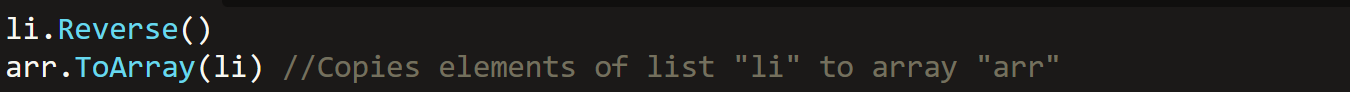
* ArrayList
* SortedList
* Stack
* Queue
* Hashtable
* BitArray

**List**

***List <data\_type> list\_name = new List <data\_type>();***

List methods and properties:-

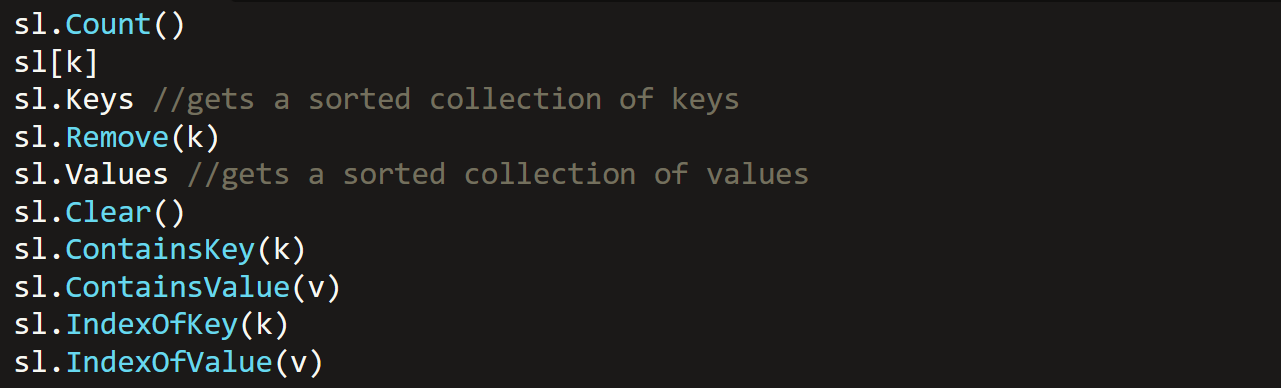




**Sorted List**

* It contains a key and a value, like dictionaries in Python.

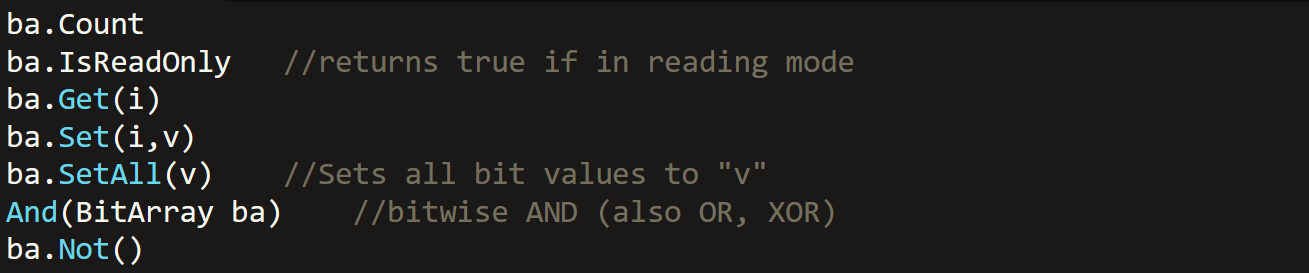
***SortedList <T,U> sl = new SortedList <T,U> ();***



**Bit Array**

* Array of bits.
* Element value can be either true or false.

***BitArray ba = new BitArray(5);***



**Stack & Queue**

***Stack <data\_type> stack\_name = new Stack <data\_type> ();***



***Queue <data\_type> queue\_name = new Queue <data\_type> ();***

\*Same as Stack but Push = Enqueue, Pop = Dequeue

**Dictionary & HashSet**

* Dictionary is similar to sorted list, but with by default sorted keys.

***Dictionary <data\_type1, data\_type2> dict\_name = new Dictionary < data\_type1, data\_type2> ();***

***HashSet <data\_type> hs\_name = new HashSet <data\_type> ();***

