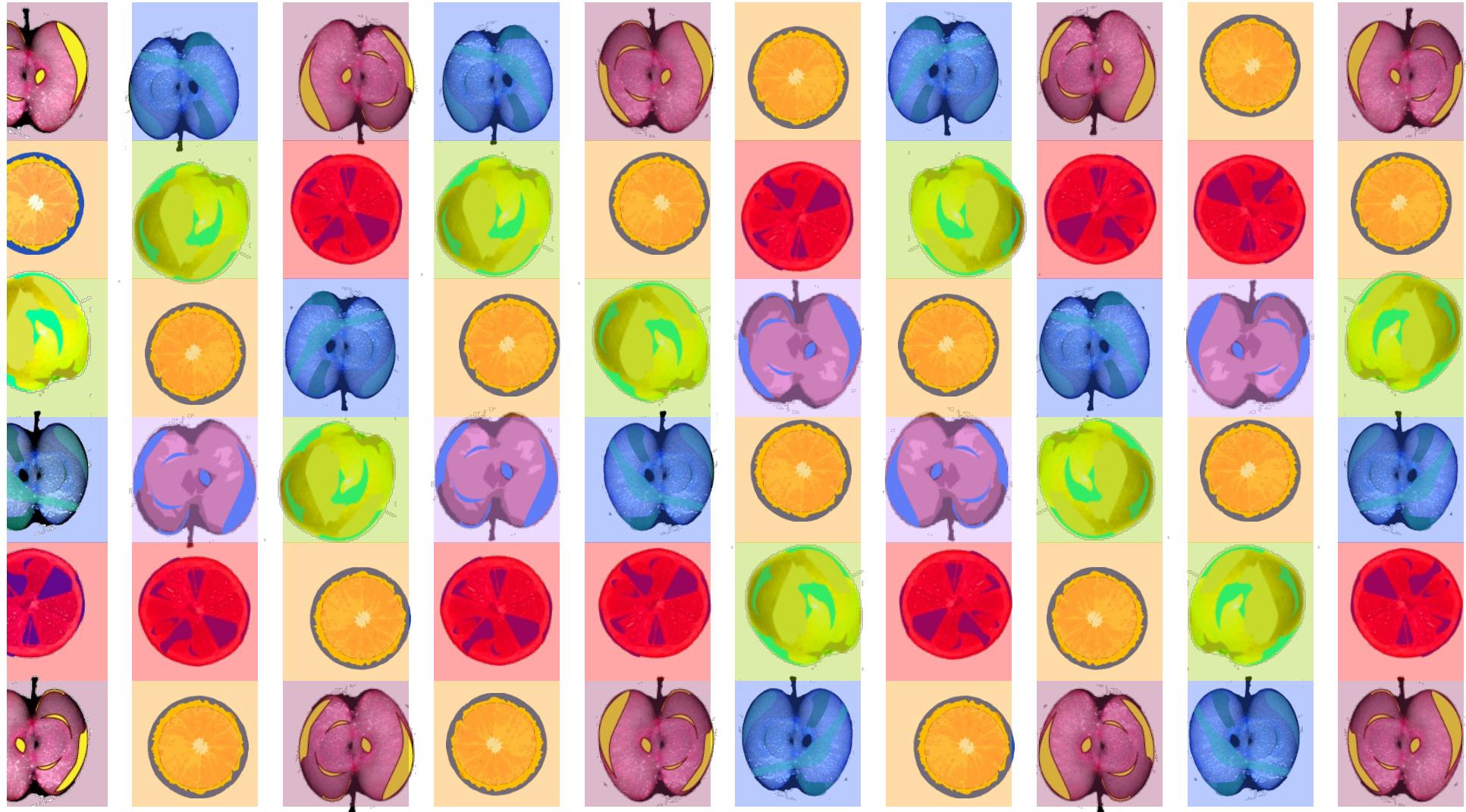


C# CĂN BẢN (Phần 2)

(TS. Trần Anh Tuấn)

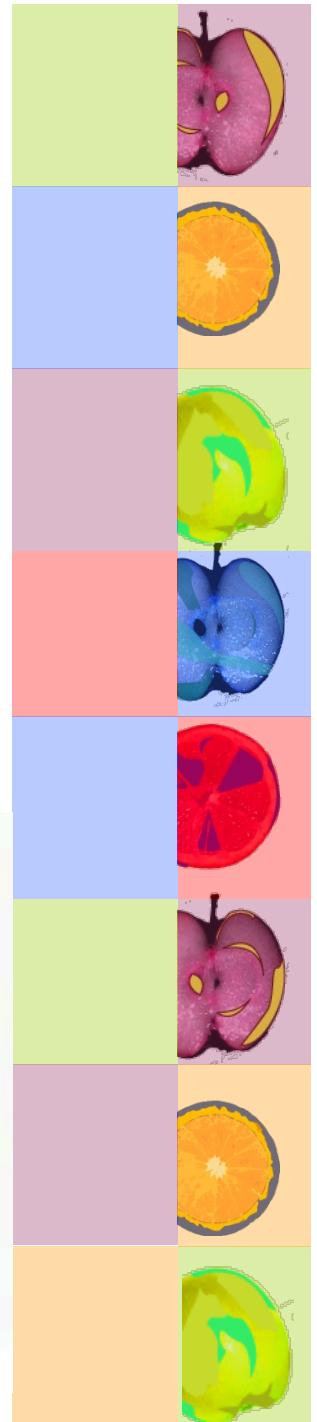


Hàm ToString()

- Tất cả các object trong C# đều có phương thức ToString() để biểu diễn object đó dưới dạng chuỗi
- Khi tạo một class, phương thức ToString() nên được override để đảm bảo xuất đúng thông tin mà người dùng mong muốn

C#

```
int x = 42;
string strx = x.ToString();
Console.WriteLine(strx);
// Output:
// 42
```



Hàm ToString()

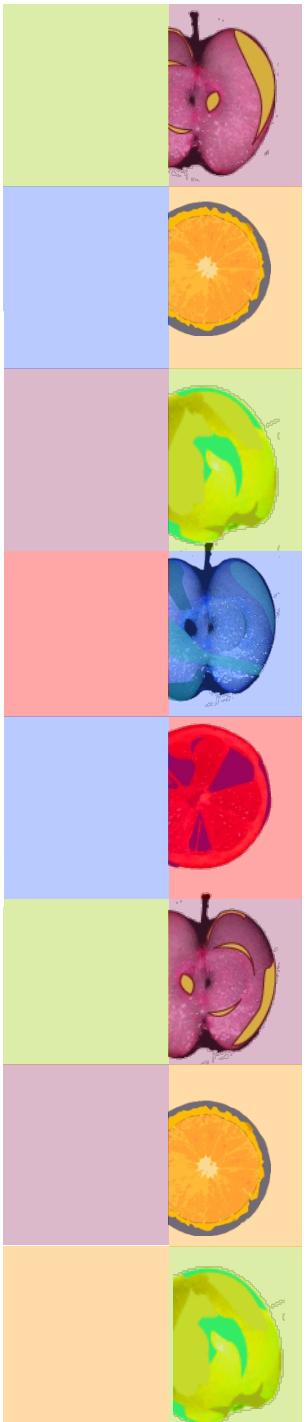
C#

```
class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

    public override string ToString()
    {
        return "Person: " + Name + " " + Age;
    }
}
```

C#

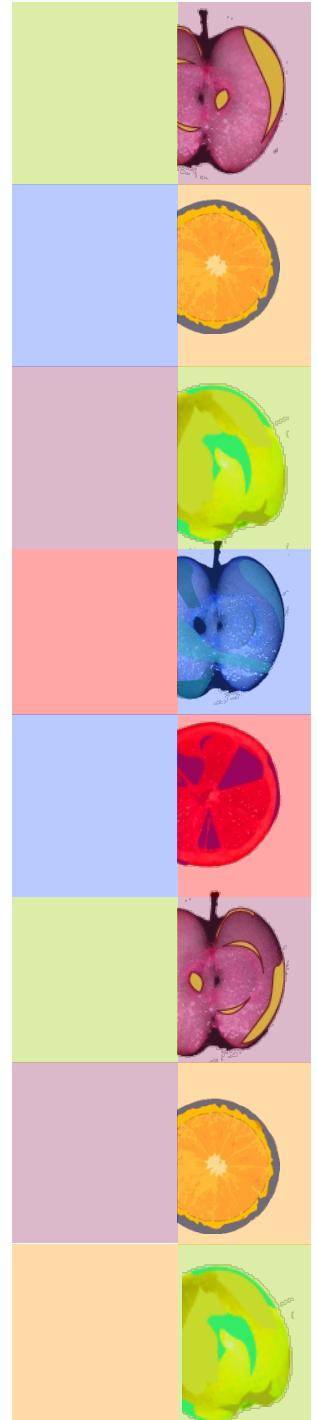
```
Person person = new Person { Name = "John", Age = 12 };
Console.WriteLine(person);
// Output:
// Person: John 12
```



Interface

- Một interface như là một “hợp đồng” mà tất cả các lớp nếu kế thừa từ nó phải tuân theo
- Interface có thể chứa thuộc tính, phương thức hay sự kiện nhưng chỉ ở mức khai báo, trách nhiệm của các lớp dẫn xuất interface đó phải định nghĩa chúng cụ thể lại

```
public interface ITransactions
{
    // interface members
    void showTransaction();
    double getAmount();
}
```



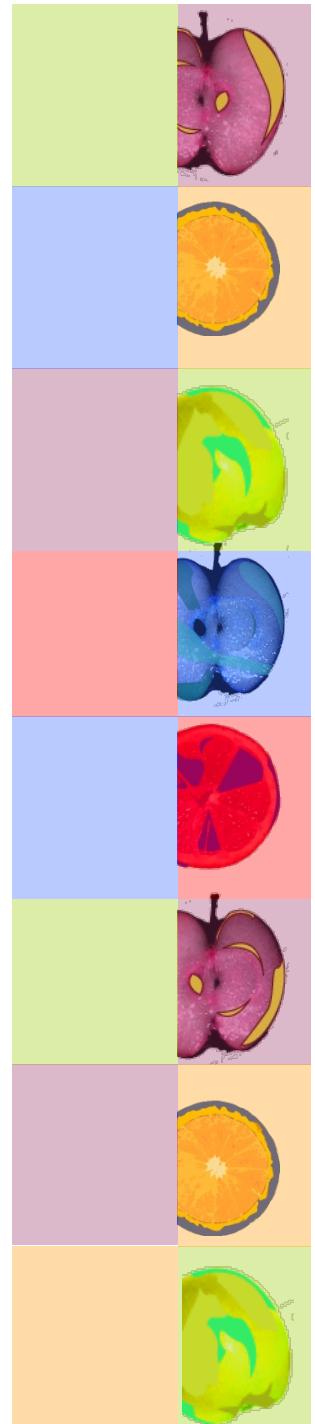
Interface

```
public class Transaction : ITransactions
{
    private string tCode;
    private string date;
    private double amount;
    public Transaction()
    {
        tCode = " ";
        date = " ";
        amount = 0.0;
    }

    public Transaction(string c, string d, double a)
    {
        tCode = c;
        date = d;
        amount = a;
    }

    public double getAmount()
    {
        return amount;
    }

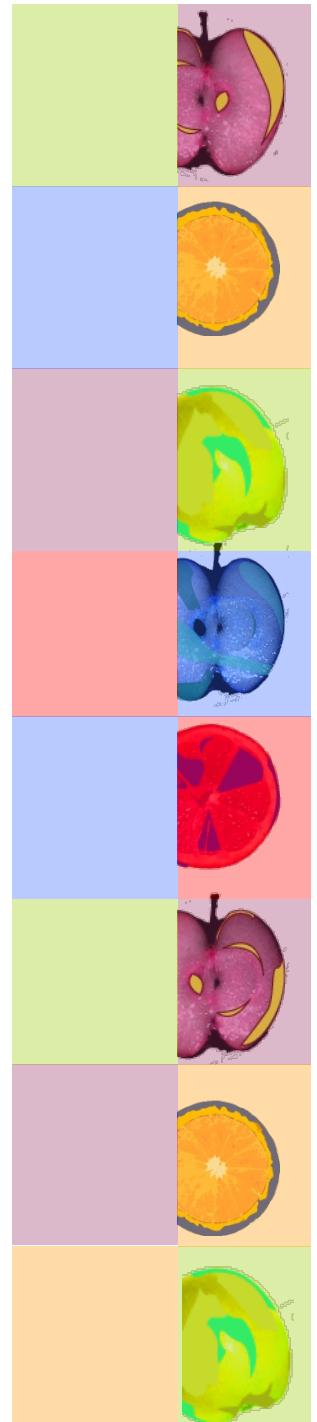
    public void showTransaction()
    {
        Console.WriteLine("Transaction: {0}", tCode);
        Console.WriteLine("Date: {0}", date);
        Console.WriteLine("Amount: {0}", getAmount());
    }
}
```



Interface

```
static void Main(string[] args)
{
    Transaction t1 = new Transaction("001", "8/10/2012", 78900.00);
    Transaction t2 = new Transaction("002", "9/10/2012", 451900.00);
    t1.showTransaction();
    t2.showTransaction();
    Console.ReadKey();
}
```

```
Transaction: 001
Date: 8/10/2012
Amount: 78900
Transaction: 002
Date: 9/10/2012
Amount: 451900
```



Interface

```
interface IControl
{
    void Paint();
}
interface ISurface
{
    void Paint();
}
```

```
public class SampleClass : IControl, ISurface
{
    void IControl.Paint()
    {
        System.Console.WriteLine("IControl.Paint");
    }
    void ISurface.Paint()
    {
        System.Console.WriteLine("ISurface.Paint");
    }
}
```

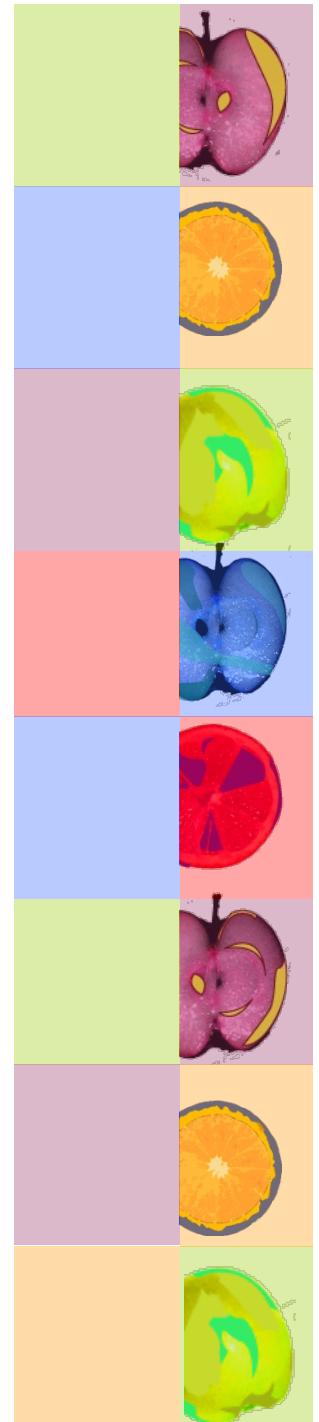
```
// Call the Paint methods from Main.

SampleClass obj = new SampleClass();
//obj.Paint(); // Compiler error.

IControl c = (IControl)obj;
c.Paint(); // Calls IControl.Paint on SampleClass.

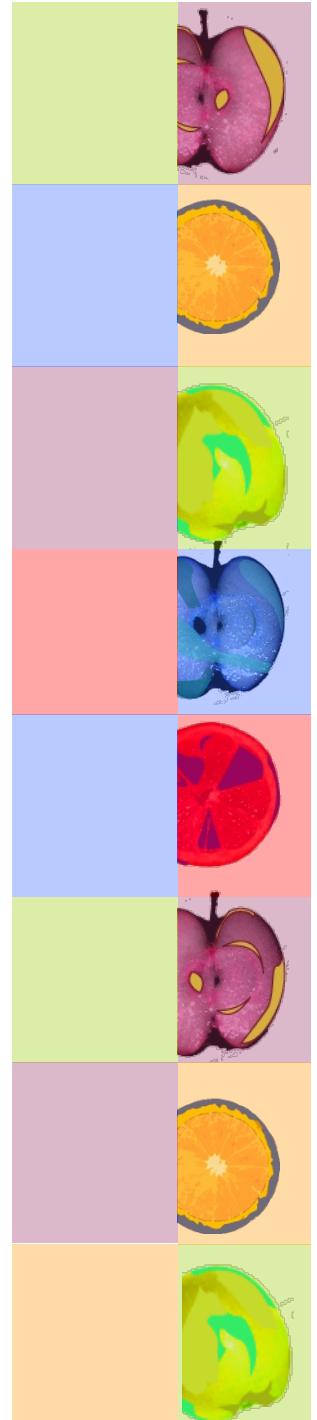
ISurface s = (ISurface)obj;
s.Paint(); // Calls ISurface.Paint on SampleClass.

// Output:
// IControl.Paint
// ISurface.Paint
```



Thuộc tính của lớp

- Thuộc tính - Property là các thành viên được đặt tên của các lớp, cấu trúc, và Interface.
- Trong C#, accessor là một thuộc tính chứa các lệnh có thể thực thi, mà giúp đỡ trong việc lấy (đọc hoặc tính toán) hoặc thiết lập (ghi) thuộc tính.
- Các khai báo accessor có thể thu được một get accessor, một set accessor, hoặc cả hai.



```
class Student
{
    private string code = "N/A";
    private string name = "unknown";
    private int age = 0;

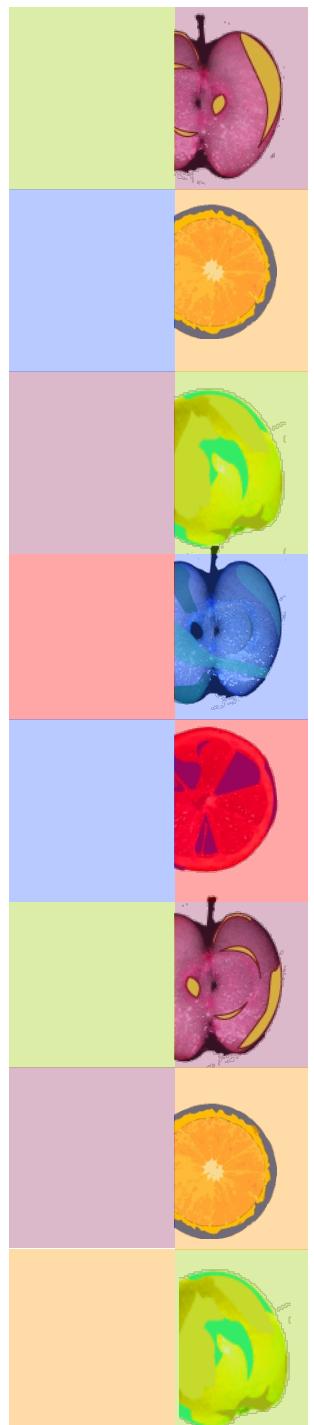
    // khai bao thuoc tinh Code co kieu string:
    public string Code
    {
        get
        {
            return code;
        }
        set
        {
            code = value;
        }
    }

    // khai bao thuoc tinh Age co kieu int:
    public int Age
    {
        get
        {
            return age;
        }
        set
        {
            age = value;
        }
    }
}
```

```
// khai bao thuoc tinh Name co kieu string:
public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}

public override string ToString()
{
    return "MSSV = " + Code + ", Ho Ten = " + Name + ", Tuoi = " + Age;
}
```

```
// khai bao thuoc tinh Name co kieu string:
public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}
```



Thuộc tính của lớp

```
static void Main(string[] args)
{
    Console.WriteLine("Thuoc tinh (Property) trong C#");
    Console.WriteLine("-----");

    // tao mot doi tuong Student
    Student s = new Student();

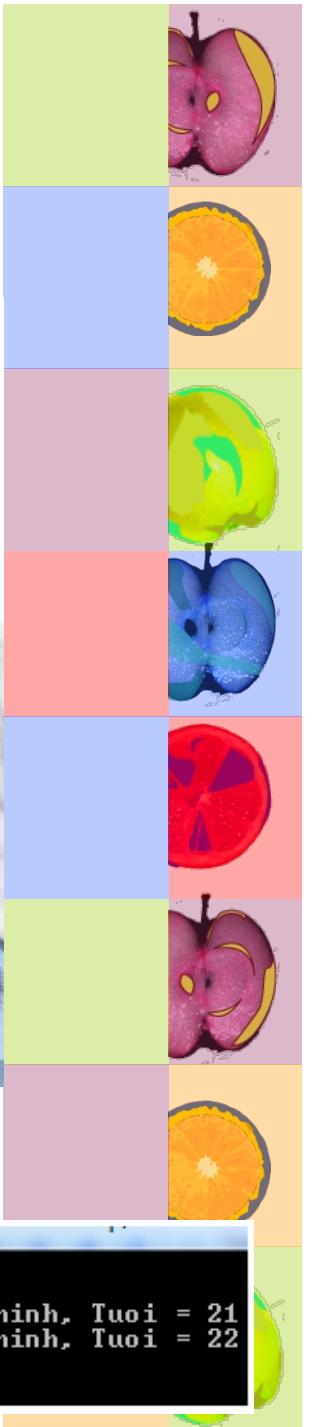
    // thiet lap cac thuoc tinh code, name va age cho Student
    s.Code = "001";
    s.Name = "Minh Chinh";
    s.Age = 21;
    Console.WriteLine("Thong tin sinh vien: {0}", s);

    //bay gio tang age them 1
    s.Age += 1;
    Console.WriteLine("Thong tin sinh vien: {0}", s);

    Console.ReadLine();
    Console.ReadKey();
}
```

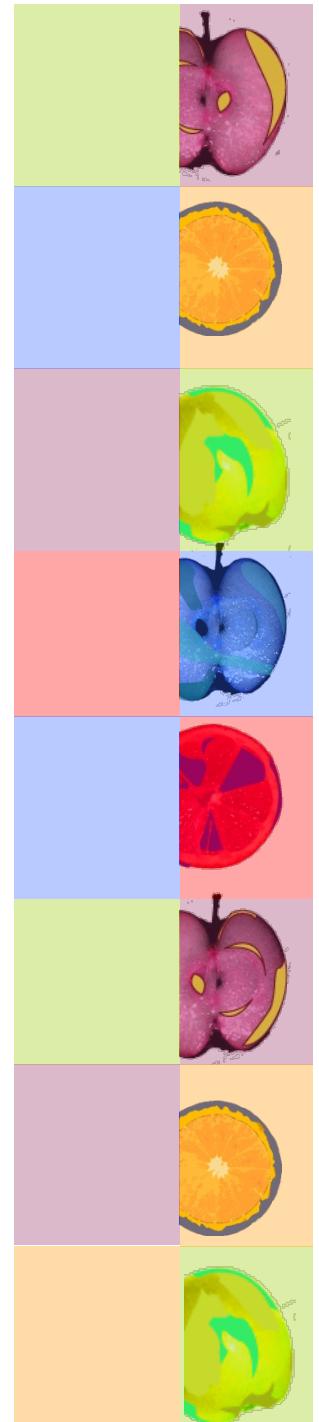


```
Thuoc tinh (Property) trong C#
-----
Thong tin sinh vien: MSSV = 001, Ho Ten = Minh Chinh, Tuoi = 21
Thong tin sinh vien: MSSV = 001, Ho Ten = Minh Chinh, Tuoi = 22
```



Indexer

- Một **indexer** trong C# cho phép một đối tượng để được lập chỉ mục, ví dụ như một mảng.
- Khi bạn định nghĩa một indexer cho một lớp, thì lớp này vận hành tương tự như một **virtual array (mảng ảo)**. Sau đó, bạn có thể truy cập instance (sự thể hiện) của lớp này bởi sử dụng toán tử truy cập mảng trong C# là ([]).

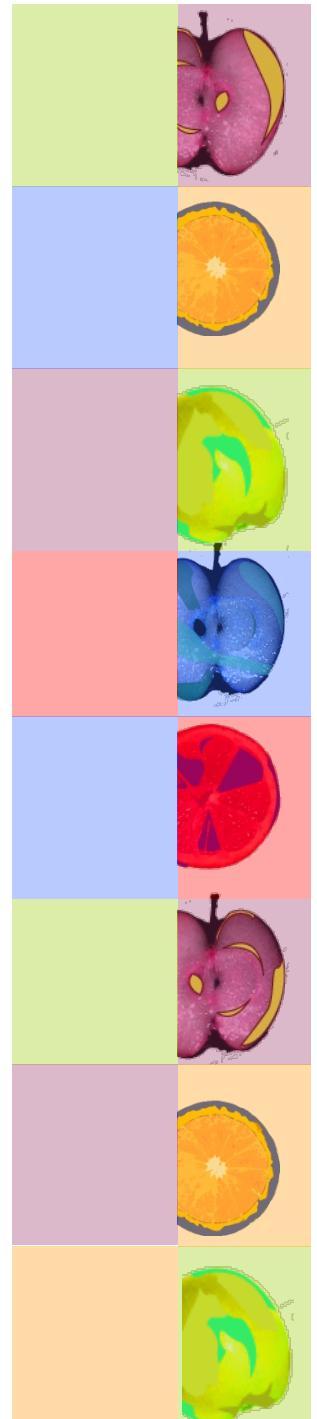


Indexer

- Cú pháp:

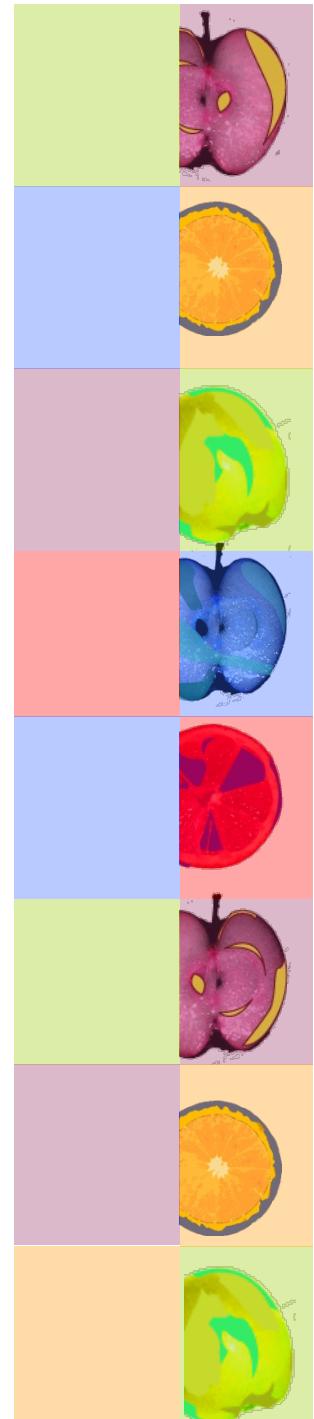
```
kiểu_phần_tử this[int index]
{
    get
    {
        // trả về giá trị được xác định bởi index
    }

    set
    {
        // thiết lập giá trị được xác định bởi index
    }
}
```



Indexer

- Việc khai báo hành vi của một Indexer là tương tự như một thuộc tính. Bạn sử dụng **get** accessor và **set** accessor để định nghĩa một Indexer.
- Indexer trả về hoặc thiết lập một giá trị cụ thể từ instance của đối tượng. Nói cách khác, nó chia dữ liệu của instance thành các phần nhỏ hơn và đánh chỉ mục mỗi phần, lấy hoặc thiết lập mỗi phần.



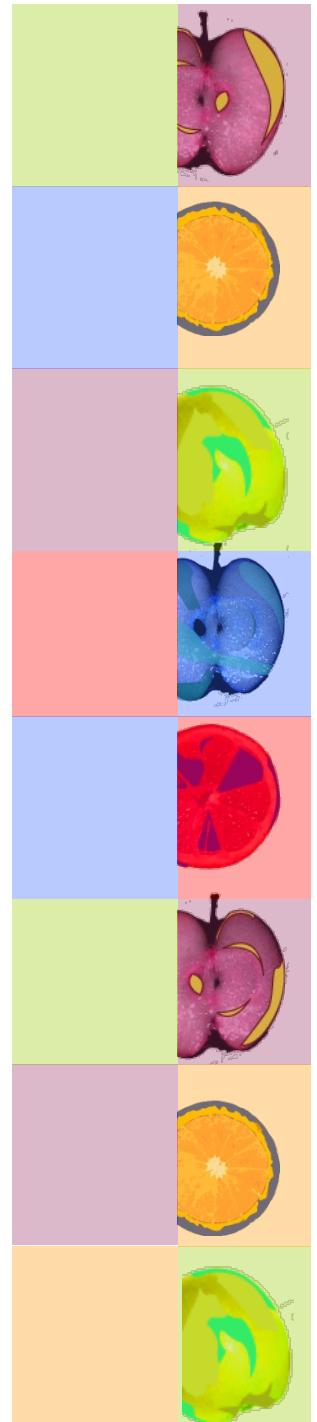
Indexer

```
private string[] namelist = new string[size];
static public int size = 10;
public TestCsharp()
{
    for (int i = 0; i < size; i++)
    {
        namelist[i] = "N/A";
    }
}

public int this[string name]
{
    get
    {
        int index = 0;
        while (index < size)
        {
            if (namelist[index] == name)
            {
                return index;
            }
            index++;
        }
        return index;
    }
}

public string this[int index]
{
    get
    {
        string tmp;

        if (index >= 0 && index <= size - 1)
        {
            tmp = namelist[index];
        }
        else
        {
            tmp = "";
        }
        return (tmp);
    }
    set
    {
        if (index >= 0 && index <= size - 1)
        {
            namelist[index] = value;
        }
    }
}
```



Indexer

```
static void Main(string[] args)
{
    Console.WriteLine("Vi du minh hoa Indexer trong C#");
    Console.WriteLine("-----");
    //tao doi tuong TestCsharp
    TestCsharp names = new TestCsharp();
    names[0] = "Nam";
    names[1] = "Hoang";
    names[2] = "Dung";
    names[3] = "Hue";
    names[4] = "Huong";
    names[5] = "Phuc";
    names[6] = "Trung";

    //su dung indexer thu nhat voi tham so int
    for (int i = 0; i < TestCsharp.size; i++)
    {
        Console.WriteLine(names[i]);
    }

    //su dung indexer thu nhat voi tham so string
    Console.WriteLine(names["Hue"]);
    Console.ReadKey();
}
```

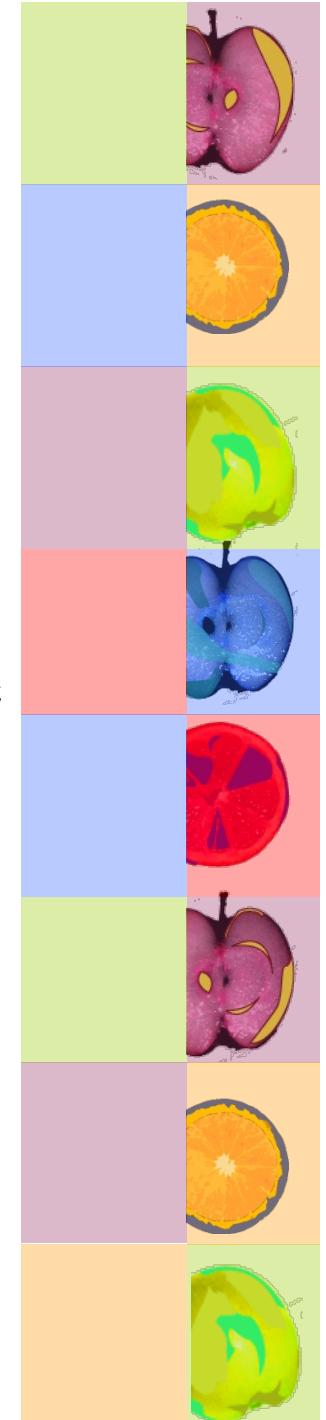
Vi du minh hoa Indexer trong C#

```
Nam
Hoang
Dung
Hue
Huong
Phuc
Trung
N/A
N/A
N/A
3
```



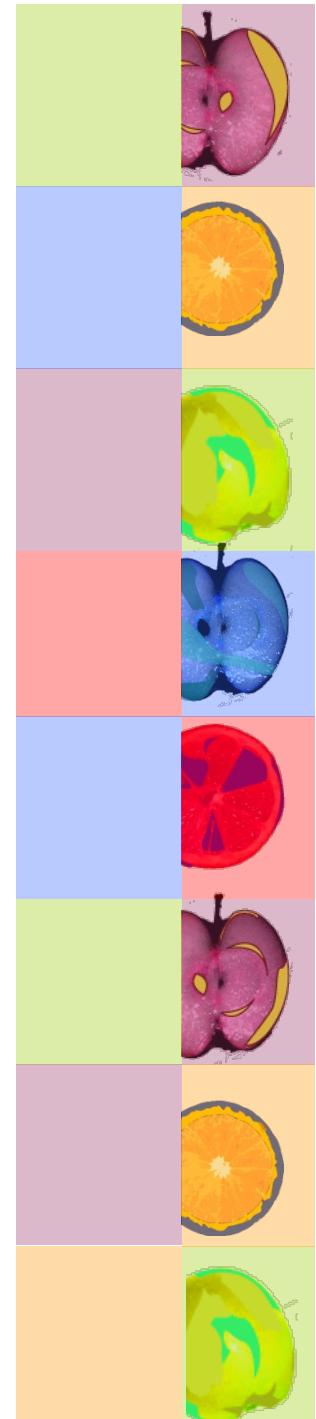
Attribute

- Attribute là một cơ chế được dùng để định nghĩa và khai báo các thông tin cần thiết, nhằm bổ sung và hỗ trợ khá nhiều chức năng liên quan đến việc soạn thảo, debug, cũng như biên dịch các chương trình.
- Các attribute được sử dụng bằng cách đặt phía trên các thành phần (assembly, lớp, phương thức, property...) với cặp ngoặc vuông [] bao lại. Nhiều attribute có thể được sử dụng trong cùng một cặp ngoặc vuông này và được ngăn cách nhau bởi dấu phẩy ,.



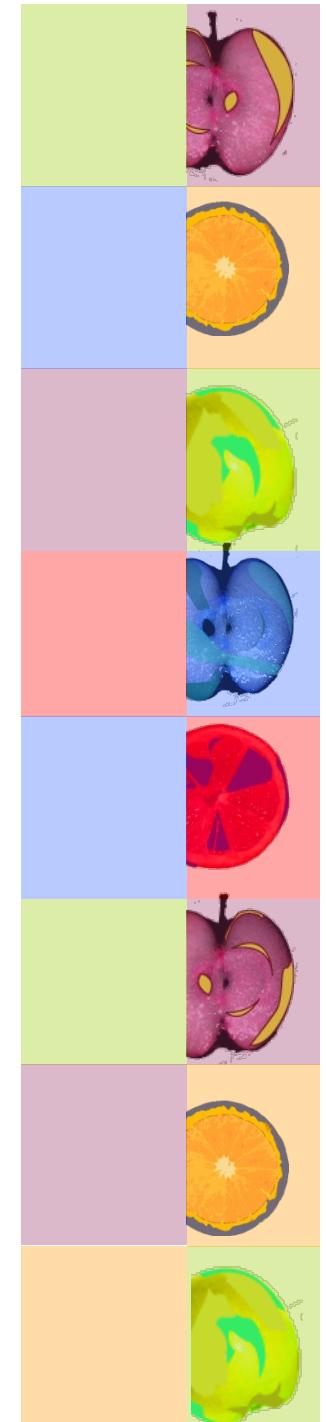
Obsolete attribute

- Attribute này chỉ có một mục đích duy nhất là đánh dấu các phương thức hoặc thành phần nào đó không nên được sử dụng nữa, thông thường attribute này sẽ chỉ ra một phương pháp thay thế cho thành phần bị “đào thải” này.
- Thay vì xóa hẳn thành phần này đi ta vẫn giữ lại và “đánh dấu” nó để các dự án liên quan không xảy ra các lỗi về tương thích phiên bản



Obsolete attribute

```
1  class Program
2  {
3      [Obsolete()]
4      class MyClass
5      {
6      }
7
8
9      static void Main(string[] args)
10     {
11         new MyClass(); //warning
12         Print("Test"); //warning
13         Console.Read();
14     }
15
16     [Obsolete("Do not use this method")]
17     public static void Print(string message)
18     {
19         Console.WriteLine(message);
20     }
21 }
```



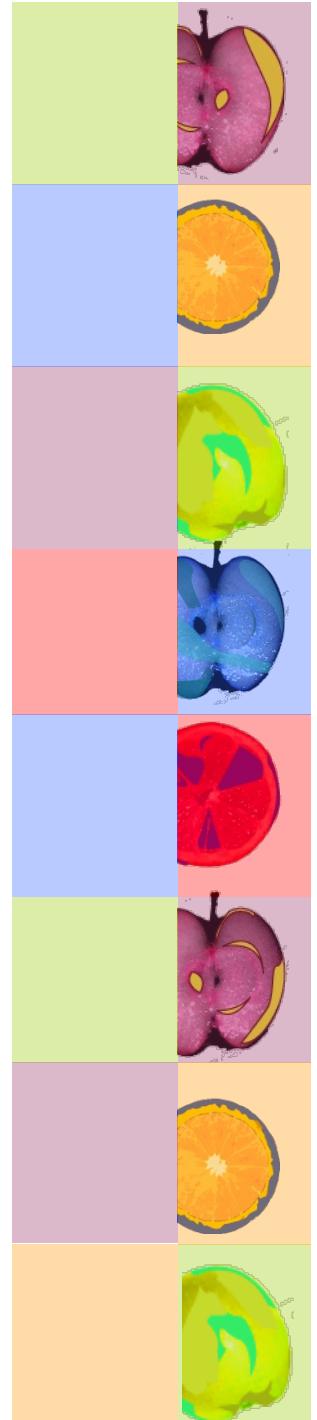
Obsolete attribute

- Khi biên dịch đoạn mã trên bạn sẽ nhận được 2 Warning trong cửa sổ Error List

```
“
1 | 'ConsoleApplication1.Program.MyClass' is obsolete
2 | 'ConsoleApplication1.Program.Print(string)' is obsolete: 'Do not use this method'
```

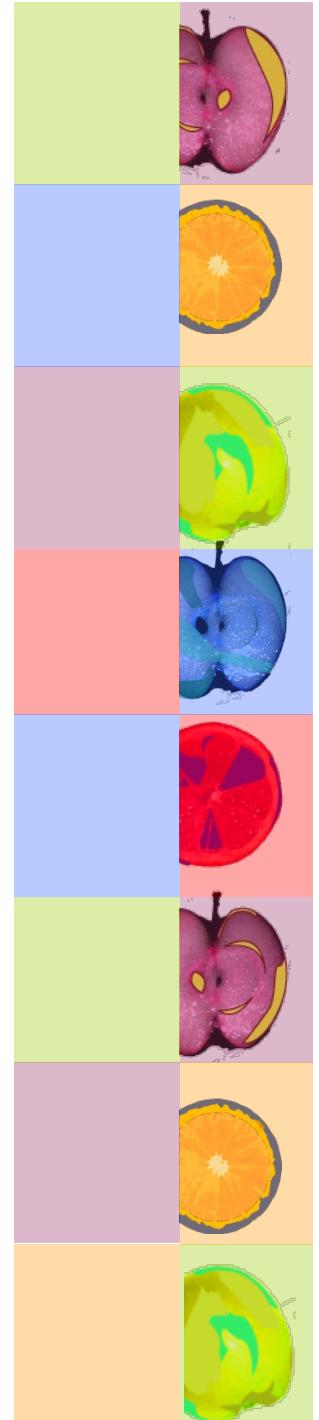
- Nếu bạn dùng gán tham số **isError** bằng true thì thay vì **warning**, trình biên dịch sẽ thay thế bằng một thông báo **error** tức là bạn không thể biên dịch được.

```
1 | [Obsolete("Do not use this method",true)]
2 | public static void Print(string message)
3 | {
4 |     Console.WriteLine(message);
5 | }
```



Conditional Attribute

- Attribute này được sử dụng để tạo ra các conditional method. Dựa vào điều kiện của các attribute mà conditional method có được biên dịch hay không.
- Để định nghĩa kí hiệu mà ta sử dụng trong Conditional attribute, ta sử dụng tiền chỉ thị `#define`, để hủy định nghĩa ta dùng tiền chỉ thị `#undef`.

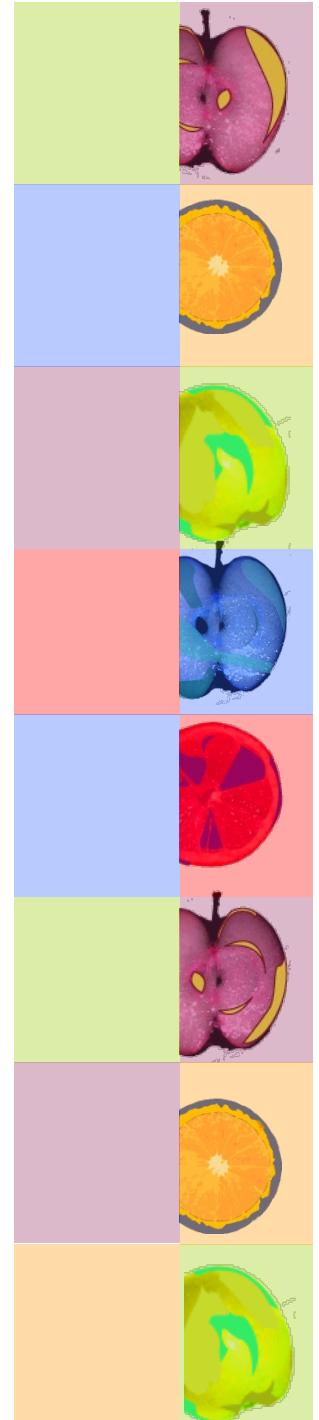


Conditional Attribute

- Các chỉ thị trên phải được đặt ở đầu mỗi tập tin mà conditional method được sử dụng. Nếu bạn sử dụng nhiều Conditional attribute thì phương thức sẽ được thực thi khi một trong các điều kiện được đáp ứng:
- Có thể được dùng để đánh dấu 1 phương thức như là 1 phương thức debug như sau:

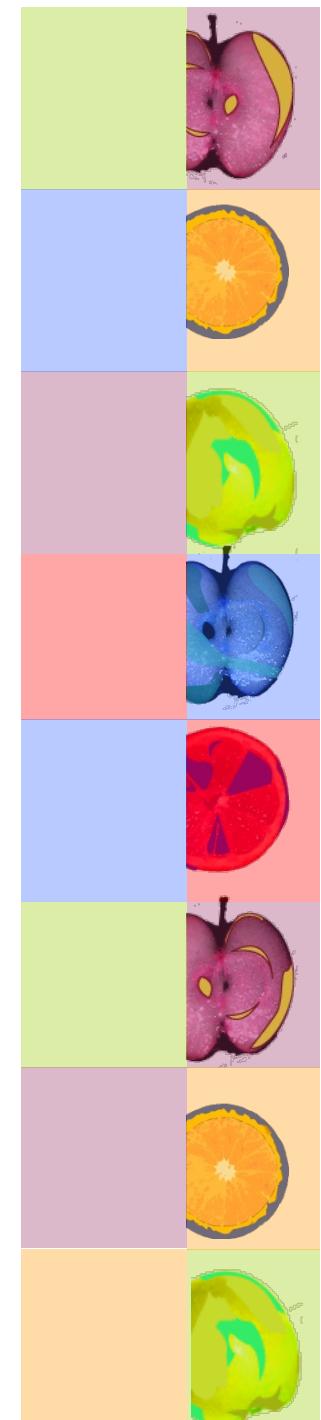
```
#if DEBUG  
    public void Foo()  
    { ... }  
#endif
```

```
[Conditional("DEBUG")]  
public void Foo()  
{ ... }
```



Conditional Attribute

```
1 #define CONDITION1
2
3 using System;
4 using System.Diagnostics;
5
6 namespace ConsoleApplication1
7 {
8     class Program
9     {
10
11         static void Main(string[] args)
12         {
13             PrintA("Test A");
14             new Program().PrintB("Test Program B");
15             new NewProgram().PrintB("Test New Program B");
16             Console.Read();
17         }
18
19         [Conditional("CONDITION1")]
20         public static void PrintA(string message)
21         {
22             Console.WriteLine(message);
23         }
24
25         [Conditional("CONDITION1"), Conditional("CONDITION2")]
26         public virtual void PrintB(string message)
27         {
28             Console.WriteLine(message);
29         }
30     }
31     class NewProgram : Program
32     {
33         public override void PrintB(string message)
34         {
35             base.PrintB(message);
36         }
37     }
38 }
```

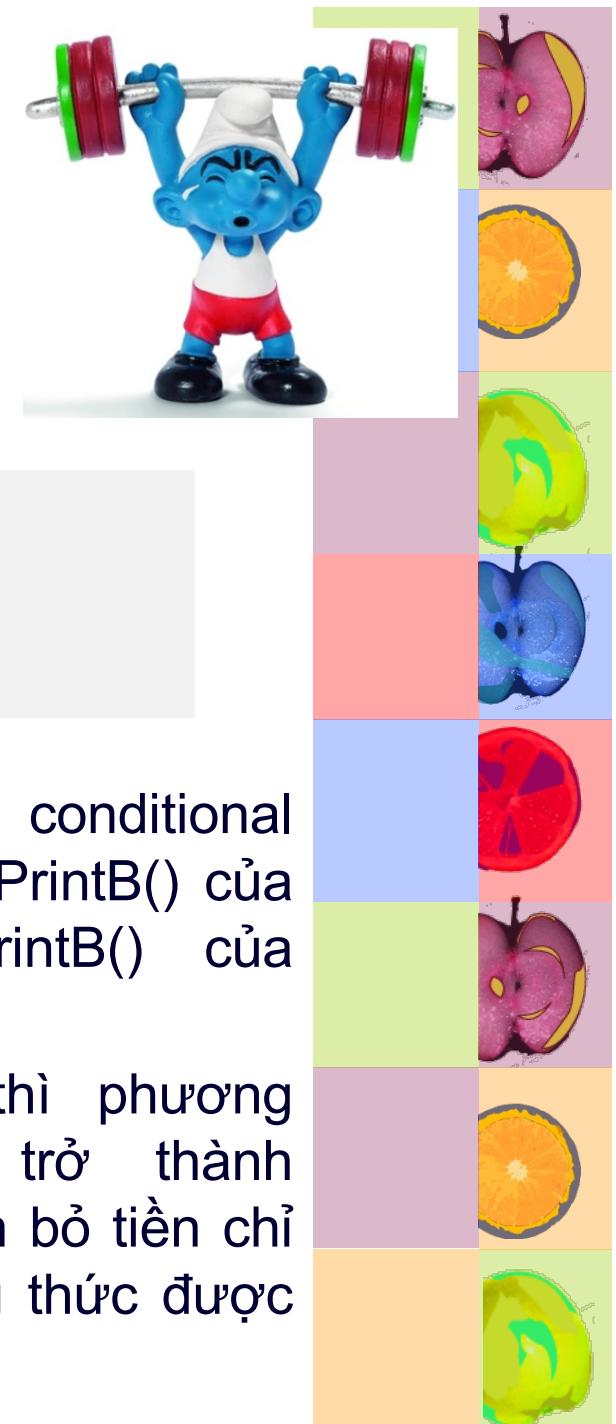


Conditional Attribute

- Kết quả xuất ra:

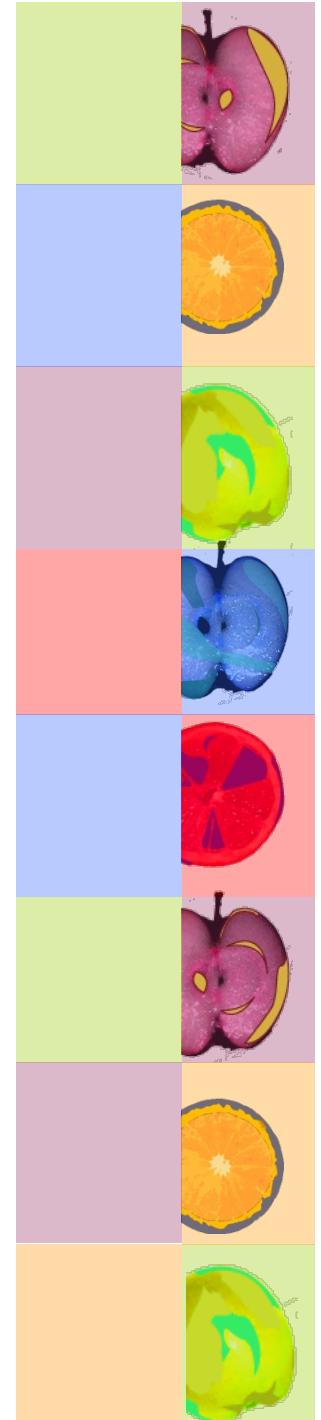
```
“  
Test A  
Test Program B  
Test NewProgram B
```

- Ví dụ trên cho ta một cái nhìn đầy đủ về conditional attribute. Bạn có thể thấy là phương thức PrintB() của lớp NewProgram được override từ PrintB() của lớp Program.
- Và theo một cách không tưởng minh thì phương thức PrintB() của NewProgram cũng trở thành một conditional method. Có nghĩa là nếu bạn bỏ tiền chỉ thị #define CONDITION1 đi thì cả 3 phương thức được gọi trong Main() đều không được thực thi.



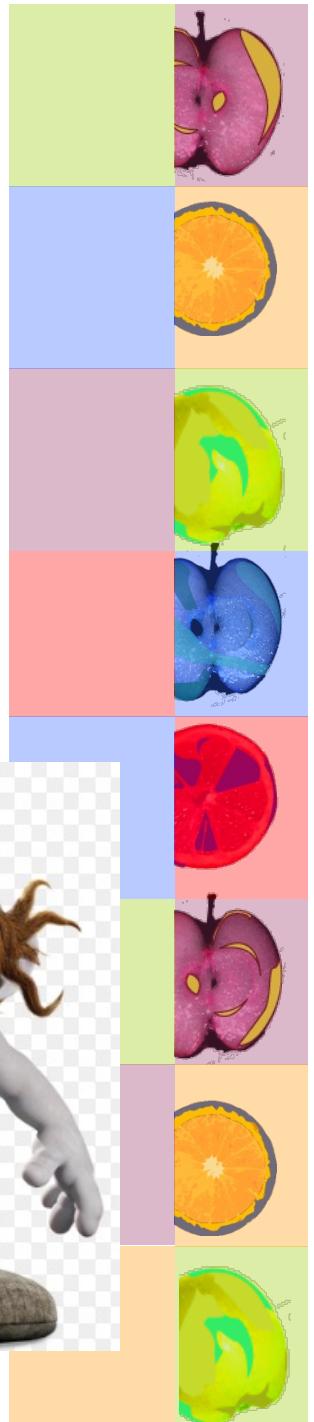
Dllimport attribute

- Ta xem xét các attribute này thông qua ví dụ Attributes. Trình bày 1 message box dùng hàm Window API, MessageBox, và cũng trình bày 2 thông điệp vá lỗi. 1 trong những thông điệp này được gọi trung gian qua 1 phương thức obsolete
- Bên trong phương thức ta định nghĩa phương thức mà ta gọi từ dll bên ngoài.hàm API MessageBox() được định nghĩa trong User32.dll, vì thế ta truyền tên tập tin vào attribute DllImport. Ta khai báo hàm giống như trong C.



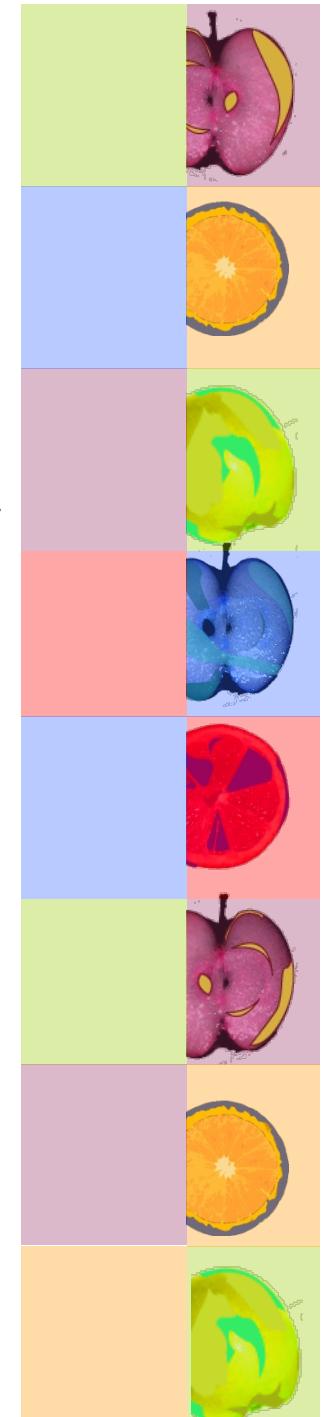
Dllimport attribute

```
1 #define DEBUG // comment out this line if doing a release build
2
3 using System;
4 using System.Runtime.InteropServices;
5 using System.Diagnostics;
6
7 namespace Wrox.ProCSharp.AdvancedCSharp
8 {
9
10    class MainEntryPoint
11    {
12        [DllImport("User32.dll")]
13        public static extern int MessageBox(int hParent, string Message,
14                                            string Caption, int Type);
15
16        static void Main()
17        {
18            DisplayRunningMessage();
19            DisplayDebugMessage();
20            MessageBox(0, "Hello", "Message", 0);
21        }
22
23        [Conditional("DEBUG")]
24        private static void DisplayRunningMessage()
25        {
26            Console.WriteLine("Starting Main routine. Current time is " +
27                               DateTime.Now);
28        }
29
30        [Conditional("DEBUG")]
31        [Obsolete()]
32        private static void DisplayDebugMessage()
33        {
34            Console.WriteLine("Starting Main routine");
35        }
36    }
37 }
```



Custom Attribute

- Chúng ta cùng xây dựng Custom Attribute có tên là *DeBugInfo*, mà lưu giữ thông tin thu được bởi việc debug bất kỳ chương trình nào. Nó có thể giữ thông tin sau:
 - Số hiệu code để bug
 - Tên lập trình viên, người nhận diện bug đó
 - Ngày review cuối cùng của code đó
 - Một thông báo dạng chuỗi để lưu giữ các lưu ý của lập trình viên

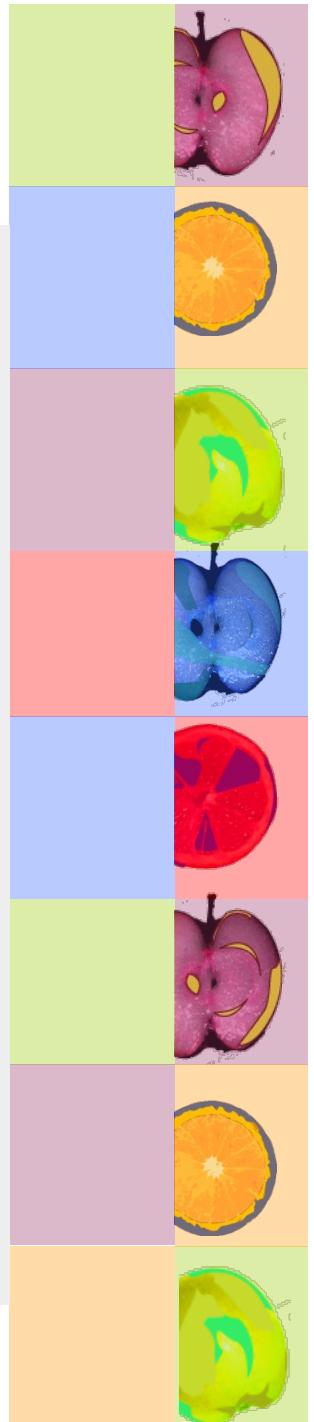


Custom Attribute

```
//Vi du minh họa một custom attribute BugFix  
[AttributeUsage(AttributeTargets.Class |  
AttributeTargets.Constructor |  
AttributeTargets.Field |  
AttributeTargets.Method |  
AttributeTargets.Property,  
AllowMultiple = true)]
```

```
public class DeBugInfo : System.Attribute  
{  
    private int bugNo;  
    private string developer;  
    private string lastReview;  
    public string message;  
  
    public DeBugInfo(int bg, string dev, string d)  
    {  
        this.bugNo = bg;  
        this.developer = dev;  
        this.lastReview = d;  
    }  
  
    public int BugNo  
    {  
        get  
        {  
            return bugNo;  
        }  
    }  
}
```

```
public string Developer  
{  
    get  
    {  
        return developer;  
    }  
}  
  
public string LastReview  
{  
    get  
    {  
        return lastReview;  
    }  
}  
  
public string Message  
{  
    get  
    {  
        return message;  
    }  
    set  
    {  
        message = value;  
    }  
}
```

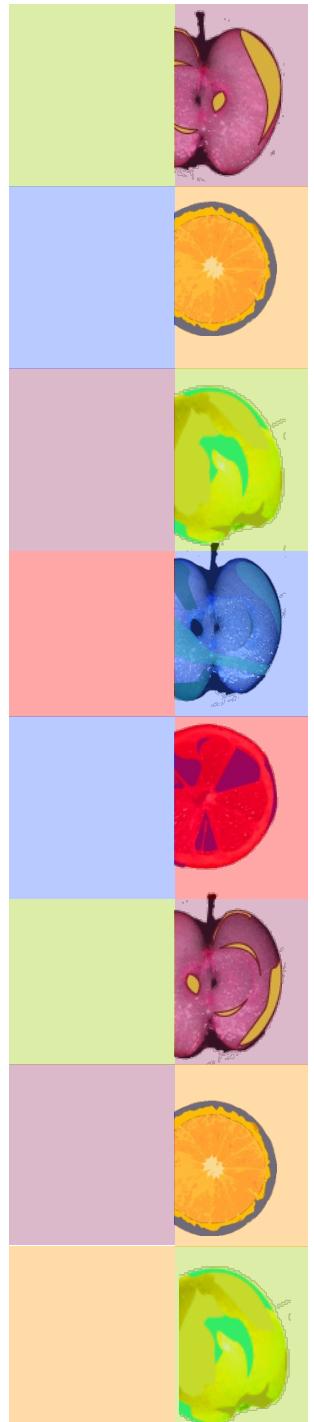


Custom Attribute

```
[DeBugInfo(45, "Tran Nam", "2/8/2016", Message = "Kieu tra ve khong hop le")]
[DeBugInfo(49, "Minh Chinh", "10/10/2016", Message = "Bien chua duoc su dung")]
class Rectangle
{
    //cac bien thanh vien
    protected double chieu_dai;
    protected double chieu_rong;
    public Rectangle(double l, double w)
    {
        chieu_dai = l;
        chieu_rong = w;
    }
    [DeBugInfo(55, "Tran Nam", "2/8/2016", Message = "Kieu tra ve khong hop le")]

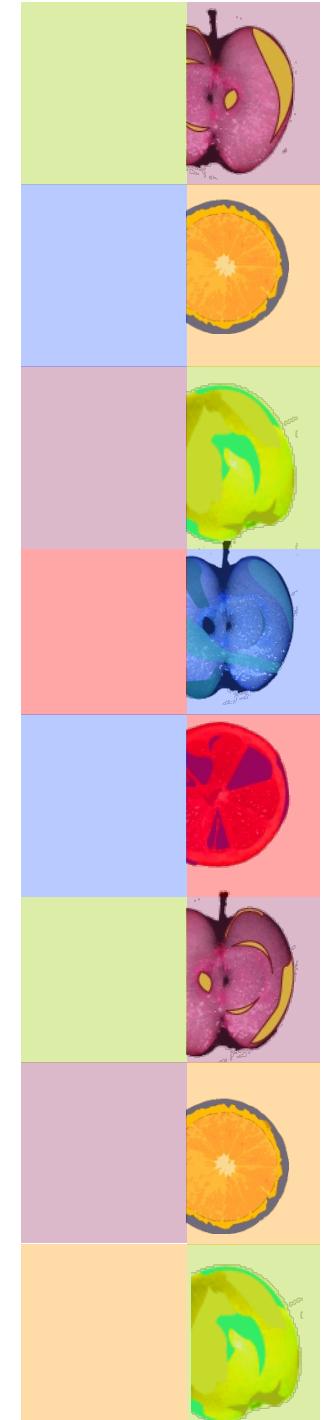
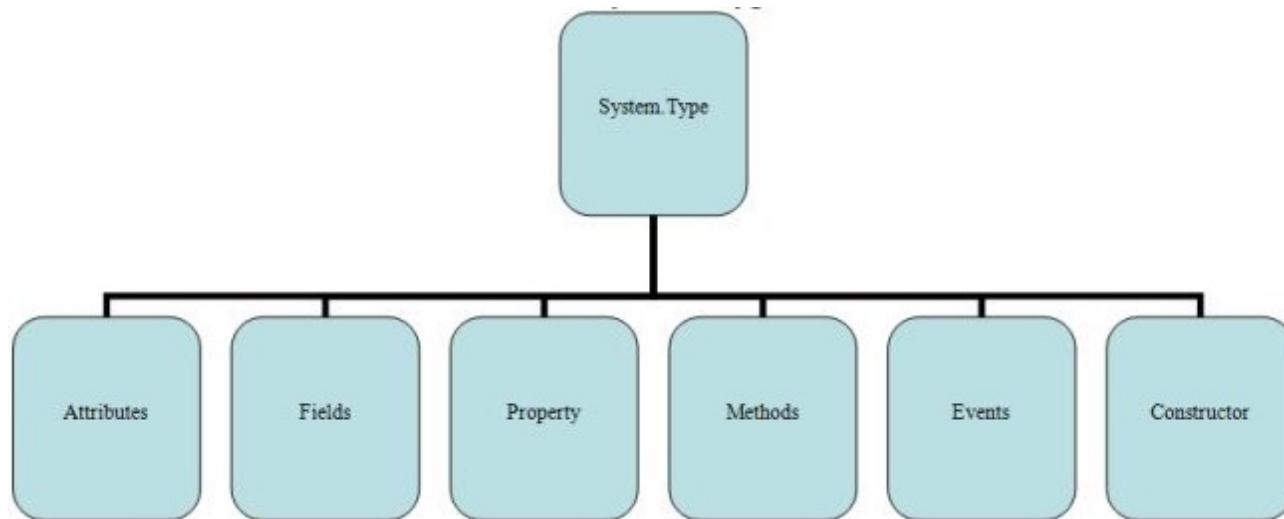
    public double tinhDienTich()
    {
        return chieu_dai * chieu_rong;
    }
    [DeBugInfo(56, "Minh Chinh", "10/10/2016")]

    public void Display()
    {
        Console.WriteLine("Chieu dai: {0}", chieu_dai);
        Console.WriteLine("Chieu rong: {0}", chieu_rong);
        Console.WriteLine("Dien tich: {0}", tinhDienTich());
    }
}
```



Reflection

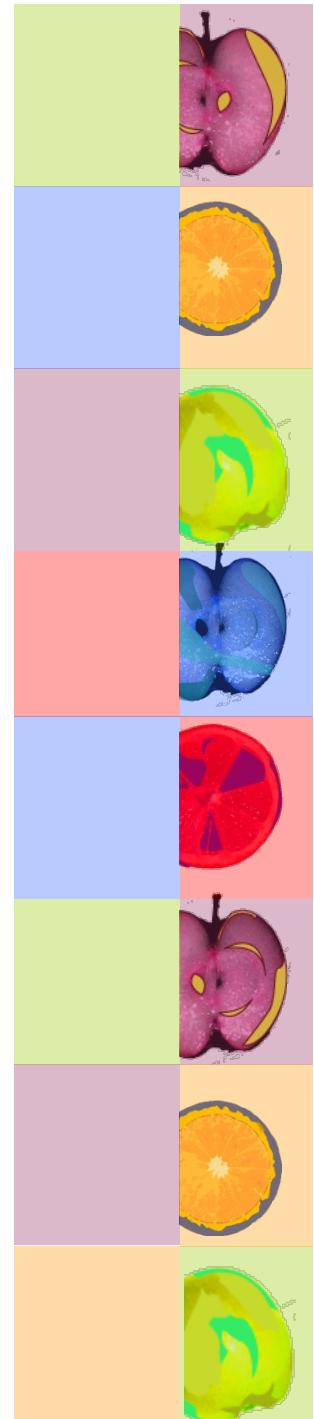
- Các đối tượng **Reflection** được sử dụng để thu được thông tin kiểu tại runtime.
- Các lớp này cung cấp truy cập tới metadata của một chương trình đang chạy là trong **System.Reflection** namespace trong C#



Reflection

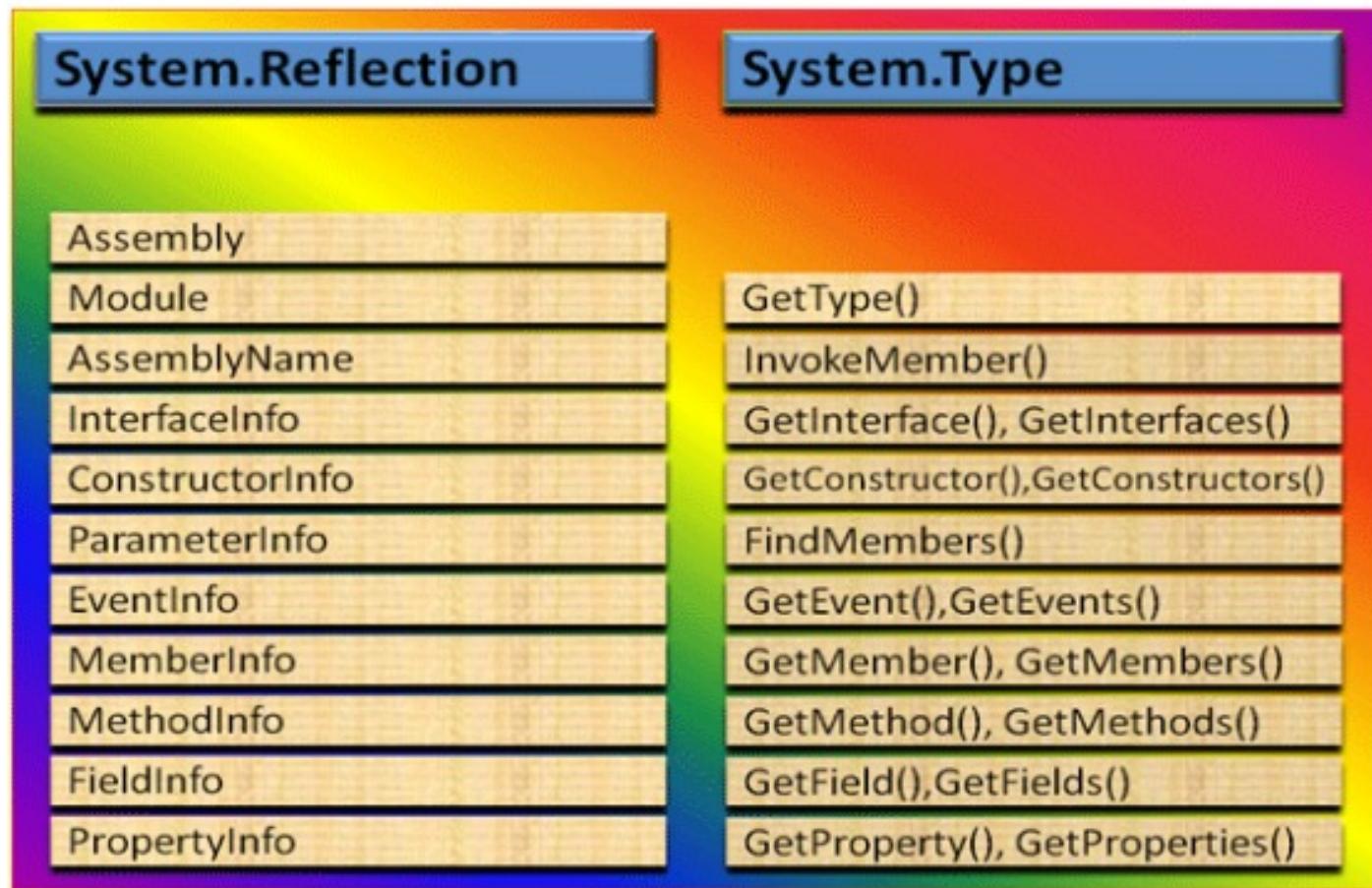
- Chúng ta đã đề cập trong chương trước rằng với việc sử dụng Reflection, bạn có thể quan sát thông tin attribute.
- Đối tượng **MethodInfo** của lớp **System.Reflection** trong C# cần được khởi tạo để phát hiện ra các attribute được liên kết với một lớp. Để làm điều này, bạn định nghĩa một đối tượng của lớp target, như:

```
System.Reflection.MemberInfo info = typeof(MyClass);
```



Reflection

.NET Reflection Road Map



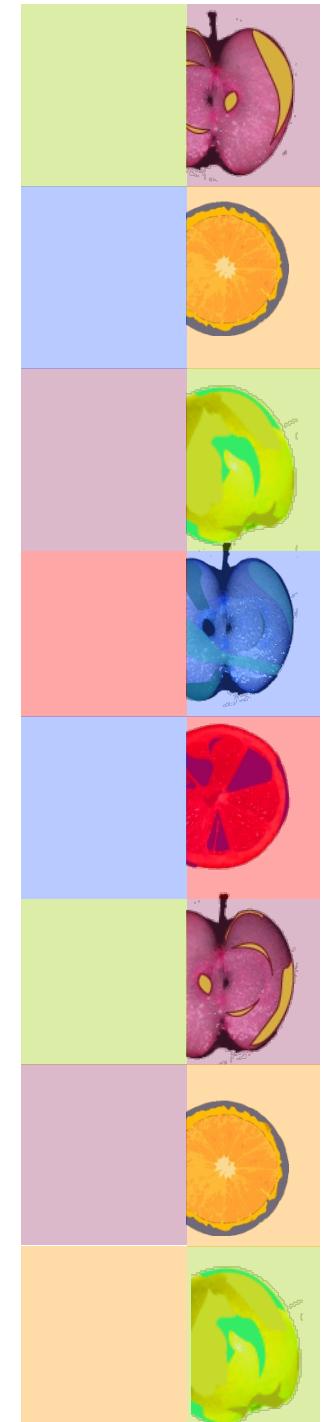
Reflection

```
using System;

namespace VietJackCsharp
{
    [AttributeUsage(AttributeTargets.All)]
    public class HelpAttribute : System.Attribute
    {
        public readonly string Url;

        public string Topic // Topic la mot name parameter
        {
            get
            {
                return topic;
            }
            set
            {
                topic = value;
            }
        }

        public HelpAttribute(string url) // url la mot positional parameter
        {
            this.Url = url;
        }
        private string topic;
    }
}
```



```
using System;

namespace VietJackCsharp
{
    [HelpAttribute("Thong tin tren lop MyClass")]
    class MyClass
    {
    }
}
```

```
using System;

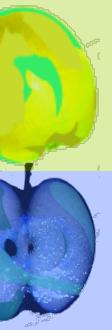
namespace VietJackCsharp
{
    class TestCsharp
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Reflection trong C#");
            Console.WriteLine("-----");

            System.Reflection.MemberInfo info = typeof(MyClass);
            object[] attributes = info.GetCustomAttributes(true);
            for (int i = 0; i < attributes.Length; i++)
            {
                System.Console.WriteLine(attributes[i]);
            }

            Console.ReadKey();
        }
    }
}
```

Reflection trong C#

VietJackCsharp.HelpAttribute



Reflection

- Trong ví dụ này, chúng ta sử dụng attribute là *DeBugInfo* được tạo trong chương trước và sử dụng Reflection để đọc metadata trong *Rectangle* class



```
using System;

namespace VietJackCsharp
{
    [DeBugInfo(45, "Tran Nam", "2/8/2016", Message = "Kieu tra ve khong hop le")]
    [DeBugInfo(49, "Minh Chinh", "10/10/2016", Message = "Bien khong duoc su dung")]

    class Rectangle
    {
        //cac bien thanh vien
        protected double chieu_dai;
        protected double chieu_rong;
        public Rectangle(double l, double w)
        {
            chieu_dai = l;
            chieu_rong = w;
        }
        [DeBugInfo(55, "Tran Nam", "2/8/2016", Message = "Kieu tra ve khong hop le")]
        public double tinhDienTich()
        {
            return chieu_dai * chieu_rong;
        }
        [DeBugInfo(56, "Minh Chinh", "19/10/2016")]
        public void Display()
        {
            Console.WriteLine("Chieu dai: {0}", chieu_dai);
            Console.WriteLine("Chieu rong: {0}", chieu_rong);
            Console.WriteLine("Dien tich: {0}", tinhDienTich());
        }
    }
}
```

Reflection

```
//Mot custom attribute BugFix
[AttributeUsage(AttributeTargets.Class |
AttributeTargets.Constructor |
AttributeTargets.Field |
AttributeTargets.Method |
AttributeTargets.Property,
AllowMultiple = true)]

public class DeBugInfo : System.Attribute
{
    private int bugNo;
    private string developer;
    private string lastReview;
    public string message;

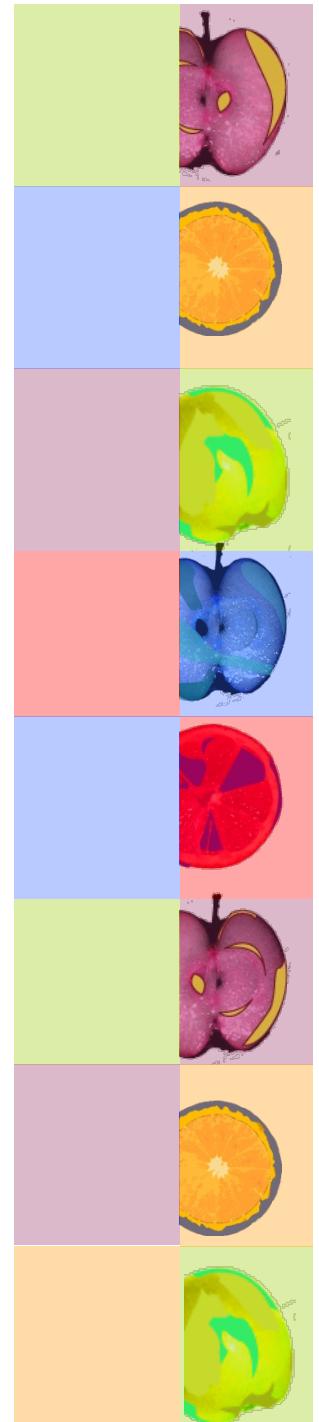
    public DeBugInfo(int bg, string dev, string d)
    {
        this.bugNo = bg;
        this.developer = dev;
        this.lastReview = d;
    }
}

public int BugNo
{
    get{ return bugNo; }
}

public string Developer
{
    get{ return developer; }
}

public string LastReview
{
    get{ return lastReview; }
}

public string Message
{
    get{ return message; }
    set{ message = value; }
}
```

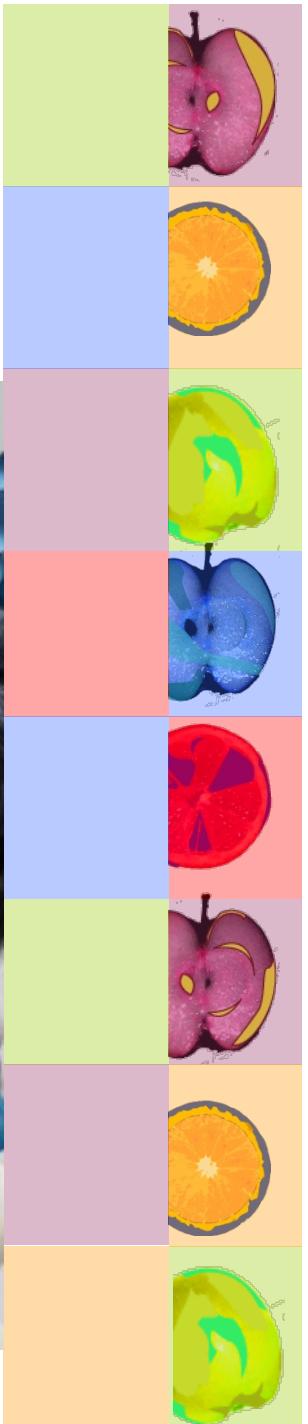


Reflection

```
static void Main(string[] args)
{
    Console.WriteLine("Reflection trong C#");
    Console.WriteLine("-----");

    Rectangle r = new Rectangle(4.5, 7.5);
    r.Display();
    Type type = typeof(Rectangle);

    //lap qua cac attribtue cua lop Rectangle
    foreach (Object attributes in type.GetCustomAttributes(false))
    {
        DeBugInfo dbi = (DeBugInfo)attributes;
        if (null != dbi)
        {
            Console.WriteLine("Bug no: {0}", dbi.BugNo);
            Console.WriteLine("Developer: {0}", dbi.Developer);
            Console.WriteLine("Last Reviewed: {0}", dbi.LastReview);
            Console.WriteLine("Remarks: {0}", dbi.Message);
        }
    }
}
```

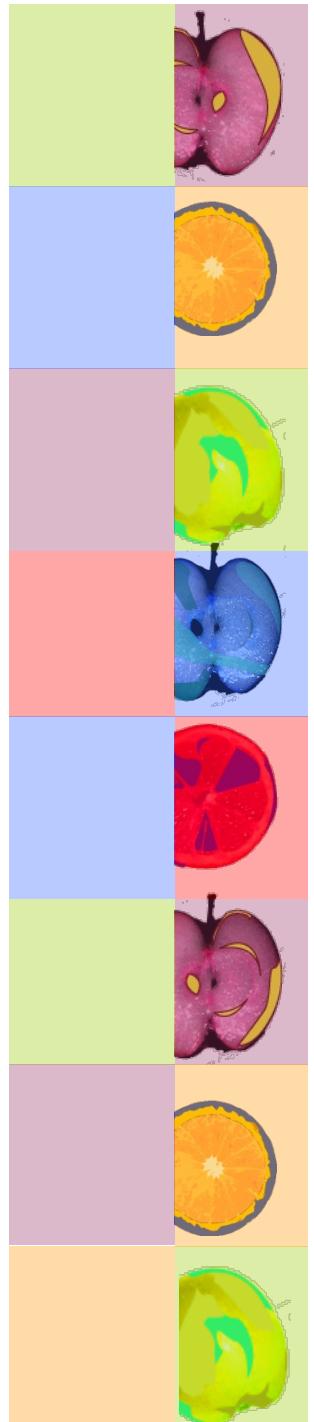


Reflection



```
//lap qua cac method attribue
foreach (MethodInfo m in type.GetMethods())
{
    foreach (Attribute a in m.GetCustomAttributes(true))
    {
        DeBugInfo dbi = (DeBugInfo)a;
        if (null != dbi)
        {
            Console.WriteLine("Bug no: {0}, for Method: {1}", dbi.BugNo, m.Name);
            Console.WriteLine("Developer: {0}", dbi.Developer);
            Console.WriteLine("Last Reviewed: {0}", dbi.LastReview);
            Console.WriteLine("Remarks: {0}", dbi.Message);
        }
    }
}

Console.ReadLine();
Console.ReadKey();
}
```



Reflection

Biên dịch và chạy chương trình C# trên sē cho kết quả sau:

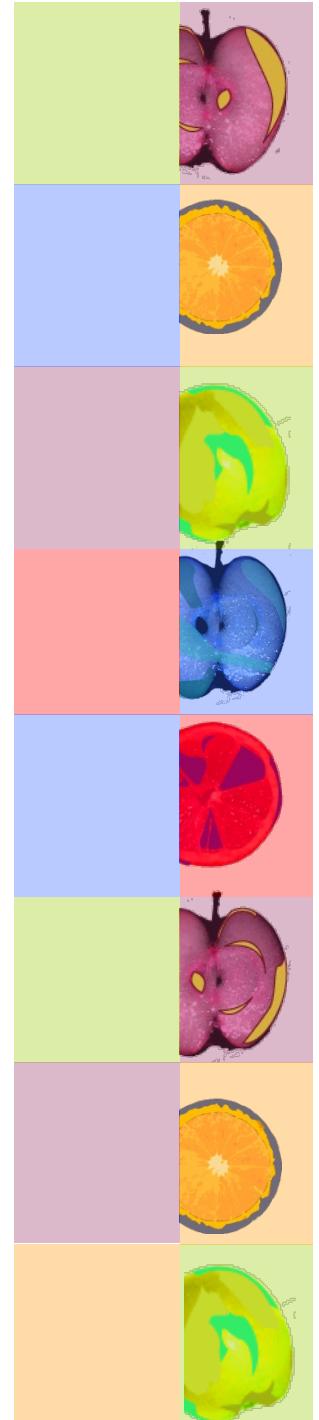
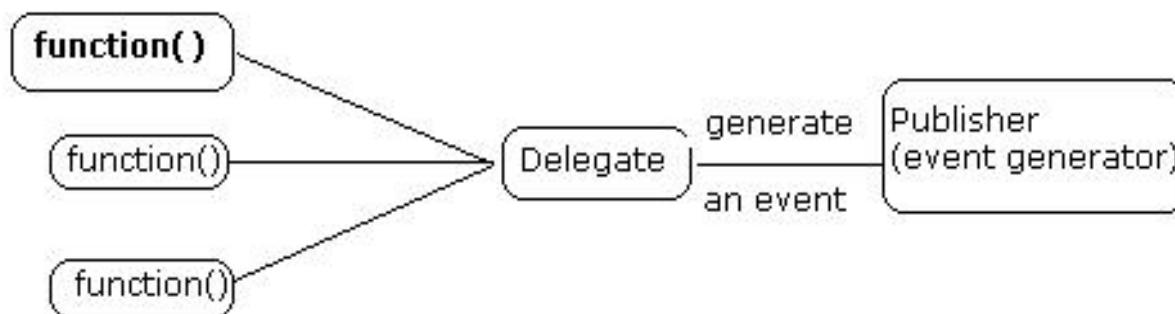


```
Reflection trong C#
-----
Chieu dai: 4.5
Chieu rong: 7.5
Dien tich: 33.75
Bug no: 49
Developer: Minh Chinh
Last Reviewed: 10/10/2016
Remarks: Bien khong duoc su dung
Bug no: 45
Developer: Tran Nam
Last Reviewed: 2/8/2016
Remarks: Kieu tra ve khong hop le
Bug no: 55, for Method: tinhDienTich
Developer: Tran Nam
Last Reviewed: 2/8/2016
Remarks: Kieu tra ve khong hop le
Bug no: 56, for Method: Display
Developer: Minh Chinh
Last Reviewed: 19/10/2016
Remarks:
```



Delegates

- Delegate trong C# là tương tự như con trỏ tới các hàm
- Một **Delegate** là một biến kiểu tham chiếu mà giữ tham chiếu tới một phương thức. Tham chiếu đó có thể được thay đổi tại runtime.
- Đặc biệt, các delegate được sử dụng để triển khai các sự kiện và các phương thức call-back. Tất cả delegate được kế thừa một cách ngầm định từ lớp **System.Delegate** trong C#.



Delegates

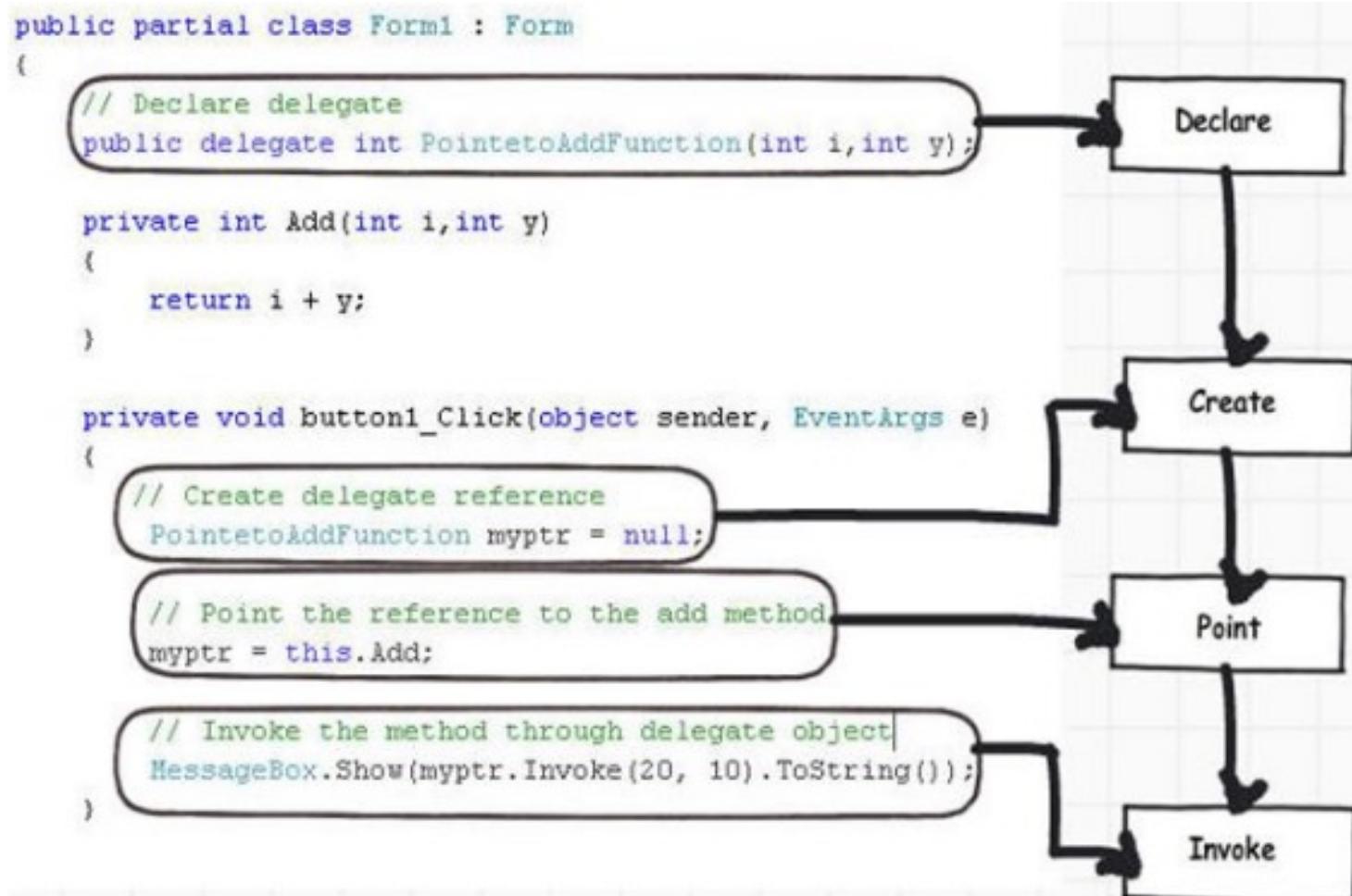
```
public partial class Form1 : Form
{
    // Declare delegate
    public delegate int PointetoAddFunction(int i,int y);

    private int Add(int i,int y)
    {
        return i + y;
    }

    private void button1_Click(object sender, EventArgs e)
    {
        // Create delegate reference
        PointetoAddFunction myptr = null;

        // Point the reference to the add method.
        myptr = this.Add;

        // Invoke the method through delegate object
        MessageBox.Show(myptr.Invoke(20, 10).ToString());
    }
}
```



Delegates

- Khai báo Delegate trong C# quyết định các phương thức mà có thể được tham chiếu bởi Delegate đó.
- Một Delegate có thể tham chiếu tới một phương thức, mà có cùng dấu hiệu như của Delegate đó

Ví dụ, bạn xét một delegate sau đây:

```
public delegate int MyDelegate (string s);
```

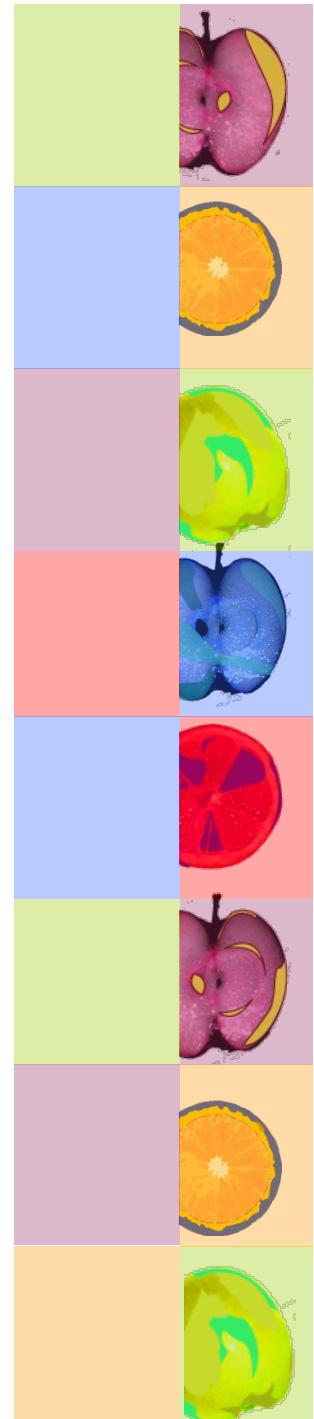
Delegate trên có thể được sử dụng để tham chiếu bất kỳ phương thức mà có một tham số *string* đơn và trả về một biến kiểu *int*.



Delegates

- Khi một kiểu delegate được khai báo, một đối tượng delegate phải được tạo với từ khóa **new** và được liên kết với một phương thức cụ thể

```
public delegate void printString(string s);
...
printString ps1 = new printString(WriteToScreen);
printString ps2 = new printString(WriteToFile);
```

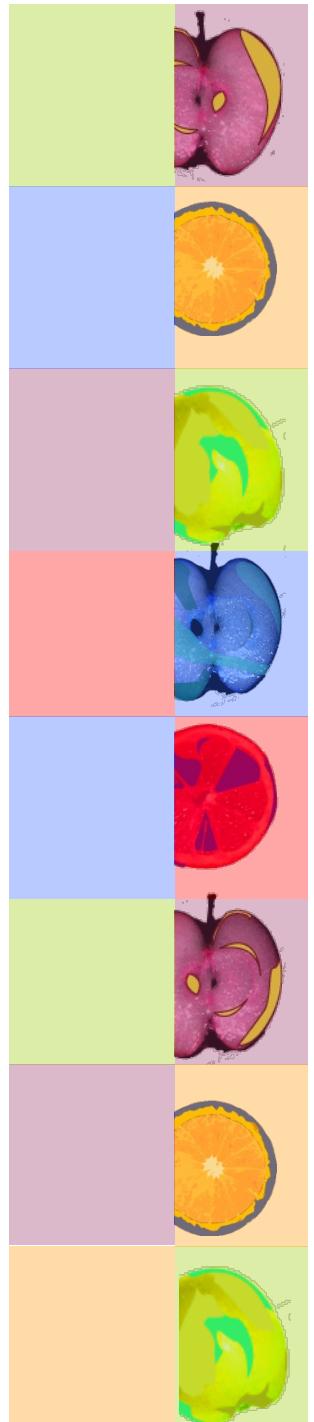


Delegates

```
using System;

delegate int NumberChanger(int n);
namespace VietJackCsharp
{
    class TestCsharp
    {
        static int num = 10;
        public static int AddNum(int p)
        {
            num += p;
            return num;
        }

        public static int MultNum(int q)
        {
            num *= q;
            return num;
        }
        public static int getNum()
        {
            return num;
        }
    }
}
```



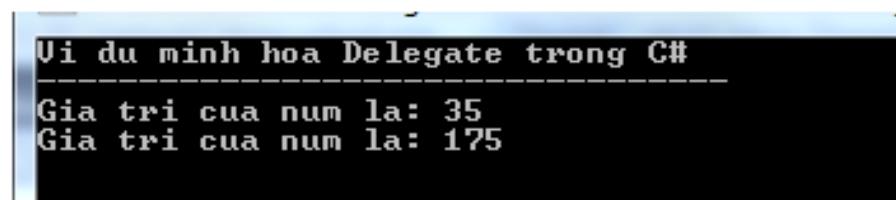
Delegates

```
static void Main(string[] args)
{
    Console.WriteLine("Vi du minh hoa Delegate trong C#");
    Console.WriteLine("-----");

    //tao cac doi tuong delegate
    NumberChanger nc1 = new NumberChanger(AddNum);
    NumberChanger nc2 = new NumberChanger(MultNum);

    //goi cac phuong thuc boi su dung cac doi tuong delegate
    nc1(25);
    Console.WriteLine("Gia tri cua num la: {0}", getNum());
    nc2(5);
    Console.WriteLine("Gia tri cua num la: {0}", getNum());
    Console.ReadKey();
}
```

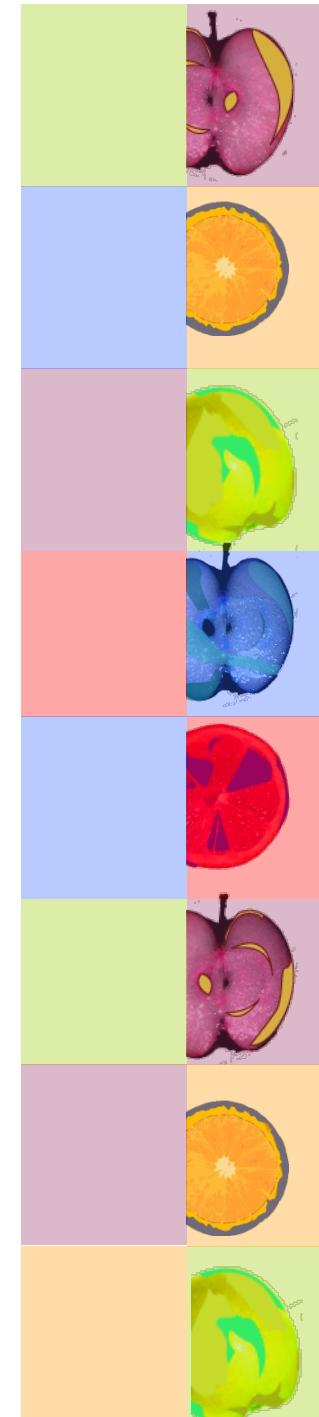
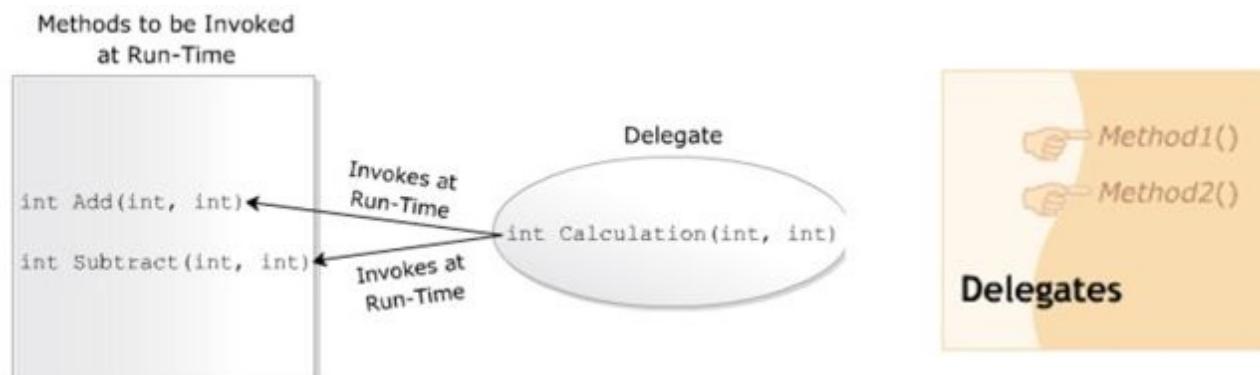
Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:



```
Vi du minh hoa Delegate trong C#
-----
Gia tri cua num la: 35
Gia tri cua num la: 175
```

Multicast một Delegate trong C#

- Các đối tượng Delegate có thể được hợp thành bởi sử dụng toán tử "+".
- Chỉ có các delegate cùng kiểu có thể được hợp thành
- Sử dụng thuộc tính này của các delegate, bạn có thể tạo một danh sách triệu hồi của các phương thức mà sẽ được gọi khi delegate đó được triệu hồi.
- Điều này được gọi là Multicasting của một Delegate.



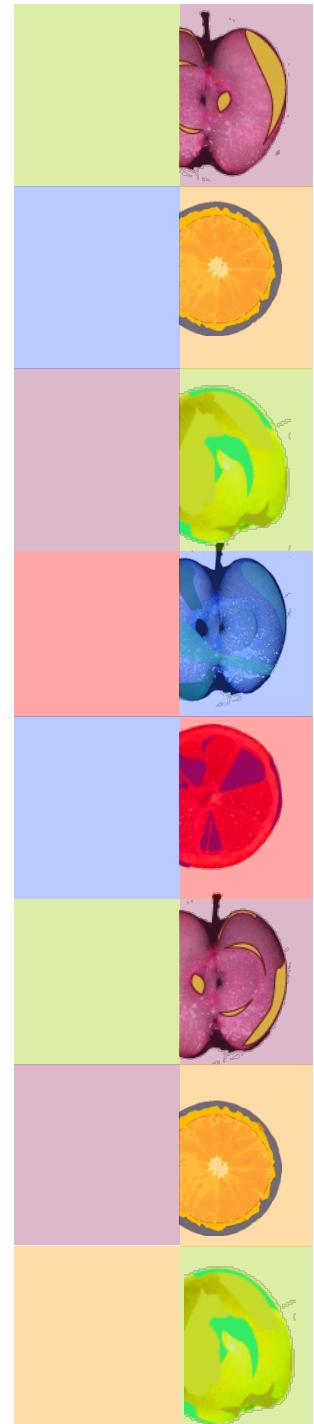
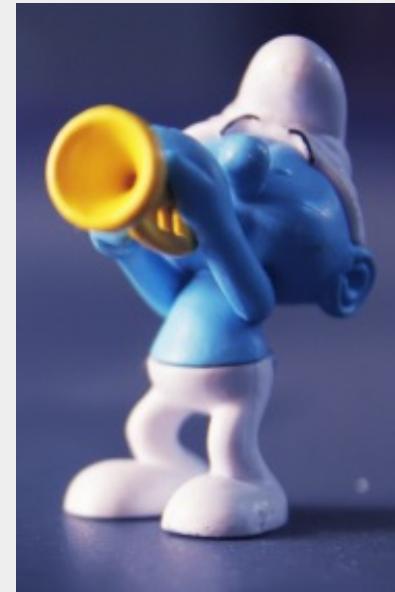
Multicast một Delegate trong C#

```
using System;

delegate int NumberChanger(int n);
namespace VietJackCsharp
{
    class TestCsharp
    {
        static int num = 10;
        public static int AddNum(int p)
        {
            num += p;
            return num;
        }

        public static int MultNum(int q)
        {
            num *= q;
            return num;
        }

        public static int getNum()
        {
            return num;
        }
    }
}
```



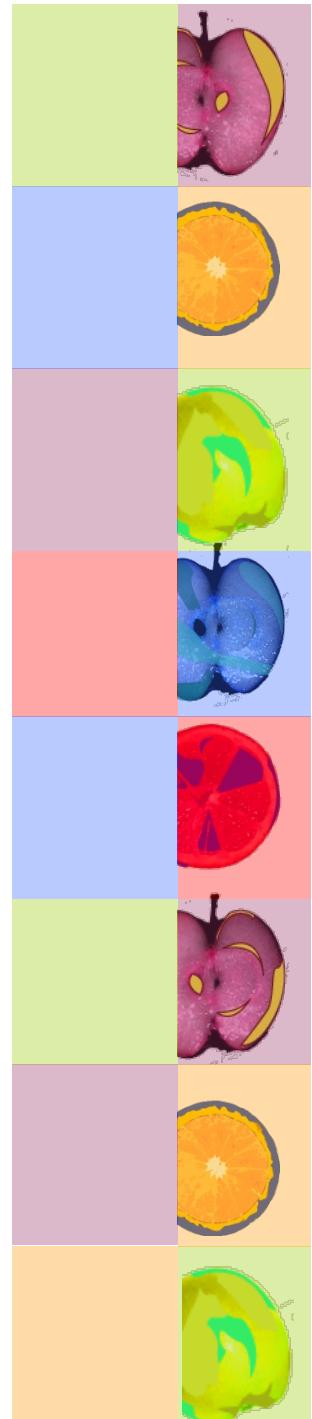
Multicast một Delegate trong C#

```
static void Main(string[] args)
{
    Console.WriteLine("Vi du minh hoa Delegate trong C#");
    Console.WriteLine("-----");

    //tao cac doi tuong delegate
    NumberChanger nc;
    NumberChanger nc1 = new NumberChanger(AddNum);
    NumberChanger nc2 = new NumberChanger(MultNum);
    nc = nc1;
    nc += nc2;

    //goi multicast
    nc(5);
    Console.WriteLine("Gia tri cua num la: {0}", getNum());
    Console.ReadKey();
}
```

```
Vi du minh hoa Delegate trong C#
-----
Gia tri cua num la: 75
```



Delegate

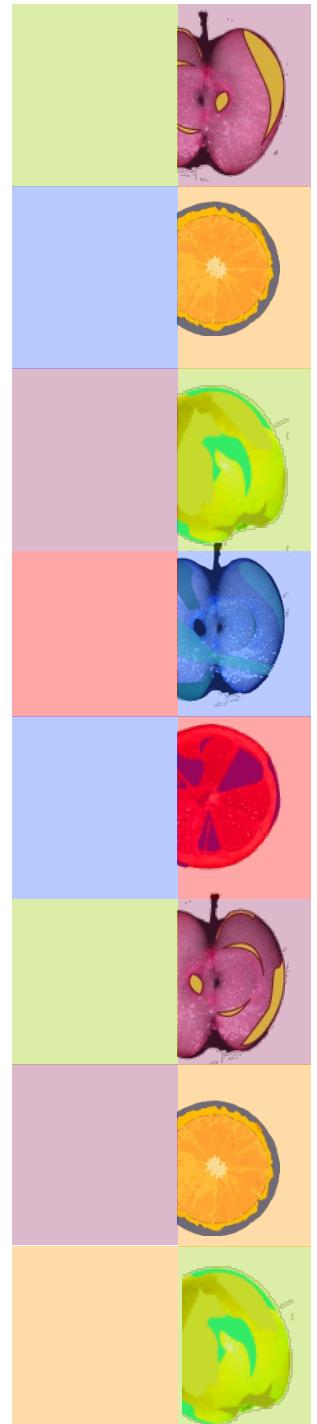
```
class TestCsharp
{
    static FileStream fs;
    static StreamWriter sw;

    // khai bao delegate
    public delegate void printString(string s);

    // phuong thuc de in tren console
    public static void WriteToScreen(string str)
    {
        Console.WriteLine("Chuoi la: {0}", str);
    }

    //phuong thuc nay de ghi du lieu vao file
    public static void WriteToFile(string s)
    {
        fs = new FileStream("c:\\\\message.txt",
            FileMode.Append, FileAccess.Write);
        sw = new StreamWriter(fs);
        sw.WriteLine(s);
        sw.Flush();
        sw.Close();
        fs.Close();
    }

    // phuong thuc nay nhan delegate lam tham so va su dung no de
    // goi cac phuong thuc neu can
    public static void sendString(printString ps)
    {
        ps("Hoc C# co ban va nang cao tai VietJack");
    }
}
```

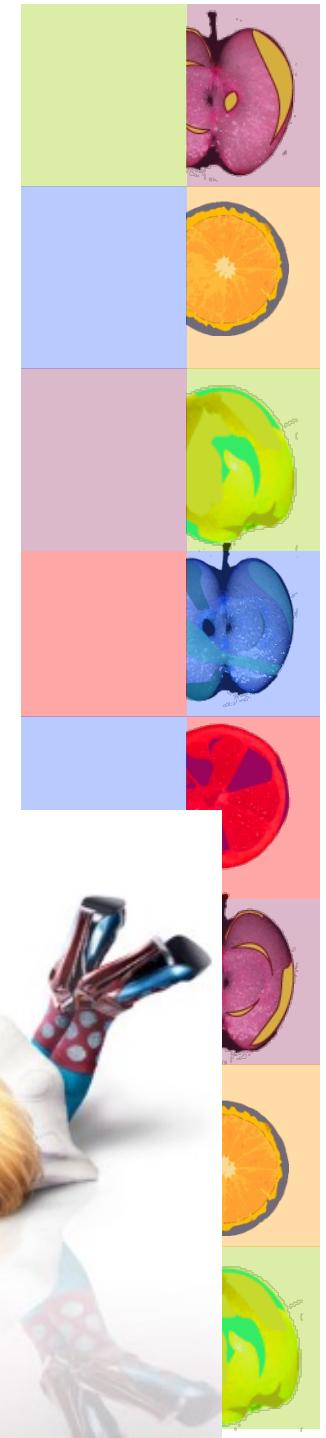


Delegate

```
static void Main(string[] args)
{
    Console.WriteLine("Vi du minh hoa Delegate trong C#");
    Console.WriteLine("-----");

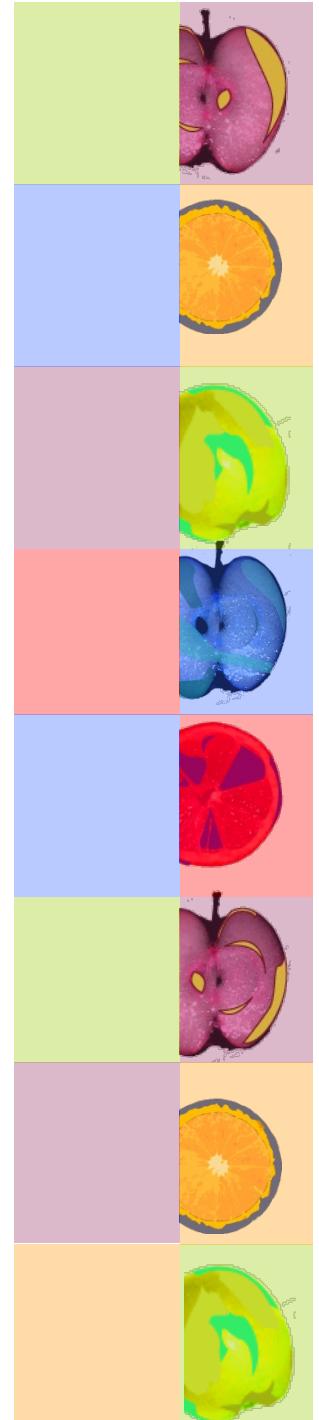
    printString ps1 = new printString(WriteToScreen);
    printString ps2 = new printString(WriteToFile);
    sendString(ps1);
    sendString(ps2);
    Console.ReadKey();
}
```

```
Vi du minh hoa Delegate trong C#
-----
Chuoi la: Hoc C# co ban va nang cao tai VietJack
```



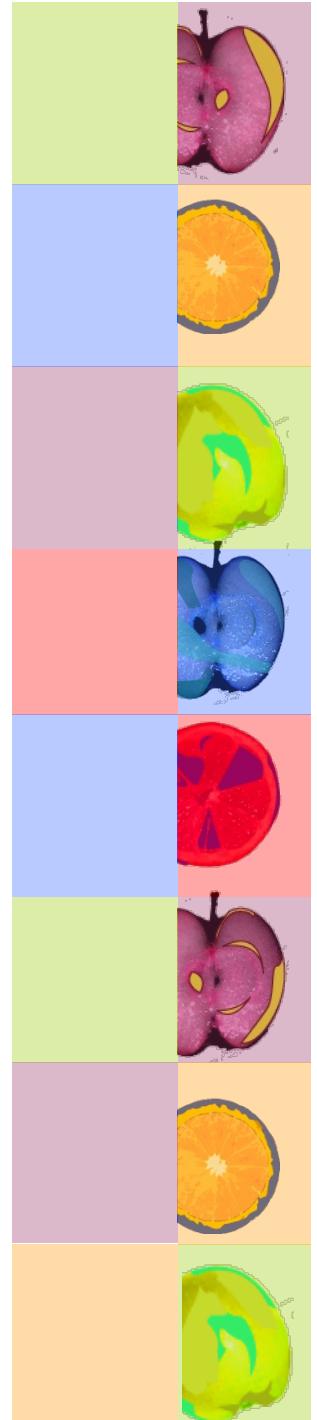
Event

- **Sự kiện (Event)** là các hành động của người dùng, ví dụ như nhấn phím, click, di chuyển chuột, ... Các Application cần phản hồi các sự kiện này khi chúng xuất hiện. Ví dụ, các ngắt (interrupt). Các sự kiện (Event) được sử dụng để giao tiếp bên trong tiến trình.
- Các Event được khai báo và được tạo trong một lớp và được liên kết với Event Handler bởi sử dụng các Delegate bên trong cùng lớp đó hoặc một số lớp khác.



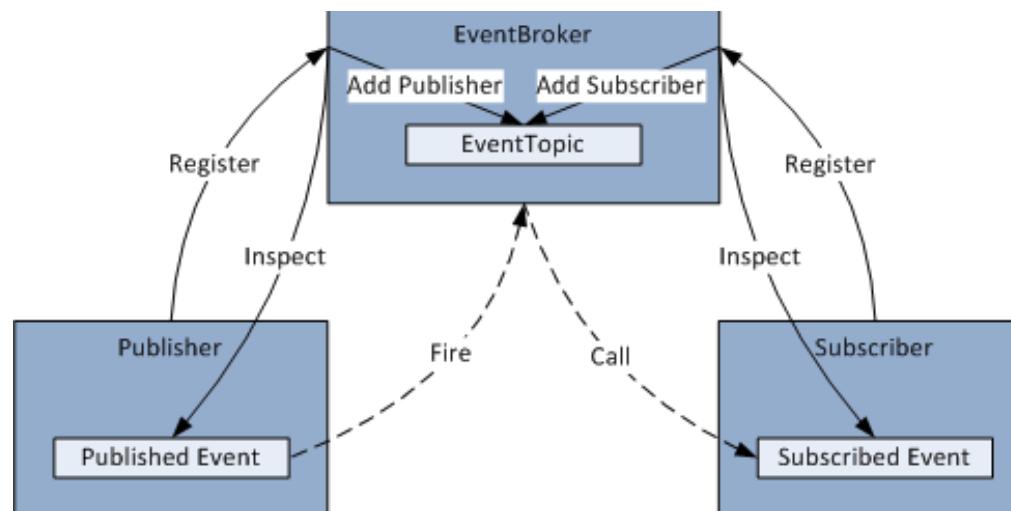
Event

- Lớp mà chứa Event được sử dụng để công bố event đó. Điều này được gọi là lớp **Publisher**. Một số lớp khác mà chấp nhận Event này được gọi là lớp **Subscriber**.
- Các Event trong C# sử dụng mô hình **Publisher-Subscriber**.



Event

- Một **Publisher** trong C# là một đối tượng mà chưa định nghĩa của event và delegate đó. Mỗi liên hệ event-delegate cũng được định nghĩa trong đối tượng này. Một đối tượng lớp Publisher triệt hồi Event và nó được thông báo tới các đối tượng khác.
- Một **Subscriber** trong C# là một đối tượng mà chấp nhận event và cung cấp một Event Handler. Delegate trong lớp Publisher triệt phương thức (Event Handler) của lớp Subscriber



Event

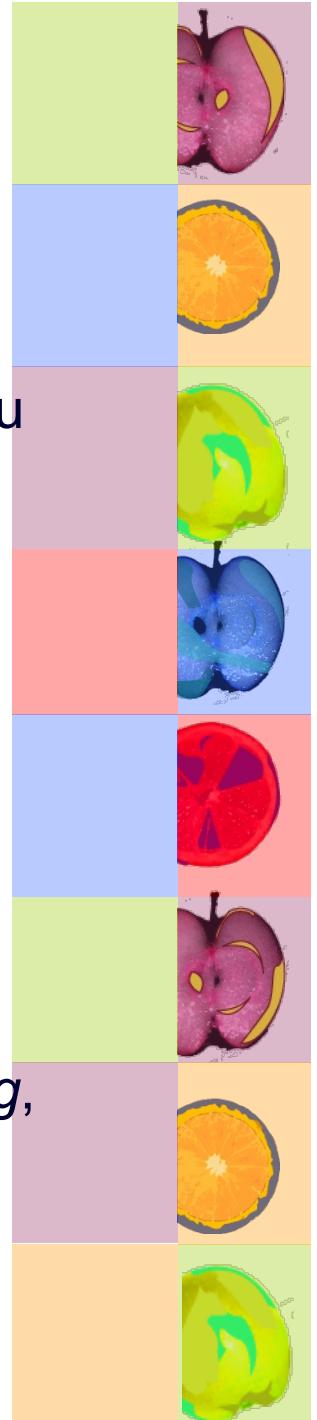
- Để khai báo một Event bên trong một lớp, đầu tiên một kiểu delegate cho Event đó phải được khai báo. Ví dụ:

```
public delegate void BoilerLogHandler(string status);
```

- Tiếp theo, chính Event đó được khai báo, bởi sử dụng từ khóa **event** trong C#:

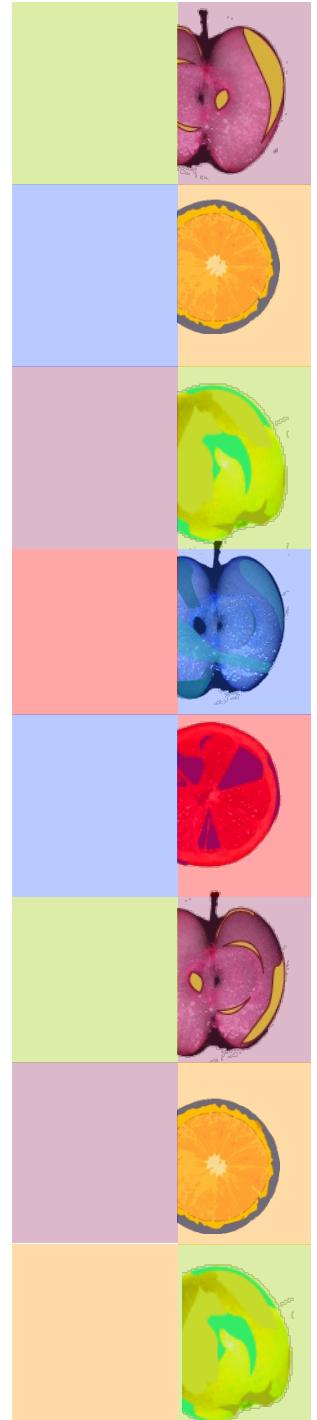
```
//định nghĩa event dựa vào delegate ở trên  
public event BoilerLogHandler BoilerEventLog;
```

- Code trên định nghĩa một delegate với tên là *BoilerLogHandler* và một Event với tên là *BoilerEventLog*, mà triệu hồi delegate đó khi nó được tạo ra.



Event

- Ví dụ này cung cấp một ứng dụng đơn giản để xử lý sự cố cho một hệ thống nồi hơi nước nóng. Khi kỹ sư bảo dưỡng kiểm tra nồi hơi, nhiệt độ và áp suất nồi hơi được tự động ghi lại vào trong một log file cùng với các ghi chú của kỹ sư bảo dưỡng này.



Event

Lớp **DelegateBoilerEvent**: đóng vai trò như là event publisher

```
class Boiler
{
    private int temp;
    private int pressure;
    public Boiler(int t, int p)
    {
        temp = t;
        pressure = p;
    }

    public int getTemp()
    {
        return temp;
    }

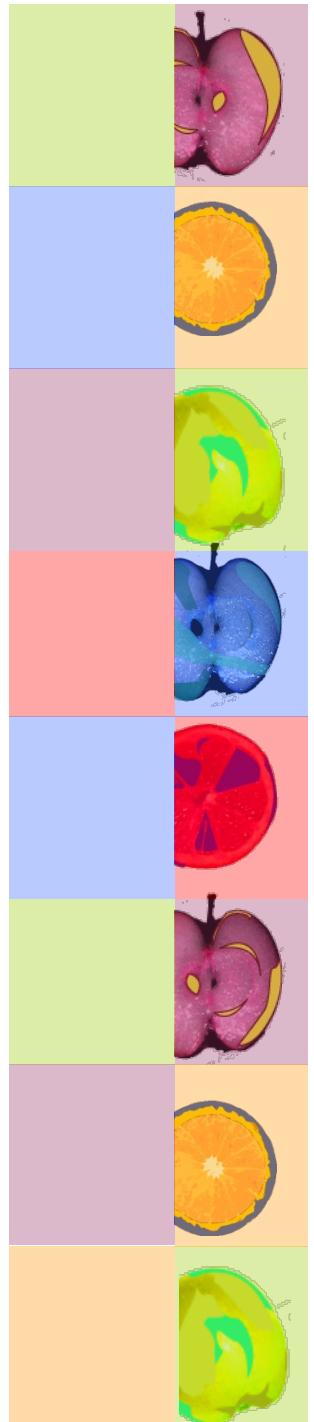
    public int getPressure()
    {
        return pressure;
    }
}
```

```
class DelegateBoilerEvent
{
    public delegate void BoilerLogHandler(string status);

    //dinh nghia su kien dua vao delegate tren
    public event BoilerLogHandler BoilerEventLog;

    public void LogProcess()
    {
        string remarks = "OK!";
        Boiler b = new Boiler(100, 12);
        int t = b.getTemp();
        int p = b.getPressure();
        if(t > 150 || t < 80 || p < 12 || p > 15)
        {
            remarks = "Can duy tri";
        }
        OnBoilerEventLog("Thong tin log:\n");
        OnBoilerEventLog("Nhiệt độ: " + t + "\nÁp suất: " + p);
        OnBoilerEventLog("\nThông báo: " + remarks);
    }

    protected void OnBoilerEventLog(string message)
    {
        if (BoilerEventLog != null)
        {
            BoilerEventLog(message);
        }
    }
}
```



Event

Lớp **TestCsharp**: event subscriber

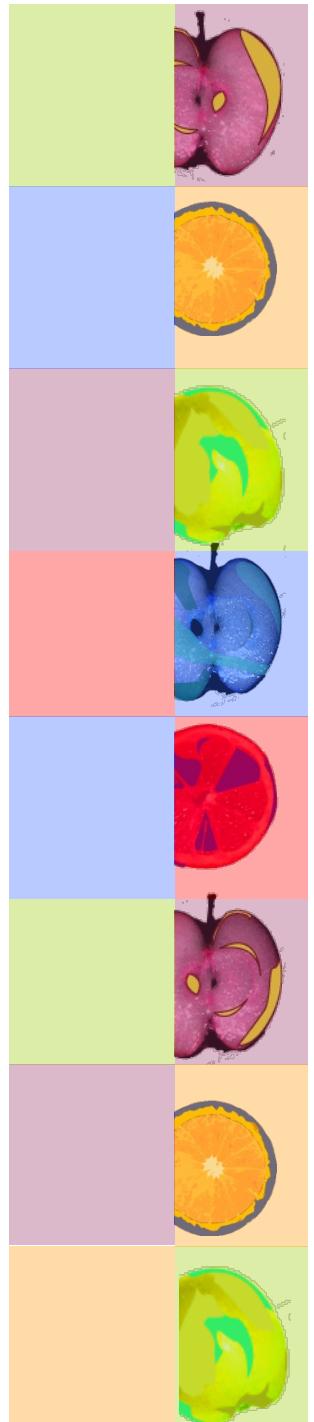
```
class TestCsharp
{
    static void Logger(string info)
    {
        Console.WriteLine(info);
    }

    static void Main(string[] args)
    {
        Console.WriteLine("Vi du minh hoa su kien trong C#");
        Console.WriteLine("-----");

        BoilerInfoLogger filelog = new BoilerInfoLogger("e:\\boiler.txt");
        DelegateBoilerEvent boilerEvent = new DelegateBoilerEvent();
        boilerEvent.BoilerEventLog += new
        DelegateBoilerEvent.BoilerLogHandler(Logger);
        boilerEvent.BoilerEventLog += new
        DelegateBoilerEvent.BoilerLogHandler(filelog.Logger);
        boilerEvent.LogProcess();
        Console.ReadLine();
        Console.ReadKey();
        filelog.Close();
    }
}
```

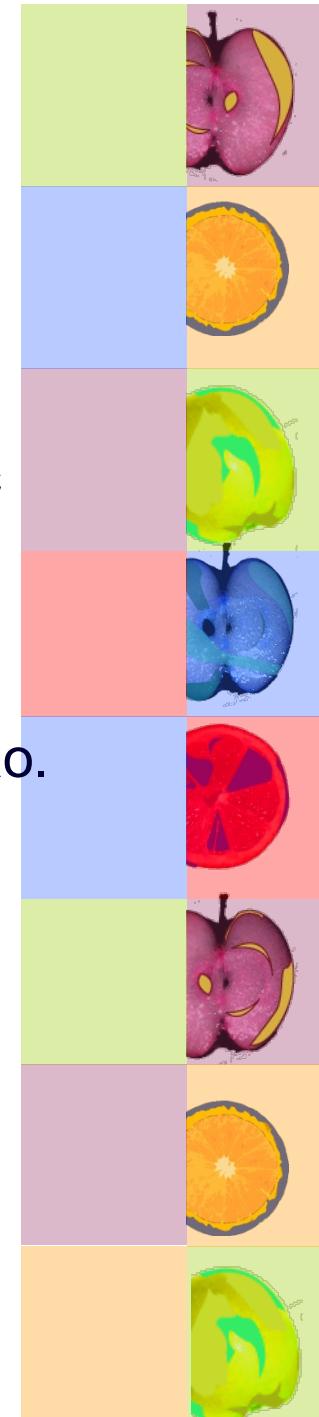
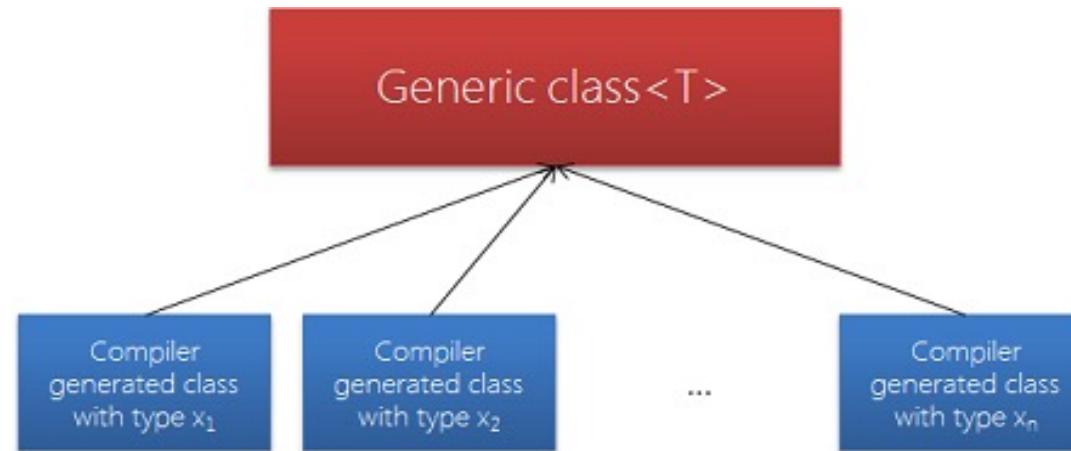
Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:

```
Vi du minh hoa su kien trong C#
Thong tin log:
Nhiет độ: 100
Áp suất: 12
Thông báo: OK!
```



Generic

- **Generic** trong C# cho phép bạn trì hoãn các đặc điểm kỹ thuật (Specification) của kiểu dữ liệu của việc lập trình các phần tử trong một lớp hoặc một phương thức, tới khi nó thực sự được sử dụng trong chương trình.
- Nói cách khác, Generic cho phép bạn viết một lớp hoặc phương thức mà có thể làm việc với bất kỳ kiểu dữ liệu nào.



Generic

```
class TestGeneric <T>
{
    private T[] array;
    public TestGeneric(int size)
    {
        array = new T[size + 1];
    }

    public T getItem(int index)
    {
        return array[index];
    }

    public void setItem(int index, T value)
    {
        array[index] = value;
    }
}
```

```
static void Main(string[] args)
{
    Console.WriteLine("Vi du minh hoa Generic trong C#");
    Console.WriteLine("-----");

    //khai bao mot mang cac so nguyen
    TestGeneric<int> intArray = new TestGeneric<int>(5);
    //thiet lap cac gia tri
    for (int c = 0; c < 5; c++)
    {
        intArray.setItem(c, c * 5);
    }

    //lay va hien thi cac gia tri
    for (int c = 0; c < 5; c++)
    {
        Console.Write(intArray.getItem(c) + " ");
    }
}
```



Generic

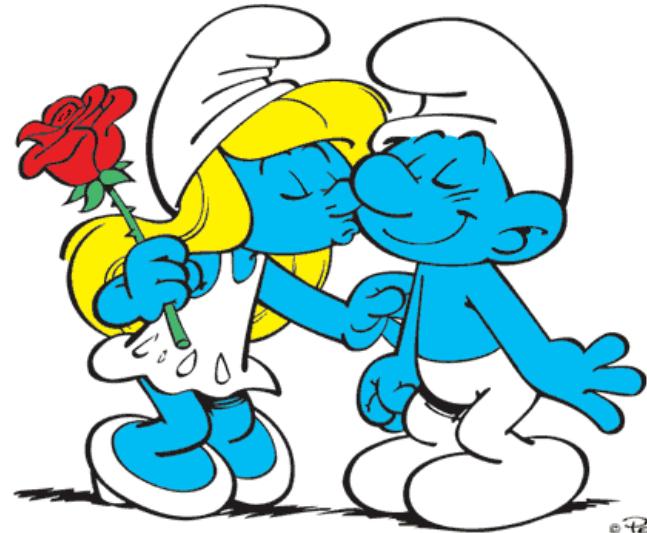
```
Console.WriteLine();

//khai bao mot mang ky tu
TestGeneric<char> charArray = new TestGeneric<char>(5);

//thiet lap gia tri
for (int c = 0; c < 5; c++)
{
    charArray.setItem(c, (char)(c + 97));
}

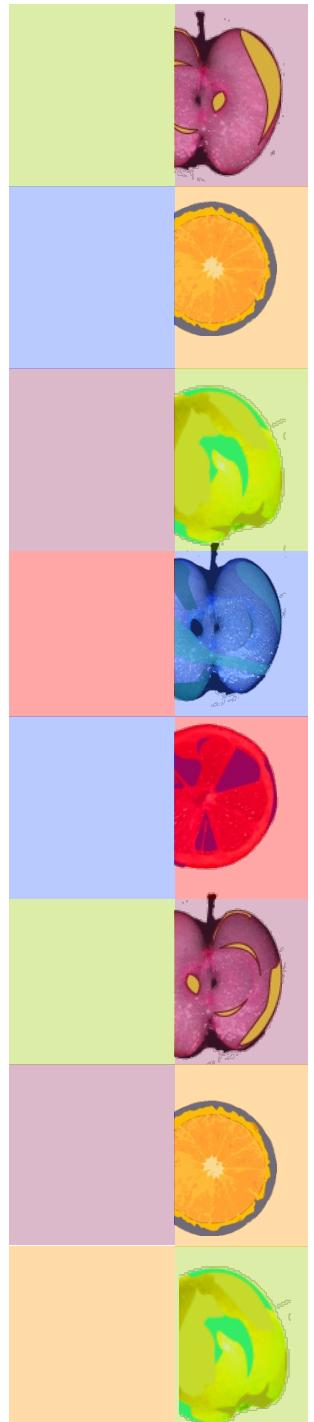
//lay va hien thi cac gia tri
for (int c = 0; c < 5; c++)
{
    Console.Write(charArray.getItem(c) + " ");
}
Console.WriteLine();

Console.ReadKey();
}
```



Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:

```
Vi du minh họa Generic trong C#
-----
0 5 10 15 20
a b c d e
```



Generic

- Các phương thức Generic trong C#

```
static void Swap<T>(ref T lhs, ref T rhs)
{
    T temp;
    temp = lhs;
    lhs = rhs;
    rhs = temp;
}
```

```
Generic trong C#
----- Vi du trao doi gia tri -----
Cac gia tri int truoc khi goi ham swap:
a = 10, b = 20
Cac gia tri char truoc khi goi ham swap:
c = I, d = V
Cac gia tri int sau khi goi ham swap:
a = 20, b = 10
Cac gia tri char sau khi goi ham swap:
c = V, d = I
```

```
static void Main(string[] args)
{
    Console.WriteLine("Generic trong C#");
    Console.WriteLine("----- Vi du trao doi gia tri -----");

    int a, b;
    char c, d;
    a = 10;
    b = 20;
    c = 'I';
    d = 'V';

    // Hien thi cac gia tri truoc khi trao doi:
    Console.WriteLine("Cac gia tri int truoc khi goi ham swap:");
    Console.WriteLine("a = {0}, b = {1}", a, b);
    Console.WriteLine("Cac gia tri char truoc khi goi ham swap:");
    Console.WriteLine("c = {0}, d = {1}", c, d);

    //goi ham swap de trao doi gia tri
    Swap<int>(ref a, ref b);
    Swap<char>(ref c, ref d);

    // Hien thi cac gia tri sau khi trao doi:
    Console.WriteLine("Cac gia tri int sau khi goi ham swap:");
    Console.WriteLine("a = {0}, b = {1}", a, b);
    Console.WriteLine("Cac gia tri char sau khi goi ham swap:");
    Console.WriteLine("c = {0}, d = {1}", c, d);

    Console.ReadKey();
}
```

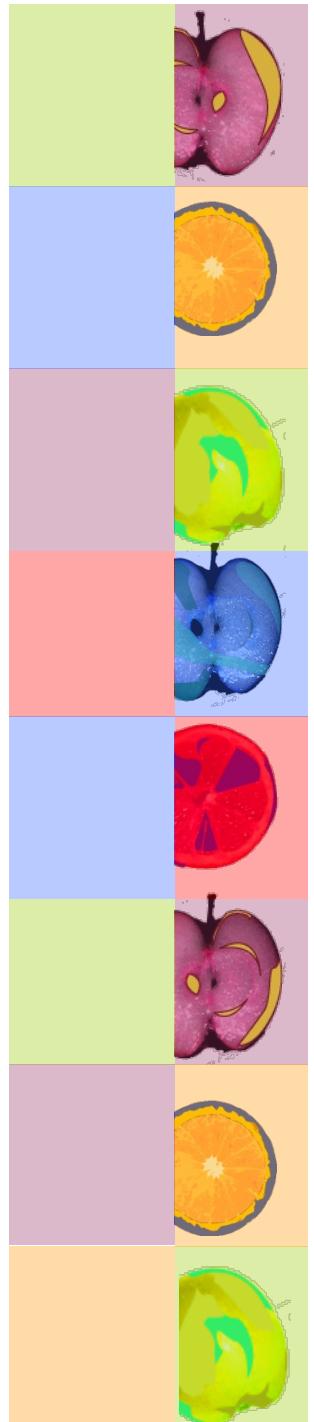
Generic Delegate trong C#

```
using System;
using System.Collections;

delegate T NumberChanger<T>(T n);

namespace VietJackCsharp
{
    class TestCsharp
    {
        static int num = 10;
        public static int AddNum(int p)
        {
            num += p;
            return num;
        }

        public static int MultNum(int q)
        {
            num *= q;
            return num;
        }
        public static int getNum()
        {
            return num;
        }
    }
}
```



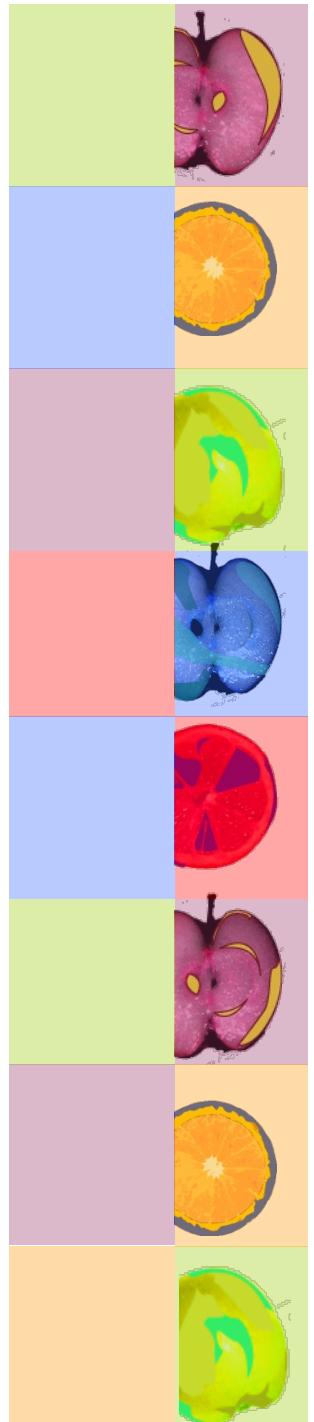
Generic Delegate trong C#

```
static void Main(string[] args)
{
    Console.WriteLine("Generic Delegate trong C#");
    Console.WriteLine("-----");

    //tao cac doi tuong delegate
    NumberChanger<int> nc1 = new NumberChanger<int>(AddNum);
    NumberChanger<int> nc2 = new NumberChanger<int>(MultNum);

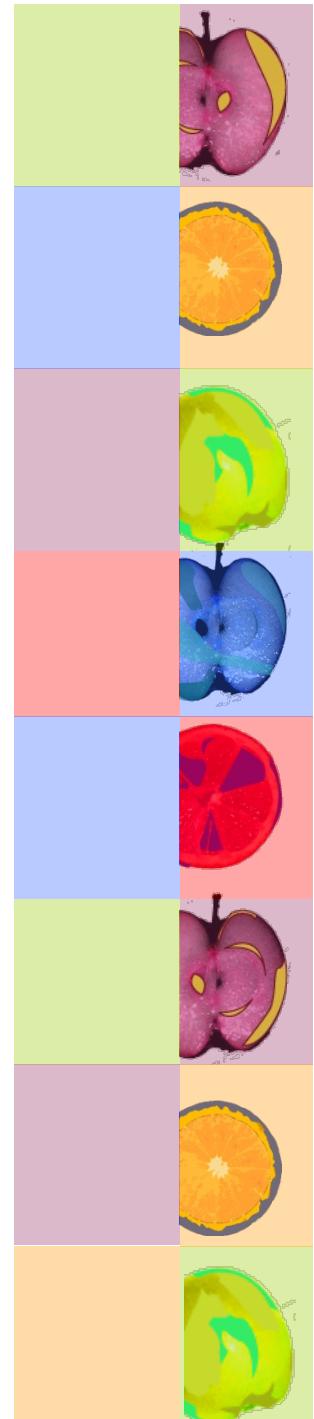
    //goi hai phuong thuc boi su dung doi tuong delegate
    nc1(25);
    Console.WriteLine("1 - Gia tri cua num la: {0}", getNum());
    nc2(5);
    Console.WriteLine("2 - Gia tri cua num la: {0}", getNum());
    Console.ReadKey();
}
```

```
Generic Delegate trong C#
-----
1 - Gia tri cua num la: 35
2 - Gia tri cua num la: 175
```



Đọc và ghi file

- Các lớp **StreamReader** và **StreamWriter** trong C# được sử dụng để đọc và ghi dữ liệu tới text file. Những lớp này kế thừa từ lớp abstract cơ sở là Stream, mà hỗ trợ việc đọc và ghi các byte vào trong File Stream.



```
static void Main(string[] args)
{
    Console.WriteLine("Vi du minh hoa doc File trong C#");
    Console.WriteLine("-----");

    try
    {
        // tao instance cua StreamReader de doc mot file.
        // lenh using cung duoc su dung de dong StreamReader.
        using (StreamReader sr = new StreamReader("textfile.txt"))
        {
            string line;

            // doc va hien thi cac dong trong file cho toi
            // khi tien toi cuoi file.
            while ((line = sr.ReadLine()) != null)
            {
                Console.WriteLine(line);
            }
        }

        Console.ReadKey();
    }
    catch (Exception e)
    {
        // thong bao loi.
        Console.WriteLine("Khong the doc du lieu tu file da cho: ");
        Console.WriteLine(e.Message);
    }
}
```



Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:

```
Vi du minh hoa doc File trong C#
-----
Hoc C# co ban va nang cao tai VietJack.
Day la chuong trinh minh hoa cach doc text file trong C#.
Moi ban tham khao loat bai: Bai tap C# co giao tieu tren trang cua chungtoi
```

Đọc và ghi file

```
static void Main(string[] args)
{
    Console.WriteLine("Vi du minh hoa doc va ghi File trong C#");
    Console.WriteLine("-----");

    string[] names = new string[] { "Tran Van A", "Nguyen Minh B" };
    using (StreamWriter sw = new StreamWriter("textfile.txt"))
    {

        foreach (string s in names)
        {
            sw.WriteLine(s);
        }
    }

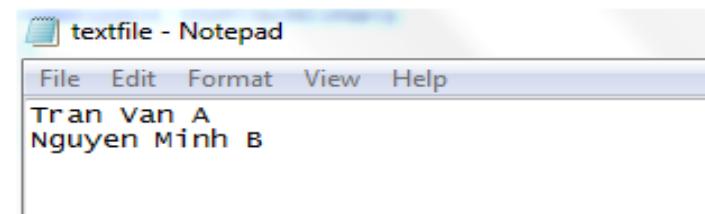
    // doc va hien thi du lieu trong textfile.txt
    string line = "";
    using (StreamReader sr = new StreamReader("textfile.txt"))
    {
        while ((line = sr.ReadLine()) != null)
        {
            Console.WriteLine(line);
        }
    }
    Console.ReadKey();
}
```



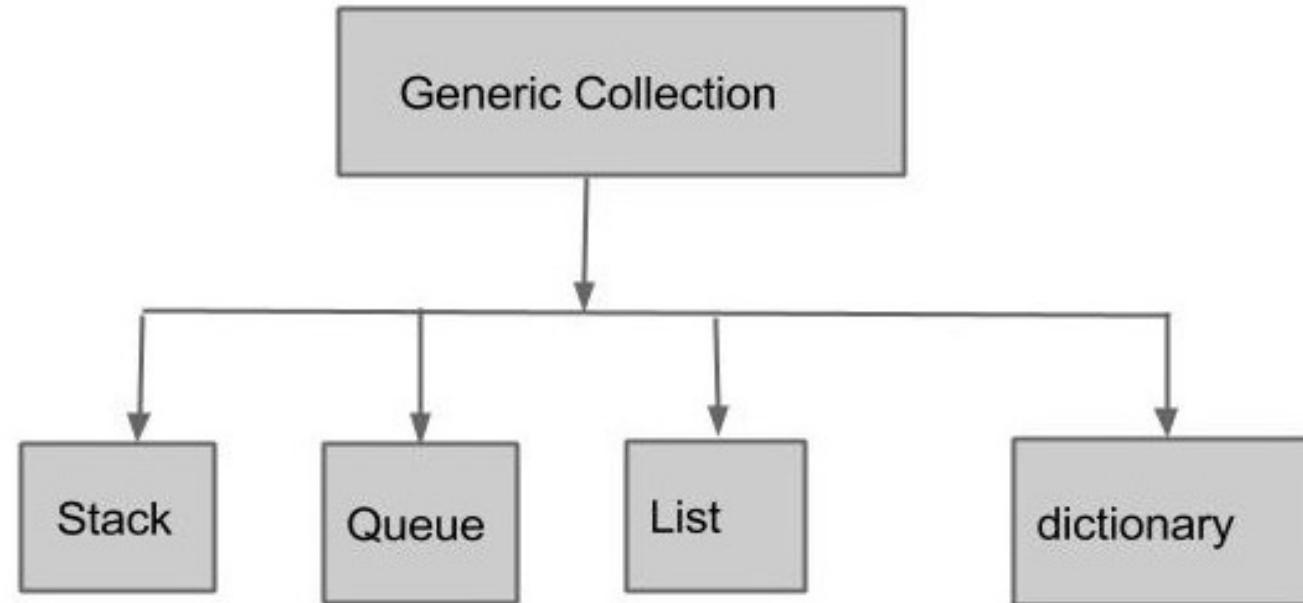
Biên dịch và chạy chương trình C# trên sẽ cho kết quả sau:

```
Vi du minh hoa doc va ghi File trong C#
Tran Van A
Nguyen Minh B
```

Mở textfile.txt và kiểm tra nội dung:



Collections

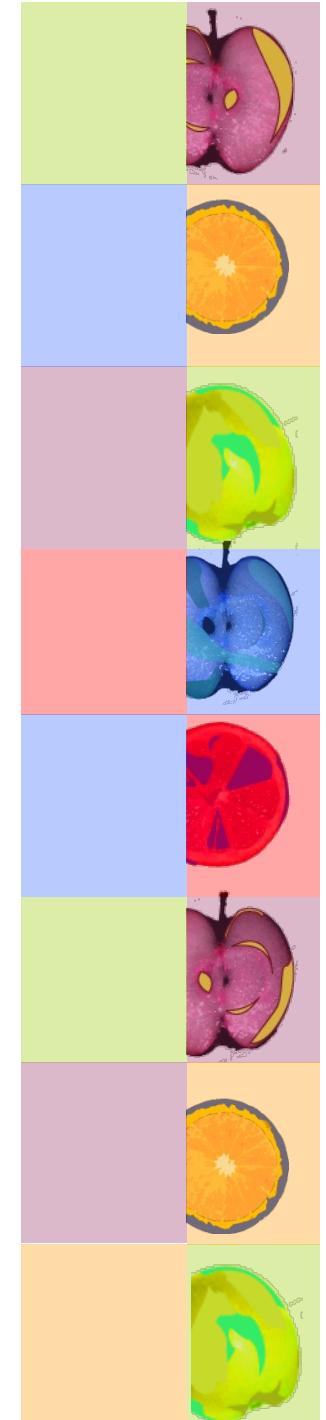


stack help
you to
access
data in
"LIFO"
sequence

Queue help
you to access
data in "FIFO"
sequence

List help you
to access data
from internal
index

Dictionary help
you to access data
by define key
value

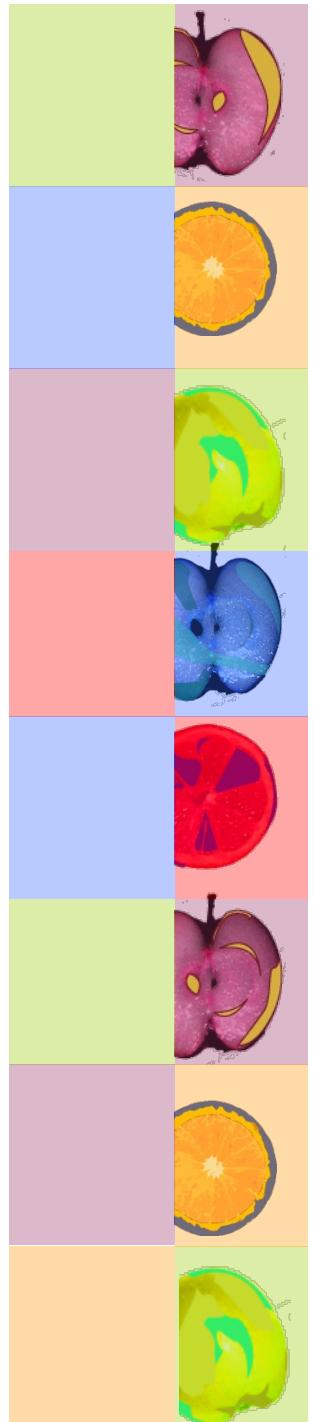


Stack

```
Stack stack = new Stack();
Stack<string> stack = new Stack<string>();
```

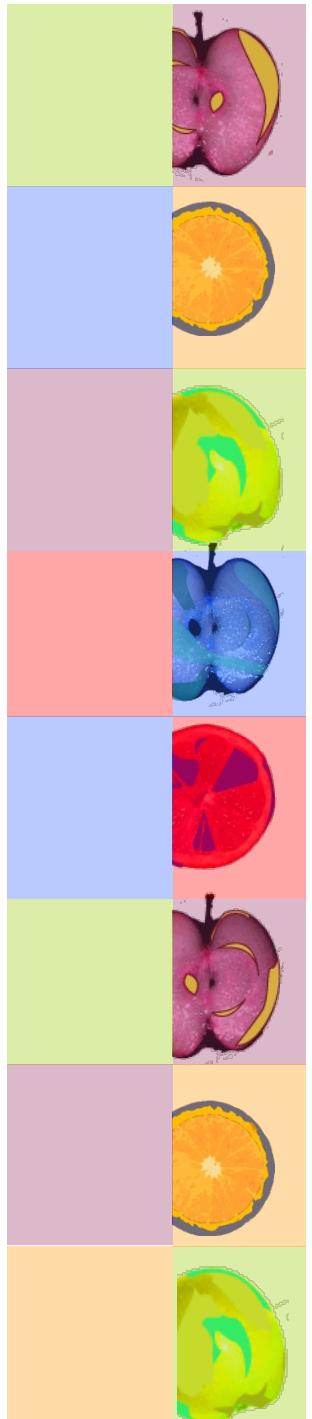
```
Stack<string> stack = new Stack<string>();
stack.Push("1");
stack.Push("2");
stack.Push("3");

while (stack.Count > 0)
{
    MessageBox.Show(stack.Pop());
}
```



Queue

```
Queue<string> queue = new Queue<string>();  
queue.Enqueue("1");  
queue.Enqueue("2");  
queue.Enqueue("3");  
  
while (queue.Count > 0)  
{  
    MessageBox.Show(queue.Dequeue());  
}
```



List

```
List<int> myInts = new List<int>();  
  
myInts.Add(1);  
myInts.Add(2);  
myInts.Add(3);  
  
for (int i = 0; i < myInts.Count; i++)  
{  
    Console.WriteLine("MyInts: {0}", myInts[i]);  
}
```



Dictionary

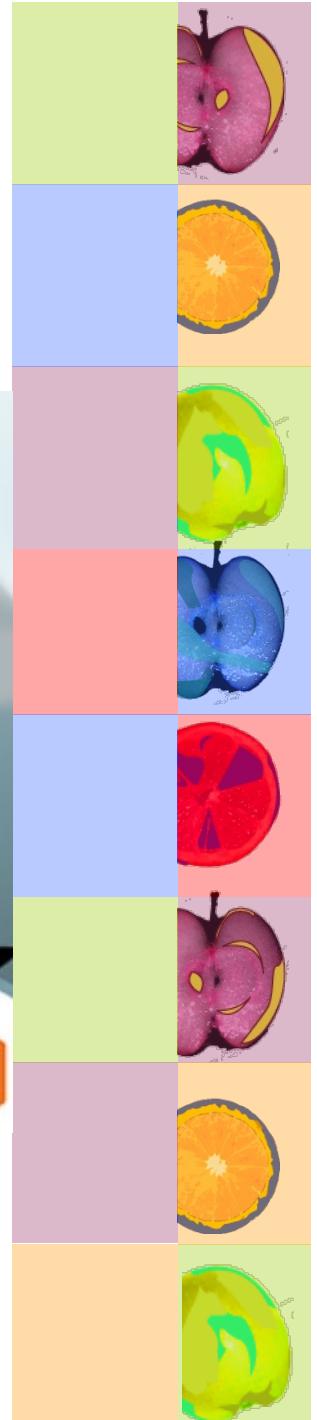
```
public class Customer
{
    public Customer(int id, string name)
    {
        ID = id;
        Name = name;
    }

    private int m_id;

    public int ID
    {
        get { return m_id; }
        set { m_id = value; }
    }

    private string m_name;

    public string Name
    {
        get { return m_name; }
        set { m_name = value; }
    }
}
```



Dictionary

```
static void Main(string[] args)
{
    List<int> myInts = new List<int>();

    myInts.Add(1);
    myInts.Add(2);
    myInts.Add(3);

    for (int i = 0; i < myInts.Count; i++)
    {
        Console.WriteLine("MyInts: {0}", myInts[i]);
    }

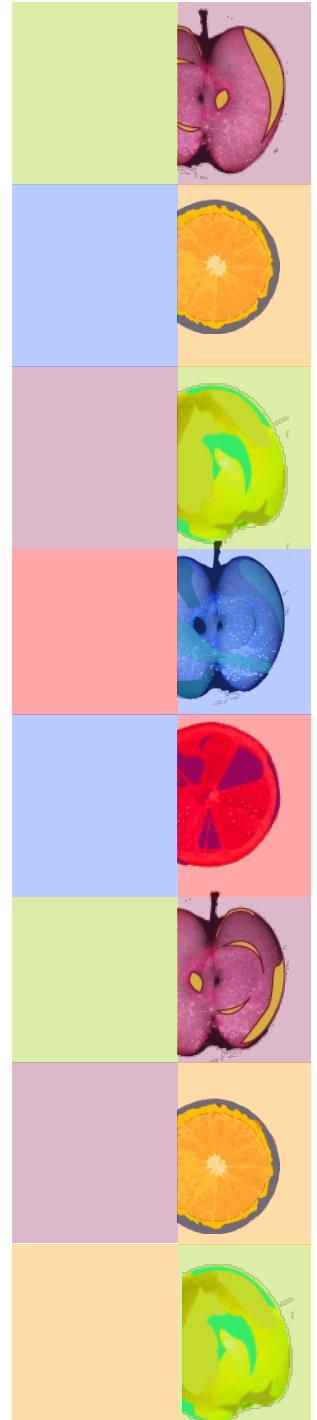
    Dictionary<int, Customer> customers = new Dictionary<int, Customer>();

    Customer cust1 = new Customer(1, "Cust 1");
    Customer cust2 = new Customer(2, "Cust 2");
    Customer cust3 = new Customer(3, "Cust 3");

    customers.Add(cust1.ID, cust1);
    customers.Add(cust2.ID, cust2);
    customers.Add(cust3.ID, cust3);

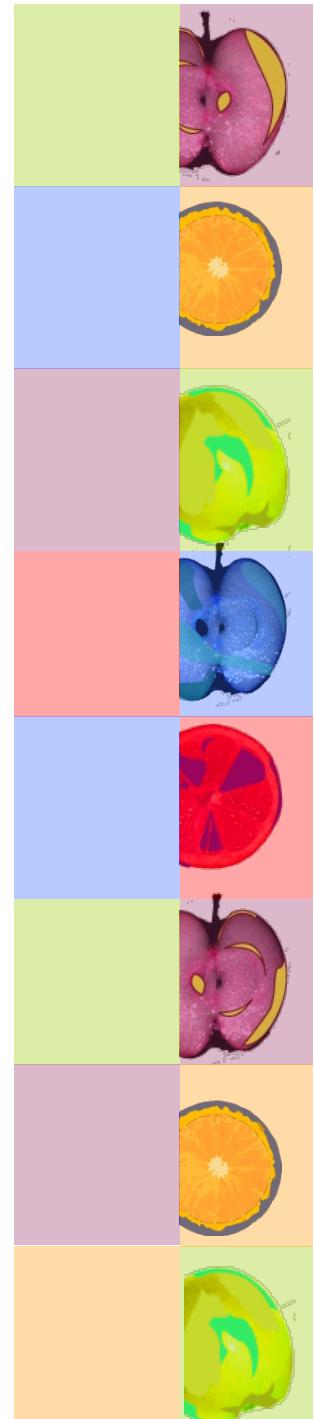
    foreach (KeyValuePair<int, Customer> custKeyVal in customers)
    {
        Console.WriteLine(
            "Customer ID: {0}, Name: {1}",
            custKeyVal.Key,
            custKeyVal.Value.Name);
    }

    Console.ReadKey();
}
```



Kiểu Var và Dynamic trong C#

var	dynamic
Introduced in C# 3.0	Introduced in C# 4.0
Statically typed – This means the type of variable declared is decided by the compiler at compile time.	Dynamically typed - This means the type of variable declared is decided by the compiler at runtime time.
Need to initialize at the time of declaration. e.g., <code>var str="I am a string";</code> Looking at the value assigned to the variable <code>str</code> , the compiler will treat the variable <code>str</code> as string.	No need to initialize at the time of declaration. e.g., <code>dynamic str;</code> <code>str="I am a string"; //Works fine and compiles</code> <code>str=2; //Works fine and compiles</code>
Errors are caught at compile time. Since the compiler knows about the type and the methods and properties of the type at the compile time itself	Errors are caught at runtime Since the compiler comes to about the type and the methods and properties of the type at the run time.



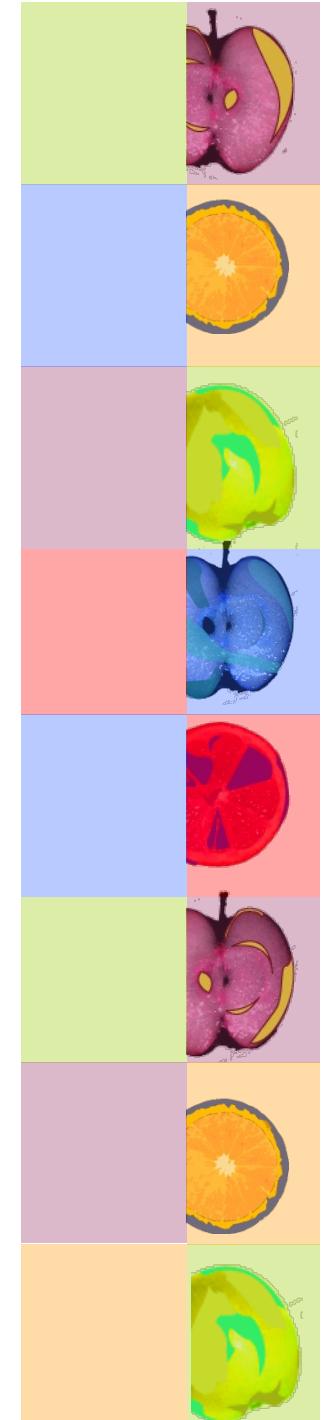
Kiểu Dynamic

- Các kiểu dynamic cho trình biên dịch rằng object được định nghĩa là dynamic và bỏ qua check kiểu tại thời điểm biên dịch; check kiểu bị trễ cho đến khi runtime

*Compile-time type
dynamic*

*Run-time type
System.Int32*

```
dynamic x = 1;  
dynamic y = "Hello";  
dynamic z = new List<int> { 1, 2, 3 };
```

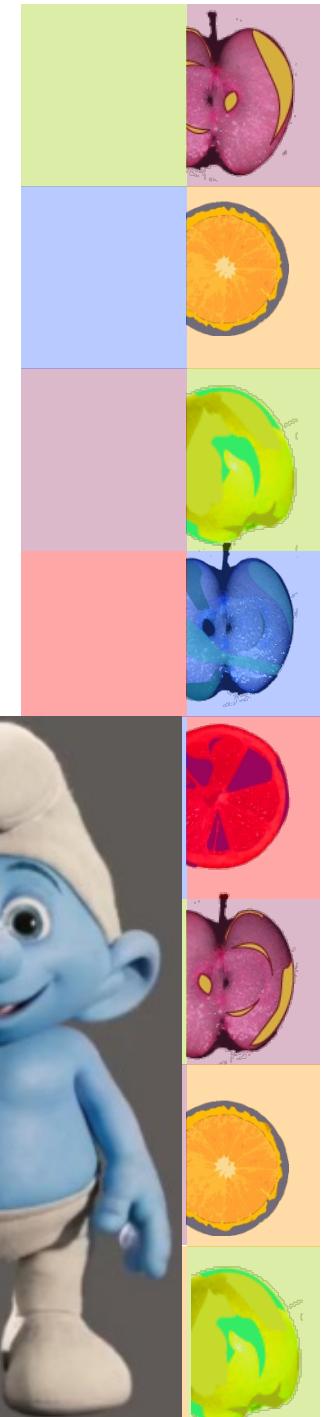


Kiểu Dynamic

```
01 static void ChangeDynamicDataType()
02 {
03     dynamic t = "Hello!";
04     Console.WriteLine("t is of type: {0}", t.GetType());
05
06     t = false;
07     Console.WriteLine("t is of type: {0}", t.GetType());
08
09     t = new System.Collections.Generic.List<int>();
10    Console.WriteLine("t is of type: {0}", t.GetType());
11 }
12 </int>
```

Output:

```
t is of type: System.String
t is of type: System.Boolean
t is of type: System.Collections.Generic.List`1[System.Int32]
```





Kiểu Dynamic

*Method chosen at compile-time:
double Abs(double x)*

```
double x = 1.75;  
double y = Math.Abs(x);
```

```
dynamic x = 1.75;  
dynamic y = Math.Abs(x);
```

```
dynamic x = 2;  
dynamic y = Math.Abs(x);
```

```
public static class Math  
{  
    public static decimal Abs(decimal value);  
    public static double Abs(double value);  
    public static float Abs(float value);  
    public static int Abs(int value);  
    public static long Abs(long value);  
    public static sbyte Abs(sbyte value);  
    public static short Abs(short value);  
}
```

*Method chosen at run-time:
double Abs(double x)*

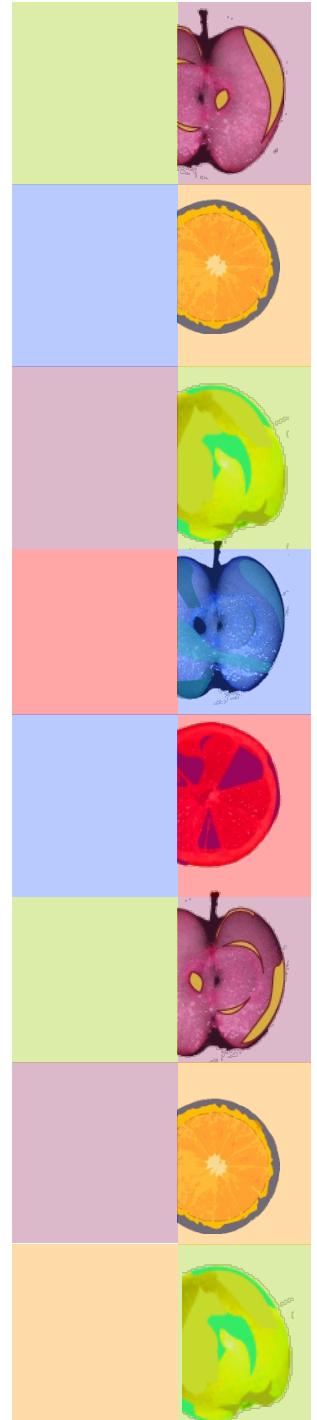
*Method chosen at run-time:
int Abs(int x)*

Kiểu Var



- Từ khóa var hỗ trợ khởi tạo biến, mảng mà không cần khai báo kiểu dữ liệu, kiểu dữ liệu sẽ được xác định khi gán giá trị cho biến, lúc này chương trình sẽ tự ép kiểu cho biến.
- Và tất nhiên sau khi đã gán giá trị thì các biến, mảng đó đã có một kiểu dữ liệu xác định và không thể thay đổi.

```
static void Main(string[] args)
{
    var varInt = 1;
    var varChar = 'a';
    var varString = "abcdef";
    Console.WriteLine("{0}\n{1}\n{2}", varInt, varChar, varString);
    Console.Read();
}
```

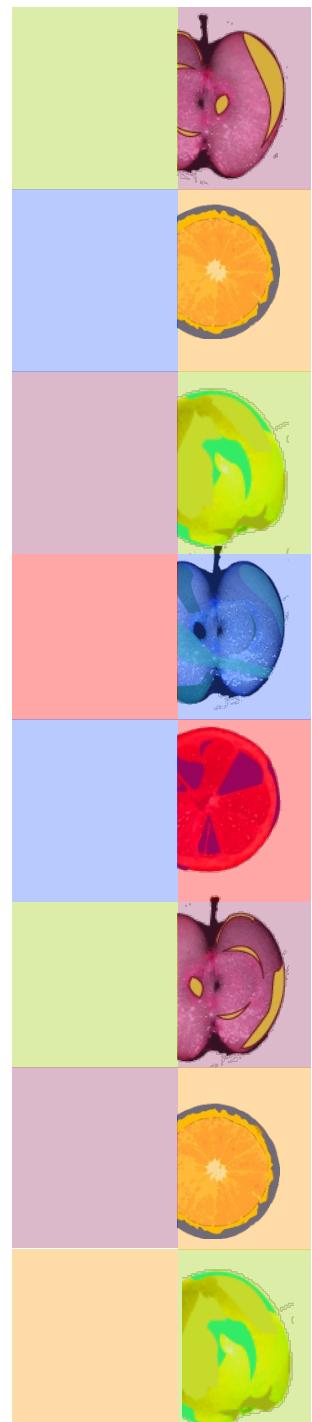


Kiểu Var

```
class Program
{
    static void Main(string[] args)
    {
        var v = new { Amount = 108, Message = "Hello" };

        Console.WriteLine(v.Amount + v.Message);

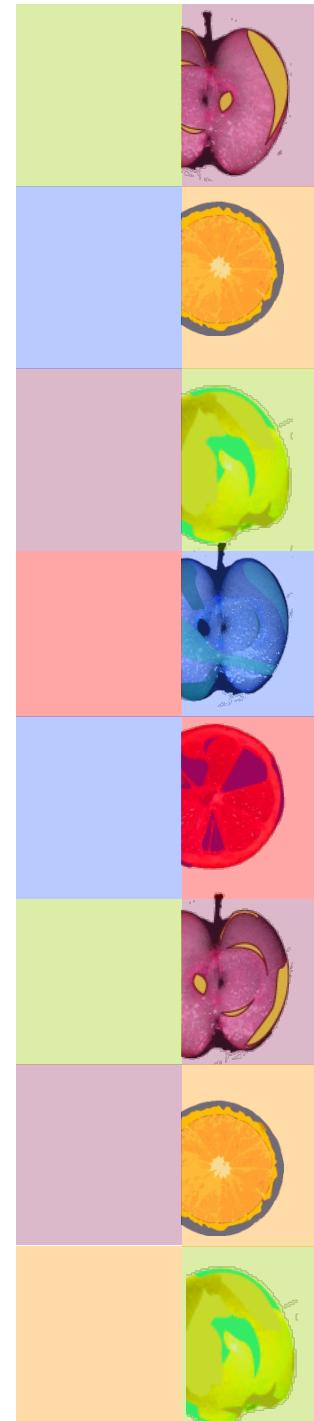
        Console.Read();
    }
}
```



Phân biệt cách dùng var, object và dynamic

- Từ khóa object mô tả cho kiểu System.Object, là lớp cha trên cùng trong phân cấp lớp C#, và có thể diễn tả cho bất cứ lớp nào khác. Nó được sử dụng khi không thể định danh được kiểu đối tượng tại thời điểm biên dịch và bạn cần sử dụng lệnh chuyển đổi sang một kiểu định nghĩa trước nếu muốn sử dụng các phương thức hoặc thuộc tính của kiểu đó.
-

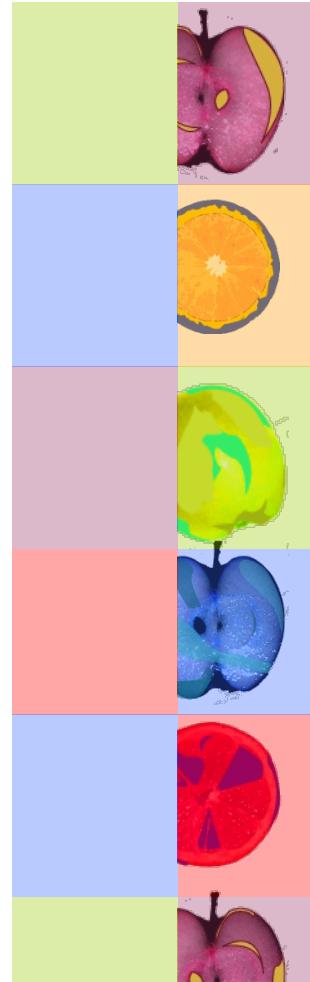
```
1 static void UseObjectVariable()
2 {
3     // Giả sử có một lớp tên Person.
4     object o = new Person() { FirstName = "Thanh", LastName = "Ho" };
5
6     // Phải đổi kiểu đối tượng thành Person
7     // để có thể sử dụng các thuộc tính của Person.
8     Console.WriteLine("Person's first name is {0}", ((Person)o).FirstName);
9 }
```



Phân biệt cách dùng var, object và dynamic

- Từ khóa var được đưa ra từ C# 3.0, được sử dụng đại diện các kiểu đã được định nghĩa sẵn hoặc kiểu nặc danh. Một biến được khai báo var sẽ được định danh kiểu tại thời điểm biên dịch và không thể thay đổi kiểu lúc runtime. Nếu trình biên dịch không thể xác định được kiểu, nó sẽ tạo ra lỗi biên dịch:

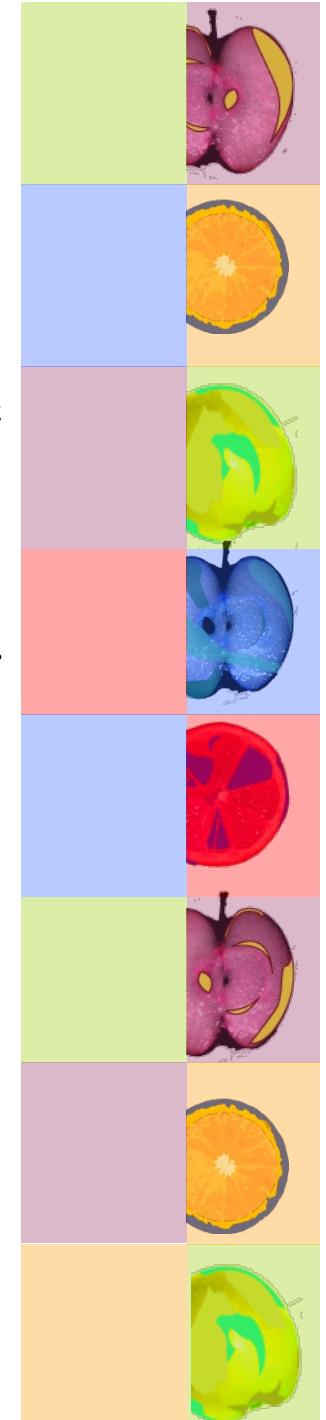
```
1 var varExample = 10;
2 Console.WriteLine(varExample.GetType()); // In ra System.Int32
3
4 //Biến varExample là kiểu System.Int32:
5 varExample = varExample + 10;
6
7 //Dòng này sẽ bị lỗi khi biên dịch và chỉ có thể gán giá trị số nguyên cho varExample:
8 varExample = "test";
```



Phân biệt cách dùng var, object và dynamic

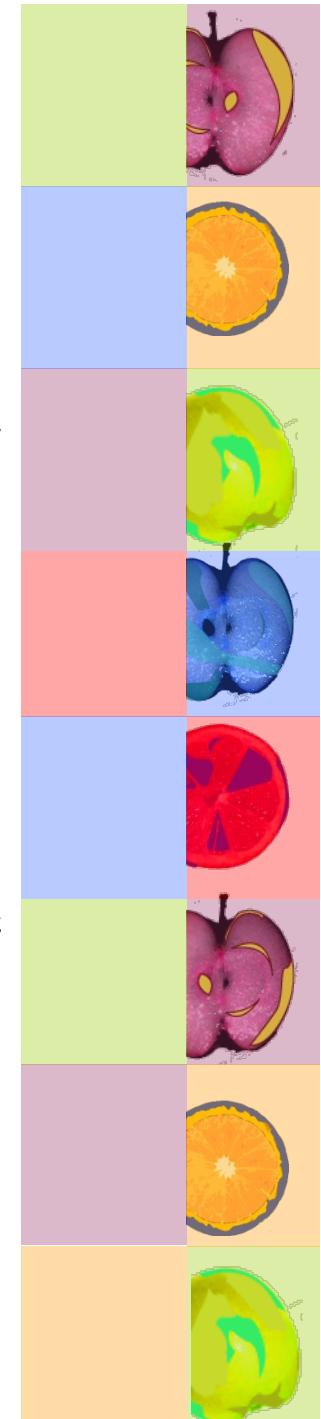
- Từ khóa dynamic, được giới thiệu trong C# 4, tạo ra các kịch bản khác dựa trên từ khóa object truyền thống nhưng đơn giản hơn trong viết và maintain code. Thực tế, kiểu dynamic sử dụng kiểu System.Object nhưng không như object, nó không đòi hỏi việc đổi kiểu lúc biên dịch, mà chỉ định danh thời điểm chạy:

```
1 static void UseDynamicVariable()
2 {
3     // Giả sử có một lớp tên Person.
4     dynamic p = new Person() { FirstName = "Thanh", LastName = "Ho" };
5
6     //Có thể sử dụng các thuộc tính của Person mà không cần đổi kiểu.
7     Console.WriteLine("Person's first name is {0}", p.FirstName);
8 }
```



Extension

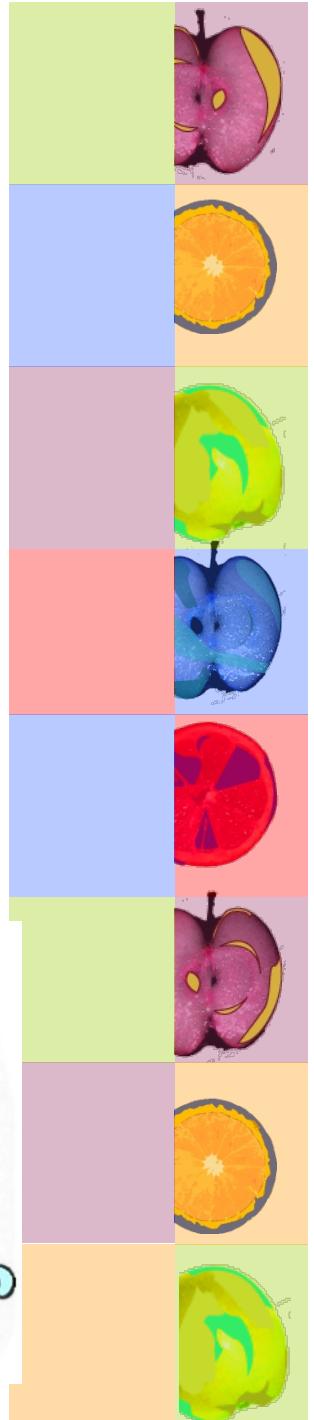
- Extension method giúp bạn tạo thêm các phương thức cho một lớp mà không cần thừa kế lại lớp đó.
- **Các quy tắc khi định nghĩa và sử dụng extension method**
 - Lớp chứa extension method phải là static
 - Extension method cũng phải là một phương thức static
 - Tham số đầu tiên của extension method xác định kiểu của đối tượng được sử dụng (extension method sẽ thêm vào lớp của đối tượng đó) với từ khóa **this**.



Extension

- Tạo một phương thức instance để tất cả các đối tượng có thể chuyển đổi nhanh chóng sang kiểu Int32. Cách thực hiện rất đơn giản là bạn tạo thêm một lớp với phương thứcToInt32()
- Tham số duy nhất của phương thức trên là một kiểu object, tức là tôi muốn phương thứcToInt32() này được gắn vào tất cả các đối tượng thừa kế từ object

```
1 public static class Y2Extensions  
2 {  
3     public static intToInt32(this object obj)  
4     {  
5         return Int32.Parse(obj.ToString());  
6     }  
7 }
```



Extension

- Sau khi định nghĩa, bạn có thể sử dụng phương thức ToInt32() này như một phương thức instance của lớp object

```
1  namespace TestExtensions
2  {
3      class Program {
4
5          public static void Main(){
6
7              string s="100";
8              int i=s.ToInt32();
9              i++;
10             Console.WriteLine(i);
11             Console.ReadKey();
12         }
13     }
14
15     public static class Y2Extensions
16     {
17         public static intToInt32(this object obj)
18         {
19             return Int32.Parse(obj.ToString());
20         }
21     }
22 }
```



Output:

“

101

Extension

```
public static class StringExtension  
{  
    public static int ConvertToNumber(this string str)  
    {  
        int result;  
        if (int.TryParse(str, out result))  
        {  
            return result;  
        }  
        return 0;  
    }  
  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            string year = "2014";  
            int y = year.ConvertToNumber();  
        }  
    }  
}
```

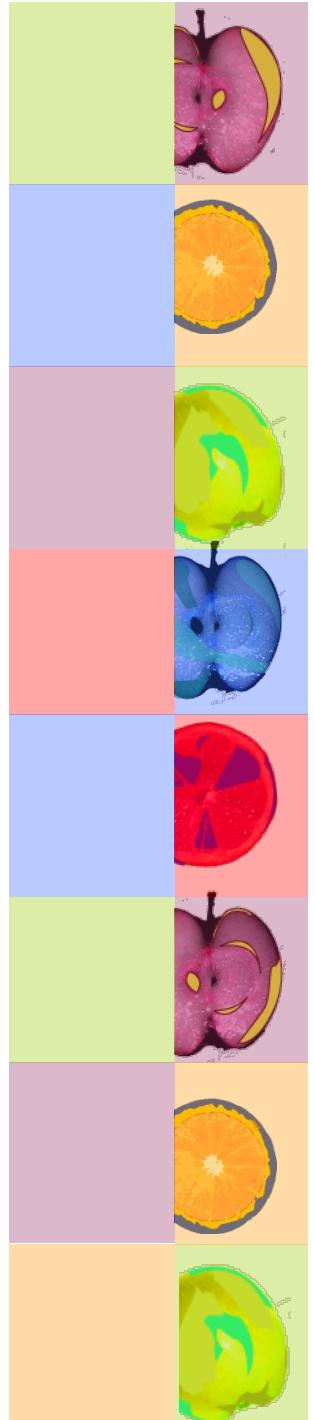
Create new Extension method ConvertToNumber for string type



Use ConvertToNumber method to convert string to int

Anonymous Method (Phương thức nặc danh)

- Chúng ta đã bàn luận rằng Delegate được sử dụng để tham chiếu bất kỳ phương thức nào mà có cùng dấu hiệu như trong Delegate đó.
- **Phương thức nặc danh (Anonymous Method)** trong C# cung cấp một kỹ thuật để truyền một khối code như là một tham số delegate. Các phương thức nặc danh là các phương thức không có tên, chỉ có thân phương thức.
- Bạn không cần xác định kiểu trả về trong một phương thức nặc danh; nó được suy ra từ lệnh return bên trong thân phương thức nặc danh đó.



Anonymous Method (Phương thức nặc danh)

- Các phương thức nặc danh (Anonymous Method) trong C# được khai báo với việc tạo instance của Delegate đó, với một từ khóa **delegate**. Ví dụ:

```
delegate void NumberChanger(int n);
...
NumberChanger nc = delegate(int x)
{
    Console.WriteLine("Phuong thuc nac danh: {0}", x);
};
```

- Khoi `Console.WriteLine("Anonymous Method: {0}", x);` là phần thân của phương thức nặc danh.



Anonymous Method (Phương thức nặc danh)

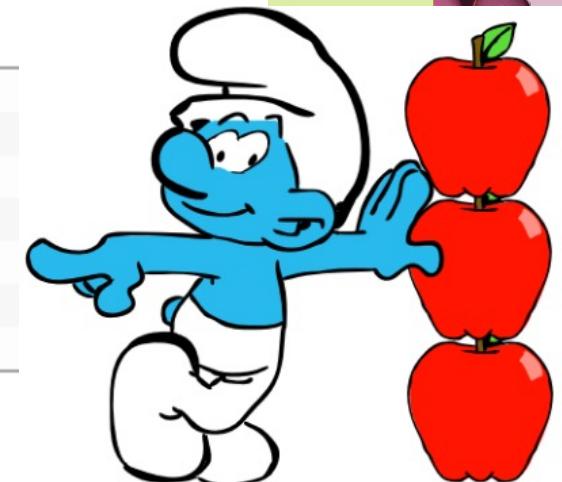
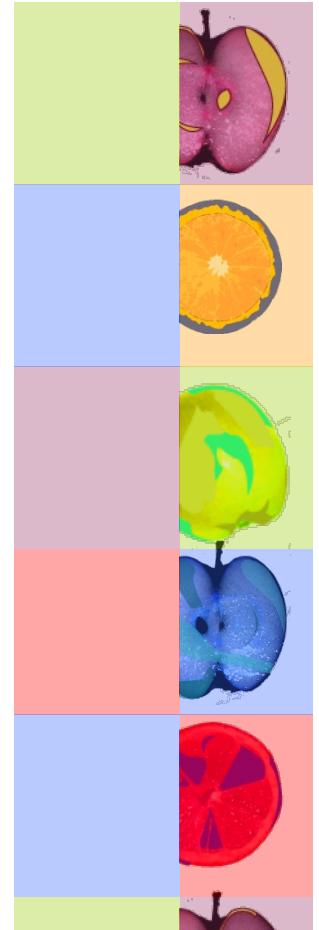
- Ví dụ về phương thức thường được sử dụng là FindAll dùng để tìm tất cả các phần tử thỏa mãn điều kiện, nó được định nghĩa như sau:

```
T[] Array.FindAll(T[] array,  
Predicate match)
```

- Dùng phương thức nặc danh

```
int[] array = { 10, 4, 3, 2, 8, 6, 5, 7, 9, 1 };
```

```
1 int[] evens = Array.FindAll(array, delegate(int n)  
2 {  
3     return n % 2 == 0;  
4 } );  
5  
6  
7 }
```



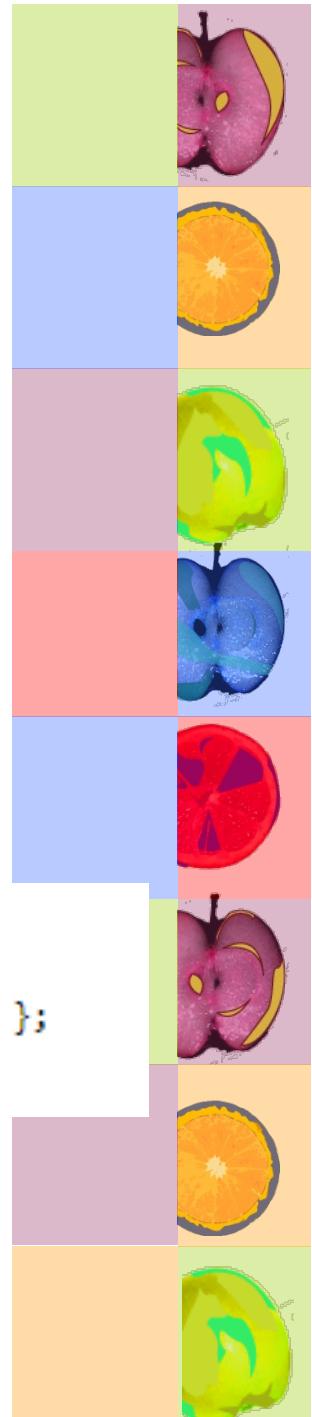
Lambda

- Khi dùng anonymous method, bạn có thể tạo các hàm inline nhằm hạn chế việc khai báo các hàm riêng lẻ không cần thiết, giúp mã lệnh ngắn gọn hơn.
- Nay với lambda expression, bạn có thể viết ngắn gọn và dễ dàng hơn nhờ việc cung cấp toán tử và cú pháp mới, đồng thời thể hiện sự “thông minh” của compiler bằng cách tự nhận diện kiểu của dữ liệu.

```
delegate void TestDelegate(string s);
...
TestDelegate myDel = n => { string s = n + " " + "World"; Console.WriteLine(s); };
myDel("Hello");

public delegate TResult Func<TArg0, TResult>(TArg0 arg0)

Func<int, bool> myFunc = x => x == 5;
bool result = myFunc(4); // returns false of course
```

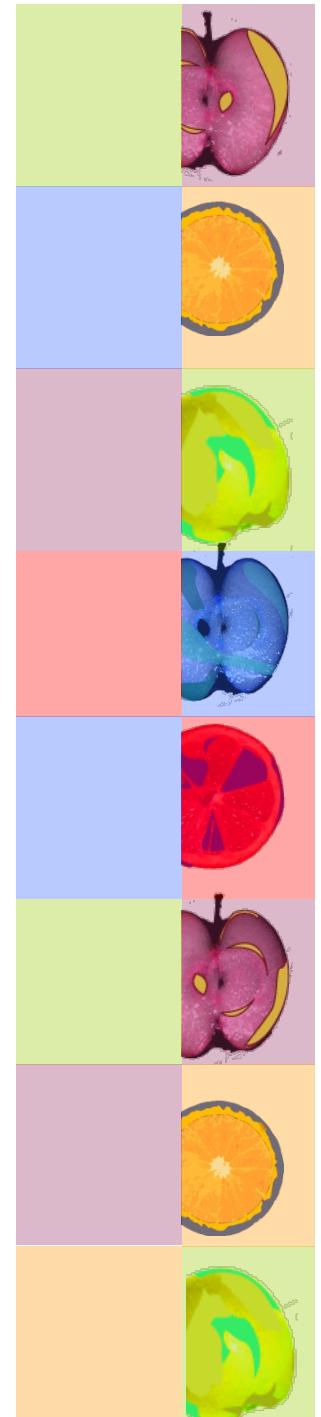


Lambda

- Bạn có một mảng các số nguyên kiểu int sau:

```
1 | int[] numbers = { 10, 4, 3, 2, 8, 6, 5, 7, 9, 1 };
```

- Bây giờ ta sẽ dùng anonymous method để in các phần tử trong mảng trên ra.
- Ta dùng phương thức tĩnh
Array.ForEach<T>(T[] array, Action<T> action),
ở đây có thể bỏ đi phần định kiểu T.
- Compiler sẽ tự động hiểu và xác định kiểu dựa vào kiểu của tham số bạn truyền vào. Cụ thể ta viết như sau:





Lambda

C# 2.0:

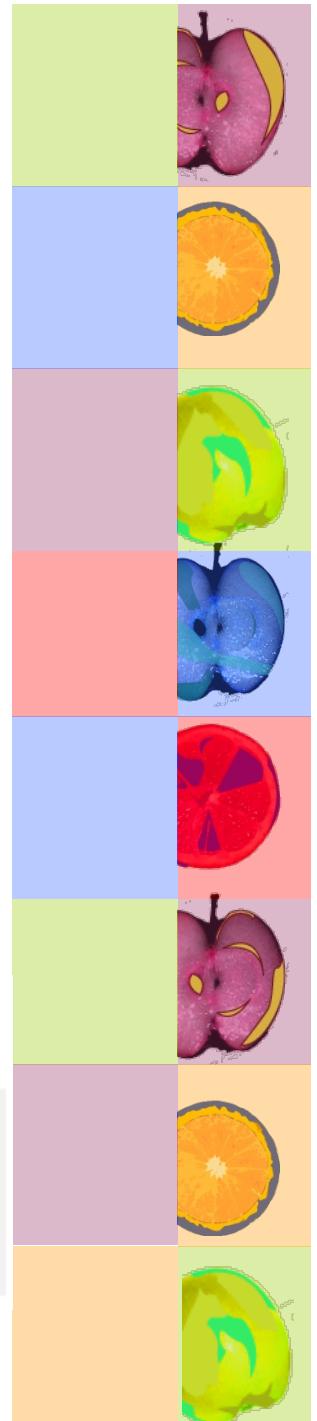
```
1 | Array.ForEach(numbers, delegate(int x){  
2 |     Console.WriteLine(x);  
3 | });
```

- Trong C# 3.0 ta có thể dùng cú pháp ngắn gọn hơn nữa để thực hiện điều này với cách hoạt động tương tự như với anonymous method. Ta viết lại đoạn mã trên như sau:

C# 3.0:

“

```
Array.ForEach(numbers, (int x) => { Console.WriteLine(x); });
```



Lambda

Ta sẽ sắp xếp mảng **numbers** trên theo thứ tự giảm dần. Ta sẽ sắp xếp bằng phương thức

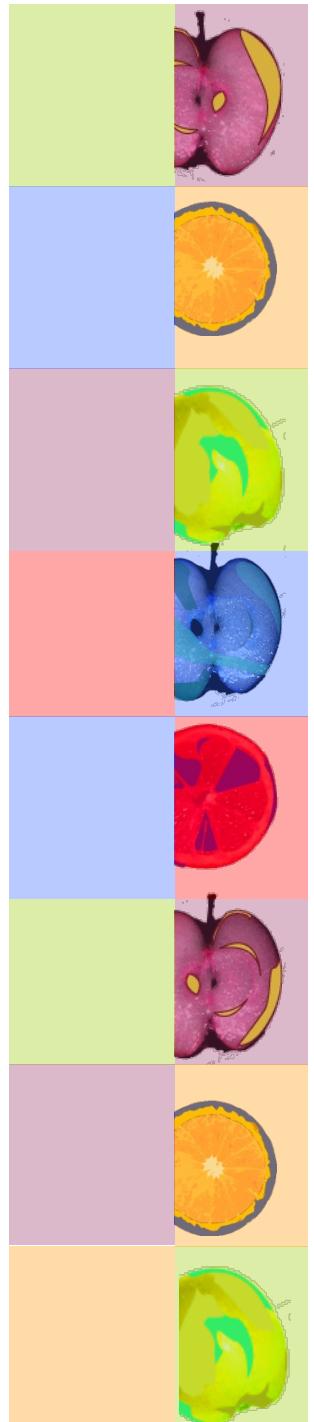
Array.Sort<T>(T[] array, Comparision<T> comparision).

- Anonymous method:

```
“
    Array.Sort(numbers, delegate(int x, int y)
    {
        return y.CompareTo(x);
    });
”
```

-Lambda expression:

```
“
    Array.Sort(numbers, (x,y) => y.CompareTo(x));
”
```



Lambda

Vẫn sử dụng các phương thức tĩnh trong lớp Array, ta sẽ kiểm tra xem các phần tử trong mảng numbers có tuân theo một quy luật nào đó không bằng cách sử dụng phương thức

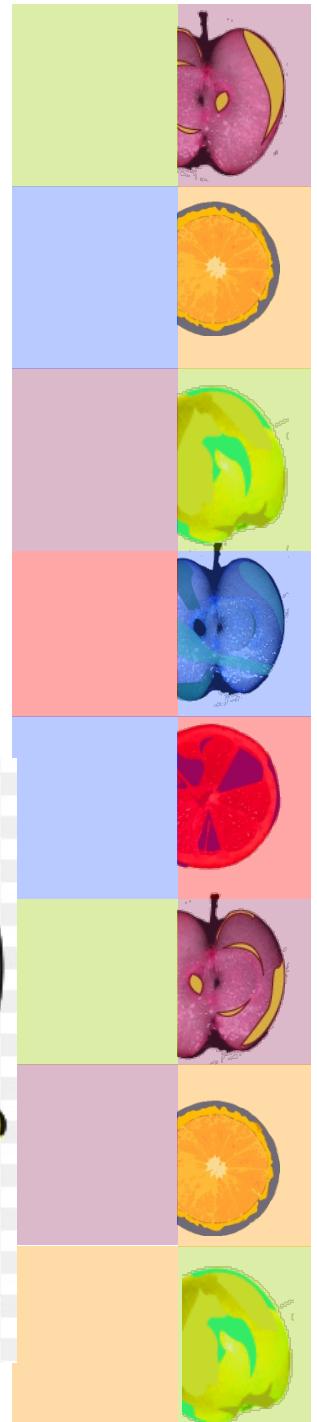
Array.TrueForAll<T>(T[] array, Predicate<T> match).

- Anonymous method:

```
“  
bool b=Array.TrueForAll(numbers, delegate(int x){  
  
    return x<11;  
  
});
```

-Lambda expression:

```
“  
bool b = Array.TrueForAll(numbers, x => x < 11);
```



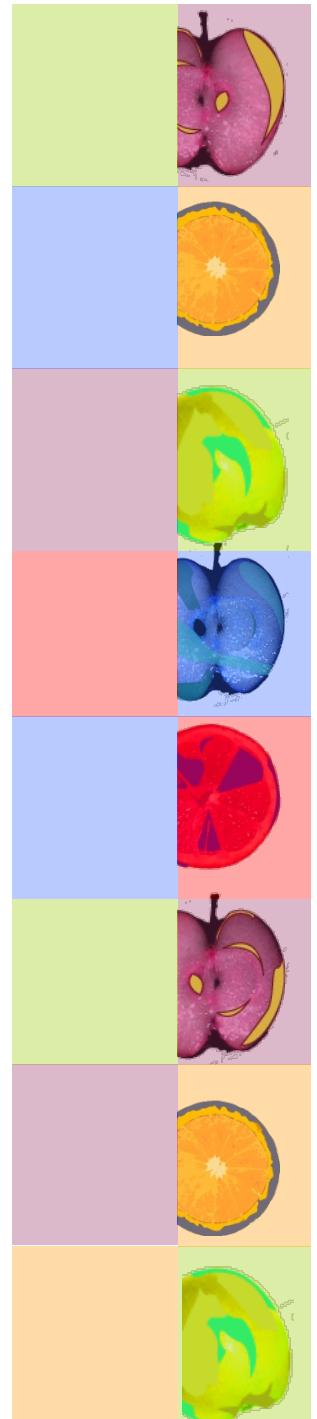
Lambda

- **Delegate Func<...>:**

- Func cho phép khai báo và tạo ra các dạng delegate với số lượng tham số và kiểu trả về khác nhau, tương tự như khi bạn tạo ra một phương thức.
- Func được dùng chủ yếu để tạo và lưu trữ một anonymous method ngắn gọn bằng lambda expression và được sử dụng như những phương thức thông thường
- Cú pháp để sử dụng Func là viết các kiểu của tham số và giá trị trả về vào cặp ngoặc '<>', theo sau từ khóa Func.
- Ví dụ: Yêu cầu một tham số kiểu string và trả về một giá trị int: lấy chiều dài của chuỗi.

“

```
Func<string, int> stringLength = s => s.Length;  
  
Console.WriteLine(stringLength ("yinyang"));
```



Lambda

+ Yêu cầu 1 tham số float, 1 tham số và trả về kiểu float: tính tích của 2 số

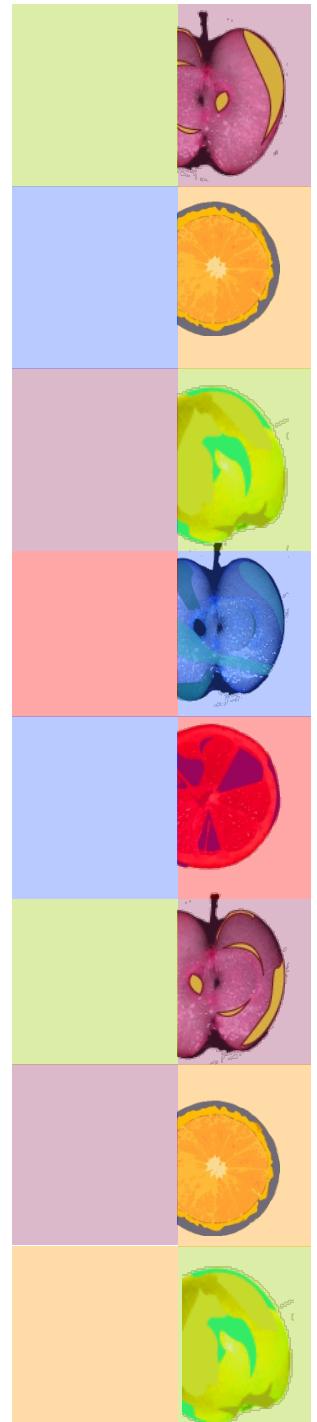
“

```
Func<float, int, float> multiply = (x, y) => x * y;  
  
Console.WriteLine(multiply(9f, 7));
```

+ Trả về max của 2 số:

“

```
Func<int, int, int> max = (x, y) => x > y ? x : y;  
  
Console.WriteLine(max(9, 7));
```



THANK YOU

