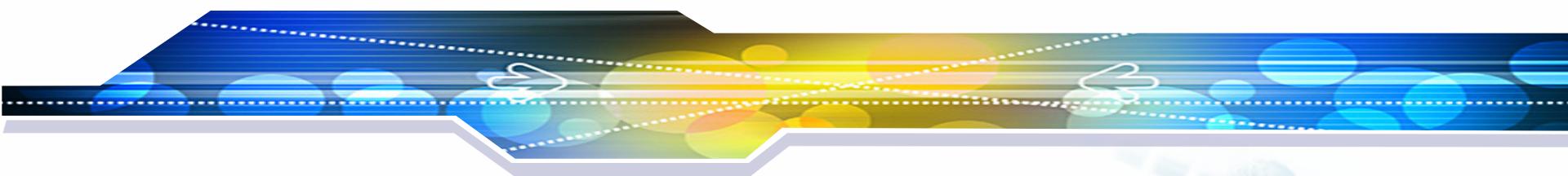




TRƯỜNG ĐẠI HỌC MỎ  
TP HỒ CHÍ MINH

BÀI GIẢNG  
**LẬP TRÌNH GIAO DIỆN**



**CHƯƠNG 6:**  
**MẢNG – CHUỖI**



## 1. Mảng.

1. Giới thiệu mảng
2. Khai báo mảng
3. Làm việc với mảng
4. Truyền mảng cho phương thức
5. Mảng nhiều chiều
6. Các lớp tập hợp thông dụng

## 2. Chuỗi.

1. Giới thiệu Chuỗi
2. Phương thức khởi tạo của lớp string
3. Các phương thức của lớp string
4. Các thao tác với chuỗi
5. Lớp StringBuilder

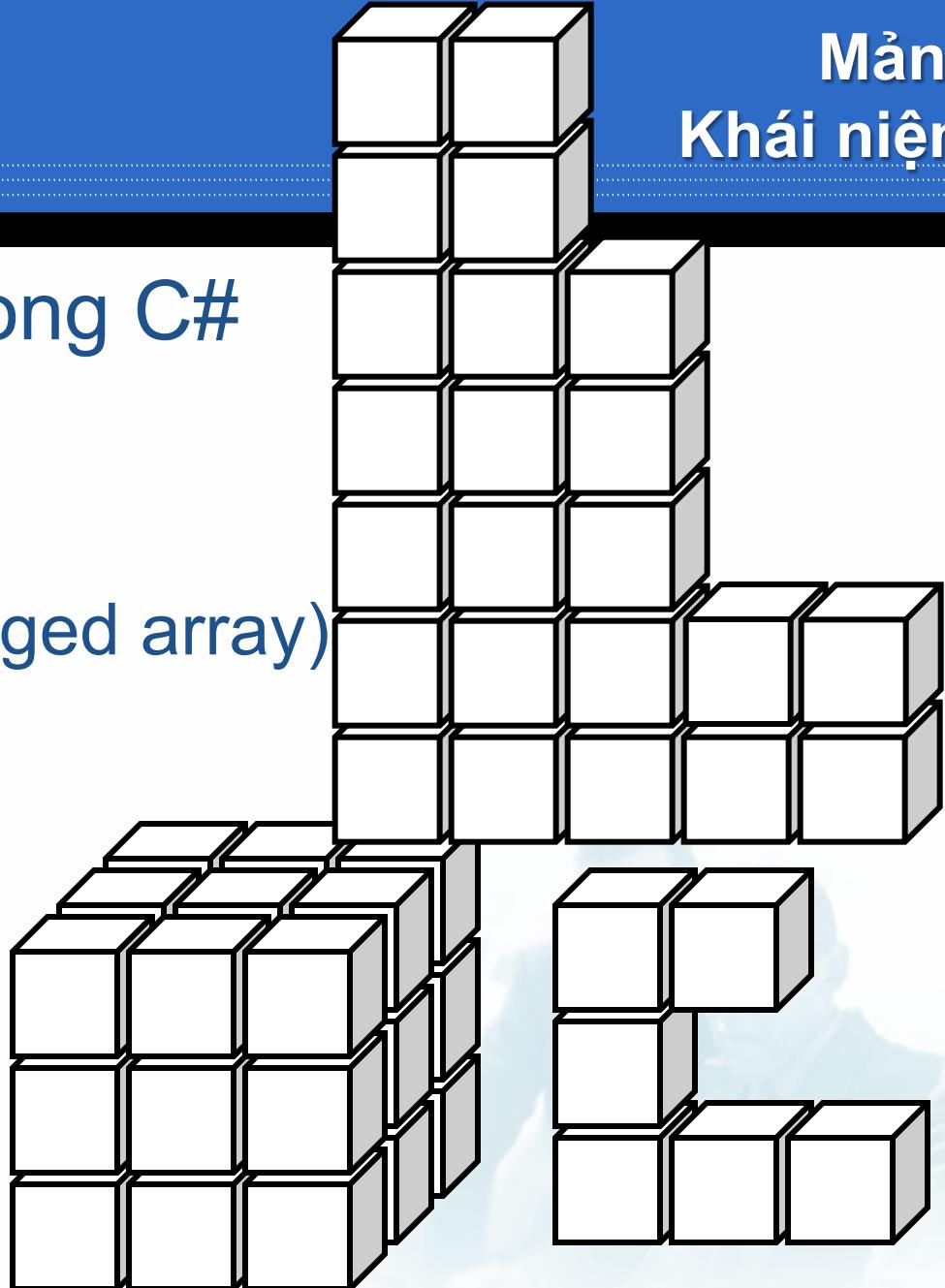
## 1. Giới thiệu mảng:

- Mảng là tập hợp có thứ tự của những đối tượng có cùng một kiểu dữ liệu.
- Các phần tử của mảng được truy xuất theo tên và vị trí.

## 2. Đặc điểm mảng trong C#

- Mảng là kiểu tham chiếu
- Tất cả các kiểu mảng đều có lớp cơ sở: System.Array

- Phân loại mảng trong C#
  - Mảng một chiều
  - Mảng nhiều chiều
  - Mảng zic zac (Zagged array)
  - Mảng hỗn hợp



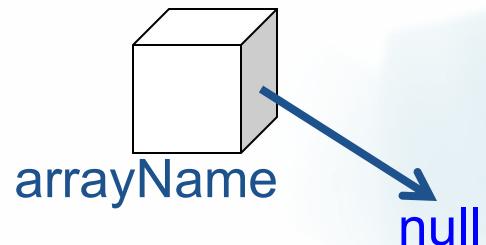
- Mảng hỗn hợp
- Mảng 1 chiều
- Mảng nhiều chiều
- Mảng zích zắc

`type [ ] arrayName;`

`type [,] arrayName;`  
`type [,,] arrayName;`

`type [ ][ ] arrayName;`

`type [ ][,,][,] arrayName;`



## ■ Mảng 1 chiều

```
type [ ] arrayName = {v1, v2, ..., vn};  
type [ ] arrayName = new type [ ]{v1, v2, ..., vn};  
type [ ] arrayName = new type [n]{v1, v2, ..., vn};
```

## ■ Mảng nhiều chiều

```
type [,] arrayName = {  
    {v1,v2,...,vn},  
    ...,  
    {v1,v2,...,vn}};  
  
type [,] arrayName = new type[,] {...};  
type [,] arrayName = new type[n1, n2] {...};
```

## ■ Mảng zích zắc

```
type[][] arrayName = {  
    new type[] {v1,...vn},  
    new type[] {v1, ..., vm}, ...};
```

```
type[][] arrayName = {  
    new type[n] {v1,...vn}, ● ●  
    new type[m] {v1, ..., vm}, ...};
```



```
type[][] arrayName = new type[][] {  
    new type[] {v1,...vn},  
    new type[] {v1, ..., vm}, ...};
```

```
type[][] arrayName = new type[n][] {  
    new type[] {v1,...vn},  
    new type[] {v1, ..., vm},  
    ...};
```



- Mảng 1 chiều

```
arrayName[x];
```

- Mảng nhiều chiều

```
arrayName[x, y];
```

```
arrayName[x, y, z];
```

- Mảng kích thước

```
arrayName[x][y];
```

- Tạo mảng
  - Dùng bộ khởi tạo
  - Dùng toán tử **new**
- Để lấy **số phần tử** của mảng chúng ta dùng thuộc tính **arrayName.Length**

## ■ Mảng 1 chiều

```
arrayName = new type[n];
```

```
type [ ] arrayName = new type [n];
```

## ■ Mảng nhiều chiều

```
arrayName = new type[n1, n2];
```

```
type [,] arrayName = new type [n1, n2];
```

## ■ Mảng zicz zắc

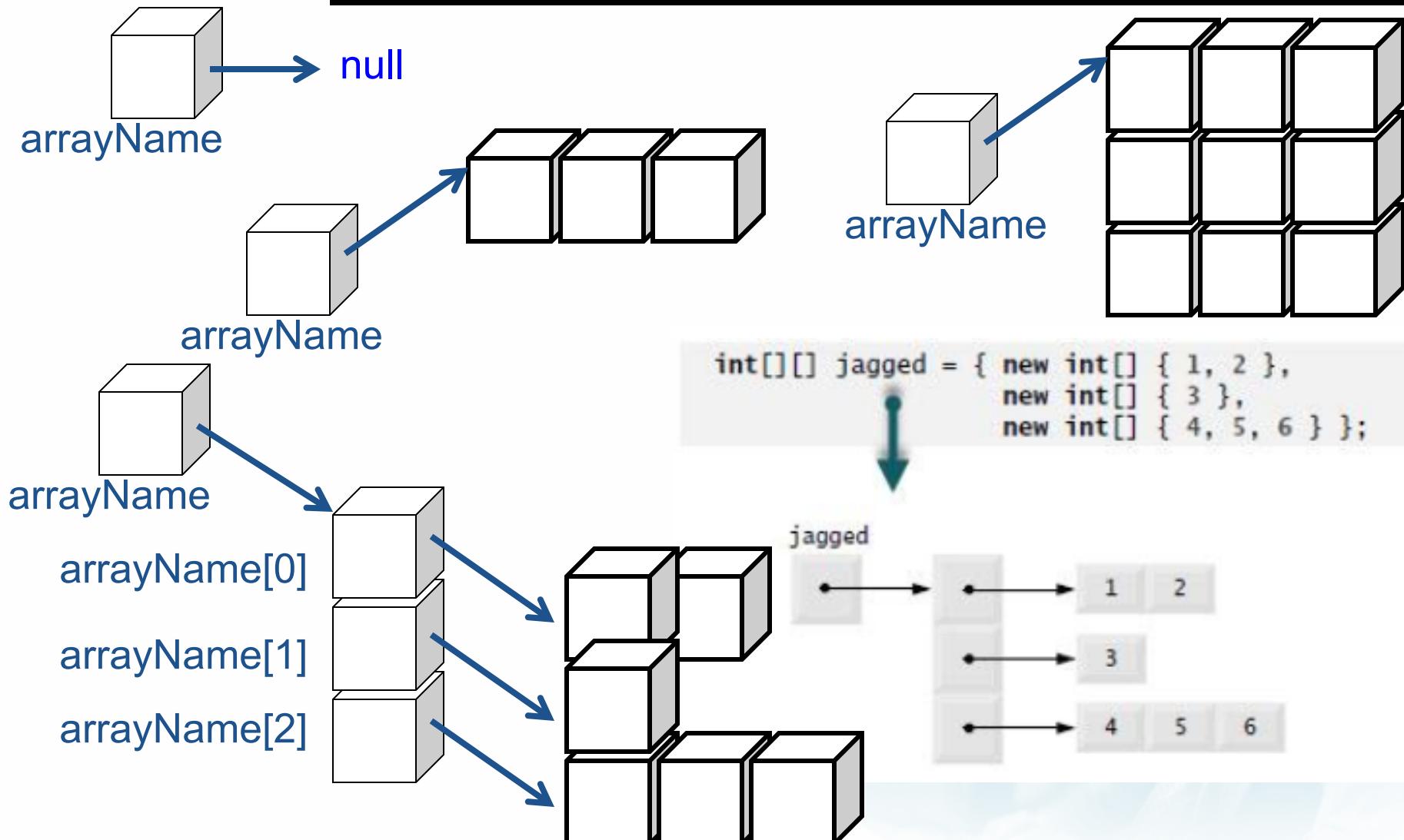
```
type [ ][ ] arrayName = new type [n][ ];
```

```
arrayName[0] = new type[m1];
```

```
arrayName[1] = new type[m2];
```

```
...
```

```
arrayName[n] = new type[mn];
```



- Sắp xếp mảng: Nếu các phần tử của mảng là kiểu định nghĩa trước, ta có thể sắp xếp tăng dần bằng **Array.Sort()**

Ví dụ

```
int[] arrayInt = { 5, 7, 3, 8, 2 };  
Array.Sort(arrayInt);
```

=> kết quả là dãy {2,3,5,7,8}

- Duyệt mảng một chiều: Duyệt từng phần tử dựa vào chỉ số

```
public static void Main( string[] args )  
{  
    const int ARRAY_LENGTH = 10; // create a named constant  
    int[] array = new int[ ARRAY_LENGTH ]; // create array  
    // calculate value for each array element  
    for ( int i = 0; i < array.Length; i++ )  
        array[ i ] = 2 + 2 * i;  
    Console.WriteLine( "{0}{1,8}", "Index", "Value" ); // headings  
    // output each array element's value  
  
    for ( int i = 0; i < array.Length; i++ )  
        Console.WriteLine( "{0,5}{1,8}", i, array[ i ] );  
    Console.ReadLine();  
} // end Main
```



Index	Value
0	2
1	4
2	6
3	8
4	10
5	12
6	14
7	16
8	18
9	20



## Mảng Làm việc với mảng

- Duyệt mảng một chiều: Dùng lệnh **foreach** duyệt từng phần tử

```
public static void Main( string[] args )
{
    const int ARRAY_LENGTH = 10; // create a named constant
    int[] array = new int[ ARRAY_LENGTH ]; // create array
    // calculate value for each array element
    for ( int i = 0; i < array.Length; i++ )
        array[ i ] = 2 + 2 * i;

    // output each array element's value
    foreach(int number in array)
        Console.Write( number + "\t");
    Console.ReadLine();
} // end Main
```



```
file:///E:/Tailieu/C#2010/VS2010 BOOKS/csfp4_examples/ch08/fig08_04/InitArr...
2      4      6      8      10     12     14     16     18     20
```

- Duyệt mảng hai chiều: Dùng hai vòng lặp **for** để duyệt qua các hàng và các cột

```
public double GetAverage(int [,] arrInt)
{
    int sum = 0;
    for (int i=0;i< arrInt.GetLength(0);i++)
        for (int j = 0; j < arrInt.GetLength(1); j++)
    {
        sum += arrInt[i, j];
    }
    return (double)sum / arrInt.GetLength(0) * arrInt.GetLength(1);
} // end method GetAverage
```



- Duyệt mảng Zagged: Sử dụng lệnh **foreach**

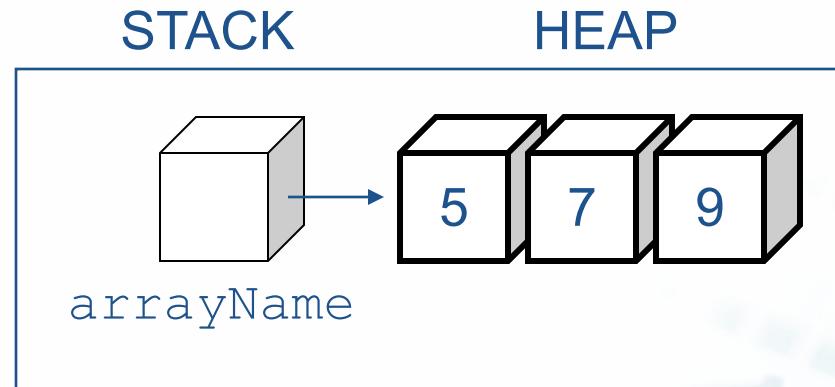
```
public int SumZaggedArray()
{
    int[][] jagged = { new int[] { 1, 2 },
                      new int[] { 3 },
                      new int[] { 4, 5, 6 } };
    int sum = 0;
    foreach (var row in jagged)
    {
        // loop through each element in current row
        foreach (var element in row)
            sum+= element;
    }
    return sum;
}
```



# Mảng Mảng giá trị - Mảng tham chiếu

## ▪ Mảng kiểu giá trị

```
int arrayName = new int[3]{5,7,9};
```

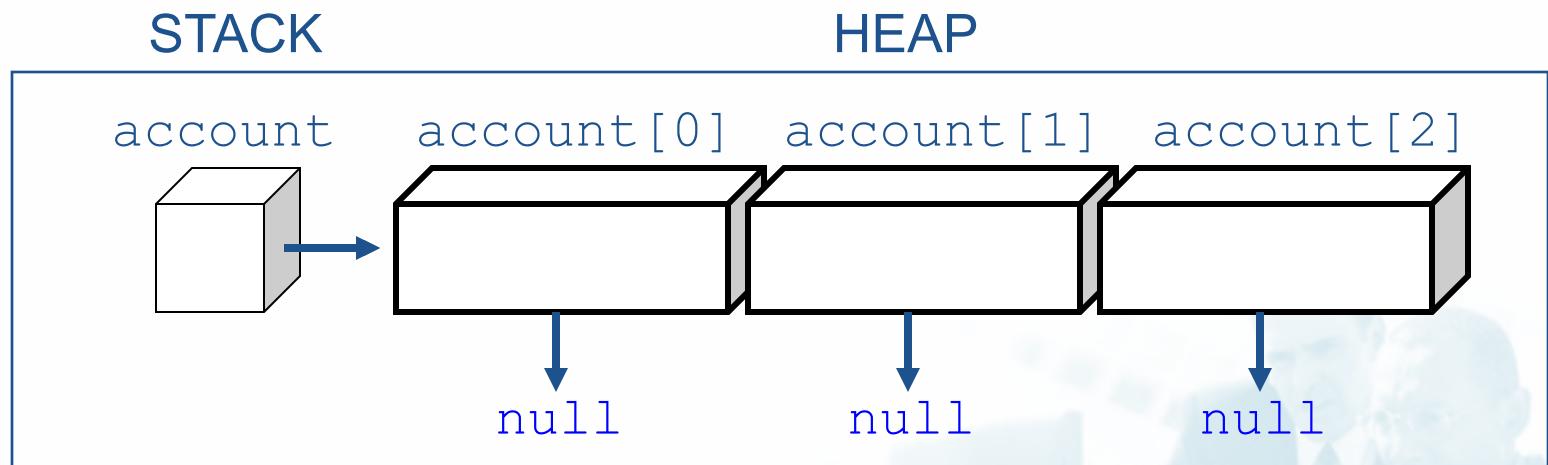




# Mảng Mảng giá trị - Mảng tham chiếu

## ■ Mảng đối tượng

```
BankAccount account = new BankAccount[3];
```

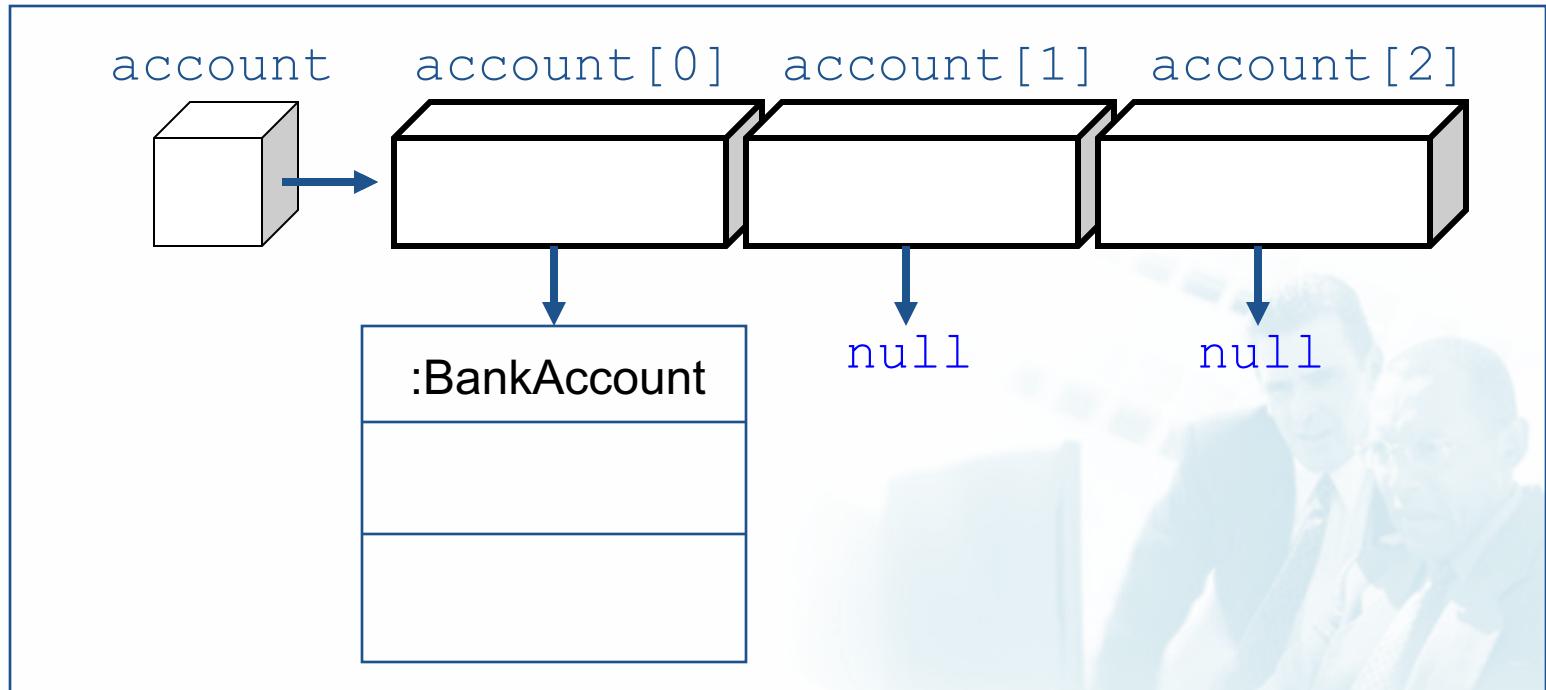




# Mảng Mảng giá trị - Mảng tham chiếu

## ■ Mảng đối tượng

```
account[0] = new BankAccount();
```



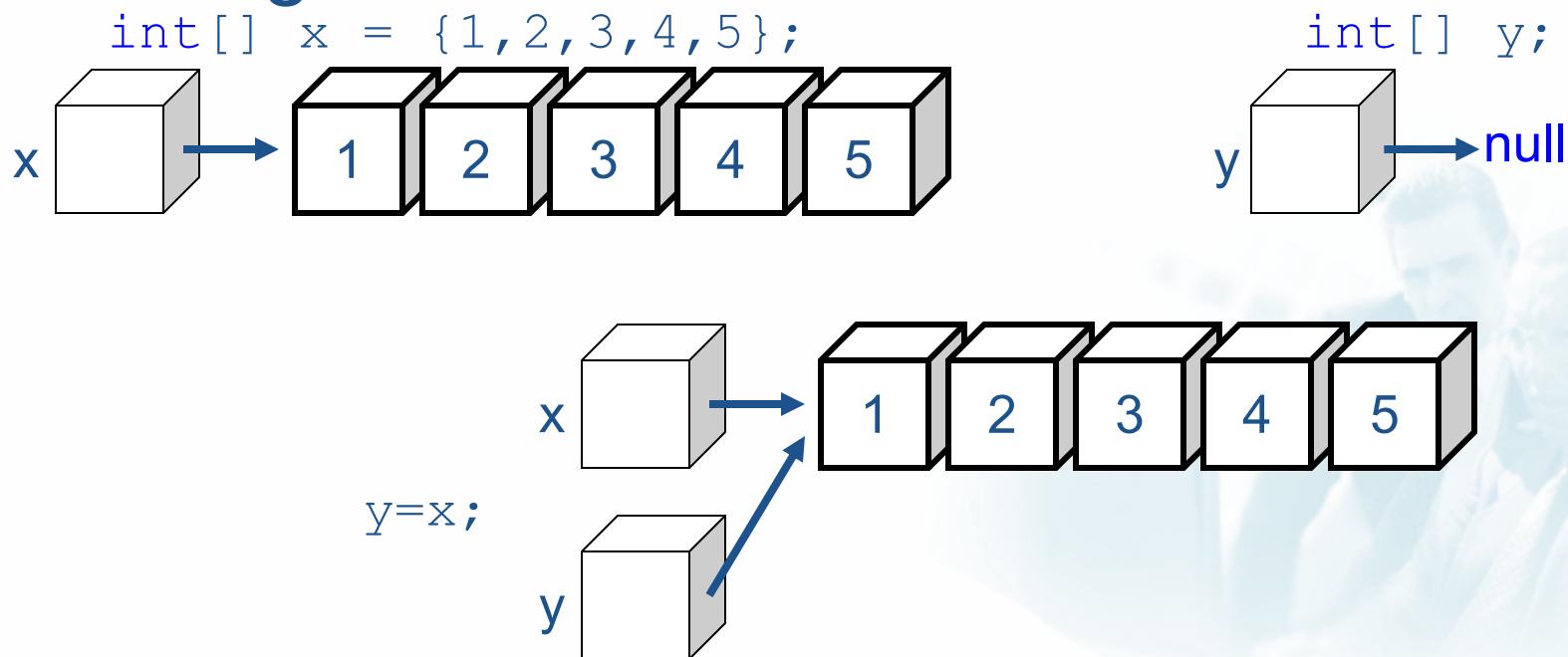
- Kiểu trả về của phương thức là mảng

```
type[ ] MethodName(...)  
{  
    type[ ] result;  
    ...  
    return result;  
}
```

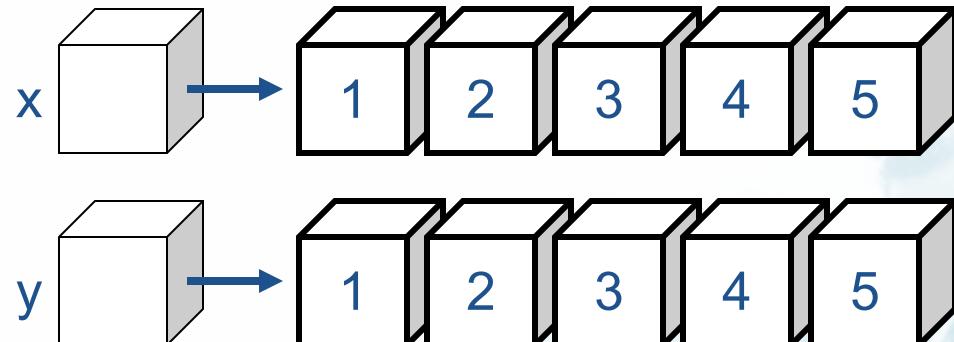
- Tham số của phương thức là mảng

```
MethodName(type[ ] arrayName)  
{  
    ...  
}
```

- Mảng là kiểu tham chiếu. Khi sao chép 1 biến mảng sang 1 biến mảng khác, chúng ta sẽ có 2 tham chiếu đến cùng 1 instance mảng



- Nếu muốn sao chép các instance mà mảng đang tham chiếu đến chúng ta thực hiện 2 bước
  - Tạo một instance mảng mới
  - Thiết lập các giá trị trong mảng mới như mảng gốc



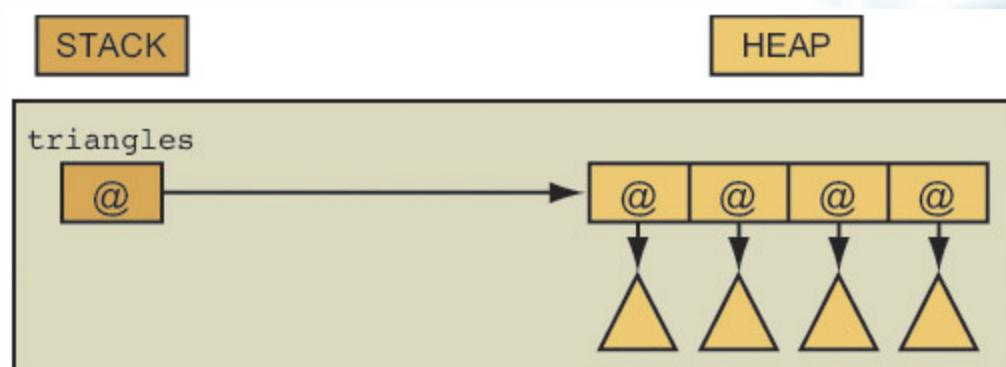
```
int[] y = new int[x.Length];  
  
for (int i=0; i<x.Length; i++)  
    y[i] = x[i]
```



## Mảng Sao chép Mảng

- Sao chép mảng là “shallow copy” chứ không phải “deep copy”

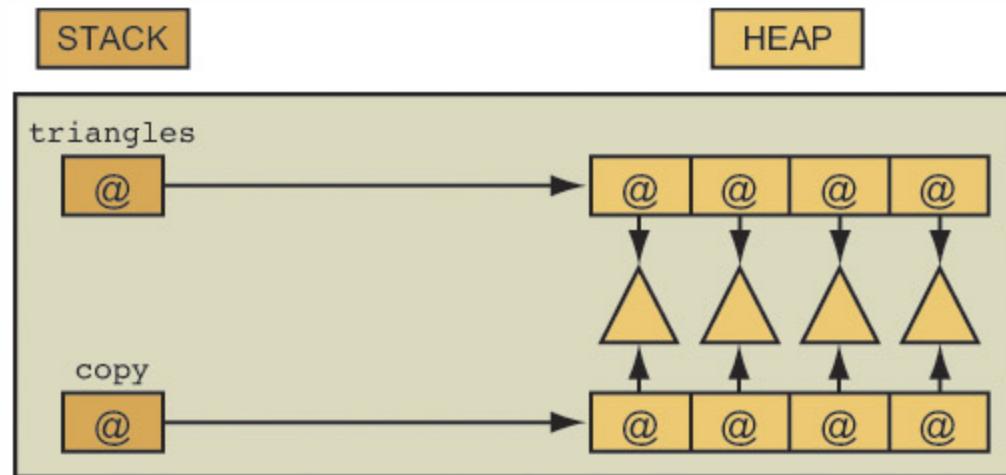
```
Triangle[] triangles = new Triangle[4];
for (int i = 0; i < triangles.Length; i++)
{
    triangles[i] = new Triangle();
}
```





# Mảng Sao chép Mảng

```
Triangle[] copy = new Triangle[triangles.Length];  
for (int i = 0; i < copy.Length; i++)  
{  
    copy[i] = triangles[i];  
}
```





# Mảng System.Array

- Mảng là đối tượng, dẫn xuất từ lớp cơ sở trừu tượng System.Array
- Chúng ta có thể dùng bất kỳ phương thức và thuộc tính nào của System.Array trên mảng do chúng ta tạo ra

Thành viên	Loại thành viên	Ý nghĩa
<b>Rank</b>	Readonly instance property	Trả về số chiều của mảng
<b>GetLength</b>	Instance method	Trả về số phần tử trong chiều được chỉ định của mảng
<b>Length</b>	Readonly instance property	Trả về tổng số phần tử của mảng
<b>GetLowerBound</b>	Instance method	Trả về cận dưới của một chiều cho trước. Phần lớn là 0
<b>GetUpperBound</b>	Instance method	Trả về cận trên của một chiều cho trước. Phần lớn là số phần tử của chiều đó trừ 1
<b>IsReadOnly</b>	Readonly instance property	Cho biết mảng chỉ đọc hay không. Luôn luôn là false
<b>IsSynchronized</b>	Readonly instance property	Cho biết mảng truy cập an toàn thread hay không. Luôn luôn là false
<b>SyncRoot</b>	Readonly instance property	Trả về một đối tượng cho phép truy cập đồng bộ đến mảng. Luôn luôn trả về chính mảng đang dùng
<b>IsFixedSize</b>	Readonly instance property	Cho biết mảng có kích thước cố định hay không. Luôn luôn là true
<b>GetValue</b>	Instance method	Trả về giá trị của phần tử

# Mảng System.Array



Thành viên	Loại thành viên	Ý nghĩa
<code>SetValue</code>	Instance method	Thiết lập giá trị của phần tử
<code>GetEnumerator</code>	Instance method	Trả về một <code>IEnumerator</code> . Cho phép dùng câu lệnh <code>foreach</code>
<code>Sort</code>	Static method	Sắp xếp các phần tử trong 1 mảng, trong 2 mảng hay trong một vùng của 1 mảng. Kiểu của các phần tử phải implement <code>IComparer</code> interface hay phải truyền một đối tượng mà kiểu implement interface <code>IComparer</code>
<code>BinarySearch</code>	Static method	Tìm kiếm theo thuật toán tìm kiếm nhị phân. Phương thức này giả sử mảng đã được sắp xếp. Kiểu của các phần tử phải implement <code>IComparer</code> interface. Chúng ta thường dùng phương thức Sort trước khi gọi <code>BinarySearch</code> .
<code>IndexOf</code>	Static method	Trả về vị trí xuất hiện đầu tiên của một giá trị trong mảng 1 chiều hay trong một phần của mảng
<code>LastIndexOf</code>	Static method	Trả về vị trí xuất hiện cuối cùng của một giá trị trong mảng 1 chiều hay trong một phần của mảng
<code>Reverse</code>	Static method	Đảo thứ tự các phần tử trong mảng 1 chiều hay trong một phần của mảng
<code>Clone</code>	Instance method	Tạo một mảng mới là bản sao của mảng gốc
<code>CopyTo</code>	Instance method	Sao chép các phần tử từ một mảng sang một mảng khác
<code>Copy</code>	Static method	Sao chép một phần các phần tử sang mảng khác, thực hiện bất kỳ yêu cầu casting nào
<code>Clear</code>	Static method	Thiết lập một vùng các phần tử thành 0 hay null
<code>CreateInstance</code>	Static method	Tạo một instance của mảng. Phương thức hiếm dùng này cho phép định nghĩa mảng động lúc thời gian chạy với bất kỳ kiểu, vùng, cận
<code>Initialize</code>	Instance method	Gọi constructor mặc nhiên của các phần tử kiểu giá trị (phương thức này không thực hiện nếu các phần tử là kiểu tham chiếu)



# Các lớp tập hợp thông dụng

## 1. Hạn chế của mảng:

- Kích thước của mảng không thể thay đổi khi chương trình thực thi.
- Các phần tử chung mảng phải có cùng kiểu dữ liệu.
- Khi thêm và xóa phần tử, ta phải dời và duyệt các phần tử liên quan.

## 2. Các phương thức thao tác trên tập hợp:

- ArrayList
- Hashtable
- Stack
- Queue
- SortedList

Namespace: System.Collection  
=> Using System.Collection

- ArrayList/List là một **mảng động**, cho phép lưu trữ và nhận một chuỗi các giá trị thông qua chỉ mục giống như một mảng
- ArrayList/List khắc phục những giới hạn của mảng
  - Kích thước của mảng có thể thay đổi lúc thực thi.
  - Có thể thêm, xóa phần tử trong mảng dễ dàng.
  - Có thể lưu trữ các phần tử khác nhau.
  - Kích thước mặc định là 0, và nếu các phần tử được lấp đầy, kích thước tự động tăng gấp đôi.

## ■ Tạo đối tượng

Properties	Ý nghĩa
<b>Capacity</b>	Số lượng phần tử có thể chứa
<b>Count</b>	Số lượng phần tử đang chứa

```
ArrayList arrList = new ArrayList();  
ArrayList arrList = new ArrayList(capacity);
```

```
List<type> list = new List<type>();  
List<type> list = new List<type>(capacity);
```

- Thêm dữ liệu

```
listName.Add(data);
```

- Truy cập dữ liệu

```
listName[index];
```

```
foreach (type variable in listName)
{
    xử lý variable
}
```

## ■ Xóa dữ liệu

```
listName.Remove(data);
```

```
listName.Clear();
```

## ■ Chèn dữ liệu

```
listName.Insert(index, data);
```

## ■ Sắp xếp

```
listName.Sort();
```



# Collections

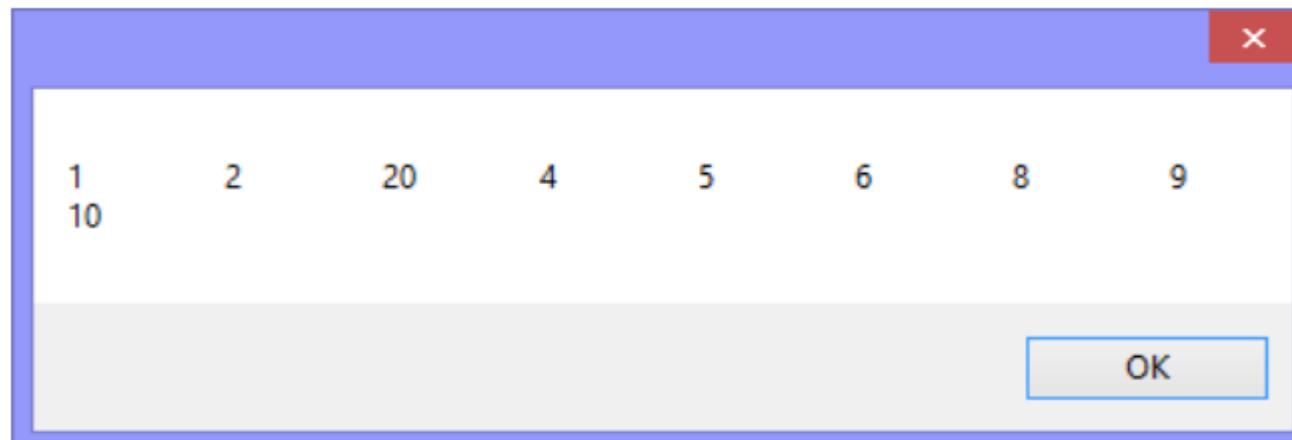
## ArrayList/List

Methods	Ý nghĩa
<code>int Add(object)</code>	Thêm một đối tượng vào cuối mảng
<code>void Clear()</code>	Xóa
<code>bool Contains(object)</code>	Kiểm tra có tồn tại một đối tượng trong mảng
<code>int IndexOf(object)</code>	Tìm kiếm từ trái sang phải
<code>int LastIndexOf(object)</code>	Tìm kiếm từ phải sang trái
<code>void Insert(index, object)</code>	Chèn một đối tượng tại vị trí
<code>void Remove(object)</code>	Xóa phần tử
<code>void RemoveAt(index)</code>	Xóa phần tử tại vị trí
<code>void RemoveRange(index, count)</code>	Xóa một số phần tử
<code>void Sort()</code>	Sắp xếp tăng dần
<code>int BinarySearch(object)</code>	Tìm kiếm nhị phân
<code>void Reverse()</code>	Đảo
<code>IEnumerator GetIEnumerator</code>	Trả về IEnumerator

- Chú ý: trong List thay object bằng kiểu tương ứng khi tạo List

- Ví dụ: sử dụng ArrayList

```
ArrayList arrInt =new ArrayList();
for (int i = 0; i < 10; i++)
    arrInt.Add(i + 1);
arrInt.Remove(3);
arrInt.RemoveAt(5);
arrInt.Insert(2, 20);
string s = "";
foreach (int n in arrInt)
    s += n.ToString() + "\t";
MessageBox.Show(s);
```



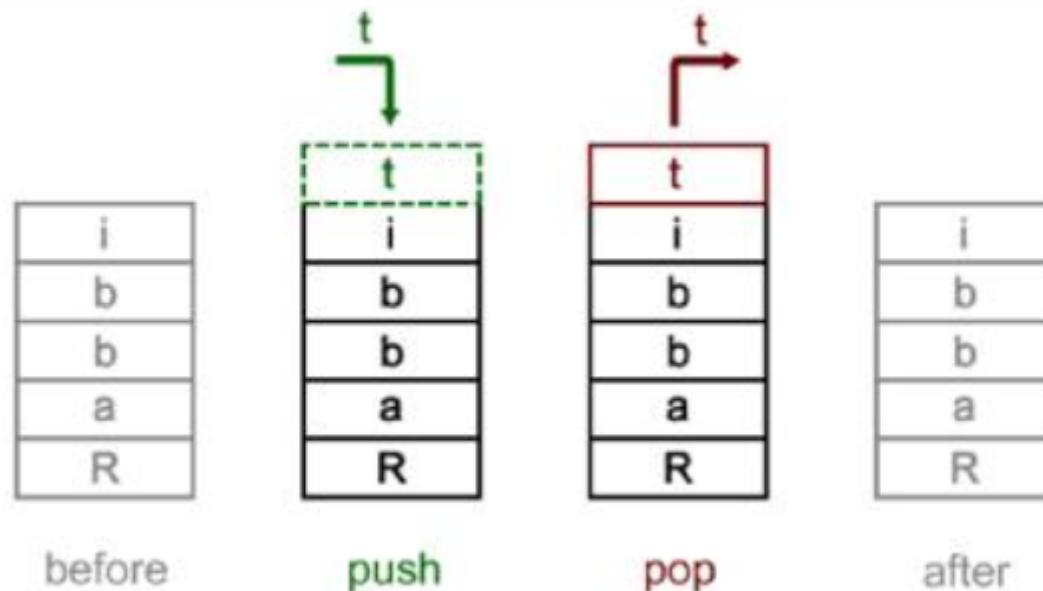
1. Là lớp danh sách kiểu từ điển.
2. Mỗi phần tử được xác định thông qua hai trường Key và Value.
3. Các phần tử chứa trong SortedList sẽ tự động được xếp thứ tự dựa vào trường Key.

### 1. SortedList

- Là lớp danh sách kiểu từ điển.
- Mỗi phần tử được xác định thông qua hai trường Key và Value.
- Các phần tử chứa trong SortedList sẽ tự động được xếp thứ tự dựa vào trường Key.

### 2. HashTable: Tương tự SortedList, nhưng các phần tử không được tự sắp xếp.

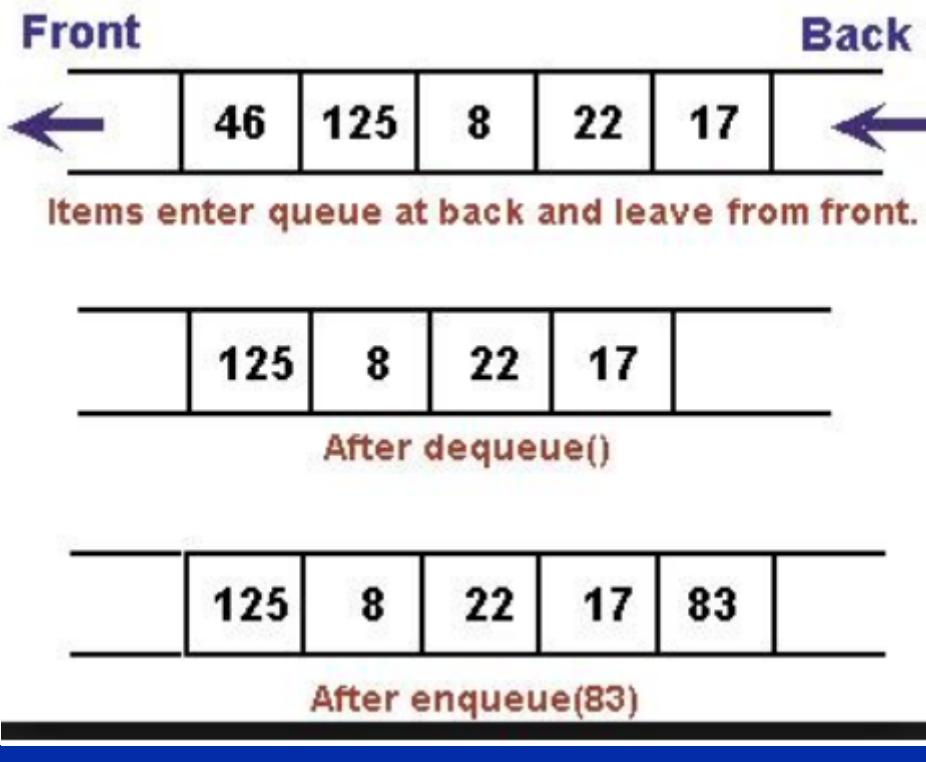
1. Là loại danh sách mà các phần tử được tổ chức theo một ngăn xếp theo thứ tự LIFO (Last In First Out)
2. Thao tác thêm phần tử và lấy phần tử đều thực hiện tại một đầu của danh sách.



- Stack hiện thực collection LIFO
  - Phương thức
    - **void Push(object)** : Thêm vào stack
    - **object Pop()** : Lấy ra khỏi stack
    - **object Peek()** : Trả về phần tử trên đỉnh stack (nhưng không xóa khỏi stack)
  - Property:
    - **int Count**
  - Phương thức khác: **Peek()**, **Clear()**, **Clone()**, **Contains()**, **CopyTo()**, **GetEnumerator()**, ...

```
Stack stList = new Stack();
foreach(int number in new int[]{1,2,3,4})
{
    stList.Push(number);          thêm phần tử vào stack
}
foreach(int number in stList )
{
    // process number;          duyệt stack
}
while(stList.Count != 0)          làm rỗng stack
{
    int number = (int)stList.Pop(); Pop()
}
```

1. Là loại danh sách mà các phần tử được tổ chức theo một ngăn xếp theo thứ tự FIFO (First In First Out)
2. Các phần tử thêm vào từ cuối danh sách và lấy ra từ đầu danh sách.



- Queue hiện thực collection FIFO
  - Phương thức
    - **void Enqueue (object)** : Thêm vào queue
    - **object Dequeue ()** Lấy ra khỏi queue
    - **object Peek ()** : Trả về phần tử đầu queue (nhưng không xóa khỏi queue)
  - Property:
    - **int Count**
  - Phương thức khác: **Clear()**, **Clone()**, **Contains()**, **CopyTo()**, **GetEnumerator()**, ...

```
string str = "";
Queue qList = new Queue();
foreach(int number in new int[]{1,2,3,4})
{
    qList.Enqueue(number);          thêm vào hàng đợi
}
foreach (int number in qList)      duyệt hàng đợi
{
    str += number.ToString() + "\t";
}
while (qList.Count != 0)          làm rỗng hàng đợi
{
    int number = (int)qList.Dequeue(); 
}
```



# Duyệt các phần tử Collection

```
ArrayList a = new ArrayList();
a.Add(1); a.Add(5); a.Add(3);

foreach (int x in a)
{
    Console.WriteLine(x);
}
```

```
ArrayList a = new ArrayList();
a.Add(1); a.Add(5); a.Add(3);

IEnumerator ie = a.GetEnumerator();
while (ie.MoveNext())
{
    int x = (int)ie.Current;
    Console.WriteLine (x);
}
```

- Câu lệnh foreach làm việc trên collection mà collection thỏa quy tắc sau
  - Hoặc hiện thực interface **IEnumerable**
  - Hoặc hiện thực theo collection pattern:
    - Collection phải có phương thức public **GetEnumerator()** không tham số và trả về kiểu struct, class hay interface
    - Kiểu trả về của phương thức **GetEnumerator()** phải chứa
      - Phương thức public **MoveNext()** không tham số, kiểu trả về bool
      - Phương thức public **Current** không tham số, kiểu trả về kiểu tham chiếu



# Collections

## Tự Tạo collection

```
class MyCollectionClass: IEnumerable
{ ... }

foreach (string s in myCollection)
{
    Console.WriteLine("String is {0}", s);
}
```

```
IEnumerator ie = (IEnumerable) myCollection;
IEnumerator e = ie. GetEnumerator();

while (e.MoveNext())
{
    string s = (string) e.Current;
    Console.WriteLine("String is {0}", s);
}
```



# Collections

## Tự Tạo collection

```
public interface IEnumerable
{
    IEnumerator GetEnumerator();
}
```

```
public interface IEnumerator
{
    object Current { get; }
    bool MoveNext();
    void Reset();
}
```



# Collections

## Tự Tạo collection

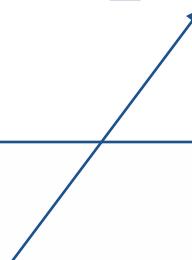
```
class MyCollectionClass : IEnumerable
{
    CollectionPart[] _parts;
    ...
    ...
    public IEnumerator GetEnumerator()
    {
    }
}
```



# Collections

## Tự Tạo collection

```
class MyCollectionClass : IEnumerable
{
    CollectionPart[] _parts;
    ...
    ...
    public IEnumerator GetEnumerator()
    {
        return _parts.GetEnumerator()
    }
}
```



Vì `_parts` là `System.Array`, là một kiểu liệt kê nên  
nó có phương thức `GetEnumerator()` trả về  
`IEnumerator`

- Giải pháp : tiếp cận dựa trên mẫu
  - Trình biên dịch C# compiler tìm kiếm:
    - GetEnumerator() trên collection
    - bool MoveNext() trên kiểu enumerator
    - Current trên kiểu enumerator

- Trong .NET, chuỗi là kiểu dữ liệu tham chiếu, mỗi chuỗi là một đối tượng của lớp **string**
- .Net Framework System.String



## ■ Các ký hiệu đặc biệt: \ và @

Escape Sequence	Character Produced	Unicode Value of Character
\'	Single quotation mark	0x0027
\"	Double quotation mark	0x0022
\\"	Backslash	0x005C
\0	Null	0x0000
\a	Alert (causes a beep)	0x0007
\b	Backspace	0x0008
\f	Form feed	0x000C
\n	New line	0x000A
\r	Carriage return	0x000D
\t	Horizontal tab	0x0009
\v	Vertical tab	0x000B



# Chuỗi – Phương thức khởi tạo

```
public static void Main( string[] args )
{
    // string initialization
    char[] characterArray =
        { 'b', 'i', 'r', 't', 'h', ' ', 'd', 'a', 'y' };
    string originalString = "Welcome to C# programming!";
    string string1 = originalString;
    string string2 = new string( characterArray );
    string string3 = new string( characterArray, 6, 3 );
    string string4 = new string( 'C', 5 );

    Console.WriteLine( "string1 = " + "\"" + string1 + "\"\n" +
        "string2 = " + "\"" + string2 + "\"\n" +
        "string3 = " + "\"" + string3 + "\"\n" +
        "string4 = " + "\"" + string4 + "\"\n" );
}
```

```
file:///E:/Tailieu/C#2010/VS2010 BO...
string1 = "Welcome to C# programming!"
string2 = "birth day"
string3 = "day"
string4 = "CCCCC"
```



# Chuỗi – Các phương thức

- Phương thức static:
- **public static int Compare**  
( **string strA, string strB** )

Ví dụ:

```
int nRet = string.Compare("A", "a");  
→nRet= 1
```

```
int nRet = string.Compare("a", "A");  
→nRet= -1
```

```
int nRet = string.Compare("a", "a");  
→nRet=0
```



## Chuỗi – Các phương thức

- **public static string Copy(string str)**

```
string str = string.Copy("Teo");
```

```
string str1 = "Ti";
```

```
string str2 = string.Copy(str1);
```



# Chuỗi – Các phương thức

- **public bool Contains( string value )**

```
string strA = "Trần Văn Tèo";
```

```
string strB = "Văn";
```

```
bool bRet = strA.Contains(strB);
```

```
if (!bRet)
```

```
    MessageBox.Show("Không có
```

```
        ["+strB+"] trong ["+strA+"]");
```

```
else
```

```
    MessageBox.Show("Có [" + strB + "] trong [“
```

```
        + strA + ”]");
```

- **public static string Format**

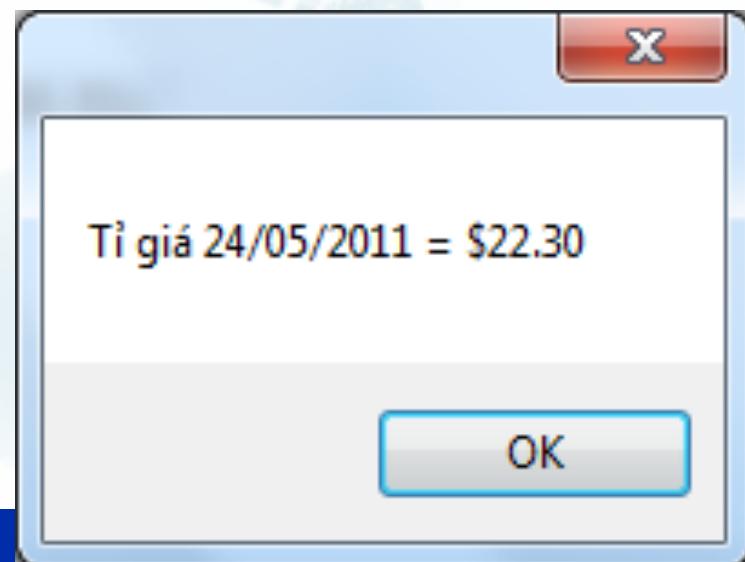
(string format, params Object[]

args )

string strFormat;

strFormat = string.Format("Tỉ giá {0:d} = {1:c}",  
DateTime.Today, 22.3);

MessageBox.Show(strFormat);



Ký tự	Mô tả	Ví dụ	Kết quả
C hoặc c	Tiền tệ (Currency)	string.Format("{0:C}", 2.5); string.Format("{0:C}", -2.5);	\$2.50 (\$2.50)
D hoặc d	Decimal	string.Format("{0:D5}", 25);	00025
E hoặc e	Khoa học (Scientific)	string.Format("{0:E}", 250000);	2.50000E+005
F hoặc f	Có định phần thập phân (Fixed-point)	string.Format("{0:F2}", 25); string.Format("{0:F0}", 25);	25.00 25
G hoặc g	General	string.Format("{0:G}", 2.5);	2.5
N hoặc n	Số (Number)	string.Format("{0:N}", 2500000);	2,500,000.00
X hoặc x	Hệ số 16 (Hexadecimal)	string.Format("{0:X}", 250); string.Format("{0:X}", 0xffff);	FA FFFF



# Chuỗi – Các phương thức

- Phương thức thành viên của lớp:

**public int CompareTo( string strB )**

```
string strA = "A";
string strB = "B";                                →nRet=-1
int nRet = strA.CompareTo(strB);
```

```
string strA = "A";
string strB = null;                               →nRet=1
int nRet = strA.CompareTo(strB);
```



# Chuỗi – Các phương thức

**public bool EndsWith( string value )**

**Ví dụ:**string strA = "<div>Tèo </div>";

string strB = "</div>";

bool bRet = strA.**EndsWith**(strB);

**if** (bRet)

    MessageBox.Show("có [" + strB+] ở cuối chuỗi");

**else**

    MessageBox.Show("Ko có [" + strB + "] ở cuối  
chuỗi");



# Chuỗi – Các phương thức

```
public bool EndsWith( string value,  
 StringComparison comparType )  
  
string strA = "<div><div>Tèo</div></div>";  
string strB = "</DIV>";  
  
bool bRet = strA.EndsWith(strB,  
 StringComparison.OrdinalIgnoreCase);  
  
if (bRet)  
    MessageBox.Show("có [" + strB + "] ở cuối chuỗi");  
else  
    MessageBox.Show("Ko có [" + strB + "] ở cuối chuỗi");
```



# Chuỗi – Các phương thức

## **public bool StartsWith ( string value )**

Ví dụ:

```
string str = "<head>Học lập trình </head>";
```

```
bool bStart=str.StartsWith("<head>");
```

```
MessageBox.Show(bStart.ToString());
```

→ True

```
bStart=str.StartsWith("<body>");
```

```
MessageBox.Show(bStart.ToString());
```

→ False



# Chuỗi – Các phương thức

```
public bool StartsWith( string value,  
StringComparison comparType )
```

Ví dụ:

```
string str = "<head>Học lập trình </head>";  
bool bStart=str.StartsWith("<HEAD>",  
StringComparison.OrdinalIgnoreCase);
```

```
MessageBox.Show(bStart.ToString());
```

→ False

```
bool bStart=str.StartsWith("<HEAD>",
```

```
StringComparison.OrdinalIgnoreCaseIgnoreCase);
```

→ True



## Chuỗi – Các phương thức

**public int IndexOfAny( char[] anyOf )**

**Ví dụ:**

**string str = "Không! không; có? có?";**

**int nRet=str.IndexOfAny(**new char[]****

**{ '>', '?', ';' } );**

**➔ nRet=12**



# Chuỗi – Các phương thức

**public string Insert( int startIndex, string value )**

**Ví dụ:**

**string str = "Không sợ , chỉ sợ không công bằng";**

**str=str.Insert(9, "thiếu");**

**→string str = "Không sợ thiếu, chỉ sợ không công bằng";**



# Chuỗi – Các phương thức

**public int LastIndexOf( char value )**

Ví dụ:

```
string str = "a , b, c; d; e; f, k";
```

```
int nRet=str.LastIndexOf('');
```

→ nRet=17

```
nRet=str.LastIndexOf('!');
```

→ nRet=-1



## Chuỗi – Các phương thức

**public int LastIndexOf(string value)**

Ví dụ:

```
string str = "Học ! Học Nữa ! Học Mai";
```

```
int nRet = str.LastIndexOf("Học");
```

→ nRet = 16

```
nRet = str.LastIndexOf("HỌC");
```

→ nRet = -1



## Chuỗi – Các phương thức

```
public int LastIndexOf( string value,  
                      StringComparison comparisonType )
```

Ví dụ:

```
string str = "Học ! Học Nữa ! Học Mãi";  
int nRet = str.LastIndexOf("Học",  
                           StringComparison.OrdinalIgnoreCase);
```

→ nRet = 16



# Chuỗi – Các phương thức

**public int Remove( int startIndex )**

**Ví dụ:**

string str = "không có gì quý hơn độc lập tự do";

str=str.Remove(11);

➔ str = "không có gì"



## Chuỗi – Các phương thức

**public int Remove( int startIndex, int count )**

**Ví dụ:**

string str = "Con đường học vấn giúp ta thoát  
khỏi nghèo khổ nhanh nhất";

str=str.Remove(0,9);

→str = "học vấn giúp ta thoát khỏi nghèo khổ  
nhanh nhất";



## Chuỗi – Các phương thức

**public int Replace( char oldChar, char newChar)**

**Ví dụ:**

**string str = "Chúc các em học tốt";**

**str = str.Replace(' ','#');**

**➔ str = "Chúc#các#em#học#tốt";**



## Chuỗi – Các phương thức

```
public int Replace(string oldValue, string  
newValue)
```

Ví dụ:

```
string str = "Phải xem tài liệu trước khi tới lớp,  
tài liệu này rất quan trọng!";
```

```
str = str.Replace("tài liệu","document");
```

→ str = "Phải xem **document** trước khi tới lớp,  
**document** này rất quan trọng!";



# Chuỗi – Các phương thức

**public string[] Split( params char[] separator)**

**Ví dụ:**

```
string str = "Trần văn An;26 tuổi;Độc thân;Nha Trang";
string []strArr=str.Split(new char[]{';'});
foreach (string s in strArr)
{
    MessageBox.Show (s);
}
```



# Chuỗi – Các phương thức

```
public string[] Split(string[] separator, StringSplitOptions  
options )
```

Ví dụ:

```
string str = "Trần văn An;26 tuổi;Độc thân;Nha Trang";  
string []strArr=str.Split(new string[] { ":" },  
StringSplitOptions.None);  
MessageBox.Show(strArr.Length+ "");  
foreach (string s in strArr)  
{  
    MessageBox.Show(s);  
}
```



# Chuỗi – Các phương thức

```
public string[] Split(string[] separator, StringSplitOptions  
options )
```

Ví dụ:

```
string str = "Trần văn An;26 tuổi;Độc thân;Nha Trang";  
string []strArr=str.Split(new string[] { ":" },  
StringSplitOptions.RemoveEmptyEntries);  
MessageBox.Show(strArr.Length+ "");  
foreach (string s in strArr)  
{  
    MessageBox.Show(s);  
}
```



# Chuỗi – Các phương thức

public string **Substring( int startIndex )**

**Ví dụ:**

string str = "Thay đổi để tồn tại";

string strSub = str.**Substring(5);**

MessageBox.Show(strSub);

→đổi để tồn tại



# Chuỗi – Các phương thức

```
public string Substring(int startIndex, int length)
```

Ví dụ:

```
string str = "Microsoft Visual Studio 2015";  
string strSub = str.Substring(10,13);  
MessageBox.Show(strSub);
```

→ **Visual Studio**

```
strSub = str.Substring(0,9);
```

→ **Microsoft**



# Chuỗi – Các phương thức

**public char[] ToCharArray()**

**Ví dụ:**

```
string str = "Microsoft Visual Studio 2015";
char []chrArr=str.ToArray();
MessageBox.Show(chrArr.Length+""); →28
foreach (char c in chrArr)
{
    MessageBox.Show(c.ToString());
}
```



## Chuỗi – Các phương thức

public string **ToLower()**

**Ví dụ:**

```
string str = "Microsoft Visual Studio 2015";
str=str.ToLower();
MessageBox.Show(str);
→microsoft visual studio 2015
```

public string **ToUpper()**



# Chuỗi – Các phương thức

public string **Trim()**

**Ví dụ:**

```
string str = " Microsoft Visual Studio 2015 ";
```

```
MessageBox.Show(str.Length+""");
```

```
MessageBox.Show(str);
```

→ 35

→ " Microsoft Visual Studio 2015 "

```
string str = " Microsoft Visual Studio 2015 ";
```

```
str=str.Trim();
```

```
MessageBox.Show(str.Length+""");
```

```
MessageBox.Show(str);
```

→ 28

→ str = "Microsoft Visual Studio 2015";



# Chuỗi – Các phương thức

```
public string TrimEnd( params char[]  
trimChars )
```

Ví dụ:

```
string str = "Microsoft Visual Studio 2015 #?! #";  
str=str.TrimEnd(new char[] { '?','!', '#' });    →28  
MessageBox.Show(str.Length+""");  
MessageBox.Show(str);  
→"Microsoft Visual Studio 2015"
```



# Chuỗi – Các phương thức

```
public string TrimStart ( params char[]  
trimChars )
```

Ví dụ:

```
string str = "#?!" #Microsoft Visual Studio 2015;  
str=str.TrimStart(new char[] { ',', '?', '!', '#' }); →28  
MessageBox.Show(str.Length+""");  
MessageBox.Show(str);  
→"Microsoft Visual Studio 2015"
```

- Lớp StringBuilder được sử dụng để tạo ra và bổ sung các chuỗi.
- Dữ liệu lớp StringBuilder có thể thay đổi dạng động.
  - Khi chúng ta bổ sung một đối tượng StringBuilder thì chúng ta đã làm thay đổi trên giá trị thật của chuỗi, chứ không phải trên bản sao



# StringBuilder - Phương thức

System.StringBuilder	
Phương thức	Ý nghĩa
Capacity()	Truy cập hay gán một số ký tự mà StringBuilder nắm giữ.
Chars()	Chỉ mục.
Length()	Thiết lập hay truy cập chiều dài của chuỗi
MaxCapacity()	Truy cập dung lượng lớn nhất của StringBuilder
Append()	Nối một kiểu đối tượng vào cuối của StringBuilder
AppendFormat()	Thay thế định dạng xác định bằng giá trị được định dạng của một đối tượng.
EnsureCapacity()	Đảm bảo rằng StringBuilder hiện thời có khả năng tối thiểu lớn như một giá trị xác định.
Insert()	Chèn một đối tượng vào một vị trí xác định
Replace()	Thay thế tất cả thể hiện của một ký tự xác định với những ký tự mới.



# Thao tác với lớp StringBuilder

- EnsureCapacity: Thay đổi kích thước chuỗi khi thực thi

```
StringBuilder buffer = new StringBuilder( "Hello, how are you?" );
// use Length and Capacity properties
Console.WriteLine( "buffer = " + buffer +
    "\nLength = " + buffer.Length +
    "\nCapacity = " + buffer.Capacity );
buffer.EnsureCapacity( 75 ); // ensure a capacity of at least 75
Console.WriteLine( "\nNew capacity = " +
    buffer.Capacity );
// truncate StringBuilder by setting Length property
buffer.Length = 10;
Console.Write( "\nNew length = " +
    buffer.Length + "\nbuffer = " );
// use StringBuilder indexer
for ( int i = 0; i < buffer.Length; i++ )
    Console.Write( buffer[ i ] );
```

## 1. Tạo màn hình đếm số từ như sau



- Xây dựng từ điển đơn giản như sau:

- Khi nhập vào combobox 1 từ cần tra thì chương trình sẽ dò tìm đến chữ nào khớp với ký tự gần nhất
- Khi nhấn nút Enter hoặc Doubleclick vào từ cần tra thì nghĩa tương ứng của từ sẽ hiển thị vào textbox bên phải tương ứng.
- Danh sách các từ lưu sẵn vào object (word) -> ArrayList

