# LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (OOP)

## I. Lập trình hướng đối tượng là gì?

- Lập trình hướng đối tượng (OOP - Object Oriented Programming) là một phương pháp lập trình sử dụng các đối tượng (object) để xây dựng hệ thống phần mềm hoặc ứng dụng web.

## II. Cách khái niệm trong hướng đối tượng.

### 1. Lớp (Class)

- Một class trong lập trình hướng đối tượng là đại diện tập hợp cái đối tướng có cúng đặc điểm, hành vi, phương thức hoạt động.

  VD:

```
class Bill
{
    //tính chất
    public double Phone { get; set; }
    public string IdCustomer { get; set; }
    public string Address { get; set; }
    public string Name { get; set; }

    // phương thức (Method)
    public void InputBill()
    {
        Console.Write("Ma khach hang: ");
        IdCustomer = Console.ReadLine();
        Console.Write("Ten khach hang: ");
        Name = Console.ReadLine();
        Console.Write("Dia chi: ");
        Address = Console.ReadLine();
        Console.Write("So dien thoai: ");
        Phone = double.Parse(Console.ReadLine());
    }
}
```

### 2. Phương thức (method)

- Phương thức là các hành vi hay phương thức hoạt động của đối tượng. Để sử dụng phương thức ta cần:

  + Định nghĩa phương thức

  + Gọi phương thức

  VD:


Cấu trúc của 1 method: <Từ khóa truy cập> <Kiểu dứ liệu trả về> <Tên phương thức>(Danh sách biến)

{

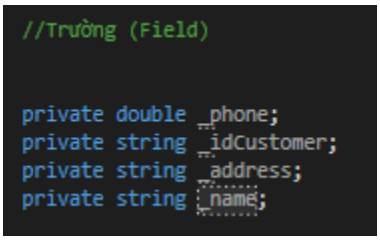
Phần thân phương thức

}

### 3. Trường (Field)

- Trường là các biến của class hay còn gọi là biến toàn cục. Trường được sủ dụng để lưu trữ dữ liệu về các đặc điểm của đối tượng. Trường nên sử dụng chỉ thị truy cập là private chỉ có thể truy cập trong class.

VD:




- Quy ước đặt tên trường hoặc biến toàn cục trong C# bắt đầu bằng dấu gạch dưới, kí tự tiếp theo là chữ thường

### 4. Thuộc tính (Property)

- Một thuộc tính là một thành viên của class, nó cung cấp một cơ chế linh hoạt để truy cập các trường (Field) của class.
- Trong thuộc tính có 2 phương thức đặc biệt được gọi là accessor là **get** và **set** trong đó **get** sẽ trả giá trị của trường, còn **set** thì sẽ gán giá trị cho trường

VD:

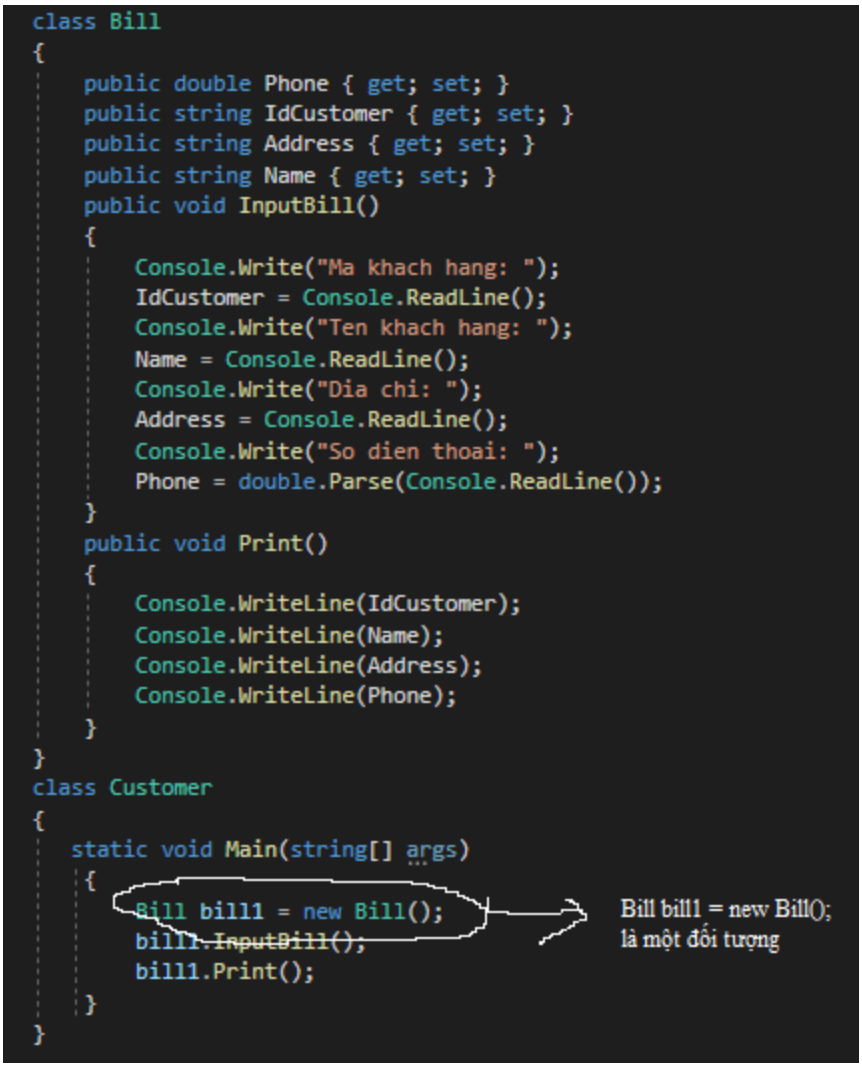
```
private double _phone;
private string _idCustomer;
private string _address;
private string _name;

public double Phone
{
    get
    {
        return _phone;
    }
    set
    {
        _phone = value;
    }
}
public string IdCustomer
{
    get
    {
        return _idCustomer;
    }
    set
    {
        IdCustomer = value;
    }
}
public string Address
{
    get
    {
        return _address;
    }
    set
    {
        _address = value;
    }
}
public string Name
{
    get
    {
        return _name;
    }
    set
    {
        _name = value;
    }
}
```

- Ngoài ra C# còn cung cấp auto-implemented property – một cách định nghĩa thuộc tính nhau sau:

```
public double Phone { get; set; }
public string IdCustomer { get; set; }
public string Address { get; set; }
public string Name { get; set; }
```

### 5. Đối tượng(Object)

- Đối tượng là một thức thể trong thế giới thực có đặc điểm và hành vi. Khi bạn khởi tạo thể hiện cho một class tức là đó là thể hiện của đối tượng
  VD:




# III. Các tính chất của lập trình hướng đối tượng:

## 1. Tính đóng gói (Encapsulation):

- Tính đóng gói là khả năng che giấu thông tin của đối tượng với môi trường bên ngoài. Việc cho phép bên ngoài tác động lên các dữ liệu nội tại của đối tượng tùy thuộc vào người viết mã

  + public: không giới hạn phạm vi truy cập

  + private: (mặc định) chỉ truy cập được từ các thành viên của lớp chứa nó

  + protected: chỉ truy cập trong nội bộ lớp hay các lớp kế thừa

  + internal: chỉ truy cập được trong cùng assembly (dll, exe)

  + protected internal: truy cập được khi cùng assembly hoặc lớp kế thừa

  + private protected: truy cập từ lớp chứa nó, lớp kế thừa nhưng phải cùng assembly

  VD:

```
protected double Phone { get; set; }
private string IdCustomer { get; set; }
internal string Address { get; set; }
protected internal string Name { get; set; }
private protected int day { get; set; }
```

**2. Tính kế thừa (Inheritance):**

- Tính kế thùa là khả năng tạo một lớp có sẵn gọi là lớp cha hay lớp cơ sở. Và các lớp mới là lớp con hay lớp dẫn xuất và các lớp mới tạo sẽ thừa kế các thành viên đã được định nghĩa ở lớp cha

VD:

```
class Bill
{
    public string IdCustomer { get; set; }
    public string ID { get; set; }
    public string Address { get; set; }
    public string Name { get; set; }
    public double Phone { get; set; }

}
class Intput : Bill
{
    public void InputBill()
    {
        Console.Write("Ma khach hang: ");
        IdCustomer = Console.ReadLine();
        Console.Write("Ten khach hang: ");
        Name = Console.ReadLine();
        Console.Write("Dia chi: ");
        Address = Console.ReadLine();
        Console.Write("So dien thoai: ");
        Phone = double.Parse(Console.ReadLine());
    }
}
class Output
{
    static void Main(string[] args)
    {
        Intput intputs = new Intput();
        intputs.InputBill();
    }
}
```

➔ Lớp Bill là lớp cha (lớp cơ sở), lớp Intput là lớp kết thừa của lớp Bill

**3. Tính đa hình (Polymorphism):**

- Tính đa hình là khả năng đối tượng có thể thực hiện một tác vụ theo nhiều các khác nhau có 2 loại là đa hình tĩnh và đa hình động:

- Đa hình tĩnh (Overloading): hay còn gọi là đa hình tại thời điểm biên dịch (compile time). Đa hình tĩnh có 2 kỹ thuật thực hiện:

+ Nạp chồng phương thức: là các phương thức có cùng tên nhưng khác nhau về số lượng hoặc kiểu dữ liệu truyền vào tham số.
VD:

```
class Bill
{
    public double Phone { get; set; }
    public void InputBill()
    {
        Console.Write("So dien thoai: ");
        Phone = double.Parse(Console.ReadLine());
    }
    public void ReInput(double _phone)
    {
        Phone = _phone;
    }
    public void ReInput(int phone)
    {
        Phone = phone;
    }
}
class Customer
{
   static void Main(string[] args)
    {
        Bill bill1 = new Bill();
        bill1.InputBill();
    }
}
```

+ Nạp chồng toán tử: là định nghĩa lại hoặc nạp chồng hầu hết các toán tử trong C#. nhờ đó, ta có thể sử dụng các toán tử với kiểu do người dùng định nghĩa.
VD:

```
 class Box
{
    private double chieu_dai;
    private double chieu_rong;
    private double chieu_cao;

    public double tinhTheTich()
    {
        return chieu_dai * chieu_rong * chieu_cao;
    }

    public void setChieuDai(double len)
    {
        chieu_dai = len;
    }

    public void setChieuRong(double bre)
    {
        chieu_rong = bre;
    }


    public void setChieuCao(double hei)
    {
        chieu_cao = hei;
    }
    public static Box operator +(Box b, Box c)
    {
        Box box = new Box();
        box.chieu_dai = b.chieu_dai + c.chieu_dai;
        box.chieu_rong = b.chieu_rong + c.chieu_rong;
        box.chieu_cao = b.chieu_cao + c.chieu_cao;
        return box;
    }
    public void In()
    {
        Console.WriteLine(tinhTheTich());
    }
}
class Main2
{
     static void Main(string []args)
    {
        Box x = new Box();
        x.setChieuCao(10);
        x.setChieuRong(20);
        x.setChieuDai(30);
        x.In();
        Console.ReadKey();
    }
}
```

- Đa hình động (overriding): là đa hình thực thi (runtime). C# cho bạn tạo các lớp trừu tượng (abstract class) được sử dụng để cung cấp triển khai lớp một phần của giao diện
  - Trong lớp trừu tượng, còn có thể khai báo phương thức trừu tượng với từ khóa abstract, phương thức này không có thân (chỉ có tên - tham số), nó yêu cầu lớp kế thừa bắt buộc phải nạp chồng (overrid)

```
abstract class Box
{
    public string ID;
    public int day, mounth, year;
    public abstract void InputBill();
}
class Bill : Box
{
    public override void InputBill()
    {
        Console.Write("Ma hoa don: ");
        ID = Console.ReadLine();
        Console.Write("Ngay lap: ");
        day = int.Parse(Console.ReadLine());
        Console.Write("Thang lap: ");
        mounth = int.Parse(Console.ReadLine());
        Console.Write("Nam lap: ");
        year = int.Parse(Console.ReadLine());
    }
    public string BilltoString()
    {
        return $"Hoa Don: {ID}\t{day}/{mounth}/{year}\n"
    }
    public void In()
    {
        Console.WriteLine(BilltoString());
    }
}

class Main2
{
     static void Main(string []args)
    {
        Bill x = new Bill();
        x.InputBill();
        x.In();
        Console.ReadKey();
    }
}
```

**4. Tính trừu tượng (Abstraction):**

- Tính trừu tượng là khả năng ẩn chi tiết triển khai, chỉ cung cấp thông tin tính năng tới người dùng. Tính trừu tượng được thể hiện qua abstract class và interface. Và ví dụ ở phần đa hình động cũng có thể làm ví dụ cho tính trừu tượng sử dụng abstract class và abstract method.

VD: Interface

```
interface IBill
{
    void InputBill();
}

class Bill : IBill
{
    public string ID;
    public int day, mounth, year;
    public void InputBill()
    {
        Console.Write("Ma hoa don: ");
        ID = Console.ReadLine();
        Console.Write("Ngay lap: ");
        day = int.Parse(Console.ReadLine());
        Console.Write("Thang lap: ");
        mounth = int.Parse(Console.ReadLine());
        Console.Write("Nam lap: ");
        year = int.Parse(Console.ReadLine());
    }

    public string BilltoString() => $"Hoa Don: {ID}\t{da
    public void Ins()
    {
        Console.WriteLine(BilltoString());
    }
}

class Program
{
    static void Main(string[] args)
    {
        Bill I = new Bill();
        I.InputBill();
        I.Ins();
        Console.ReadKey();
    }
}
```