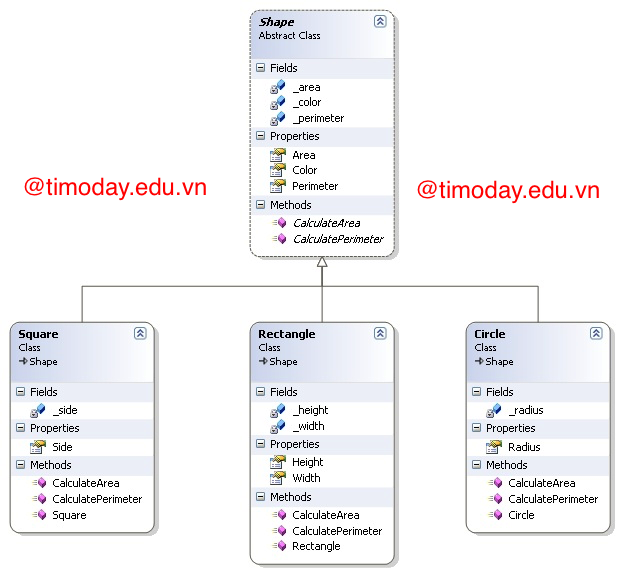
**Lập trình hướng đối tượng trong C#**

**Lập trình hướng đối tượng** (**Object-Oriented Programming – OOP**) là một kỹ thuật lập trình hỗ trợ hướng đối tượng, trở thành một chuẩn lập trình của các ngôn ngữ lập trình tiên tiến hiện nay. Điểm mạnh của lập trình hướng đối tượng là 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, giúp lập trình viên đơn giản hóa độ phức tạp của bài toán.

Ngôn ngữ C# là ngôn ngữ hướng đối tượng tiên tiến nhất hiện nay, nó kế thừa những điểm mạnh của ngôn ngữ lập trình hướng đối tượng C++ và Java, đồng thời nó cũng loại bỏ đi những sự phức tạp của lập trình hướng đối tượng của hai ngôn ngữ này, ví dụ như nó loại bỏ tính đa thừa kế trong C++, hoặc nó cho phép viết chồng các toán tử mà Java không có …

Trong bài này chúng ta sẽ triển khai lập trình hướng đối tượng trong C# để xây dựng các lớp theo mô hình UML của các lớp *Shape*, *Square*, *Rectangle* và lớp *Circle* được mô tả như hình dưới. Hãy nhận diện các Field, Properties và Methods trong mỗi class .

[](http://timoday.edu.vn/wp-content/uploads/2017/02/OOPshapes.png)

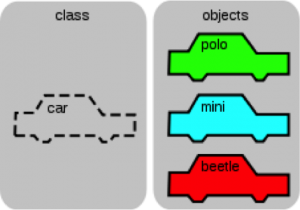
Hình 1: Bài tập lập trình hướng đối tượng

**Nội dung**

* [Class và Object](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc1)
* [Variable hoặc Field](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc2)
* [Method hoặc Function](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc3)
* [Access Modifier](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc4)
* [Constructor, Destructor và Properties](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc5)
* [Encapsulation](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc6)
* [Abstraction](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc7)
* [Inheritance](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc8)
* [Polymorphism](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc9)
* [Abstract Class/Method](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc10)
* [Virtual Class/Method](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc11)
* [Sealed Class/Method](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc12)
* [Static Class/Method](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc13)
* [Interface](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc14)
* [Xem hướng dẫn](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc16)
* [Bài tập làm thêm](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc17)
* [Kết luận](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#muc15)
* [Tham khảo thêm](https://timoday.edu.vn/phan-3-lap-trinh-huong-doi-tuong-trong-c/#thamkhao)

**Class và Object**

* **Class** trong OOP định nghĩa trừu tượng các đặc tính của đối tượng, chúng ta hiểu nôm na giống như bản thiết kế hay khuôn mẫu của đối tượng nào đó, ví dụ như bản thiết kế của ô tô, bản vẽ của toà nhà, các control như TextBox, Button, Label.
* **Object** là một sản phẩm được tạo ra từ bản thiết kế của **Class**, ví dụ bản thiết kế ô tô đó được mang đi sản xuất, bản vẽ toà nhà được dùng để xây dựng, TextBox, Button … được kéo vào Form thì lúc đó sản phẩm tạo từ các bản thiết kế đó gọi là **Object**.

[](http://timoday.edu.vn/wp-content/uploads/2017/02/Class_Object.png)

Sự khác nhau Class và Object trong OOP

**Class** nên bắt đầu với từ khoá “*Class*” và tiếp theo là tên của Class

class Shape

{

}

Để tạo **Object** từ **Class**, dùng từ khoá **new**, lúc đó chúng ta có thể truy cập vào các phương thức, thuộc tính … của Class.

Shape objShape = new Shape();

**Variable hoặc Field**

* **Variable** hoặc **Field** được sử dụng để lưu trữ dữ liệu.
* Các biến có thể là cục bộ (local) hoặc toàn cục (global)
* Cú pháp: ***Access-Modifier******Data-Type******Variable-Name***
* Mặc định Access-Modifier là private. Trong class Shape, \_area, \_color, \_perimeter là các biến hay các field của class này.

class Shape

{

private float \_area;

private System.Drawing.Color \_color;

private float \_perimeter;

}

**Method hoặc Function**

là một khối các câu lệnh để thực thi một công việc nào đó, được khai báo ở trong **class** hoặc **struct**. Cấu trúc của Method:  
**Access-Modifiers Return-Type Method-Name (Parameter-List)**

* **Access-Modifier**: xác định giới hạn truy cập của method, xem phần sau.
* **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 method

class Rectangle

{

public void CalculateArea()

{

this.Area = \_height \* \_width;

}

}

**Access Modifier**

Được dùng để giới hạn khả năng truy cập của các **variable**, **method** và **class**

* **private**: chỉ được truy cập trong cùng **class** hoặc **struct**
* **public**: có thể truy cập ở bất kỳ đâu
* **internal**: giới hạn truy cập trong cùng một project
* **protected**: truy cập được trong cùng class hoặc struct và các class dẫn xuất (thừa kế)
* **protected internal**: giới hạn truy cập trong project hoặc lớp dẫn xuất

**Constructor, Destructor và Properties**

**Constructor hay gọi là hàm khởi tạo**

* Là một Method đặc biệt của class
* Sử dụng để khởi tạo các giá trị trong đối tượng của class khi đối tượng được khởi tạo
* Tên của Constructor trùng với tên của class
* Trong constructor có thể có các tham số

class Rectangle

{

private float \_height;

private float \_width;

//Constructor không có tham số, khởi tạo các biến mang giá trị mặc định là 0

public Rectangle()

{

\_height = 0;

\_width = 0;

}

//Contructor có tham số, khởi tạo biến mang giá trị truyền vào khi khởi tạo Constructor

public Rectangle(float height, float width)

{

\_height = height;

\_width = width;

}

}

**Destructor hay gọi là hàm huỷ**

* chỉ được sử dụng trong class, không cho struct
* một class chỉ có một Detructor
* Detructor không thể được thừa kế hoặc viết chồng
* Detructor không được gọi trực tiếp, mà .Net sẽ gọi tự động
* Detructor không có Access Modifier hoặc các Parameter

class Rectangle

{

~Rectangle() // Destructor

{

// Các câu lệnh giải phóng biến, đối tượng..

}

}

**Properties**

* Là các thành viên của class cho phép truy cập dữ liệu bên trong của class
* Có thể tạo Properties có thể chỉ đọc, chỉ ghi hoặc vừa đọc vừa ghi bằng cách viết phần code get và set

class Rectangle

{

private float \_height;

private float \_width;

//Tạo thuộc tính Height để truyền giá trị và lấy giá trị từ biến \_height

public float Height

{ //Lấy giá trị

get

{

return \_height;

}

//Thiết lập giá trị

set

{

\_height = value;

}

}

}

**Abstraction**

Dùng để **định nghĩa** những thuộc tính và phương thức **chung** cho những lớp kế thừa.  
Một lớp trừu tượng là một lớp cha cho phép thừa kế nhưng không thể tạo đối tượng từ nó.  
Lớp trừu tượng chứa một hoặc nhiều phương thức nhưng không thực thi và nó phải được viết chồng ở lớp kế thừa.  
Hình 1 thiết kế lớp Shape là một lớp trừu tượng. Thực tế, bạn không thể nào tính diện tích và chu vi cho một đối tượng Shape, bạn phải biết Shape đó là loại nào ví dụ như Circle, Square hoặc Rectangle thì bạn mới áp dụng được công thức tính toán cho nó. Lớp cha Shape định nghĩa các phương thức **CalculateArea()** và **CalculatePerimeter()** và các lớp con kế thừa lớp Shape bắt buộc phải viết chồng các phương thức này và áp dụng các công thức tính toán phù hợp với mỗi loại hình đó.

namespace Phan3

{

public abstract class Shape

{

//Khai báo các biến cục bộ

private float \_area;

private System.Drawing.Color \_color;

private float \_perimeter;

//Tạo các Properties

//Thuộc tính diện tích

public float Area

{

get

{

return \_area;

}

set

{

\_area = value;

}

}

//Thuộc tính màu sắc

public System.Drawing.Color Color

{

get

{

return \_color;

}

set

{

\_color = value;

}

}

//Thuộc tính chu vi

public float Perimeter

{

get

{

return \_perimeter;

}

set

{

\_perimeter = value;

}

}

//Khai báo các method trừu tượng

public abstract void CalculateArea();

public abstract void CalculatePerimeter();

}

}

**Inheritance**

Được sử dụng để tái sử dụng lại code, lớp con có thể kế thừa các Field, Methods, Properties của lớp cha  
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  
Để tạo thừa kế lớp Rectangle từ lớp Shape trong C#, cần phải sử dụng câu lệnh:

class Rectangle : Shape

{

}

**Polymorphism**

Cho phép các đối tượng có thể được thể hiện dưới nhiều khuôn dạng khác nhau. Thậm chí các lớp được thừa kế từ cùng lớp cha, có cùng phương thức, nhưng mỗi lớp con sẽ có cách thực hiện riêng. Tính đa hình đảm bảo mỗi lớp sẽ có cùng chức năng nhưng lại có cách thực thi khác nhau.  
Như hình 1, lớp Shape định nghĩa hai phương thức trừu tượng CalculateArea() và CalculatePerimeter(), điều này cho phép các lớp kế thừa như Circle, Square và Rectangle viết chồng cách thức thực thi method của lớp cha.

namespace Phan3

{

class Rectangle : Shape

{

private float \_height;

private float \_width;

public Rectangle(float height, float width)

{

\_height = height;

\_width = width;

}

public float Height

{

get

{

return \_height;

}

set

{

\_height = value;

}

}

public float Width

{

get

{

return \_width;

}

set

{

\_width = value;

}

}

public override void CalculateArea()

{

this.Area = \_height \* \_width;

}

public override void CalculatePerimeter()

{

this.Perimeter = (\_height \* 2) + (\_width \* 2);

}

}

}

**Abstract Class/Method**

* 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)

**Virtual Class/Method**

Từ khoá Virtual cũng cho phép thực thi tính đa hình. Thuộc tính và Method virtual có phần thực thi trong lớp cơ sở và có thể được viết chồng ở lớp dẫn xuất (lớp kế thừa).  
So sánh giữa Abstract method và Virtual method:  
Giống nhau: đều sử dụng từ khoá override ở lớp dẫn xuất  
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.

**Sealed Class/Method**

Selead class: được dùng để ngăn thừa kế hay không thể được thừa kế bởi các class khác  
Sử dụng từ khoá sealead

* 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

**Static Class/Method**

Cả hai loại Static và Sealed đều không thể thừa kế  
Sự khác nhau giữa 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

Một số câu hỏi ta nên tổng kết lại:

* Có thể thừa kế một lớp từ một Static class?
* Có thể khai báo một non-Static method trong một Static class?
* Có thể tạo object từ một Static class?
* Có thể gọi method Static trong class mà không cần tạo Object được không?

**Interface**

Interface định nghĩa các thuộc tính, phương thức, indexers và sự kiện, là các thành viên của Interface đó.  
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?

//Tạo lớp giao diện IGuest

interface IGuest

{

void GuestWelcomeMessage();

void NoofGuestes();

}

//Tạo lớp giao diện IFriendsAndRelation

interface IFriendsAndRelations

{

void FriendWelcomeMessage();

void FriendName();

}

//Lớp HouseOwnerClass thừa kế từ cả hai Interface trên

class HouseOwnerClass : IGuest, IFriendsAndRelation

{

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 FriendWelcomeMessage ()

{

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.FriendWelcomeMessage();

obj.FriendName();

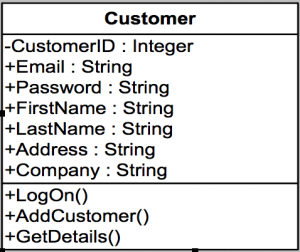
Console.ReadLine();

}

}

**Xem hướng dẫn**

**Bài tập: Thiết kế lớp Customer**

[](http://timoday.edu.vn/wp-content/uploads/2017/02/ClassCustomer.png)

Thiết kế lớp Customer

**Yêu cầu:**

* Tạo Class Customer có các thuộc tính và phương thức như trên thiết kế. CustomerID là biến cục bộ tự động tăng.
* Phương thức LogOn(): cho phép kiếm tra một User (username, password) có hợp lệ không.
* Phương thức AddCustomer() cho phép thêm một khách hàng với các thông tin ở trên.
* Phương thức GetDetails() lấy toàn bộ thông tin khách hàng.

Tạo form giao diện để nhập dữ liệu tương ứng truyền vào cho Class Customer và thực hiện các chức năng trên. Lưu dữ liệu vào file text và kiểm tra dữ liệu khách hàng trong file text này.  
**Chú ý:** Có thể thêm các tham số cần thiết cho các phương thức như LogOn, AddCustomer, GetDetails.

**Kết luận**

Đây là bài tổng hợp một cách ngắn gọn nhất phần lập trình hướng đối tượng trong .NET được triển khai bằng ngôn ngữ C#, chúng tôi đã giới thiệu cách lập trình hướng đối tượng qua một bài tập đơn giản, qua đó chúng ta biết được và nắm được cách tạo **class**, tạo **object**, triển khai các đặc trưng của lập trình hướng đối tượng như tính thừa kế, tính trừu tượng, tính đa hình, tính đóng gói, cách thực thi Interface, đa thừa kế và cách viết chồng …Trong bài học này chúng ta thấy tính thừa kế cho phép các lập trình viên có thể tổng quát hoá và đặc tả được mối quan hệ giữa các đối tượng. Các khái niệm về tính trừu tượng và tính đa hình giúp định nghĩa các mô hình đối tượng một cách tốt hơn mà ở đó có sự phân cấp và sử dụng lại giữa các đối tượng.