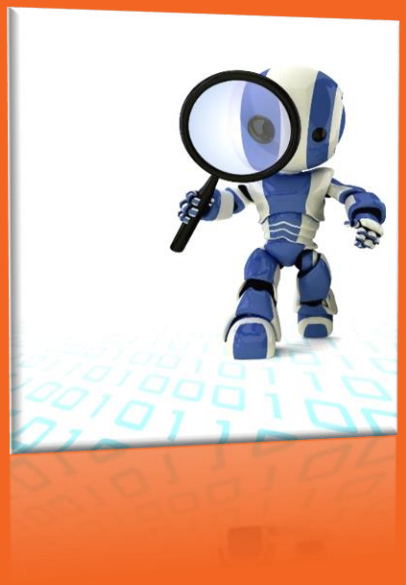
FPT Education

FPT POLYTECHNIC



LẬP TRÌNH C# 1

BÀI 5: THỪA KẾ VÀ TRỪU TƯỢNG

www.poly.edu.vn

Mục tiêu

- Kế thừa
- Trừu tượng

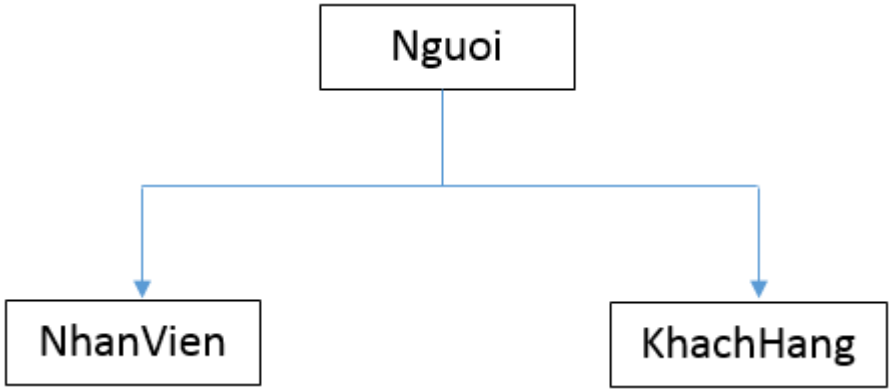
NỘI DUNG

- Phần I: Kế thừa
  - Các loại kế thừa
  - Từ khóa base, new
  - Constructor và tính kế thừa
- Phần II: Trừu tượng
  - Overiding
  - Lớp, phương thức trừu tượng

FPT POLY

SỰ PHÂN CẤP THỪA KẾ

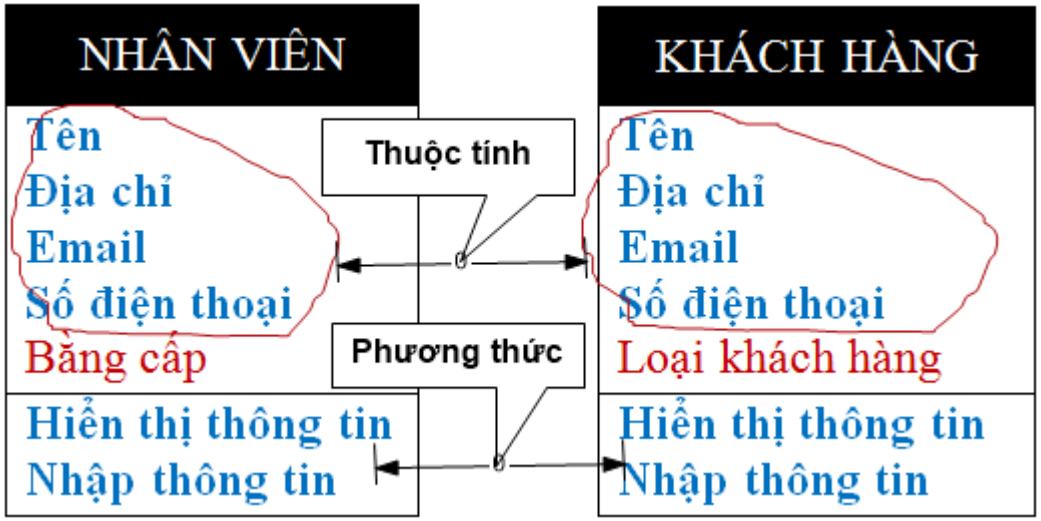
- Các lớp trong C# tồn tại trong một hệ thống thứ bậc phân cấp, gọi là cây thừa kế
- Lớp bậc trên gọi là lớp cha (base_class) trong khi các lớp bậc dưới gọi là lớp con (derived_class)
- Trong C# một lớp chỉ có một lớp cha duy nhất (đơn thừa kế)

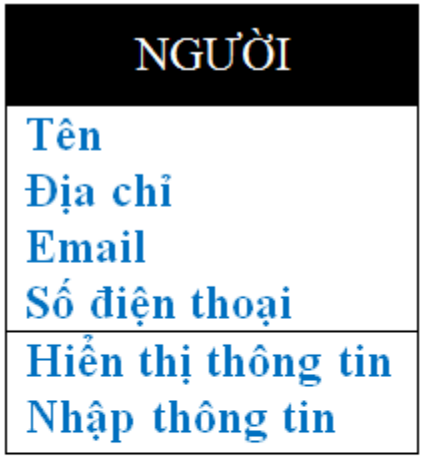


TÍNH THỪA KẾ

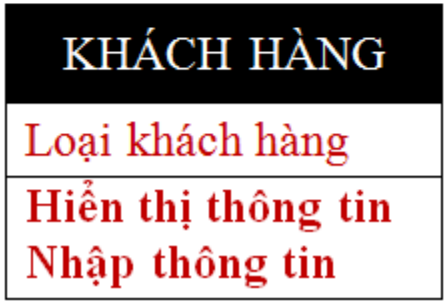
- Cho phép tạo ra một lớp mới kế thừa thuộc tính và phương thức của một lớp khác
- Lớp mới có thể xây dựng thêm các thuộc tính hoặc phương thức riêng
- Giúp nhất quán các thuộc tính và phương thức
- Tái sử dụng code: phần code chung định nghĩa một lần tại lớp cha, lớp con chỉ việc sử dụng mà không cần định nghĩa lại
- Thuận tiện bảo trì và phát triển

TÍNH THỪA KẾ


NHÂN VIÊN
Tên
Địa chỉ
Email
Số điện thoại
Bằng cấp
Hiển thị thông tin
Nhập thông tin
Thuộc tính
Phương thức
KHÁCH HÀNG
Tên
Địa chỉ
Email
Số điện thoại
Loại khách hàng
Hiển thị thông tin
Nhập thông tin


NGƯỜI
Tên
Địa chỉ
Email
Số điện thoại
Hiển thị thông tin
Nhập thông tin


NHÂN VIÊN
Bằng cấp
Hiển thị thông tin
Nhập thông tin


KHÁCH HÀNG
Loại khách hàng
Hiển thị thông tin
Nhập thông tin

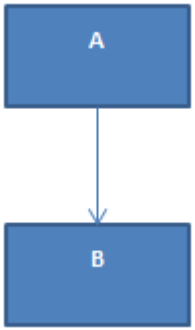
TÍNH THỪA KẾ

- Cú pháp:

```
[<quyen_truy_cap>] class <Ten_lop_cha>
{
    ...
}

[<quyen_truy_cap>] class <Ten_lop_con> : <Ten_lop_cha>
{
    ...
}
```

- Trong c# có nhiều loại thừa kế được hỗ trợ như thừa kế đơn (*Single Inheritance*), thừa kế đa cấp độ (*Single Inheritance*), thừa kế phân cấp (*Hierarchical Inheritance*)
- Lớp con chỉ sử dụng được các thuộc tính và phương thức có phạm vi truy cập là public, internal, protected của lớp cha

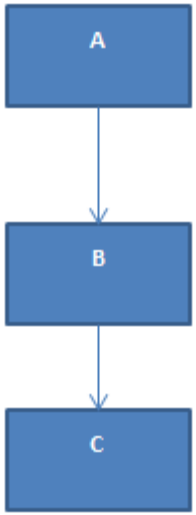
- Single Inheritance: một lớp chỉ kế thừa từ một lớp cha




```
public class NhanVien
{
    public long luong = 2000000000;
}

public class KeToan: NhanVien
{
    public long tienThuong = 5000000;
}

class Program
{
    public static void Main(string[] args)
    {
        KeToan kt = new KeToan();

        Console.WriteLine("Luong: " + kt.luong);
        Console.WriteLine("Tien Thuong: " + kt.tienThuong);
        Console.ReadKey();
    }
}
```

TÍNH THỪA KẾ

- Multilevel Inheritance: một lớp kế thừa từ một lớp cha, mà lớp cha đó lại kế thừa từ một lớp khác

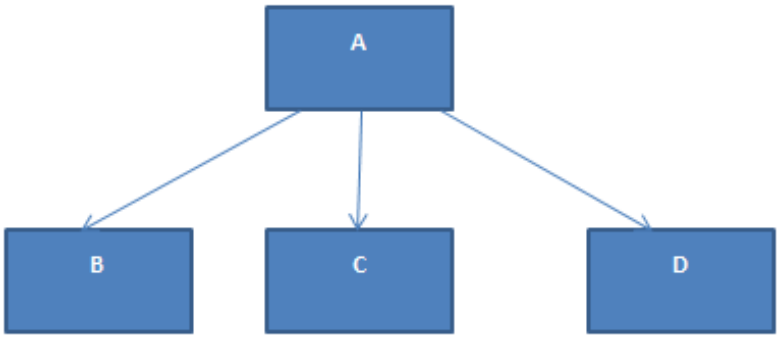



```
public class DongVat
{
    public string mauSac;
    public string tiengKeu;
}

public class Cho : DongVat
{
    public void TiengKeu()
    {
        tiengKeu = "gau gau";
        Console.WriteLine("Con cho keu: " + tiengKeu);
    }
}
```

```
public class ChoCon : Cho
{
    public void TiengKeuChoCon()
    {
        TiengKeu();
        tiengKeu = "e e";
        Console.WriteLine("Con cho con keu: " + tiengKeu);
    }
}

class Program
{
    public static void Main(string[] args)
    {
        ChoCon cc = new ChoCon();
        cc.TiengKeuChoCon();
        Console.ReadKey();
    }
}
```

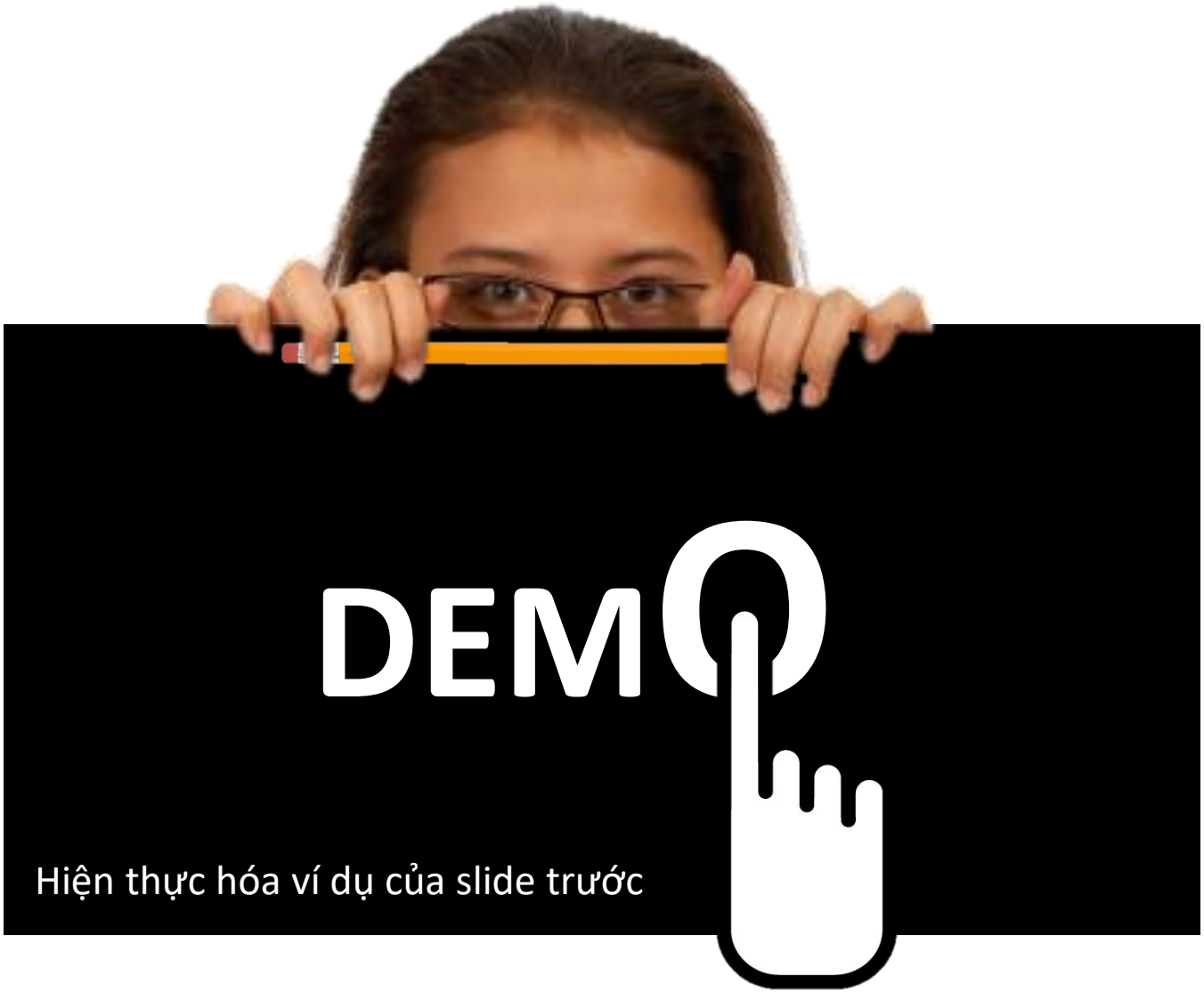
TÍNH THỪA KẾ

- Hierarchical Inheritance: một lớp có nhiều lớp con hay nói cách khác là có nhiều lớp cùng thừa kế từ một lớp cha




```
public class NhanVien
{
    public long luong = 20000000000;
}
public class KeToan : NhanVien
{
    public long hoaHong = 200000;
}
public class LapTrinhVien : NhanVien
{
    public long hoaHong = 500000;
}
public class QuanTriVien : NhanVien
{
    public long hoaHong = 800000;
}
```

```
class Program
{
    public static void Main(string[] args)
    {
        KeToan kt = new KeToan();
        Console.WriteLine("Luong ke toan: " + kt.luong);
        Console.WriteLine("Hoa hong ke toan: " + kt.hoaHong);

        LapTrinhVien ltv = new LapTrinhVien();
        Console.WriteLine("\nLuong lap trinh vien: " + ltv.luong);
        Console.WriteLine("Hoa hong lap trinh vien: " + ltv.hoaHong);

        QuanTriVien qtv = new QuanTriVien();
        Console.WriteLine("\nLuong quan tri Vien: " + qtv.luong);
        Console.WriteLine("Hoa hong quan tri vien: " + qtv.hoaHong);
        Console.ReadKey();

    }
}
```


DEMO
Hiện thực hóa ví dụ của slide trước

TÍNH THỪA KẾ

- Đối tượng lớp cha có thể tham chiếu đến đối tượng lớp con nhưng ngược lại thì không

```
1  // Tạo đối tượng con cún
2  ConCun cun1 = new ConCun();     // OK
3
4  DongVat cun2 = new ConCun();    // OK
5
6  //ConCun cun3 = new DongVat();  // ERROR: Cannot implicitly convert type 'DongVat' to 'C
```

- Dùng từ khóa base truy cập đến các thành phần bên trong lớp cha từ lớp con
- Không dùng từ khóa base cho các thành phần static

TÍNH THỪA KẾ

- Từ khóa new trong kế thừa để xác định phạm vi của phương thức(nó không phải là toán tử khởi tạo đối tượng)
- Dùng từ khóa new cho phương thức lớp con trùng tên phương thức lớp cha

```
class DongVat
{
    public void Chay()
    {
        Console.WriteLine( "Dong vat chay." );
    }

    // Phương thức tĩnh
    public static void Ngu()
    {
        Console.WriteLine( "Dong vat ngủ." );
    }
}
```

```
class ConCun : DongVat      // Kế thừa
{
    // Phương thức CÙNG TÊN với lớp cha
    public new void Chay()
    {
        // Gọi hàm từ lớp cha
        base.Chay();

        Console.WriteLine( "Con cun chay." );
    }
}
```

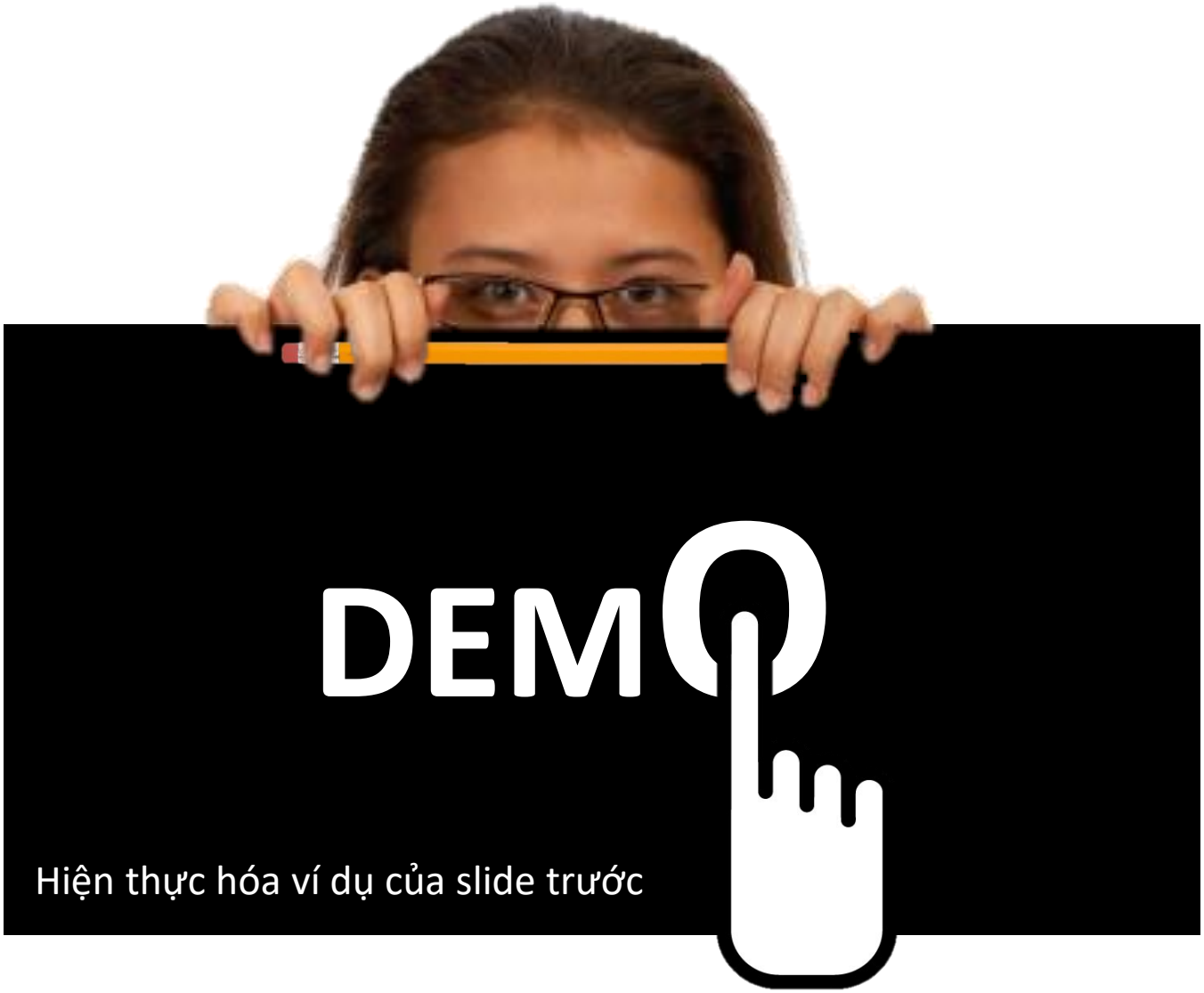
```
class Program
{
    static void Main(string[] args)
    {
        // Tạo đối tượng Con Cún
        ConCun cun = new ConCun();

        // Gọi phương thức Chạy
        cun.Chay();

        // Gọi phương thức tĩnh bằng tên lớp
        DongVat.Ngu();

        Console.ReadKey();
    }
}
```

CONSTRUCTOR VÀ TÍNH KẾ THỪA

- Lớp cha (lớp cơ sở) có định nghĩa phương thức khởi tạo (constructor) có tham số thì tại các lớp dẫn xuất (lớp con hay lớp được kế thừa) **phải gọi phương thức khởi tạo của lớp cha**

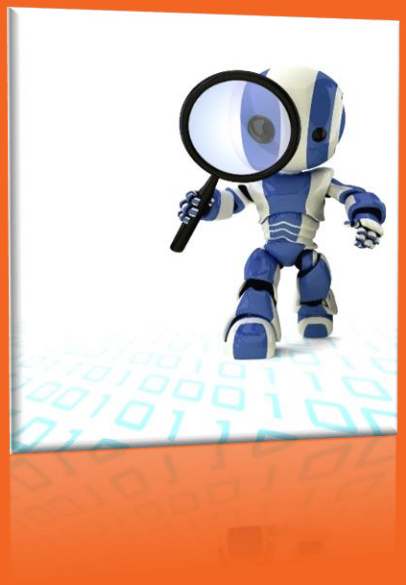
```
class Nguoi
{
 // Khai báo thuộc tính
 public string maso;
 public string hoten;
 public string gioitinh;
 // Phương thức khởi tạo có 4 tham số
 public Nguoi(string maso, string hoten, string gioitinh)
 {
   this.maso = maso;
   this.hoten = hoten;
   this.gioitinh = gioitinh;
 }
```

```
class NhanVien : Nguoi
{
    private string bangcap;
    0 references
    public NhanVien(string maso, string hoten,
        string gioitinh, string bangcap)
    : base(maso, hoten, gioitinh)
    {
        this.bangcap = bangcap;
    }
}
```


DEMO
Hiện thực hóa ví dụ của slide trước

FPT Education

FPT POLYTECHNIC



LẬP TRÌNH C# 1

BÀI 5: THỪA KẾ VÀ TRỪU TƯỢNG(P2)

www.poly.edu.vn

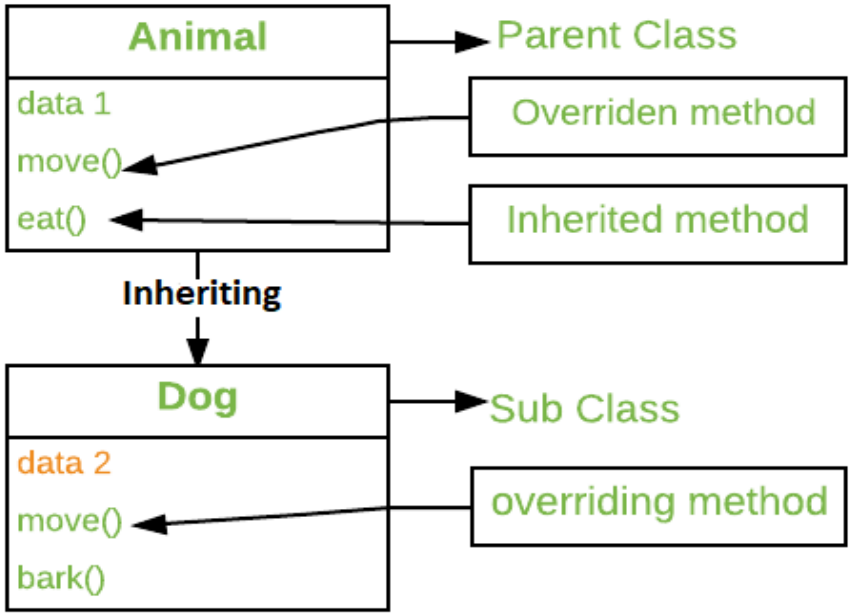
OVERLOAD

- Tạo ra nhiều phương thức(*method*) có **cùng tên**, trong **cùng một phạm vi**(scope), nhưng **khác nhau về đối số đầu vào**(arguments) hoặc **kiểu trả về**
- Trình biên dịch dựa vào số lượng tham số, data type của tham số, hoặc thứ tự của các tham số để gọi đúng phương thức cần thực hiện.

```
class DemoOverloading
{
    0 references
    public int Sum(int A, int B)
    {
        return A + B;
    }

    0 references
    public float Sum(int A, float B)
    {
        return A + B;
    }
}
```

OVERRIDING

- Cho phép lớp con định nghĩa lại phương thức của lớp cha
- Phương thức được **override** (ở lớp cha) và phương thức **override** (ở lớp con) phải giống hệt nhau ở cả 3 phần: kiểu dữ liệu trả về, tên phương thức và danh sách tham số
- Không thể **override** *constructor*.



OVERRIDING

```
class baseClass
{
    3 references
    public virtual void show()
    {
        Console.WriteLine("Base class");
    }
}
1 reference
class derived : baseClass
{
    3 references
    public override void show()
    {
        Console.WriteLine("Derived class");
    }
}
```

```
static void Main(string[] args)
{
    baseClass obj;
    obj = new baseClass();
    obj.show();
    obj = new derived();
    obj.show();
}
```

OVERRIDE VÀ OVERLOAD

Override	Overload
– Kiểu dữ liệu trả về, tên phương thức, danh sách tham số của phương thức *override* và phương thức được *override* phải giống nhau.	– Kiểu dữ liệu trả về của các phương thức *overload* có thể giống nhau hoặc khác nhau. – Số lượng tham số hoặc kiểu dữ liệu của tham số ở các phương thức *overload* phải khác nhau.
– Không thể thu hẹp phạm vi truy cập(*access modifier*) của phương thức được override.	– Có thể mở rộng hoặc thu hẹp phạm vi truy cập(*access modifier*) của phương thức được overload.
– Không thể overriding *constructor method*.	– Overloading được *constructor method*.
– Chỉ thực hiện được đối với các class có quan hệ kế thừa. Do đó overriding thực hiện ở ngoài phạm vi của một class.	– Chỉ thực hiện trong cùng phạm vi, trong nội bộ của một class.
– Là hình thức đa hình khi chạy(*runtime*) (tức là chỉ khi chương trình chạy, thì chúng ta mới biết phương thức được gọi từ lớp nào).	– Là hình thức đa hình khi biên dịch(*compiler*) (tức là khi biên dịch mới biết đang sử dụng phương thức ở trong lớp nào).
– Không cho phép tạo ra những ngoại lệ khác loại hoặc không phải đối tượng thuộc lớp con của lớp có thể hiện là ngoại lệ từ phương thức được *override*.	– Cho phép tạo ra những ngoại lệ hoàn toàn mới so với những ngoại lệ từ phương thức được *overload*.

Từ khoá Sealed

- Một lớp được chỉ định với từ khoá sealed là một lớp không cho phép kế thừa

```
sealed class Class_Name
{
 //body of the class
}
```

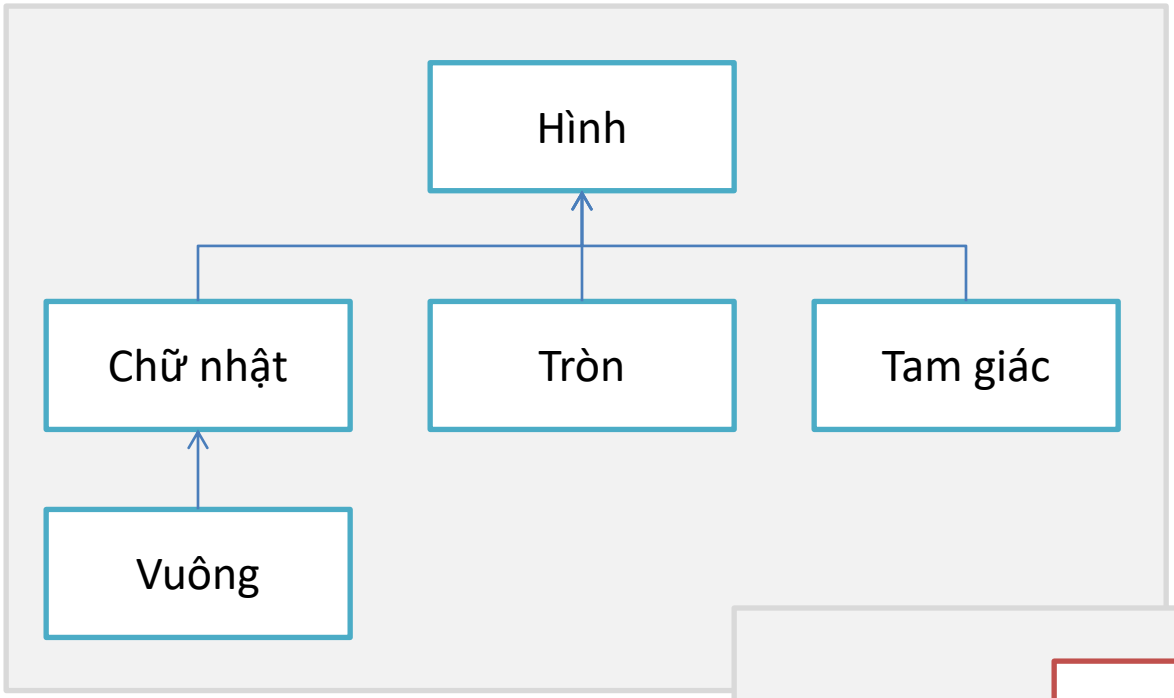
- Một phương thức được chỉ định với từ khoá sealed là một phương thức không cho overriding.

```
access_modifier sealed override return_type Method_Name
{
 //body of the method
}
```

LỚP TRỪU TƯỢNG

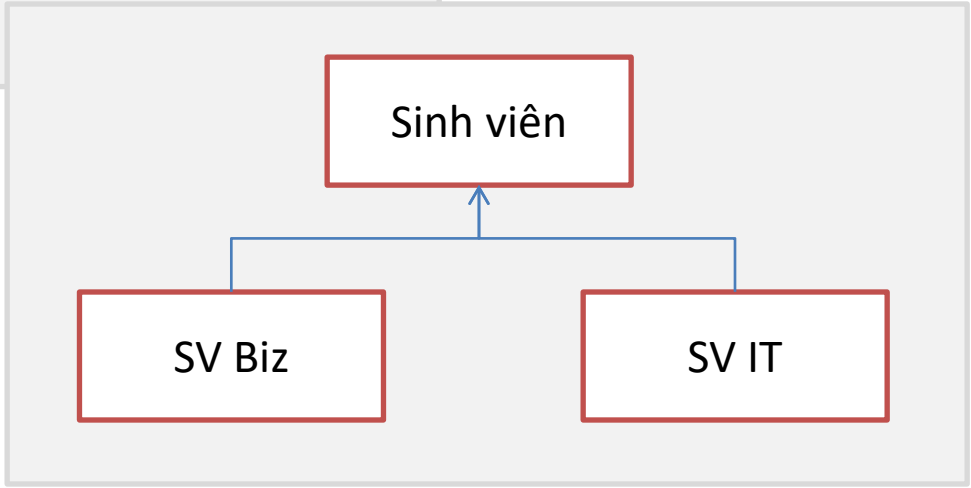
- Lớp trừu tượng là lớp có các hành vi chưa được xác định rõ
  - Ví dụ 1: Đã là hình thì chắc chắn là có diện tích và chu vi nhưng chưa xác định được cách tính mà phải là một hình cụ thể như chữ nhật, tròn, tam giác… mới có thể xác định cách tính
  - Ví dụ 2: Sinh viên thì chắc chắn có điểm trung bình nhưng chưa xác định được cách tính như thế nào mà phải là sinh viên của ngành nào mới biết được môn học và công thức tính điểm cụ thể.
- Vậy lớp hình và lớp sinh viên là các lớp trừu tượng vì phương thức tính chu vi, diện tích và tính điểm chưa thực hiện được.

LỚP TRỪ TƯỢNG



Hình và Sinh viên là các lớp trừu tượng

Chữ nhật, Tròn, Tam giác, Vuông, SV IT, SV Biz là các lớp cụ thể



ĐỊNH NGHĨA LỚP TRỪU TƯỢNG

```
abstract public class MyClass{
    abstract public type MyMethod();
}
```

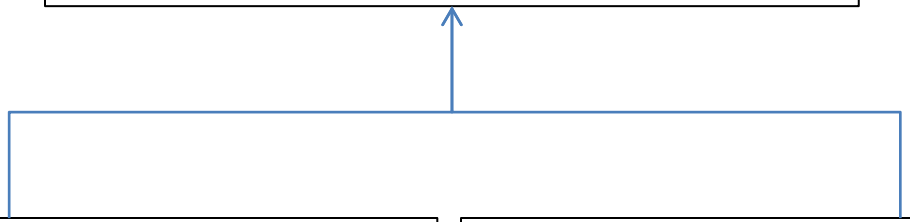
Sử dụng từ khóa abstract để định nghĩa lớp và phương thức trừu tượng

```
abstract public class SinhVien{
    abstract public double getDiemTB();
}
```

```
abstract public class Hinh{
    abstract public double getChuVi();
    abstract public double getDienTich();
}
```

ĐỊNH NGHĨA LỚP TRỪU TƯỢNG

```
abstract public class SinhVien{
    public String hoTen;
    abstract public double getDiemTB();
}
```

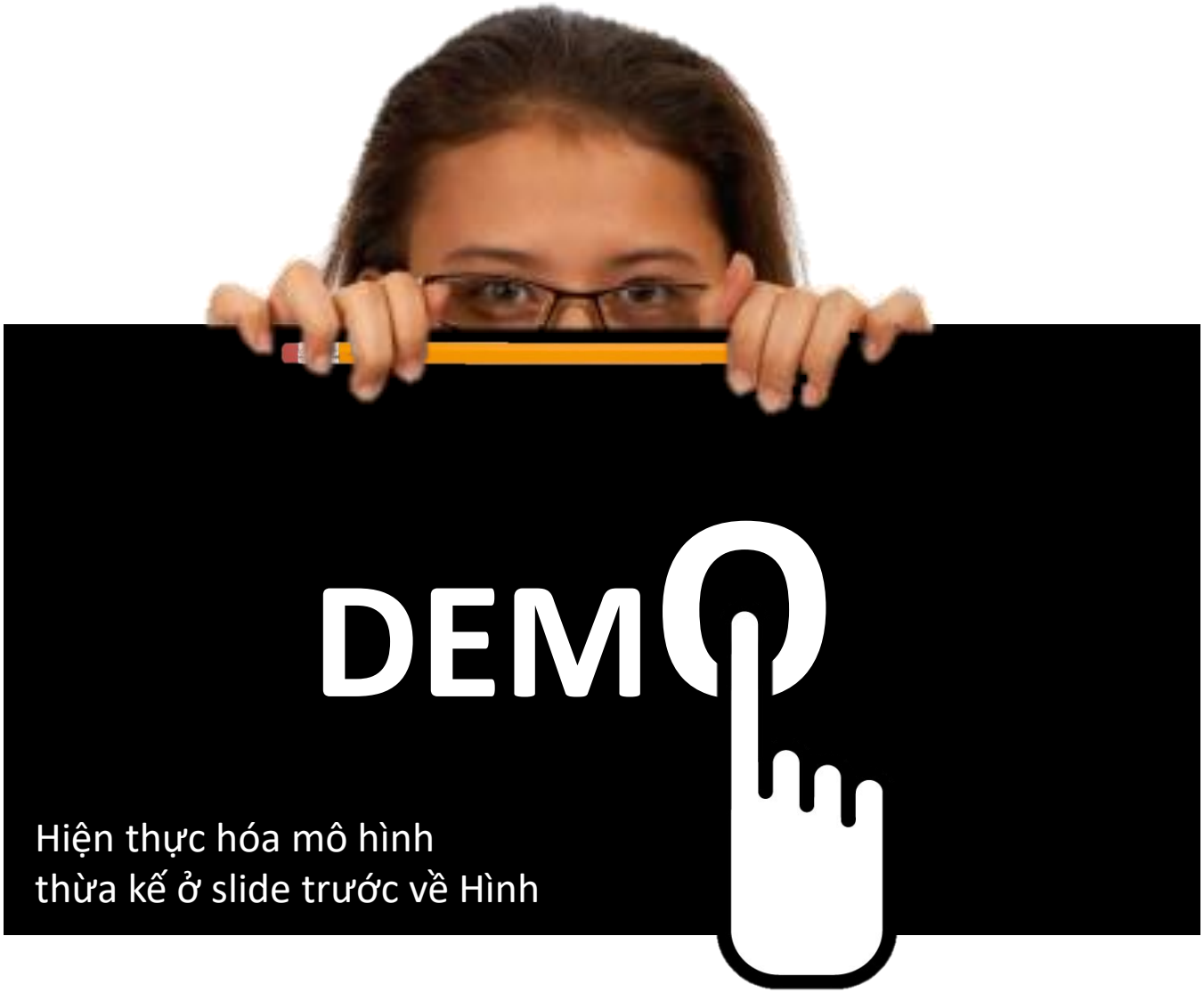


```
public class SinhVienIT : SinhVien{
    public double diemJava;
    public double diemCss;
    public override double getDiemTB(){
        return (2 * diemC# + diemCss)/3;
    }
}
```

```
public class SinhVienBiz : SinhVien {
    public double keToan;
    public double marketting;
    public double banHang;
    public override double getDiemTB(){
        return
        (keToan + marketting + banHang)/3;
    }
}
```

ĐỊNH NGHĨA LỚP TRỪU TƯỢNG

- Từ khóa **abstract** được sử dụng để định nghĩa lớp và phương thức trừu tượng
- Phương thức trừu tượng là phương thức không có phần thân xử lý và được khai báo bằng từ khóa abstract.
- Lớp chứa phương thức trừu tượng thì lớp đó phải là lớp trừu tượng.
- Trong lớp trừu tượng có thể định nghĩa các phương thức cụ thể hoặc khai báo các trường
- Không thể sử dụng new để tạo đối tượng từ lớp trừu tượng.


DEMO
Hiện thực hóa mô hình
thừa kế ở slide trước về Hình

Tổng kết bài học

- Phần I: Kế thừa
  - Các loại kế thừa
  - Từ khóa base, new
  - Constructor và tính kế thừa
- Phần II: Trừu tượng
  - Overiding
  - Lớp, phương thức trừu tượng

KẾT THÚC