# LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP) VỚI C#

PHAN TRỌNG TIẾN
BM Công nghệ phần mềm
Khoa Công nghệ thông tin, VNUA
Email: phantien84@gmail.com
Website: http://timoday.edu.vn



## Mục tiêu

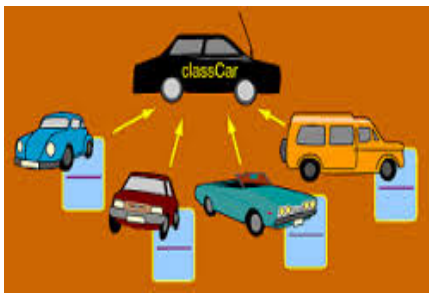
- Hiểu được lập trình hướng đối tượng
- Các đặc trưng của lập trình hướng đối tượng
- Các khai báo và sử dụng lập trình hướng đối tượng trong C#
- Nguồn bài giảng:
  - http://www.c-sharpcorner.com/UploadFile/asmabegam/basic-concept-of-oop-in-C-Sharp/
  - Programming in C# (C0057) – Aptech Worldwide

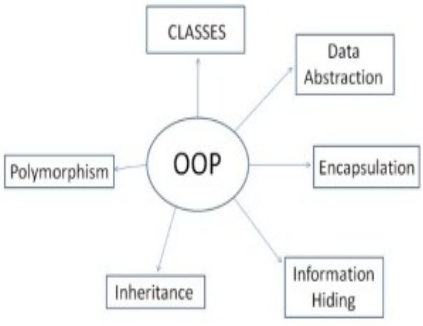
# Nội dung chính

- Lập trình hướng đối tượng là gì?
- Điểm mạnh của lập trình OOP
- Khái niệm Class và Object
- Triển khai OOP bằng C#



# Lập trình hướng đối tượng (OOP)




http://www.tutorialhub.in/2014/11/30/object-oriented-programming-oop-concepts-interview-questions/

# Điểm mạnh của OOP

- Tái sử dụng lại code
- Cung cấp một cấu trúc các module của chương trình một cách rõ ràng
- Che dấu được dữ liệu bên trong
- Bảo trì và chỉnh sửa code dễ dàng
- Cung cấp một framework thuận tiện với các thư viện ở đó có các component có thể dễ dàng tương thích được và thay đổi bởi lập trình viên



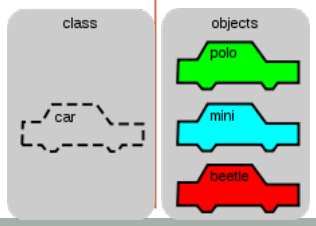
# Class và Object

## Class

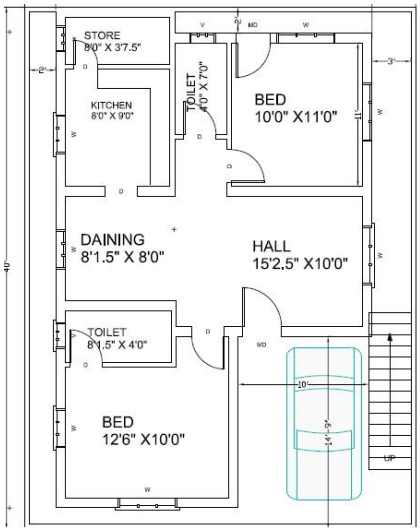
- Định nghĩa trừu tượng các đặc tính của đối tượng
  - Khách hàng
  - Nhân viên
  - Xe hơi
- Bản thiết kế hoặc khuân mẫu

## Object

- Một bản mẫu của class
- "Xe hơi" có một bản mẫu được gọi "Xe hơi của Peters"



# Class




- Class giống như một bản thiết kế, ví dụ như thiết kế của ngôi nhà,
- Sử dụng class chúng ta có thể viết các phương thức riêng và khai báo các biến
- Sử dụng object để truy cập các phương thức và các biến của class
- Classes và Objects là cơ sở của OOP



# Các thuật ngữ bạn cần biết

- Classes
- Objects
- Properties
- Methods
- Events
- #1 Inheritance
- #2 Encapsulation
- #3 Polymorphism
- #4 Abstraction

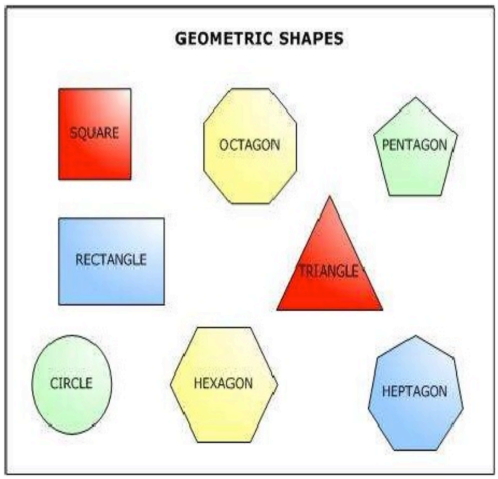
# Các thuật ngữ

- Properties (Thuộc tính)
  - Thay đổi các đặc tính của đối tượng
  - Ex: "Màu sắc" của chiếc xe hơi
- Methods (Phương thức)
  - Các hành động của một đối tượng
  - Ex: "Xe hơi" có phương thức "Tăng tốc"
- Events (Sự kiện)
  - Để thực hiện các tương tác với đối tượng
  - Ex: "Xe hơi" có sự kiện "Mở cửa"



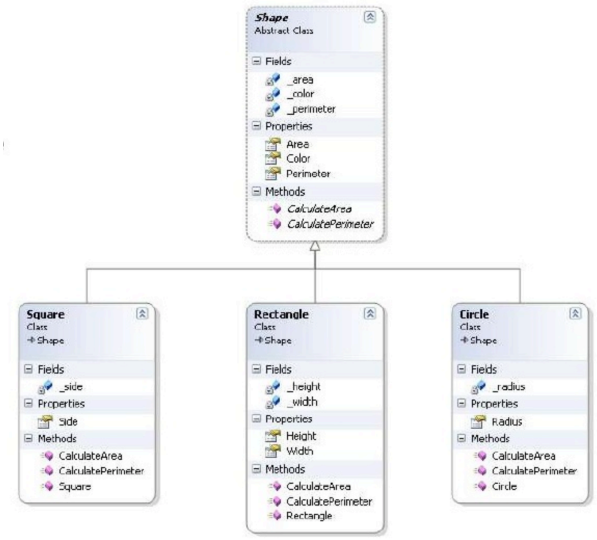
# #1 Inheritance

- "Square" là một "Shape"



# #1 Inheritance

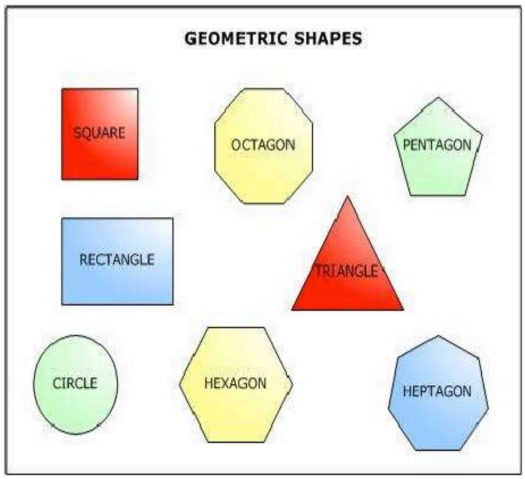
- "Shape" định nghĩa thuộc tính chung "color"
- "Square" thừa kế thuộc tính "color"



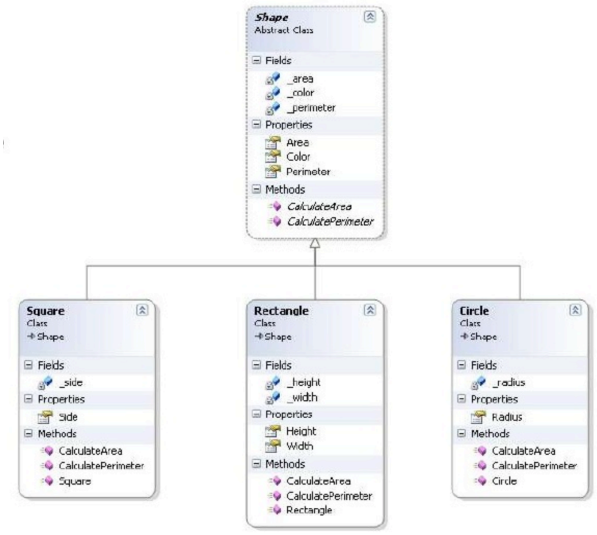



# #2 Encapsulation

- Che dấu thông tin
- Ex:
  - "Shape" che dấu được dữ liệu bên trong đối tượng
    - Toạ độ điểm thứ nhất
    - Toạ độ điểm thứ hai



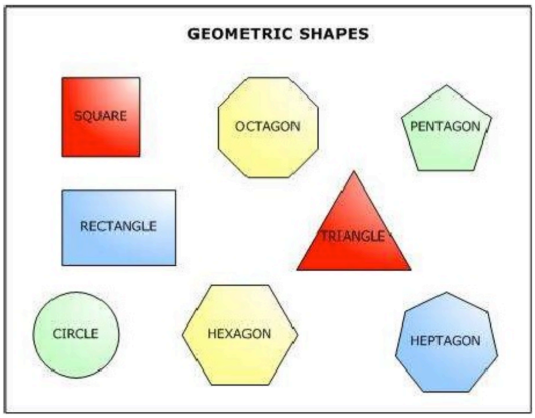
## "Square" có một trường bên trong "_slide"





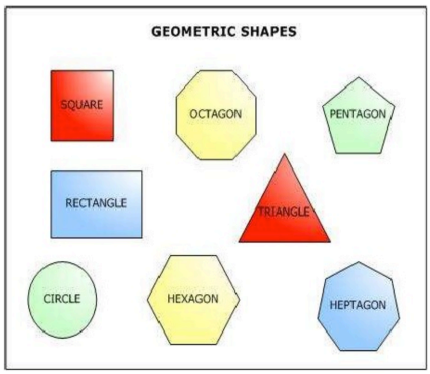

## #3 Polymorphism

- Xuất hiện như các đối tượng khác
- Được sử dụng như các đối tượng khác



# #3 Polymorphism

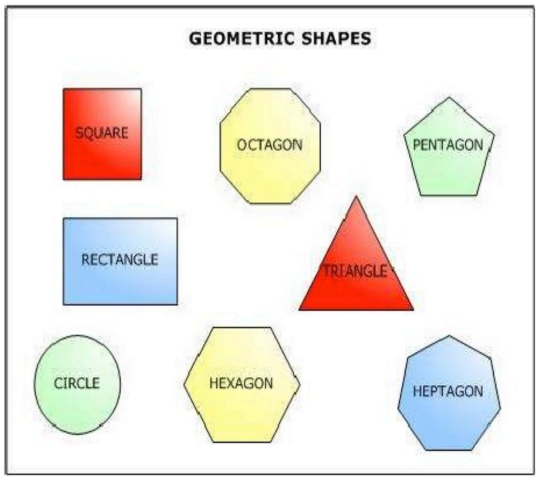
```
var shapes = new List<IShapes>() {
    new Square("Red"),
    new Rectangle("Blue"),
    new Triangle("Red")
};

foreach (var shape in shapes) {
    Console.WriteLine(shape.Color +
    ": " + shape.CalcSize());
}
```





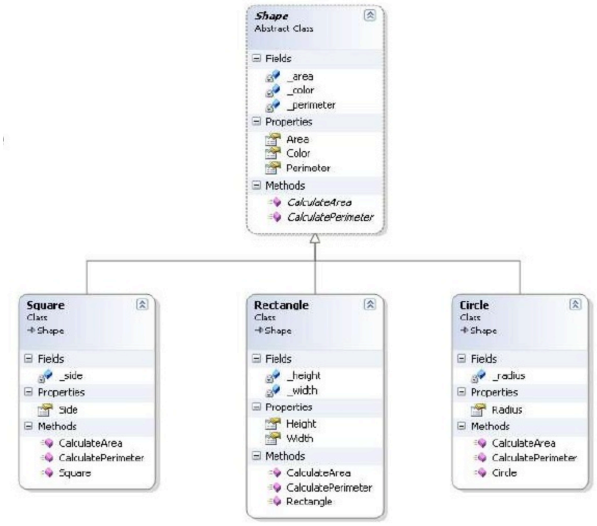

# #4 Abstraction

- Người dùng không cần hiểu chi tiết về công nghệ bên trong (ví dụ các bộ phận của xe hơi)
- Chỉ "hiển thị" các tính năng cần thiết của đối tượng
- Miêu tả tính năng chỉ liên quan về đối tượng (Object) gọi là tính trừu tượng



# Không có instance từ lớp "Shape"

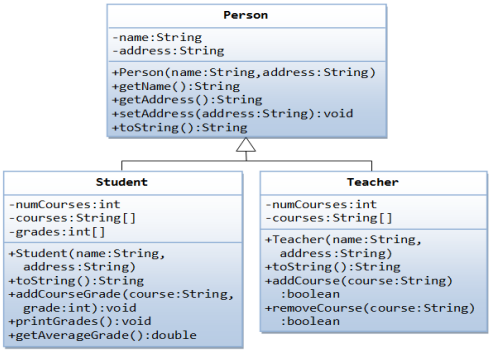
- Không sử dụng toán tử ***new*** để khởi tạo từ đối tượng Shape






# Bài tập

- Nhận diện các thành phần trong đối tượng của mô hình bên phải: ví dụ thuộc tính, phương thức, thừa kế, tính đa hình …



# Triển khai OOP trong C#

- Class
- Object
- Variable
- Method hoặc Functions
- Access Modifiers
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism
- Abstract Class/Method
- Virtual Class/Method
- Sealed Class/Method
- Static Class/Method
- Interface



# Class và Object

- Class nên bắt đầu với từ khoá "Class" và tiếp theo là tên của Class

```
class ShanuHouseClass
{
}
```

- Để truy xuất vào các phương thức và các biến của class, chúng ta sử dụng object

```
ShanuHouseClass objHouseOwner = new ShanuHouseClass();
```

# Variable

- Các biến được sử dụng để lưu trữ các giá trị
- Các biến có thể là ***local*** hoặc ***global***
- Cú pháp:

  *Access-Modifier Data-Type Variable-Name*
- Mặc định access modifier là *private*, cũng có thể là *public*

```
int noOfTV = 0;
public String yourTVName;
private Boolean doYouHaveTV = true;
class ShanuHouseClass
{
   int noOfTV = 2;
   public String yourTVName = "SAMSUNG";

   static void Main(string[] args)
   {
      ShanuHouseClass objHouseOwner = new ShanuHouseClass();

      Console.WriteLine("You Have total " + objHouseOwner.noOfTV + " no of TV :");
      Console.WriteLine("Your TV Name is :" + objHouseOwner.yourTVName);

   }
}
```



# Method hoặc Function

- Là một nhóm các câu lệnh code

```
class ShanuHouseClass
{
   int noOfTV = 2;
   public String yourTVName = "SAMSUNG";

   public void MyFirstMethod()
   {
      Console.WriteLine("You Have total " + noOfTV + "no of TV :");
      Console.WriteLine("Your TV Name is :" + yourTVName);
      Console.ReadLine();
   }

   static void Main(string[] args)
   {
      ShanuHouseClass objHouseOwner = new ShanuHouseClass();
      objHouseOwner.MyFirstMethod();
   }
}
```

# Method hoặc Function

- Cấu trúc của Fucntion:

*Access-Modifiers Return-Type Method-Name (Parameter-List)*

- Access-Modifiers
- Return-Type: kiểu giá trị trả về như string, int, v.v hoặc kiểu "void" nếu không trả về giá trị
- Method-Name: tên của method
- Parameter-List: danh sách các tham số hoặc đối số truyền vào function



# Method hoặc Function

- Method với kiểu void

```
public void Veranda()
{
  Console.WriteLine("Welcome to Veranda");
  Console.WriteLine("How Many Chairs Do you have in your Veranda");
  NoofChair = Convert.ToInt32(Console.ReadLine());
  Console.WriteLine("I have total " + NoofChair + " Chairs in my Veranda");
}
```

- Method với kiểu trả về

```
public string TVNAME()
{
  Console.WriteLine("Enter Your TV Brand NAME");
  YOURTVName = Console.ReadLine();
  return YOURTVName;
}
```

- Method với Parameter-List

```
public void BedRoom(String MemberName,String Color)
{
   Console.WriteLine(MemberName + " Like " + Color + "Color");
}
```

# Kiểu *ref* và *out* trong danh parameter-list

- *ref* và *out* thường được đặt các đối số dạng tham chiếu trong Method thay vì dạng Value trong C#
- *out* không cần phải khởi gán giá trị trước khi gọi hàm
- *ref* nên khởi gán giá trị như các biến thông thường,

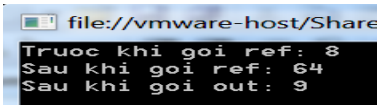
```
//Methods
public static void MethodRef(ref int a){
}
public static void MethodOut(out int a){
}
//Calling
MethodRef(ref b);
MethodOut(out b);
```



# Ví dụ ref và out

```
class Program
   {
      static void Main(string[] args)
      {
         int num = 8;
         CalcSquare(num);
         Console.WriteLine("Truoc
khi goi ref: {0}", num);
         RefSquare(ref num);
         Console.WriteLine("Sau khi
goi ref: {0}", num);
         OutSquare(out num);
         Console.WriteLine("Sau khi
goi out: {0}", num);
         Console.ReadLine();
      }
}
```

```
static void CalcSquare(int inum)
      {
         inum = inum * inum;
      }
static void RefSquare(ref int inum)
      {
         inum = inum * inum;
      }
static void OutSquare(out int inum)
      {
         inum = 3;
         inum = inum*inum;
      }
```



# Access Modifiers

- Được dùng để giới hạn khả năng truy cập của các variable, method và class
  - private: có mức độ bảo vệ cao nhất
  - public: không giới hạn truy cập
  - internal: giới hạn truy cập trong cùng một project
  - protected: truy cập được trong class chính và các class dẫn xuất (thừa kế)
  - protected internal: giới hạn truy cập trong project hoặc lớp dẫn xuất



# Access Modifier

```
public class SampleProtectedInternalClass
{
   protected internal String myProtectedInternal =
"Iam Protected Internal Variable";

   public void ProtectedInternalMethod()
   {

Console.WriteLine("Protected Internal Variable Example :" +
myProtectedInternal);
   }
}
public class DerivedClass : SampleProtectedInternalClass
{
   public void DerivedProtectedInternal()
   {

Console.WriteLine("Derived Protected Internal Variable Exam
ple :" + myProtectedInternal);
   }
}
```

```
class ShanuHouseClass
{
   int noOfTV = 2;
   public String yourTVName = "SAMSUNG";

   public void MyFirstMethod()
   {
      Console.WriteLine("You Have total " + noOfTV +
"no of TV :");
      Console.WriteLine("Your TV Name is :" + yourTVName);

   }

   static void Main(string[] args)
   {
      ShanuHouseClass objHouseOwner = new ShanuHouseClass();
      objHouseOwner.MyFirstMethod();
      SampleProtectedInternalClass intObj = new
SampleProtectedInternalClass();

      intObj.ProtectedInternalMethod();

      DerivedClass proIntObj = new DerivedClass();
      proIntObj.DerivedprotectedInternal();
      Console.ReadLine();

   }
}
```

# Constructor và Properties

- Constructor
  - Là một Method đặc biệt của class
  - Sử dụng để khởi tạo đối tượng của class khi class được khởi tạo
  - Tên của Constructor trùng với tên của class
  - Trong constructor có thể có các tham số
- Properties
  - Là các thành viên của class cho phép truy cập dữ liệu bên trong của class



# Ví dụ Constructor và Properties

```
class Student
{
   private string strNumber;
   private string strName;
   private string strSex;
   public Student()
   { //Constructor
      strNumber = "";
      strName = "";
      strSex = "";
   }
   //Constructor with parameters
   public Student(string id, string name, string sex)
   {
      strNumber = id;
      strName = name;
      strSex = sex;
   }
   public void SetNumber()
   {
      Console.WriteLine("Enter student's
number:");
      strNumber = Console.ReadLine();
   }
   public void GetNumber()
   {
      Console.WriteLine("Student's number is:
{0}", strNumber);
   }
   public string StuNumber
   { //stunumber property
      set { strNumber = value; }
      get { return strNumber; }
   }
}
```

# Encapsulation

- Ẩn các thành viên hay các biến từ bên ngoài class

```
public class HouseSecurityClass
{
   public int noofSecurity;
   public String SecurityName = String.Empty;
}

public class HouseClass
{
   private int noofLockerinHosue = 2;
   public string OwnerName = String.Empty;
}
```

- Class "*HouseSecurityClass*" có 2 biến public, vì vậy "HouseClass" có thể truy cập cả hai.
- Clas "*HouseClass*" có 1 biến pubic và 1 biến private, biến private không thể truy cập ngoài class



# Abstraction

- Hiển thị và chia sẻ những thông tin thông dụng cho người dùng

```
public class HouseServerntClass
{

   private int SaftyLoackerKeyNo = 10001;
   public String roomCleanInstructions = "Clean All rooms";

   private void SaftyNos()
   {
      Console.WriteLine("My SaftyLoackerKeyNo is" + SaftyLoackerKeyNo);
   }

   public void RoomCleanInstruction()
   {
      Console.WriteLine(roomCleanInstructions);
   }
}
```

# Inheritance

- Được sử dụng để tái sử dụng lại code
- Có hai loại:
  - Single level Inheritance: một lớp cơ sở và một lớp dẫn xuất
  - Multi level Inheritance: nhiều hơn một lớp dẫn xuất
- .Net không hỗ trợ Multiple Inheritance, Interface là giải pháp cho Multiple Inheritance



# Inheritance

```
public class BaseClass
{
   private void PrivateMethod()
   {
      Console.WriteLine("private Method");
   }

   public void PublicMethod()
   {
      Console.WriteLine("This Method Shared");
   }
}
```

```
public class DerivedClass1 : BaseClass
{
   public void DerivedClass1()
   {
      Console.WriteLine("Derived Class 1");
   }
}
public class DerivedClass2 : DerivedClass1
{
   static void Main(string[] args)
   {
      DerivedClass2 obj = new DerivedClass2();
      obj.PublicMethod();
      obj.DerivedClass1();
      //obj.privateMethod(); //Sẽ bị lỗi
   }
}
```

# Polymorphism

- Có nhiều hơn một khuân dạng, ví dụ cùng tên một phương thức nhưng có các tham biến khác nhau
- Có 2 loại polymorphism:
  - Method Overloading: cùng tên method nhưng khác đối số
  - Method Overriding: cùng method và các đối số và kiểu dữ liệu nhưng method được sử dụng lại ở lớp dẫn xuất. Có thể được sử dụng trong **Abstract Method**, **Virtual Method** và **Sealed Method**
- Có 2 kiểu thực thi:
  - Compile Time Polymorphism
  - Run time Polymorphism



# Ví dụ Method Overloading

```
class HouseOwnerClass
{
   //Function with parameter
   public void BedRoom(String nameandColor)
   {
      Console.WriteLine(nameandColor);
   }
   // Same Function Name with Different Paramenter
   public void BedRoom(String MemberName, String Color)
   {
      Console.WriteLine(MemberName + " Like " + Color + "Color");
   }

   static void Main(string[] args)
   {
      HouseOwnerClass objHouseOwner = new HouseOwnerClass();

      objHouseOwner.BedRoom("My Name is Shanu I like Lavender color");
      objHouseOwner.BedRoom("SHANU", "Lavender");
      Console.ReadLine();
   }
}
```

# Abstract Class/Method

- Abstract Class: là lớp có từ khoá "abstract"

```
abstract class GuestVist
{
}
```

- Lớp trừu tượng là  một lớp cha (super) của tất các các class.
- Không thể tạo **object** từ một lớp trừu tượng
- Lớp trừu tượng có thể có cả hai Abstract Method và Method thông thường.
- Abstract method không có thân chương trình (khối lệnh)



# Ví dụ Abstract Class/Method

```
abstract class GuestVist
{
   public void GuestWelcomeMessage()
   {

Console.WriteLine("Welcome to our Abst
ractHome");
   }
   public void GuestName()
   {

Console.WriteLine("Guest name is: Abstr
act");
   }
   public abstract void PurposeOfVisit();

}
```

```
// derived class to inherit the abstract clas
s
public class Houseclass : GuestVist
{
   static void Main(string[] args)
   {
      Houseclass objHouse = new
Houseclass();
      objHouse.GuestWelcomeMessage();
   }

   public override void PurposeOfVisit()
   {

Console.WriteLine("Abstract just came fo
r a Meetup and spend some time ");
   }
}
```

# Virtual Class/Method

- Virtual method là rất hữu ích
- So sánh giữa Abstract method và Virtual method:
  - Giống nhau: đều sử dụng từ khoá override
  - Khác nhau:
    - Abstract method: chỉ khai báo trong Abstract class
    - Abstract method không có thân chương trình, còn Virtual method có thân chương trình hoàn chỉnh.



# Virtual Class/Method

```
abstract class GuestVist
{
   public void Guestwelcomemessage()
   {

Console.WriteLine("Welcome to our AbstractHo
me");
   }
   public void GuestName()
   {

Console.WriteLine("Guest name is: Abstract");
   }
   public abstract void
purposeofVisit();  // Abstract Method
   public virtual void
NoofGuestwillvisit()  // Virtual Method
   {

Console.WriteLine("Total 5 Guest will Visit your
Home");
   }
}
```

```
 class AbstractHouseClass : GuestVist
{
   public override void
purposeofVisit()  // Abstract method Override
   {

Console.WriteLine("Abstract just for a Meetup and spend som
e time ");
   }
   public override void
NoofGuestwillvisit() // Virtual method override
   {
      Console.WriteLine("Total 20 Guest Visited our Home");
   }
   static void Main(string[] args)
   {
      AbstractHouseClass objHouse = new
AbstractHouseClass();
      objHouse.Guestwelcomemessage();
      objHouse.purposeofVisit();
      objHouse.NoofGuestwillvisit();
      Console.ReadLine();
   }
}
```

# Selead Class/Method

- Selead class: không thể được thừa kế bởi các class khác
  - Sử dụng từ khoá **selead**
- Selead Method: các phương thức không thể viết chồng (overriden) trong lớp dẫn xuất.
  - Virtual method có thể viết chồng trong lớp dẫn xuất nhưng Virtual Sealed Method thì không



# Ví dụ Sealed Class/Method

```
 //Base Class with Sealed Method
public class OwnerOfficialroomwithrestriction
{
   public virtual void message()
   {

Console.WriteLine("Every one belongs to this hou
se can access my items in my room except my sea
led Item");


   }

   public virtual sealed void
myAccountsLoocker()
   {

Console.WriteLine("This Loocker can not be inhe
rited by other classes");
   }
}
```

```
 //
Deraived method which inherits Selaed Method cl
ass
class HouseSealedClass : OwnerOfficialroomwith
restriction
{
   public override void message()
   {
      Console.WriteLine("overrided in the derived
class");
   }

   public override void myAccountsLoocker()
   {
      Console.WriteLine("The sealed method Over
rides");
   }
}
```

# Static Class/Method

- Cả hai Static và Sealed không thể thừa kế
- Sự khác nhau giữu Static Class và Selead Class:
  - Có thể tạo được một object (instance) từ Selead Class, còn Static Class thì không thể.
  - Trong Static Class, chỉ có Static members là được phép, không thể viết các method không là static
- Vấn đề gì sảy ra khi:
  - Thừa kế một lớp từ một Static class?
  - Khai báo một non-Static method trong một Static class?
  - Khi chúng ta tạo object từ một Static class?
  - Có thể gọi metho Static trong class mà không cần tạo Object được không?
  - Có thể tạo một Static Method trong một non-Static Class?



# Ví dụ Static Class/Method

```
public class OwnerofficialRoom
{
   public static void AllMyPersonalItems()
   {

Console.WriteLine("No need to create object for
me just use my class name to access me :)");
   }

   public void non_staticMethod()
   {

Console.WriteLine("You need to create an Object
to Access Me :(");
   }
}
```

```
class StaticmethodClass
{
   static void Main(string[] args)
   {
      // for statci method no need to create object j
ust call directly using the classname.
      OwnerofficialRoom.AllMyPersonalItems();

      // for non-
static method need to create object to access the m
ethod.
      OwnerofficialRoom obj = new
OwnerofficialRoom();
      obj.non_staticMethod();
      Console.ReadLine();

   }
}
```

# Interface

- Giống như một Abstract class nhưng chỉ khai báo các method
- Các method của lớp Interface phải được thi hành ở trong class mà nó kế thừa
- Sự khác nhau giữu Abstract class và Interface:
  - Abstract class có cả hai Abstract method và Non-Abstract method
  - Tất cả các method trong Interface mặc định là abstract (không có Non-Abstract)
  - Tất cả các Method khai báo trong Interface nên được viết chồng trong lớp dẫn xuất
- Có thể khai báo một non-abstract method có phần thân chương trình hay không?



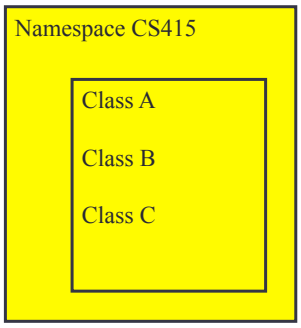
# Ví dụ Interface

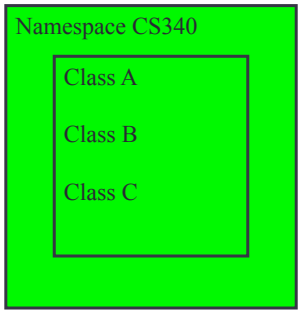
```
interface GuestInterface
{
   void GuestWelcomeMessage();
   void NoofGuestes();
}
interface FriendsandRelationsInterface
{
   void friendwelcomememessage();
   void FriendName();
}
```

```
class HouseOwnerClass : GuestInterface, FriendsandRelationsInterface
{
   public void GuestWelcomeMessage()
   {
      Console.WriteLine("All guests are well come to our home");
   }

   public void NoofGuestes()
   {
      Console.WriteLine("Total 15 Guestes has visited");
   }

   public void friendwelcomememessage()
   {
      Console.WriteLine("Welcome to our Home");
   }
   public void FriendName()
   {
      Console.WriteLine("Friend  name is: Afraz");
   }
   static void Main(string[] args)
   {
      HouseOwnerClass obj = new HouseOwnerClass();
      obj.GuestWelcomeMessage();
      obj.NoofGuestes();
      obj.friendwelcomememessage();
      obj.FriendName();
      Console.ReadLine();
   }
```

# Namespaces

- Cho phép phân tách các lớp mà có cùng tên
- Một namespace có thể có nhiều class
- Truy xuất đầy đủ các đối tượng




```
CS415.A...
CS340.A...
```