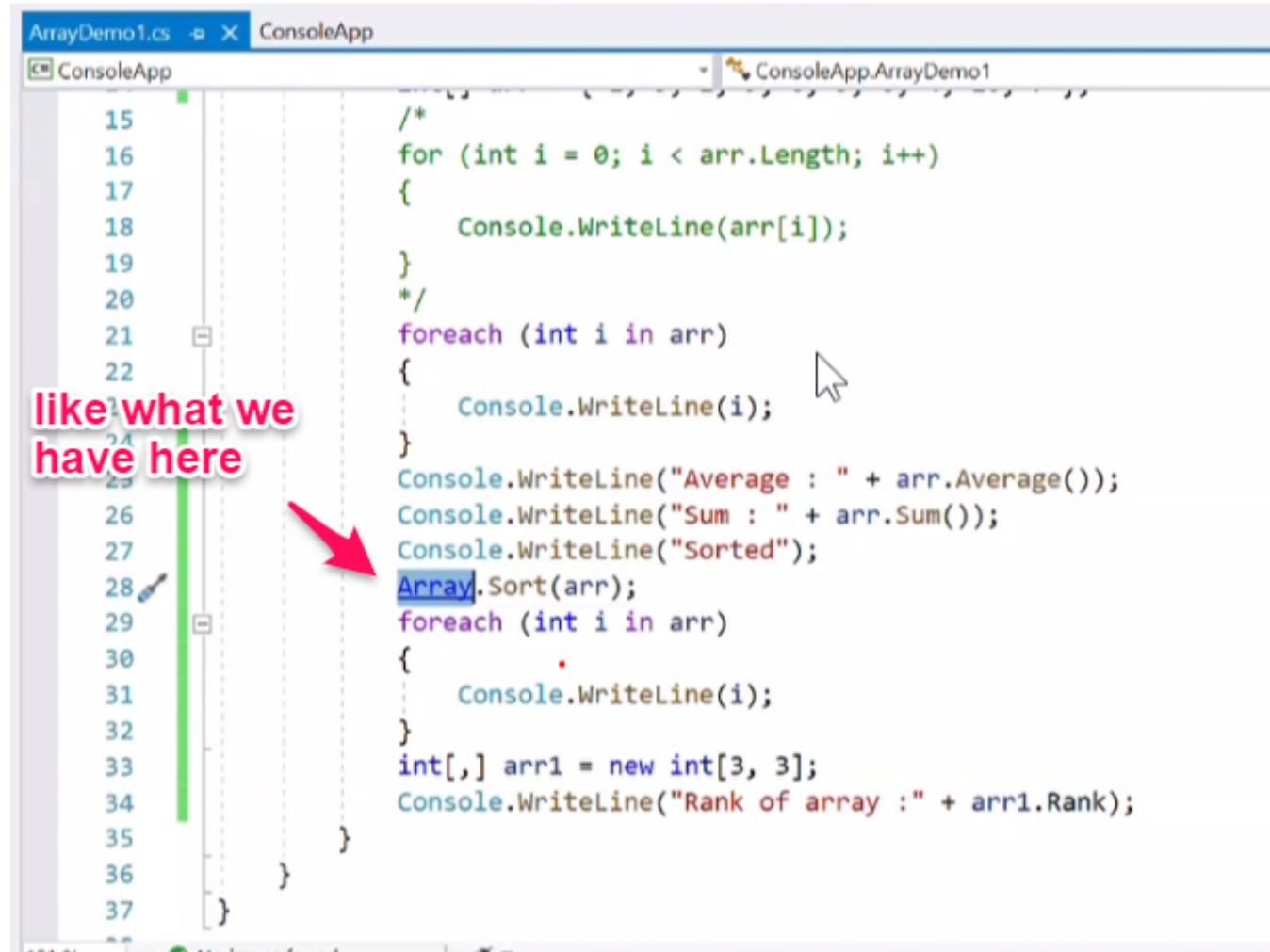


## C#-5

That's how like any class which inherits from IEnumerable



The screenshot shows a code editor window with the file "ArrayDemo1.cs" open under the project "ConsoleApp". The code is as follows:

```
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
```

/\*  
for (int i = 0; i < arr.Length; i++)  
{  
 Console.WriteLine(arr[i]);  
}  
\*/  
foreach (int i in arr)  
{  
 Console.WriteLine(i);  
}  
Console.WriteLine("Average : " + arr.Average());  
Console.WriteLine("Sum : " + arr.Sum());  
Console.WriteLine("Sorted");  
Array.Sort(arr);  
foreach (int i in arr)  
{  
 .  
 Console.WriteLine(i);  
}  
int[,] arr1 = new int[3, 3];  
Console.WriteLine("Rank of array :" + arr1.Rank);  
}

A red arrow points from the text "like what we have here" to the first "foreach" statement at line 21.

like what we have here

ArrayDemo1.cs      ConsoleApp

mscorlib

Assembly mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

System.Array

IsSynchronized

Properties    Changes    Notifications    Solution Explorer

```
1  Assembly mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
4
5  using ...
11
12  namespace System
13  {
14      public abstract class Array : ICloneable, IList, ICollection, IEnumerable, IStructuralComparable, IStructuralEquatable
15  {
16          public bool IsSynchronized { get; }
17          public bool IsReadOnly { get; }
18          public object SyncRoot { get; }
19          public int Rank { get; }
20          public long LongLength { get; }
21          public int Length { get; }
22          public bool IsFixedSize { get; }
23
24
25          public static ReadOnlyCollection<T> AsReadOnly<T>(T[] array);
26          public static int BinarySearch(Array array, object value);
27          public static int BinarySearch(Array array, object value, IComparer comparer);
28          public static int BinarySearch(Array array, int index, int length, object value, IComparer comparer);
29          public static int BinarySearch<T>(T[] array, T value);
30          public static int BinarySearch<T>(T[] array, T value, IComparer<T> comparer);
31          public static int BinarySearch<T>(T[] array, int index, int length, T value);
32          public static int BinarySearch<T>(T[] array, int index, int length, T value, IComparer<T> comparer);
33
34
35
36
37
38
39
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
20100
20101
20102
20103
20104
20105
20106
20107
20108
20109
20110
20111
20112
20113
20114
20115
20116
20117
20118
20119
20120
20121
20122
20123
20124
20125
20126
20127
20128
20129
20130
20131
20132
20133
20134
20135
20136
20137
20138
20139
20140
20141
20142
20143
20144
20145
20146
20147
20148
20149
20150
20151
20152
20153
20154
20155
20156
20157
20158
20159
20160
20161
20162
20163
20164
20165
20166
20167
20168
20169
20170
20171
20172
20173
20174
20175
20176
20177
20178
20179
20180
20181
20182
20183
20184
20185
20186
20187
20188
20189
20190
20191
20192
20193
20194
20195
20196
20197
20198
20199
201200
201201
201202
201203
201204
201205
201206
201207
201208
201209
201210
201211
201212
201213
201214
201215
201216
201217
201218
201219
201220
201221
201222
201223
201224
201225
201226
201227
201228
201229
201230
201231
201232
201233
201234
201235
201236
201237
201238
201239
201240
201241
201242
201243
201244
201245
201246
201247
201248
201249
201250
201251
201252
201253
201254
201255
201256
201257
201258
201259
201260
201261
201262
201263
201264
201265
201266
201267
201268
201269
201270
201271
201272
201273
201274
201275
201276
201277
201278
201279
201280
201281
201282
201283
201284
201285
201286
201287
201288
201289
201290
201291
201292
201293
201294
201295
201296
201297
201298
201299
2012000
2012001
2012002
2012003
2012004
2012005
2012006
2012007
2012008
2012009
20120010
20120011
20120012
20120013
20120014
20120015
20120016
20120017
20120018
20120019
20120020
20120021
20120022
20120023
20120024
20120025
20120026
20120027
20120028
20120029
20120030
20120031
20120032
20120033
20120034
20120035
20120036
20120037
20120038
20120039
20120040
20120041
20120042
20120043
20120044
20120045
20120046
20120047
20120048
20120049
20120050
20120051
20120052
20120053
20120054
20120055
20120056
20120057
20120058
20120059
20120060
20120061
20120062
20120063
20120064
20120065
20120066
20120067
20120068
20120069
20120070
20120071
20120072
20120073
20120074
20120075
20120076
20120077
20120078
20120079
20120080
20120081
20120082
20120083
20120084
20120085
20120086
20120087
20120088
20120089
20120090
20120091
20120092
20120093
20120094
20120095
20120096
20120097
20120098
20120099
201200100
201200101
201200102
201200103
201200104
201200105
201200106
201200107
201200108
201200109
201200110
201200111
201200112
201200113
201200114
201200115
201200116
201200117
201200118
201200119
201200120
201200121
201200122
201200123
201200124
201200125
201200126
201200127
201200128
201200129
201200130
201200131
201200132
201200133
201200134
201200135
201200136
201200137
201200138
201200139
201200140
201200141
201200142
201200143
201200144
201200145
201200146
201200147
201200148
201200149
201200150
201200151
201200152
201200153
201200154
201200155
201200156
201200157
201200158
201200159
201200160
201200161
201200162
201200163
201200164
201200165
201200166
201200167
201200168
201200169
201200170
201200171
201200172
201200173
201200174
201200175
201200176
201200177
201200178
201200179
201200180
201200181
201200182
201200183
201200184
201200185
201200186
201200187
201200188
201200189
201200190
201200191
201200192
201200193
201200194
201200195
201200196
201200197
201200198
201200199
201200200
201200201
201200202
201200203
201200204
201200205
201200206
201200207
201200208
201200209
201200210
201200211
201200212
201200213
201200214
201200215
201200216
201200217
201200218
201200219
201200220
201200221
201200222
201200223
201200224
201200225
201200226
201200227
201200228
201200229
201200230
201200231
201200232
201200233
201200234
201200235
201200236
201200237
201200238
201200239
201200240
201200241
201200242
201200243
201200244
201200245
201200246
201200247
201200248
201200249
201200250
201200251
201200252
201200253
201200254
201200255
201200256
201200257
201200258
201200259
201200260
201200261
201200262
201200263
201200264
201200265
201200266
201200267
201200268
201200269
201200270
201200271
201200272
201200273
201200274
201200275
201200276
201200277
201200278
201200279
201200280
201200281
201200282
201200283
201200284
201200285
201200286
201200287
201200288
201200289
201200290
201200291
201200292
201200293
201200294
201200295
201200296
201200297
201200298
201200299
201200300
201200301
201200302
201200303
201200304
201200305
201200306
201200307
201200308
201200309
201200310
201200311
201200312
201200313
201200314
201200315
201200316
201200317
201200318
201200319
201200320
201200321
201200322
201200323
201200324
201200325
201200326
201200327
201200328
201200329
201200330
201200331
201200332
201200333
201200334
201200335
201200336
201200337
201200338
201200339
201200340
201200341
201200342
201200343
201200344
201200345
201200346
201200347
201200348
201200349
201200350
201200351
201200352
201200353
201200354
201200355
201200356
201200357
201200358
201200359
201200360
201200361
201200362
201200363
201200364
201200365
201200366
201200367
201200368
201200369
201200370
201200371
201200372
201200373
201200374
201200375
201200376
201200377
201200378
201200379
201200380
201200381
201200382
201200383
201200384
201200385
201200386
201200387
201200388
201200389
201200390
201200391
201200392
201200393
201200394
201200395
201200396
201200397
201200398
201200399
201200400
201200401
201200402
201200403
201200404
201200405
201200406
201200407
201200408
201200409
201200410
201200411
201200412
201200413
201200414
201200415
201200416
201200417
201200418
201200419
201200420
201200421
201200422
201200423
201200424
201200425
201200426
201200427
201200428
201200429
201200430
201200431
201200432
201200433
201200434
201200435
201200436
201200437
201200438
201200439
201200440
201200441
201200442
201200443
201200444
201200445
201200446
201200447
201200448
201200449
201200450
201200451
201200452
201200453
201200454
201200455
201200456
201200457
201200458
201200459
201200460
201200461
201200462
201200463
201200464
201200465
201200466
201200467
201200468
201200469
201200470
201200471
201200472
201200473
201200474
201200475
201200476
201200477
201200478
201200479
201200480
201200481
201200482
201200483
201200484
201200485
201200486
201200487
201200488
201200489
201200490
201200491
201200492
201200493
201200494
201200495
201200496
201200497
201200498
201200499
201200500
201200501
201200502
201200503
201200504
201200505
201200506
201200507
201200508
201200509
201200510
201200511
201200512
201200513
201200514
201200515
201200516
201200517
201200518
201200519
201200520
201200521
201200522
201200523
201200524
201200525
201200526
201200527
201200528
201200529
201200530
201200531
201200532
201200533
201200534
201200535
201200536
201200537
201200538
201200539
201200540
201200541
201200542
201200543
201200544
201200545
201200546
201200547
201200548
201200549
201200550
201200551
201200552
201200553
201200554
201200555
201200556
201200557
201200558
201200559
201200560
201200561
201200562
201200563
201200564
201200565
201200566
201200567
201200568
201200569
201200570
201200571
201200572
201200573
201200574
201200575
201200576
201200577
201200578
201200579
201200580
201200581
201200582
201200583
201200584
201200585
201200586
201200587
201200588
201200589
201200590
201200591
201200592
201200593
201200594
201200595
201200596
201200597
201200598
201200599
201200600
201200601
201200602
201200603
201200604
201200605
201200606
201200607
201200608
201200609
201200610
201200611
201200612
201200613
201200614
201200615
201200616
201200617
201200618
201200619
201200620
201200621
201200622
201200623
201200624
201200625
201200626
201200627
201200628
201200629
201200630
201200631
201200632
201200633
201200634
201200635
201200636
201200637
201200638
201200639
201200640
201200641
201200642
201200643
201200644
201200645
201200646
201200647
201200648
201200649
201200650
201200651
201200652
201200653
201200654
201200655
201200656
201200657
201200658
201200659
201200660
201200661
201200662
201200663
201200664
201200665
201200666
201200667
201200668
201200669
201200670
201200671
201200672
201200673
201200674
201200675
201200676
201200677
201200678
201200679
201200680
201200681
201200682
201200683
201200684
201200685
201200686
201200687
201200688
201200689
201200690
201200691
201200692
201200693
201200694
201200695
201200696
201200697
201200698
201200699
201200700
201200701
201200702
201200703
201200704
201200705
201200706
201200707
201200708
201200709
201200710
201200711
201200712
201200713
201200714
201200715
201200716
201200717
201200718
201200719
201200720
201200721
201200722
201200723
201200724
201200725
201200726
201200727
201200728
201200729
201200730
201200731
201200732
201200733
201200734
201200735
201200736
201200737
201200738
201200739
201200740
201200741
201200742
201200743
201200744
201200745
201200746
201200747
201200748
201200749
201200750
201200751
201200752
201200753
201200754
201200755
201200756
201200757
201200758
201200759
201200760
201200761
201200762
201200763
201200764
201200765
201200766
201200767
201200768
201200769
201200770
201200771
201200772
201200773
201200774
201200775
201200776
201200777
201200778
201200779
201200780
201200781
201200782
201200783
201200784
201200785
201200786
201200787
201200788
201200789
201200790
201200791
201200792
201200793
201200794
201200795
201200796
201200797
201200798
201200799
201200800
201200801
201200802
201200803
201200804
201200805
201200806
201200807
201200808
201200809
201200810
201200811
201200812
201200813
201200814
201200815
201200816
201200817
201200818
201200819
201200820
201200821
201200822
201200823
201200824
201200825
201200826
201200827
201200828
201200829
20120
```

The screenshot shows a Microsoft Visual Studio code editor window. The title bar says "ArrayDemo1.cs" and "ConsoleApp". The code editor displays the following C# code:

```
15     /*
16      for (int i = 0; i < arr.Length; i++)
17      {
18          Console.WriteLine(arr[i]);
19      }
20  */
21  foreach (int i in arr)
22  {
23      Console.WriteLine(i);
24  }
25  Console.WriteLine("Average : " + arr.Average());
26  Console.WriteLine("Sum : " + arr.Sum());
27  Console.WriteLine("Sorted");
28  Array.Sort(arr);
29  foreach (int i in arr) ←
30  {
31      Console.WriteLine(i);
32  }
33  int[,] arr1 = new int[3, 3];
34  Console.WriteLine("Rank of array :" + arr1.Rank);
35
36
37 }
```

The line "foreach (int i in arr)" is highlighted with a red rectangle and has a red arrow pointing to it from the explanatory text on the right.

So when you say int i in arr  
Internally it will call those methods

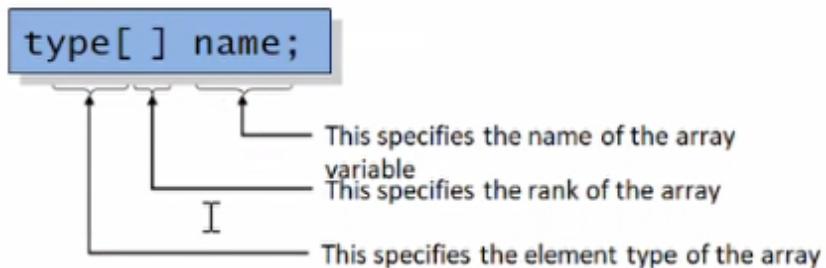
this i will call .current value  
in for loop we say i++  
so here internally it calls --> movenext() method

you can read more about that

like when ever you want to know more about internal methods, you just have to hold ctrl key and click on that

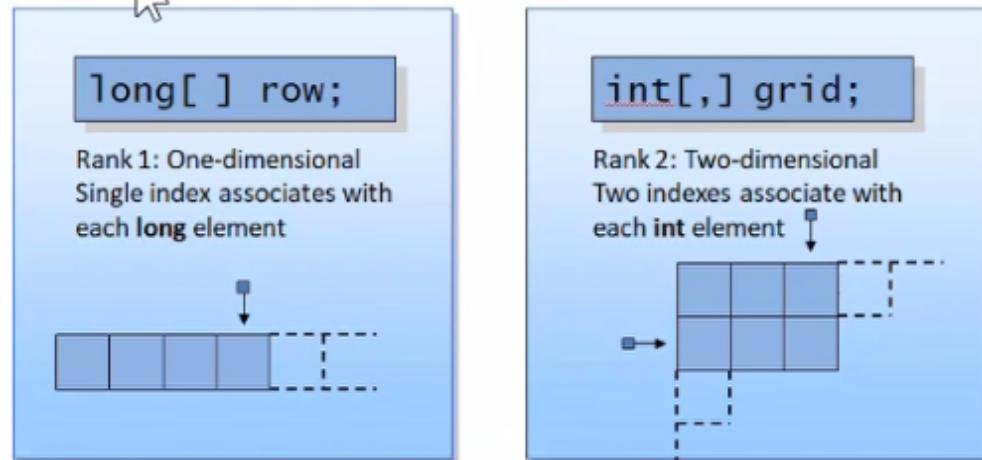
## Array Notation in C#

- You declare an array variable by specifying:
  - The element type of the array
  - The rank of the array
  - The name of the variable



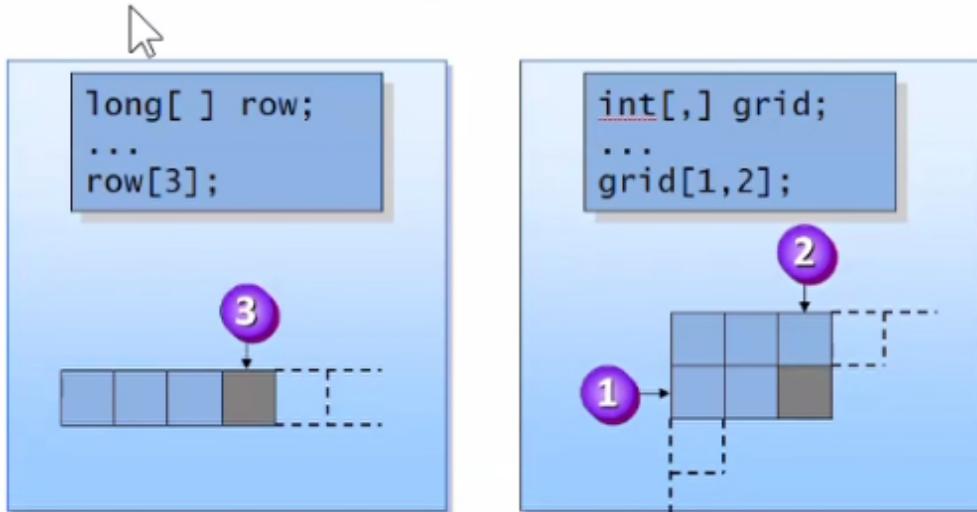
## Array Rank

- Rank is also known as the array dimension
- The number of indexes associated with each element



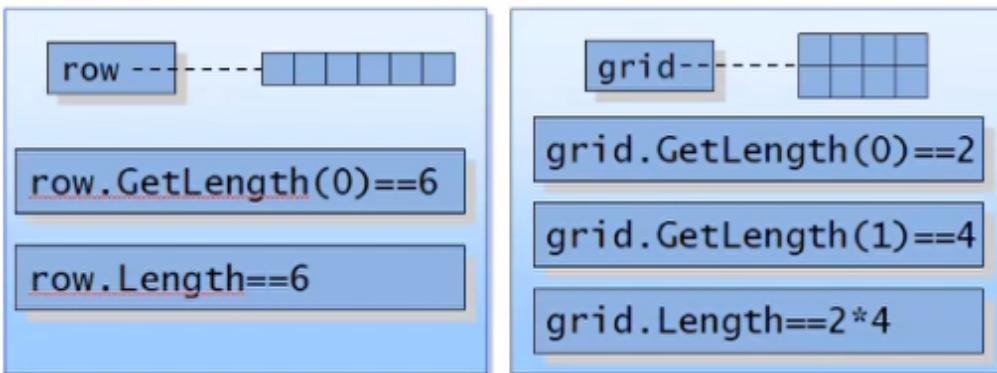
## Accessing Array Elements

- Supply an integer index for each rank
  - Indexes are zero-based



## Checking Array Bounds

- All array access attempts are bounds checked
  - A bad index throws an `IndexOutOfRangeException`
  - Use the `Length` property and the `GetLength` method



A screenshot of the Microsoft Visual Studio IDE interface. The main window shows a code editor with a C# file named 'ArrayDemo1.cs' open. The code implements a console application to demonstrate array operations. The code includes initializing an array with values 1 through 10, printing each value to the console, outputting the value at the 5th position, calculating and printing the average and sum of the array, sorting the array, and then printing each sorted value.

```
10
11 static void Main()
12 {
13     //int[] arr = new int[5];
14     int[] arr = { 1, 5, 2, 9, 6, 3, 8, 4, 10, 7 };
15     /*
16     for (int i = 0; i < arr.Length; i++)
17     {
18         Console.WriteLine(arr[i]);
19     }
19 */
21     Console.WriteLine('Value at 5th position is :" + arr[5]);
22     foreach (int i in arr)
23     {
24         Console.WriteLine(i);
25     }
26     Console.WriteLine("Average : " + arr.Average());
27     Console.WriteLine("Sum : " + arr.Sum());
28     Console.WriteLine("Sorted");
29     Array.Sort(arr);
30     foreach (int i in arr)
31     {
```

C:\WINDOWS\system32\cmd.exe

Value at 5th position is :3

1

5

2

9

6

3

8

4

10

7

Average : 5.5

Sum : 55

Sorted

1

2

3

4

5

6

7

8

9

10

Rank of array :2

Press any key to continue . . .

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar includes 'File', 'Edit', 'View', 'Git', 'Project', 'Build', 'Debug', 'Test', 'Analyze', 'Tools', 'Extensions', 'Window', 'Help', and 'Search (Ctrl+Q)'. Below the title bar are standard toolbar icons. The main window displays a C# file named 'ArrayDemo1.cs' under the project 'ConsoleApp'. The code editor shows the following Main() method:

```
10  {
11      static void Main()
12      {
13          //int[] arr = new int[5];
14          int[] arr = { 1, 5, 2, 9, 6, 3, 8, 4, 10, 7 };
15          /*
16          for (int i = 0; i < arr.Length; i++)
17          {
18              Console.WriteLine(arr[i]);
19          }
20          */
21          Console.WriteLine("Value at 5th position is :" + arr[10]);
22          foreach (int i in arr)
23          {
24              Console.WriteLine(i);
25          }
26          Console.WriteLine("Average : " + arr.Average());
27          Console.WriteLine("Sum : " + arr.Sum());
28          Console.WriteLine("Sorted");
29          Array.Sort(arr);
30          foreach (int i in arr)
31          {
```

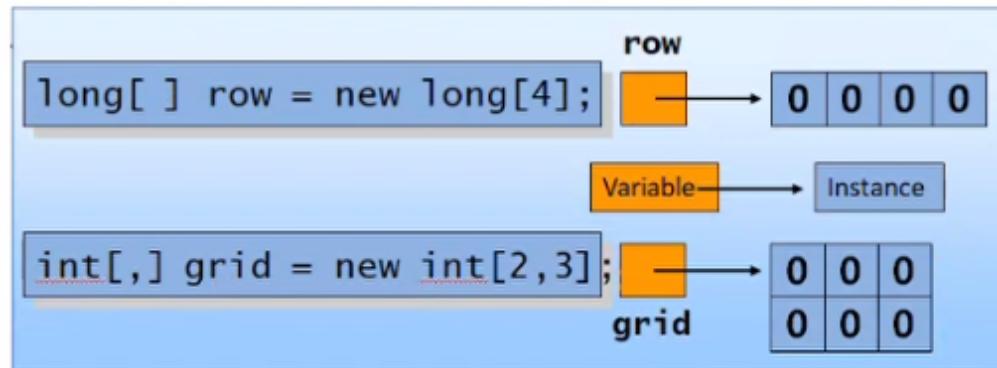
A mouse cursor is positioned over the array element reference 'arr[10]' on line 21. A tooltip or status message is visible near the cursor, indicating an error related to index 10.

Now when i want to access array element at index 10 i'll get indexoutofrange exception

```
Select C:\WINDOWS\system32\cmd.exe
Unhandled Exception: System.IndexOutOfRangeException: Index was outside the bounds of the array.
  at ConsoleApp.ArrayDemo1.Main() in C:\Shashi\Training\Clients\2021\Mphasis\Oct_Batch\Datas\ConsoleApp\ConsoleApp\ConsoleApp\ArrayDemo1.cs:line 21
```

## Creating Array Instances

- Declaring an array variable does not create an array!
  - You must use **new** to explicitly create the array instance



What do you mean declaring an array variable does not create an array

The screenshot shows a Microsoft Visual Studio IDE window with the following details:

- Title Bar:** ArrayDemo1.cs - ConsoleApp
- Code Editor:** The file is named `ConsoleApp`. The code defines a class `ArrayDemo1` with a static void `Main()` method. A line `int[] arr2;` is highlighted with a red box.
- Annotations:**
  - A pink callout points to the `arr2` declaration with the text: "look at this i get error it says unassigned local variable of arr2".
  - A pink callout points to the `arr2` reference in the `Rank` property with the text: "so int[] arr2 is just a declaration".
  - A pink callout points to the `arr2` reference in the `Value at 5th position` line with the text: "you have not created a integer array array got created only when you initialize them or you say = new int[5](5 is the range of array)]".
- Tooltips:** A tooltip for `arr2` shows "(local variable) int[] arr2".
- Errors:** A tooltip for the `Rank` property shows "CS0165: Use of unassigned local variable 'arr2'".
- Output Tab:** Shows "121 %", "1 0", and navigation icons.
- Bottom Navigation:** Output, Package Manager Console, Error List, Immediate Window.

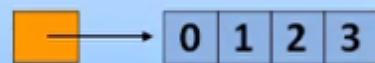
## Initializing Array Elements

- The elements of an array can be explicitly initialized
  - You can use a convenient shorthand

```
long[ ] row = new long[4] {0, 1, 2, 3};
```

```
long[ ] row = {0, 1, 2, 3};
```

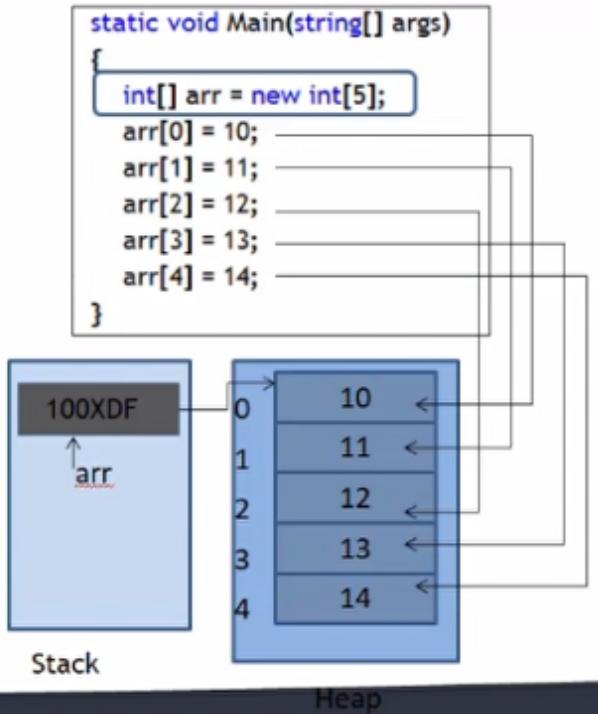
↑  
Equivalent



row

## Single dimensional Array

- Array containing a single row of values
- Example:
  - `int[] numbers = new int[5] {1, 2, 3, 4, 5};`
  - `string[] names = new string[3] {"Matt", "Joanne", "Robert"};`



## Initializing Multidimensional Array Elements

- Array containing multiple rows with same number of items for each row.
- You can also initialize multidimensional array elements
  - All elements must be specified

```
int[,] grid = {  
    {5, 4, 3},  
    {2, 1, 0}  
};
```

Implicitly a new int[2,3]  
array  
grid

5	4	3
2	1	0

```
int[,] grid = {  
    {5, 4, 3},  
    {2, 1 }  
};
```

x      you cannot do like this, this will give you error

Why this is a error ?

because of memory allocation, you are not allocating the memory  
when you say 3 x3 so you should have 9 elements

0	0	1	2	1	0	1	2
0	1	2	3	0	1	2	3
1	4	5	6	1	4	5	6
2	7	8		2	7	8	9

1	0	1	2	1	0	1	2
0	1	2	3	0	1	2	3
1	4	5	6	1	4	5	6
2	7	8	9	2	7	8	9

## Creating a Computed Size Array

- The array size does not need to be a compile-time constant
  - Any valid integer expression will work
  - Accessing elements is equally fast in all cases
  - Array size specified by compile-time integer constant:

```
long[ ] row = new long[4];
```

- Array size specified by run-time integer value:

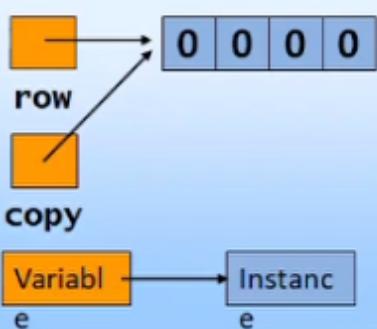
```
string s = Console.ReadLine();
int size = int.Parse(s);
long[ ] row = new long[size];
```

you can compute the array size based on particular array , may be you can ask user to give a particular array length and based on that you can pass the particular size as a parameter to your array that is possible

## Copying Array Variables

- Copying an array variable copies the array variable only
  - It does not copy the array instance
  - Two array variables can refer to the same array instance

```
long[ ] row = new long[4];
long[ ] copy = row;
...
row[0]++;
long value = copy[0];
Console.WriteLine(value);
```



ArrayDemo1.cs X ConsoleApp

ConsoleApp

```
7  namespace ConsoleApp
8  {
9      class ArrayDemo1
10     {
11         static void Main()
12         {
13             //int[] arr = new int[5];
14             int[] arr = { 1, 5, 2, 9, 6, 3, 8, 4, 10, 7 };
15             /*
16             for (int i = 0; i < arr.Length; i++)
17             {
18                 Console.WriteLine(arr[i]);
19             }
19             */
20
21             foreach (int i in arr)
22             {
23                 Console.WriteLine(i);
24             }
25             Console.WriteLine("Average : " + arr.Average());
26             Console.WriteLine("Sum : " + arr.Sum());
27             Console.WriteLine("Sorted");
28         }
    
```

121 % No issues found

Output Package Manager Console Error List Immediate Window

Item(s) Saved

Now original array is having this data 1 5 2 9 6 3 8 4 10 7

```
Console.WriteLine("Sorted");
Array.Sort(arr);
foreach (int i in arr)
{
    Console.WriteLine(i);
}
int[,] arr1 = new int[3, 3];
Console.WriteLine("Rank of array :" + arr1.Rank);

//copy array element
int[] newArray = new int[arr.Length];
arr.CopyTo(newArray, 0);
Console.WriteLine("New array value");
foreach (var i in newArray)
{
    Console.WriteLine(i); | I
}
```

so what will be the data  
this array will gone to have  
after executing this particular program

Will it be same data in same order?

-----No we will get sorted array because we are sorting it here

ψ Select C:\WINDOWS\system32\cmd.exe

```
8  
4  
10  
7
```

```
Average : 5.5  
Sum : 55  
Sorted
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

```
Rank of array :2
```

```
New array value
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

```
Press any key to continue . . .
```

The screenshot shows the Microsoft Visual Studio IDE with the code editor open. The file is named 'ArrayDemo1.cs' and contains a class named 'ConsoleApp'. The code demonstrates various array operations:

```
21 //Console.WriteLine("Value at 5th position is :" + arr[10]);
22 foreach (int i in arr)
23 {
24     Console.WriteLine(i);
25 }
26 Console.WriteLine("Average : " + arr.Average());
27 Console.WriteLine("Sum : " + arr.Sum());
28 Console.WriteLine("Sorted");
29 int[] newArray = new int[arr.Length];
30 arr.CopyTo(newArray, 0);
31 Array.Sort(newArray);
32 foreach (int i in newArray)
33 {
34     Console.WriteLine(i);
35 }
36 int[,] arr1 = new int[3, 3];
37 Console.WriteLine("Rank of array :" + arr1.Rank);
38
39 //copy array element
40
41 Console.WriteLine("New array value");
42 foreach (var i in newArray)
43 {
```

Annotations and arrows highlight specific parts of the code:

- A red box surrounds the line `int[] newArray = new int[arr.Length];`. A red arrow points from this box to the explanatory text.
- A red arrow points from the line `arr.CopyTo(newArray, 0);` to the explanatory text.
- The explanatory text is:

if you want to keep original array  
as such then  
then you have to write this particular line here  
before sorting  
now here you pass the temp array  
and fetch the data from the temp array
- A green box surrounds the line `Console.WriteLine("Sorted");`. A red arrow points from this box to the explanatory text.
- The explanatory text is:

So original array remains as such  
we are not touching the  
original array
- A red arrow points from the line `Array.Sort(newArray);` to the explanatory text.
- The explanatory text is:

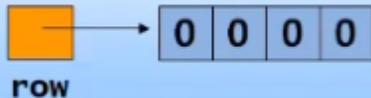
what ever the manipulations  
you are doing is always on  
the temp array the newArray

thats how your original data will not get spoiled

## Array Class Properties

- Some Important Properties:
  - Length and Rank

```
long[ ] row = new long[4];
```



```
int[,] grid = new int[2,3];
```



row.Rank

1

row.Length

4

grid.Rank

2

grid.Length

6



## Array Class Methods

- Commonly used methods
  - **Clear:** Sets a range of elements to zero or **null**
  - **Clone:** Creates a copy of the array
  - **Copy:** It is a static method. Copies the Array with data to another array.
  - **CopyTo:** Copies the Data only to another Array.
  - **CreateInstance:** It is a static method Allows to create an Instance of a new Array. This function is used to create Arrays using System.Array class. (To Create Arrays Dynamically).
  - **GetLength:** Returns the length of a given dimension
  - **IndexOf:** Returns the index of the first occurrence of a value
  - **Sort:** Sorts the elements in an array of rank 1



Clone - creates a new array copy will copy the statical data

Copy - will copy the statical data

DO reading on shallow copy and deep copy

## Returning Arrays from Methods

- You can declare methods to return arrays

```
class Example {  
    static void Main( ) {  
        int[ ] array = CreateArray(42);  
        ...  
    }  
    static int[ ] CreateArray(int size) {  
        int[ ] created = new int[size];  
        return created;  
    }  
}
```

## Passing Arrays as Parameters

- An array parameter is a copy of the array variable
  - Not a copy of the array instance

```
class Example2 {  
    static void Main( ) {  
        int[ ] arg = {10, 9, 8, 7};  
        Method(arg);  
        System.Console.WriteLine(arg[0]);  
    }  
    static void Method(int[ ] parameter) {  
        parameter[0]++;  
    }  
}
```

This method will modify  
the original array  
instance created in Main

we'll gone to push this printing code outside this method ,  
now when you have some piece of code which you want to push outside the method , so  
instead of cutting from here and pasting it,  
you can always use refactoring

A screenshot of the Microsoft Visual Studio IDE interface. The main window shows a code editor with the file `ArrayDemo1.cs` open, containing C# code for a console application. A context menu is displayed over the code at line 26, with the first item, "Quick Actions and Refactorings...", highlighted.

The code in `ArrayDemo1.cs`:

```
24         Console.WriteLine(i);
25     }
26     Console.WriteLine("Average : " + arr.Average());
27     Console.WriteLine();
28     int[] newArray = new int[10];
29     arr.CopyTo(newArray);
30     Array.Sort(newArray);
31     foreach (int i in newArray)
32     {
33         Console.WriteLine(i);
34     }
35     int[,] arr1 = new int[2, 5];
36     Console.WriteLine();
37     //copy array elements
38     Console.WriteLine();
39     foreach (var i in arr)
40     {
41         Console.WriteLine(i);
42     }
43 }
```

The context menu items include:

- Quick Actions and Refactorings... (highlighted)
- Rename... (Ctrl+R, Ctrl+R)
- Remove and Sort Usings (Ctrl+R, Ctrl+G)
- View Code (F7)
- Peek Definition (Alt+F12)
- Go To Definition (F12)
- Go To Base (Alt+Home)
- Go To Implementation (Ctrl+F12)
- Find All References (Shift+F12)
- View Call Hierarchy (Ctrl+K, Ctrl+T)
- Create Unit Tests
- Breakpoint
- Run To Cursor (Ctrl+F10)
- Execute in Interactive (Ctrl+E, Ctrl+E)
- Snippet
- Cut (Ctrl+X)
- Copy (Ctrl+C)
- Paste (Ctrl+V)
- Annotation
- Outlining

select code which you want to move from one method to another method ,right click on it , click on quick action and refactoring .., click on extract method

The screenshot shows a Microsoft Visual Studio IDE window. The title bar displays "ArrayDemo1.cs" and "ConsoleApp". The code editor contains C# code related to arrays and console output. A context menu is open at line 35, with the "Extract method" option highlighted. Other options visible in the menu include "Extract local function", "Convert to 'for'", and "Use implicit type". The code itself includes methods like Average(), Sum(), Sort(), CopyTo(), and WriteLine().

```
24     Console.WriteLine(i);
25 }
26 Console.WriteLine("Average : " + arr.Average());
27 Console.WriteLine("Sum : " + arr.Sum());
28 Console.WriteLine("Sorted");
29 int[] newArray = new int[arr.Length];
30 arr.CopyTo(newArray, 0);
31 Array.Sort(newArray);
32 foreach (int i in newArray)
33 {
34     Console.WriteLine(i);
35 }
36 int[,] arr1 = new int[3, 3];
37 Console.WriteLine("Rank of array :" + arr1.Rank);
38 //copy array element
39
40 Console.WriteLine("New array value");
41 foreach (var i in newArray)
42 {
43     Console.WriteLine(i);
44 }
45 }
```

you can see it in preview

A screenshot of the Microsoft Visual Studio IDE interface. The main window displays a C# file named 'ConsoleApp.cs' with the following code:

```
        Console.WriteLine(i);
    }
}

static void Main()
{
    //int[] arr = new int[5];
Lines 31 to 36
    Array.Sort(newArray);
    foreach (int i in newArray)
    {
        Console.WriteLine(i);
    }
    NewMethod(newArray);
    int[,] arr1 = new int[3, 3];
Lines 39 to 41
    //copy array element

    Console.WriteLine("New array value");
Lines 46 to 47
}

private static void NewMethod(int[] newArray)
{
    foreach (int i in newArray)
    {
        Console.WriteLine(i);
    }
}
```

The code editor shows several regions highlighted with different colors: green for the first section, red for the second, light green for the third, and pink for the fourth. A context menu is open at the bottom of the first green-highlighted section, containing the following items:

- Extract method
- Extract local function
- Convert to 'for'
- Use implicit type

The status bar at the bottom right shows the current position as Ln: 35 Ch: 5 Col: 14 TABS.

just click on this and a new method is created for you

A screenshot of the Microsoft Visual Studio IDE interface. The main window shows the code editor with a C# file named 'ArrayDemo1.cs' open. The code demonstrates how to create a 2D array, print its rank, and copy its elements to a new array using a foreach loop. A context menu is open over the word 'NewMethod', with the title 'Rename: NewMethod'. The menu includes options to 'Include comments', 'Include strings', and 'Preview changes'. It also states that renaming will update 2 references. The status bar at the bottom shows '121 %' zoom, 'No issues found', 'Ln: 32 Ch: 13', and tabs for 'Output', 'Package Manager Console', 'Error List ...', and 'Immediate Window'. The taskbar at the bottom features icons for the Start button, a search bar, and various pinned applications.

```
int[,] arr1 = new int[3, 3];
Console.WriteLine("Rank of array :" + arr1.Rank);

//copy array element

Console.WriteLine("New array value");
foreach (var i in newArray)
{
    Console.WriteLine(i);
}

private static void NewMethod(int[] newArray)
{
    foreach (int i in newArray)
    {
        Console.WriteLine(i);
    }
}
```

arrayDemo1.cs ➔ X ConsoleApp

## ConsoleApp

ConsoleApp

now you  
just have to  
give name  
of this new  
method

<sup>29</sup>  
<sup>39</sup>  
we named it  
Display  
Done hogaya

```
Console.WriteLine("Sum : " + arr.Sum());
Console.WriteLine("Sorted");
int[] newArray = new int[arr.Length];
arr.CopyTo(newArray, 0);
Array.Sort(newArray);
Display(newArray);
int[,] arr1 = new int[3, 3];
Console.WriteLine("Rank of array :" + arr1.Rank);
```

//copy array element

```
Console.WriteLine("New array value");
foreach (var i in newArray)
{
    Console.WriteLine(i);
}
```

**ye bhi nahi chiye yaha bhi hum display ko  
call kar lenge**

```
1 reference
private static void Display(int[] newArray)
{
    foreach (int i in newArray)
```

121 %   No issues found

```
rayDemo1.cs  X ConsoleApp
ConsoleApp
ConsoleApp.ArrayDemo1

27     Console.WriteLine("Sum : " + arr.Sum());
28     Console.WriteLine("Sorted");
29     int[] newArray = new int[arr.Length];
30     arr.CopyTo(newArray, 0);
31     Array.Sort(newArray);
32
33     int[,] arr1 = new int[3, 3];
34     Console.WriteLine("Rank of array :" + arr1.Rank);
35
36     //copy array element
37
38     Console.WriteLine("New array value");
39     Display(newArray);
40 }
41
42 1 reference
43 private static void Display(int[] newArray)
44 {
45     foreach (int i in newArray)
46     {
47         Console.WriteLine(i);
48     }
49 }

No issues found
```

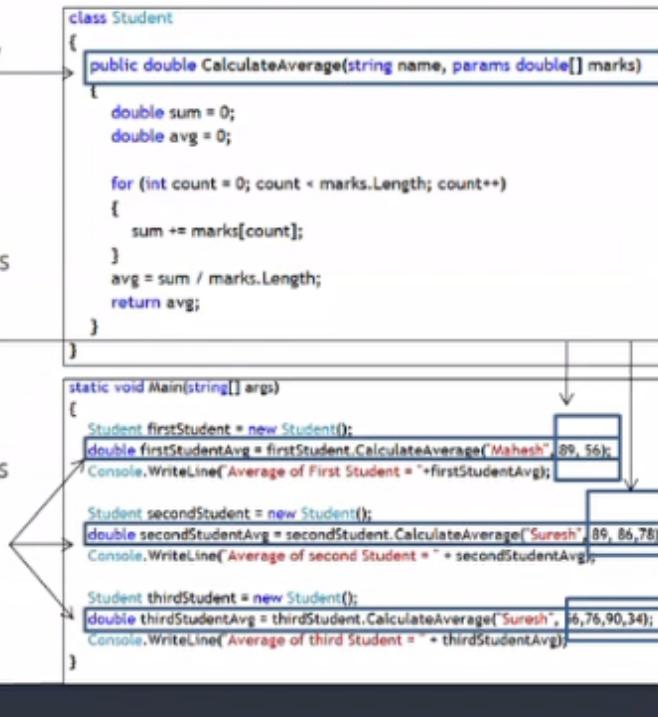
Here you can see i'm passing an array

and here we are taking an array

This is how we can pass array from  
one place to another place

## 'params' keyword and Param Array

- 'params' keyword is used with an array declaration provided the array is part of the method argument
- This enables the array to accept values directly that is passed to the method directly when the method is called
- While passing parameter during a method call, if the caller is not sure about how many parameters to pass then the method should possess a param array to accept unknown number of arguments



Imagine i have written a particular program and what this program does is, it will return a integer value , minimum how many numbers you can add----alteast 2

ParamsDemo.cs   X   ConsoleApp

ConsoleApp

ConsoleApp.ParamsDemo

```
7  namespace ConsoleApp
8  {
9      class ParamsDemo
10     {
11         static void Main()
12         {
13             int result = AddNumbers(10, 29);
14             Console.WriteLine("Sum of numbers is :" + result);
15         }
16
17         private static int AddNumbers(int no1, int no2)
18         {
19             //logic to add nos. and return
20             return (no1 + no2);
21         }
22     }
23 }
24
```

C:\WINDOWS\system32\cmd.exe

```
Sum of numbers is :39
Press any key to continue . . .
```

So this code is working perfectly fine, now the business will come and say this is fabulous but I want to add more numbers, I want to add 3 numbers

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar displays "ParamsDemo.cs" and "ConsoleApp". The code editor window contains the following C# code:

```
ParamsDemo.cs  ▾ X ConsoleApp
ConsoleApp
7  namespace ConsoleApp
8  {
9      class ParamsDemo
10     {
11         static void Main()
12         {
13             int result = AddNumbers(10, 29);
14             Console.WriteLine("Sum of numbers is :" + result);
15             result = AddNumbers(10, 29, 30);
16             Console.WriteLine("Sum| of numbers is :" + result);
17         }
18
19         private static int AddNumbers(int no1, int no2)
20         {
21             //logic to add nos. and return
22             return (no1 + no2);
23         }
24     }
25 }
26
```

The code includes a Main() method that calls the AddNumbers() method twice. The first call uses two parameters (10, 29), and the second call uses three parameters (10, 29, 30). The output of the first call is displayed on the screen, while the output of the second call is highlighted with a red underline and a cursor, indicating it is currently being edited.

So how do we do this ?

```
ParamsDemo.cs  ✘ ConsoleApp
ConsoleApp
ConsoleApp.ParamsDemo

7  namespace ConsoleApp
8  {
9      class ParamsDemo
10     {
11         static void Main()
12         {
13             int result = AddNumbers(10, 29);
14             Console.WriteLine("Sum of numbers is :" + result);
15             result = AddNumbers(10, 29, 30);
16             Console.WriteLine("Sum of numbers is :" + result);
17         }
18     }
19     private static int AddNumbers(int no1, int no2)
20     {
21         //logic to add nos. and return
22         return (no1 + no2);
23     }
24 }
25
26
```

Now i cannot touch this code because these is already been implemented and already in production

A screenshot of the Visual Studio IDE showing a C# file named ParamsDemo.cs. The code defines a class ParamsDemo with a Main() method that calls two different AddNumbers() methods. The first AddNumbers() method takes three parameters and returns a result. The second AddNumbers() method takes two parameters and returns a result. A red box highlights the third parameter 'no3' in the first AddNumbers() method. A pink arrow points from this highlighted area to the text 'So I need to add one more number here'. The status bar at the bottom shows 'No issues found'.

```
ParamsDemo.cs*  X  ConsoleApp
ConsoleApp
ConsoleApp.ParamsDemo
7  namespace ConsoleApp
8  {
9      class ParamsDemo
10     {
11         static void Main()
12         {
13             int result = AddNumbers(10, 29);
14             Console.WriteLine("Sum of numbers is :" + result);
15             result = AddNumbers(10, 29, 30);
16             Console.WriteLine("Sum of numbers is :" + result);
17         }
18
19         private static int AddNumbers(int no1, int no2, int no3)
20         {
21             //logic to add nos. and return
22             return (no1 + no2 + no3);
23         }
24
25         private static int AddNumbers(int no1, int no2)
26         {
27             //logic to add nos. and return

```

So this is the concept of  
**METHOD OVERLOADING**

So I need to add one more number here

SO this is the concept of

# **overloading**

**you can have the same method name but  
with different parameter**

So, this is working fine now bussiness will say i want to add 4 numbers ,so you will do the same thing again

**So, looking at this kind of code, developers of .NET thought -----boss whenever you have this kind of situation, you just have to do a shortcut**

```
ParamsDemo.cs*  X  ConsoleApp
ConsoleApp  ConsoleApp.ParamsDemo

12
13     {
14         int result = AddNumbers(10, 29);
15         Console.WriteLine("Sum of numbers is :" + result);
16         result = AddNumbers(10, 29, 30);
17         Console.WriteLine("Sum of numbers is :" + result);
18         result = AddNumbers(10, 29, 30, 56);
19         Console.WriteLine("Sum of numbers is :" + result);
20     }
21
22     3 references
23     private static int AddNumbers(int[] nos)
24     {
25         //logic to add nos. and return
26         return (no1 + no2 + no3);
27     }
28
29     /*
30     private static int AddNumbers(int no1, int no2, int no3)
31     {
32         //logic to add nos. and return
33         return (no1 + no2 + no3);
34     }

121 %  6  0  ← → | ⌂ ▾
```

you just say, this is my array  
and i don't know how many  
number i'll get, now when i do this  
it expects to be an array  
whatever i'm getting  
but what they are passing is  
**just a numbers**

```
ParamsDemo.cs*  X  ConsoleApp
ConsoleApp
ConsoleApp.ParamsDemo
12
13
14
15
16
17
18
19
20
21 3 references
22 private static int AddNumbers(params int[] nos)
23 {
24     //logic to add nos. and return
25     return (no1 + no2 + no3);
26
27
28 /*
29 private static int AddNumbers(int no1, int no2, int no3)
30 {
31     //logic to add nos. and return
32     return (no1 + no2 + no3);
33 }
34
```

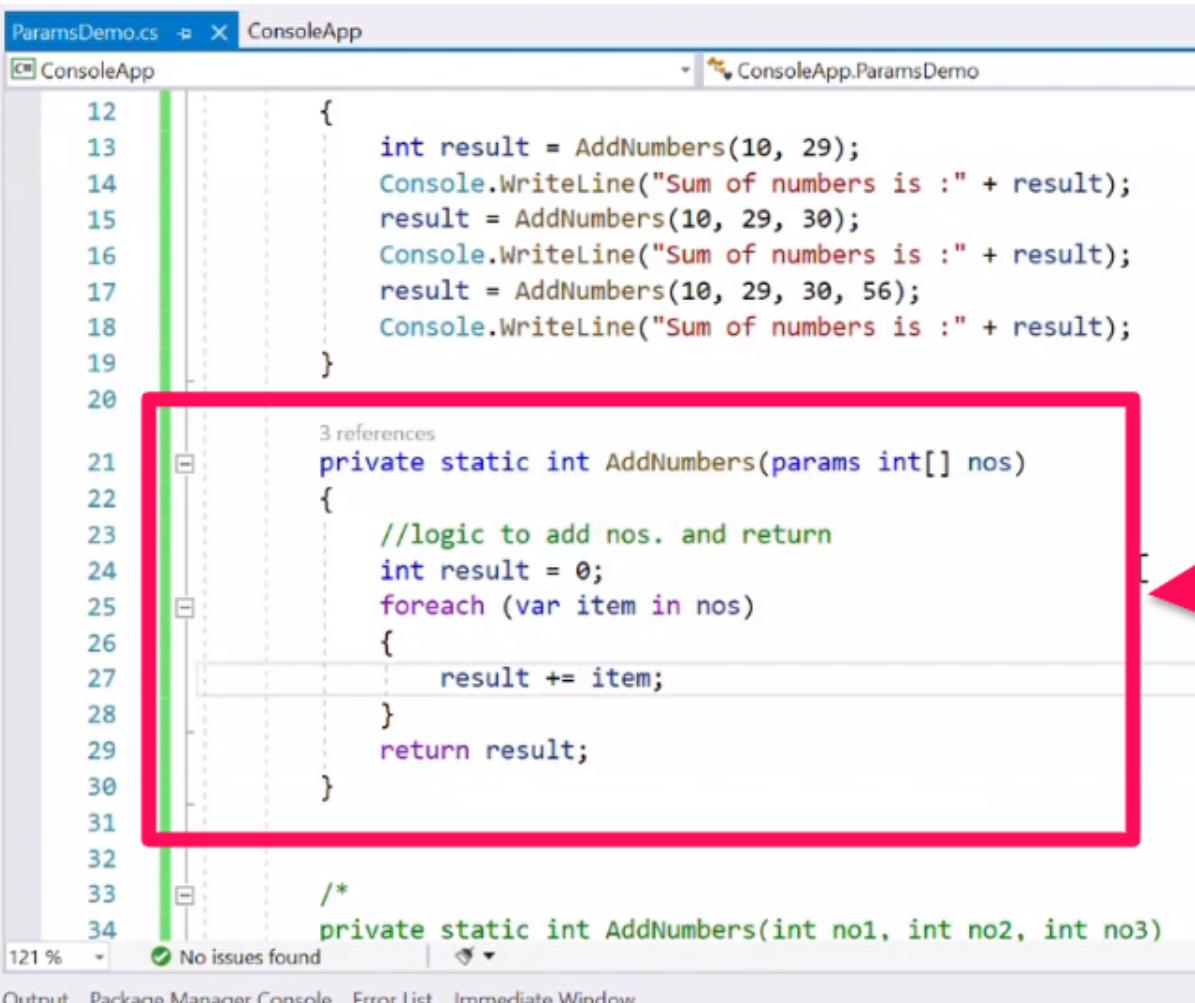
2

now look at this all the error  
what i has here  
is gone

1

Now what i do here is ,  
i'll say boss you pass whatever it is  
-----i will converts this to my array

So this will be my  
Parameterized Array



```
ParamsDemo.cs  X  ConsoleApp
ConsoleApp  ConsoleApp.ParamsDemo

12     {
13         int result = AddNumbers(10, 29);
14         Console.WriteLine("Sum of numbers is :" + result);
15         result = AddNumbers(10, 29, 30);
16         Console.WriteLine("Sum of numbers is :" + result);
17         result = AddNumbers(10, 29, 30, 56);
18         Console.WriteLine("Sum of numbers is :" + result);
19     }
20
21     3 references
22     private static int AddNumbers(params int[] nos)
23     {
24         //logic to add nos. and return
25         int result = 0;
26         foreach (var item in nos)
27         {
28             result += item;
29         }
30         return result;
31     }
32
33     /*
34      * 
35      private static int AddNumbers(int no1, int no2, int no3)
```

121 %    No issues found

Now how to you add the number  
and fetched the sum of it  
-----By using foreach loop

Now you add 4 number or 400 number no worries , Bussiness will happy you are also happy

ParamsDemo.cs   X   ConsoleApp

ConsoleApp

ConsoleApp.ParamsDemo

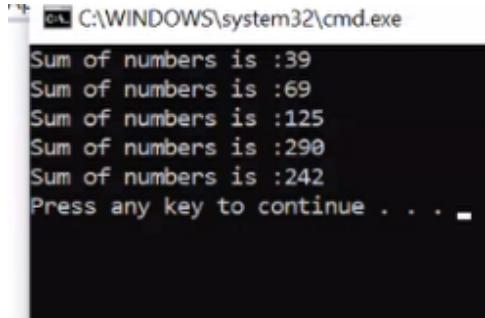
Main()

```
12     {
13         int result = AddNumbers(10, 29);
14         Console.WriteLine("Sum of numbers is :" + result);
15         result = AddNumbers(10, 29, 30);
16         Console.WriteLine("Sum of numbers is :" + result);
17         result = AddNumbers(10, 29, 30, 56);
18         Console.WriteLine("Sum of numbers is :" + result); I
19         result = AddNumbers(10, 29, 30, 56, 46, 59, 60);
20         Console.WriteLine("Sum of numbers is :" + result);
21         result = AddNumbers(10, 29, 30, 56, 56, 35, 3, 3, 4, 4, 4, 4, 4);
22         Console.WriteLine("Sum of numbers is :" + result);
23     }
24
25     private static int AddNumbers(params int[] nos)
26     {
27         //logic to add nos. and return
28         int result = 0;
29         foreach (var item in nos)
30         {
31             result += item;
32         }
33         return result;
34     }
```

121 %    No issues found

Output   Package Manager Console   Error List ...   Immediate Window

tem(s) Saved



```
C:\WINDOWS\system32\cmd.exe
Sum of numbers is :39
Sum of numbers is :69
Sum of numbers is :125
Sum of numbers is :290
Sum of numbers is :242
Press any key to continue . . .
```

Remember when you don't know how many argument user will pass and that should be of same type

```
result = AddNumbers(10, 29, 30, 56, "Shashi");
```

You cannot do like this, it should be of the same type, when it is of the same type you can always use **PARAMS**, So params are very powerfull

Have we done something like this in java ?

java also supports this, in java we will write **int... nos**

## Jagged Array

- It is an array of arrays
- It is an array containing multiple rows, but each with different number of elements
- DataTable class, used in ADO.NET contains a jagged array to store multiple records, where each record might have different number of attributes (columns)

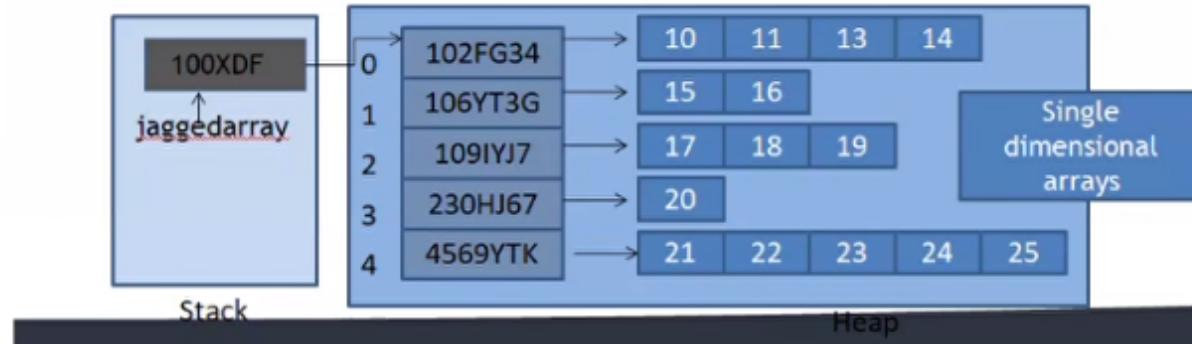
```
//ask the user to enter no of rows for the jagged array
Console.WriteLine("Enter no of rows for the jagged array: ");
int no_of_rows = Convert.ToInt32(Console.ReadLine());

//jagged array declaration with the specified no of rows
int[][] jaggedarray = new int[no_of_rows][];

for (int row = 0; row < jaggedarray.Length; row++)
{
    //ask user to enter no of elements in a row
    Console.WriteLine("Enter no of elements for jaggedarray[" + row + "]");
    int no_of_elements = Convert.ToInt32(Console.ReadLine());

    //create a single dimensional array with same number of elements
    int[] singlearray = new int[no_of_elements];

    //pass the reference of that single dimensional array to
    //a row of jagged array
    jaggedarray[row] = singlearray;
}
```



Note: Jagged array could be an array of single or multi dimensional arrays as well as even jagged arrays, too.

Suppose in array

in 1st row i want to store 1 2 3

in 2nd row i want to store 2 4 5 7

int 3rd row i want to store 1 2

Something like this

So, how is this possible ?

So , where you don't have the data it will add 0 automatically

```
1, 2, 3, 0  
2, 4, 5, 7  
1, 2, 0, 0
```

But i don't want to do it i want this 2 number to be store here and i want this 3 number to be stored here

```
1, 2, 3|  
2, 4, 5, 7  
1, 2
```

so then what i need to do, i need to plan my self

i write

int [ ] [ ] -----1st box i'll say how many rows i need, 2nd box i'll say how many columns i need

int [1] [3]-----1row with 3 columns

```
int[1][3] = 1, 2, 3  
int[1][4] = 2, 4, 5, 7  
int[1][2] = |1, 2
```

this is like 3 elements, 3 arrays i'm creating, but i don't want to create 3 array i want to handle them in one array because of the space complexity and all  
So, i know how many rows i need, i know that i need 3 rows, but i don't know how many columns i need So, i'm not bothered

```
int[3][]
```

## So, this becomes my **Jagged Array**

Remember jagged array are different they are not multidimensional array  
i'm not saying here

```
int[3,4]
```

i'm not doing this

```
int[3][4]
```

i'm giving 2 different boxes-----and this clearly says this is a array which will have a array inside it  
its like a array inside a array

**What will be the rank of this jagged array?** --remember rank is dimension so what will be the dimension of this array?

if i say **int [ ]**this is one dimension correct

when i put a comma **int [ , ]** this is two dimension correct

when i put one more comma **int [ , , ]** this is three dimension correct

when i put **int [ , , , ]** this is four dimension correct

if i put comma then only dimension will change it will be **2,2 3,3 2,3 .....**it will be like that so this will be a square matrix or a rectangular matrix where the number of rows and the number of columns are fixed

but this **int [ ]** is still single dimensional array the rank will be one right

**so for that i'm adding new array int [ ] [ ]** So, what will be the rank ----- **This is**

## **Single Dimension**

So, in case of jagged array

```
| int[1][3] = 1, 2, 3  
| int[1][4] = 2, 4, 5, 7  
| int[1][2] = 1, 2 |
```

No of Rows i know but column i don't, it can be 3 column, it can be 4 column, it can be 2 column

So, in this particular case what it does

to this  
particular  
Array

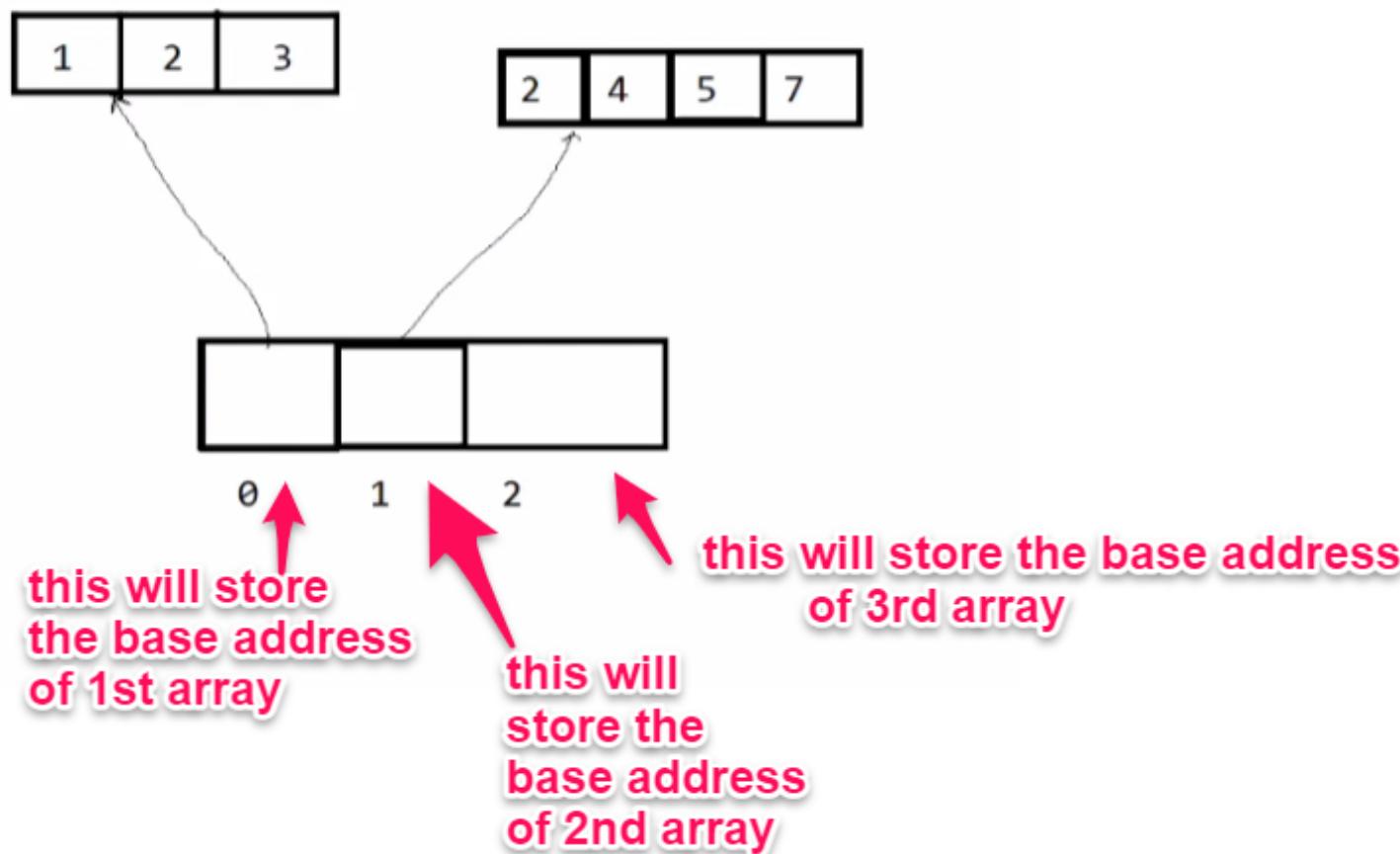
it will add  
this as  
Array

**It is like pointing to  
that particular array**

```
int[1][3] = 1, 2, 3  
int[1][4] = 2, 4, 5, 7  
int[1][2] = 1, 2
```



`int [3] []` will be look like this



The screenshot shows a code editor window with the file 'JadArrayDemo.cs' open. The code defines a class 'JadArrayDemo' with a static void Main() method. Inside Main(), a jagged array 'arr' is declared and initialized with three rows. The first row has 3 elements (1, 2, 3), the second has 4 elements (2, 4, 5, 7), and the third has 2 elements (1, 2). The code editor has syntax highlighting and a vertical yellow margin bar on the left. The status bar at the bottom shows '121 %' zoom and 'No issues found'.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp
8  {
9      class JadArrayDemo
10     {
11         static void Main()
12         {
13             int[][] arr = new int[3][];
14             arr[0] = new int[3] { 1, 2, 3 };
15             arr[1] = new int[4] { 2, 4, 5, 7 };
16             arr[2] = new int[2] { 1, 2 };
17         }
18     }
19 }
20
```

this is how we create jagged array

Now, how do you read the data from this array so, this is 2 different array -----array inside array

JadArrayDemo.cs X ConsoleApp\*

ConsoleApp

```
7  namespace ConsoleApp
8  {
9      class JadArrayDemo
10     {
11         static void Main()
12         {
13             int[][] arr = new int[3][];
14             arr[0] = new int[3] { 1, 2, 3 };
15             arr[1] = new int[4] { 2, 4, 5, 7 };
16             arr[2] = new int[2] { 1, 2 };
17
18             foreach (var outerArray in arr)
19             {
20                 foreach (var innerArray in outerArray)
21                 {
22                     Console.Write(innerArray + " ");
23                 }
24                 Console.WriteLine();
25             }
26         }
27     }
28 }
```

121 % ✓ No issues found

Output Package Manager Console Error List ... Immediate Window

```
C:\WINDOWS\system32\cmd.exe
1 2 3
2 4 5 7
1 2
Press any key to continue . . .
```

## Entering Items Into and Displaying Items From Jagged Array

```
for (int row = 0; row < jaggedarray.Length; row++)
{
    for (int column = 0; column < jaggedarray[row].Length;
        column++)
    {
        //ask the user to enter item in particular column of
        //particular row
        Console.Write("Enter item in jaggedarray["
            + row + "][" + column + "]: ");
        jaggedarray[row][column] = Convert.ToInt32(
            Console.ReadLine());
    }
    Console.WriteLine("\n");
}

for (int row = 0; row < jaggedarray.Length; row++)
{
    for (int column = 0; column < jaggedarray[row].Length;
        column++)
    {
        //display to the user an item from particular column from
        //particular row
        Console.Write("Item at jaggedarray["
            + row + "][" + column + "]: "
            + jaggedarray[row][column]);
    }
    Console.WriteLine("\n");
}
```

## Enumerations

- The `enum` keyword is used to declare an enumeration, a user defined type consisting of a set of named constants called enumerator list.
- Every enumeration member value has an underlying type, which can be any integral type numerical value such as `short`, `ushort`, `byte`, `sbyte`, `int`, `uint`, `long`, `ulong` but can't be of type `char`, `double`, `decimal`, `float`, `bool`, `string` etc. The default underlying data type of the enumeration member value is `int`.
- Enumeration is useful when you have to supply some predefined set of values, such as designation of employees in an organization or set of colors for the background of a Windows or Web Form

Suppose i am a particular programmer who takes a particular day of the week  
lets take a real time flight example  
if it is weekends then the flight charges will be little heigh  
but where as in week days it will be less  
and if it is a weekend + any festival it will more heigh

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar displays "EnumDemo.cs" and "ConsoleApp\*". The code editor window contains the following C# code:

```
10 0 references
11 class EnumDemo
12 {
13     0 references
14     private static double CalculateFlightCharge(string dow, double baseRate)
15     {
16         //logic
17         double amount = 0;
18         switch (dow)
19         {
20             case "Sun": amount = baseRate + (baseRate * 0.05); break; //5% on sunday
21             case "Sat": amount = baseRate + (baseRate * 0.07); break; //7% on satday
22             case "Mon":
23             case "Tue":
24             case "Wed":
25             case "Thur":
26             case "Fri":
27                 amount = baseRate - (baseRate * 0.025); break; //2.5% on Mon..Fri Discount
28         }
29     return amount;
30 }
31 }
```

The code implements a switch statement to calculate a flight charge based on the day of the week. It adds 5% for Sunday and 7% for Saturday, and subtracts 2.5% for Monday through Friday. The code editor shows syntax highlighting and line numbers. A green vertical bar on the left indicates code coverage or selection.

now we have a logic which we have already written ,  
now somebody will use our program  
we know that any code you write is not for you , mostly someone else will use it

EnumDemo.cs\* ▶ X ConsoleApp\*

ConsoleApp

ConsoleApp.EnumDemo

Main()

```
10 class EnumDemo
11 {
12     private static double CalculateFlightCharge(string dow, double baseRate)...
13
14     static void Main()
15     {
16         double amount = 12000;
17         double calAmount = CalculateFlightCharge("Sun", amount);
18         Console.WriteLine("On Sunday the charges are :" + amount + " Rate :" + calAmount);
19     }
20 }
21 
```

On Sunday the charges are :12000 Rate :12600

Press any key to continue . . .

what if it is wednesday

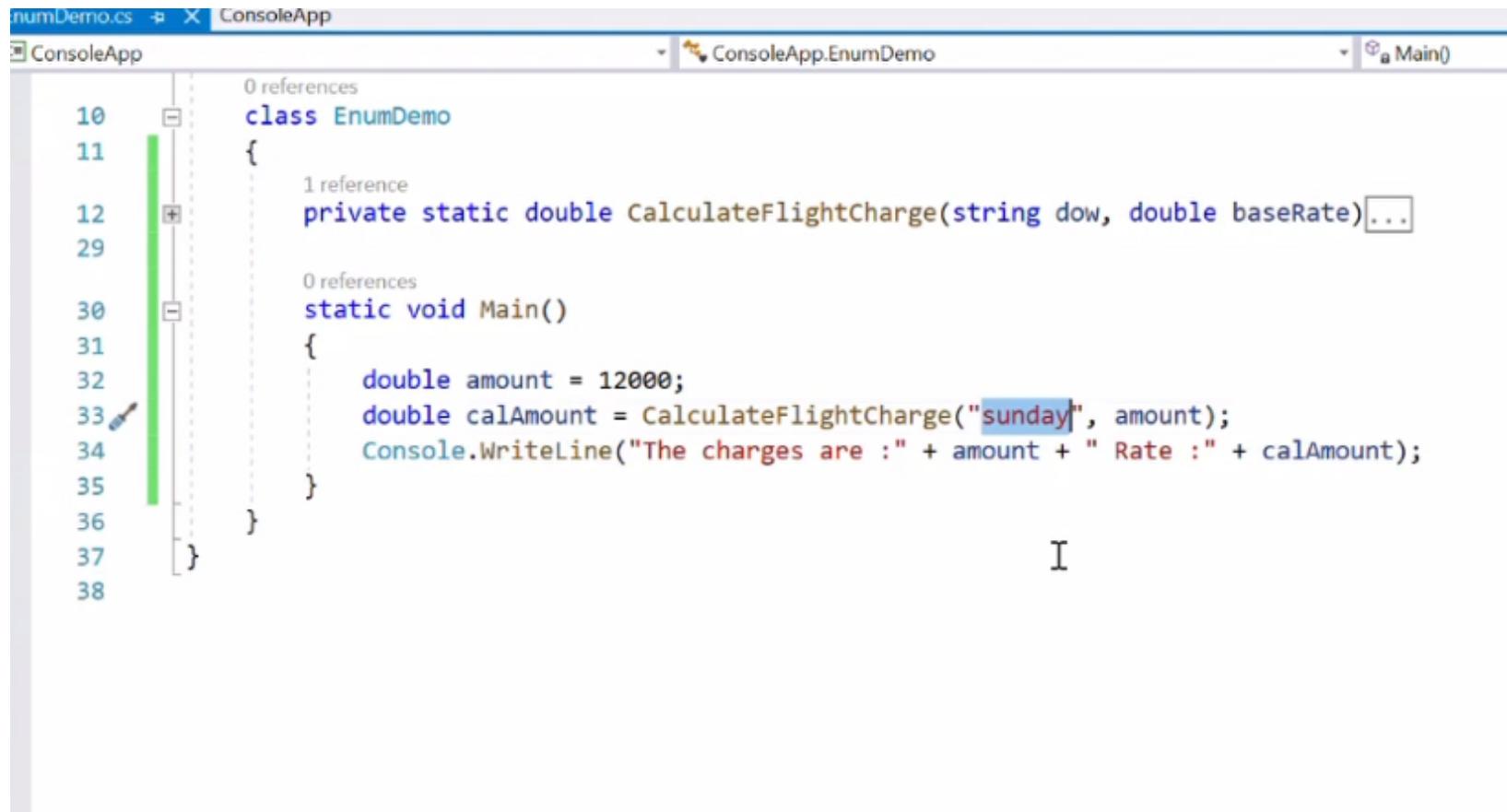
```
EnumDemo.cs  ✘ X  ConsoleApp
ConsoleApp  ConsoleApp.EnumDemo  Main()

0 references
10  class EnumDemo
11  {
12      1 reference
13      private static double CalculateFlightCharge(string dow, double baseRate)...
14
15      0 references
16      static void Main()
17      {
18          double amount = 12000;
19          double calAmount = CalculateFlightCharge("wed", amount);
20          Console.WriteLine("The charges are :" + amount + " Rate :" + calAmount);
21      }
22  }
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
```

```
The charges are :12000 Rate :0
Press any key to continue . . .
```

## Why the rate is 0

What if its a sunday?

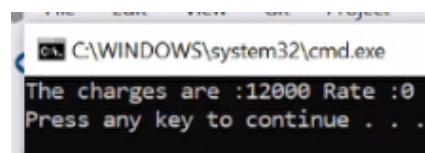


```
numDemo.cs  ✘ X  ConsoleApp
ConsoleApp
ConsoleApp.EnumDemo
Main()

0 references
class EnumDemo
{
    1 reference
    private static double CalculateFlightCharge(string dow, double baseRate)...

    0 references
    static void Main()
    {
        double amount = 12000;
        double calAmount = CalculateFlightCharge("sunday", amount);
        Console.WriteLine("The charges are :" + amount + " Rate :" + calAmount);
    }
}

1
```

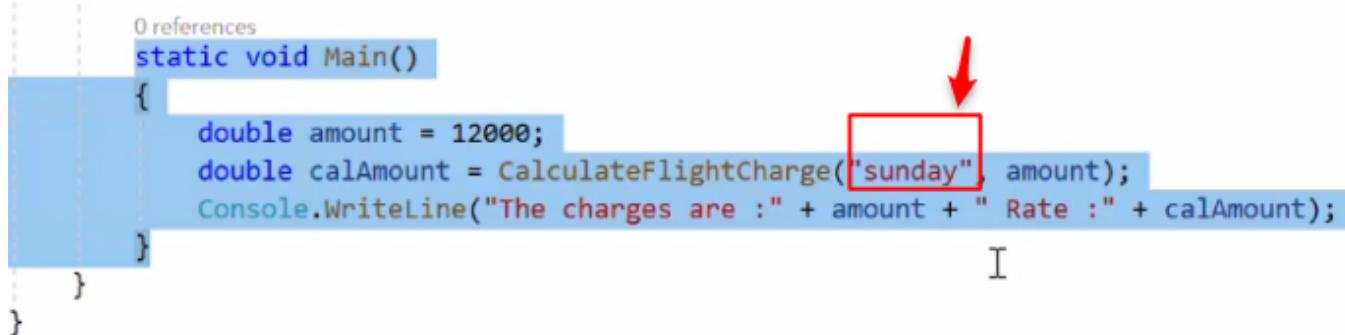


```
C:\WINDOWS\system32\cmd.exe
The charges are :12000 Rate :0
Press any key to continue . . .
```

**Why the rate is again 0**

**Because the ediot who is using our code**

he passed sunday but or logic is for Sun



```
0 references
static void Main()
{
    double amount = 12000;
    double calAmount = CalculateFlightCharge("sunday", amount);
    Console.WriteLine("The charges are :" + amount + " Rate :" + calAmount);
}
```

So, how will you make sure that the code whatever you have written will not break as it is breaking now

So, for that i need to have an agreement with this guy who is using my code you have to pass this particular value only, i'm restricting him

**So, how do you do that?**

----I'll just create a **ENUM**

```
EnumDemo.cs  ✘ X ConsoleApp
ConsoleApp                                         ConsoleApp.EnumDemo                                     CalculateFlightCharge
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace ConsoleApp
8   {
9       enum Dow {Sun, Mon, Tue, Wed, Thur, Fri, Sat}; ←
10      //Flight
11      class EnumDemo
12      {
13          private static double CalculateFlightCharge(string dow, double baseRate)...
14
15
16          static void Main()
17          {
18              double amount = 12000;
19              double calAmount = CalculateFlightCharge("sunday", amount);
20              Console.WriteLine("The charges are :" + amount + " Rupees");
21          }
22      }
23
24
25 }
```

now instead of calling this as string i call this  
as Dow

[? (local variable) double amount]

A screenshot of the Microsoft Visual Studio IDE showing a C# file named `EnumDemo.cs`. The code defines a class `EnumDemo` with a static method `CalculateFlightCharge` that takes a day of the week (`Dow`) and a base rate, and returns a charged amount based on the day. A red arrow points from the text "i get error here so instead of putting them as string here i'll directly specify the value" to the line where the `case` labels are defined as strings ("Sun", "Sat", etc.). The code editor shows syntax highlighting and a vertical ruler on the left.

```
12  class EnumDemo
13 {
14     private static double CalculateFlightCharge(Dow dow, double baseRate)
15     {
16         //logic
17         double amount = 0;
18         switch (dow)
19         {
20             case "Sun": amount = baseRate + (baseRate * 0.05); break; //5% on sunday
21             case "Sat": amount = baseRate + (baseRate * 0.07); break; //7% on satday
22             case "Mon":
23             case "Tue":
24             case "Wed":
25             case "Thur":
26             case "Fri":
27                 amount = baseRate - (baseRate * 0.025); break; //2.5% on Mon..Fri Discount
28         }
29         return amount;
30     }
31
32     static void Main()
33     {
34     }
}
```

to do this look at the magical thing i'll write switch ctrl + space  
then dow and i press down arrow

The screenshot shows the Microsoft Visual Studio IDE interface. The top bar displays the project name "ConsoleApp" and the file "EnumDemo.cs". The code editor window contains the following C# code:

```
12 class EnumDemo
13 {
14     private static double CalculateFlightCharge(Dow dow, double baseRate)
15     {
16         //logic
17         double amount = 0;
18         switch (dow)
19         {
20             default:
21         }
22         return amount;
23     }
24
25     static void Main()
26     {
27         double amount = 12000;
28         double calAmount = CalculateFlightCharge("sunday", amount);
29         Console.WriteLine("The charges are :" + amount + " Rate :" + calAmount);
30     }
31 }
32 }
```

The code editor features several visual indicators: a green vertical bar on the left margin, a yellow lightbulb icon at line 18 indicating a warning or suggestion, and a red squiggle under the string "sunday" in the Main() method. The status bar at the bottom shows the zoom level (121%), error counts (2 errors, 0 warnings), and navigation icons.

look at this the code is written for me

ConsoleApp

ConsoleApp.EnumDemo

```
double amount = 0;
switch (dow)
{
    case Dow.Sun:
        break;
    case Dow.Mon:
        break;
    case Dow.Tue:
        break;
    case Dow.Wed:
        break;
    case Dow.Thur:
        break;
    case Dow.Fri:
        break;
    case Dow.Sat:
        break;
    default:
        break;
}
return amount;
}
```

0 ⏪ ⏹ ⏷ ⏸ ⏵

The screenshot shows a Microsoft Visual Studio interface with the following details:

- Title Bar:** Shows "EnumDemo.cs\*" and "ConsoleApp".
- Code Editor:** Displays the following C# code:

```
16 // logic
17     double amount = 0;
18     switch (dow)
19     {
20         case DOW.Sun:
21             amount = baseRate + (baseRate * 0.05); //5% on sunday
22             break;
23         case DOW.Mon:
24         case DOW.Tue:
25         case DOW.Wed:
26         case DOW.Thur:
27         case DOW.Fri:
28             amount = baseRate - (baseRate * 0.025); //2.5% on Mon..Fri Discount
29             break;
30         case DOW.Sat:
31             amount = baseRate + (baseRate * 0.07); //7% on satday
32             break;
33     }
34     return amount;
35 }
36
37 static void Main()
38 {
```
- Status Bar:** Shows "121 %", "0", and other icons.
- Bottom Navigation:** Shows tabs for "Output", "Package Manager Console", "Error List ...", and "Immediate Window".
- Bottom Bar:** Shows "Ready".

## Now what's the advantage we'll get with enum

Now the enum whatever i'm creating is my user defined type  
now look at the magic -----this guy cannot pass----- sunday it wil give him error

The screenshot shows a Microsoft Visual Studio interface with the following details:

- Title Bar:** Shows the project name "ConsoleApp" and the file name "EnumDemo.cs".
- Status Bar:** Shows the zoom level "121 %", error count "1", warning count "0", and navigation icons.
- Code Editor:** Displays the following C# code:

```
7  namespace ConsoleApp
8  {
9      enum Dow {Sun, Mon, Tue, Wed, Thur, Fri, Sat};
10
11     //Flight
12     class EnumDemo
13     {
14         private static double CalculateFlightCharge(Dow dow, double baseRate)...
15
16
17         static void Main()
18         {
19             double amount = 12000;
20             double calAmount = CalculateFlightCharge("sunday", amount);
21             Console.WriteLine("The charges are :" + amount + " Rate :" + calAmount);
22         }
23     }
24 }
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
```

he just need to write Dow. and he will get the suggestion

EnumDemo.cs\* ConsoleApp

ConsoleApp

ConsoleApp.EnumDemo

Main()

```
7  namespace ConsoleApp
8  {
9      enum Dow {Sun, Mon, Tue, Wed, Thur, Fri, Sat};
10
11     //Flight
12     class EnumDemo
13     {
14         private static double CalculateFlightCharge(Dow dow, double baseRate)...
15
16
17         static void Main()
18         {
19             double amount = 12000;
20             double calAmount = CalculateFlightCharge(Dow.)...
21             Console.WriteLine("The charges are :" + amount + " Rate :" + calAmount);
22         }
23     }
24 }
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
```

121% ✖ 3 ⚠ 0 ↶ ↷ ⌂

Output Package Manager Console Error List ... Immediate Window

he'll pass Sun or whatever and he'll pass the amount

Now we don't have any error which he or she can make, when ever they are using my method this is the main purpose of using enum

- Mon
- Sat
- Sun
- Thur
- Tue
- Wed

you can initialize your enums

The screenshot shows a code editor in Visual Studio with the file 'EnumDemo.cs' open. The code defines a namespace 'ConsoleApp' containing an enum 'Dow' with values Sun, Mon, Tue, Wed, Thur, Fri, Sat, and a class 'EnumDemo' with a static method 'Main'. A tooltip for the value 'Dow.Sun = 0' is displayed, and a red arrow points to it. Handwritten text in red says: 'by default you can see whenever i move my mouse pointer on it, it will say Dow.Sun = 0 Dow.Mon = 1 .....soon.....'.

```
EnumDemo.cs  X  ConsoleApp
ConsoleApp
7  namespace ConsoleApp
8  {
9      enum Dow {Sun, Mon, Tue, Wed, Thur, Fri, Sat};
10     //Flight|   ↗ Dow.Sun = 0
11     class EnumDemo
12     {
13         private static double CalculateFlightCharge(Dow dow, double baseRate){...}
14
15         static void Main()
16         {
17             double amount = 12000;
18             double calAmount = CalculateFlightCharge(Dow.Sun, 12000);
19             Console.WriteLine("The charges are :" + amount + " Rate :" + calAmount);
20         }
21     }
22 }
```

121 % No issues found

EnumDemo.cs\* ✘ X ConsoleApp

ConsoleApp

ConsoleApp.EnumDemo

ConsoleApp

```
7  namespace ConsoleApp
8  {
9      enum Dow {Sun, Mon, Tue, Wed, Thur, Fri, Sat};
10
11     //Flight
12     class EnumDemo
13     {
14         private static double CalculateFlightCharge(Dow dow, double baseRate)
15         {
16             //logic
17             double amount = 0;
18             switch (dow)
19             {
20                 case 0:           meaning in case instead of Dow.Sun
21                     amount = baseRate + (baseRate * 0.05); //5% on sunday
22                     break;
23                 case Dow.Mon:
24                 case Dow.Tue:
25                 case Dow.Wed:
26                 case Dow.Thur:
27                 case Dow.Fri:
```

121 % No issues found

Output Package Manager Console Error List ... Immediate Window

Item(s) Saved

meaning in case instead of Dow.Sun  
you can specify 0 it will still work



Now, i don't want to pass 0 , i want to pass 100, so you can always initialize it by saying Sun = 100

The screenshot shows a code editor window for a C# file named 'EnumDemo.cs' within a project called 'ConsoleApp'. The code defines an enum 'Dow' with values 'Sun=100', 'Mon', 'Tue', 'Wed', 'Thur', and 'Sat'. It also contains a method 'CalculateFlightCharge' that uses a switch statement. The line 'case 100:' is highlighted with a red box, and the assignment 'amount = baseRate + (baseRate \* 0.05);' is also highlighted with a red box. A yellow circle with a question mark is placed on the line 'case Dow.Mon:'. The status bar at the bottom indicates 'No issues found'.

```
EnumDemo.cs*  X  ConsoleApp
ConsoleApp
ConsoleApp.EnumDemo

7  namespace ConsoleApp
8  {
9      enum Dow {Sun=100, Mon, Tue, Wed, Thur, Fri, Sat};
10
11     //Flight
12     class EnumDemo
13     {
14         private static double CalculateFlightCharge(Dow dow, double baseRate)
15         {
16             //logic
17             double amount = 0;
18             switch (dow)
19             {
20                 case 100:
21                     amount = baseRate + (baseRate * 0.05); //5% on sunday
22                     break;
23                 case Dow.Mon:
24                 case Dow.Tue:
25                 case Dow.Wed:
26                 case Dow.Thur:
27                 case Dow.Fri:
```

But usually we'll not do this , we'll always go with the standard format, but if you want you can always do it

## Using Enumeration

- Defining an Enumeration Type

```
enum Color { Red, Green, Blue }
```

- Using an Enumeration Type

```
Color colorPalette = Color.Red;
```

- Displaying an Enumeration Variable

```
Console.WriteLine("{0}", colorPalette); // Displays Red
```

- Displaying an Enumeration Variable's hidden value

```
Console.WriteLine("{0}", Convert.ToInt32( colorPalette));  
// Displays hidden numerical value of member Red
```

## Some Important Points To Remember About Enum

- First member of any enumeration has default value 0, if not assigned anything else
- Values for next members get incremented by 1 automatically

```
enum Color { Red, Green, Blue } //Red=0, Green=1, Blue=2
```

- You can assign value to first member and remaining members gets their values automatically, incremented by 1.

```
enum Color { Red = 100, Green, Blue } //Green=101,Blue=102
```

- You can assign distinct values to each and every member,

```
enum Color { Red = 100, Green = 200, Blue = 300 }
```

## Strings in C#

- Strings are objects in .NET Framework; therefore represented through **System.String** class.
- **System.String** class provides string related functions.
- It also has keyword representation **string** (*blue colored*) which is said to be an alias for **System.String**.
- The **string** type represents a string of Unicode characters.
- A string is a reference type.

```
Console.WriteLine("Enter a statement for Vowels and Consonents count");
string statement = Console.ReadLine();
int vowels_count = 0; int consonants_count = 0;
char[] arr = statement.ToCharArray();
for (int i = 0; i < arr.Length; i++)
{
    switch (arr[i])
    {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u': vowels_count++; break;
        default: consonants_count++; break;
    }
}
Console.WriteLine("No. of Vowels :" + vowels_count);
Console.WriteLine("No. of Consonents :" + consonants_count);
Console.Read();
```



StringDemo.cs\* ✘ ConsoleApp\*

ConsoleApp

ConsoleApp.StringDemo

```
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace ConsoleApp
8  {
9      class StringDemo
10     {
11         static void Main()
12         {
13             string name1 = "Shashi";
14             string name2 = "Shashi";
15
16             if(name1 == name2)
17             {
18                 Console.WriteLine("Same");
19             }
20             else
21             {
22                 Console.WriteLine("Different");
23             }
24
25     }
```

121 % ✓ No issues found

Output Package Manager Console Error List Immediate Window

StringDemo.cs   X ConsoleApp\*

ConsoleApp

0 references

```
11     static void Main()
12     {
13         string name1 = "Shashi";
14         string name2 = "Shashi";
15
16         if(name1 == name2)
17         {
18             Console.WriteLine("Same");
19         }
20         else
21         {
22             Console.WriteLine("Different");
23         }
24
25         if (name1.Equals(name2))
26         {
27             Console.WriteLine("Same");
28         }
29         else
30         {
31             Console.WriteLine("Different");
32         }
33     }
```

121 %    No issues found

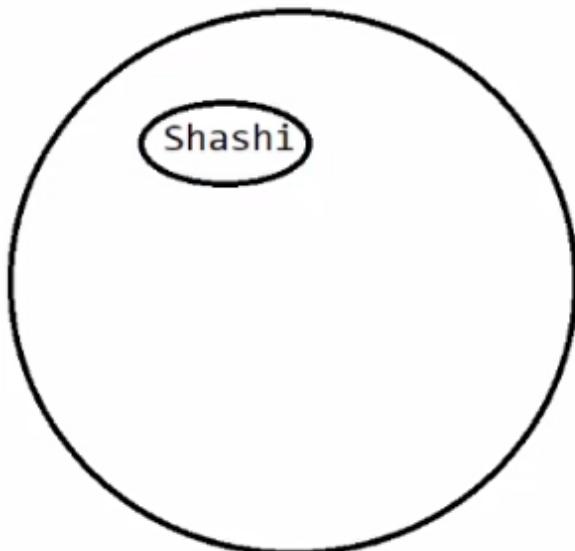
Output   Package Manager Console   Error List ...   Immediate Window

C:\WINDOWS\system32\cmd.exe

```
Same
Same
Press any key to continue . . .
```

So when ever you work with string in java, whenever you create a string so string will always be stored in a string pool

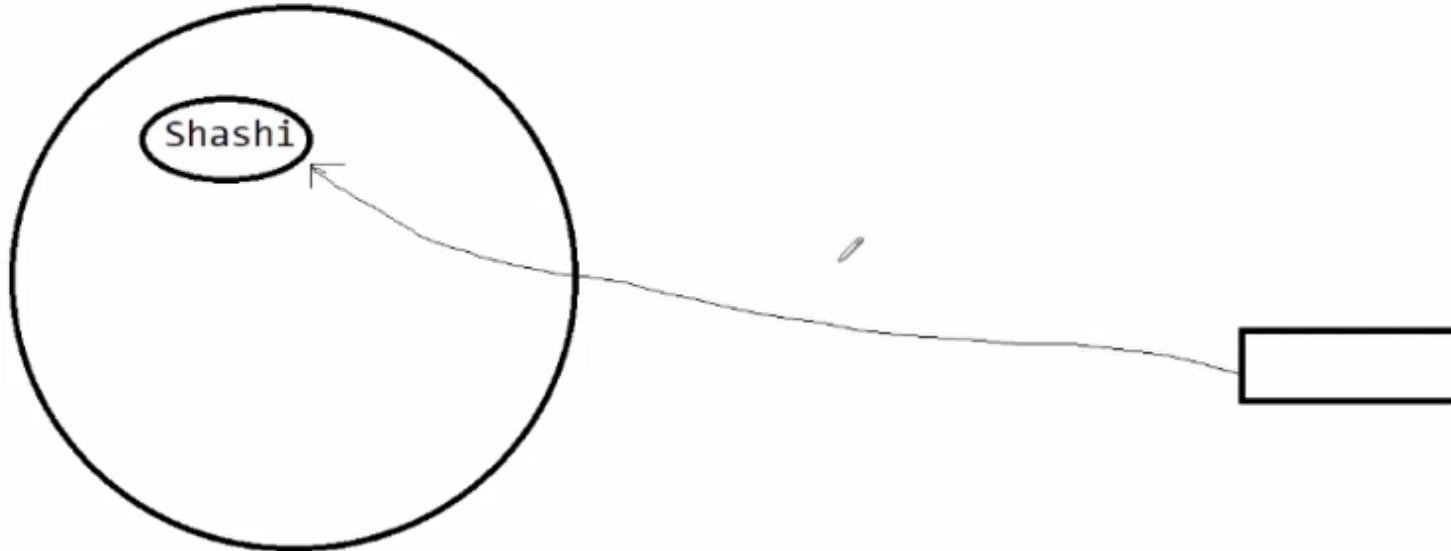
when i specify string name = "Shashi" so what it does is ,it has a string pool , so in the string pool --- shashi will be created and this is one of the pool instance



when every you say

```
string name1 = "Shashi";
```

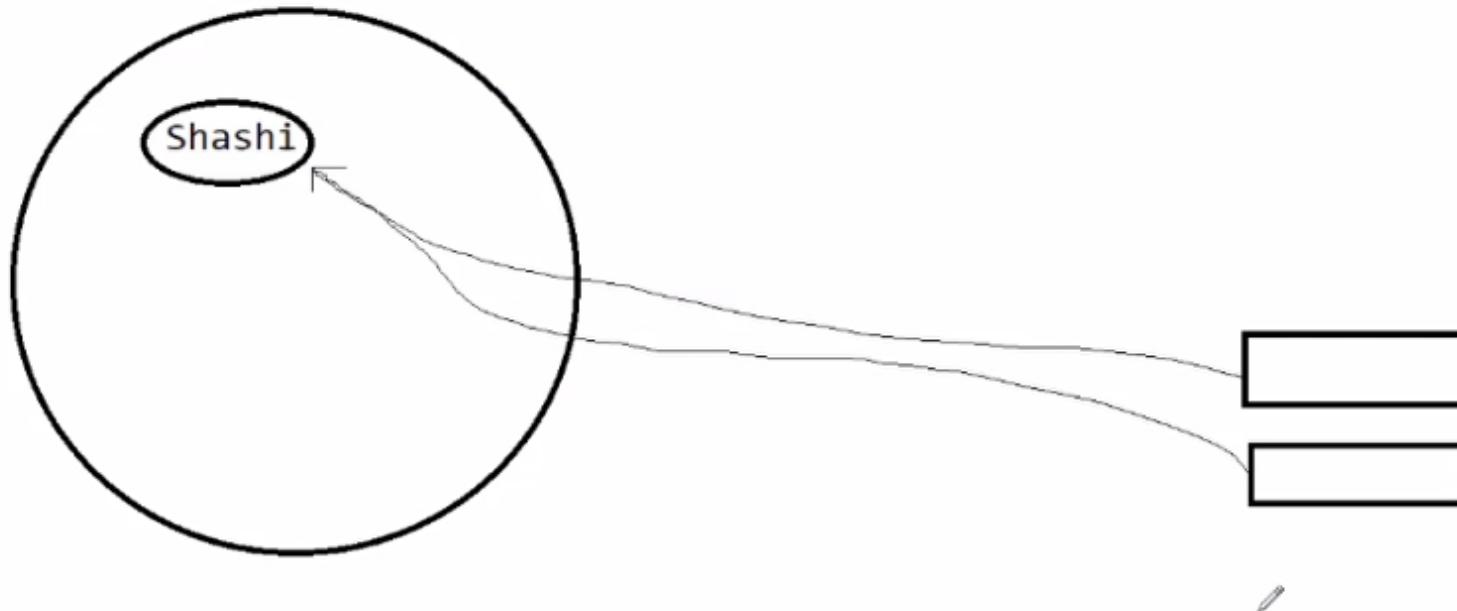
So , it will gone to this name1 = shashi refered from this particular code



Now when you are saying

```
string name2 = "Shashi";
```

then again name2 will be created here and name2 will also refer to the same string



that's why when you write

```
if(name1 == name2)  
    T
```

it is checking for the data , it is checking weather both of them are reffering to the same location  
So, yes they have the same data so you can see the  
outputs of this become true and true

StringDemo.cs X ConsoleApp

ConsoleApp

ConsoleApp.StringDemo

0 references

```
11     static void Main()
12     {
13         char[] name = { 'S', 'h', 'a', 's', 'h', 'i' };
14         string name1 = new string(name);
15         string name2 = new string(name);
16
17         if(name1 == name2)
18         {
19             Console.WriteLine("Same");
20         }
21         else
22         {
23             Console.WriteLine("Different");
24         }
25
26         if (name1.Equals(name2))
27         {
28             Console.WriteLine("Same");
29         }
30         else
31         {
32             Console.WriteLine("Different");
33         }

```

121 % No issues found

```
Same
Same
Press any key to continue . . .
```

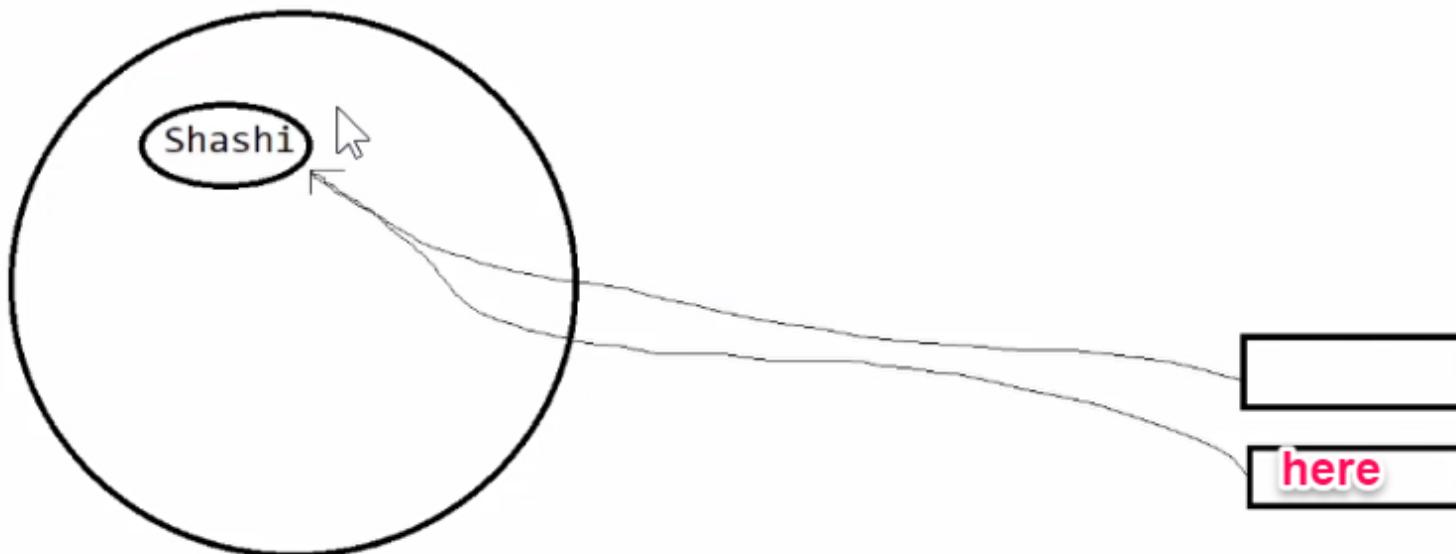
**Assignment :- Read What is the start index and end index ?**

So, strings are **Mutable**

what are Mutable ?

Once the string is been allocated if you try to modify this particular string , so it will allocate the new location it will not be changed in the same place and the new location address will be stored here

---



We have done the same thing in java also,

The screenshot shows a Visual Studio code editor with a C# file named StringDemo.cs. The code creates a character array 'name' and two string objects 'name1' and 'name2'. It then compares them using == and Equals methods.

```
StringDemo.cs  ConsoleApp
ConsoleApp  ConsoleApp.StringDemo
0 references
11 static void Main()
12 {
13     char[] name = { 'S', 'h', 'a', 's', 'h', 'i' };
14     string name1 = new string(name); ← so in java when ever you specify
15     string name2 = new string(name); new string ()
16
17     if(name1 == name2) ← and you write == it will give you false
18     {
19         Console.WriteLine("Same");
20     }
21     else
22     {
23         Console.WriteLine("Different");
24     }
25
26     if (name1.Equals(name2)) ← but where as .Equal() gives you TRUE
27     {
28         Console.WriteLine("Same");
29     }
30     else
31     {
32         Console.WriteLine("Different");
33     }

```

so in java when ever you specify  
new string ()

and you write == it will give you false

but where as .Equal() gives you TRUE

Because in java these 2 name1 and name2  
will be reffering to the same location  
whenever you specify " shashi" directly like  
string name1 = "shashi";  
string name2 = "shashi";

So because they have the same value they reffer to same location , and this is  
Same in case of .NET

Now here the main difference in java and .NET would be

in java when ever you specify name = new string() and name is of string type ,so there will be lot of change

## What will be the change there?

So, it will be create 2 different instances in java



and whenever you compare it you say == it will give you FALSE at that time when you create new instance in java

That's a reason we have **.Equal()** method using which we compare in java world because we want to compare not the instances rather the value

**So, to compare value in java we always go with the .Equals () method**

But where as when you come down to .NET that's not the case there you use .Equals() or == it always compare the values

But in .Net when you modify string it will always break, it will always store data in new reference , Every time you keep adding the data it will treat that as a new string

That's a reason if you are asking the user to enter some string and trying to store that in a string

The screenshot shows the Microsoft Visual Studio IDE interface. The title bar displays three tabs: "StringDemo1.cs\*", "StringDemo.cs", and "ConsoleApp\*". The status bar at the bottom indicates "No issues found". The main code editor window contains the following C# code:

```
StringDemo1.cs*  X StringDemo.cs      ConsoleApp*          ConsoleApp.StringDemo1
ConsoleApp
9   class StringDemo1
10  {
11    static void Main()
12    {
13      string readData = string.Empty;
14      Console.WriteLine("Enter document. Press '0' to exit:");
15      string data = string.Empty;
16      do
17      {
18        data = Console.ReadLine();
19        if(data == "0")
20        {
21          break;
22        }
23        readData = readData + data;
24      } while (1==2);
25    }
26  }
27
28
```

The code demonstrates a loop where the user can enter multiple lines of text. The variable `readData` is initially set to an empty string. Inside the loop, each line entered by the user is appended to `readData`. This results in a new string object being created for each line, as the original reference to `readData` is overwritten.

So, if you have such scenario never use string

Maybe you are writing a particular program which will read a particular data from the file, you are reading lot of content from the file and you are trying to store it in a string variable, so you should not use a string because as and when you are reading some content it keeps growing and that will be a **Performance head**

So, instead of putting that to a string you can you can always put it in a **StringBuilder**

We can take one string or multiple string and put it to a string builder

A screenshot of the Visual Studio IDE showing a C# code editor. The code is as follows:

```
StringDemo1.cs*  X  StringDemo.cs  ConsoleApp*
```

```
ConsoleApp
7  namespace ConsoleApp
8  {
9      0 references
10     class StringDemo1
11     {
12         0 references
13         static void Main()
14         {
15             StringBuilder readData = new StringBuilder();
16             Console.WriteLine("Enter");
17             string data = string.Empty;
18             data = Console.ReadLine();
19             if(data == "0")
20             {
21                 break;
22             }
23             readData = readData + data;
24         } while (1==2);
25     }
26 }
27 }
28 }
```

The cursor is at line 13, and a tooltip is displayed over the line of code: **▲ 1 of 6 ▼ StringBuilder()**. The tooltip text is: "Initializes a new instance of the `StringBuilder` class." A red arrow points from the text "look at this our `StringBuilder` has 6 overloads" to the tooltip.

look at this our `StringBuilder` has 6 overloads

`StringBuilder` without any parameter, So this is a constructor

So **Constructor** are those special methods which are automatically been called whenever you create a new instance/ new object



you can specify what will be the initial capacity, initial capacity meaning like if you know how many word you can capture so you can pass that as a initial capacity

you can see it has 6 overloads

```
StringDemo1.cs*      StringDemo.cs      ConsoleApp*  
mscorlib  
1  Assembly mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089  
4  
5  using ...  
10  
11 namespace System.Text  
12 {  
13     public sealed class StringBuilder : ISerializable  
14     {  
15         ...public StringBuilder();  
16         ...public StringBuilder(int capacity);  
17         ...public StringBuilder(string value);  
18         ...public StringBuilder(string value, int capacity);  
19         ...public StringBuilder(int capacity, int maxCapacity);  
20         ...public StringBuilder(string value, int startIndex, int length, int capacity);  
21         ...  
22         ...public char this[int index] ...  
23         ...public int MaxCapacity { get; }  
24         ...public int Length { get; set; }  
25         ...public int Capacity { get; set; }  
26         ...  
27         ...public StringBuilder Append(object value);  
28         ...public StringBuilder Append(char[] value);  
29         ...public StringBuilder Append(ulong value);  
30         ...public StringBuilder Append(uint value);  
31     }  
32 }  
121 %  No issues found
```

Output Package Manager Console Error List Immediate Window