Tính đóng gói:

 Tính chất này không cho phép người sử dụng các đối tượng thay đổi trạng thái nội tại của một đối tượng. Chỉ có các phương thức nội tại của đối tượng cho phép thay đổi trạng thái của nó. Việc cho phép môi trường bên ngoài tác động lên các dữ liệu nội tại của một đối tượng theo cách nào là hoàn toàn tùy thuộc vào người viết mã. Đây là tính chất đảm bảo sự toàn vẹn của đối tượng.

Tính đa hình:

1 phương thức lớp cha dùng ở các lớp con khác nhau cho ra các bộ mặt khac nhau.

VD;

abstract class Nguoi

{

public string Ten { get; set; }

public abstract string TTHoSo();

public void HienThiHoSo()

{

string tt = TTHoSo();

Console.WriteLine("Ho So Gom Co:");

Console.WriteLine(tt);

}

}

class Hocsinh : Nguoi

{

public override string TTHoSo()

{

return "Ho So Hoc Sinh";

}

}

class Sinhvien:Nguoi

{

public override string TTHoSo()

{

return "Ho So Sinh Vien..... Dai hon nhieu";

}

}

class Program

{

static void Main(string[] args)

{

Nguoi anhA = new Hocsinh();

anhA.HienThiHoSo();

anhA = new Sinhvien();

anhA.HienThiHoSo();

Console.ReadKey();

}

}

Phương thức TTHoSo ở đối tượng HocSinh cho ra 1 kiểu; ở đối tượng SinhVien cho ra 1 kiểu khác.

Tính kế thừa:

Lớp A kế thừa lớp B thì lớp A sẽ có mọi thuộc tính phương thức của lớp B nếu thuộc tính hay phương thức dố không phải private.

Tính trừu tượng;

 Đây là khả năng của chương trình bỏ qua hay không chú ý đến một số khía cạnh của thông tin mà nó đang trực tiếp làm việc lên, nghĩa là nó có khả năng tập trung vào những cốt lõi cần thiết. Mỗi đối tượng phục vụ như là một "động tử" có thể hoàn tất các công việc một cách nội bộ, báo cáo, thay đổi trạng thái của nó và liên lạc với các đối tượng khác mà không cần cho biết làm cách nào đối tượng tiến hành được các thao tác. Tính chất này thường được gọi là *sự trừu tượng của dữ liệu*.  
Tính trừu tượng còn thể hiện qua việc một đối tượng ban đầu có thể có một số đặc điểm chung cho nhiều đối tượng khác như là sự mở rộng của nó nhưng bản thân đối tượng ban đầu này có thể không có các biện pháp thi hành. Tính trừu tượng này thường được xác định trong khái niệm gọi là *lớp trừu tượng* hay *lớp cơ sở trừu tượng*.

Virtual

Khi 1 lớp muốn kế thừa A 1 phương thức nhưng muốn sửa 1 it cho phù hợp với mình mà ko muốn ảnh hưởng dến lớp khác thì dùng Virtual.

class Nguoi

{

public string Ten { get; set; }

public virtual void GioiThieu()

{

Console.WriteLine("Xin Chao, toi ten la {0}", Ten);

}

}

class Hocsinh : Nguoi { }

class Sieunhan : Nguoi

{

public override void GioiThieu()

{

base.GioiThieu();

Console.WriteLine("Toi La Sieu Nhan (Nhac Nen)");

}

}

class Program

{

static void Main(string[] args)

{

Hocsinh hs = new Hocsinh() { Ten = "Nguyen Van A" };

Sieunhan sn = new Sieunhan() { Ten = "Clark Kent" };

hs.GioiThieu();

sn.GioiThieu();

Console.ReadKey();

}

}

Abstract

Dùng khi lớp cha muốn làm gì đó cho lớp con nhưng không biết lớp con nào cân đến. 1 lớp có 1 pthức abstract trở lên sẽ là 1 lớp abstact . 1 lớp abstact ko thể khai báo mới vd : nguoi a = new nguoi();//ko co câu lênh này.

abstract class Nguoi

{

public string Ten { get; set; }

public abstract string TTHoSo();

public void HienThiHoSo()

{

string tt = TTHoSo();

Console.WriteLine("Ho So Gom Co:");

Console.WriteLine(tt);

}

}

class Hocsinh : Nguoi

{

public override string TTHoSo()

{

return "Ho So Hoc Sinh";

}

}

class Sinhvien:Nguoi

{

public override string TTHoSo()

{

return "Ho So Sinh Vien..... Dai hon nhieu";

}

}

class Program

{

static void Main(string[] args)

{

Nguoi anhA = new Hocsinh();

anhA.HienThiHoSo();

anhA = new Sinhvien();

anhA.HienThiHoSo();

Console.ReadKey();

}

}

Interface

Dùng khi 1 phương thức có nhiều lớp con kế thừa mà đều muốn override nó

Thì Interface dùng như 1 p thức ngoài dùng để điều khiển p thức ấy.

namespace OOP\_5

{

class BietBay : IBay { public void Bay() { Console.WriteLine("Toi Biet Bay"); } }

class KoBietBay : IBay { public void Bay() { Console.WriteLine("Toi Ko Biet Bay"); } }

interface IBay{ void Bay(); }

class Vit

{

IBay ib;

public void SetIBay(IBay \_ib) { ib = \_ib; }

public void Bay() { ib.Bay(); }

}

class VitTroi : Vit { }

class Program

{

static void Main(string[] args)

{

VitTroi vt = new VitTroi();

vt.SetIBay(new BietBay());

vt.Bay();

vt.SetIBay(new KoBietBay());

vt.Bay();

Console.ReadKey();

}

}

}

Tính đơn nhất:

namespace OOP\_5

{

class Sinhvien

{

public string Ten { get; set; }

public int Tuoi { get; set; }

public void Hienthi()

{

Console.WriteLine("{0} co tuoi la {1}", Ten, Tuoi);

}

}

class SingletonPattern

{

private static SingletonPattern \_obj = null;

private SingletonPattern() { }

public static SingletonPattern GetLatest()

{

if (\_obj == null)

\_obj = new SingletonPattern();

return \_obj;

}

//////

public Sinhvien SV = new Sinhvien();

}

class Program

{

static void Ham1()

{

Sinhvien sv = SingletonPattern.GetLatest().SV;

sv.Ten = "Nguyen Van A";

sv.Tuoi = 10;

}

static void Ham2()

{

Sinhvien sv = SingletonPattern.GetLatest().SV;

sv.Ten = "Tran Van B";

sv.Tuoi = 20;

}

static void Main(string[] args)

{

Ham1();

//Ham2();

Sinhvien sv = SingletonPattern.GetLatest().SV;

sv.Hienthi();

Console.ReadKey();

}

}

}

SingletonPattern

Giữ chi ta luôn tương tác, làm việc với dối tượng cuối cùng mà ta đã làm việc trước đó. VD ta lam việc với cả Ham1 và Ham2 nhưng Ham2 ta tương tác sau cùng nên câu lệnh này

Sinhvien sv = SingletonPattern.GetLatest().SV;

sv.Hienthi();

sẽ in ra kết quả của ham2.