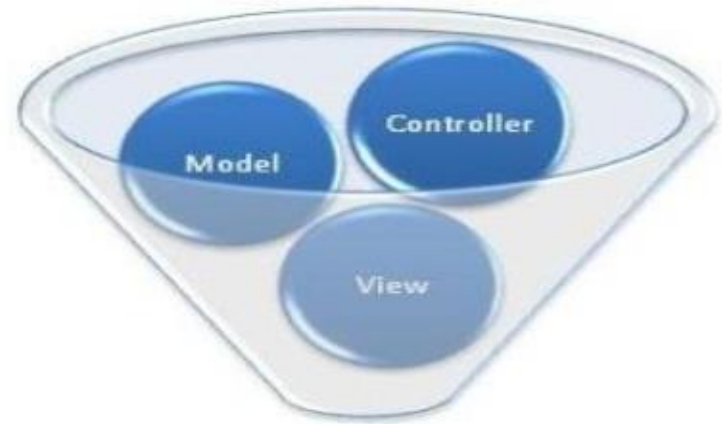


Microsoft®

ASP.net MVC **5**



NEW FEATURE IN C#



☐ Kỹ thuật mới trong C#

- ☒ Thuộc tính tự động
- ☒ Khởi tạo đối tượng
- ☒ Biến cục bộ tự suy
- ☒ Kiểu nặc danh
- ☒ Phương thức mở rộng
- ☒ Khởi tạo danh sách

☐ Xử lý thời gian


☐ Biểu thức chính qui

```
public class Student  
{
```

```
    private String _Name;  
    public String Name  
    {  
        get  
        {  
            return _Name;  
        }  
        set  
        {  
            _Name = value;  
        }  
    }  
}
```

```
public class Student  
{
```

```
    public String Name { get; set; }  
}
```



Tự sinh trường
để lưu dữ liệu
của thuộc tính

```
public class Student
{
    public String Name { get; set; }
    public Double Marks { get; set; }
}
```

Cung cấp giá trị cho các thuộc tính cần thiết lúc khởi tạo

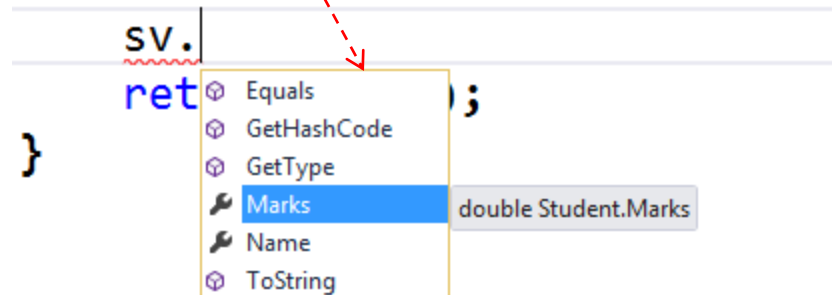
```
public ActionResult Index()
{
    Student sv = new Student
    {
        Name = "Nguyễn Nghiêm",
        Marks = 9
    };
    return View();
}
```

```
public class Student
{
    public String Name { get; set; }
    public Double Marks { get; set; }
}
```

Tự nhận biết kiểu
thông qua giá trị
gán cho biến.

```
public ActionResult Index()
{
    var sv = new Student
    {
        Name = "Nguyễn Nghiêm",
        Marks = 9
    };

    sv.  
ret
```



double Student.Marks

- ❑ Bạn có thể tạo đối tượng mà không cần định nghĩa lớp.
- ❑ Đối tượng có kiểu nặc danh không thể truyền cho view hoặc chia sẻ với các thành phần khác được

```
public ActionResult Index()  
{  
    var employee = new  
    {  
        Name = "Nguyễn Nghiêm",  
        Salary = 1000  
    };  
}
```

employee.

return Vi

}

- Equals
- GetHashCode
- GetType
- Name
- Salary**
- ToString

int 'a.Salary

Anonymous Types:

'a is new { string Name, int Salary }

- ❑ Bạn có thể viết các phương thức bổ sung cho một lớp đã tồn tại trước đó mà bạn không có mã nguồn.

Theo qui ước

```
public ActionResult Index()
{
    var name = "Nguyễn Nghiệm".T;
    return View();
}
```

- Split
- StartsWith
- Substring
- Sum<>
- Take<>
- TakeWhile<>
- ToArray<>
- ToBase64**
- ToCharArray

```
public static class XString
{
    public static String ToBase64(this String s)
    {
        byte[] data = Encoding.UTF8.GetBytes(s);
        return Convert.ToBase64String(data);
    }
}
```

**Lớp được bổ
sung phương
thức**

```
var list = new List<Student>
{
    new Student {Name="Tuấn", Marks=5},
    new Student {Name="Hoa", Marks=7}
};
```

Danh sách có định
kiểu

list[0].

- Equals
- GetHashCode
- GetType
- Marks**
- Name
- ToString

double Student.Marks

Liệt kê các phần tử cách
nhau bởi dấu phẩy

Danh sách không
định kiểu

```
var list = new ArrayList
{
    new {Name="Tuấn", Marks=5},
    new {Name="Hoa", Marks=7}
};
```


- ❑ Chuyển đổi kiểu dữ liệu từ chuỗi nhằm có các hoạt động thao tác đúng với kiểu mong muốn.
- ❑ C# có 2 cách để chuyển đổi kiểu
 - ~~✍~~ **<kiểu> x = <kiểu>.Parse(String)**
 - ~~✍~~ **<kiểu> x = Convert.To<kiểu>(String)**
- ❑ Ví dụ:
 - ~~✍~~ **int x = int.Parse("123");**
 - ~~✍~~ **bool x = bool.Parse("true");**
 - ~~✍~~ **DateTime x = DateTime.Parse("31-12-2012");**
 - ~~✍~~ **int x = Convert.ToInt32("123");**
 - ~~✍~~ **DateTime x = Convert.ToDateTime("2000-12-31");**

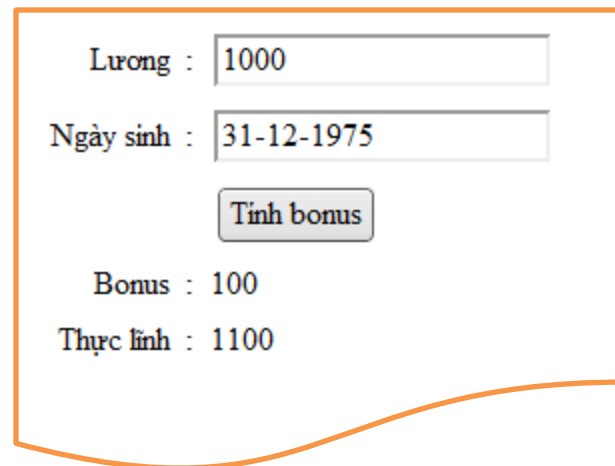
□ Tính bonus theo độ tuổi

✎ Nhập lương

✎ Nhập ngày sinh

✎ Tính bonus

- ✓ 5%lương nếu tuổi dưới 25
- ✓ 10%lương nếu tuổi từ 26 đến 40
- ✓ 15%lương nếu tuổi trên 40



The screenshot shows a web form with the following elements:

- A label "Lương :" followed by a text input field containing the value "1000".
- A label "Ngày sinh :" followed by a text input field containing the value "31-12-1975".
- A button labeled "Tính bonus".
- Below the button, the calculated values are displayed: "Bonus : 100" and "Thực lĩnh : 1100".

- ❑ Là một **dạng thức** được sử dụng để kiểm tra một chuỗi khác có **so khớp** với nó hay không.
- ❑ Ví dụ: Chuỗi có đúng định dạng của
 - ✎ user@abc.com là **Email**
 - ✎ 255579321 là số **CMND**
 - ✎ 54-P6-6661 là số **xe máy Sài Gòn**
 - ✎ 0913745789 là số **điện thoại di động**
 - ✎ 192.168.11.200 là số **IP**
- ❑ Ví dụ: Biểu thức chính qui sau đây dùng để so khớp các chuỗi có định dạng số điện thoại.
 - ✎ String pattern = **"^(0[0-9]{9, 10})\$"**;

Biểu thức chính quy

Ký tự đại diện

[xyz]	<u>đại diện một ký tự x, y hay z</u>
[ad-f]	<u>đại diện một ký tự a, d, e hay f</u>
[^xyz]	<u>đại diện ký tự không thuộc [xyz]</u>
\d	<u>tương đương [0-9]</u>
\w	<u>tương đương [0-9a-zA-Z_]</u>
\D	<u>tương đương [^\d]</u>
\W	<u>tương đương [^\w]</u>
\s	<u>đại diện ký tự trắng (\r\n\t\f)</u>
.	<u>đại diện ký tự bất kỳ</u>
^	<u>chỉ ra mẫu bắt đầu</u>
\$	<u>chỉ ra mẫu kết thúc</u>
\\, \., \\$, \^	<u>đại diện '\', '.', '\$' hay '^'</u>

Số lần xuất hiện

{M,N}	<u>Ít nhất M, nhiều nhất N lần</u>
{N}	<u>Đúng N lần</u>
?	0-1
*	0-N
+	1-N
<u>Không</u>	1

□ Các biểu thức chính qui thường dùng

✎ Số CMND

[0-9]{9}

✎ Số điện thoại di động việt nam

0\d{9,10}

✎ Số xe máy sài gòn

5\d-[A-Z]\d-((\d{4})|(\d{3}\.\d{2}))

✎ Địa chỉ email

\w+@\w+\.\w{2,4}

❑ **Regex.IsMatch**(input, pattern)

- ✍ Kiểm tra chuỗi input có khớp với pattern hay không

❑ **Regex.Split**(input, pattern)

- ✍ Tách chuỗi input thành mảng chuỗi theo các chuỗi phân cách khởi với pattern

❑ **Regex.Replace**(input, pattern, replacement)

- ✍ Thay thế chuỗi khởi với pattern bằng replacement trong chuỗi input

❑ **Regex.Matches**(input, pattern)

- ✍ Lấy tập các thành phần trong chuỗi input khớp với pattern.

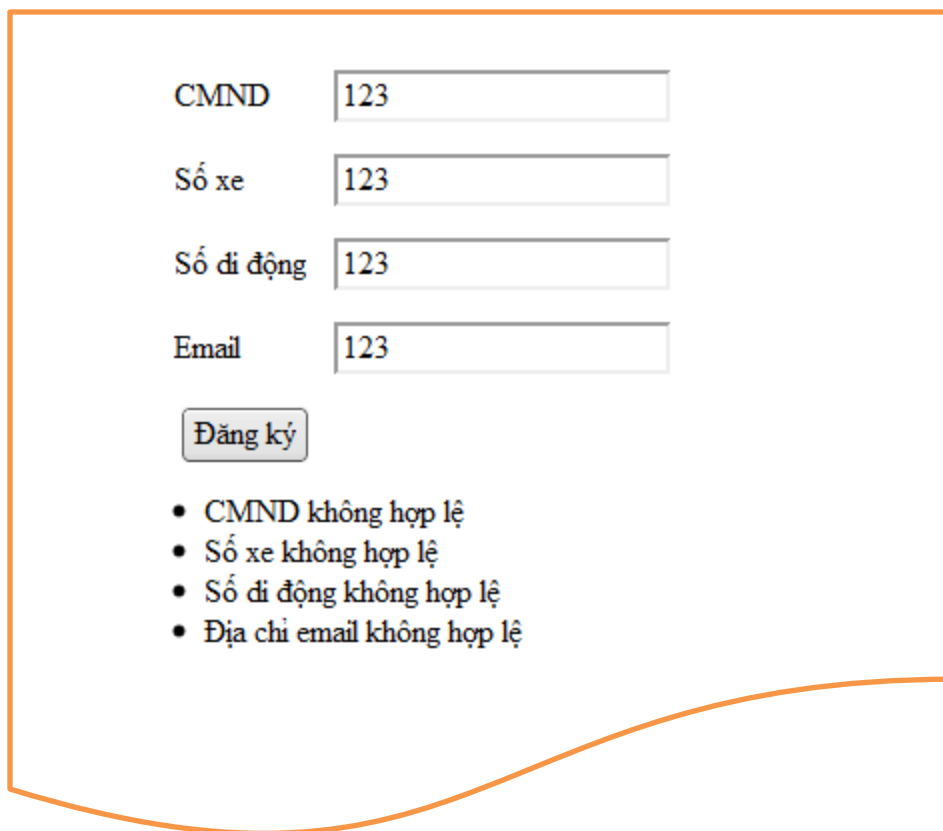
❑ Thêm các thành phần sau đây vào form nhân viên để kiểm tra lỗi đầu vào.

✎ CMND

✎ Điện thoại

✎ Số xe máy

✎ Email



CMND 123

Số xe 123

Số di động 123

Email 123

Đăng ký

- CMND không hợp lệ
- Số xe không hợp lệ
- Số di động không hợp lệ
- Địa chỉ email không hợp lệ

- ☐ Tách lấy email có đuôi gmail.com từ chuỗi hỗn hợp