CHƯƠNG 1 INTERNET & NETWORK PROGRAMMING

NỘI DUNG

- Giới thiệu
- Lý do lập trình mạng trên nền tảng .NET
- Phạm vi
- Địa chỉ IP
- Network stack
- Ports
- Internet standards
- .NET framework
- Visual Studio .NET
- .NET SDK

GIỚI THIỆU

- Khả năng phát triển ứng dụng mạng dựa trên ngôn ngữ lập trình C#
- Khảo sát các kỹ thuật lập trình trên mạng
- Thiết kế ứng dụng mạng
  - Bảo mật
  - Hiệu suất
  - Linh hoạt

LÝ DO CHỌN NỀN TẢNG .NET

- Hỗ trợ lập trình mạng tốt nhất so với các sản phẩm khác của Microsoft
- Có hỗ trợ đa nền tảng
- Cung cấp nhiều khả năng lập trình mạng mạnh mẽ
- Tuy nhiên, .Net không phải là lựa chọn duy nhất

PHẠM VI

- Một chương trình mạng là chương trình dùng mạng máy tính để truyền thông tin đến/từ ứng dụng khác
- Lập trình mạng có những điểm khác biệt với lập trình Web

ĐỊA CHỈ IP

- Mỗi máy tính kết nối trực tiếp vào Internet phải có địa chỉ IP duy nhất
- Phân biệt địa chỉ public IP và private IP
  - Ví dụ: 192.168.0.1 là private IP, 81.98.59.133 là public IP
- Một máy tính có thể có nhiều địa chỉ IP
- Nếu máy tính nhận địa chỉ 127.0.0.1 thì nó không kết nối với bất kỳ mạng nào

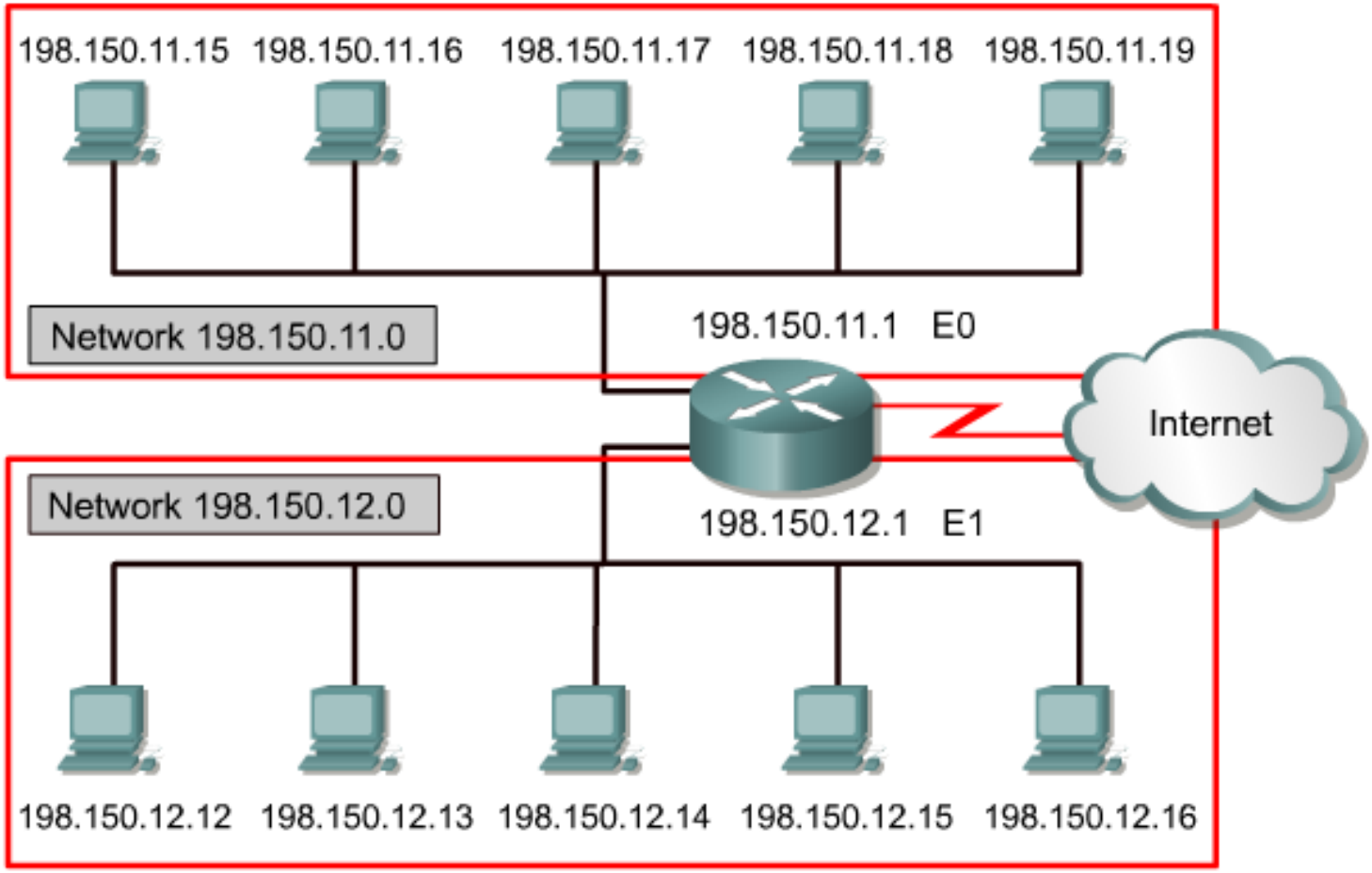
ĐỊA CHỈ DÀNH RIÊNG

Class	RFC 1918 internal address range
A	10.0.0.0 to 10.255.255.255
B	172.16.0.0 to 172.31.255.255
C	192.168.0.0 to 192.168.255.255

Private address: Địa chỉ dành riêng

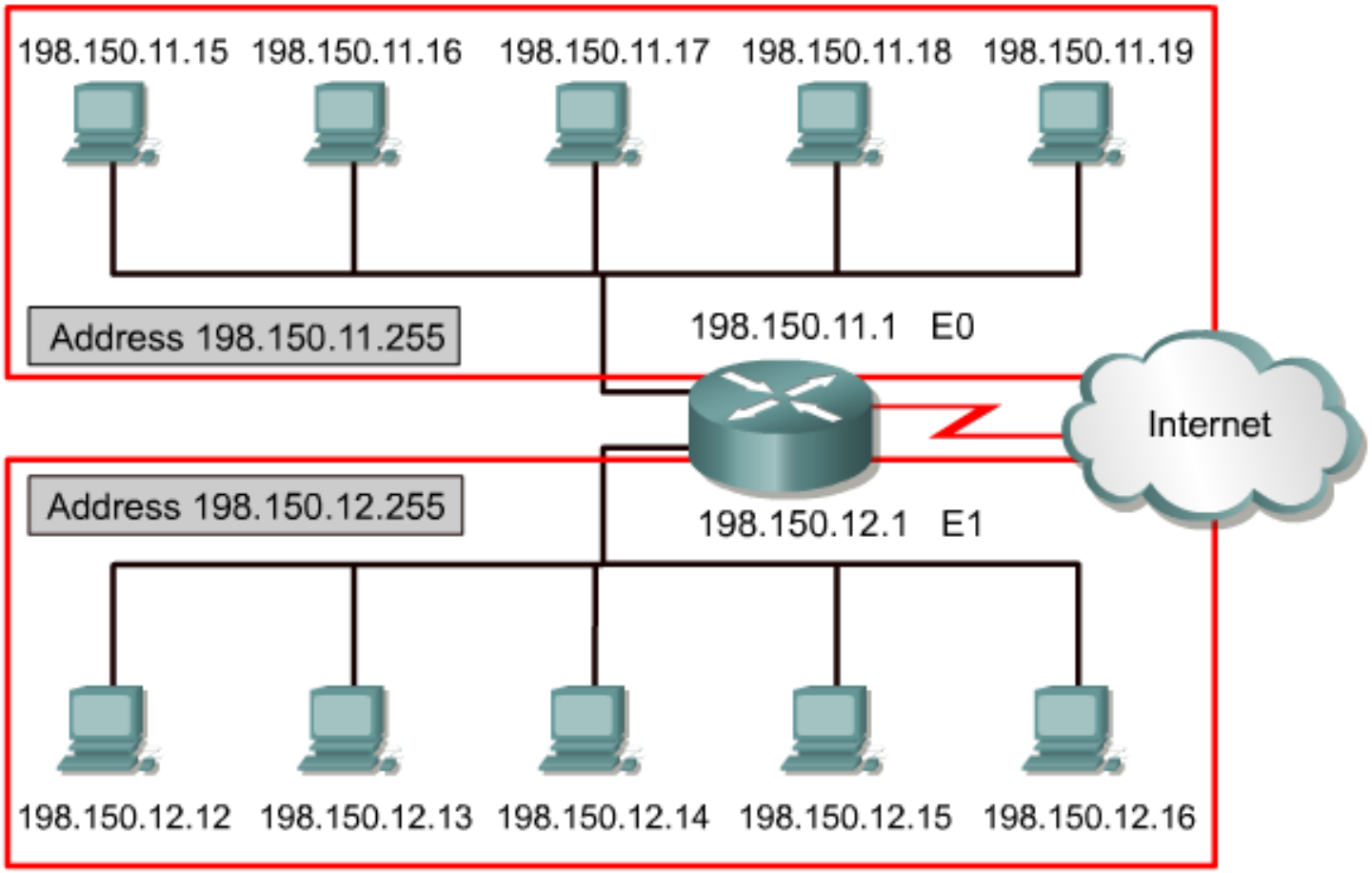
Public address: Địa chỉ dùng chung

CÁC LỚP ĐỊA CHỈ IP



Địa chỉ mạng

CÁC LỚP ĐỊA CHỈ IP



Địa chỉ broadcast

ĐỊA CHỈ IP

- Tất cả các máy tính với địa chỉ dành riêng phải kết nối với ít nhất 1 máy tính hoặc 1 router với địa chỉ dùng chung để truy cập Internet
- Địa chỉ IP của một máy tính có thể thay đổi → vai trò DHCP server
- Một địa chỉ duy nhất không thay đổi gắn với card mạng là địa chỉ MAC (còn gọi là địa chỉ phần cứng)

NETWORK STACK

- Quá trình lưu thông trên mạng của các tín hiệu là cực kỳ phức tạp, nếu không có khái niệm đóng gói (encapsulation) thì người lập trình sẽ "sa lầy" vào những chi tiết nhỏ
- Người lập trình chỉ cần tập trung vào điều gì xảy ra ở tầng cao trong OSI

NETWORK STACK CỔ ĐIỂN

Số thứ tự	Tên tầng	Giao thức
7	Application	FTP
6	Presentation	XNS
5	Session	RPC
4	Transport	TCP
3	Network	IP
2	Data link	Ethernet frames
1	Physical	Điện áp

NETWORK STACK HIỆN ĐẠI

Số thứ tự	Tên tầng	Giao thức
4	Structured Information	SOAP
3	Message	HTTP
2	Stream	TCP
1	Packet	IP

NETWORK STACK

- Không cần quan tâm thông tin được "lan truyền như thế nào", mà chỉ quan tâm "gửi cái gì"
- Không khảo sát các giao thức ở tầng vật lý → công việc của các kỹ sư điện tử, của hệ điều hành

PORTS

- Mỗi máy tính có thể có nhiều ứng dụng mạng chạy đồng thời
- Dữ liệu phải đính kèm thông tin cho biết ứng dụng nào dùng nó → port number
- Ví dụ: 80 cho ứng dụng Web, 110 cho ứng dụng email
- Thông tin port được chứa trong header của TCP, UDP packet

PORTS

Số thứ tự port	Giao thức
20	FTP (data)
21	FTP (control)
25	SMTP (email, outgoing)
53	DNS (domain names)
80	HTTP (Web)
110	POP3 (email, incoming)
119	NNTP (news)
143	IMAP (email, incoming)

INTERNET STANDARDS

- Người lập trình phải chú ý đến các chuẩn do 2 tổ chức lớn đưa ra là Internet Engineering Task Force (IETF) và World Wide Web Consortium (W3C)
- RFC do IETF định nghĩa
- HTML, XML do W3C định nghĩa

CÁC RFC QUAN TRỌNG

RFC Document	Giao thức
RFC 821	SMTP (email, outgoing)
RFC 954	WHOIS
RFC 959	FTP (uploading and downloading)
RFC 1939	POP3 (email, incoming)
RFC 2616	HTTP (Web browsing)
RFC 793	TCP (runs under all above protocols)
RFC 792	ICMP (ping)
RFC 791	IP (runs under TCP and ICMP)

.NET FRAMEWORK

- .NET không phải là ngôn ngữ lập trình, nó cung cấp một framework cho 4 NNLT làm việc với nhau, gồm: C#, VB.NET, Managed C++, J# .NET.
- Framework định nghĩa Framework Class Library (FCL)
- 4 NNTL trên chia sẻ FCL và Common Language Runtime (CLR) – nền tảng môi trường thực thi cho các ứng dụng .NET

.NET FRAMEWORK

- CLR là máy ảo (tương tự VM trong Java) được thiết kế riêng cho Windows
- Các NNLT trong .NET có khả năng cộng tác với nhau rất tốt
- 2 NNLT phổ biến nhất được chọn để minh họa là C# và VB.NET
- Common Type System (CTS) chỉ khái niệm tái sử dụng mã giữa các NNLT trong .NET. Ví dụ: class (đã biên dịch) trong C# dùng được trong VB.NET và ngược lại

.NET FRAMEWORK

- Khi ứng dụng viết trong .NET được biên dịch trở thành mã trung gian gọi là Microsoft Intermediate Language (MSIL) byte code, thực thi được trên CLR
- Để khắc phục nhược điểm của thông dịch, .NET dùng trình biên dịch Just-in-time (JIT).

.NET FRAMEWORK

- JIT hoạt động theo yêu cầu, bất kỳ khi nào mã MSIL thực thi lần đầu tiên
- JIT biên dịch mã MSIL sang mã máy
- Không hỗ trợ đa thừa kế
- Mọi lớp trong .NET đều thừa kế từ System.Object

HƯỚNG ĐỐI TƯỢNG TRONG C#

- Namespace, Lớp và Đối tượng
- Các thành phần của Lớp, Đối tượng
- Constructors và Destructors
- Nạp chồng phương thức (Overloading)
- Các phương thức chồng toán tử (Operator Overloading)
- Viết lại các thành phần của lớp (Overriding)
- Kế thừa (Inheritance)

NAMESPACE

- Tránh sự trùng lắp khi đặt tên lớp
- Quản lý mã được dễ dàng
- Giảm bớt sự phức tạp khi chạy với các ứng dụng khác

```
namespace Tên_Namespace
{
        //Khai báo các lớp…
}
```

- Có thể khai báo các namespace, class… bên trong namespace khác.

NAMESPACE: VÍ DỤ 1

```
namespace  Sample
{
        public class A
        {
        }
        public class B
        {
        }
}
```

NAMESPACE: VÍ DỤ 2

```
namespace  Sample_2
{
        public class A
        {
        }
        namespace Sample_3
        {
                //….
        }
}
```

LỚP VÀ ĐỐI TƯỢNG

- Khai báo:

```
class Tên_lớp
{
        //khai báo các thành phần…
}
```

- Ví dụ:

```
class KhachHang
{
        private int mMaKhachHang;
        private string mTenKhachHang;
}
```

LỚP VÀ ĐỐI TƯỢNG

```
class KhachHang
{
        //Các thành phần
        //Các phương thức
        public  void  In()
        {
                //Các câu lệnh…
        }
}
```

CÁC THÀNH PHẦN TRONG LỚP

- Thành phần của lớp

  - Khai báo với từ khóa static

```
class KhachHang
{
        private static int mMaKH;
        public static string mTenKH;
        public static void In() {
        // Các câu lệnh
        }
}
```

CÁC THÀNH PHẦN TRONG LỚP

- Sử dụng: TênLớp.TênThànhPhần
- Ví dụ:

```
KhachHang.mMaKH = 1;
KhachHang.In();
```

THÀNH PHẦN CỦA ĐỐI TƯỢNG

```
class KhachHang
{
        private int mMaKH;
        public string mTenKH;
        public void In() {
                // Các câu lệnh
        }
}
```

- Sử dụng: TênĐốiTượng.TênThànhPhần
- Ví dụ:

```
        KhachHang objKH = new KhachHang();
        objKH.In() ; objKH.mTenKH = “ABC”;
```

CONSTRUCTORS TRONG C#

- Phương thức đặc biệt trong lớp
- Được gọi khi đối tượng được tạo
- Dùng để khởi dựng đối tượng
- Cùng tên với tên lớp
- Không có giá trị trả về
- Constructor có thể có tham số

CONSTRUCTORS TRONG C#: VÍ DỤ 1

```
class KhachHang
{
        private int mMaKH;
        private string mTenKH;
        public KhachHang()
        {
                mMaKH = 0;
                mTenKH = “ABC”;
        }
 }
```

CONSTRUCTORS TRONG C#: VÍ DỤ 2

```
class KhachHang
{
        private int mMaKH;
        private string mTenKH;
        public KhachHang(int MaKH, string TenKH)
        {
                mMaKH = MaKH;
                mTenKH = TenKH;
        }
}
```

STATIC CONSTRUCTOR

- Gọi một lần duy nhất trước khi đối tượng được tạo
- Không có tham số

```
class KhachHang
{
        public KhachHang() {
                mMaKH = 0;
                mTenKH = “ABC”;
        }
        static KhachHang()
        {
                // Các câu lệnh…
        }
}
```

PRIVATE CONSTRUCTOR

- Sử dụng khi các thành phần trong lớp là static
- Không cần thiết tạo đối tượng cho lớp

```
class KhachHang
{
        private static int mMaKH;
        public static string mTenKH;
        public static void In() {
                // Các câu lệnh
        }
        private KhachHang()
        {
        }
}
```

DESTRUCTORS TRONG C#

- Được gọi bởi Garbage Collector
- Được gọi tự động khi đối tượng được hủy

```
class KhachHang
{
        public KhachHang() {
                mMaKH = 0; mTenKH = "ABC";
        }
        ~KhachHang()
        {
                // Các câu lệnh…
        }
}
```

OVERLOADING METHODS

Các phương thức có cùng tên, khác danh sách tham số hoặc kiểu tham số hoặc kiểu giá trị trả về

```
public void In()
{
        // Các câu lệnh
}
public void In(string s)
{
        // Các câu lệnh
}
public void In(int s)
{
        // Các câu lệnh
}
```

PHÁT BIỂU IF

- Cú pháp:

```
if ( <biểu thức Boolean> )
{
        //Các câu lệnh …
}
[ else
{
        //Các câu lệnh …
} ]
```

PHÁT BIỂU SWITCH

- Cú pháp:

```
switch (<biểu thức kiểu rời rạc> )
   {
        case <giá trị 1>:
                  //Các câu lệnh 1…
                 break;
        case <giá trị 2>:
                  //Các câu lệnh 2…
                 break;
        …
        default:
                  //Các câu lệnh default…
       break;
   }
```

PHÁT BIỂU LẶP

- Thực hiện một số lệnh nào đó trong thân vòng lặp với một số lần xác định hoặc khi một biểu thức đánh giá điều kiện còn đúng (true)
- Các loại phát biểu lặp gồm:
  - while
  - do .. while
  - for
  - foreach

PHÁT BIỂU WHILE

- Cú pháp:

```
while (<biểu thức điều kiện>)
{
        //các phát biểu
}
```

- Ý nghĩa: Thực hiện lặp các phát biểu trong thân vòng lặp trong khi <biểu thức điều kiện> còn đúng. Kiểm tra <biểu thức điều kiện> trước khi thực hiện các phát biểu

PHÁT BIỂU DO .. WHILE

- Cú pháp:

```
do {
        //các phát biểu
} while (<biểu thức điều kiện>)
```

- Ý nghĩa: Thực hiện lặp các phát biểu trong thân vòng lặp khi <biểu thức điều kiện> còn đúng. Kiểm tra <biểu thức điều kiện> sau khi đã thực hiện các phát biểu 1 lần

PHÁT BIỂU FOR

- Cú pháp:

  for (<biến điều khiển> = <giá trị 1>; <biểu thức điều kiện>; <tăng/giảm biến điều khiển>)

  {

  //các phát biểu

  }

- Ý nghĩa: gán <biến điều khiển> = <giá trị 1> kiểm tra <biểu thức điều kiện> đúng thì thực hiện các phát biểu, tăng/giảm biến, quay lại kiểm tra <biểu thức điều kiện>

PHÁT BIỂU FOREACH

- Cú pháp:

  foreach (<biến điều khiển> in <tập hợp/nhóm control/mảng>)

  {

  //các phát biểu

  }

- Ý nghĩa: duyệt qua tất cả các phần tử trong <tập hợp/nhóm control/mảng> và thực hiện các phát biểu

PHÁT BIỂU BREAK VÀ CONTINUE

- break – ngắt ngang vòng lặp bất kỳ lúc nào
- continue – bỏ qua lần duyệt hiện tại và bắt đầu với lần kế tiếp
- Có thể được dùng trong bất kỳ loại vòng lặp nào

KIỂU DỮ LIỆU MẢNG

- Một tập hợp các giá trị có cùng kiểu dữ liệu
- Cú pháp khai báo:

- Truy xuất: <Tên mảng>[<chỉ số phần tử>]

```
DataType[số lượng phần tử] ArrayName;
```

- Ví dụ: `int[6] array1;`

KIỂU DỮ LIỆU CẤU TRÚC

- Các kiểu dữ liệu người dùng tự định nghĩa
- Chứa các thành phần có thể có kiểu dữ liệu khác nhau
- Có thể định nghĩa phương thức bên trong
- Không thể thừa kế

```
struct structEx
{
    public int
strIntDataMember;
    private string
strStrDataMember;
    public void
structMethod1()
    {
        //các phát biểu
    }
}
```

KIỂU DỮ LIỆU LIỆT KÊ (ENUMERATOR)

```
public class Holiday
{
    public enum WeekDays
    {
        Monday,
        Tuesday,
        Wednesday,
        Thursday,
        Friday
    }
    public void GetWeekDays (String EmpName, WeekDays DayOff)
    {
         //Process WeekDays
    }
    static void Main()
    {
      Holiday myHoliday = new Holiday();
      myHoliday.GetWeekDays ("Richie", Holiday.WeekDays.Wednesday);
    }
}
```

KẾ THỪA TRONG C#

- Cho phép khai báo 1 lớp mới được dẫn xuất từ 1 lớp đã có
- Sử dụng lại các đoạn mã đã viết
- Hỗ trợ đơn thừa kế
- Không cho phép đa thừa kế
- Cho phép thực thi nhiều interface

KẾ THỪA TRONG C#

```
class Software
{
        private int m_z;
        public int m_v;
        protected int m_x;
        public Software()
        {
                m_x = 100;
        }
        public Software(int y)
        {
                m_x = y;
        }
}
```

KẾ THỪA TRONG C#

```
class MicrosoftSoftware : Software
{
        public MicrosoftSoftware()
        {
                Console.WriteLine(m_x);
        }
}
```

KẾ THỪA TRONG C#

```
class IBMSoftware : Software
{
        public IBMSoftware(int y) : base(y)      {
                Console.WriteLine(m_x);
                }
                public IBMSoftware(string s, int f) : this(f)
  {
                Console.WriteLine(s);
                }
}
```

KẾ THỪA TRONG C#

```
static void Main(string[] args)
{
 MicrosoftSoftware objMS = new
 MicrosoftSoftware();
 IBMSoftware objIBM1 = new IBMSoftware(50);
 IBMSoftware objIBM2 = new IBMSoftware("test",
 75);
 Console.ReadLine();
}
```

KẾ THỪA TRONG C#

Từ khóa sealed: Lớp không cho phép kế thừa

```
public sealed class A
{
}
public class B : A
{
}
```

Lớp B không được phép kế thừa lớp A.

OVERRIDING METHOD

```
class Animal
{
	public Animal()
	{
		Console.WriteLine("Animal constructor");
	}
	public void Talk()
	{
		Console.WriteLine("Animal talk");
	}
}
```

OVERRIDING METHOD

```
class Dog : Animal
{
      public Dog()
      {
            Console.WriteLine("Dog constructor");
      }
      public new void Talk()
      {
            Console.WriteLine("Dog talk");
      }
}
```

OVERRIDING METHOD

```
class Test
 {
      static void Main(string[] args)
      {
           Animal a1 = new Animal();
           a1.Talk();
           Dog d1 = new Dog();
           d1.Talk();
      }
}
```

TÍNH ĐA HÌNH - POLYMORPHISM

```
class Animal
{
	public Animal()
	{
		Console.WriteLine("Animal constructor");
	}
	public virtual void Talk()
	{
		Console.WriteLine("Animal talk");
	}
}
```

TÍNH ĐA HÌNH - POLYMORPHISM

```
class Dog : Animal
{
	public Dog() {
		Console.WriteLine("Dog constructor");
	}
	public override void Talk()
	{
		Console.WriteLine("Dog talk");
	}
}
```

TÍNH ĐA HÌNH - POLYMORPHISM

```
class Test
 {
       static void Main(string[] args)
       {
              Animal objA = new Animal();
              Dog objD = new Dog();
              objA = objD ;
              objA.Talk();
       }
 }
```

LỚP TRỪU TƯỢNG – ABSTRACT CLASS

- Không được tạo đối tượng
- Có thể định nghĩa các phương thức
- Có thể mở rộng từ lớp dẫn xuất
- Dùng để làm lớp cơ sở
- Có thể thực thi interface

LỚP TRỪU TƯỢNG – ABSTRACT CLASS

```
abstract class Shape
{
      protected float m_Height = 5;
      protected float m_Width = 10;
      public abstract void CalculateArea();
      public abstract void CalculateCircumference();
      public  void PrintHeight() {
            Console.WriteLine("Height =
{0}",m_Height);
      }
      public void PrintWidth()      {
            Console.WriteLine("Width =
{0}",m_Width);
      }
}
```

LỚP TRỪU TƯỢNG – ABSTRACT CLASS

```
class Rectangle:Shape
{
        public Rectangle() {
                m_Height = 20;
                m_Width = 30;
        }
        public override void CalculateArea() {
                Console.WriteLine("Area : {0}", m_Height * m_Widt
);
        }
        public override void CalculateCircumference() {
                Console.WriteLine("Circumference =
{0}",(m_Height+m_Width)*2);
        }
}
```

LỚP TRỪU TƯỢNG – ABSTRACT CLASS

```
 class Test
{
        static void Main(string[] args)
        {
                Rectangle objRec =  new Rectangle();
                objRec.CalculateArea();
                objRec.CalculateCircumference();
        }
}
```

GIAO DIỆN – INTERFACE

- Không được tạo đối tượng
- Không thể định nghĩa các phương thức
- Lớp thực thi interface phải thực thi tất cả các phương thức của interface
- Interface có thể được kế thừa các interface khác

GIAO DIỆN – INTERFACE

```
interface ITest
{
        void Print();
}
class Base : ITest
{
        public void Print()
        {
                Console.WriteLine("Print method called");
        }
    }
```

GIAO DIỆN – INTERFACE

```
static void Main(string[] args)
{
        Base obj = new Base();
        obj.Print();
         //Gọi phương thức Print() bằng  interface  ITest
         ITest ib = (ITest)obj ;
         ib.Print();
        //Gọi phương thức Print() bằng cách ép kiểu Interface ITest về
lớp Base
         Base ojB = (Base)ib;
        ojB.Print();
}
```

BÀI TẬP

- Tham khảo thêm một số tài liệu hướng dẫn lập trình với ngôn ngữ C#
- Cài đặt một số chương trình để luyện tập