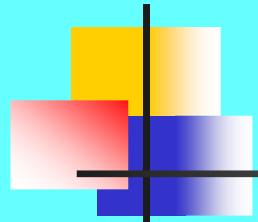


BÀI GIẢNG MÔN HỌC

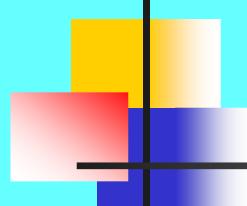


LẬP TRÌNH MẠNG



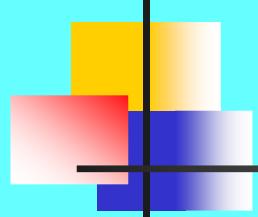
Nội Dung

- Mở đầu
- Chương 1 : Tổng quan về mạng máy tính
- Chương 2 : Các thiết bị và các chuẩn kết nối
- Chương 3 : Các giao thức cơ bản
- Chương 4 : Ngôn ngữ lập trình Java
- Chương 5 : Lập trình Socket với giao thức TCP
- Chương 6 : Lập trình ứng dụng cho giao thức UDP
- Chương 7 : Lập trình cơ sở dữ liệu
- Chương 8 : Lập trình ứng dụng với các giao thức khác (JSP, SERVLET, MVC)



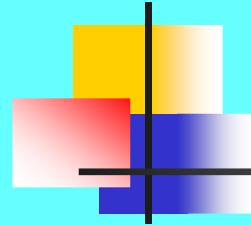
Tài liệu tham khảo

- Slide Bài giảng lập trình mạng – Khoa CNTT
- David Reilly, Michael Reilly (2002) *Java Network Programming and Distributed Computing*, Publisher : Addison Wesley, March 25, 2002, 496 p. IBM (2002) *Socket programming*, 198 p.
- Douglas E. Comer, *Internetworking with TCP/IP*, Vol 1 & 3, Prentice-Hall, 1993.
- Behrouz A. Forouzan, DeAnza College, *TCP/IP Protocol Suite*, second edition, McGraw-Hill, 2000.
- Douglas E. Comer, *Computer Networks and Internets with Internet Applications*, Prentice-Hall, 1993.
- James F. Kurose, Keith W. Ross, *Computer Networking – A top-down Approach Featuring the Internet*, Addison Wesley, 2001.



Mở đầu

- Khái niệm lập trình mạng
- Đối tượng và phạm vi môn học
- Các loại hệ điều hành
- Giao thức mạng
- Ngôn ngữ lập trình mạng
- Các phương pháp lập trình



Khái niệm lập trình mạng

- Tạo ra các thực thể phần mềm hoạt động trên một tầng
 - Sử dụng các thực thể ở tầng phía dưới
 - Cung cấp dịch vụ cho các thực thể tầng kế trên
- Chủ yếu tạo ra các thực thể phần mềm ở tầng ứng dụng
 - Cung cấp dịch vụ cho người dùng

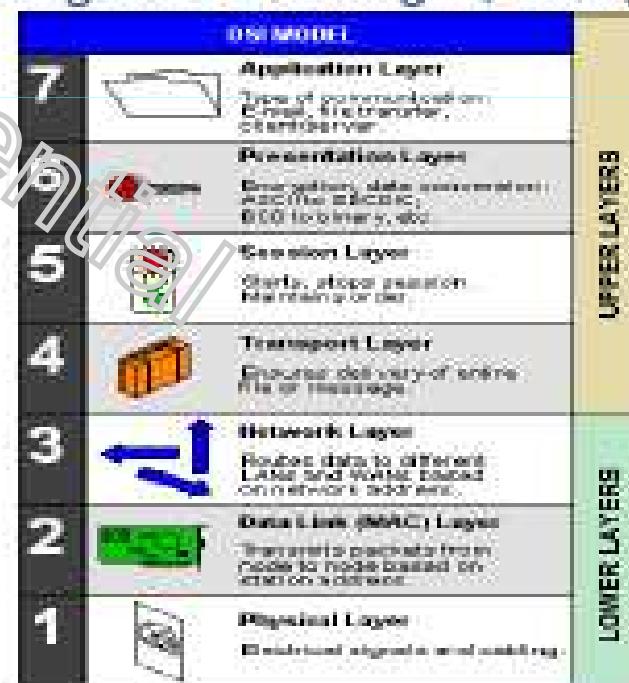
Đối tượng và phạm vi môn học

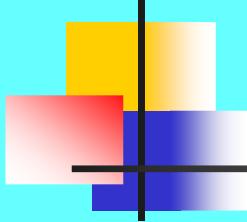
■ Đối tượng lập trình mạng

▫ Các thực thể phần mềm thực thi giao thức trong hệ thống

mạng.

- ✓ được xây dựng dựa trên nền tảng hệ thống máy tính
- ✓ phần cứng và hệ điều hành, kiến trúc phân tầng mạng





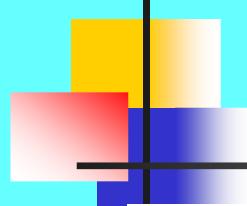
Đối tượng và phạm vi môn học

■ Phạm vi môn học

- Tập trung vào kỹ thuật lập trình sử dụng dịch vụ tại tầng transport để xây dựng các ứng dụng mạng.

■ Hạ tầng truyền thông

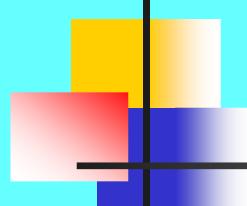
- Một ứng dụng hay một dịch vụ mạng cần có hạ tầng mạng bên dưới khi hoạt động.
- Tùy theo yếu tố kỹ thuật hay yêu cầu đối với ứng dụng mà ta cần phải lựa chọn loại mạng cho ứng dụng và dịch vụ



Các loại hệ điều hành

■ Unix

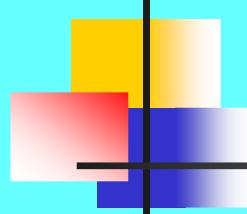
- Do các nhà khoa học tại nhiều viện NC, phòng thí nghiệm (Bell Lab) khởi đầu xây dựng nên.
- Là hệ điều hành đa nhiệm, cả người sử dụng và phục vụ truyền thông rất tốt.
- Hạn chế: có nhiều phiên bản, phức tạp trong quản trị và sử dụng, đòi hỏi chạy trên các máy Server cấu hình rất mạnh.



Các loại hệ điều hành

■ LINUX

- Linus Tovald phát triển từ nhân của MINIX (một phiên bản của UNIX thu nhỏ) với mục đích tạo ra một hệ điều hành mới cho PC.
- Có nhiều phiên bản khác nhau
 - ✓ Redhat Linux, Mandrake Linux...
 - ✓ LINUX cho từng quốc gia: Hoa Kỳ, Trung Quốc, Vietkey Linux...
- Dùng cho cả máy trạm, máy chủ và siêu máy tính.
- Linux là hệ đa nhiệm, đa người dùng, tinh ồn định cao, hỗ trợ truyền thông tốt, và là hệ điều hành gần như miễn phí.



Các loại hệ điều hành

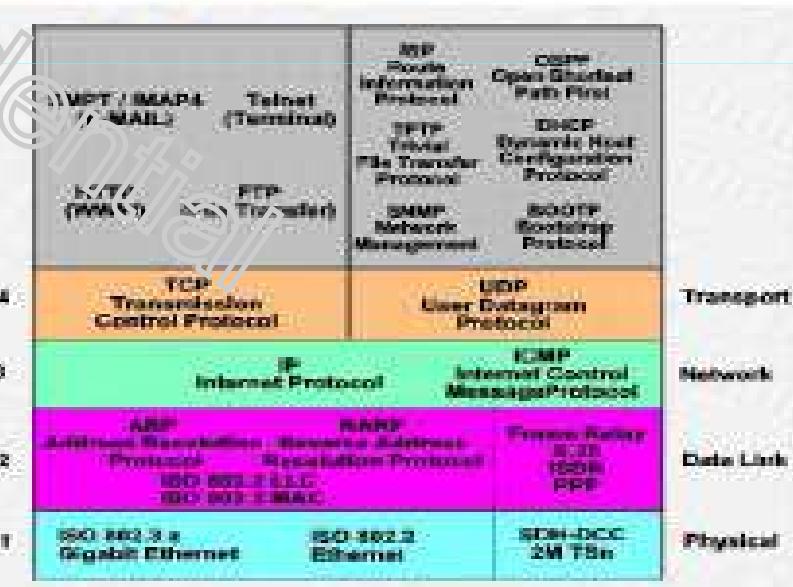
■ Windows

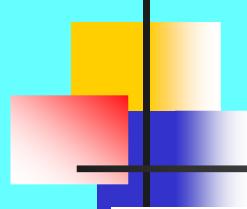
- Cũng là hệ điều hành đa nhiệm, đa người sử dụng, với các tính năng hỗ trợ mạng.
- Dễ sử dụng
- Có các phiên bản cho cả máy trạm và máy chủ.
- Hỗ trợ rất nhiều loại dịch vụ.
- Tuy nhiên, có nhiều hạn chế
 - ✓ bảo mật kém và ít ổn định so với UNIX và LINUX.

Giao thức mạng

■ Trong phạm vi môn học này, trọng tâm sử dụng bộ giao thức TCP/IP do các lý do sau:

- Là bộ giao thức phổ biến nhất, có thể dùng:
 - ✓ mọi loại mạng.
 - > LAN, WAN, và Internet.
 - ✓ mọi hệ điều hành,
 - ✓ các thiết bị phần cứng





Ngôn ngữ lập trình mạng

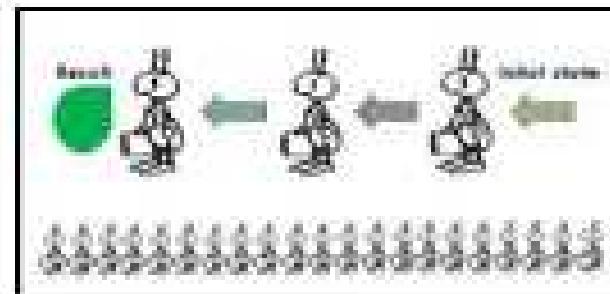
Ngôn ngữ lập trình và công cụ phát triển

- Có rất nhiều ngôn ngữ cho phép thực thi các tác vụ qua mạng dựa trên các bộ thư viện khác nhau.
- Các ngôn ngữ phổ biến nhất:
 - C/C++
 - Java
 - .NET
 - BASIC
 - DELPHI

Các phương pháp lập trình

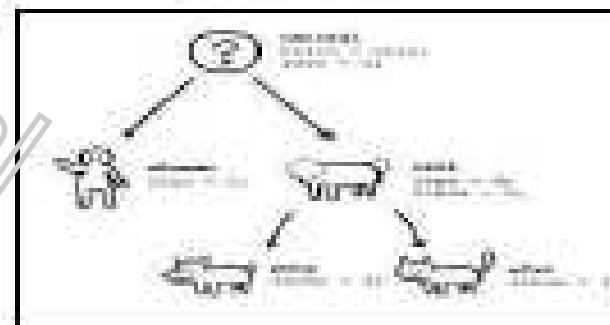
■ LT thủ tục

- Chia chương trình thành các chương trình con (chia để trị)
 - ✓ Hàm, thủ tục



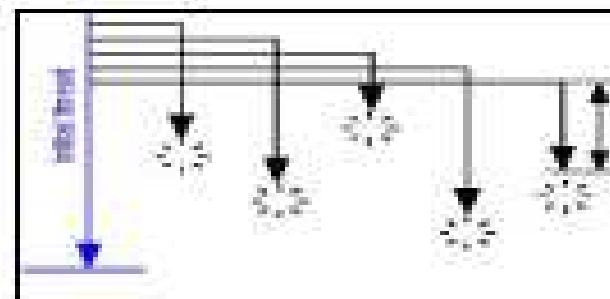
■ LT hướng đối tượng

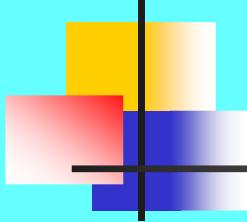
- Thiết kế chương trình theo hướng đối tượng.
 - ✓ tạo thư viện phục vụ LT mang thành các gói, lớp đối tượng
 - ✓ sử dụng một số các thư viện đối tượng sẵn có



■ LT đa tuyến

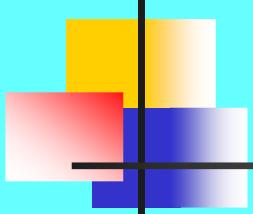
- Tận dụng tối đa khả năng của bộ vi xử lý, thực hiện nhiều tác vụ đồng thời





Chương 1 : Tổng quan về mạng máy tính

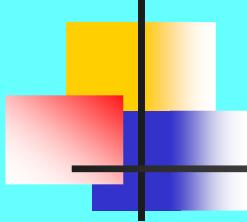
- 1.1 Khái niệm và mục đích kết nối mạng
- 1.2 Đặc trưng kỹ thuật của mạng máy tính
- 1.3 Phân loại mạng máy tính
- 1.4 Chuẩn hóa mạng máy tính
- 1.5 Mô hình phân tầng thu gọn
- 1.6 Nguyên tắc truyền thông



Chương 1 : Tổng quan về mạng máy tính

1.1 Khái niệm và mục đích kết nối mạng

- *Là tập hợp các máy tính được kết nối với nhau thông qua các đường truyền vật lý và tuân theo các quy ước truyền thông nào đó.*
- Các đường truyền vật lý được hiểu là các môi trường truyền (có thể là hữu tuyến hoặc vô tuyến).
- Các quy ước truyền thông (giao thức) chính là cơ sở để các máy tính có thể trao đổi được với nhau và là một yếu tố quan trọng hàng đầu khi nói về công nghệ mạng máy tính.



Chương 1 : Tổng quan về mạng máy tính

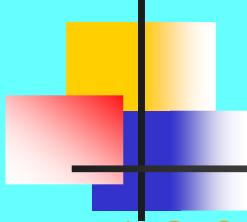
1.2 Đặc trưng kỹ thuật của mạng máy tính

1.2.1. Đường truyền

- Là phương tiện dùng để truyền các tín hiệu điện tử giữa các máy tính.
- Các tín hiệu điện tử chính là các thông tin, dữ liệu được biểu thị dưới dạng các xung nhị phân (ON/OFF), mọi tín hiệu truyền giữa các máy tính với nhau đều thuộc sóng điện tử
- Tuỳ theo tần số mà ta có thể dùng các đường truyền vật lý khác nhau
 - Đường truyền hữu tuyến (dây dẫn)
 - Đường truyền vô tuyến: thông qua các sóng vô tuyến

1.2.2. Kỹ thuật chuyển mạch

- Là đặc trưng kỹ thuật chuyển tín hiệu giữa các nút trong mạng, có các kỹ thuật chuyển mạch như sau:
 - Kỹ thuật chuyển mạch điện (chuyển mạch kênh)
 - Kỹ thuật chuyển mạch thông báo
 - Kỹ thuật chuyển mạch gói



Chương 1 : Tổng quan về mạng máy tính

1.2.2.1 Chuyển mạch điện:

+ Sự liên lạc thông qua chuyển mạch điện gồm có 3 pha: xác lập mạch, truyền số liệu và giải phóng mạch.

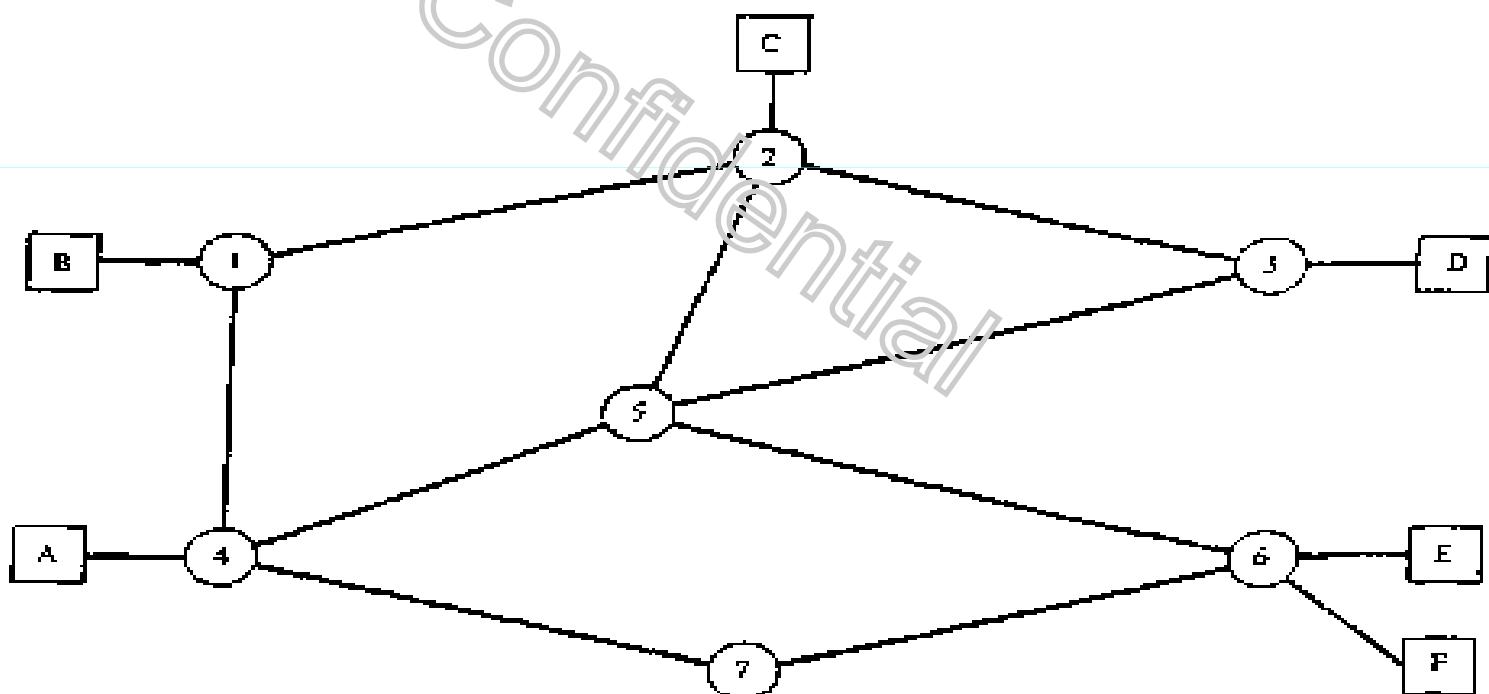
a. Xác lập mạch:

+ Trước khi có thể truyền được số liệu từ đầu này đến đầu kia. Đường truyền trong mạng cần được xác lập.

Ví dụ: Thuê bao A cần truyền số liệu cho E. Nó yêu cầu Node 4 sự nối mạch đến E. Đường nối từ A đến 4 trên thực tế đã có. Node 4 cần tìm nhánh tiếp theo để tạo đường đến 6.

Chương 1 : Tổng quan về mạng máy tính

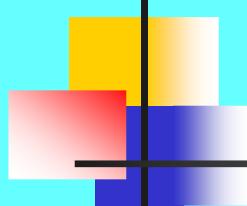
1.2.2.1 Chuyển mạch điện:



Chương 1 : Tổng quan về mạng máy tính

1.2.2.1 Chuyển mạch điện:

- + Dựa vào những thông tin về tìm đường và sự cho phép về các thông số khác.
- + Node 4 chọn đường đến 5 trong khi có nhiều thuê bao được nối đến 4 nó cần phải xác lập con đường bên trong từ nhiều thuê bao đến nhiều Node khác.
- + Node 5 sẽ tạo cho ta con đường từ 5 đến 6 và như vậy con đường từ 4 đến 6 được xác lập và Node 6 hoàn thành công việc nối với E, trước khi hoàn tất nó cần phải kiểm tra xem E có bị bận hay không



Chương 1 : Tổng quan về mạng máy tính

1.2.2.1 Chuyển mạch điện:

b. Truyền dữ liệu:

- + Thông tin có thể bắt đầu truyền từ A sang E. Dữ liệu có thể trong dạng digital hoặc dạng analog.
- + Thông tin theo đường A đến 4, qua điểm nối mạch bên trong của 4 đến 5, qua điểm nối mạch bên trong của 5 đến 6, qua điểm nối mạch bên trong của 6 đến E.
- + Thông thường sự nối mạch cho phép hai chiều toàn phần và dữ liệu có thể truyền 2 chiều.



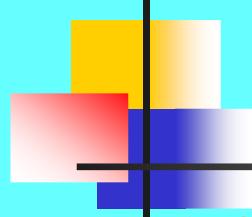
Chương 1 : Tổng quan về mạng máy tính

1.2.2.1 Chuyển mạch điện:

c. Giải phóng mạch:

- + Sau khi hoàn thành sự truyền. Có tín hiệu báo của A hoặc E, báo cho 4, 5, 6 giải phóng sự nối mạch. Đường nối từ A đến E không còn nữa.
- + Tuy nhiên khi đường nối giữa 2 thuê bao được nối thì dữ liệu được truyền trên một đường cố định, không có thời gian giữ lại ở các Node.

Chương 1 : Tổng quan về mạng máy tính

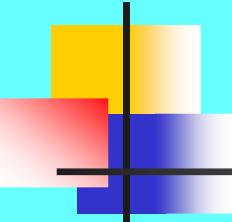


1.2.2.1 Chuyển mạch điện:

Đặc điểm :

- + Mỗi nút mạng đóng vai trò là một tổng đài.
- + Đường nối giữa các nút mạng : đường trung kế được thiết kế dựa trên sự dự báo lưu lượng truyền dẫn trong những khu vực khác nhau
- + Liên lạc thông qua chuyển mạch mạch điện đặc trưng bởi việc cung cấp đường nối cố định giữa 2 thuê bao. Nó có thể là đường nối vật lý hay kênh thông tin.

Chương 1 : Tổng quan về mạng máy tính



1.2.2.1 Chuyển mạch điện:

Đặc điểm :

- + Mất khoảng thời gian để thiết lập mạch.
- + Một kênh truyền dẫn sẽ được thiết lập và tồn tại từ khi bắt đầu cuộc truyền dữ liệu cho đến khi kết thúc cuộc truyền dữ liệu
- + Sau khi hai thuê bao hoàn thành công việc của mình thì một trong hai thuê bao có thể gửi yêu cầu cắt mạch và mạng sẽ giải phóng kênh truyền. Khi đó kênh mới được cấp phát cho thuê bao khác.

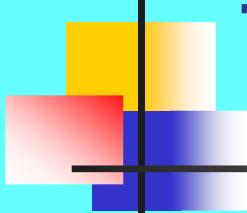
Chương 1 :

Tổng quan về mạng máy tính

1.2.2.1 Chuyển mạch điện:

Nhược điểm :

- + Mất thời gian thiết lập mạch=>thời gian đáp ứng nhanh
- + Hai thuê bao cần phải hoạt động trong cùng thời gian
- + Những nguồn cung cấp cũng phải ổn định và cung cấp qua mạng giữa 2 thuê bao.
- + Nếu có sự cố trên đường truyền dẫn đến gián đoạn đường truyền thì mạng không có khả năng định tuyến lại



Chương 1 : Tổng quan về mạng máy tính

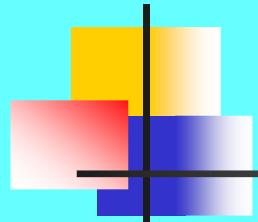
1.2.2.2 Chuyển mạch thông báo:

- + Hiện nay những bức điện báo, thư điện tử, files của máy tính được gọi là những thông báo, và nó được truyền qua mạng như sự trao đổi những dữ liệu số được trao đổi 2 chiều giữa các thuê bao.
- * Đặc điểm :
 - + Các nút mạng là các máy tính làm nhiệm vụ nhận bản tin, dịch bản tin, đóng dấu phát bản tin.
 - + Dữ liệu truyền dẫn qua mạng phải được tổ chức thành các bản tin trong đó ngoài phần dữ liệu thì phải kèm theo địa chỉ đích và một vài thông tin về định tuyến
 - + Với chuyển mạch thông báo không tồn tại sự thiết lập và cung cấp lộ trình cố định

Chương 1 : Tổng quan về mạng máy tính

1.2.2.2 Chuyển mạch thông báo:

- + Mỗi nút mạng chỉ hoàn thiện định tuyến và gởi bản tin đến nút kế tiếp. Một nút mạng muốn truyền một thông báo, nó sẽ gán địa chỉ của người nhận vào thông báo.
- + Ở tại mỗi Node thông báo được nhận, tạm giữ và truyền sang Node khác.
- + Thời gian trễ ở mỗi Node bao gồm cả thời gian nhận thông báo vào Node và thời gian sắp hàng chờ đến lượt mình để chuyển đến Node khác.



Chương 1 : Tổng quan về mạng máy tính

1.2.2.2 Chuyển mạch thông báo:

Ví dụ : Thông báo cần truyền từ A đến E

+ Thuê bao A gán địa chỉ của E vào thông báo và gửi nó vào Node 4. Node 4 gửi thông báo và tìm nhánh tiếp theo (ví dụ như 5) và nó sắp hàng thông báo để chờ truyền trên đường nối 4-5.

+ Khi đường nối cho phép, thông báo sẽ được gửi sang 5 và như thế nó được đưa đến 6 và E.

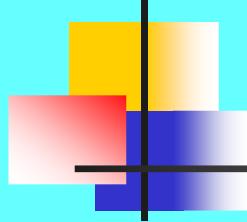
Hệ thống chuyển mạch thông báo là hệ thống luôn giữ và gửi tiếp các thông báo.

Chương 1 : Tổng quan về mạng máy tính

1.2.2.2 Chuyển mạch thông báo:

* Ưu điểm :

- + Không mất thời gian thiết lập mạch
- + Có khả năng định tuyến trở lại
- + Không cần đồng thời ổn định cho cả bên phát và thu, mạng có thể **tạm giữ thông báo** khi bộ phận thu chưa sẵn sàng.
- + Thông báo có thể đồng thời gửi đến nhiều thuê bao nhận, có thể sao chép thông báo để gửi đến người nhận cần thiết
- + Có thể kiểm tra sai và quản lý quá trình trong mạng thông báo.



Chương 1 : Tổng quan về mạng máy tính

1.2.2.2 Chuyển mạch thông báo:

* Nhược điểm:

- + Chưa đáp ứng được thời gian truyền dẫn khi lưu lượng hoạt động của mạng cao
- + Phải thuê đường trung kế, các kênh truyền dẫn => chi phí cao

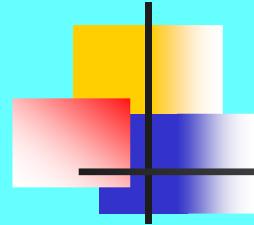
Chương 1 :

Tổng quan về mạng máy tính

1.2.2.3 Chuyển mạch gói :

- + Chuyển mạch gói gần giống như chuyển mạch thông báo.
- + Chỗ khác nhau cơ bản là độ dài của một khối dữ liệu đưa vào mạng được hạn chế thành từng gói. Độ dài điển hình mỗi gói từ 1000 cho đến hàng chục ngàn bit.
- + Trong PS thông báo đã được chia thành từng gói nhỏ. **Gói bao gồm dữ liệu cùng với địa chỉ và các thông số cần thiết.**

Chương 1 : Tổng quan về mạng máy tính



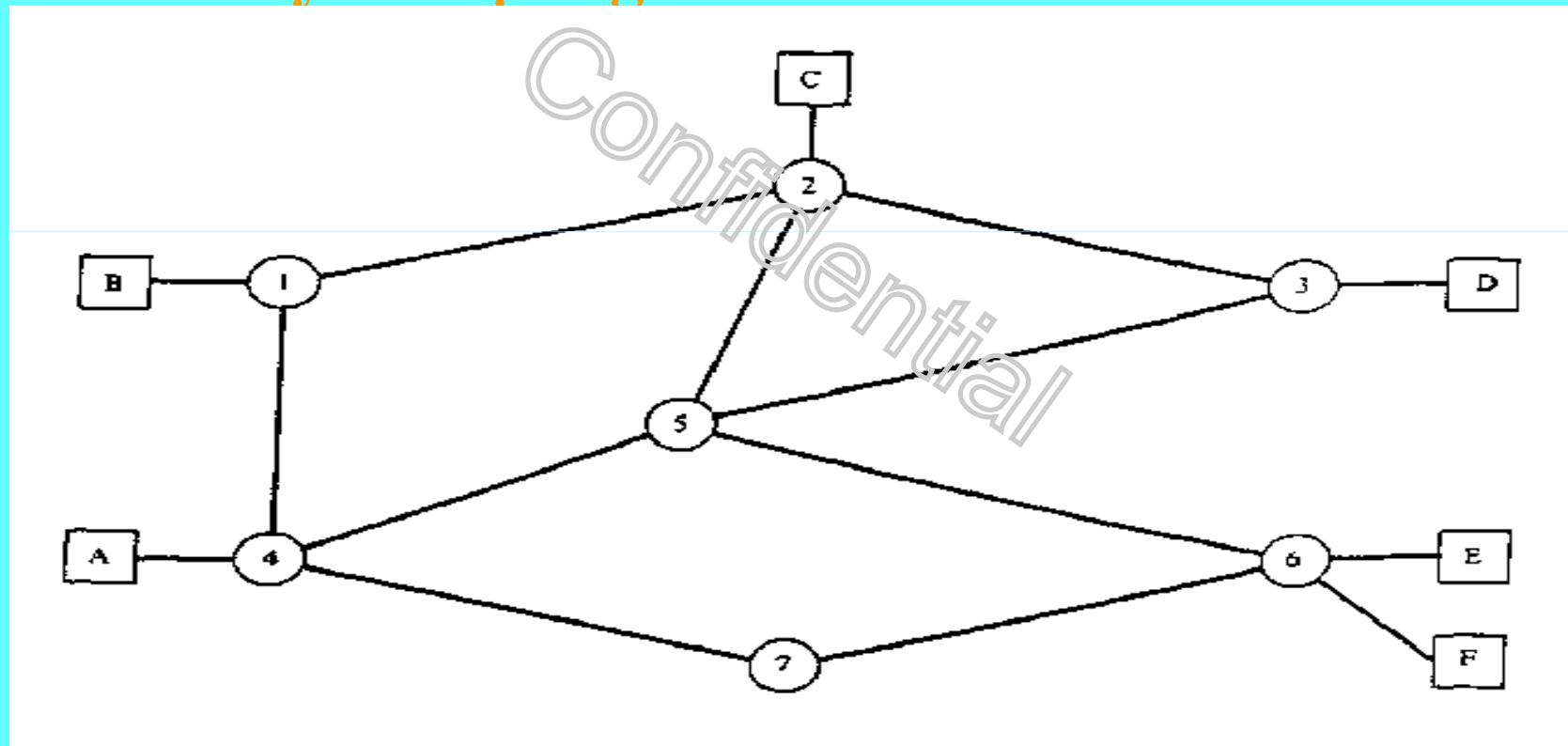
1.2.2.3 Chuyển mạch gói :

* Đặc điểm :

- + Không thiết lập mạch từ thuê bao này đến thuê bao khác
- + Mỗi nút mạng có kích thước gói tin khác nhau. Mỗi gói ngoài dữ liệu còn có thông tin về địa chỉ, định tuyến, và các thông tin được xác định theo nhiều giao thức khác nhau.
- + Một gói có thể có nhiều bản tin và đi nhiều đường khác nhau để đến cùng một đích.
- + Thừa kế các ưu điểm của mô hình chuyển mạch thông báo. Các nút mạng đều có khả năng sắp xếp lại bản tin và xử lý trên nguyên tắc lặp địa chỉ để loại trừ khả năng thất lạc gói.

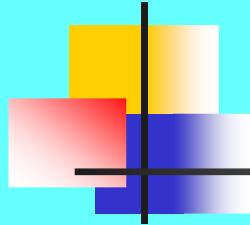
Chương 1 : Tổng quan về mạng máy tính

1.2.2.3 Chuyển mạch gói :



Mạng chuyển mạch gói.

Chương 1 : Tổng quan về mạng máy tính



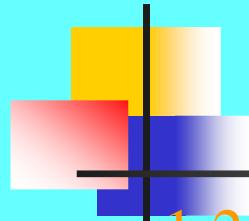
1.2.2.3 Chuyển mạch gói :

* Ưu điểm :

- + Cho phép truyền dẫn với lưu lượng lớn
- + Thời gian đáp ứng nhanh
- + Hiệu suất sử dụng cao nhất

Thông thường mạng chuyển mạch gói cung cấp hai dịch vụ truyền thông : *data-gram và mạch ảo*

Chương 1 : Tổng quan về mạng máy tính



1.2.2.3 Chuyển mạch gói :

a. Data-gram:

+ Đối với dịch vụ data-gram luồng thông tin được chia thành nhiều gói nhỏ gọi là data-gram. Trên mỗi packet đó phần header chứa hai thông tin cần thiết là **địa chỉ nguồn, địa chỉ đích, thứ tự gói**

+ Không có sự thiết lập đường liên lạc cố định giữa hai DTE mà các data-gram có thể được truyền trên nhiều hướng khác nhau. Trong quá trình truyền tin, gói thứ nhất có thể đến sau gói thứ hai, thứ ba,...

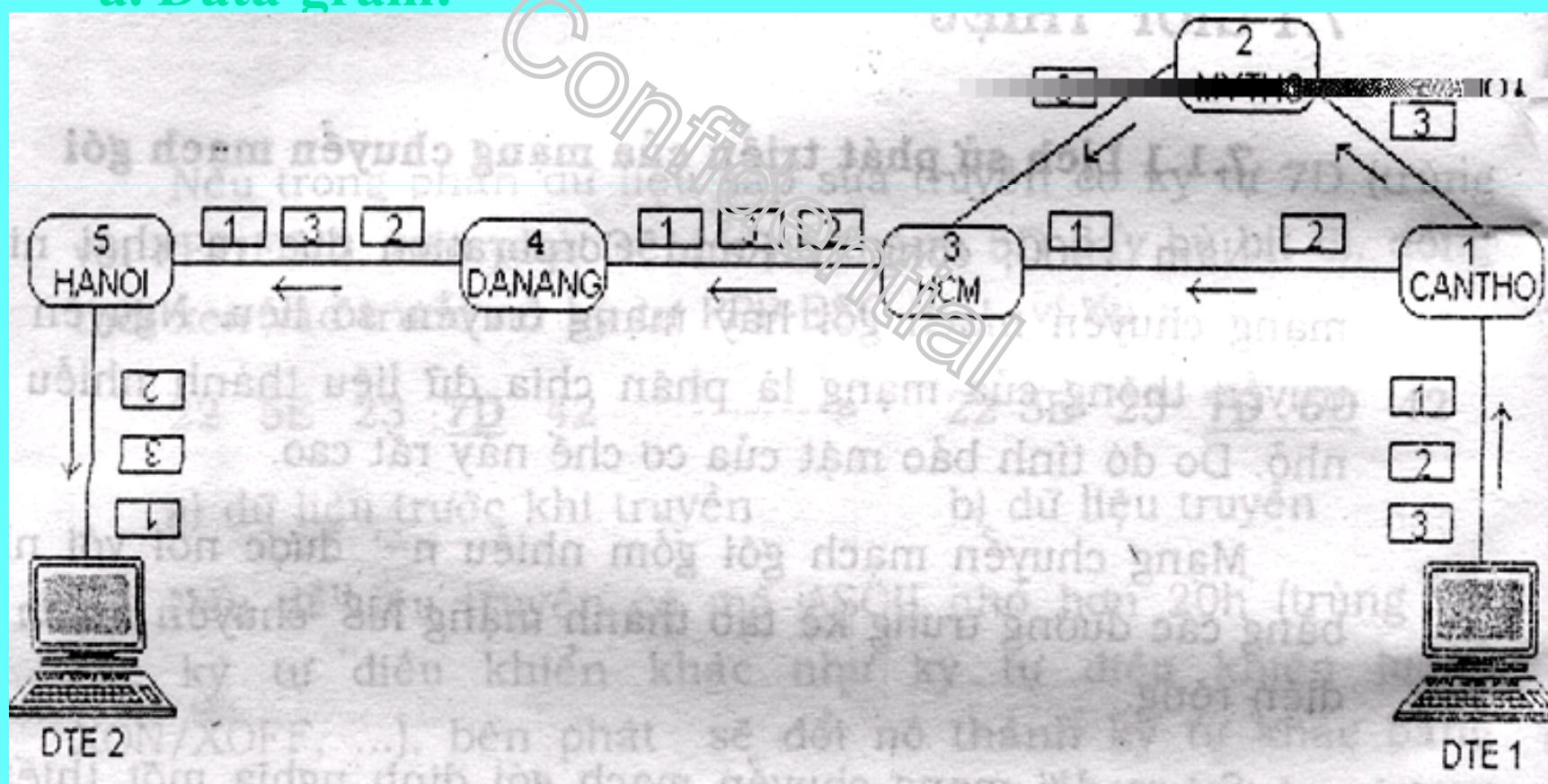
+ Khi bên nhận đã nhận đủ số lượng gói, bên nhận sẽ tái hợp các gói lại thành message ban đầu.

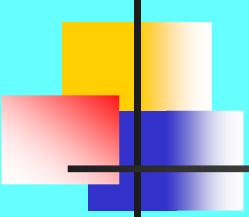
Chương 1 :

Tổng quan về mạng máy tính

1.2.2.3 Chuyển mạch gói :

a. Data-gram:





Chương 1 : Tổng quan về mạng máy tính

1.2.2.3 Chuyển mạch gói :

a. Data-gram:

Message được chia thành ba data-gram và được truyền đi trên hai đường khác nhau.

+ Packet 1 và 2 : DTE1-CANTHO-HCM-DANANG-HANOI-DTE2

+ Packet 3 : DTE1-CANTHO-MYTHO-HCM-DANANG-HANOI-DTE2

Kết quả nhận được là : packet 1-packet 3 –packet 2

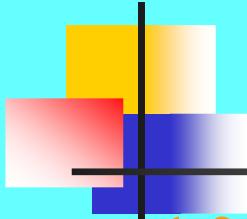
Chương 1 : Tổng quan về mạng máy tính

1.2.2.3 Chuyển mạch gói :

b. Mạch ảo :

- + Mỗi mạch ảo cung cấp một link giữa hai DTE trên mạng.
- + Tuỳ thuộc vào công nghệ chuyển mạch và tốc độ chuyển mạch của các PSE (tổng dài chuyển mạch gói)
- + Một kênh trung kế trong mạng truyền số liệu có thể cho phép thiết lập đến 4096 mạch ảo tại một thời điểm.

Chương 1 : Tổng quan về mạng máy tính



1.2.2.3 Chuyển mạch gói :

b. Mạch ảo :

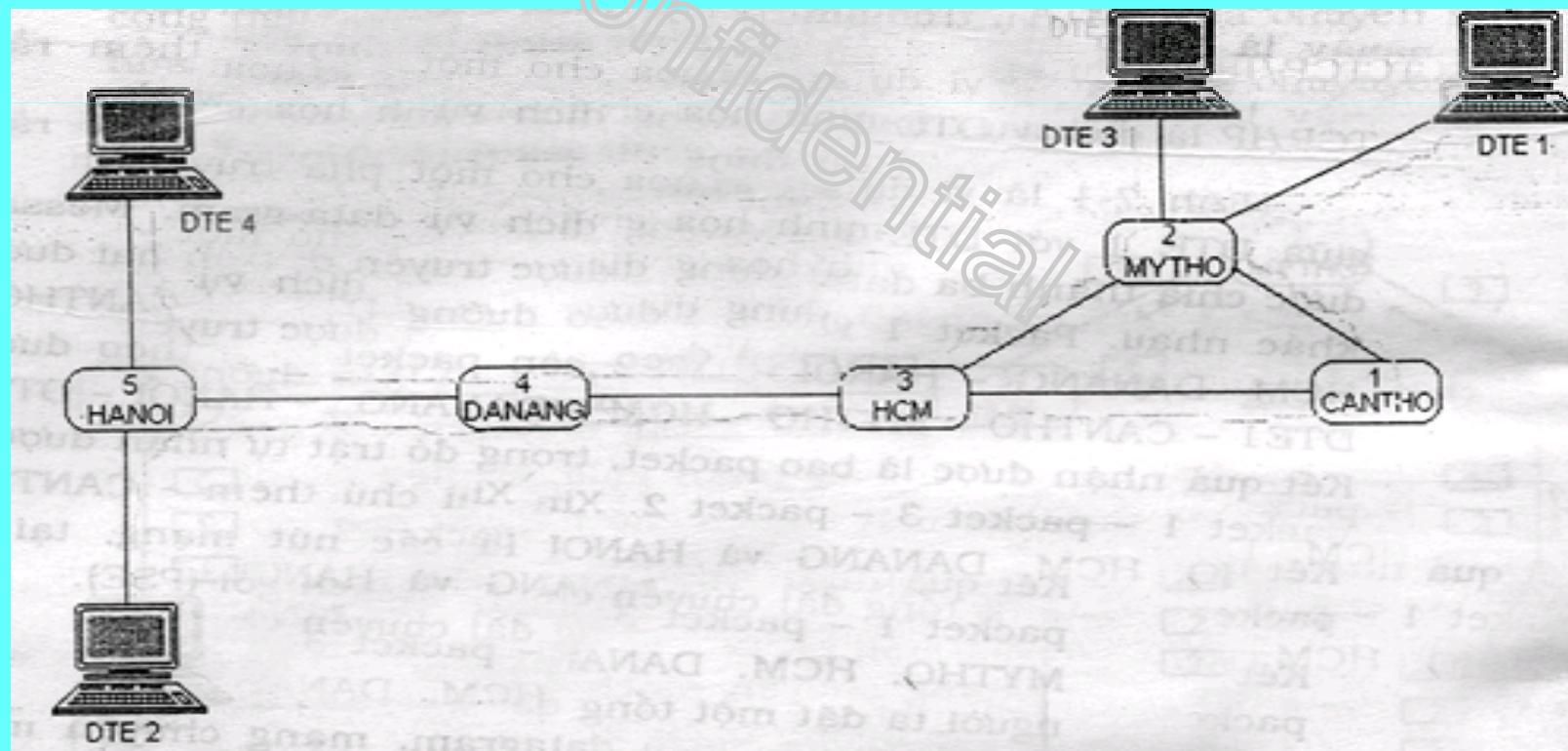
- + **Mạch ảo tạm thời** : không cố định về lộ trình, có nghĩa rằng mỗi lần kết nối ảo, các tổng đài trong mạng truyền số liệu tự chọn một lộ trình thích hợp trên cơ sở **chọn tuyến có mật độ thông lượng loãng nhất để thiết lập mạch ảo cho thuê bao**.
- + Những lần kết nối khác nhau có thể có những lộ trình khác nhau mặc dù hai DTE đó không thay đổi vị trí vật lý
- + **Mạch ảo thường xuyên** : được thiết lập một lần và cố định lộ trình. Mạch ảo này thường dùng cho các kết nối hoạt động suốt 24 giờ mỗi ngày. Có thể so sánh mạch ảo thường xuyên như kênh leased-line.

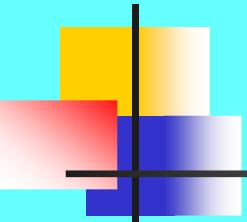
Chương 1 :

Tổng quan về mạng máy tính

1.2.2.3 Chuyển mạch gói :

b. Mạch ảo :





Chương 1 : Tổng quan về mạng máy tính

1.2.2.3 Chuyển mạch gói :

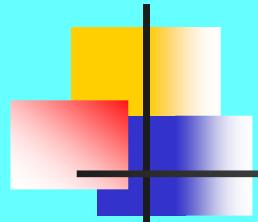
b. Mạch ảo :

+ Mạch ảo thứ nhất được thiết lập cho kết nối giữa DTE1 VÀ DTE2 :

DTE1-MYTHO-HCM-DANANG-HANOI-DTE2

+ Mạch ảo thứ hai được thiết lập cho kết nối giữa DTE3 VÀ DTE4 :

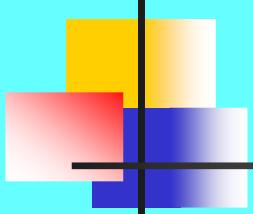
DTE3-MYTHO-CANTHO-HCM-DANANG-HANOI-DTE4



Chương 1 : Tổng quan về mạng máy tính

1.2.2.3 Chuyển mạch gói :

Tóm lại : Đối với mạng chuyển mạch gói thì không thể dựa hoàn toàn vào khoảng cách giữa hai đầu truyền tin, vì cơ chế **chuyển mạch ảo** và **datagram tự chọn** **lấy lộ trình**, có khi phải qua vòng hàng trăm km trong khi hai DTE chỉ cách nhau vài chục km.

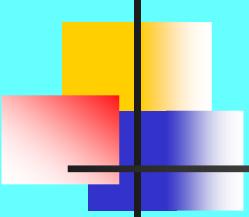


Chương 1 : Tổng quan về mạng máy tính

1.2 Đặc trưng kỹ thuật của mạng máy tính

1.2.3. Kiến trúc mạng

- Kiến trúc mạng máy tính thể hiện cách nối các máy tính với nhau và tập hợp các quy tắc, quy ước mà tất cả các thiết bị tham gia truyền thông trên mạng phải tuân theo để đảm bảo cho mạng hoạt động tốt.
- Khi nói đến kiến trúc của mạng thì có hai vấn đề :
 - hình trạng mạng (Network topology)
 - giao thức mạng (Network protocol)



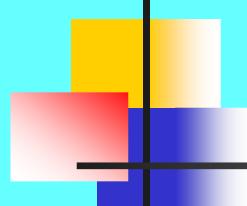
Chương 1 : Tổng quan về mạng máy tính

1.2 Đặc trưng kỹ thuật của mạng máy tính

1.2.4. Hệ điều hành mạng

Hệ điều hành mạng là một phần mềm hệ thống có các chức năng sau:

- Quản lý tài nguyên của hệ thống
- Quản lý người dùng và các công việc trên hệ thống. Hệ điều hành đảm bảo giao tiếp giữa người sử dụng, chương trình ứng dụng với thiết bị của hệ thống.
- Cung cấp các tiện ích cho việc khai thác hệ thống thuận lợi (Format đĩa, sao chép tệp và thư mục, in ấn, ...)



Chương 1 : Tổng quan về mạng máy tính

1.3 Phân loại mạng máy tính

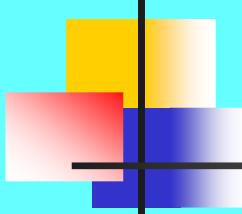
1.3.1. Phân loại ~~mạng~~ theo khoảng cách địa lý

Nếu lấy ~~khoảng~~ cách địa lý thì ta có mạng cục bộ (LAN), mạng đô thị (MAN), mạng diện rộng (WAN), mạng toàn cầu (INTERNET).

1.3.2. Phân loại theo ~~kiến trúc~~ mạng sử dụng

Kiến trúc của mạng bao gồm hai vấn đề: hình trạng mạng (Network topology) và giao thức mạng (Network protocol)

- Khi phân loại theo topo mạng người ta thường có phân loại thành: mạng hình sao, tròn, tuyễn tính
- Phân loại theo giao thức mà mạng sử dụng người ta phân loại thành mạng : TCP/IP, mạng NETBIOS . . .



Chương 1 : Tổng quan về mạng máy tính

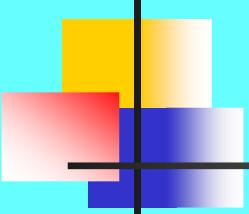
1.3 Phân loại mạng máy tính

1.3.2. Phân loại theo kiến trúc mạng sử dụng

- **Mạng hình sao**: Mạng hình sao có tất cả các trạm được kết nối với một thiết bị trung tâm có nhiệm vụ nhận tín hiệu từ các trạm và chuyển đến trạm đích. Độ dài đường truyền nối một trạm với thiết bị trung tâm bị hạn chế

- **Mạng trực tuyến tính (Bus)**:

Trong mạng trục tất cả các trạm phân chia một đường truyền chung (bus). Đường truyền chính được giới hạn hai đầu bằng hai đầu nối đặc biệt gọi là terminator. Mỗi trạm được nối với trục chính qua một đầu nối chữ T (Tconnector) hoặc một thiết bị thu phát (transceiver).



Chương 1 : Tổng quan về mạng máy tính

1.3 Phân loại mạng máy tính

1.3.2. Phân loại theo kiến trúc mạng sử dụng

- Mạng hình vòng

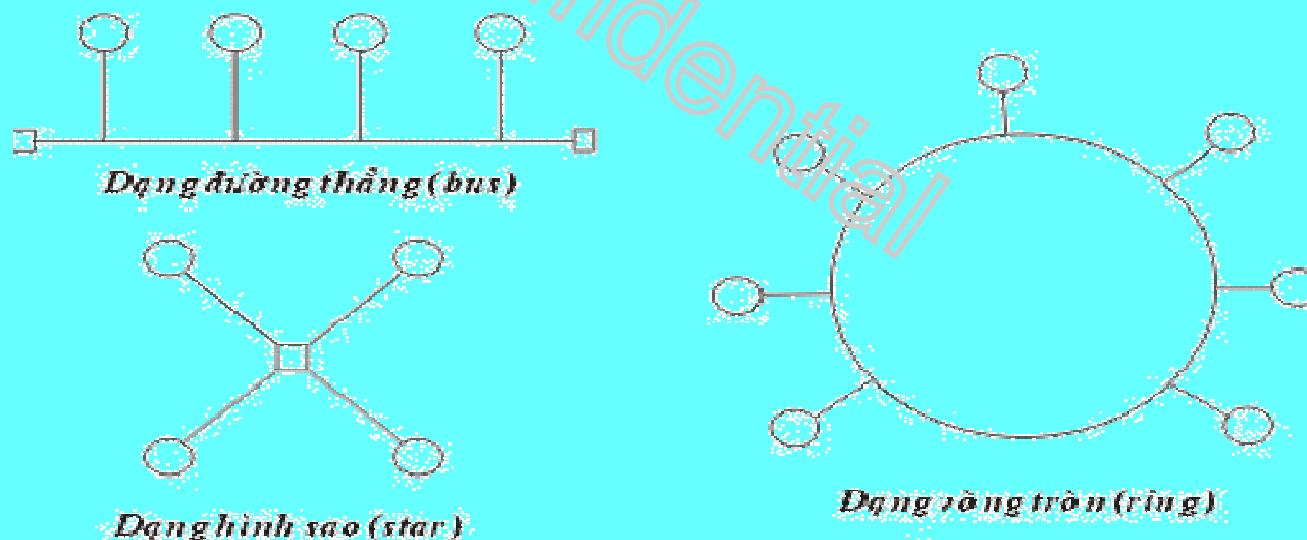
Trên mạng hình vòng tín hiệu được truyền đi trên vòng theo một chiều duy nhất. Mỗi trạm của mạng được nối với vòng qua một bộ chuyển tiếp (repeater) do đó cần có giao thức điều khiển việc cấp phát quyền được truyền dữ liệu trên vòng mạng cho trạm có nhu cầu.

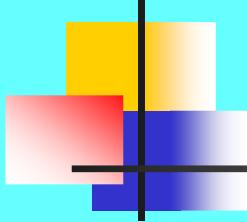
Mạng hình vòng có ưu nhược điểm tương tự mạng hình sao, tuy nhiên mạng hình vòng đòi hỏi giao thức truy nhập mạng phức tạp hơn mạng hình sao

Chương 1 : Tổng quan về mạng máy tính

1.3 Phân loại mạng máy tính

1.3.2. Phân loại theo kiến trúc mạng sử dụng





Chương 1 : Tổng quan về mạng máy tính

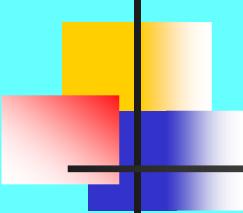
1.3 Phân loại mạng máy tính

1.3.3. Phân loại theo kỹ thuật chuyển mạch

- + Mạng chuyển mạch kênh
- + Mạng chuyển mạch thông báo
- + Mạng chuyển mạch gói.

1.3.4. Phân loại theo hệ điều hành mạng

Nếu phân loại theo hệ điều hành mạng người ta chia ra theo mô hình mạng ngang hàng, mạng khách/chủ hoặc phân loại theo tên hệ điều hành mà mạng sử dụng: Windows, Unix, Novell . . .



Chương 1 : Tổng quan về mạng máy tính

1.4. Chuẩn hóa mạng máy tính

- Khi thiết kế các giao thức mạng, các nhà thiết kế tự do lựa chọn kiến trúc cho riêng mình. Từ đó dẫn tới tình trạng không tương thích giữa các mạng máy tính với nhau. Vấn đề không tương thích đó làm trở ngại cho sự tương tác giữa những giao thức mạng khác nhau.
- Nhu cầu trao đổi thông tin càng lớn thúc đẩy việc xây dựng khung chuẩn về kiến trúc mạng để làm căn cứ cho các nhà thiết kế và chế tạo thiết bị mạng .
- Chính vì lý do đó, tổ chức tiêu chuẩn hóa quốc tế ISO (International Organization for Standardization) đã xây dựng mô hình tham chiếu cho việc kết nối các hệ thống mở OSI (reference model for Open Systems Interconnection).

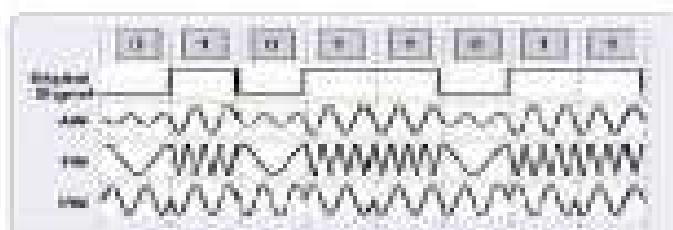
Mô hình tham chiếu OSI

7 tầng của mô hình tham chiếu OSI



Tầng vật lý

- Kiểm soát mức thấp nhất việc truyền thông giữa 2 nút mạng.
- Card mạng và cáp mạng. Truyền dãy các bit giữa 2 nút.
- Các nhà lập trình mạng không làm việc ở mức này.
 - ✓ Trách nhiệm của các nhà phát triển driver phần cứng và các kỹ sư điện.
- Lỗi có thể xảy ra trong quá trình truyền dữ liệu ở tầng này do điện áp, hay nhiễu đường truyền trên mạng.



Sample section of a signal transmitted on copper cable

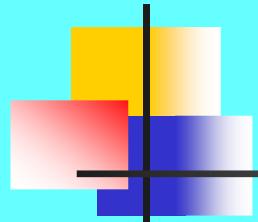
Physical layer signal

Microwave (radio wave) signal

Tầng liên kết dữ liệu

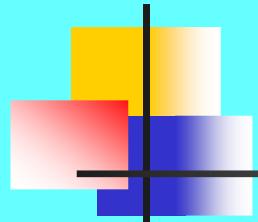
- Chịu trách nhiệm truyền dữ liệu tin cậy hơn
- Nhóm dữ liệu thành các frames.
 - Frames tương tự như các packet dữ liệu, nhưng chúng là các khối dữ liệu, được đặc tả theo kiến trúc phần cứng (trong khi đó packet được dùng ở tầng cao hơn và có thể di chuyển từ kiểu mạng này sang kiểu mạng khác).
- Frames có trường kiểm tra lỗi truyền (checksums, TTL...)
- Đảm bảo dữ liệu bị méo không được truyền lên tầng trên.





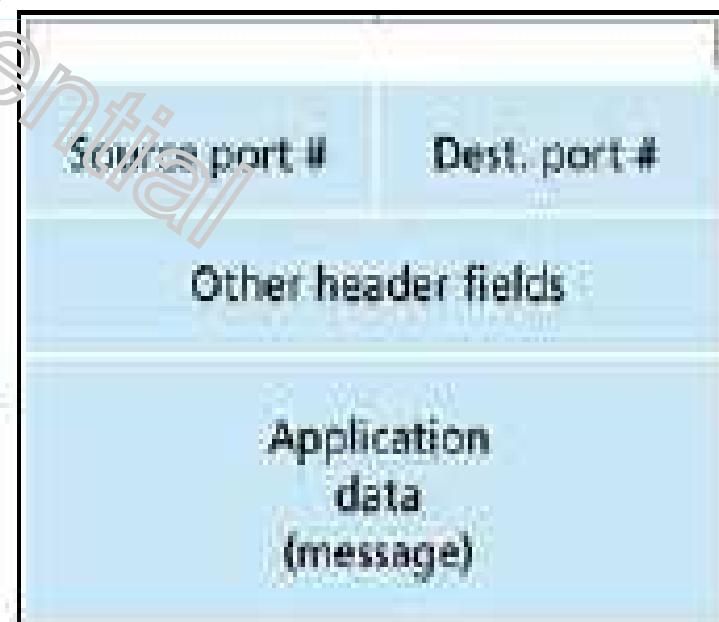
Tầng mạng

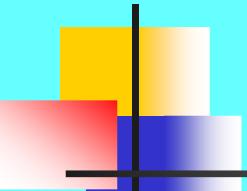
- Các frames truyền từ tầng Datalink lên hoặc các segments từ tầng transport xuống.
- Dữ liệu dạng các packets
- Phần header chứa các thông tin quan trọng:
 - ✓ Địa chỉ mạng (network address) và định tuyến mạng (routing).
- Packets được gửi qua lại giữa các mạng.
- Các packets thường được định tuyến khác nhau;
 - ✓ việc định tuyến được thực hiện nhờ các routers.
- Các lập trình mạng cũng hiếm được yêu cầu lập trình các dịch vụ phần mềm cho tầng này.



Tăng vận chuyển

- Liên quan đến việc dữ liệu được truyền như thế nào
- Dữ liệu dạng segments
- Chịu trách nhiệm:
 - ✓ Xử lý việc kết nối,
 - ✓ Phát hiện lỗi một cách tự động,
 - ✓ Điều khiển luồng dữ liệu



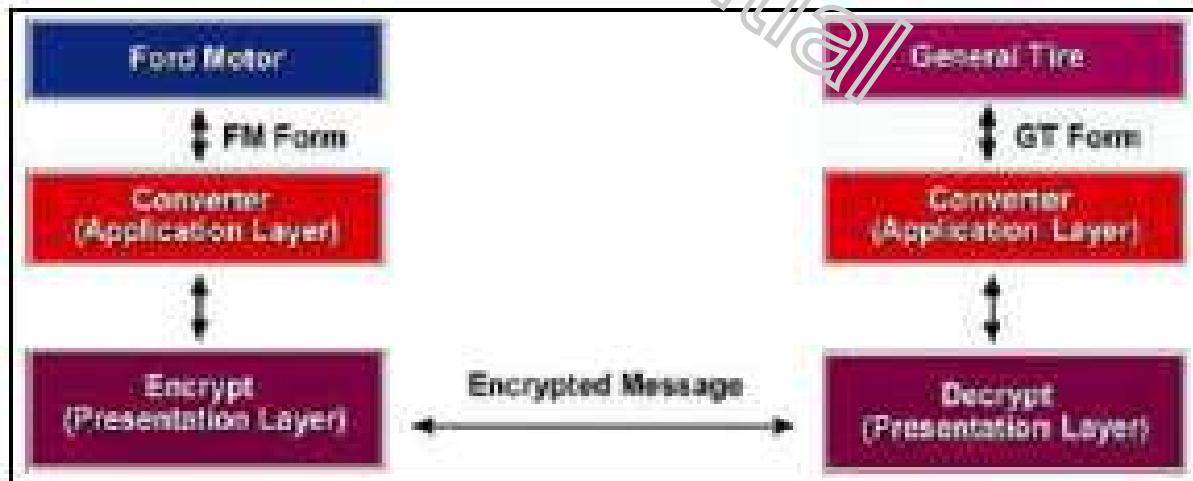


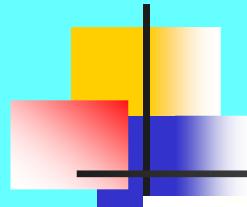
Tầng phiên

- Làm cho dễ dàng việc trao đổi dữ liệu,
- Quản lý phiên truyền thông giữa các ứng dụng.
 - ✓ Thiết lập một phiên,
 - ✓ đồng bộ một phiên,
 - ✓ thiết lập lại phiên nếu một phiên bị kết thúc đột ngột.
- Không phải tắt cả các ứng dụng đều sử dụng giao thức có kết nối
 - ✓ Do vậy việc quản lý phiên không phải lúc nào cũng được yêu cầu.

Tầng trình bày

- Nhiệm vụ đảm bảo hiển thị và chuyển đổi dữ liệu.
 - ✓ Các máy tính khác nhau có thể sử dụng các kiểu biểu diễn dữ liệu khác nhau (ví dụ một số nguyên có thể 8 bit hoặc 16 bit).
 - ✓ Một vài giao thức muốn nén hoặc mã hóa dữ liệu.

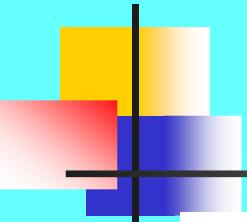




Tầng ứng dụng

- Tầng cao nhất trong mô hình mạng.
- Hầu hết các ứng dụng mạng được viết ở tầng này.

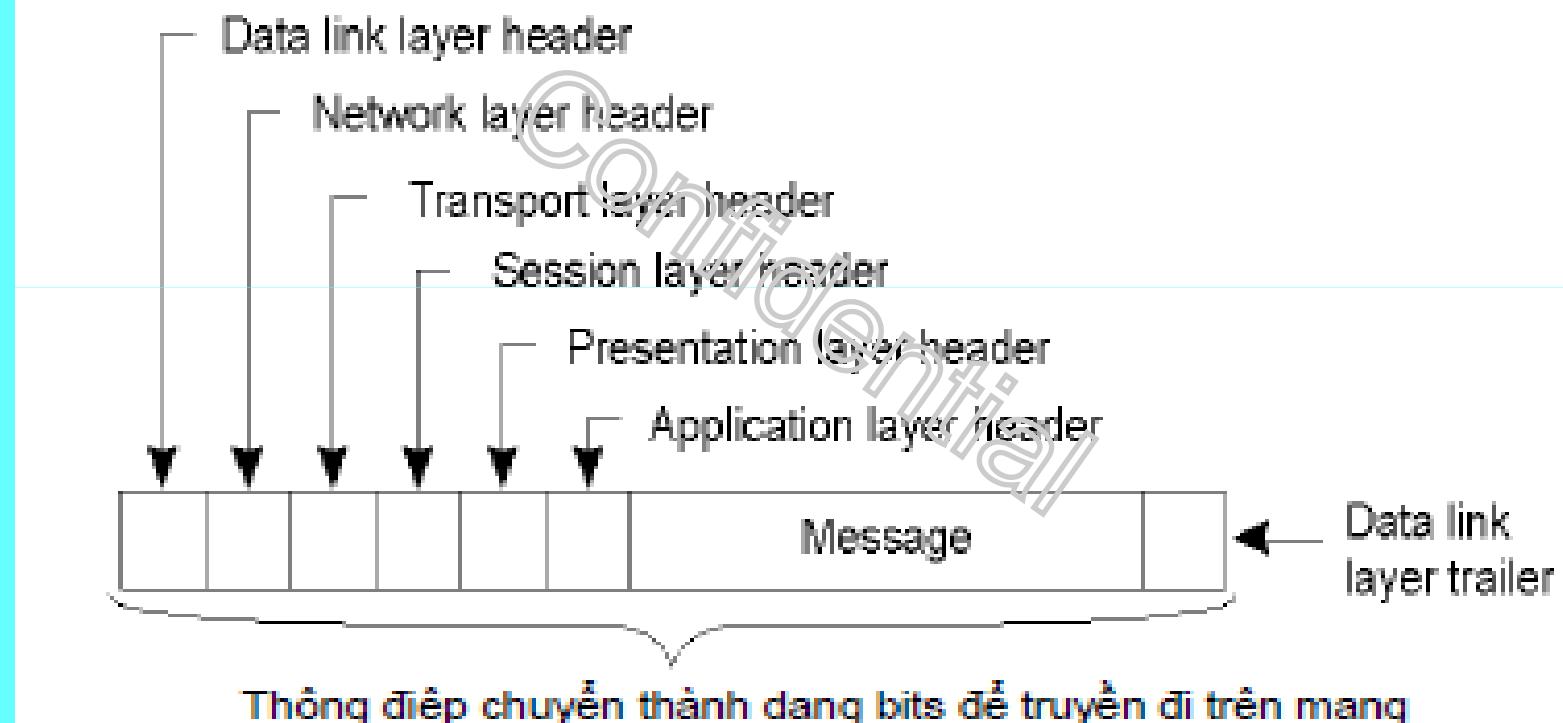




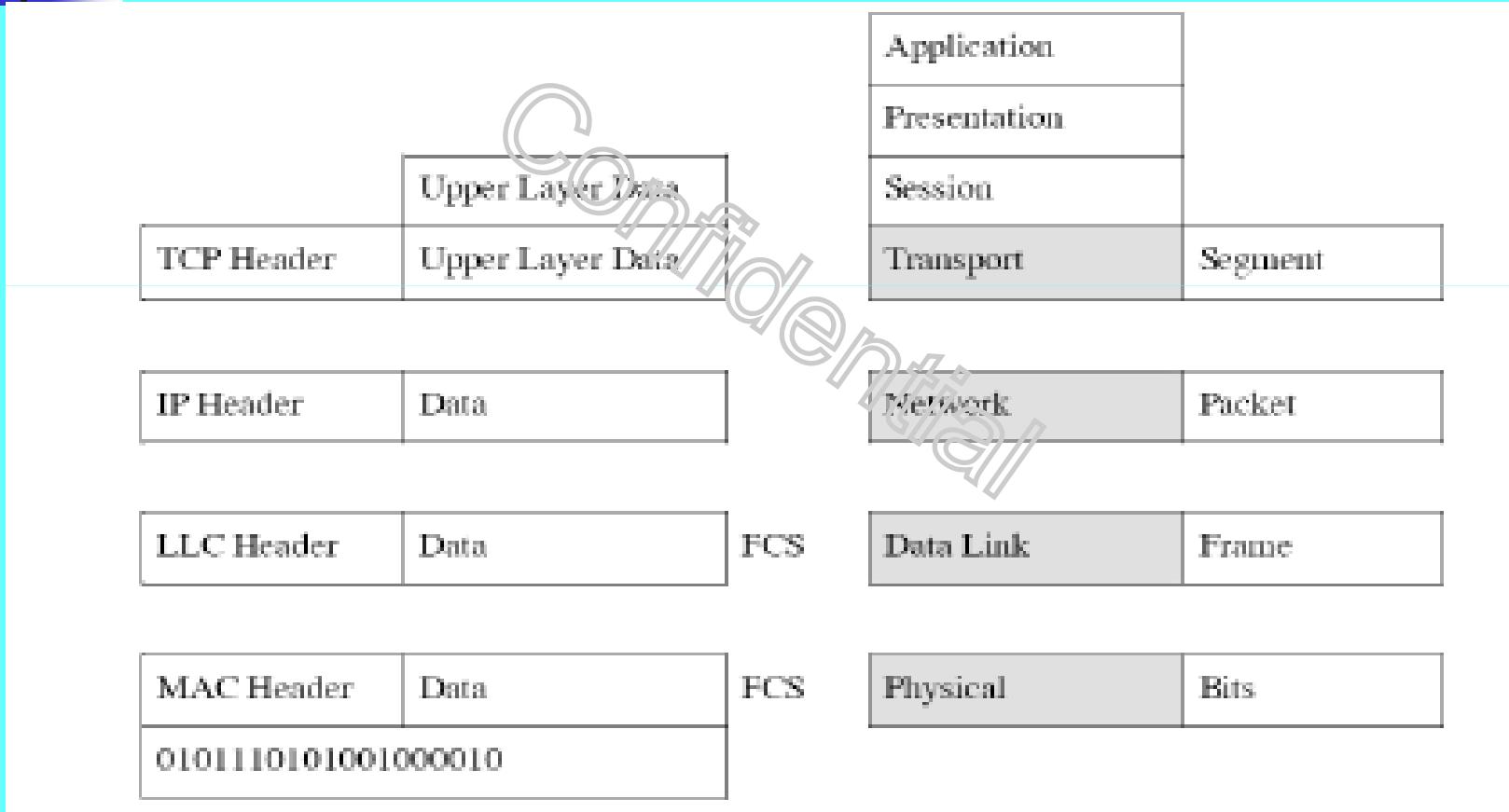
Các giao thức các tầng

- Application
 - HTTP, RTP, SMTP, NSF, Telnet, SSH, ECHO, ...
- Presentation
 - SMB, NCR, ...
- Session
 - SSH, NetBIOS, RPC, ...
- Transport
 - TCP, UDP, ...
- Network
 - IP, ICMP, IPX, ...
- Data link
 - Ethernet, Token Ring, ISDN, ...
- Physical
 - 100BASE-T, 1000BASE-T, 802.11, ...

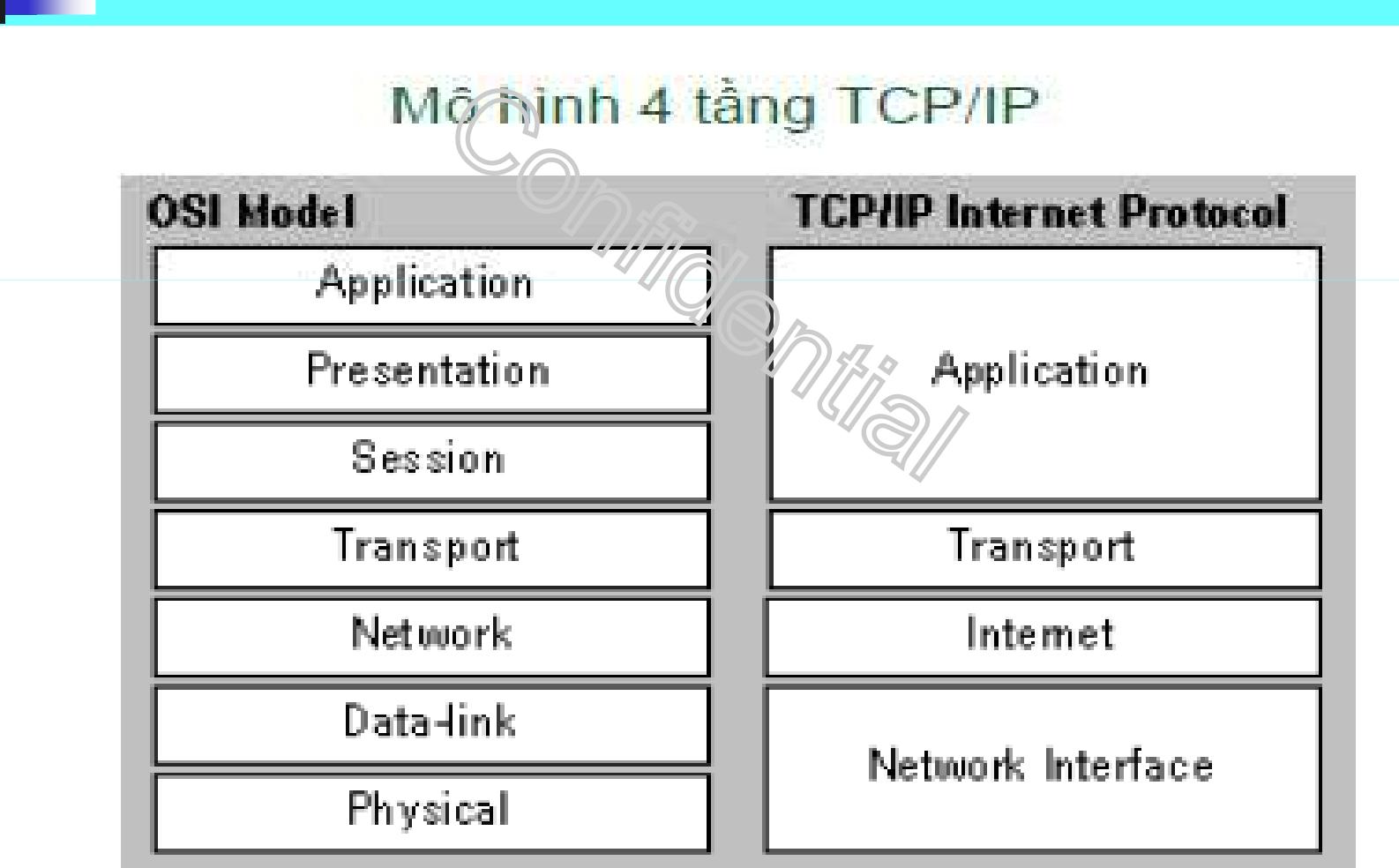
Metadata trong một thông điệp

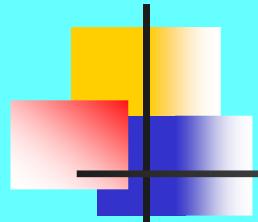


Đóng gói dữ liệu



Mối quan hệ giữa OSI và TCP





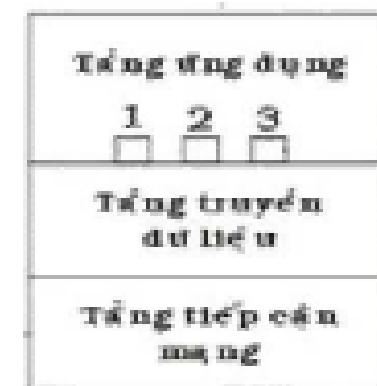
Mô hình phân tầng thu gọn

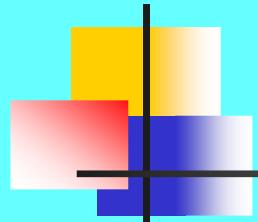
- Một số mô hình được phát triển

- Mô hình 7 tầng OSI
 - Mô hình 4 tầng TCP/IP

- Xét trên phương diện lập trình

- Mô hình truyền thông đơn giản gồm 3 tầng.
 - ✓ Tầng ứng dụng,
 - ✓ Tầng giao vận
 - ✓ Tầng tiếp cận mạng

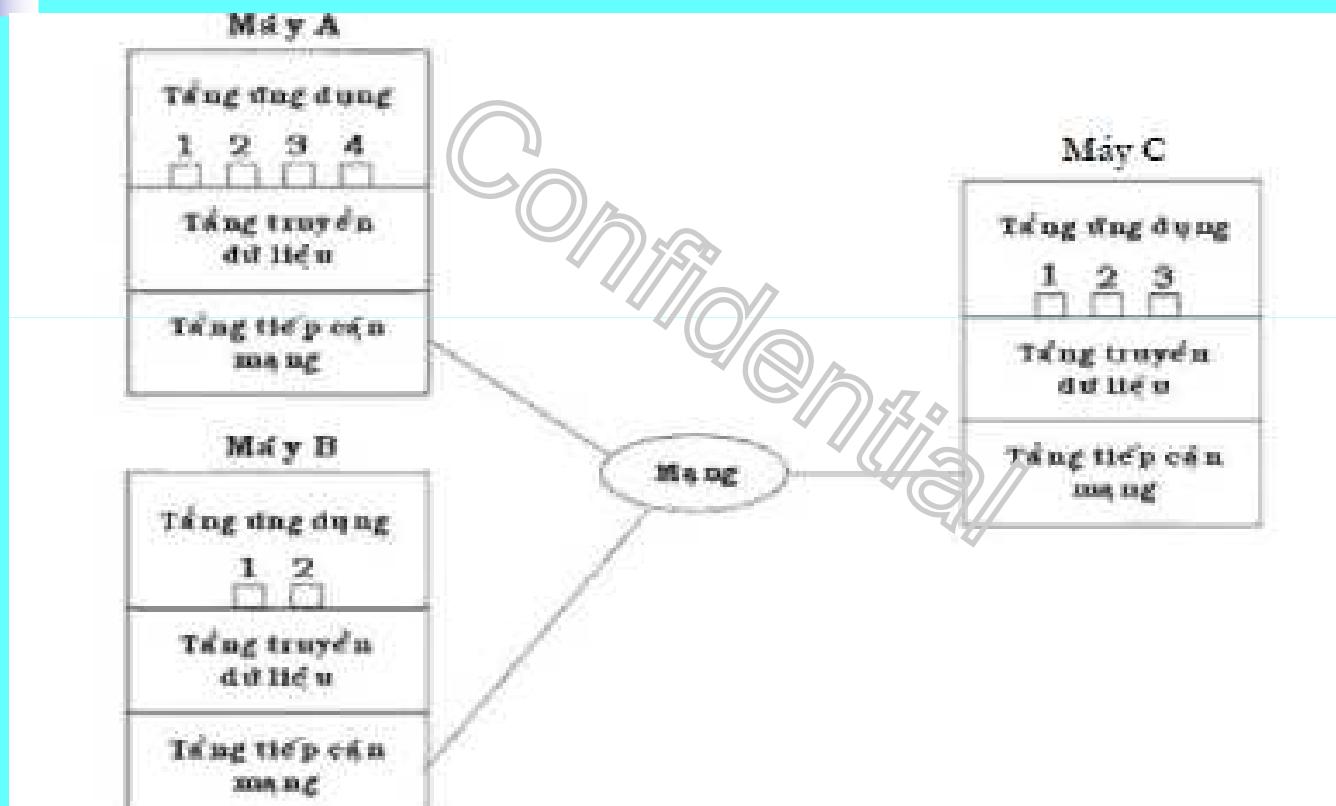


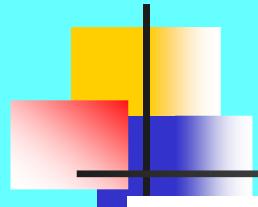


Mô hình phân tầng thu gọn

- Các thành phần tham gia trong quá trình truyền thông
 - Các chương trình ứng dụng,
 - Các chương trình truyền thông,
 - Các máy tính và các mạng
- Gửi dữ liệu giữa các ứng dụng
 - Máy tính gửi:
 - ✓ Ứng dụng gửi chuyển dữ liệu cho chương trình truyền thông
 - ✓ Chương trình truyền thông sẽ gửi dữ liệu cho máy tính nhận.
 - Máy tính nhận:
 - ✓ Chương trình truyền thông sẽ tiếp nhận và kiểm tra dữ liệu trước
 - ✓ Sau đó chuyển cho ứng dụng đang chờ nhận dữ liệu.

Ví dụ mô hình truyền thông





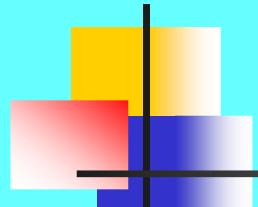
Ví dụ mô hình truyền thông (tt)

■ Máy A:

- Ứng dụng 1 cần gửi một khối dữ liệu
- Dữ liệu được chuyển cho tầng giao vận
 - ✓ chia dl thành nhiều đoạn và đóng thành các gói tin (packets)
 - ✓ bổ sung thêm các thông tin điều khiển (header) vào mỗi gói tin.
- Dữ liệu tiếp tục được chuyển cho tầng tiếp cận mạng và chuyển cho máy B.

■ Máy B:

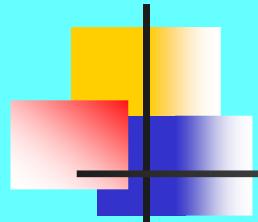
- Tầng tiếp cận mạng sẽ tập hợp dữ liệu và chuyển cho tầng giao vận.
 - ✓ kiểm tra và ghép dl lại thành khối (nhờ tt header).
 - ✓ Khối dữ liệu sẽ được chuyển lên cho tầng ứng dụng.



Nguyên tắc truyền thông

■ Một mạng máy tính trở thành một môi trường truyền dữ liệu cần có các yếu tố sau:

- Các máy tính phải được kết nối nhau theo một cấu trúc kết nối (topology) nào đó.
- Việc chuyển dữ liệu thực hiện dưới những qui định thống nhất gọi là giao thức mạng (protocol).
- Phân chia hoạt động truyền thông của hệ thống thành nhiều lớp theo các nguyên tắc nhất định.



Nguyên tắc truyền thông

Mô hình truyền thông trong kiến trúc mạng (tt)

■ Phương pháp phân tầng mạng

- Tách và xét mô hình mạng thành các môđun độc lập:

- ✓ giảm độ phức tạp cho việc thiết kế và cài đặt.

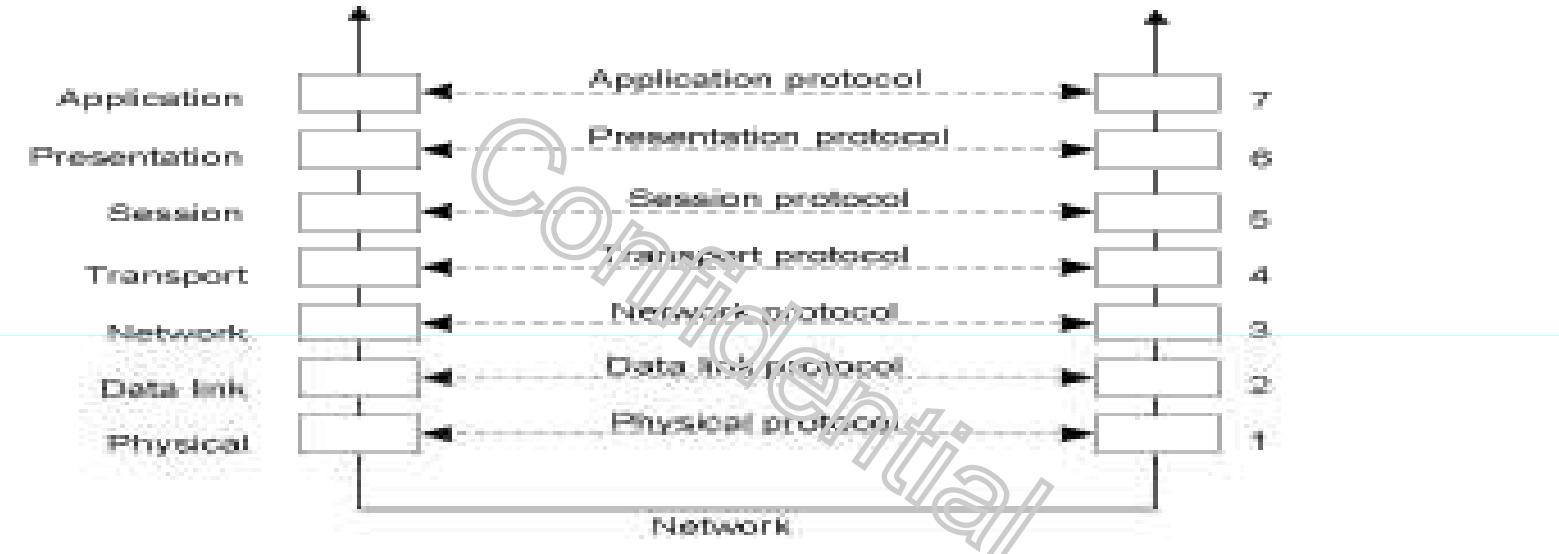
■ Nguyên tắc

- Mỗi hệ thống được xây dựng như một cấu trúc nhiều tầng

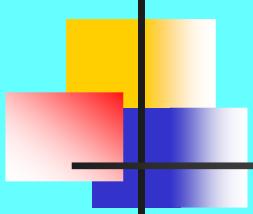
- ✓ và có cấu trúc giống nhau:

- ✓ số lượng tầng và chức năng của các tầng

Nguyên tắc truyền thông



- Dữ liệu chỉ được truyền giữa 2 tầng kề nhau.
- Bên gửi: Dữ liệu từ tầng cao nhất lân lượt đến tầng thấp nhất.
- Bên nhận: Dữ liệu từ tầng thấp nhất ngược lên đến tầng cao nhất.



Chương 2 : Các thiết bị và các chuẩn kết nối

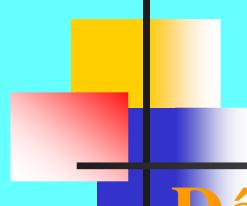
2.1 Các môi trường truyền

2.1.1 Môi trường định hướng

- a. Dây song hành
- b. Cáp xoắn đôi
- c. Cáp đồng trục
- d. Cáp quang

2.1.2 Môi trường không định hướng

- a. Viba mặt đất
- b. Viba vệ tinh
- c. Sóng radio
- d. Vô tuyến tể bào



MÔI TRƯỜNG TRUYỀN

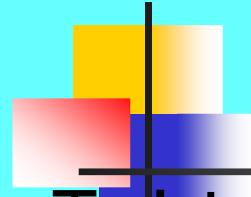
Dây song hành (Twisted Pair Cable)

* Cấu tạo :

- + Gồm có 2 sợi đặt song hành. Cặp dây đó là đường liên lạc đơn.
- + Nhiều cặp dây như vậy được đặt chung trong một cáp có vỏ bọc. Những cáp dài có thể chứa hàng trăm cặp.
- + Các cặp dây được cách ly để tránh ảnh hưởng điện từ với nhau.
- + Lõi dây thường từ 0,016 - 0,036 inches.

CÁP SONG HÀNH





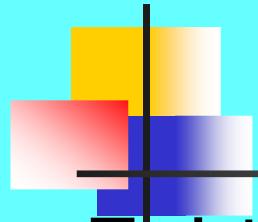
MÔI TRƯỜNG TRUYỀN

Twisted pair cable :

- + UTP (unshielded twisted pair)
- + SFTP (shielded twisted pair)



UTP Category 5 cable : 100 million bits per second.



MÔI TRƯỜNG TRUYỀN

Twisted pair cable :

- + UTP (unshielded twisted pair)
- + STP (shielded twisted pair)

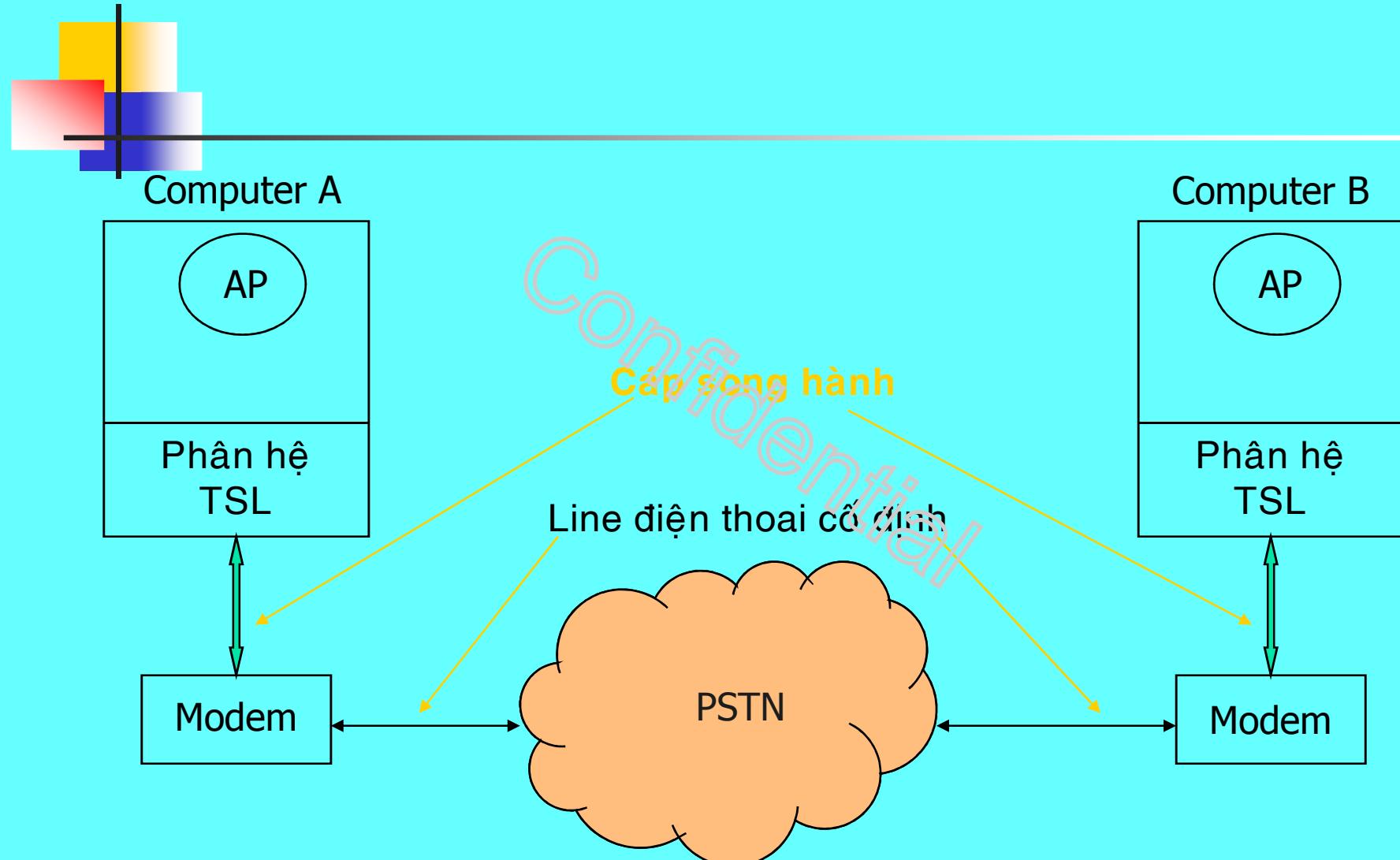


STP Category 3 cable : 10 million bits per second.

ĐẶC ĐIỂM CƠ BẢN CỦA CÁP SONG HÀNH

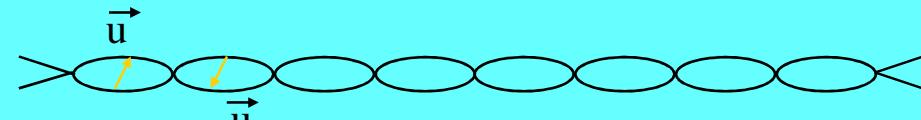
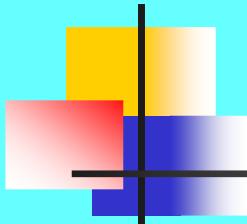
- Là môi trường truyền dẫn đơn giản nhất và có chất lượng kém nhất, lý do:
 - Không chống được nhiễu từ bên ngoài.
 - Ảnh hưởng lớn của nhiễu xuyên âm
- Khoảng cách truyền khoảng 50 m.
- Tốc độ bít khoảng 19,2 Kbit/s.
- Ví dụ sử dụng cáp song hành: kết nối modem (DCE = Data circuit equipment) với máy tính (DTE = Data terminal equipment) khi truyền số liệu qua mạng PSTN.

ỨNG DỤNG CÁP SONG HÀNH



Liên kết qua mạng PSTN sử dụng Modem

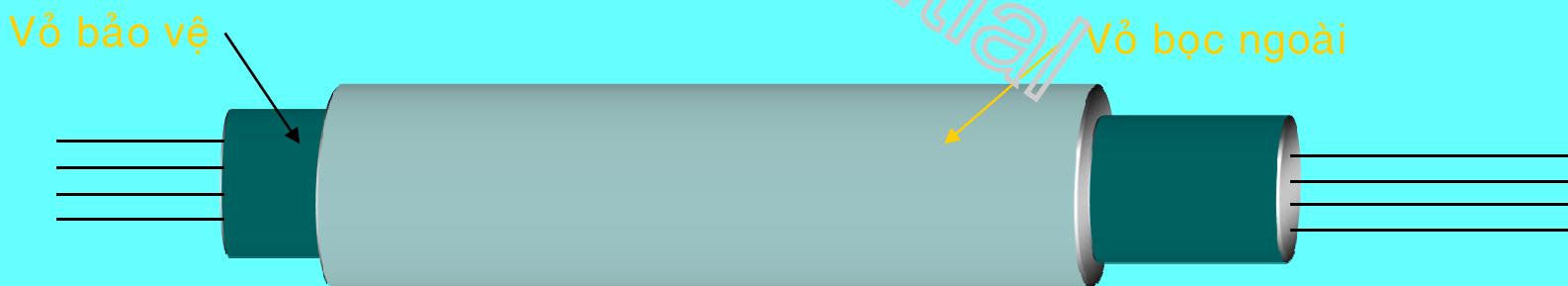
CÁP XOĂN ĐÔI



a) Cáp cáp xoắn đôi



b) Sợi cáp xoắn nhiều đôi – loại không vỏ bảo vệ (UTP)

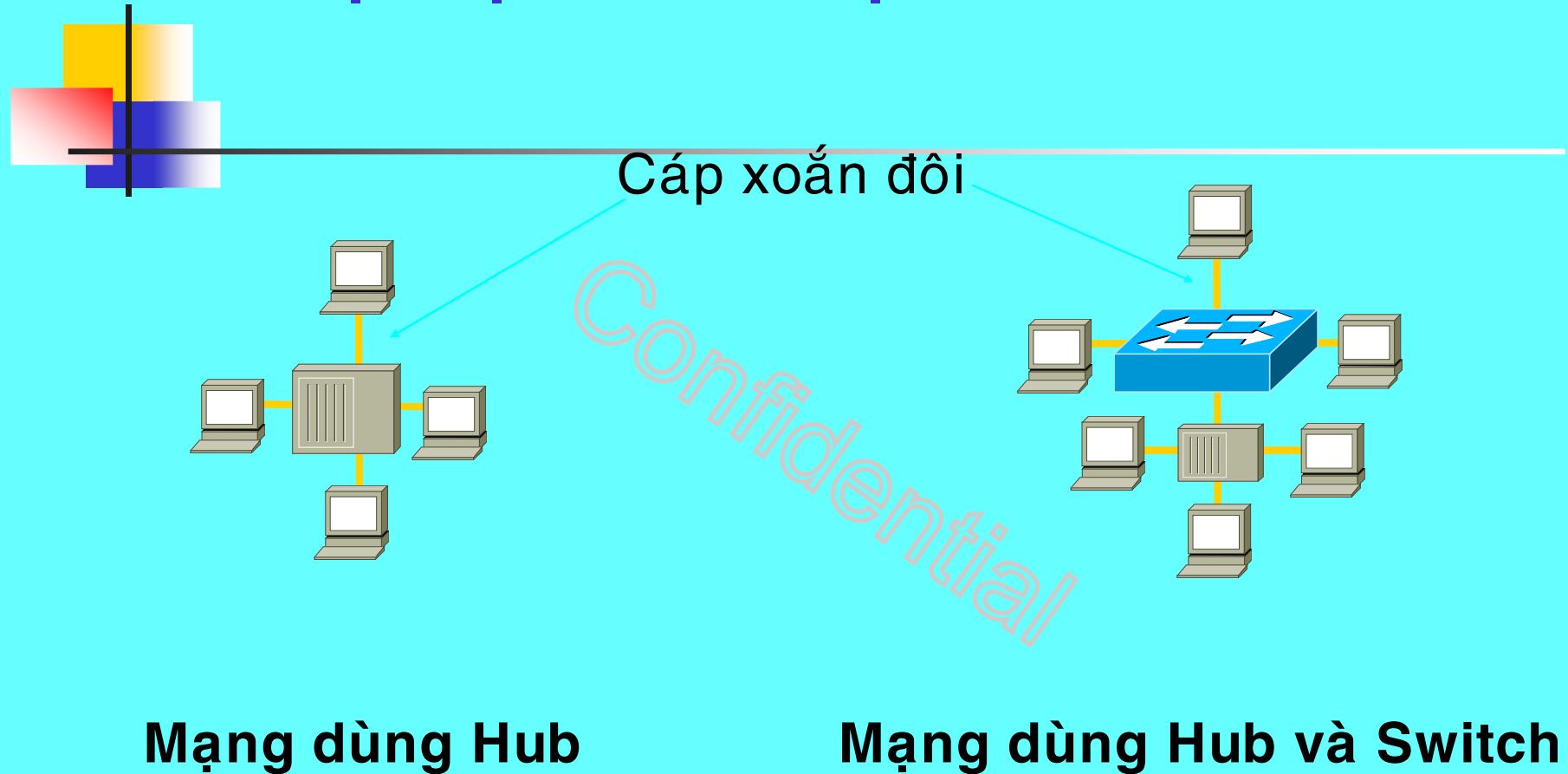


c) Sợi cáp xoắn nhiều đôi – loại có vỏ bảo vệ (STP)

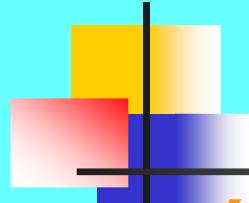
ĐẶC ĐIỂM CƠ BẢN CỦA CÁP XOẮN ĐÔI

- Là môi trường truyền dẫn có chất lượng tốt hơn cáp song hành, lý do:
 - Chống được nhiễu xuyên âm
 - Cáp STP hạn chế được nhiễu từ bên ngoài
- Tốc độ bít khoảng 10Mbit/s với khoảng cách 100m và có thể tăng lên khi khoảng cách giảm xuống.
- Giá thành không cao lắm, dễ thi công.

VÍ DỤ MẠNG SỬ DỤNG CÁP XOĂN



MÔI TRƯỜNG TRUYỀN



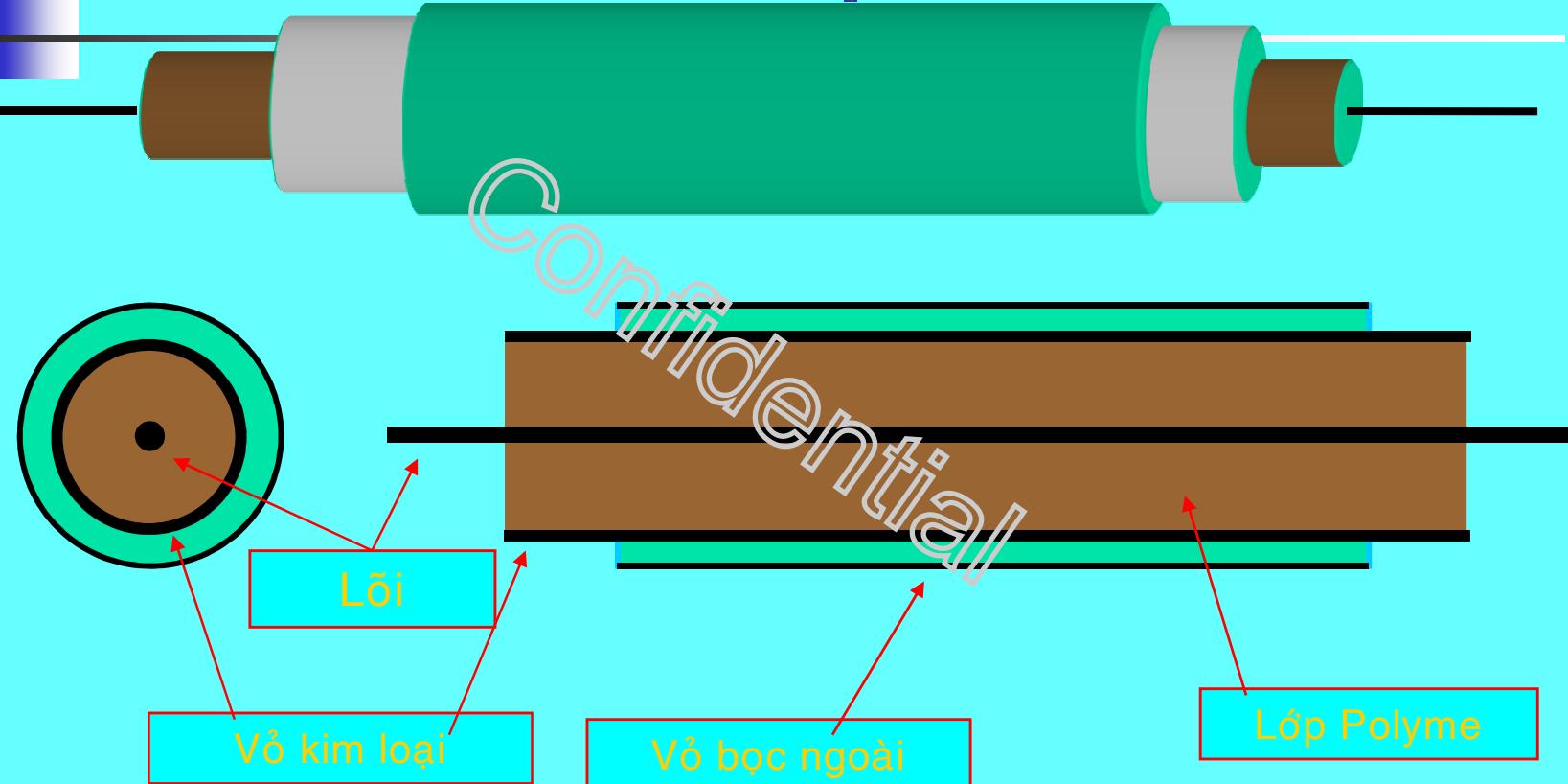
Cáp đồng trục (Coaxial Cable) :

* Cấu tạo :

- Bao gồm ống trực bên ngoài và một dây dẫn bên trong.
- Giữa trực lõi và ống bên ngoài được đặt cách đều nhau và cách ly bởi phần cách điện.
- Trục bên ngoài được bao bởi một lớp bảo vệ hoặc vỏ bọc.



CÁP ĐỒNG TRỰC

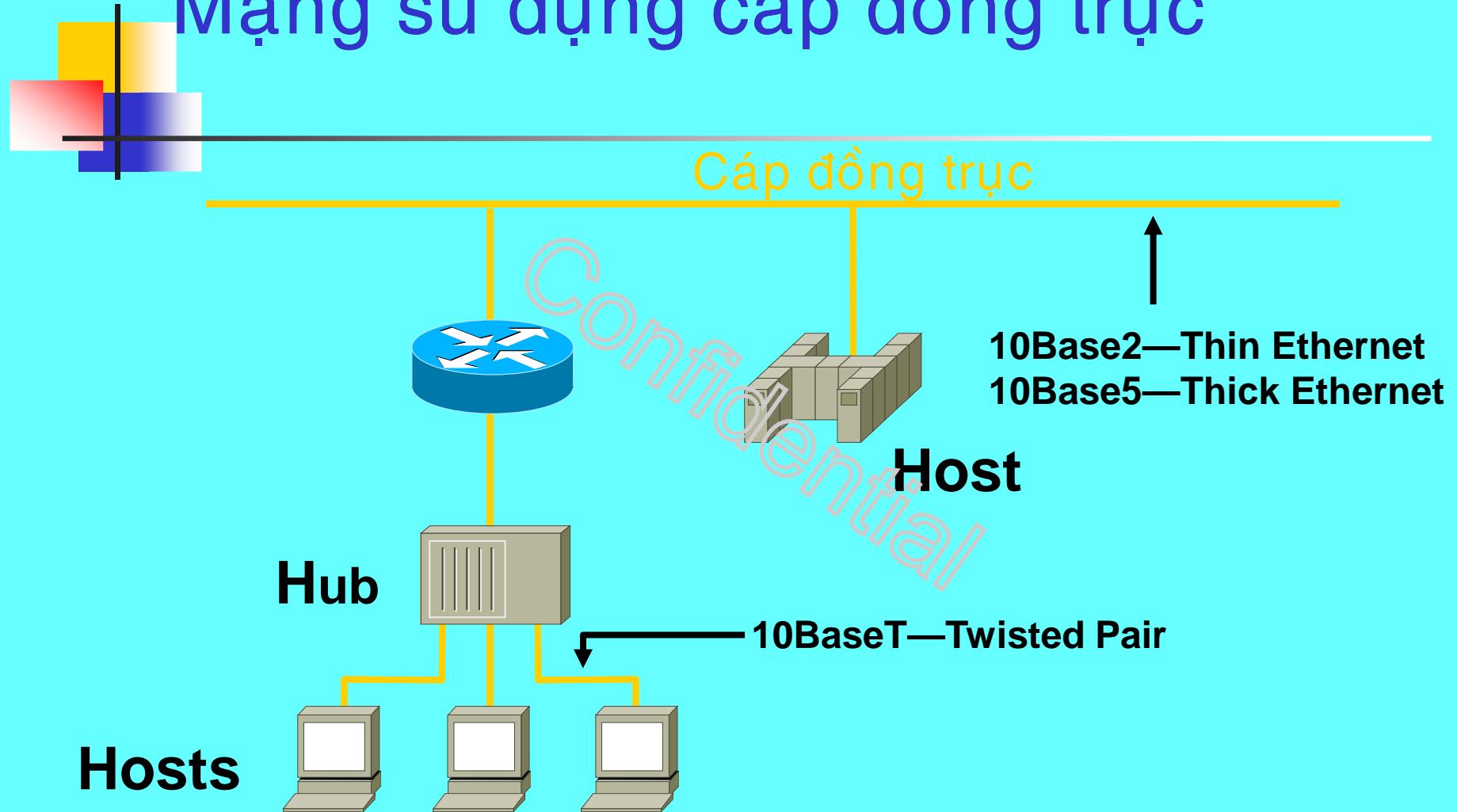


Cấu tạo của cáp đồng trực

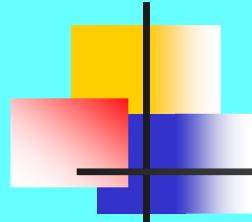
ĐẶC ĐIỂM CƠ BẢN CỦA CÁP ĐỒNG TRỰC

- Là môi trường truyền dẫn có chất lượng tốt hơn cáp xoắn đôi, lý do:
 - Không có nhiễu xuyên âm
 - Hạn chế được nhiễu từ bên ngoài
- Tốc độ bít có thể đạt đến 100Mbit/s.
- Giá thành cao, khó thi công.
- Bị giới hạn về khoảng cách và số kết nối.

Mạng sử dụng cáp đồng trục



MÔI TRƯỜNG TRUYỀN



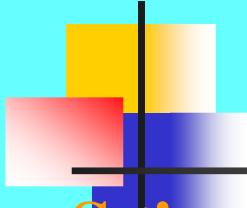
Sợi quang :

* Cấu tạo :

+ Gồm các sợi thủy tinh silica (hợp chất của silic dưới dạng thạch anh hoặc đá lửa và những đá khác) và được bao phủ bởi một lớp chất dẻo bên ngoài



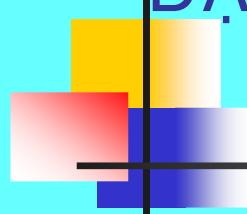
MÔI TRƯỜNG TRUYỀN



Sợi quang :

* Đặc tính truyền :

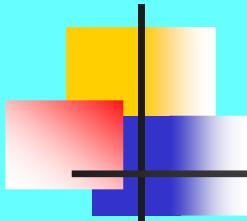
- + **Băng thông rộng:** Tiềm lực về băng thông và tốc độ truyền của vật dẫn tăng với tần số. Với dải tần rộng lớn của sợi quang, tốc độ dữ liệu >2 Gbps trên đoạn đường hàng chục Km
- + **Kích thước nhỏ, trọng lượng nhẹ:** Sợi quang thường nhỏ hơn cáp đồng trực và cặp dây song hành → rất thuận tiện khi sử dụng.
- + **Suy giảm ít:** Cáp quang suy giảm quá nhỏ so với cáp đồng trực và cặp dây song hành và nó là hằng số với khoảng cách xa.
- + **Cách ly điện từ:** Sợi quang không bị ảnh hưởng bởi trường điện từ, do đó không sợ nhiễu xuyên âm, nhiễu xung.
- + **Khoảng cách lặp lại lớn:** Ít cần phải repeater có nghĩa là giá thành giảm và ít bị sai số. Đó là ưu điểm rất lớn của sợi quang.



ĐẶC ĐIỂM CƠ BẢN CỦA CÁP SỢI QUANG

- Tín hiệu truyền dẫn là sóng ánh sáng theo nguyên tắc chớp (bit 1) và tắt (bit 0).
- Là môi trường truyền dẫn có chất lượng tốt nhất do cáp quang không bị nhiễu bởi sóng điện từ.
- Tốc độ bít có thể đạt đến hàng Gbit/s với khoảng cách truyền rất xa.
- Thường sử dụng ở các mạng trực

MÔI TRƯỜNG TRUYỀN



Viba mặt đất :

* Cấu tạo :

+ Loại ăng-ten thường dùng cho nó thường là đĩa parabol

+ ăng-ten được cố định và hướng chùm tia đến đường dẫn nhìn thấy được đến ăng-ten bộ thu.

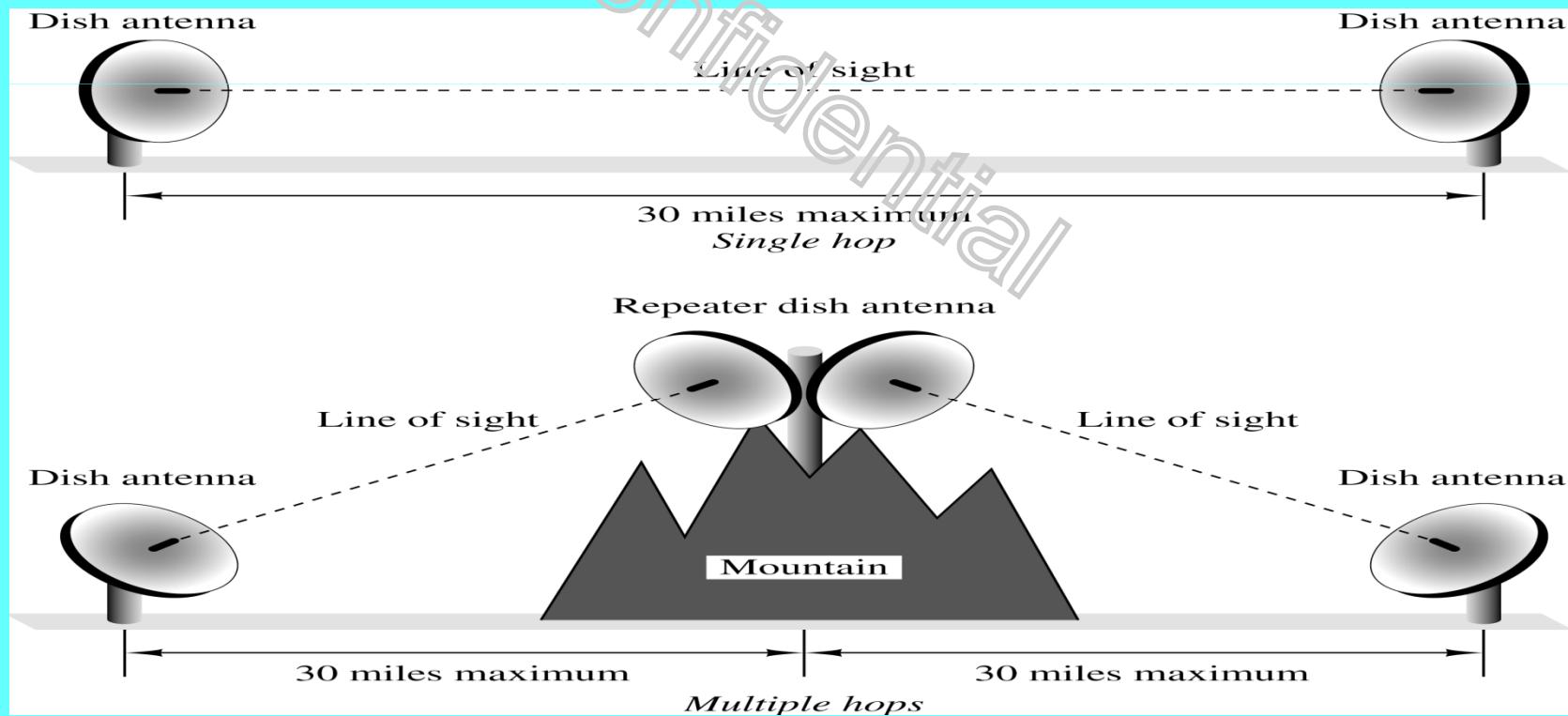
+ ăng-ten vi ba được gắn ở độ cao để phạm vi hoạt động giữa 2 ăng-ten không bị vật cản.



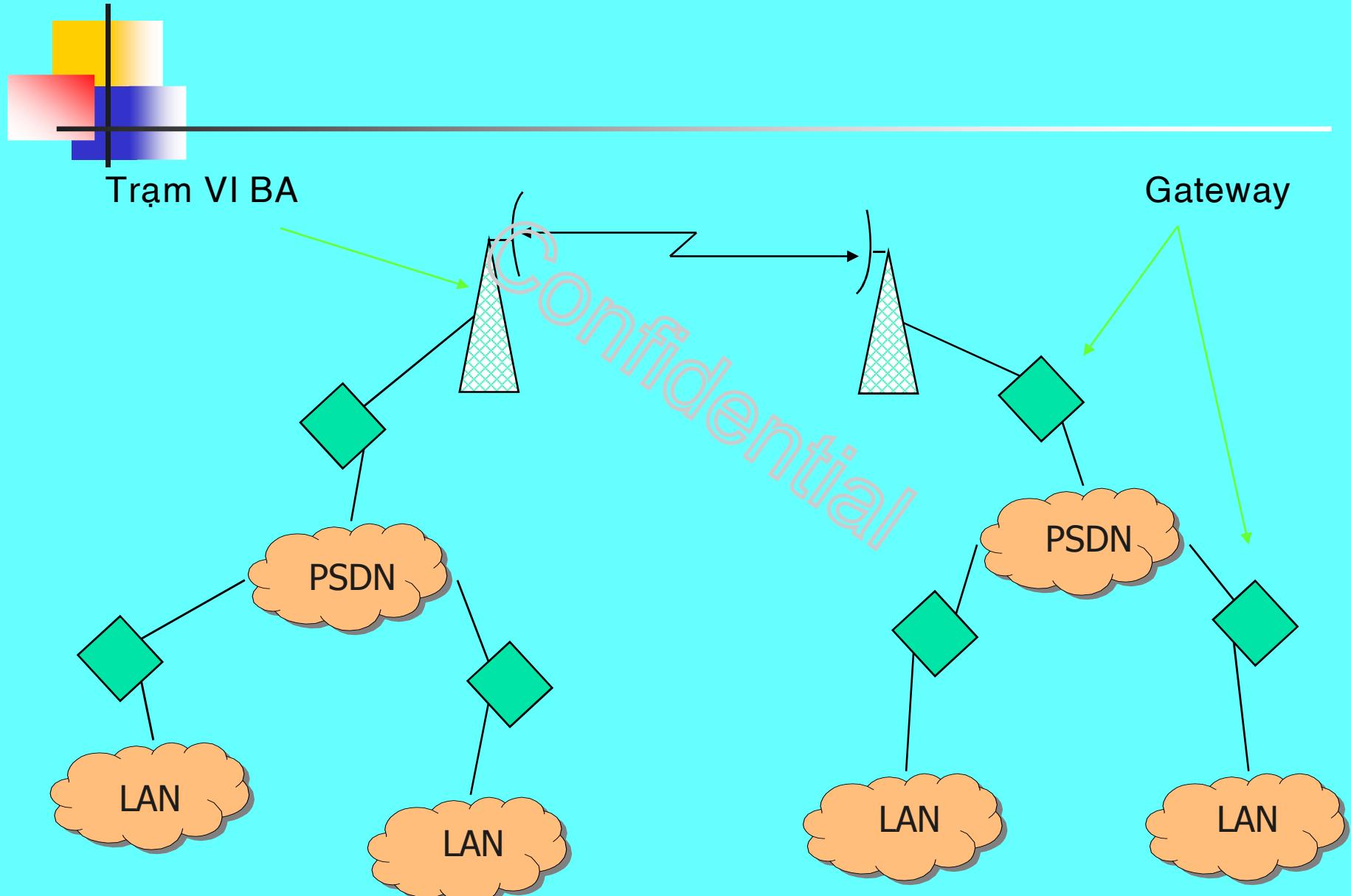
MÔI TRƯỜNG TRUYỀN

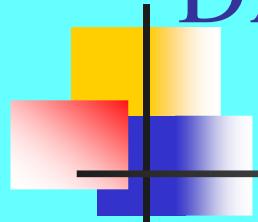
Viba mặt đất :

* Đặc tính truyền



HỆ THỐNG VI BA MẶT ĐẤT





ĐẶC ĐIỂM CỦA HỆ THỐNG VI BA

- Sử dụng sóng điện từ ở giải tần GHz để chuyển tiếp thông tin trên mặt đất.
- Chất lượng đường truyền không cao.
- Không sử dụng để truy nhập trực tiếp mạng.
- Thường sử dụng để chuyển tiếp thông tin giữa các nút mạng.

MÔI TRƯỜNG TRUYỀN

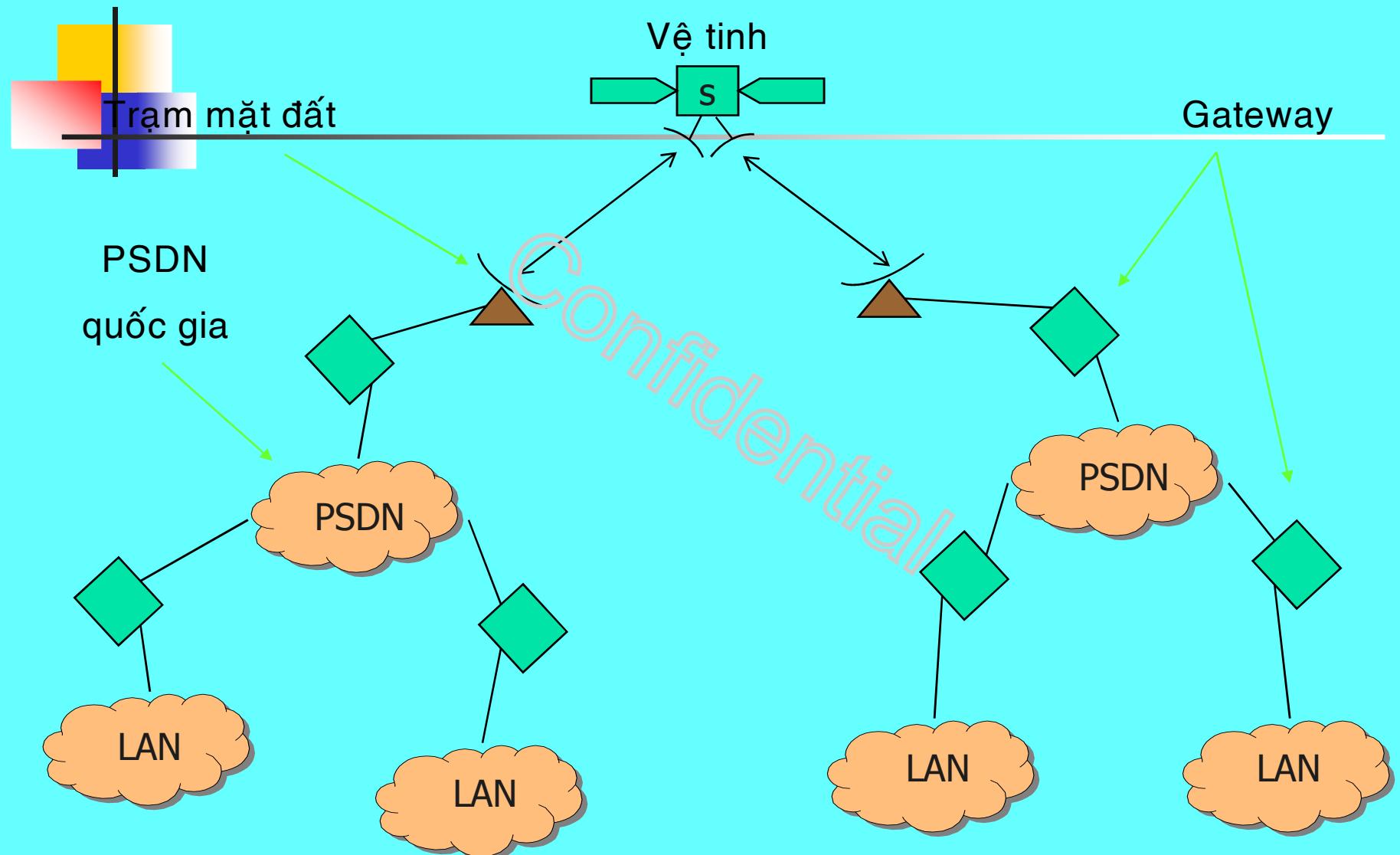


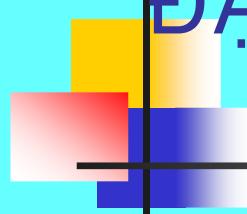
Viba vê tinh :

* Cấu tạo :

- + Vê tinh thông tin là một trạm chuyển tiếp, được dùng để nối hai hoặc nhiều bộ thu phát cơ bản và được coi như là trạm mặt đất hay trạm đất.
- + Một vệ tinh quỹ đạo đơn giản có thể tác động trên nhiều băng tần mà ta gọi là kênh, hay đơn giản là cầu truyền thông.
- + Để một vệ tinh liên lạc làm việc có hiệu quả, thông thường yêu cầu nó phải tự quay quanh nó.
- + Mặt khác vì phải nhìn thấy nhau nên tốc độ quay của nó phải bằng tốc độ quay của trái đất.

HỆ THỐNG VỆ TINH

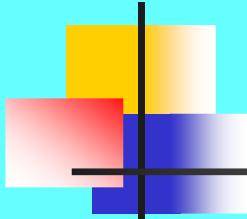




ĐẶC ĐIỂM CỦA HỆ THỐNG VỆ TINH

- Sử dụng sóng điện từ ở giải tần GHz để chuyển tiếp thông tin
- Chất lượng đường truyền khá cao.
- Có thể sử dụng để truy nhập trực tiếp mạng.
- Thường sử dụng để chuyển tiếp thông tin giữa các nút mạng liên quốc gia.

MÔI TRƯỜNG TRUYỀN



Viba vệ tinh :

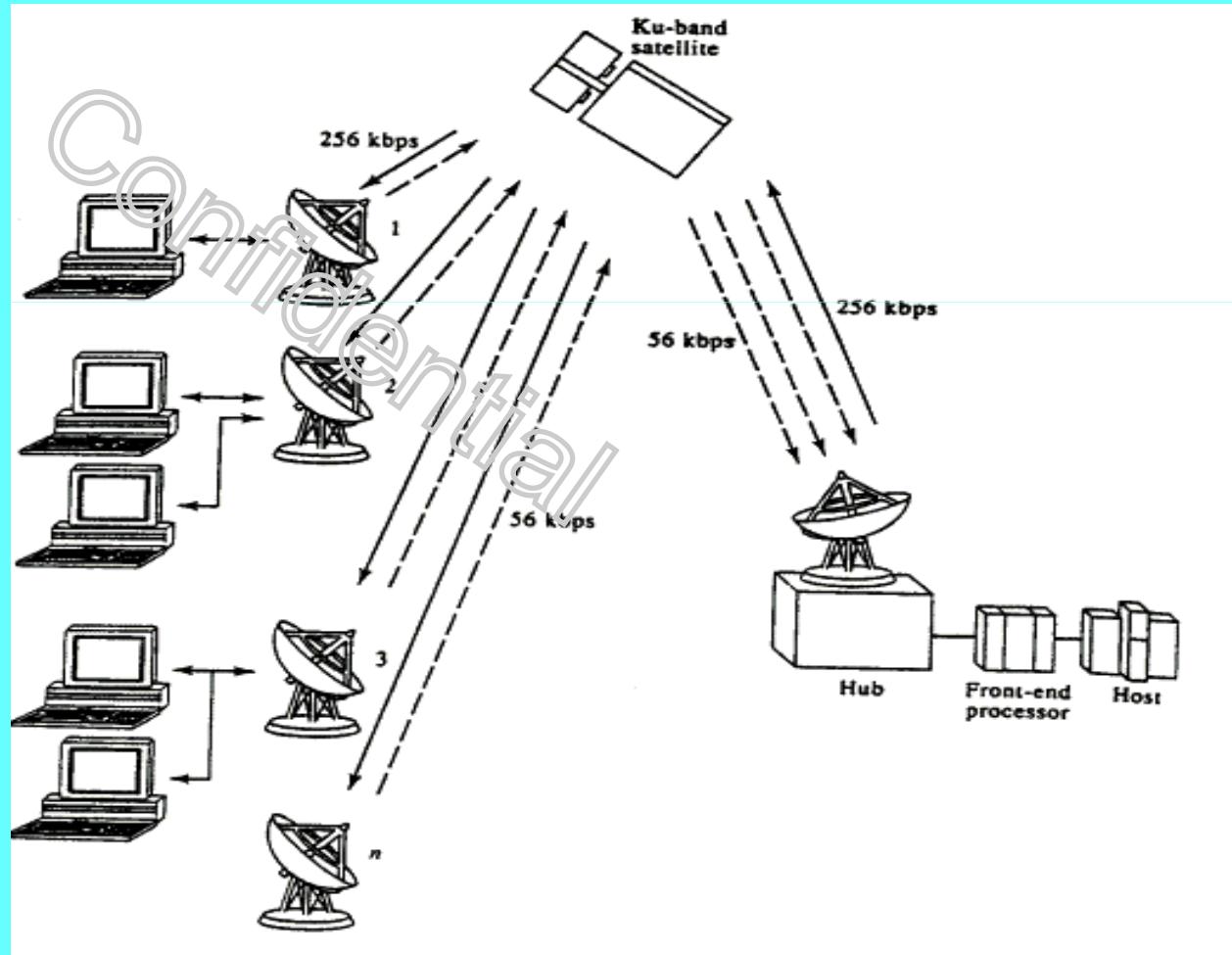
* Ứng dụng :

Vệ tinh liên lạc là cuộc cách mạng về kỹ thuật. Nó quan trọng cũng như sợi quang. Sau đây là một số ứng dụng quan trọng của nó:

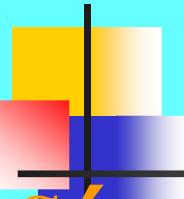
- * phân phối truyền hình
- * truyền điện thoại khoảng cách xa
- * mạng thương mại tư nhân

MÔI TRƯỜNG TRUYỀN

Viba vệ tinh :
* Ứng dụng :



MÔI TRƯỜNG TRUYỀN

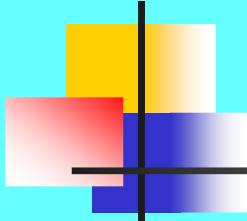


Sóng Radio :

* Cấu tạo :



MÔI TRƯỜNG TRUYỀN



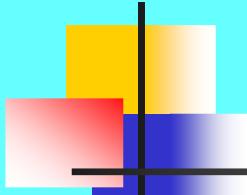
Sóng Radio :

* **Cấu tạo :**

+ Nguyên lý căn bản để phân biệt giữa radio và sóng viba là: radio tri không định hướng, còn viba là tập trung.

+ Như vậy radio không cần ăng ten đĩa và ăng ten của nó cũng không cần đặt ở trên độ cao và có kích thước chính xác.

MÔI TRƯỜNG TRUYỀN



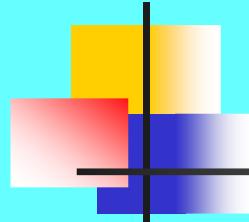
Sóng Radio :

* Đặc tính truyền :

+ các nguyên nhân ảnh hưởng đến sóng radio: phản xạ mặt đất, nước, các vật cản thiên nhiên giữa các ăng ten. Những hiệu ứng đó sẽ làm cho khi nhận tivi tạo thành nhiều ảnh, không nét.

+ Tốc độ trung bình, giá cả vừa phải, có thể sử dụng ở khoảng cách xa.

MÔI TRƯỜNG TRUYỀN

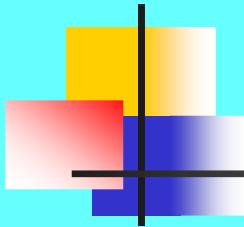


Sóng Radio :

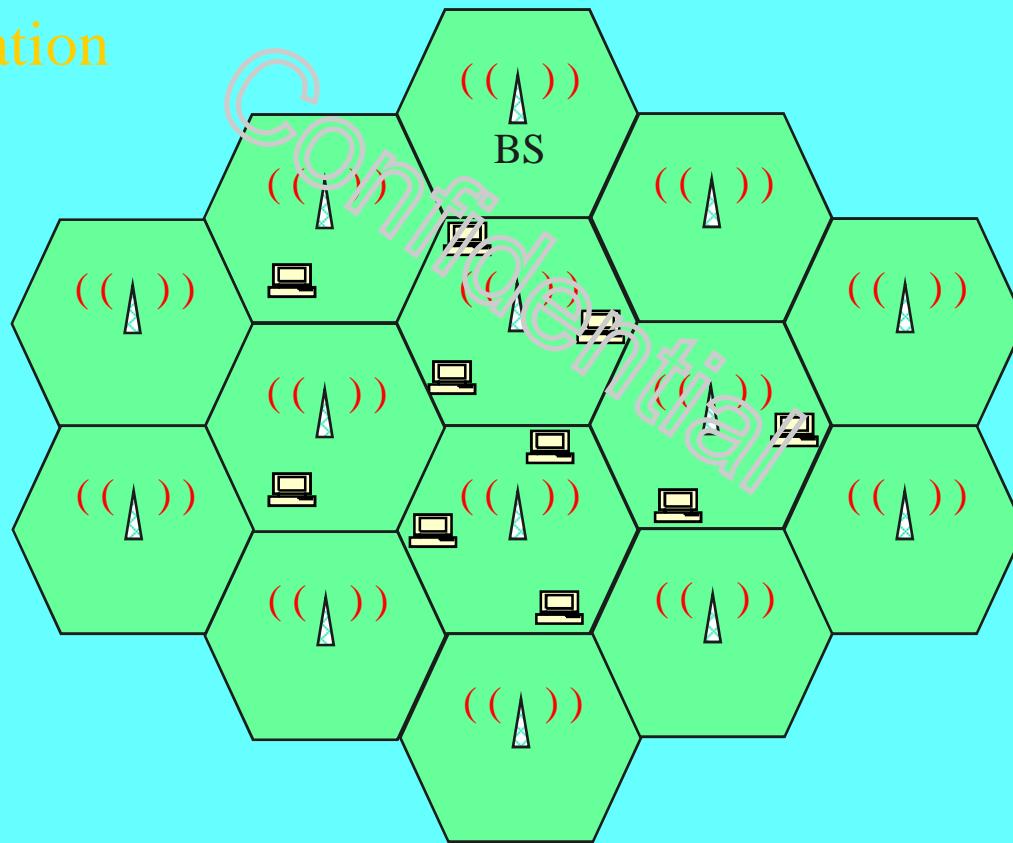
* Ứng dụng :

- + Chúng ta dùng sóng radio với khoảng sử dụng rộng hơn gồm cả VHF và một phân băng UHF : 30MHz – 1GHz.
- + Băng này bao gồm cả FM và UHF, VHF cho truyền hình.
- + Một loại thông tin dữ liệu số thường dùng là radio gói. Một hệ thống radio gói sử dụng ăng ten mặt đất cho nhiều địa điểm trong mạng truyền dữ liệu.

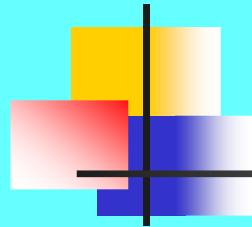
HỆ THỐNG VÔ TUYẾN TẾ BÀO



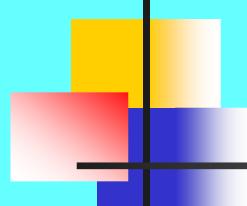
BS = Base Station



ĐẶC ĐIỂM CỦA HỆ THỐNG THÔNG TIN VÔ TUYẾN TẾ BÀO



- Toàn bộ vùng phục vụ được chia thành các tế bào (cell), thường có hình lục giác.
- Mỗi tế bào do một trạm gốc (BS) phục vụ kết nối với các trạm đầu cuối bằng sóng điện từ.
- Chất lượng đường truyền khá cao.
- Có thể sử dụng để truy nhập trực tiếp mạng cho các khu vực khó kéo cáp hoặc đầu cuối không cố định.



Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.1. Card giao tiếp mạng (Network Interface Card)

Đó là một card ~~được~~ cắm trực tiếp vào máy tính trên khe cắm mở rộng ISA hoặc PCI hoặc tích hợp vào bo mạch chủ PC. Trên đó có các mạch điện giúp cho việc **tiếp nhận (receiver) hoặc phát (transmitter) tín hiệu lên mạng**.

2.2.2 Bộ chuyển tiếp (REPEATER)

Repeater là loại thiết bị phần cứng đơn giản nhất trong các thiết bị liên kết mạng, nó được **hoạt động trong tầng vật lý** của mô hình hệ thống mở OSI.

Repeater dùng để nối 2 mạng giống nhau hoặc các phần một mạng cùng có một giao thức và một cấu hình.

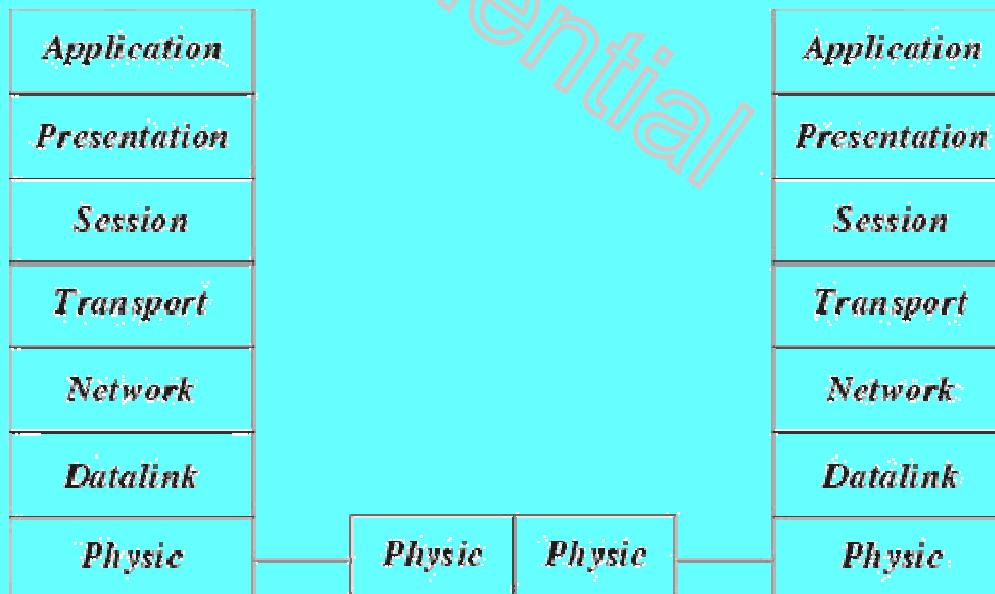
Khi Repeater nhận được một tín hiệu từ một phía của mạng thì nó sẽ phát tiếp vào phía kia của mạng.

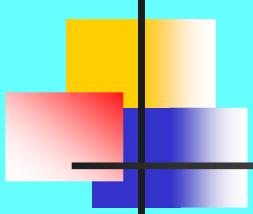
Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.2 Bộ chuyển tiếp (REPEATER)

Hoạt động của bộ chuyển tiếp trong mô hình OSI





Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

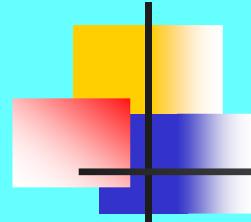
2.2.2 Bộ chuyển tiếp (REPEATER)

Repeater ~~không có xử lý tín hiệu, chỉ loại bỏ các tín hiệu méo, nhiễu, khuếch đại tín hiệu đã bị suy hao và khôi phục lại tín hiệu ban đầu.~~

Việc sử dụng Repeater để làm tăng thêm chiều dài của mạng. Việc sử dụng Repeater ~~không thay đổi nội dung các tín hiệu đi qua nó~~

Chỉ được dùng để nối hai mạng có cùng giao thức truyền thông (như hai mạng Ethernet hay hai mạng Token ring).

Chương 2 : Các thiết bị và các chuẩn kết nối



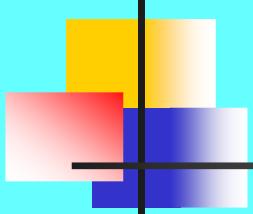
2.2 Các thiết bị kết nối

2.2.2 Bộ chuyển tiếp (REPEATER)

Nhiệm vụ của các repeater là hồi phục tín hiệu để có thể truyền tiếp cho các trạm khác bao gồm cả công tác khuếch đại tín hiệu, điều chỉnh tín hiệu



Mô hình liên kết mạng sử dụng Repeater



Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.3 Cầu nối ~~BRIDGE~~

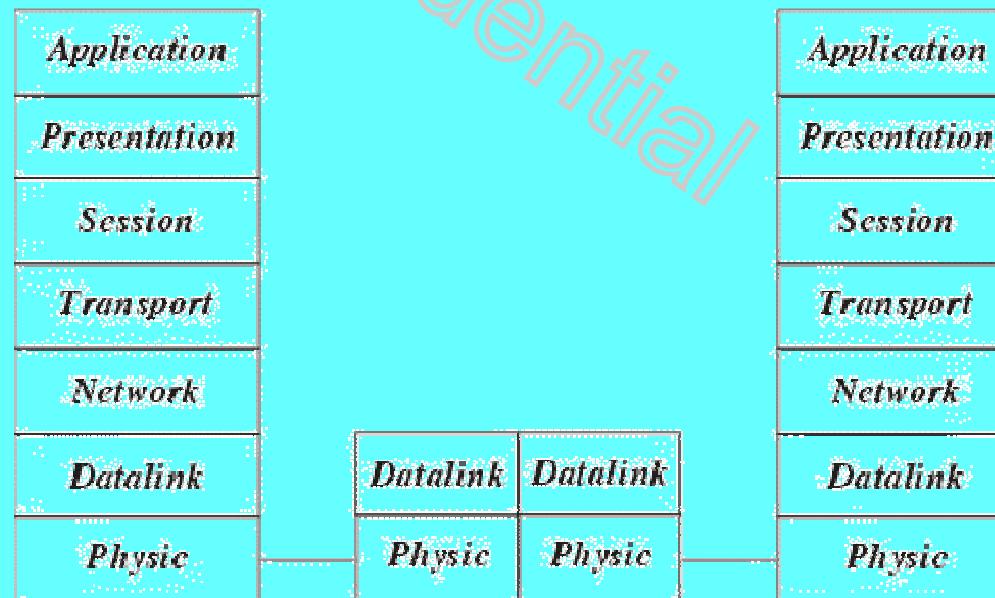
- Bridge là một thiết bị có xử lý dùng **để nối hai mạng giống nhau hoặc khác nhau**, nó có thể được dùng với các mạng có các giao thức khác nhau.
- **Cầu nối hoạt động trên tầng liên kết dữ liệu nên không phát lại tất cả những gì nó nhận được** và xử lý chúng trước khi quyết định có chuyển đi hay không.
- Khi nhận được các gói tin Bridge **chọn lọc và chỉ chuyển những gói tin mà nó thấy cần thiết**. Điều này làm cho Bridge trở nên có ích khi nối một vài mạng với nhau và cho phép nó hoạt động một cách mềm dẻo.

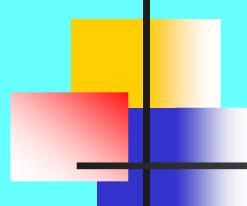
Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.3 Cầu nối BRIDGE

Hoạt động của Bridge trong mô hình OSI





Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

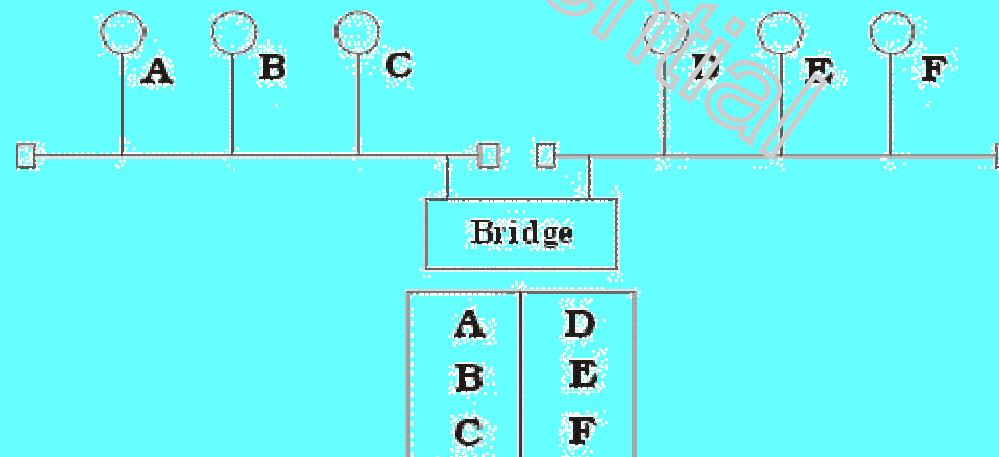
2.2.3 Cầu nối BRIDGE

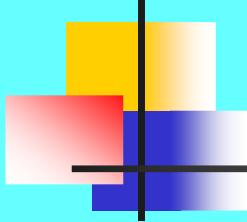
- Khi đọc địa chỉ nơi gửi Bridge kiểm tra xem trong bảng địa chỉ của phần mạng nhận được gói tin có địa chỉ đó hay không, nếu không có thì Bridge tự động bổ sung bảng địa chỉ.
- Khi đọc địa chỉ nơi nhận Bridge kiểm tra xem trong bảng địa chỉ của phần mạng nhận được gói tin có địa chỉ đó hay không, nếu có thì Bridge sẽ cho rằng đó là gói tin nội bộ thuộc phần mạng mà gói tin đến nên không chuyển gói tin đó đi, nếu ngược lại thì Bridge mới chuyển sang phía bên kia.

Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.3 Cầu nối BRIDGE





Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.3 Cầu nối BRIDGE

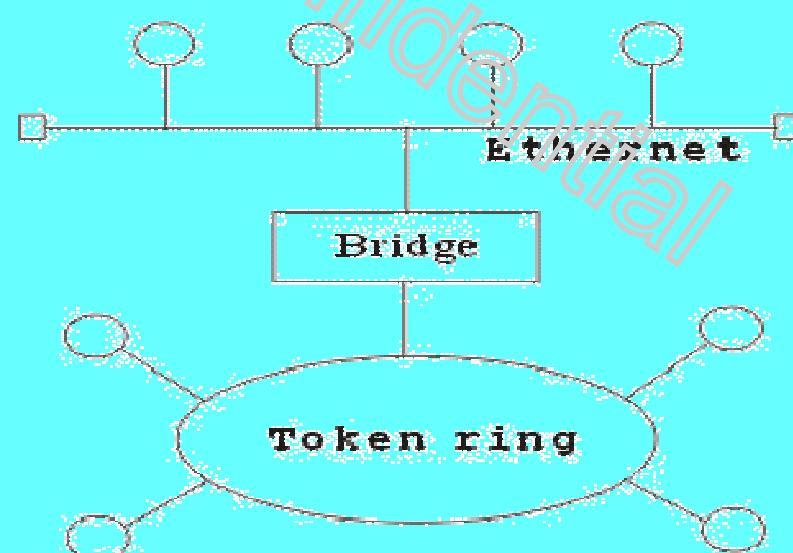
- Có hai loại Bridge đang được sử dụng là Bridge vận chuyển và Bridge biên dịch.
- *Bridge vận chuyển* dùng để nối hai mạng cục bộ cùng sử dụng một giao thức truyền thông của tầng liên kết dữ liệu.
 - Bridge vận chuyển **không có khả năng thay đổi cấu trúc các gói tin mà nó nhận được** mà chỉ quan tâm tới việc xem xét và chuyển vận gói tin đó đi.
- *Bridge biên dịch* dùng để nối hai mạng cục bộ có giao thức khác nhau nó có khả năng chuyển một gói tin thuộc mạng này sang mạng kia.

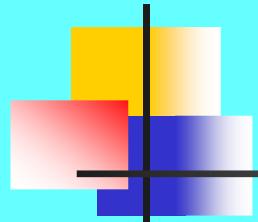
Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.3 Cầu nối BRIDGE

- Ví dụ Bridge biên dịch:





Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.3 Cầu nối BRIDGE

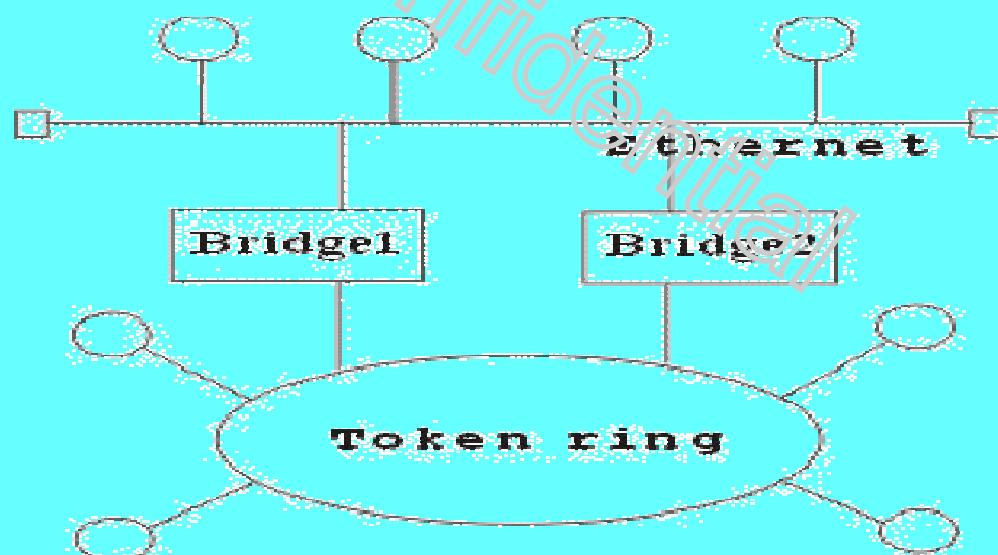
Người ta sử dụng Bridge trong các trường hợp sau :

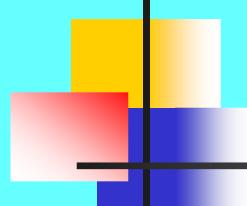
- Mở rộng mạng hiện tại khi đã đạt tới khoảng cách tối đa do Bridge sau khi xử lý gói tin đã phát lại gói tin trên phần mạng còn lại nên tín hiệu tốt hơn bộ Repeater.
- Giảm bớt tắc nghẽn mạng khi có quá nhiều trạm bằng cách sử dụng Bridge, khi đó chúng ta chia mạng ra thành nhiều phần bằng các Bridge, các gói tin trong nội bộ từng phần mạng sẽ không được phép qua phần mạng khác.
- Để nối các mạng có giao thức khác nhau.
- Bridge còn có khả năng lựa chọn đối tượng vận chuyển
 - (Cho phép gói tin của máy A, B qua Bridge 1, gói tin của máy C, D qua Bridge 2.

Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.3 Cầu nối BRIDGE





Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

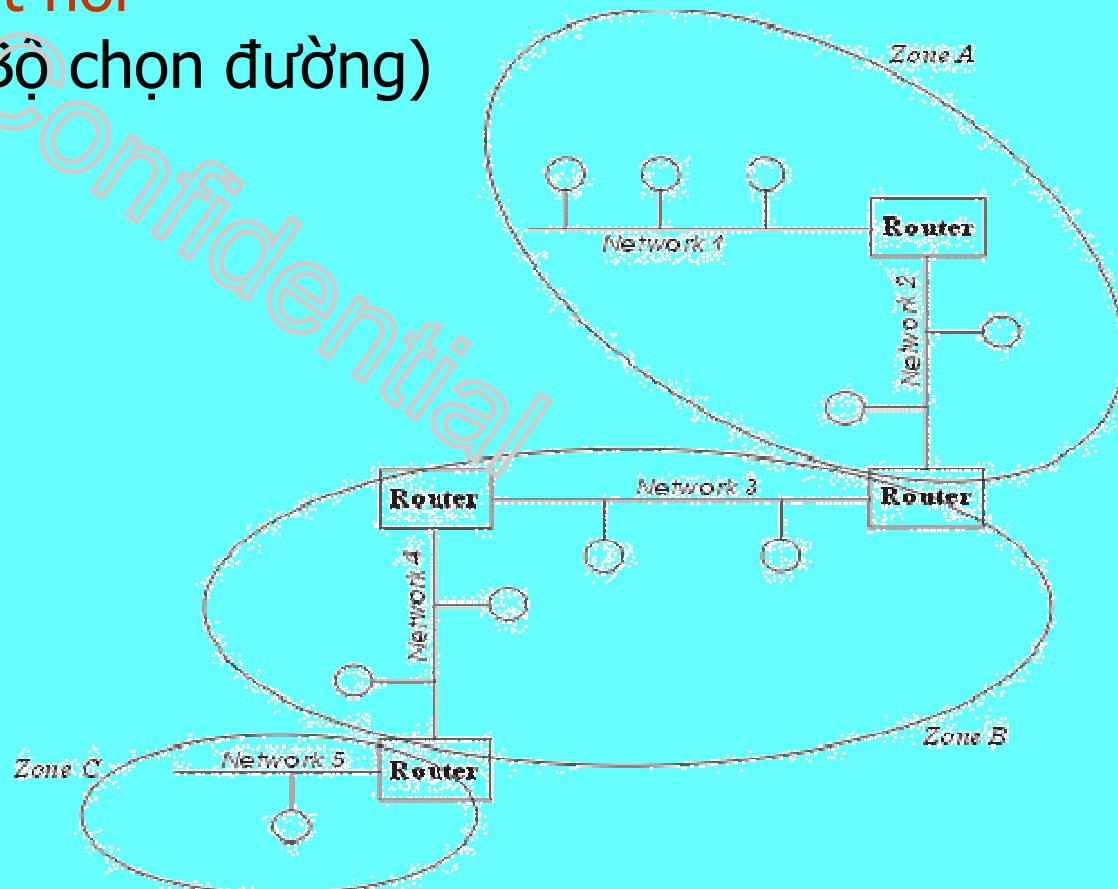
2.2.4. Router (Bộ chọn đường)

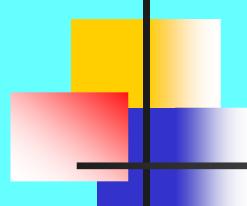
- Router là một thiết bị hoạt động trên tầng mạng
- Có thể tìm được đường đi tốt nhất cho các gói tin qua nhiều kết nối để đi từ trạm gửi thuộc mạng đầu đến trạm nhận thuộc mạng cuối.
- Router có thể được sử dụng trong việc **nối nhiều mạng với nhau** và cho phép các gói tin có thể đi theo nhiều đường khác nhau để tới đích.

Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.4. Router (Bộ chọn đường)





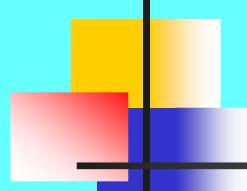
Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.4. Router (Bộ chọn đường)

Khác với Bridge hoạt động trên tầng liên kết dữ liệu nên Bridge phải xử lý mọi gói tin trên đường truyền thì Router có địa chỉ riêng biệt và nó chỉ tiếp nhận và xử lý các gói tin gửi đến nó mà thôi.

Khi một trạm muốn gửi gói tin qua Router thì nó phải gửi gói tin với địa chỉ trực tiếp của Router thì Router mới xử lý và gửi tiếp.



Chương 2 : Các thiết bị và các chuẩn kết nối

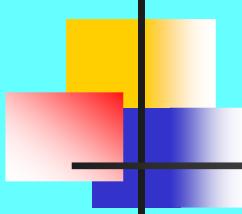
2.2 Các thiết bị kết nối

2.2.4. Router (Bộ chọn đường)

Khi xử lý một gói tin Router phải tìm được đường đi của gói tin qua mạng.

Để làm được điều đó Router phải tìm được đường đi tốt nhất trong mạng dựa trên các thông tin nó có về mạng, thông thường trên mỗi Router có một bảng chỉ đường (Router table).

Dựa trên dữ liệu về Router gần đó và các mạng trong liên mạng, Router tính được bảng chỉ đường (Router table) tối ưu dựa trên một thuật toán xác định trước



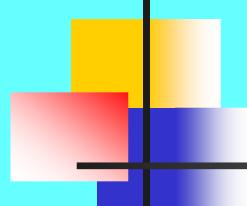
Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.4. Router (Bộ chọn đường)

Router có phụ thuộc giao thức: Chỉ thực hiện việc tìm đường và truyền gói tin từ mạng này sang mạng khác chứ không chuyển đổi phương cách đóng gói của gói tin cho nên cả hai mạng phải dùng chung một giao thức truyền thông.

Router không phụ thuộc vào giao thức: có thể liên kết các mạng dùng giao thức truyền thông khác nhau và có thể chuyển đổi gói tin của giao thức này sang gói tin của giao thức kia (Router có thể chia nhỏ một gói tin lớn thành nhiều gói tin nhỏ trước truyền trên mạng).



Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.5. Các bộ ~~tập~~ trung (Concentrator hay HUB)

HUB là một loại thiết bị có nhiều đầu cắm cáp mạng. Ưu điểm của kiểu nối này là tăng độ độc lập của các máy khi một máy bị sự cố dây cáp.

Có loại HUB thụ động (passive HUB) là HUB chỉ đảm bảo chức năng kết nối hoàn toàn ~~không~~ xử lý lại tín hiệu.

HUB chủ động (active HUB) là HUB có chức năng khuếch đại tín hiệu để chống suy hao.

HUB thông minh (intelligent HUB) là HUB chủ động nhưng có khả năng tạo ra các gói tin mang tin tức về hoạt động của mình và gửi lên mạng để người quản trị mạng có thể thực hiện quản trị tự động

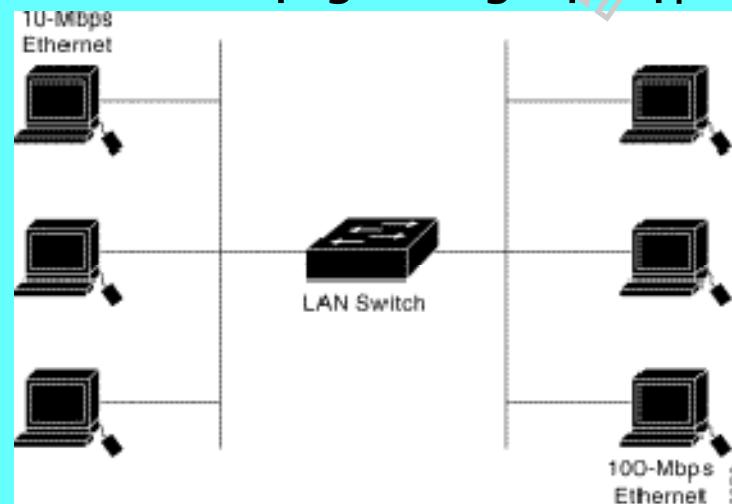
Chương 2 : Các thiết bị và các chuẩn kết nối

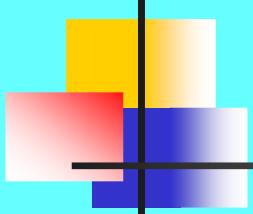
2.2 Các thiết bị kết nối

2.2.6. Switching Hub (hay còn gọi tắt là switch)

Là các bộ chuyển mạch thực sự. Khác với HUB thông thường, thay vì chuyển một tín hiệu đến từ một cổng cho tất cả các cổng, nó chỉ chuyển tín hiệu đến cổng có trạm đích.

Do vậy Switch là một thiết bị quan trọng trong các mạng cục bộ lớn dùng để phân đoạn mạng. Nhờ có switch mà dung độ trên mạng giảm hẳn. Ngày nay switch là các thiết bị mạng quan trọng cho phép tùy biến trên mạng chẳng hạn lập mạng ảo VLAN.





Chương 2 : Các thiết bị và các chuẩn kết nối

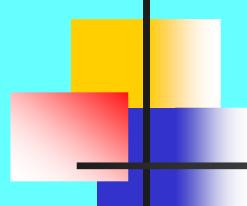
2.2 Các thiết bị kết nối

2.2.7. Modem

Là tên viết tắt từ hai từ điều chế (MOdulation) và giải điều chế (DEModulation) là thiết bị cho phép điều chế để biến đổi tín hiệu số sang tín hiệu tương tự để có thể gửi theo đường thoại và khi nhận tín hiệu từ đường thoại có thể biến đổi ngược lại thành tín hiệu số.

2.2.8. Multiplexor - Demultiplexor

Bộ dồn kênh có chức năng tổ hợp nhiều tín hiệu để cùng gửi trên một đường truyền. Bộ tách kênh có chức năng ngược lại ở nơi nhận tín hiệu



Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.9. Gateway (Cổng nối)

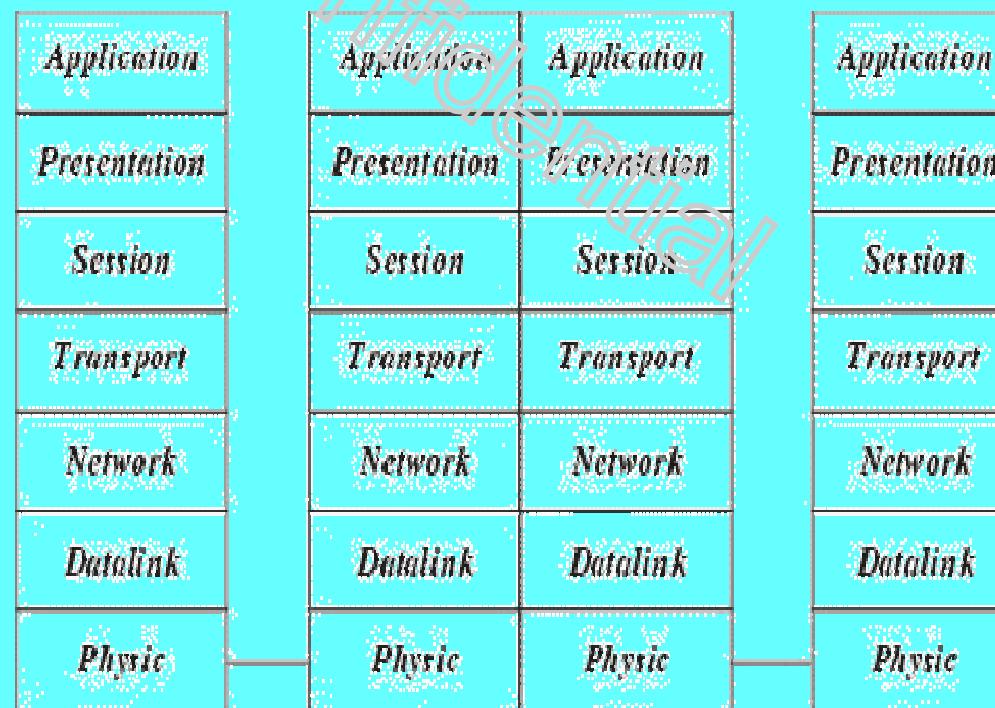
- Gateway dùng để kết nối các mạng không thuần nhất chẳng hạn như các mạng cục bộ và các mạng máy tính lớn (Mainframe)
- Do các mạng hoàn toàn không thuần nhất nên việc chuyển đổi thực hiện trên cả 7 tầng của hệ thống mở OSI. Thường được sử dụng nối các mạng LAN vào máy tính lớn.
- Gateway có các giao thức xác định trước thường là nhiều giao thức, một Gateway đa giao thức thường được chế tạo như các Card có chứa các bộ xử lý riêng và cài đặt trên các máy tính hoặc thiết bị chuyên biệt.

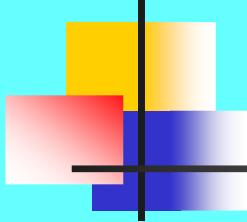
Chương 2 : Các thiết bị và các chuẩn kết nối

2.2 Các thiết bị kết nối

2.2.9. Gateway (Cổng nối)

Hoạt động của Gateway trong mô hình OSI





Chương 3: Các giao thức cơ bản

- Giới thiệu về giao thức
- IP (Internet Protocol)
- ICMP (Internet Control Message Protocol)
- IGMP (Internet Group Management Protocol)
- TCP (Transmission Control Protocol)
- UDP (User Datagram Protocol)

Chương 3: Các giao thức cơ bản

3.1. Giới thiệu về giao thức :

3.1.1. Giới thiệu :

Protocol là một bộ các qui ước ràng buộc về trao đổi thông tin. Khi cài đặt có thể là một driver hoặc một đoạn mã trong ROM, trong chương trình ứng dụng, ...

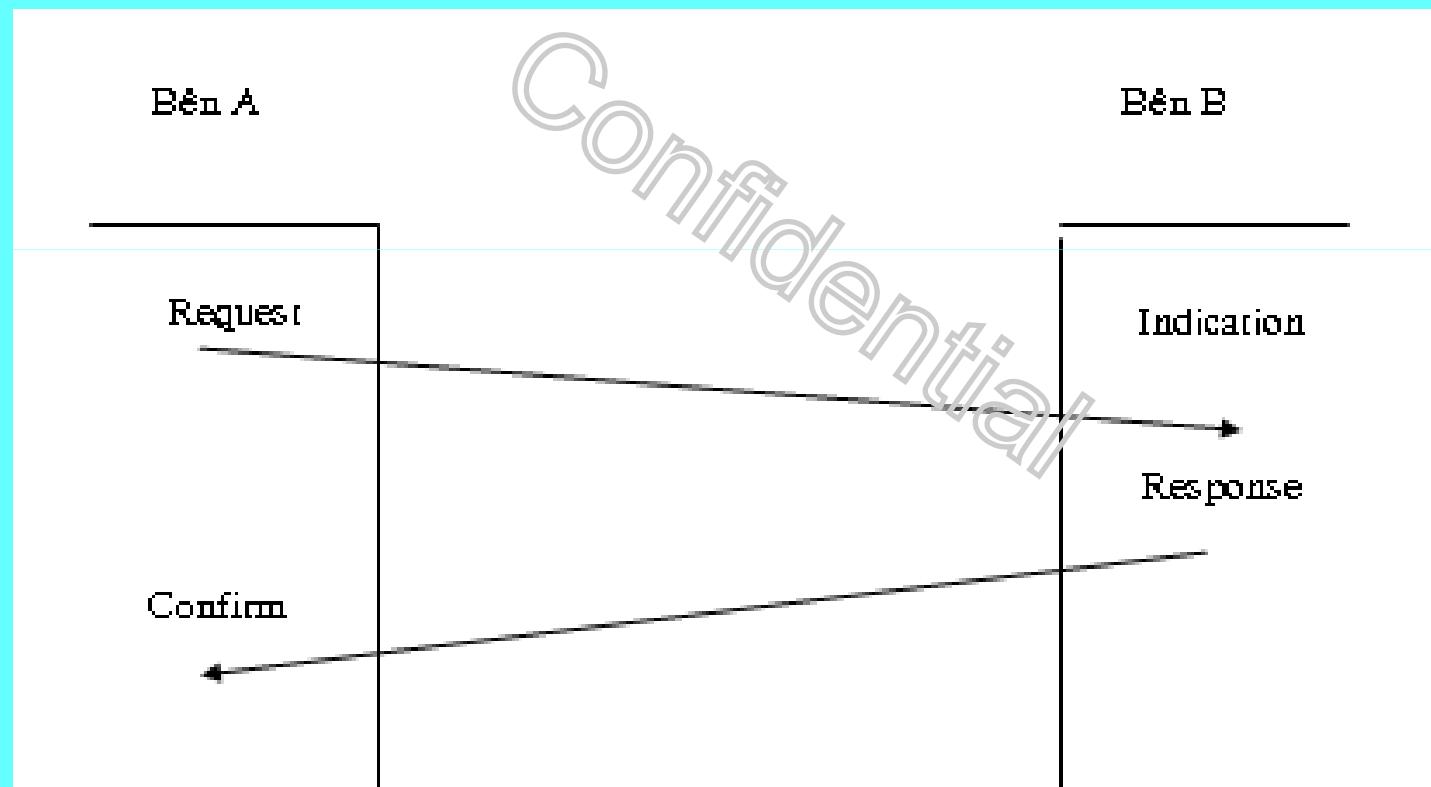
Protocol có 4 tác vụ cơ bản:

- + Request : yêu cầu thực hiện một thao tác
- + Indication : Thông báo đã nhận được một sự kiện đang chờ xử lý (thông thường đó là một request của một layer khác)
- + Response : Trả lời chấp nhận hoặc không chấp nhận đối với yêu cầu sự kiện đang chờ xử lý.
- + Confirm : Báo cáo rằng layer khác đã phúc đáp yêu cầu

Chương 3:

Các giao thức cơ bản

3.1.1 Giới thiệu :



Các tác vụ cơ bản của giao thức

Chương 3: Các giao thức cơ bản

3.1.2 Các kiểu liên lạc của Protocol :

Gồm có liên lạc không kết nối và liên lạc hướng kết nối

+ Đối với các giao thức không kết nối : thì chỉ cần chuyển các gói dữ liệu đến layer mà nó quan hệ trực tiếp mà không cần biết gói sẽ đi đường nào để tới nơi nhận

Ví dụ : gửi thư thông qua bưu điện

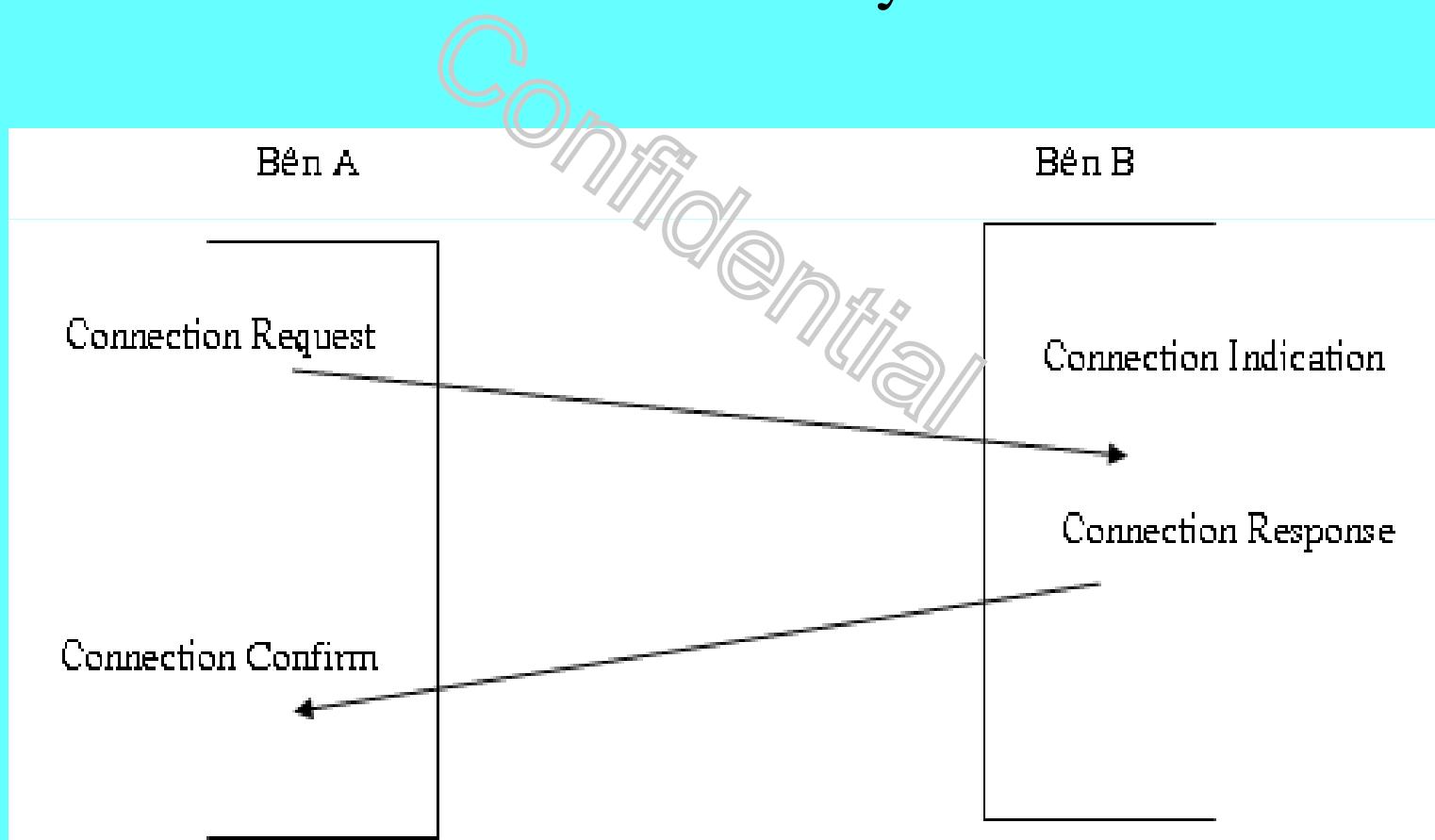
+ Đối với các giao thức có kết nối : việc trao đổi dữ liệu tiến hành cẩn thận hơn. Đầu tiên là gửi yêu cầu kết nối đến bên nhận, kế tiếp là thủ tục hand shaking và sau đó là quá trình trao đổi thông tin. Cuối cùng là thủ tục kết thúc kết nối.

Ví dụ : Tiến trình này tương tự như gọi điện thoại để trao đổi thông tin với một người ở xa.

Chương 3: Các giao thức cơ bản

3.1.2 Các kiểu liên lạc của Protocol :

Bước 1 : Kết nối và bắt tay



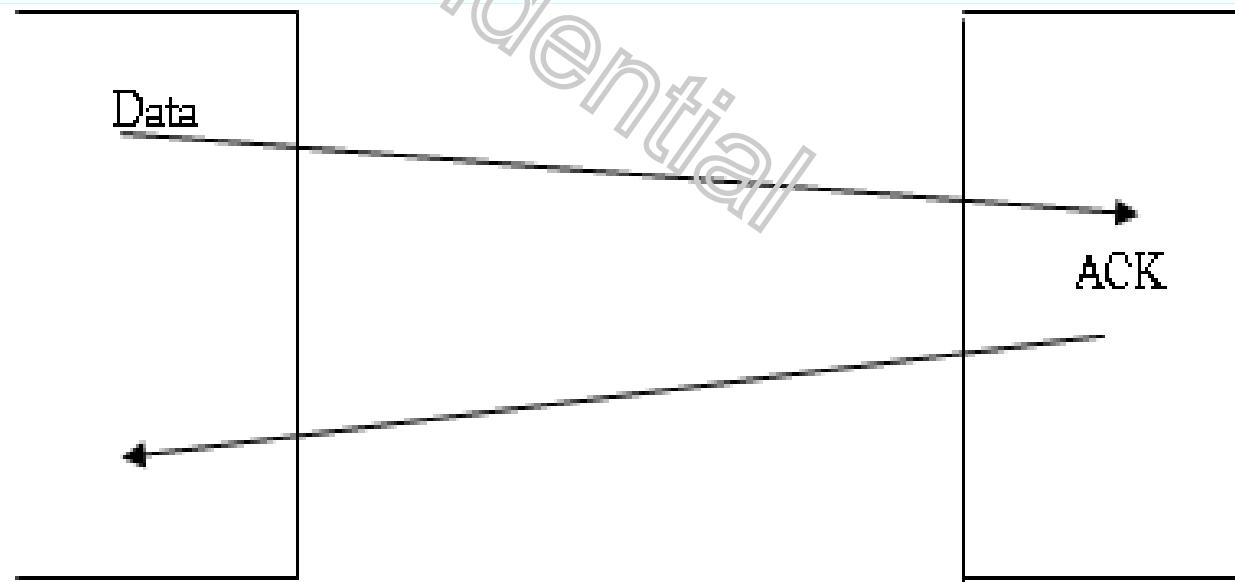
Chương 3: Các giao thức cơ bản

3.1.2 Các kiểu liên lạc của Protocol :

Bước 2 : Truyền dữ liệu

Data

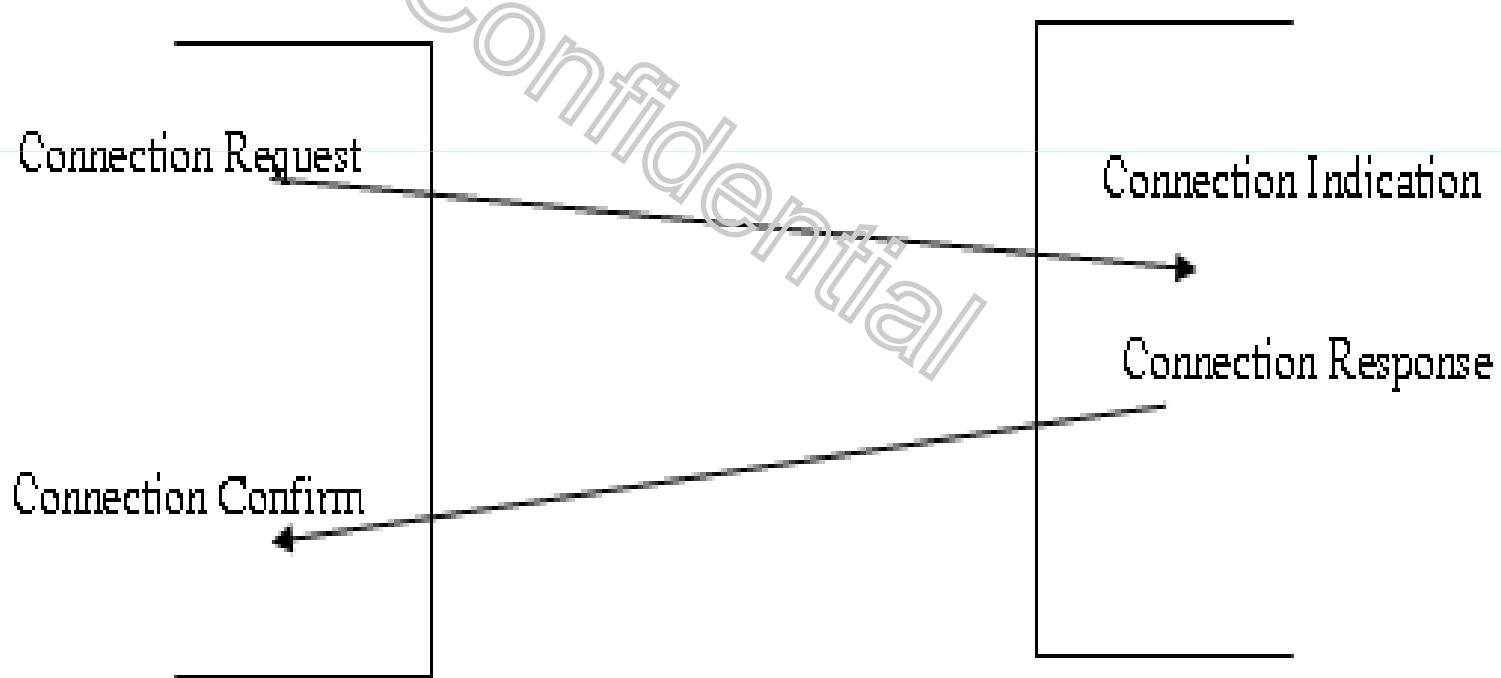
ACK



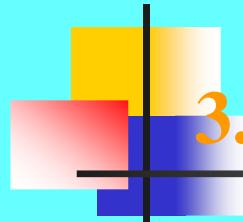
Chương 3: Các giao thức cơ bản

3.1.2 Các kiểu liên lạc của Protocol :

Bước 3 : Chấm dứt kết nối



Chương 3: Các giao thức cơ bản



3.1.3 Cơ chế kiểm soát lỗi của Protocol :

Tổng hợp lại ta dùng phương pháp yêu cầu phát lại ARQ (Automatic Repeat reQuest). Sau đây chúng ta khảo sát hai ~~phương~~ pháp kiểm soát lỗi quen thuộc là Idle-RQ và Continuous RQ.

a. Phương pháp kiểm lỗi Idle-RQ :

Idle-RQ còn có hai tên gọi khác là Stop and Wait và Send and Wait, làm việc với kiểu kết nối đường truyền half-duplex.

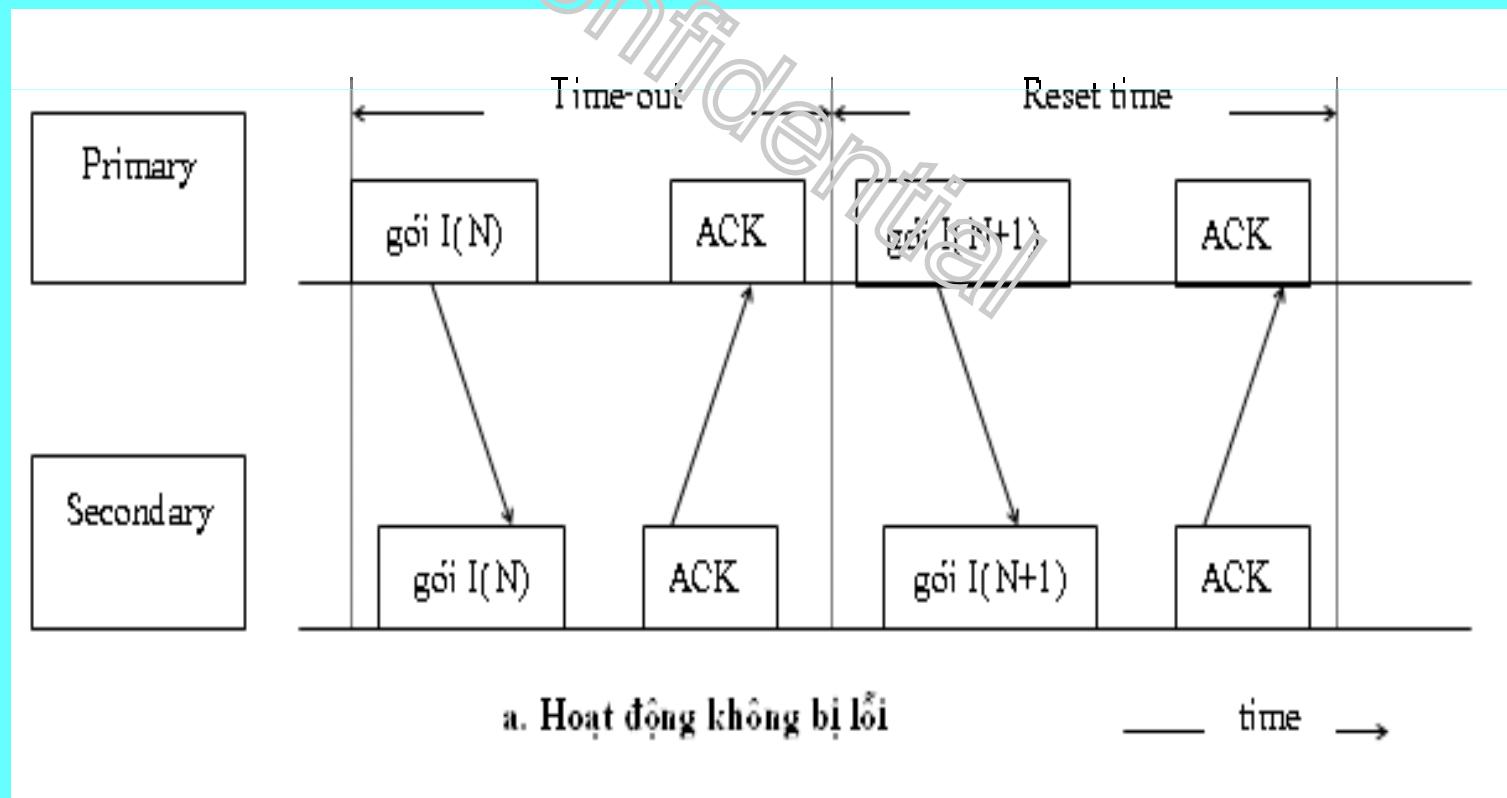
Bên phát gọi là primary và bên nhận secondary. Stop and Wait có hai kiểu hoạt động gọi là kiểu ẩn và kiểu hiện được đề cập sau đây:

Chương 3: Các giao thức cơ bản

3.1.3 Cơ chế kiểm soát lỗi của Protocol :

a. Phương pháp kiểm lỗi Idle-RQ :

* Hoạt động kiểu ẩn :

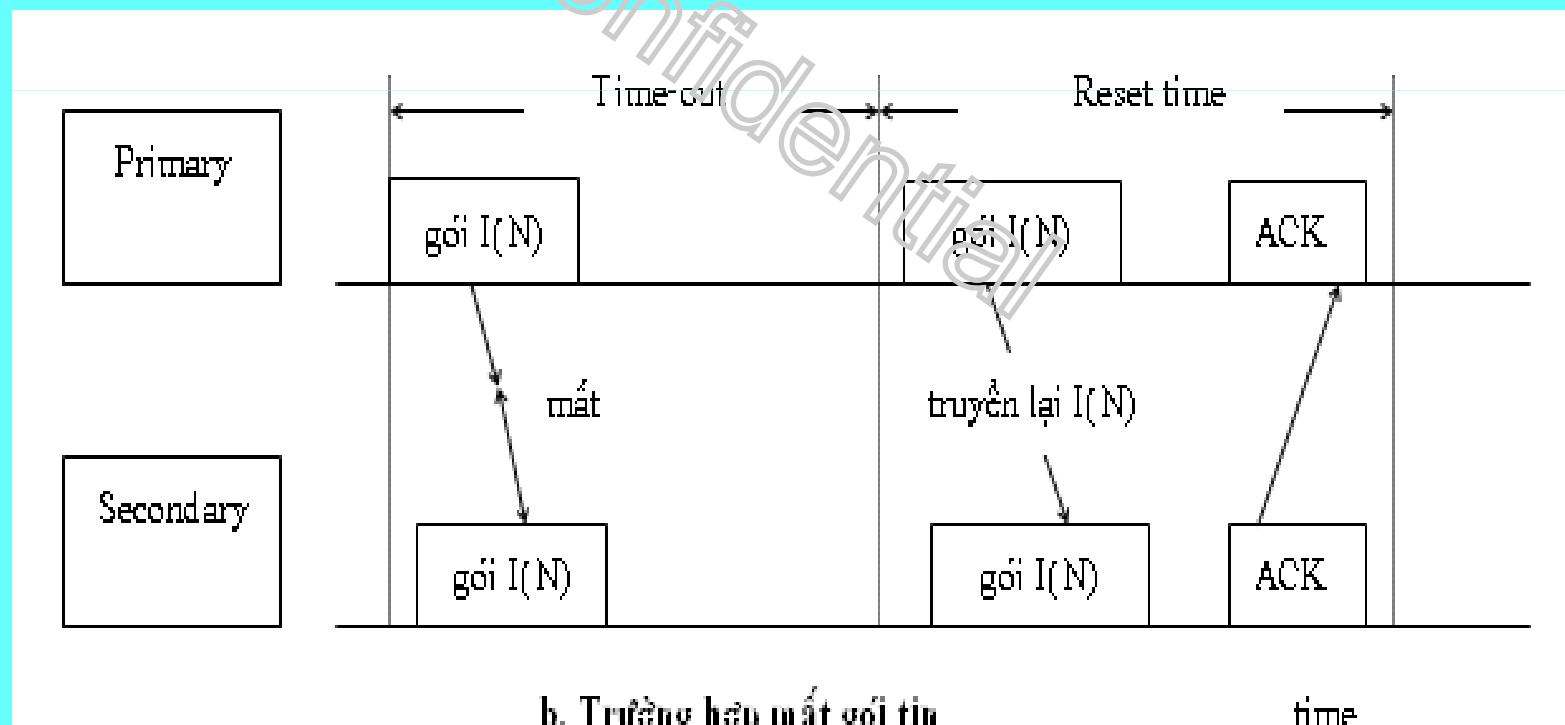


Chương 3: Các giao thức cơ bản

3.1.3 Cơ chế kiểm soát lỗi của Protocol :

a. Phương pháp kiểm lỗi Idle-RQ :

* Hoạt động ~~kiểu~~ ẩn :

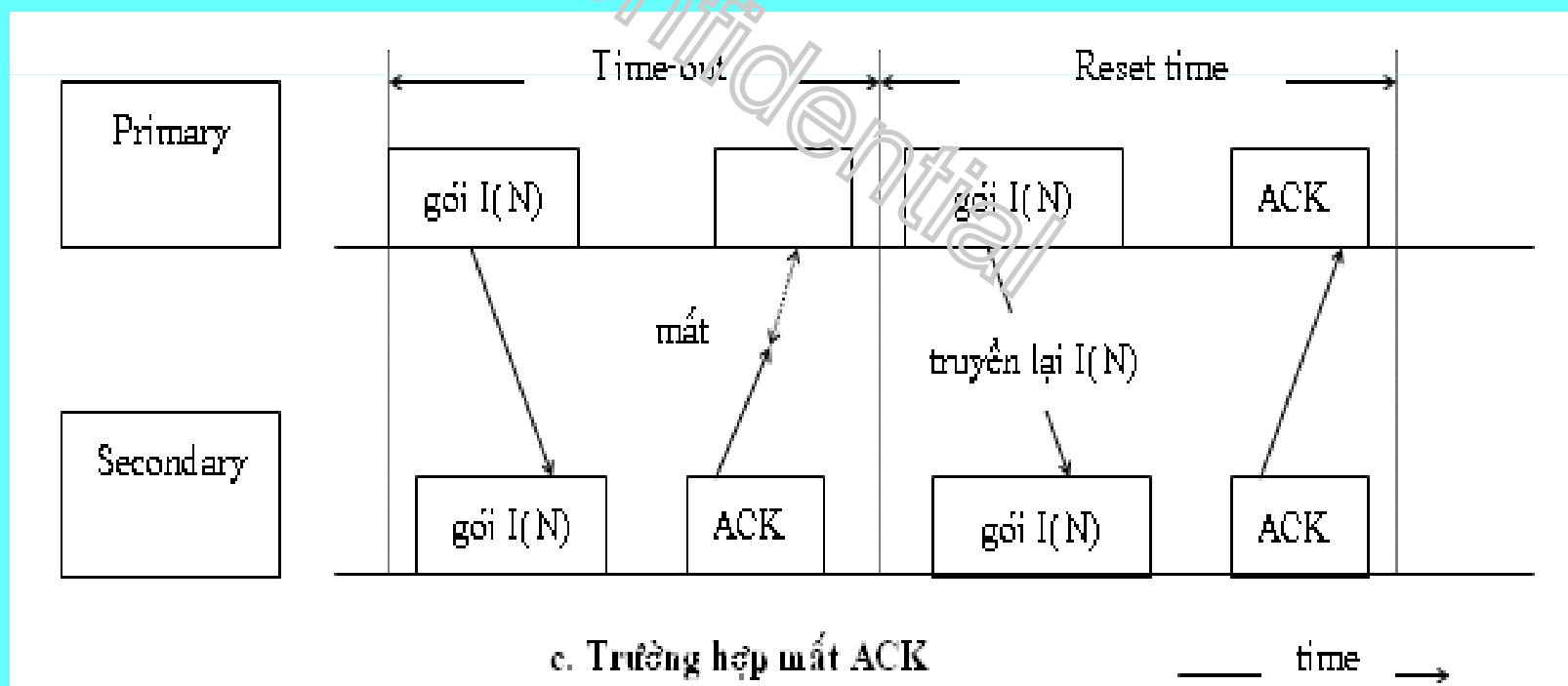


Chương 3: Các giao thức cơ bản

3.1.3 Cơ chế kiểm soát lỗi của Protocol :

a. Phương pháp kiểm lỗi Idle-RQ :

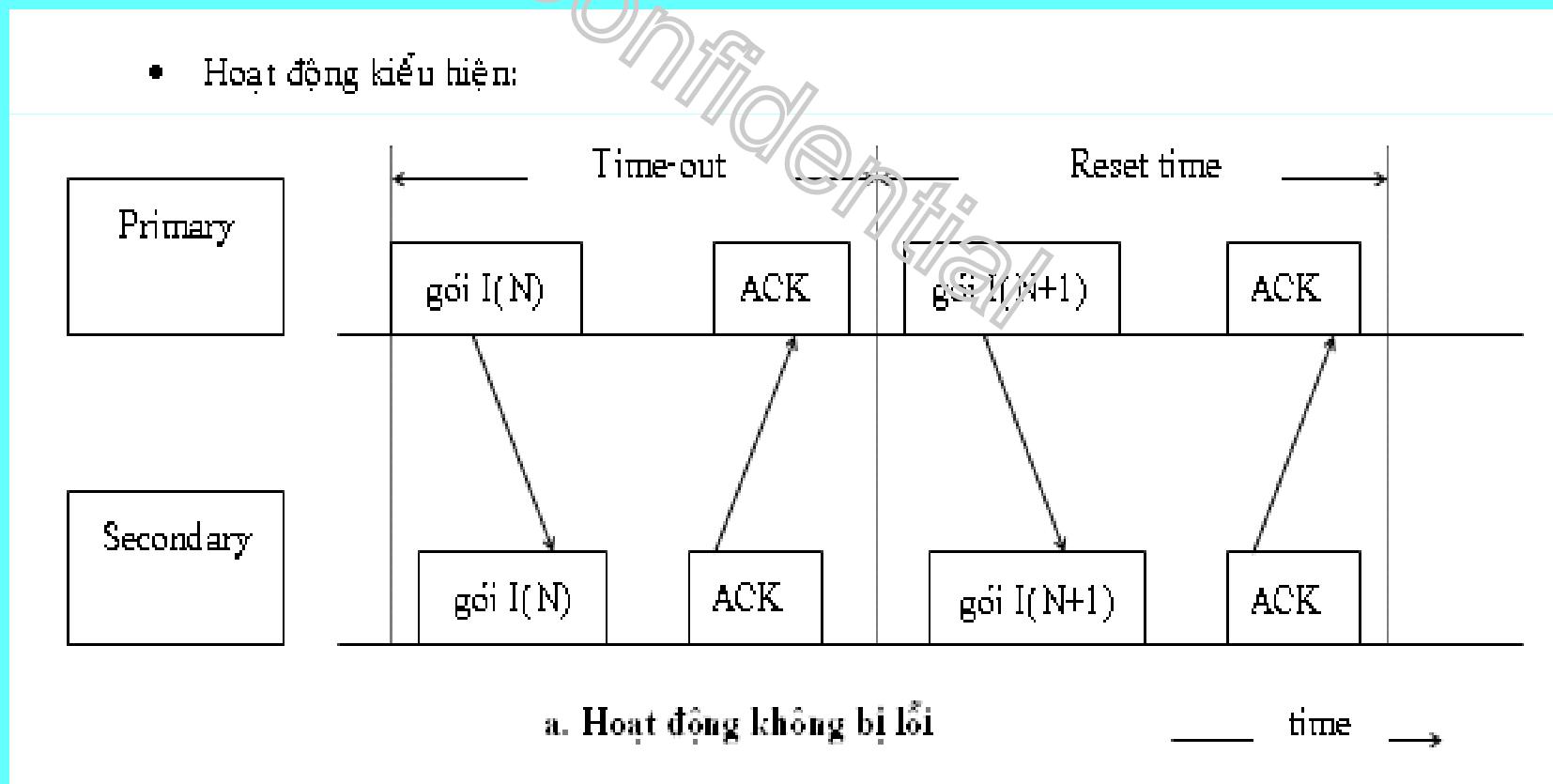
* Hoạt động ~~kiểu~~ ẩn :



Chương 3: Các giao thức cơ bản

3.1.3 Cơ chế kiểm soát lỗi của Protocol :

a. Phương pháp kiểm lỗi Idle-RQ :

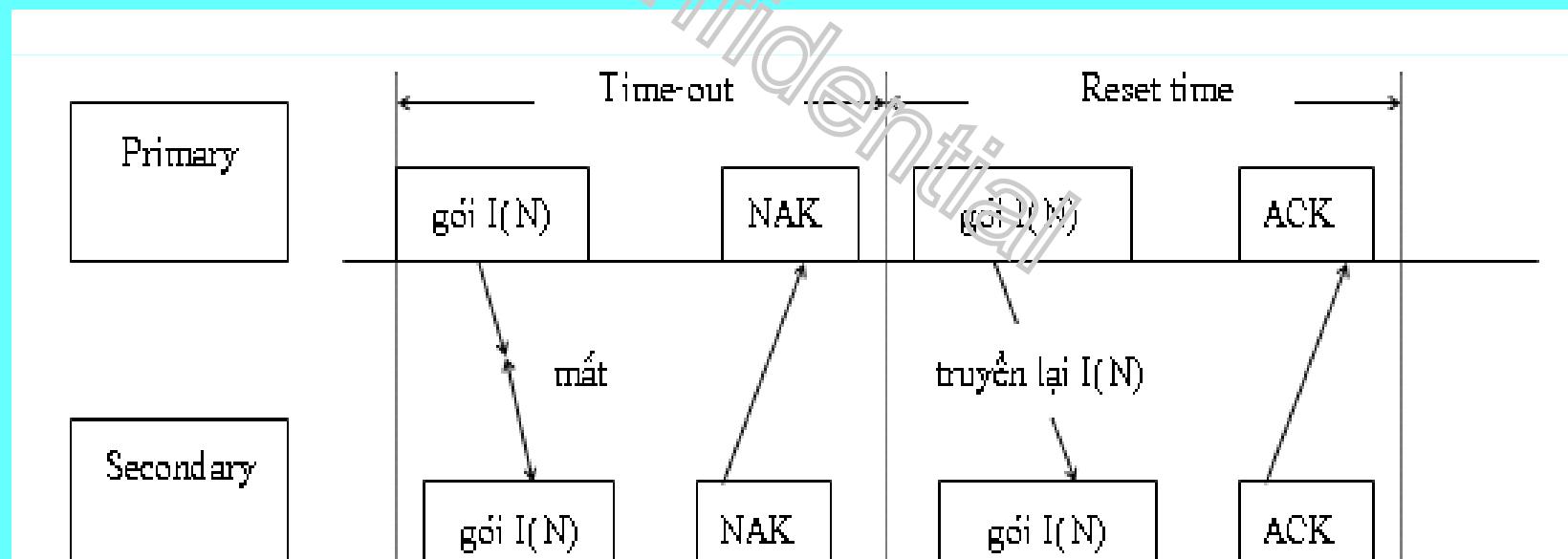


Chương 3: Các giao thức cơ bản

3.1.3 Cơ chế kiểm soát lỗi của Protocol :

a. Phương pháp kiểm lỗi Idle-RQ :

* Hoạt động ~~kiểu~~ hiện :



b. Trường hợp mất gói tin

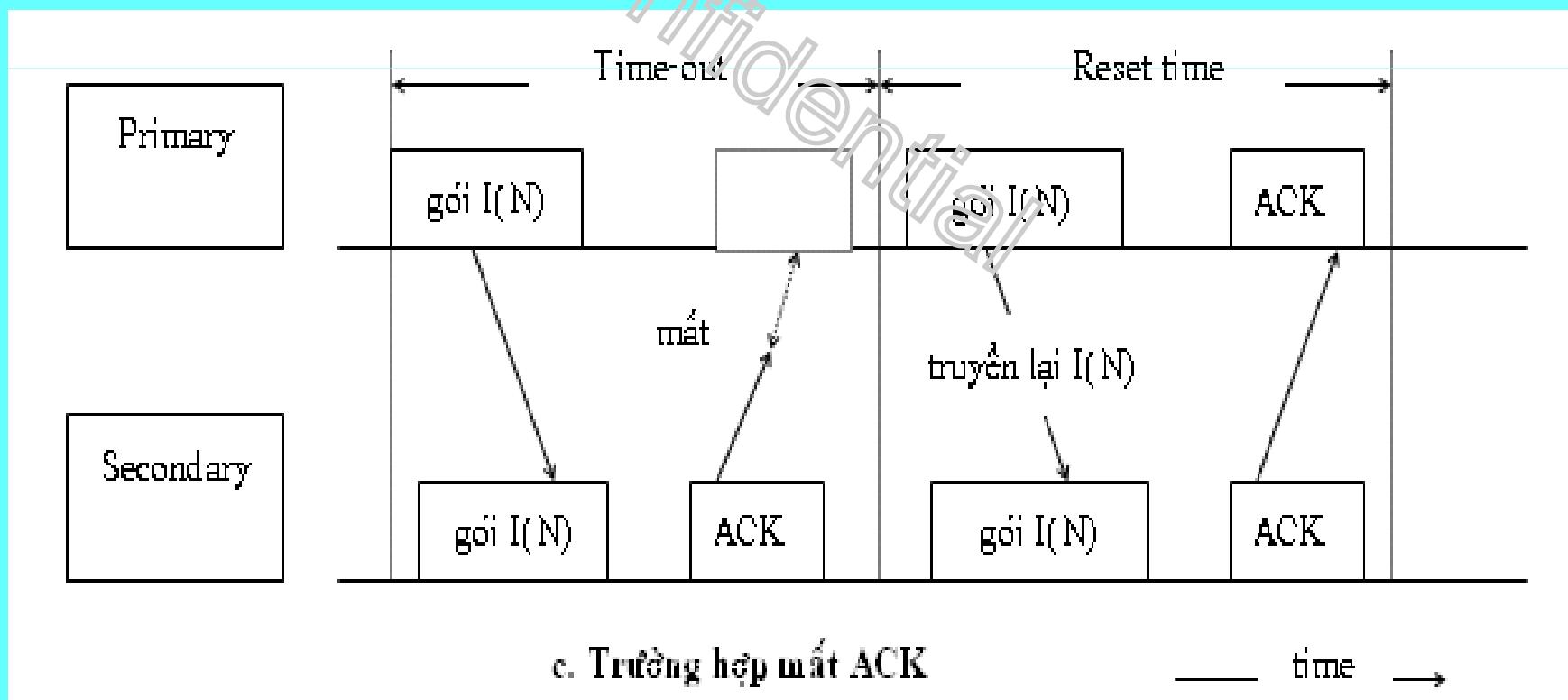
time →

Chương 3: Các giao thức cơ bản

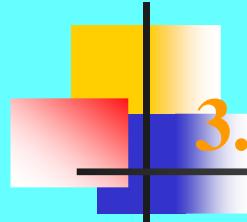
3.1.3 Cơ chế kiểm soát lỗi của Protocol :

a. Phương pháp kiểm lỗi Idle-RQ :

* Hoạt động kiểm hiện :



Chương 3: Các giao thức cơ bản



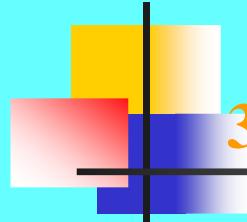
3.1.3 Cơ chế kiểm soát lỗi của Protocol :

a. Phương pháp kiểm lỗi Idle-RQ :

- + Đối với kiểu ~~ẩn~~, nếu gói giữ liệu bị mất, bên nhận sẽ không gửi một thông báo nào, phải làm cho bên phát phải đợi đến thời gian time-out, và nó cũng không biết rằng gói dữ liệu bị mất hay ACK bị mất.
- + Ngược lại đối với kiểu hiện, nếu gói dữ liệu mất, bên nhận sẽ gửi thông báo NAK về ~~cho~~ cho bên phát.
- + Ưu điểm của phương pháp Idle RQ là ít tốn bộ nhớ, nó chỉ cần một vùng RAM đủ chứa một gói thông tin. Nói cách khác Idle RQ điều khiển luồng bằng cửa sổ trượt có cách thước bằng 1.

Phương pháp này được dùng trong các giao thức hướng kí tự (Character Oriented) như giao thức BSC , DDCMP,...

Chương 3: Các giao thức cơ bản

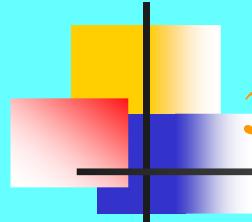


3.1.3 Cơ chế kiểm soát lỗi của Protocol :

b. Phương pháp kiểm lỗi Continuous RQ:

- + Trong khi Idle RQ là kỹ thuật kiểm lỗi theo nguyên tắc truyền gói nào, chờ kết quả gói đó xong mới tiếp tục truyền gói kế tiếp thì Continuous RQ cho phép truyền hàng loạt.
- + Bên nhận làm việc theo nguyên tắc nhận gói nào thì phục đáp gói đó. Như vậy bên nhận làm việc theo nguyên tắc FIFO chứa danh sách ghi nhớ tất cả các gói mà nó đã truyền đi.
- + Khi nhận được phúc đáp thành công của gói nào (ACK) từ phía bên nhận, nó sẽ xoá gói đó ra khỏi danh sách ghi nhớ. Bên nhận cũng lập danh sách ghi nhớ tất cả các gói mà nó đã nhận được.

Chương 3: Các giao thức cơ bản



3.1.3 Cơ chế kiểm soát lỗi của Protocol :

b. Phương pháp kiểm lỗi Continuous RQ:

* Selective Repeat:

+ Hồi đáp kiểu ẩn :

Bên nhận chỉ phục đáp cho những gói mà nó nhận được. Bên phát dựa vào đó để suy đoán là gói nào bị thất lạc trên đường truyền.

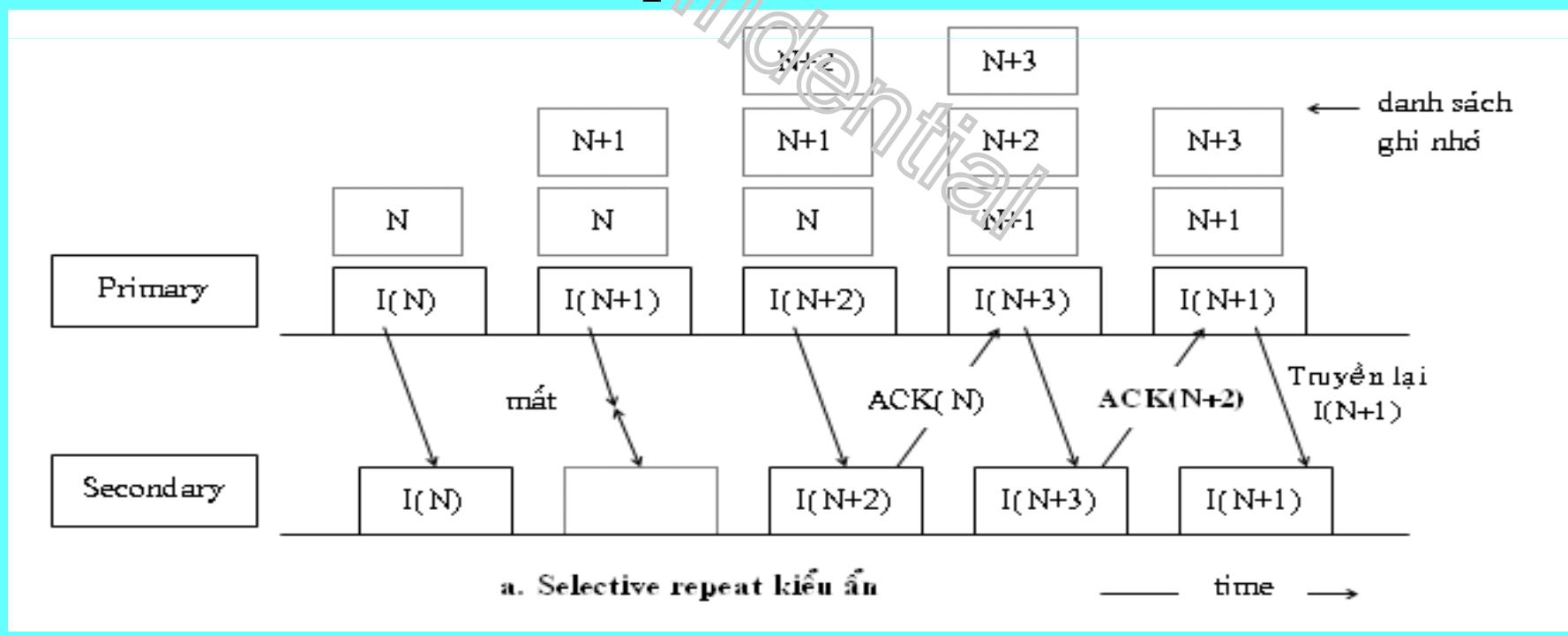
Chương 3: Các giao thức cơ bản

3.1.3 Cơ chế kiểm soát lỗi của Protocol :

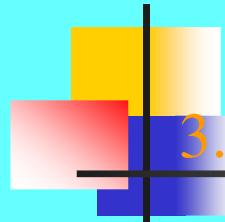
b. Phương pháp kiểm lỗi Continuous RQ:

* Selective Repeat:

+ Hồi đáp kiểu ẩn :



Chương 3: Các giao thức cơ bản



3.1.3 Cơ chế kiểm soát lỗi của Protocol :

b. Phương pháp kiểm lỗi Continuous RQ:

* Selective Repeat:

+ Hồi đáp kiểu hiện :

Đối với kiểu ẩn việc hồi đáp được thực hiện theo nguyên tắc nhận được gói mới phúc đáp mà không quan tâm gói nào bị mất, trong khi kiểu hiện lại quan tâm đến những gói bị mất.

Ví dụ : Nếu bên nhận được hai gói I(1) và I(3) thì nó đoán chắc rằng gói I(2) bị mất và lập tức thông báo về bên truyền rằng NAK(2). Trên đường phản hồi, nếu các gói ACK hoặc NAK bị mất thì bên phát cũng suy đoán được

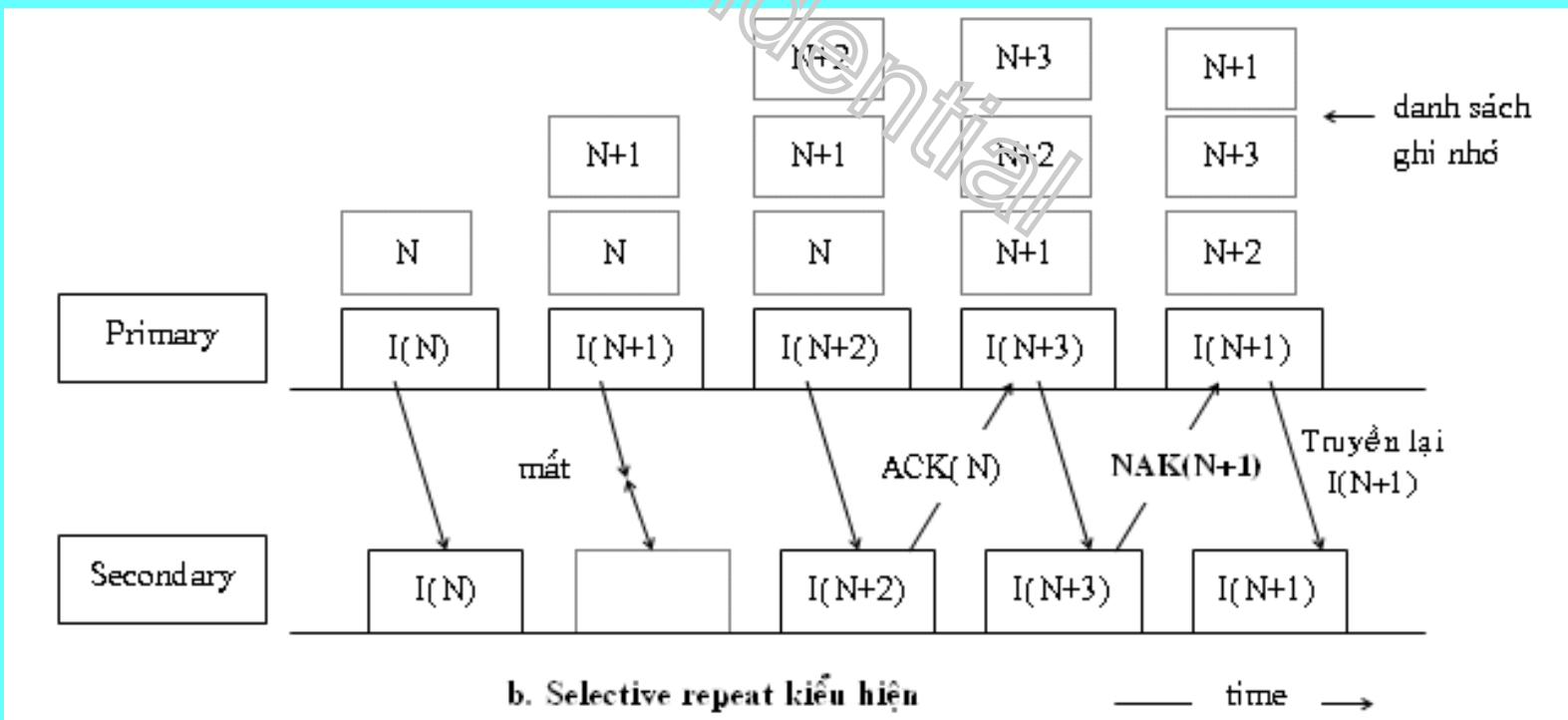
Chương 3: Các giao thức cơ bản

3.1.3 Cơ chế kiểm soát lỗi của Protocol :

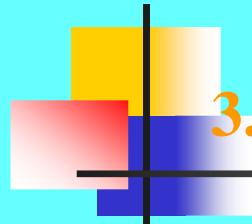
b. Phương pháp kiểm lỗi Continuous RQ:

* Selective Repeat:

+ Hồi đáp kiểu hiện :



Chương 3: Các giao thức cơ bản



3.1.3 Cơ chế kiểm soát lỗi của Protocol :

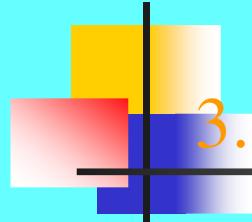
b. Phương pháp kiểm lỗi Continuous RQ:

* Selective Repeat:

+ Phương pháp Selective Repeat dùng kỹ thuật điều khiển luồng cửa sổ trượt với kích thước là K (kể cả cửa sổ nhận),

+ Khi truyền sẽ truyền một lượt K gói nhưng không bắt buộc các gói phải có thứ tự liên tục và khi nhận cũng nhận một lượt K gói.

Chương 3: Các giao thức cơ bản



3.1.3 Cơ chế kiểm soát lỗi của Protocol :

b. Phương pháp kiểm lỗi Continuous RQ:

* Phương pháp Goback-N:

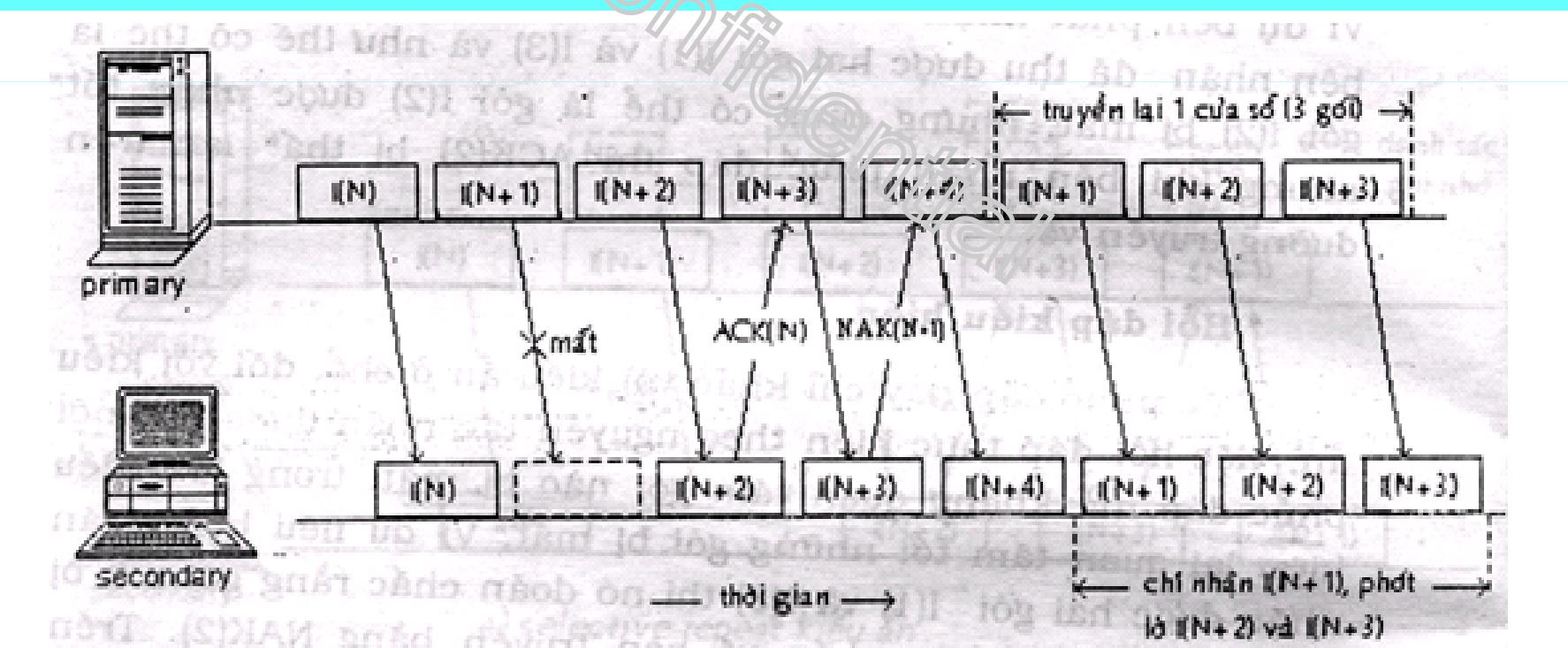
+ Phương pháp này dùng kỹ thuật điều khiển luồng cửa sổ trượt với kích thước cửa sổ gửi là K gói và cửa sổ nhận có kích thước là 1, có nghĩa là khi truyền sẽ truyền một lượt K gói có thứ tự liên tục và khi nhận chỉ nhận một gói.

Chương 3: Các giao thức cơ bản

3.1.3 Cơ chế kiểm soát lỗi của Protocol :

b. Phương pháp kiểm lỗi Continuous RQ:

* Phương pháp Goback-N:



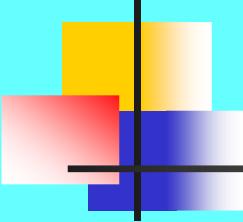
Chương 3:

Các giao thức cơ bản

3.1.3 Cơ chế kiểm soát lỗi của Protocol :

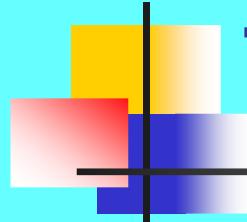
Bảng so sánh 3 phương pháp kiểm soát lỗi :

Protocol	Kích thước của số truy cập	Kích thước của số lỗi
Idle-RQ	1	1
Selective Repeat	K	K
Go-back-N	K	1



Chương 3: Các giao thức cơ bản

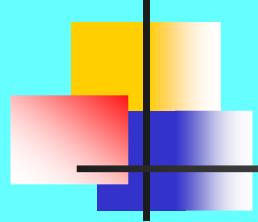
- Họ giao thức TCP/IP hiện nay là giao thức được sử dụng rộng rãi nhất để liên kết các máy tính và các mạng.
- Giao thức TCP/IP thực chất là một họ giao thức cho phép các hệ thống mạng cùng làm việc với nhau thông qua việc cung cấp phương tiện truyền thông liên mạng
- TCP (Transmission Control Protocol) là giao thức thuộc tầng vận chuyển và IP (Internet Protocol) là giao thức thuộc tầng mạng của mô hình OSI.



Tầng mạng – Internet Layer

- Giao thức tầng mạng – Internet Protocol
- Cấu trúc gói tin IP
- Địa chỉ IP
- Dịch vụ DHCP
- Giao thức ICMP

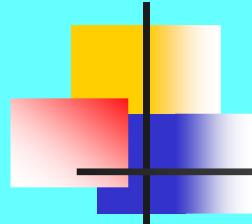
Confidential



Giao thức tầng mạng – Internet Protocol

- Khái niệm cơ bản
- Nguyên lý lưu và chuyển tiếp
- Giao thức IP

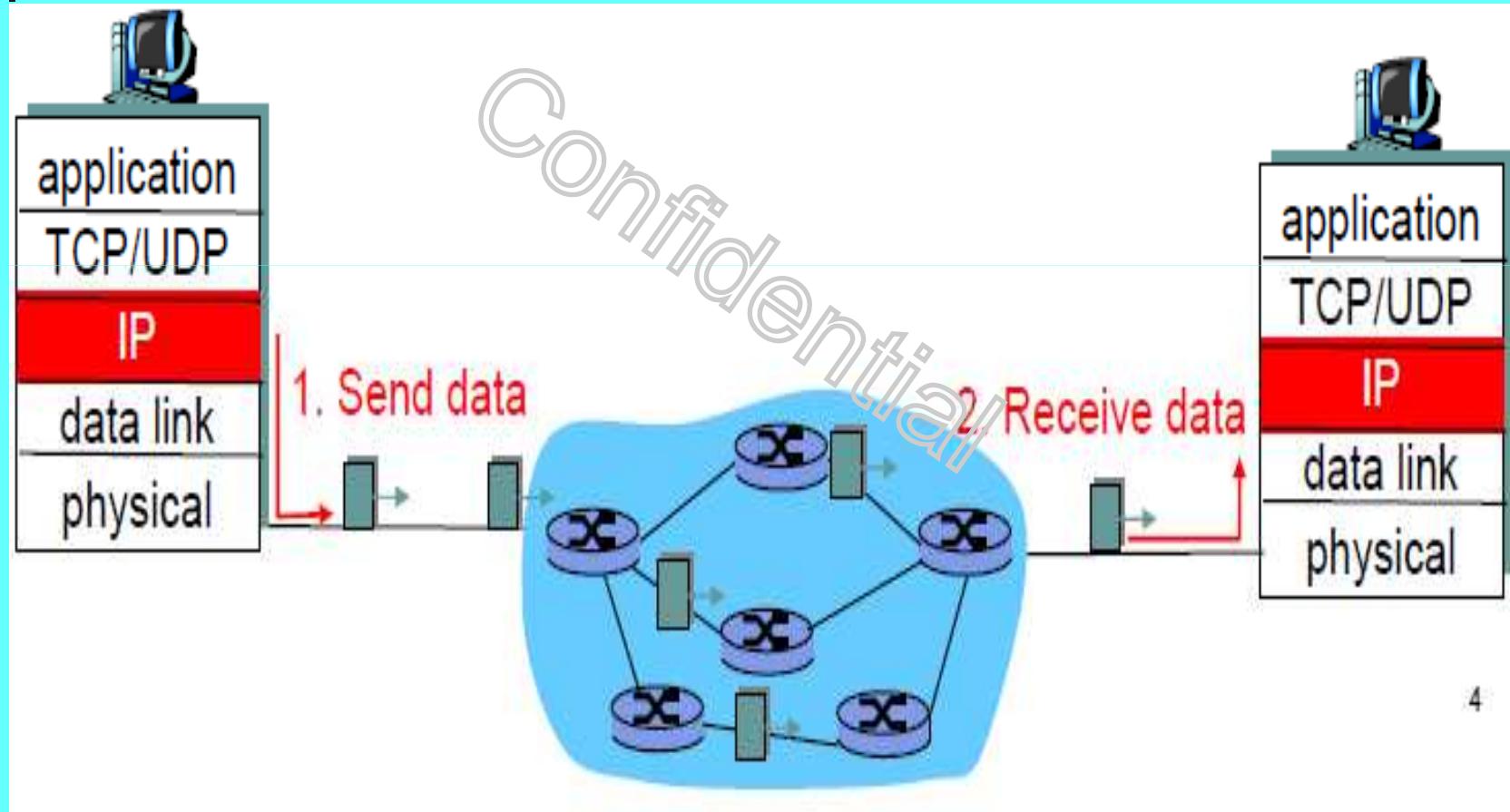
Confidential



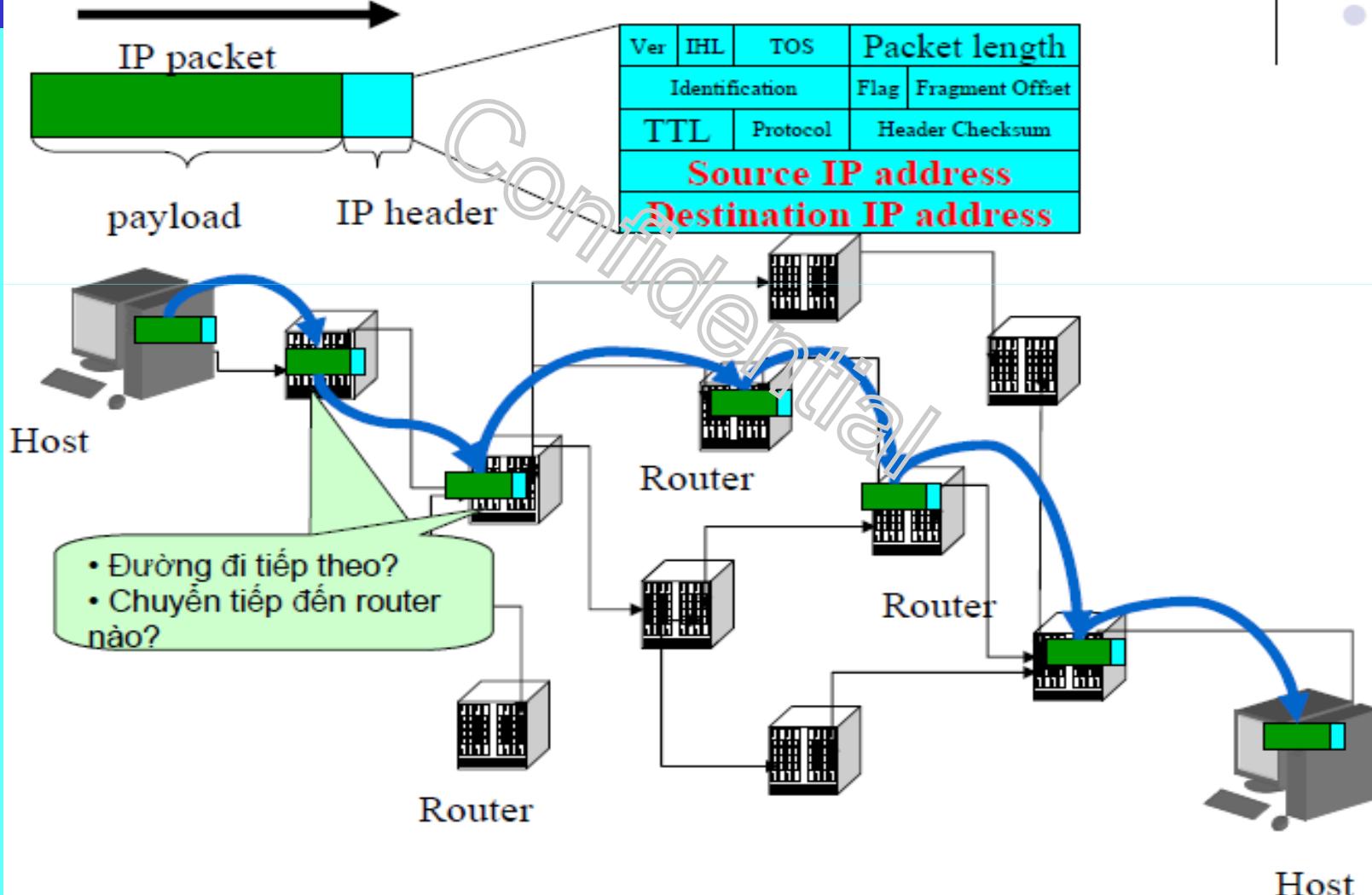
Internet Protocol

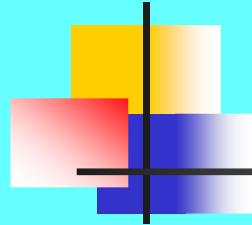
- Là một giao thức tầng mạng
- Hai chức năng cơ bản
 - Chọn đường (*Routing*): Xác định đường đi của gói tin từ nguồn đến đích
 - Chuyển tiếp (*Forwarding*): Chuyển dữ liệu từ đầu vào đến đầu ra của bộ định tuyến (router)

Internet Protocol



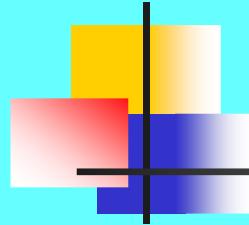
Chọn đường và chuyển tiếp gói tin





Internet Protocol

- Network Layer: Giữa các máy trạm hoặc các bộ định tuyến (Hosts)
- Transport Layer: Giữa các tiến trình trên máy trạm (Processes)



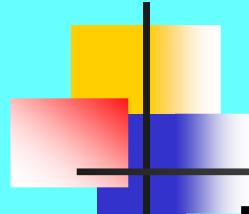
Đặc điểm giao thức IP

- Không tin cậy/Nhanh

- Truyền dữ liệu theo phương thức không tin cậy
- Không có cơ chế phục hồi lỗi
- Khi cần sẽ sử dụng dịch vụ tầng trên để đảm bảo độ tin cậy

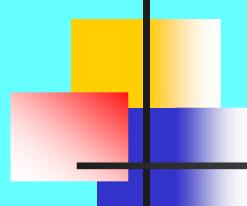
- Giao thức không liên kết

- Các gói tin được xử lý độc lập



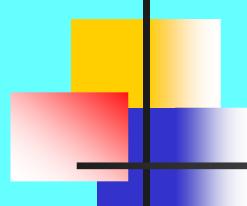
Hoạt động của giao thức IP

- Khi giao thức IP được khởi động nó trở thành một thực thể tồn tại trong máy tính
- Bắt đầu thực hiện những chức năng của mình
- Lúc đó thực thể IP là ~~cấu~~ thành phần của tầng mạng
- Nhận yêu cầu từ các tầng trên nó và gửi yêu cầu xuống các tầng dưới nó.



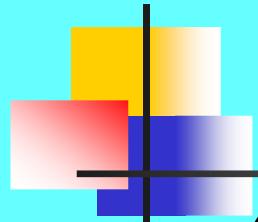
Hoạt động của giao thức IP

- Đối với thực thể IP ở máy nguồn, khi nhận được một yêu cầu gửi từ tầng trên, nó thực hiện các bước sau đây:
 - ❖ Tạo một IP datagram dựa trên tham số nhận được.
 - ❖ Tính checksum và ghép vào header của gói tin.
 - ❖ Ra quyết định chọn đường: hoặc là trạm đích nằm trên cùng mạng hoặc một gateway sẽ được chọn cho chặng tiếp theo.
 - ❖ Chuyển gói tin xuống tầng dưới để truyền qua mạng.



Hoạt động của giao thức IP

- ❑ Đối với router, khi nhận được một gói tin đi qua, nó thực hiện các động tác sau:
 - ❖ Tính checksum, nếu sai thì loại bỏ gói tin
 - ❖ Giảm giá trị tham số Time - to Live, nếu thời gian đã hết thì loại bỏ gói tin
 - ❖ Ra quyết định chọn đường
 - ❖ Phân đoạn gói tin, nếu cần
 - ❖ Kiến tạo lại IP header, bao gồm giá trị mới của các vùng Time - to -Live, Fragmentation và Checksum
 - ❖ Chuyển datagram xuống tầng dưới để chuyển qua mạng



Hoạt động của giao thức IP

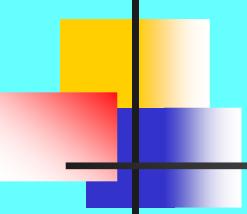
- ❑ Cuối cùng khi một datagram nhận bởi một thực thể IP ở trạm đích, nó sẽ thực hiện bởi các công việc sau:
 - ❖ Tính checksum. Nếu sai thì loại bỏ gói tin
 - ❖ Tập hợp các đoạn của gói tin (nếu có phân đoạn)
 - ❖ Chuyển dữ liệu và các tham số điều khiển lên tầng trên

Cấu trúc gói tin IP

Bit

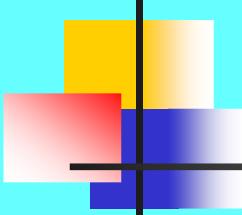
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

VER	IHL	Type of service	Total Length						
Identification		Flags	Fragment offset						
Time to live	Protocol	Header Checksum							
Source address									
Destination Address									
Option + Padding									
Data									

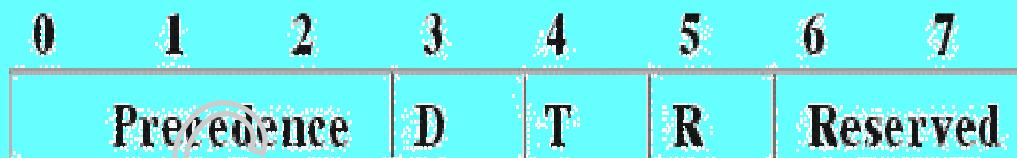


Cấu trúc gói tin IP

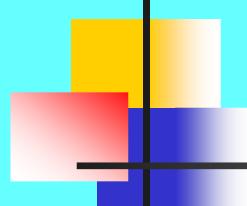
- VER (4 bits): chỉ version hiện hành của giao thức IP hiện được cài đặt
- IHL (4 bits): chỉ độ dài phần đầu (Internet header Length) của gói tin datagram, tính theo đơn vị từ (32 bits).
 - Trường này bắt buộc phải có vì phần đầu IP có thể có độ dài thay đổi tùy ý. Độ dài tối thiểu là 5 từ (20 bytes), độ dài tối đa là 15 từ hay là 60 bytes.
- Type of service (8 bits): đặc tả các tham số về dịch vụ nhằm thông báo cho mạng biết dịch vụ nào mà gói tin muốn được sử dụng
 - Chẳng hạn ưu tiên, thời hạn chậm trễ, năng suất truyền và độ tin cậy.



Cấu trúc gói tin IP

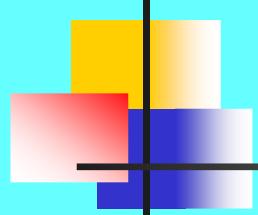


- Precedence (3 bit): chỉ thị về quyền ưu tiên gửi datagram, nó có giá trị từ 0 (gói tin bình thường) đến 7 (gói tin kiểm soát mạng).
- D (Delay) (1 bit): chỉ độ trễ yêu cầu trong đó
 - D = 0 gói tin có độ trễ bình thường
 - D = 1 gói tin độ trễ thấp
- T (Throughput) (1 bit): chỉ độ thông lượng yêu cầu sử dụng để truyền gói tin với lựa chọn truyền trên đường thông suất thấp hay đường thông suất cao.
 - T = 0 thông lượng bình thường
 - T = 1 thông lượng cao
- R (Reliability) (1 bit): chỉ độ tin cậy yêu cầu
 - R = 0 độ tin cậy bình thường
 - R = 1 độ tin cậy cao



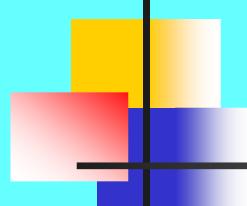
Cấu trúc gói tin IP

- Total Length (16 bits): chỉ độ dài toàn bộ gói tin, kể cả phần đầu tính theo đơn vị byte
- Identification (16 bits): cùng với các tham số khác (như Source Address và Destination Address) tham số này dùng để định danh duy nhất cho một datagram trong khoảng thời gian nó vẫn còn trên liên mạng.



Cấu trúc gói tin IP

- Flags (3 bits): liên quan đến sự phân đoạn (fragment) các datagram
 - Các gói tin khi đi trên đường đi có thể bị phân thành nhiều gói tin nhỏ, trong trường hợp bị phân đoạn thì trường Flags được dùng để điều khiển phân đoạn và tái lắp ghép bó dữ liệu.
 - Tùy theo giá trị của Flags sẽ có ý nghĩa là gói tin sẽ không phân đoạn, có thể phân đoạn hay là gói tin phân đoạn cuối cùng.
- Fragment Offset : cho biết vị trí dữ liệu thuộc phân đoạn tương ứng với đoạn bắt đầu của gói dữ liệu gốc.

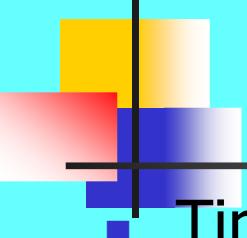


Cấu trúc gói tin IP

- Ý nghĩa cụ thể của trường Flags là:

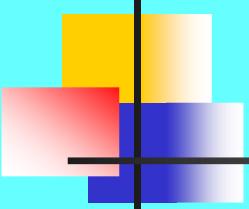


- bit 0: reserved - chưa sử dụng, luôn lấy giá trị 0.
- bit 1: (DF) = 0 (May Fragment) = 1 (Don't Fragment)
- bit 2: (MF) = 0 (Last Fragment) = 1 (More Fragments)
- Fragment Offset (13 bits): chỉ vị trí của đoạn (fragment) ở trong datagram tính theo đơn vị 8 bytes, có nghĩa là phần dữ liệu mỗi gói tin (trừ gói tin cuối cùng) phải chứa một vùng dữ liệu có độ dài là bội số của 8 bytes. Điều này có ý nghĩa là phải nhân giá trị của Fragment offset với 8 để tính ra độ lệch byte.



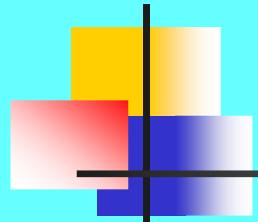
Cấu trúc gói tin IP

- Time to Live (8 bits): qui định thời gian tồn tại (tính bằng giây) của gói tin trong mạng để tránh tình trạng một gói tin bị quẩn trên mạng. Sau đây là 1 số điều cần lưu ý về trường Time To Live:
 - Nút trung gian của mạng không được gởi 1 gói tin mà trường này có giá trị= 0.
 - Một giao thức có thể ấn định Time To Live để thực hiện cuộc ra tìm tài nguyên trên mạng trong phạm vi mở rộng.
 - Một giá trị cố định tối thiểu phải đủ lớn cho mạng hoạt động tốt.
- Protocol (8 bits): chỉ giao thức tầng trên kế tiếp sẽ nhận vùng dữ liệu ở trạm đích (hiện tại thường là TCP hoặc UDP được cài đặt trên IP).
 - Ví dụ: TCP có giá trị trường Protocol là 6, UDP có giá trị trường Protocol là 17



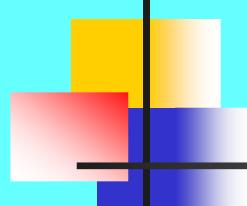
Cấu trúc gói tin IP

- Header Checksum (16 bits): Mã kiểm soát lỗi của header gói tin IP.
- Source Address (32 bits): Địa chỉ của máy nguồn.
- Destination Address (32 bits): địa chỉ của máy đích
- Options (độ dài thay đổi): Khai báo các lựa chọn do người gửi yêu cầu (tùy theo từng chương trình).
- Padding (độ dài thay đổi): Vùng đệm, được dùng để đảm bảo cho phần header luôn kết thúc ở một mốc 32 bits.
- Data (độ dài thay đổi): Trên một mạng cục bộ như vậy, hai trạm chỉ có thể liên lạc với nhau nếu chúng biết địa chỉ vật lý của nhau. Như vậy vấn đề đặt ra là phải thực hiện ánh xạ giữa địa chỉ IP (32 bits) và địa chỉ vật lý (48 bits) của một trạm.



Địa chỉ IP

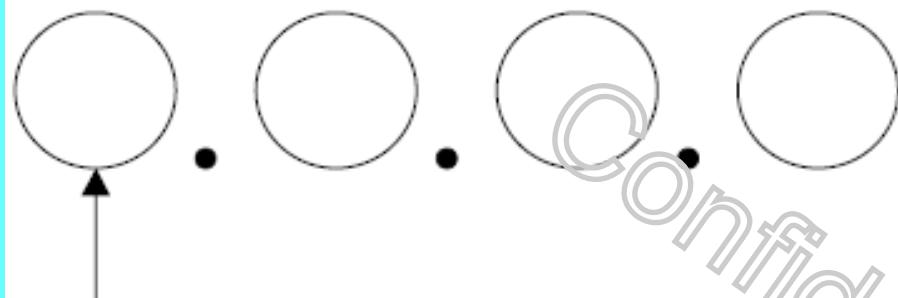
- Để định danh các trạm (host) trong liên mạng người ta dùng địa chỉ IP.
- Mỗi địa chỉ IP có độ dài 32 bits được tách thành 4 vùng (octet), mỗi vùng 1 byte.
- Mục đích của địa chỉ IP là để định danh duy nhất cho một máy tính bất kỳ trên liên mạng.



Địa chỉ IP

- Để địa chỉ không được trùng nhau cần phải có cấu trúc địa chỉ đặc biệt quản lý thống nhất
- Trung tâm thông tin mạng Internet - Network Information Center (NIC) chủ trì phân phối
- NIC chỉ phân địa chỉ mạng (Net ID) còn địa chỉ máy chủ trên mạng đó (Host ID) do các Tổ chức quản lý Internet của từng quốc gia một tự phân phối

Địa chỉ IP



8 bits
0 – 255 integer

Ví dụ:
203.178.136.63
259.12.49.192
133.27.4.27

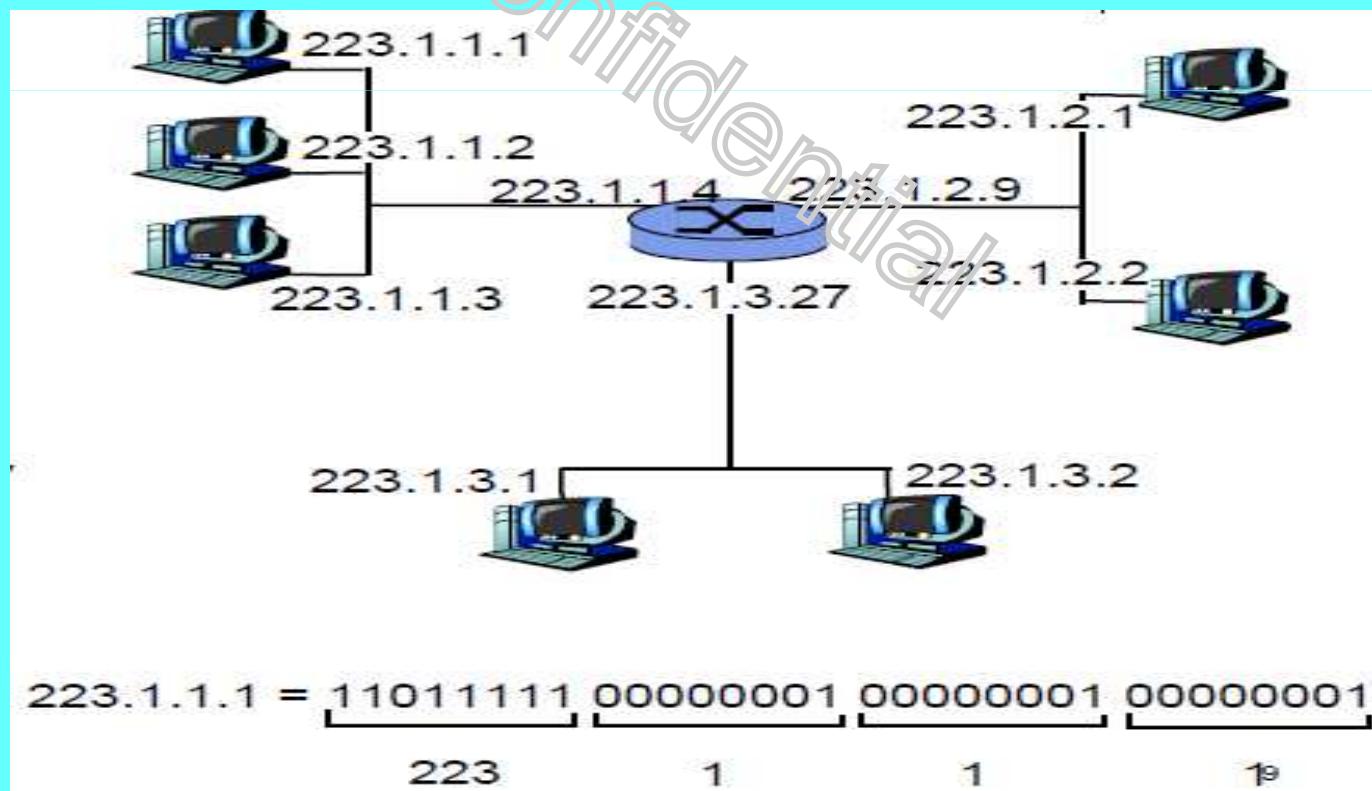
Sử dụng 4 phần 8 bits để miêu tả một địa chỉ 32 bits

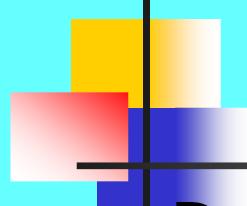
1	1	0	0	1	0	1	1	1	0	1	1	0	0	1	0	1	0	0	1	1	1	1	0	1	1	0	0	1	0	0		
203									178									143					100									

10

Địa chỉ IP

- Mỗi địa chỉ IP được gán cho giao diện
- Địa chỉ IP có tính duy nhất





Địa chỉ IP

- Do tổ chức và độ lớn của các mạng con (subnet) của liên mạng có thể khác nhau
- Người ta chia các địa chỉ IP thành 5 lớp :
 - A, B, C, D và E
- Các bit đầu tiên của byte đầu tiên được dùng để định danh lớp địa chỉ

0 - lớp A

10 - lớp B

110 - lớp C

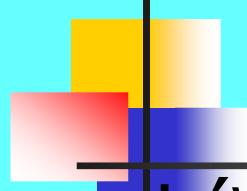
1110 - lớp D

11110 - lớp E

Địa chỉ IP

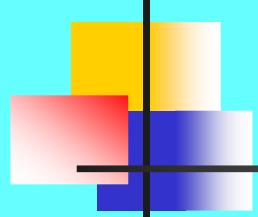
	8bits				8bits				8bits				8bits			
Class A	0	7bit				H	H	H								
Class B	1	0	6bit				N	H	H							
Class C	1	1	0	5bit				N	N	N						
Class D	1	1	1	0	Multicast											
Class E	1	1	1	1	Reserve for future use											

	# of network	# of hosts
Class A	128	2^{24}
Class B	16384	65536
Class C	2^{21}	256



Địa chỉ IP

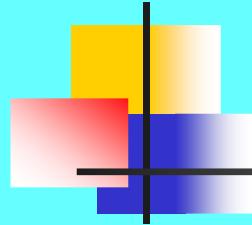
- ❑ Lớp A cho phép định danh tới 128 mạng, với tối đa 16 triệu host trên mỗi mạng.
 - ❑ Lớp này được dùng cho các mạng có số trạm cực lớn.
- ❑ Lớp B cho phép định danh tới 16384 mạng, với tối đa 65536 host trên mỗi mạng.
- ❑ Lớp C cho phép định danh tới 2 triệu mạng, với tối đa 256 host trên mỗi mạng.
 - ❑ Lớp này được dùng cho các mạng có ít trạm.
- ❑ Lớp D dành riêng cho lớp kỹ thuật multicasting.
- ❑ Lớp E được dành những ứng dụng tương lai.



Địa chỉ IP

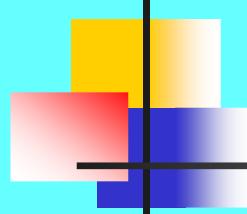
■ Khoảng IP của các lớp

- Lớp A : Từ 0.0.0.0 đến 127.0.0.0
- Lớp B : Từ 128.0.0.0 đến 191.255.0.0
- Lớp C : Từ 192.0.0.0 đến 223.255.255.0
- Lớp D : Từ 224.0.0.0 đến 239.255.255.0
Không phân
- Lớp E : Từ 240.0.0.0 đến 255.0.0.0 Không phân



Các dạng địa chỉ

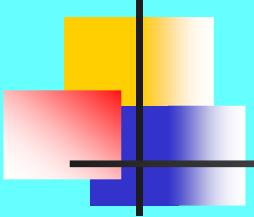
- Địa chỉ mạng
 - Địa chỉ IP gán cho một mạng
- Địa chỉ máy trạm
 - Địa chỉ IP gán cho một card mạng
- Địa chỉ quảng bá
 - Địa chỉ dùng để gửi cho tất cả máy trạm trong mạng
 - Toàn bit 1 phần ứng với địa chỉ máy trạm



Cấu trúc logic

- Địa chỉ IP có 2 phần :
 - Net ID : địa chỉ mạng
 - Host ID : địa chỉ máy trạm trên NetID



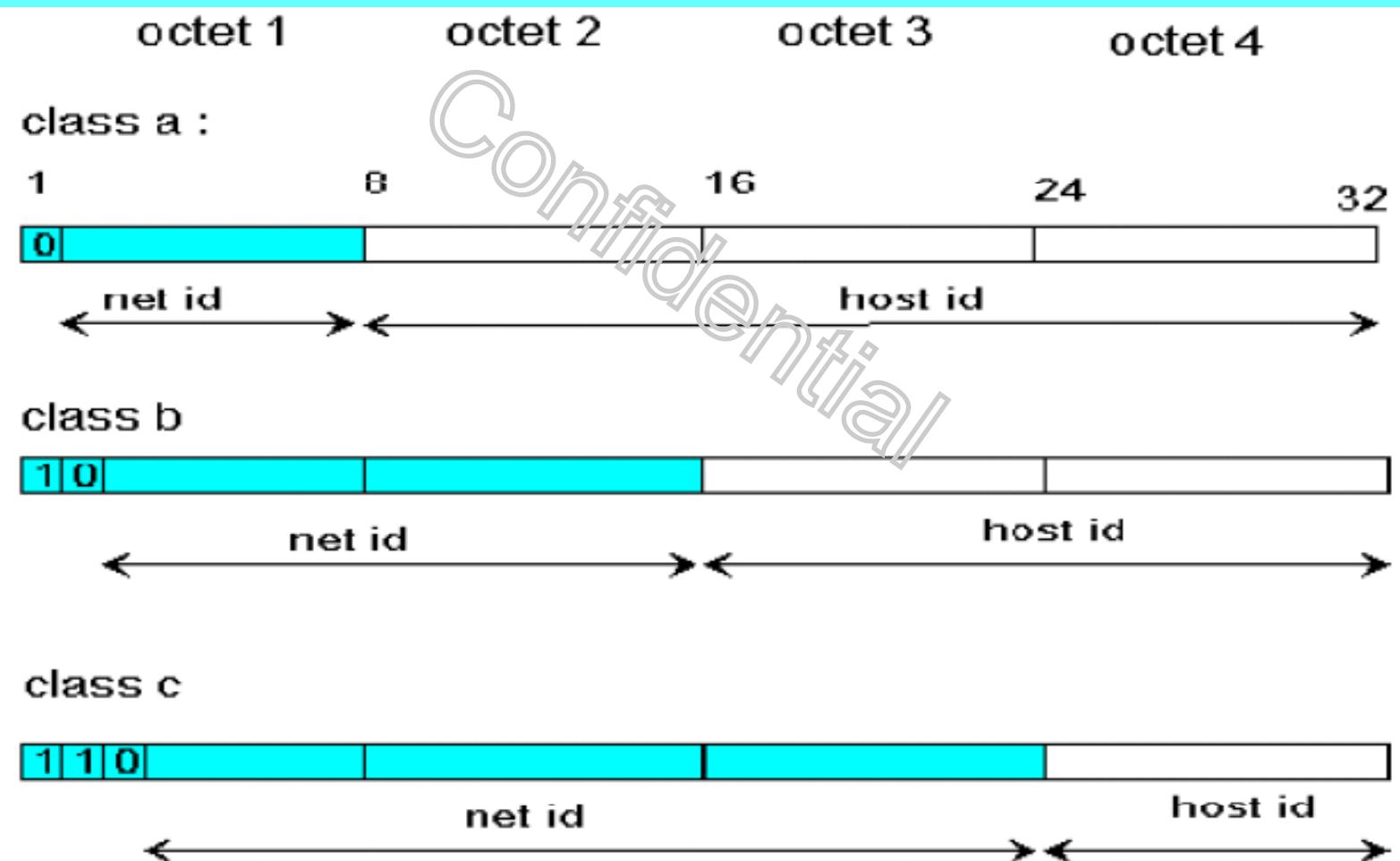


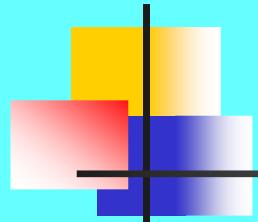
Địa chỉ IP

Cấu trúc của các địa chỉ IP như sau:

- Mạng lớp A: ~~địa chỉ mạng~~ (netid) là 1 Byte và ~~địa chỉ host~~ (hostid) là 3 byte.
- Mạng lớp B: ~~địa chỉ mạng~~ (netid) là 2 Byte và ~~địa chỉ host~~ (hostid) là 2 byte.
- Mạng lớp C: ~~địa chỉ mạng~~ (netid) là 3 Byte và ~~địa chỉ host~~ (hostid) là 1 byte.

Địa chỉ IP





Địa chỉ IP

- + Một địa chỉ có hostid = 0 được dùng để hướng tới mạng định danh bởi vùng netid.
- + Ngược lại, một địa chỉ có vùng hostid gồm toàn số 1 được dùng để hướng tới tất cả các host nối vào mạng netid,
- + Nếu vùng netid cũng gồm toàn số 1 thì nó hướng tới tất cả các host trong liên mạng

Địa chỉ IP

Ví dụ :

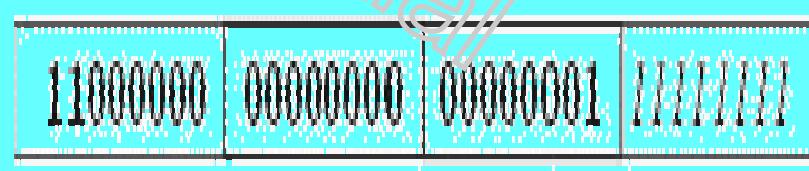
- 128.3.2.3
- 192.0.1.255
- 192.168.3.17
- 19.45.2.3
- 86.2.3.5



= 128.3.2.3 (lớp B)

netid = 128.3

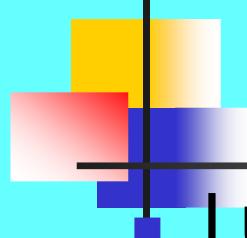
hostid = 2.3



= 192.0.1.255 (lớp C)

netid = 192.0.1

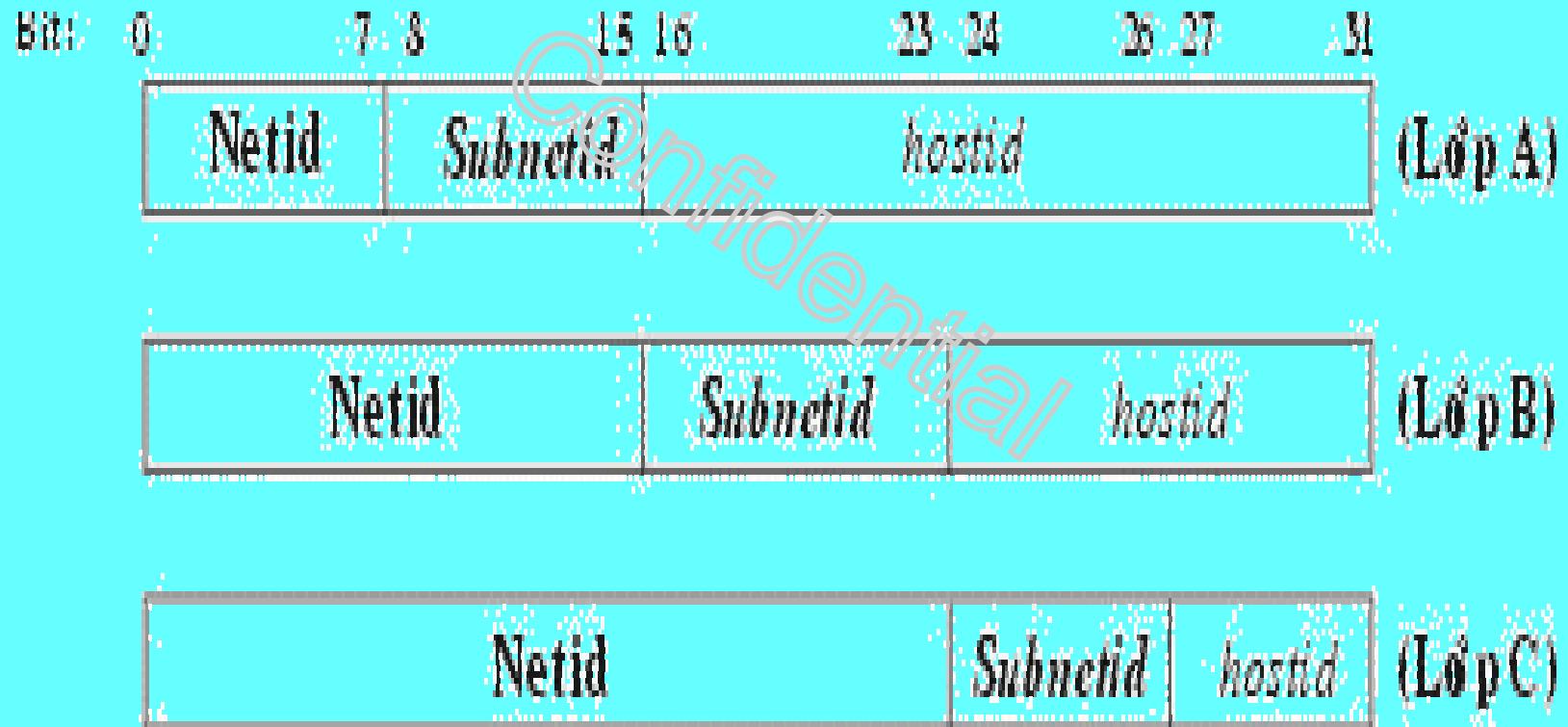
hostid = 255 → bao gồm
tất cả các host



Địa chỉ IP

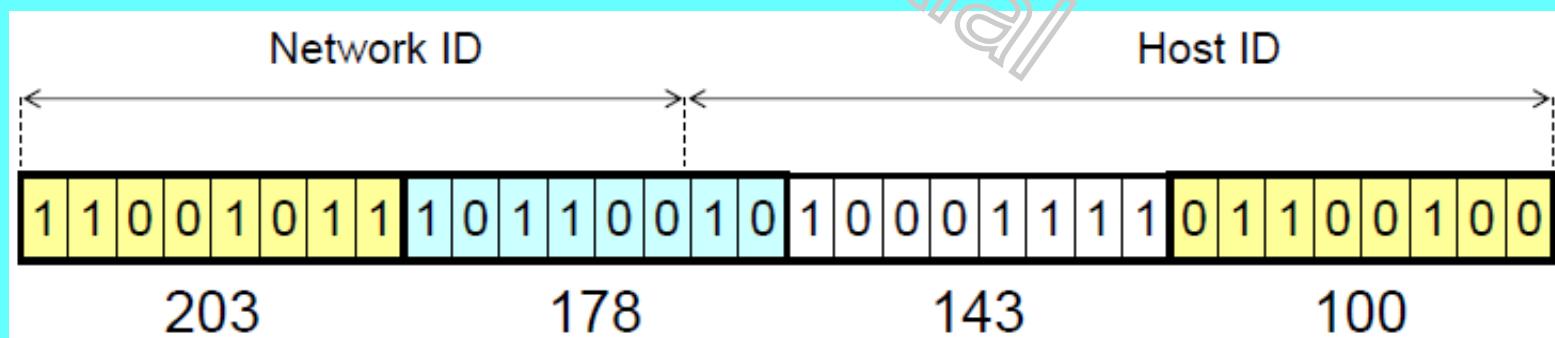
- Lưu ý rằng các địa chỉ IP dùng để định danh các host và mạng ở tầng mạng của mô hình OSI
 - Chúng không phải là các địa chỉ vật lý (hay địa chỉ MAC) của các trạm trên một mạng cục bộ (Ethernet, Token Ring,...).
 - Trong nhiều trường hợp, một mạng có thể được chia thành nhiều mạng con (subnet), lúc đó có thể đưa thêm các vùng subnetid để định danh các mạng con.
 - Vùng **subnetid** được lấy từ vùng **hostid**, cụ thể đối với lớp A, B, C như ví dụ sau:

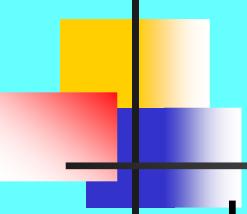
Địa chỉ IP



Địa chỉ IP

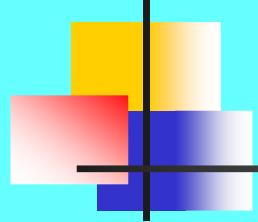
- Làm sao để biết phần nào cho máy trạm, phần nào cho mạng
 - Phân lớp địa chỉ
 - Không phân lớp





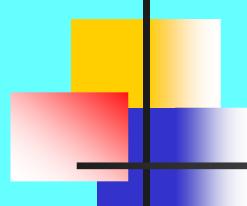
Địa chỉ IP

- Lãng phí không gian địa chỉ
 - Việc phân chia thành các lớp hạn chế việc sử dụng toàn bộ không gian địa chỉ
- Nhằm hạn chế việc sử dụng lãng phí không gian địa chỉ (CIDR :Classless Inter-Domain Routing)
 - Phần địa chỉ mạng sẽ có độ dài bất kỳ
 - Dạng địa chỉ : a.b.c.d/x (x: là mặt nạ mạng, chính là số bit trong phần ứng với địa chỉ mạng)
 - Với cách địa chỉ hóa theo CIDR, địa chỉ IP và mặt nạ mạng luôn phải đi cùng nhau



Mặt nạ mạng

- Mặt nạ mạng chia địa chỉ IP thành 2 phần :
 - Phần ứng với máy trạm
 - Phần ứng với mạng
- Dùng toán tử AND
 - Tính địa chỉ mạng
 - Tính khoảng địa chỉ IP

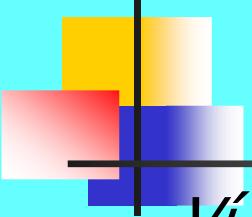


Mặt nạ mạng

- Là một dãy số 32 bit dùng để nhận diện ra netID
 - $\text{netID} = \text{IP address} \text{ AND } \text{netmask}$
- Chuẩn thì :
 - Lớp A có subnetmask: 255.0.0.0
 - Lớp B có subnetmask 255.255.0.0
 - Lớp C có subnetmask 255.255.255.0

Mặt nạ mạng

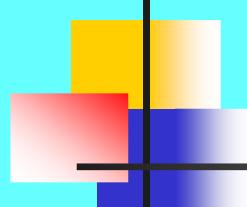




Mặt nạ mạng

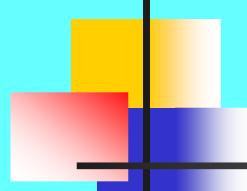
Ví dụ:

- ▶ IP 203.165.7.123 netmask 255.255.255.0
- ▶ → netID= IP 203.165.7.0
- ▶ IP 20.162.7.123 thì netmask ????
- ▶ → netID= IP 20.0.0.0
- ▶ IP 153.162.7.123 netmask ????
- ▶ → netID= IP 153.162.0.0
- ▶ IP 203.162.7.123 netmask 255.255.0.0
- ▶ → netID= IP ????
- ▶ IP 203.162.0.123 netmask 255.255.255.0
- ▶ → netID= IP ????



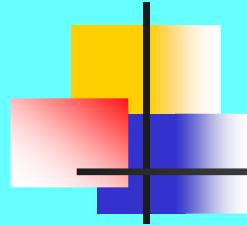
Bảng tóm tắt

Lớp	netID	netmask	Số máy/mạng	Số mạng
A	x.0.0.0	255.0.0.0	256*256*256	128
B	x.x.0.0	255.255.0.0	256*256	64*256
C	x.x.x.0	255.255.255.0	256	32*256*256



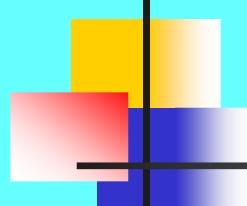
Các địa chỉ đặc biệt

- ▶ Địa chỉ loopback là các địa chỉ có **127.x.x.x**
- ▶ Địa chỉ **broadcast** là địa chỉ mạng sẽ dùng để quảng bá mạng mình cho các mạng khác biết.
 - ▶ Mục đích giúp cho các router cập nhật bảng định tuyến
- ▶ Địa chỉ broadcast được quy định là địa chỉ cuối cùng của một mạng
- ▶ Ví dụ: 167.8.5.0 thì broadcast là 167.8.255.255
192.45.67.1
154.63.72.5



Chú ý:

- Các máy cùng mạng sẽ có cùng NETID, subnetmask và khác host ID
- Hai máy khác mạng hoặc khác NET ID hoặc khác subnet mask

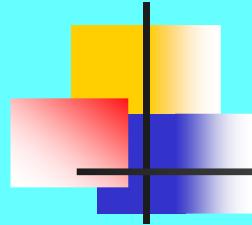


Private address

- ▶ Là vùng địa chỉ cho phép cấu hình cục bộ mà không cần phải mua
- ▶ Không được sử dụng trên internet để cấp phát

Confidential

Lớp	Khoảng mạng	Số mạng
A	10.0.0.0	1
B	172.16.0.0-172.31.0.0	16
C	192.168.x.0	256

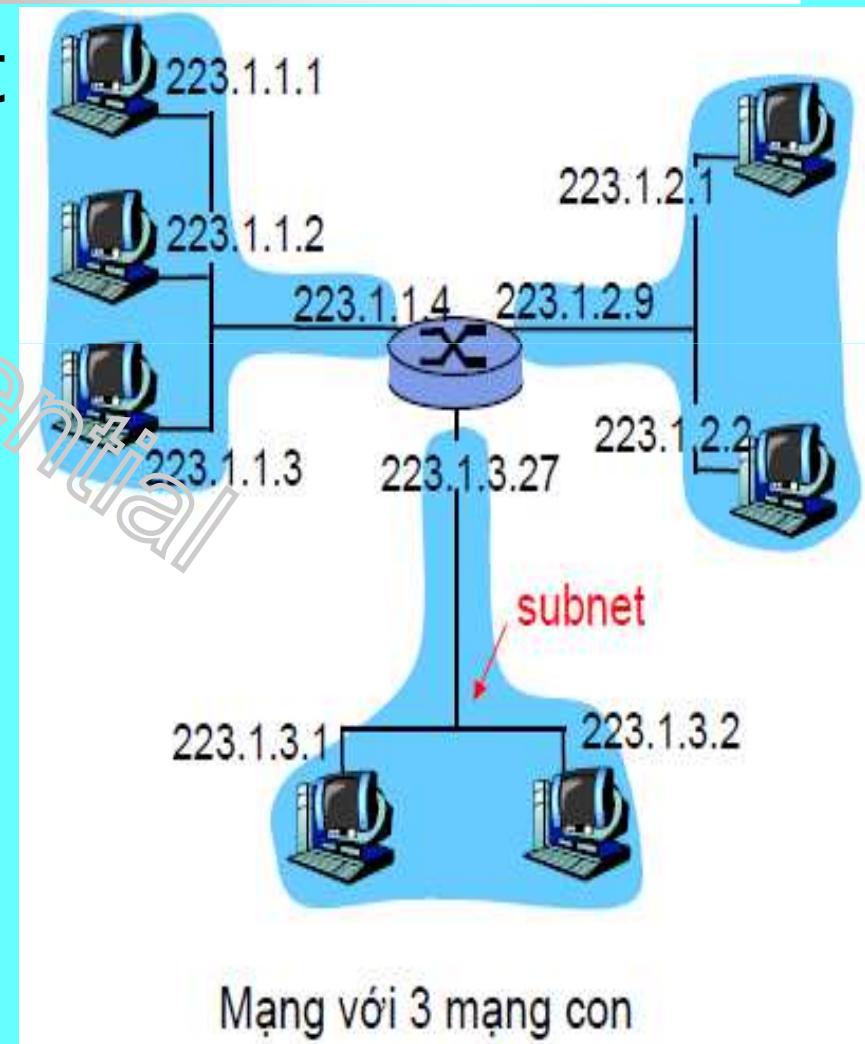


Mạng con- Subnet

- Một NETID thật trên internet phải mua
- Do nhu cầu chia tách mạng con trong một đơn vị
- Số máy trên một mạng quá nhiều nên cần chia nhỏ mạng → nhiều mạng con

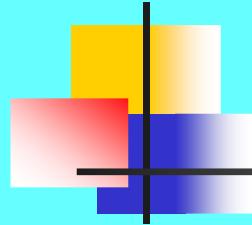
Mạng con - Subnet

- Là một phần của một mạng nào đó
 - ISP thường được gán một khối địa chỉ IP
 - Một vài mạng con sẽ được tạo ra
- Tạo subnet như thế nào?
 - Sử dụng một mặt nạ mạng dài hơn

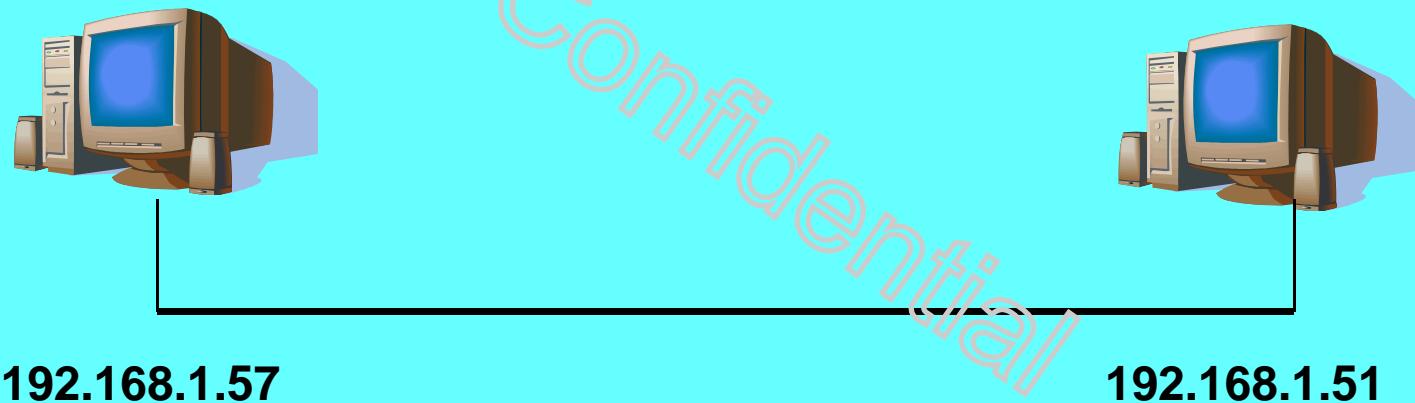


Mạng con - Subnet

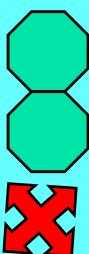


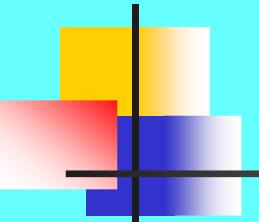


Mạng con - Subnet



255.255.255.0
255.255.255.192
255.255.255.248





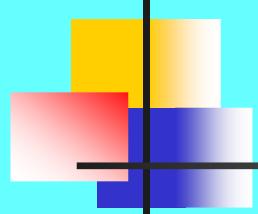
Mạng con - Subnet

- 0... 10000000
- 1... 11000000
- 00... 00000000
- 01... 00100000
- 10... 01000000
- 11... 01100000

Confidence

000...
001...
010...
011...
100...
101...
110...
111...

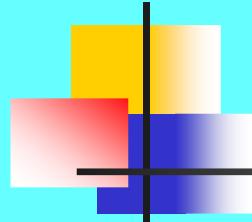
11100000



Mạng con - Subnet

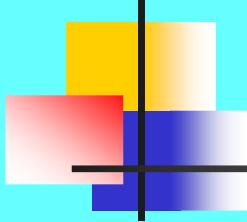
Cách chia :

- Có m bit còn dư
- Mượn n bit
- Vậy có thể có 2^n mạng, $2^m - 2$ mạng được sử dụng
- Số bù là $m < 8 \rightarrow (2^{m-n}) \text{ mod } 256$
- $m > 8 \rightarrow ((2^{m-n})/256) \text{ mod } 256$ (***)
- Số trong subnetmask : $(2^m - 2^{m-n}) \text{ mod } 256$
 - Nếu $n < 8 \rightarrow x$
 - Nếu $n > 8 \rightarrow 255.x$
 - Nếu $n > 16 \rightarrow 255.255.x$



Mạng con - Subnet

- ▶ Số bù dùng để ghi khoảng mạng
- ▶ Vd : số bù là 32 của mạng 172.16.0.0
- ▶ Mạng 1 : 172.16.0.0
 172.16.31.255
- ▶ Mạng 2 : 172.16.32.0
 172.16.63.255
- ▶ Mạng 3 : 172.16.64.0
 172.16.95.255
- ▶ Mạng 4 : 172.16.96.0
- ▶

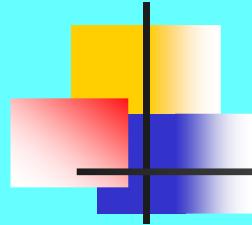


Mạng con - Subnet

- Số máy trên một mạng con là:

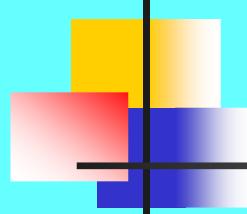
$2^{m-n} - 2$

Confidential



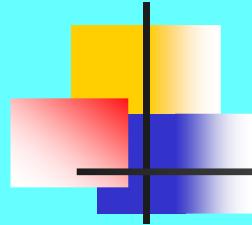
Mạng con - Subnet

- Mượn 1 bít có 2 mạng → số bù là 128 → 128
- Mượn 2 bít ~~có~~ 4 mạng → số bù là 64 → 192
- Mượn 3 bít có 8 mạng → số bù là 32 → 224
- Mượn 4 bít có 16 mạng → số bù là 16 → 240
- Mượn 5 bít có 32 mạng → số bù là 8 → 248
- Mượn 6 bít có 64 mạng → số bù là 4 → 252
- Mượn 7 bít có 128 mạng → số bù là 2 → 254
- Mượn 8 bít có 256 mạng → số bù là 1 → 255



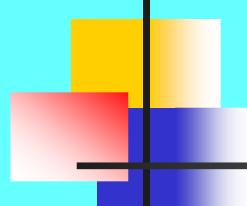
Gán địa chỉ IP

- Do người quản trị gán trực tiếp
 - Windows: Control panel->network->configuration->Tcp/ip->properties
- Sử dụng dịch vụ DHCP (Dynamic Host Configuration Protocol)
 - Plug and Play



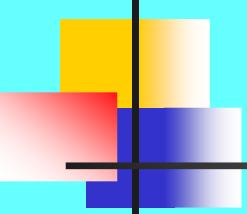
Quản lý địa chỉ IP

- Một mạng con lấy địa chỉ IP từ đâu?
 - Chia ra từ không gian địa chỉ ISP
- ISP lấy địa chỉ IP từ đâu?
- ICANN : Internet Corporation for Assigned Names and Numbers
 - Cấp phát địa chỉ
 - Quản DNS,..



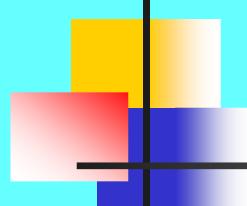
DHCP (Dynamic Host Configuration Protocol)

- Mỗi thiết bị trên mạng có dùng bộ giao thức TCP/IP đều phải có một địa chỉ IP hợp lệ, phân biệt
- Để hỗ trợ cho vấn đề theo dõi và cấp phát các địa chỉ IP được chính xác, đã phát triển ra giao thức DHCP (Dynamic Host Configuration Protocol)
- Dịch vụ DHCP này cho phép chúng ta cấp động các thông số cấu hình mạng cho các máy trạm (client)



DHCP (Dynamic Host Configuration Protocol)

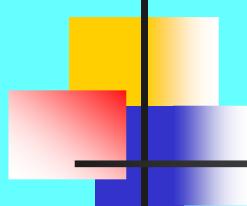
- Cơ chế sử dụng các thông số mạng được cấp phát động có ưu điểm hơn so với cơ chế khai báo tĩnh các thông số mạng như:
 - Khắc phục được tình trạng đụng địa chỉ IP và giảm chi phí quản trị cho hệ thống mạng.
 - Giúp cho các nhà cung cấp dịch vụ (ISP) tiết kiệm được số lượng địa chỉ IP thật (Public IP).
 - Phù hợp cho các máy tính thường xuyên di chuyển qua lại giữa các mạng.
 - Kết hợp với hệ thống mạng không dây (Wireless) cung cấp các điểm Hotspot như: nhà ga, sân bay, trường học,...



HOẠT ĐỘNG CỦA GIAO THỨC DHCP

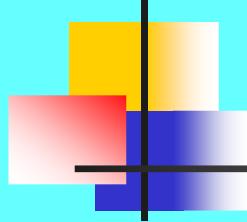
Giao thức **DHCP** làm việc theo mô hình **client/server**. Quá trình tương tác giữa **DHCP client** và **server** diễn ra theo các bước sau:

- Khi máy **client** khởi động, máy sẽ gửi **broadcast** gói tin **DHCP DISCOVER**, yêu cầu một **server** phục vụ mình.
- Gói tin này cũng chứa địa chỉ **MAC** của máy **client**.



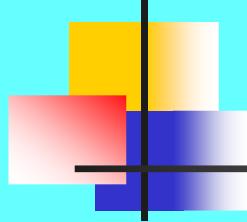
HOẠT ĐỘNG CỦA GIAO THỨC DHCP

- Các máy **Server** trên mạng khi nhận được gói tin yêu cầu đó, nếu còn khả năng cung cấp địa chỉ **IP**, đều gửi lại cho máy **Client** gói tin **DHCPOFFER**
 - Nhằm đề nghị cho thuê một địa chỉ **IP** trong một khoảng thời gian nhất định, kèm theo là một **subnet mask** và địa chỉ của **Server**.
 - **Server** sẽ không cấp phát địa chỉ **IP** vừa đề nghị cho những **Client** khác trong suốt quá trình thương thuyết.



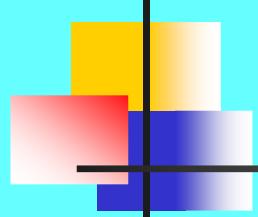
Hoạt động của giao thức DHCP

- Máy **Client** sẽ lựa chọn một trong những lời đề nghị (**DHCPOFFER**) và gửi **broadcast** lại gói tin **DHCPREQUEST** chấp nhận lời đề nghị đó.
 - Điều này cho phép các lời đề nghị không được chấp nhận sẽ được các **Server** rút lại và dùng để cấp phát cho **Client** khác.



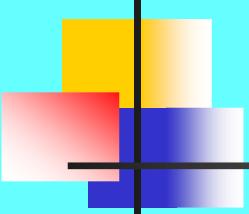
Hoạt động của giao thức DHCP

- Máy **Server** được **Client** chấp nhận sẽ gửi ngược lại một gói tin **DHCPACK** như là một lời xác nhận
- Cho biết là địa chỉ IP đó, **subnet mask** đó và thời hạn cho ~~sử dụng~~ đó sẽ chính thức được áp dụng.
 - Ngoài ra **Server** còn gửi kèm theo những thông tin cấu hình bổ sung như địa chỉ của **gateway** mặc định, địa chỉ **DNS Server**, ...



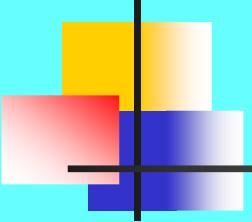
Các giao thức điều khiển

- Để mạng với giao thức IP hoạt động được tốt người ta cần một số giao thức bổ sung
- *Giao thức ICMP (Internet Control Message Protocol): Giao thức này thực hiện truyền các thông báo điều khiển (báo cáo về các tình trạng lỗi trên mạng...) giữa các gateway hoặc một nút của liên mạng.*
 - Tình trạng lỗi có thể là: một gói tin IP không thể tới đích của nó, hoặc một router không đủ bộ nhớ đệm để lưu và chuyển một gói tin IP
 - Một thông báo ICMP được tạo và chuyển cho IP. IP sẽ "bọc" (encapsulate) thông báo đó với một IP header và truyền đến cho router hoặc trạm đích



Các giao thức điều khiển

- *Giao thức ARP (Address Resolution Protocol):* Ở đây cần lưu ý rằng các địa chỉ IP được dùng để định danh các host và mạng ở tầng mạng của mô hình OSI, và chúng không phải là các địa chỉ vật lý.
 - Trên một mạng cục bộ hai trạm chỉ có thể liên lạc với nhau nếu chúng biết địa chỉ vật lý của nhau.
 - Vấn đề đặt ra là phải tìm được ảnh xem giữa địa chỉ IP (32 bits) và địa chỉ vật lý của một trạm.
 - Giao thức ARP đã được xây dựng để tìm địa chỉ vật lý từ địa chỉ IP khi cần thiết.
- *Giao thức RARP (Reverse Address Resolution Protocol):* Là giao thức ngược với giao thức ARP.
 - Giao thức RARP được dùng để tìm địa chỉ IP từ địa chỉ vật lý.



Tổng quan ICMP

- IP là giao thức không tin cậy, không liên kết
 - Thiếu các cơ chế hỗ trợ và kiểm soát lỗi
- ICMP được sử dụng ở tầng mạng để trao đổi thông tin
 - Báo lỗi gói tin không đến được máy trạm, một mạng, một cổng, một giao thức
 - Thông điệp phản hồi

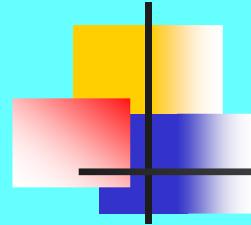
Tổng quan ICMP

- Cũng là giao thức tầng mạng, “phía trên” IP
 - Thông điệp ICMP chứa trong các gói tin IP
- ICMP message : Type, Code cùng với 8 bytes đầu tiên của gói tin IP bị lỗi



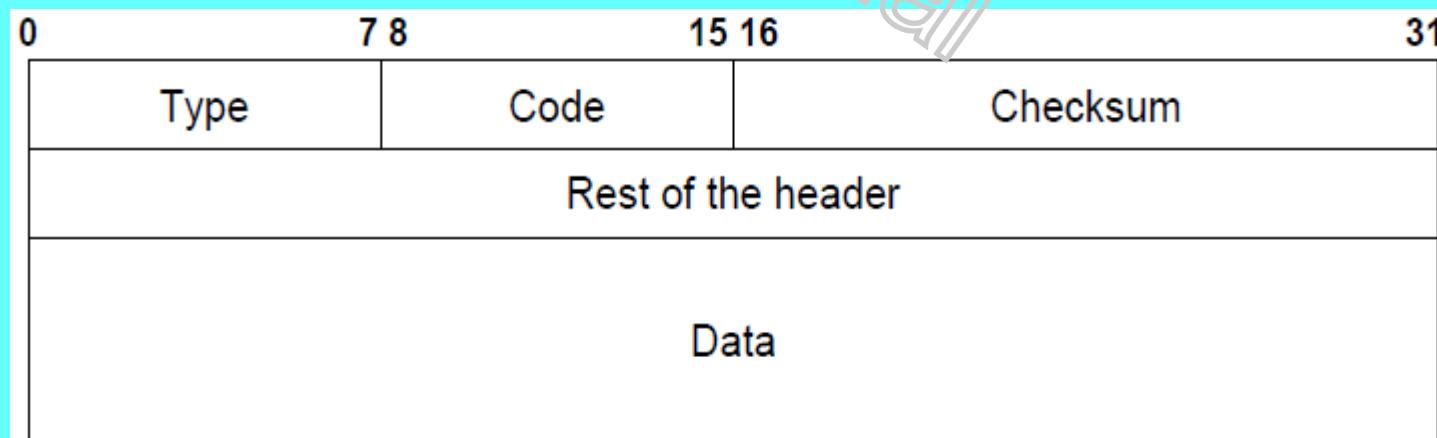
Ver	HLEN	DS	Total Length
Identification		Flags	Fragmentation offset
TTL	Protocol	Header Checksum	
Source IP address			
Destination IP address			
Option			

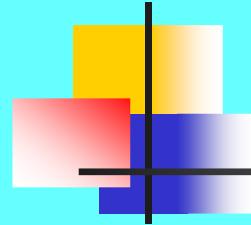
Protocol:
1: ICMP
2: IGMP
6: TCP
17: UDP
89: OSPF



Khuôn dạng gói tin ICMP

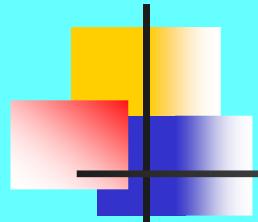
- Type : dạng gói tin ICMP
- Code : Nguyên nhân gây lỗi
- Checksum
- Mỗi dạng có phần còn lại tương ứng





Khuôn dạng gói tin ICMP

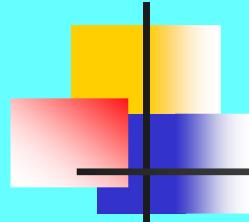
- ICMP luôn hoạt động song trong suốt với người sử dụng
- Có thể sử dụng ICMP thông qua các công cụ debug
 - ping : sử dụng để kiểm tra kết nối
 - traceroute : Công cụ dò vết đường đi



Tầng giao vận

- Tổng quan tầng giao vận
- Giao thức UDP
- Giao thức TCP

Confidential

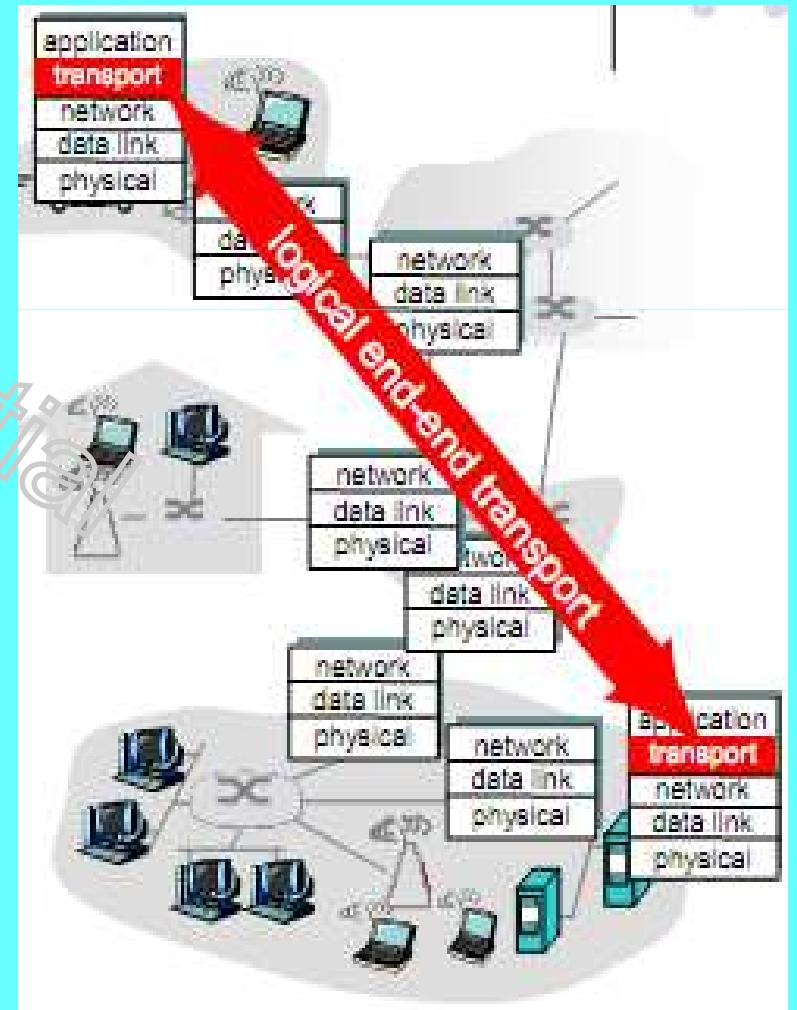


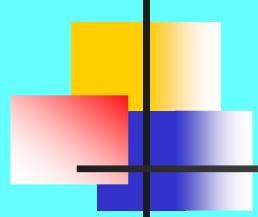
Tổng quan tầng giao vận

- Cung cấp phương tiện truyền giữa các ứng dụng cuối
- Bên gửi :
 - Nhận dữ liệu từ ứng dụng
 - Đặt dữ liệu vào các đoạn tin và chuyển cho tầng mạng
 - Nếu dữ liệu quá lớn thì sẽ được chia thành nhiều phần và sẽ đặt vào các đoạn tin khác nhau.
- Bên nhận :
 - Nhận các đoạn tin từ tầng mạng
 - Tập hợp dữ liệu và chuyển lên cho ứng dụng

Tổng quan tầng giao vận

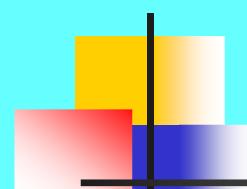
- Được cài đặt trên các hệ thống cuối
 - Không cài đặt trên các routers, switches,..
- Hai dạng dịch vụ giao vận
 - Không tin cậy, không kết nối : UDP
 - Tin cậy và hướng kết nối : TCP





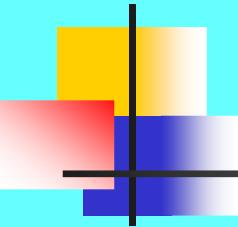
Sử dụng hai loại dịch vụ

- Các yêu cầu đến từ tầng ứng dụng thì đa dạng
- Các ứng dụng cần dịch vụ với 100% độ tin cậy : Web, Email, ...
 - Sử dụng dịch vụ của TCP
- Các ứng dụng cần chuyển dữ liệu nhanh, có khả năng chịu lỗi : VoIP, Video Streaming
 - Sử dụng dịch vụ của UDP



Ứng dụng và dịch vụ giao vận

Ứng dụng	Giao thức ứng dụng	Giao thức giao vận
e-mail	SMTP	TCP
remote terminal access	Telnet	TCP
Web	HTTP	TCP
file transfer	FTP	TCP
streaming multimedia	giao thức riêng (e.g. RealNetworks)	TCP or UDP
Internet telephony	giao thức riêng (e.g., Vonage, Dialpad)	thường là UDP

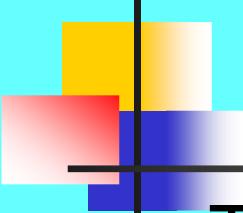


UDP

(User Datagram Protocol)

- Tổng quan UDP
- Khuôn dạng gói tin

Confidential



Tổng quan UDP

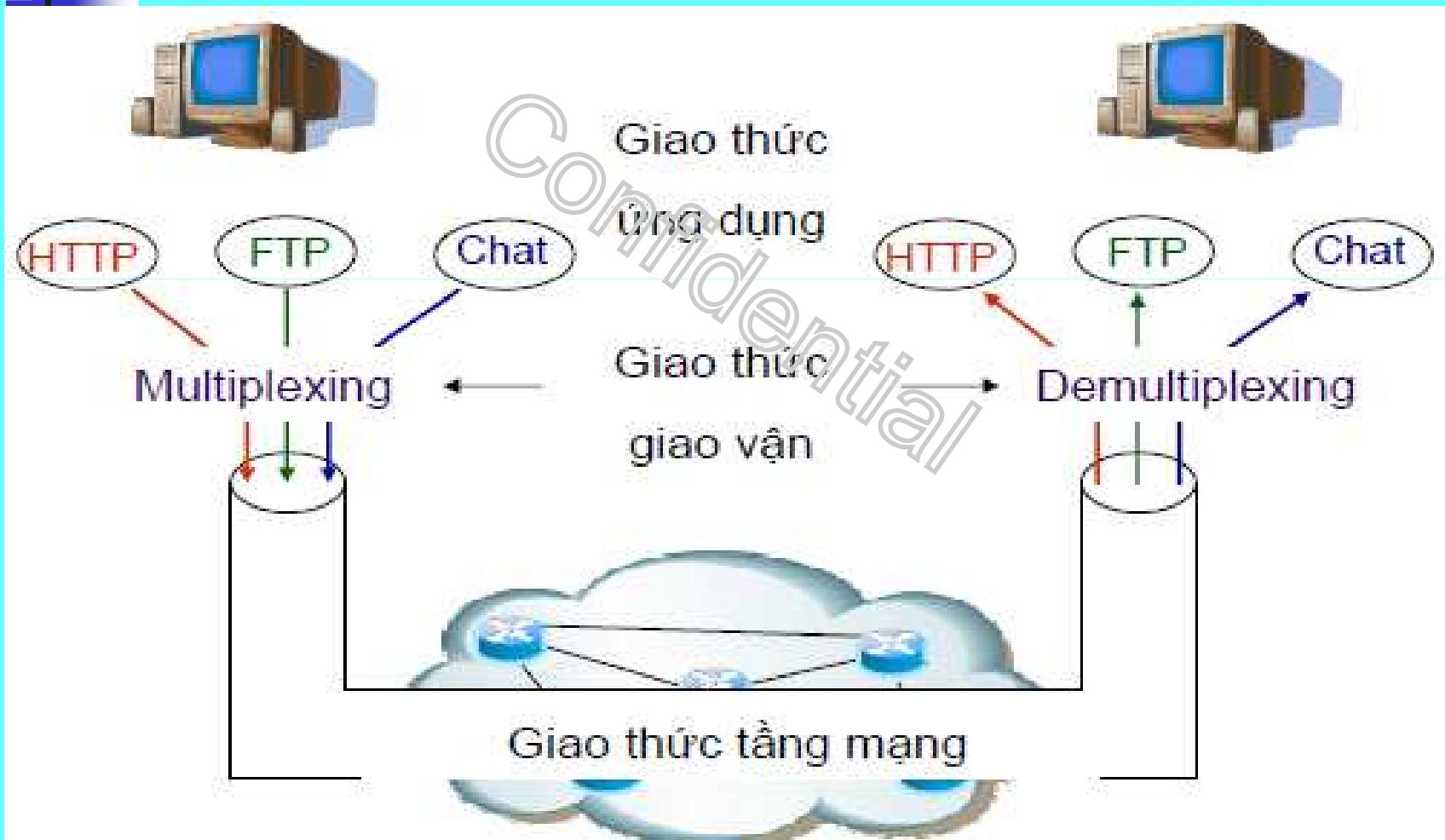
- Tại sao sử dụng UDP

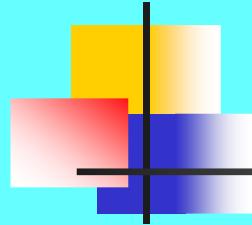
- Không cần phải thiết lập liên kết
- Phần đầu đoạn tin nhỏ
- Không cần lưu lại trạng thái liên kết ở bên gửi và bên nhận
- Không có quản lý tắc nghẽn: UDP gửi dữ liệu nhanh nhất, nhiều nhất nếu có thể

- Chức năng UDP

- Hợp kênh/phân kênh
- Phát hiện lỗi bit bằng checksum

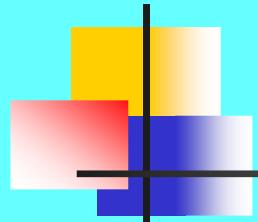
Hợp kênh/phân kênh





Hợp kênh/phân kênh

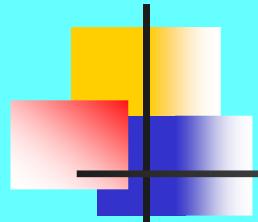
- Tại tầng mạng, gói tin được định danh bởi địa chỉ IP để xác định máy trạm
- Để phân biệt các ứng dụng trên cùng một máy
 - Sử dụng số hiệu cổng
 - Mỗi ứng dụng được gán một cổng
- UDP cũng cung cấp cơ chế gán và quản lý các số hiệu cổng để định danh duy nhất cho các ứng dụng chạy trên một trạm của mạng.
- Socket : Một cặp địa chỉ IP và số hiệu cổng



Checksum

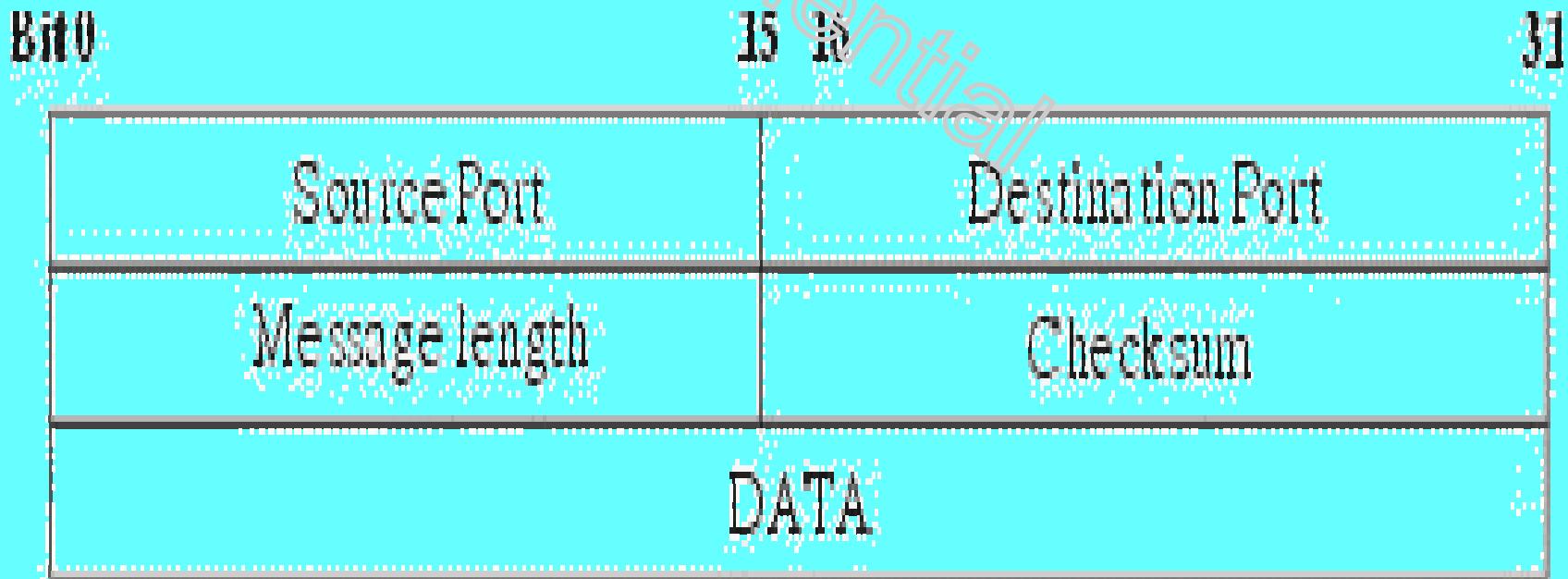
- Phát hiện lỗi bit trong các gói tin/đoạn tin
- Giống checksum của giao thức IP

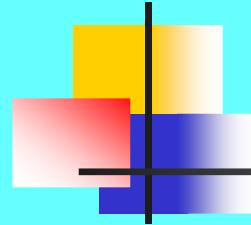
Confidential



Khuôn dạng gói tin UDP

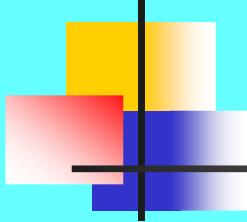
- Do ít chức năng phức tạp nên UDP thường có xu thế hoạt động nhanh hơn so với TCP
- Thường được dùng cho các ứng không đòi hỏi độ tin cậy cao trong giao vận





Các vấn đề khác của UDP

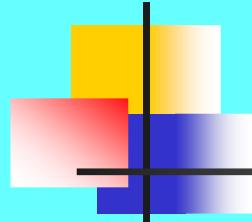
- Không có kiểm soát tắc nghẽn
 - Làm Internet bị quá tải
- Không đảm bảo độ tin cậy
 - Các ứng dụng phải ~~cài đặt~~ cơ chế tự kiểm soát lỗi (ARQ)
- Phát triển ứng dụng sẽ phức tạp hơn



TCP

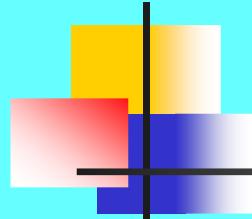
Transmission Control Protocol

- Tổng quan về TCP
- Khuôn dạng gói tin TCP
- Quản lý liên kết
- Kiểm soát luồng
- Kiểm soát tắc nghẽn



Tổng quan về TCP

- TCP là một giao thức có liên kết
- Cần thiết lập liên kết (logic), giữa một cặp thực thể TCP trước khi chúng trao đổi dữ liệu với nhau.
- TCP cung cấp khả năng truyền dữ liệu một cách an toàn giữa các máy trạm trong hệ thống các mạng.
- Cung cấp thêm các chức năng nhằm kiểm tra tính chính xác của dữ liệu khi đến và bao gồm cả việc gửi lại dữ liệu khi có lỗi xảy ra



Tổng quan về TCP

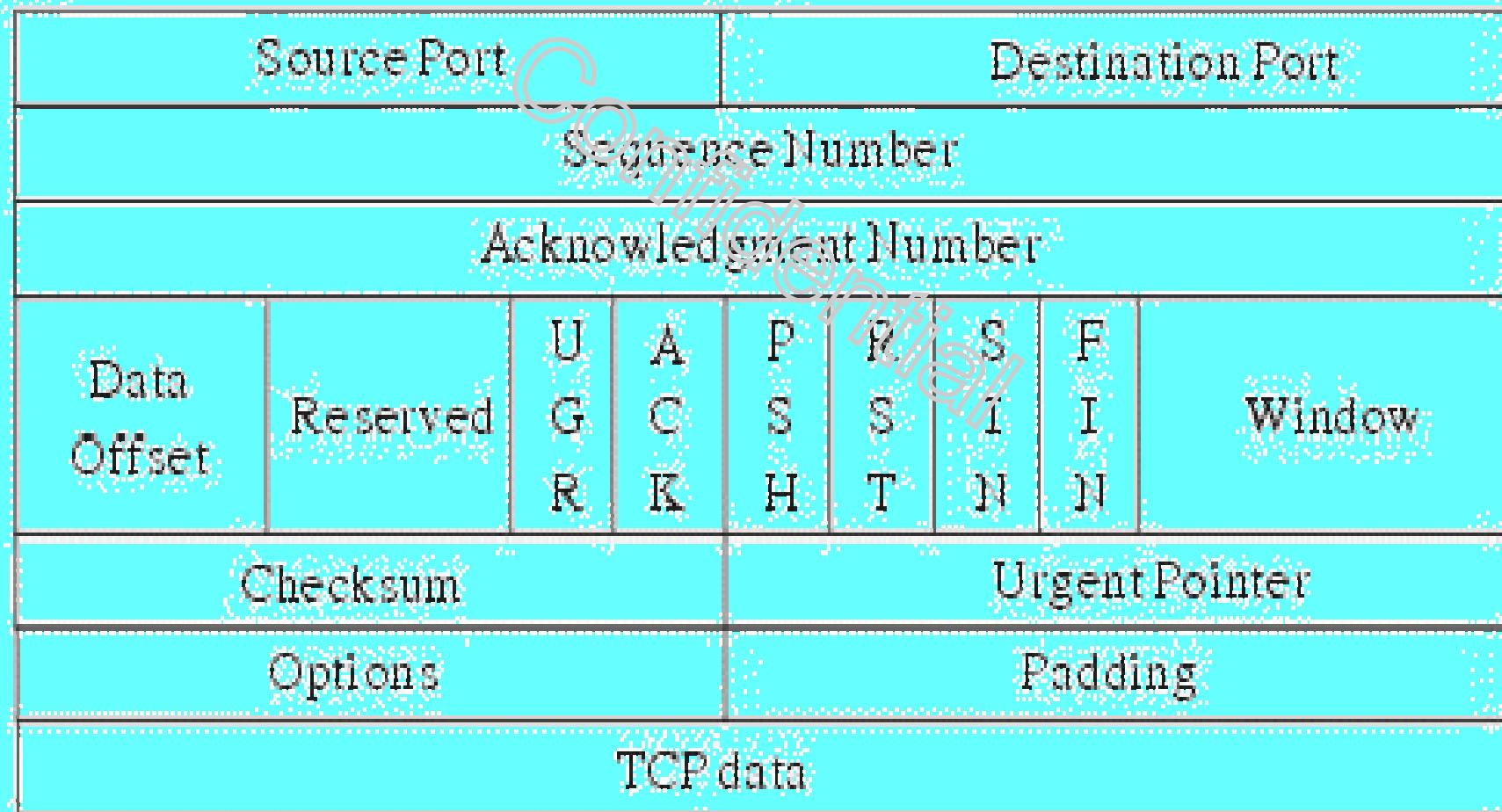
- TCP cung cấp các chức năng chính sau:
 - Thiết lập, duy trì, kết thúc liên kết giữa hai quá trình.
 - Phân phát gói tin một cách tin cậy.
 - Đánh số thứ tự (sequencing) các gói dữ liệu nhằm truyền dữ liệu một cách tin cậy.
 - Cho phép điều khiển lõi.
 - Cung cấp khả năng đa kết nối với các quá trình khác nhau giữa trạm nguồn và trạm đích nhất định thông qua việc sử dụng các cổng.
 - Truyền dữ liệu sử dụng cơ chế song công (full-duplex).

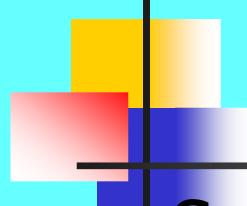
Khuôn dạng gói tin TCP

Bit 0

15 16

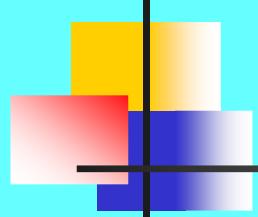
31





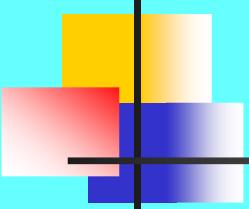
Khuôn dạng gói tin TCP

- Source Port (16 bits): Số hiệu cổng TCP của trạm nguồn.
- Destination Port (16 bit): Số hiệu cổng TCP của trạm đích.
- Sequence Number (32 bit): số hiệu của byte đầu tiên của segment trừ khi bit SYN được thiết lập. Nếu bit SYN được thiết lập thì Sequence Number là số hiệu tuần tự khởi đầu (ISN) và byte dữ liệu đầu tiên là ISN+1.
- Acknowledgment Number (32 bit): số hiệu của segment tiếp theo mà trạm nguồn đang chờ để nhận. Ngầm ý báo nhận tốt (các) segment mà trạm đích đã gửi cho trạm nguồn.



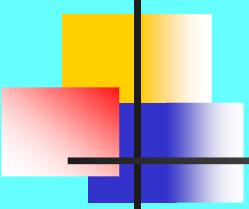
Khuôn dạng gói tin TCP

- Data offset (4 bit): số lượng bội của 32 bit (32 bit words) trong TCP header (tham số này chỉ ra vị trí bắt đầu của nguồn dữ liệu).
- Reserved (6 bit): dành để dùng trong tương lai
- Control bit (các bit điều khiển):
- URG: Vùng con trỏ khẩn (Urgent Pointer) có hiệu lực.
- ACK: Vùng báo nhận (ACK number) có hiệu lực.
- PSH: Chức năng PUSH.



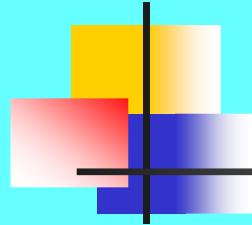
Khuôn dạng gói tin TCP

- RST: Khởi động lại (reset) liên kết.
- SYN: Đồng bộ hóa số hiệu tuần tự (sequence number).
- FIN: Không còn dữ liệu từ trạm nguồn.
- Window (16 bit): cấp phát credit để kiểm soát nguồn dữ liệu (cơ chế của sổ). Đây chính là số lượng các byte dữ liệu, bắt đầu từ byte được chỉ ra trong vùng ACK number, mà trạm nguồn đã sẵn sàng để nhận.
- Checksum (16 bit): mã kiểm soát lỗi cho toàn bộ segment (header + data)



Khuôn dạng gói tin TCP

- Urgent Poiter (16 bit): con trỏ này trỏ tới số hiệu tuần tự của byte đi theo sau dữ liệu khẩn. Vùng này chỉ có hiệu lực khi bit URG được thiết lập.
- Options (độ dài thay đổi): khai báo các option của TCP, trong đó có độ dài tối đa của vùng TCP data trong một segment.
- Padding (độ dài thay đổi): phần chèn thêm vào header để đảm bảo phần header luôn kết thúc ở một mốc 32 bit. Phần thêm này gồm toàn số 0.
- TCP data (độ dài thay đổi): chứa dữ liệu của tầng trên, có độ dài tối đa ngầm định là 536 byte. Giá trị này có thể điều chỉnh bằng cách khai báo trong vùng options.

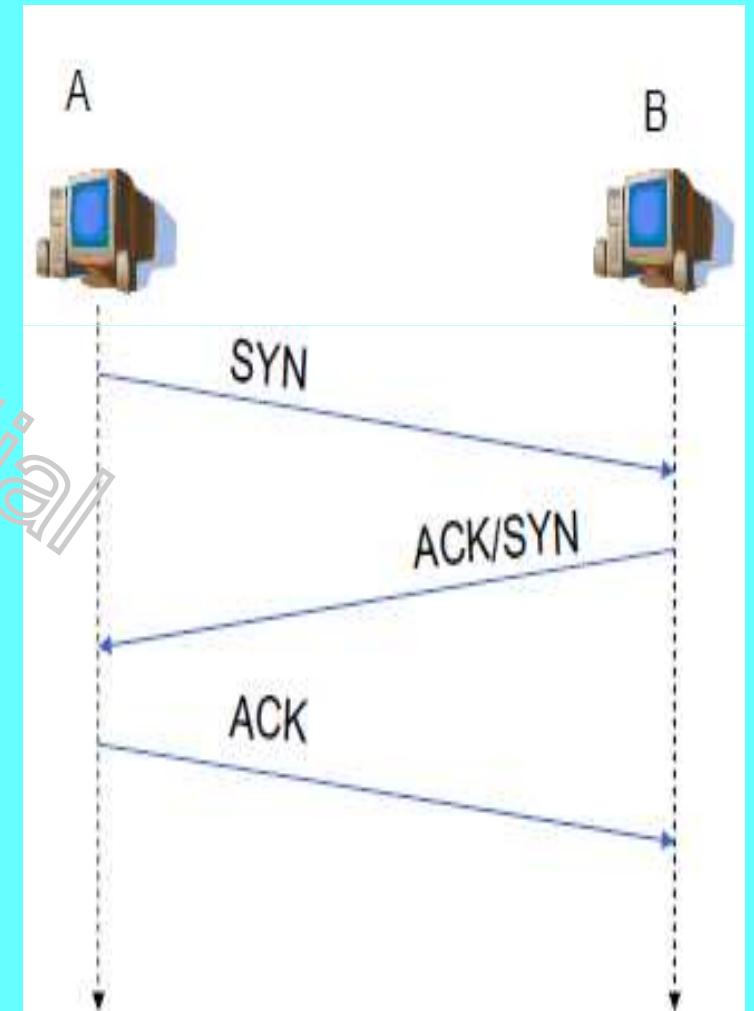


Quản lý liên kết

- Chu trình làm việc của TCP:
 - Thiết lập liên kết
 - Bắt tay ba bước
 - Truyền/nhận dữ liệu
 - Đóng liên kết

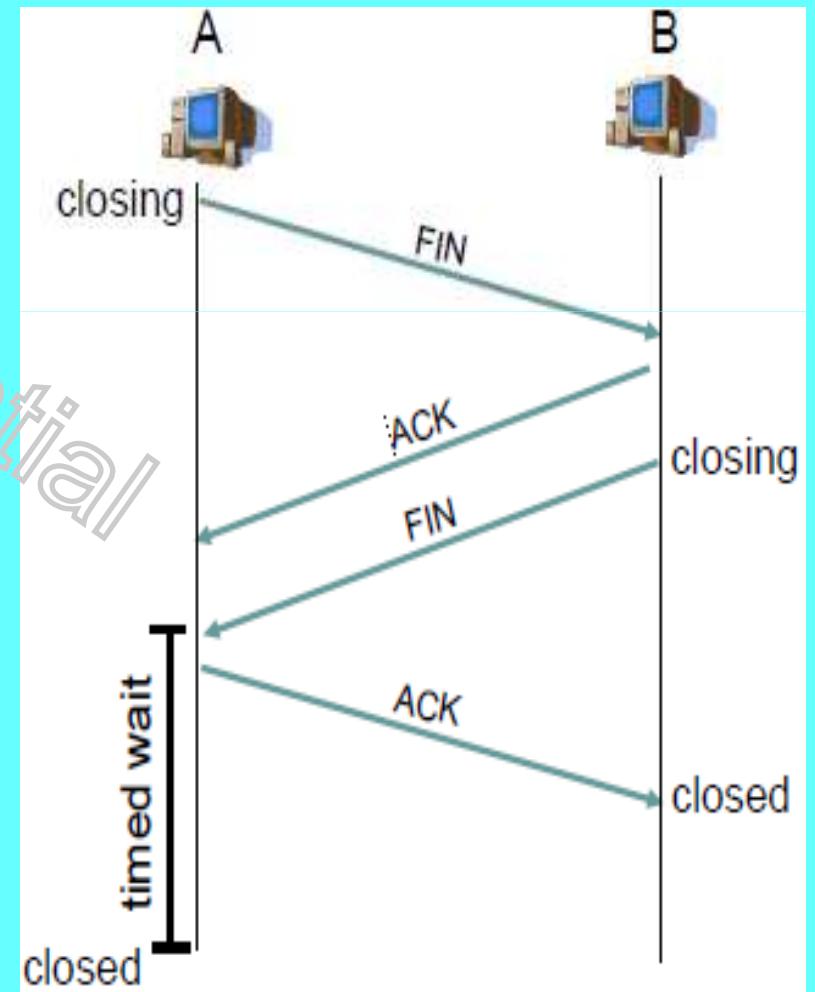
Thiết lập liên kết

- Bước 1: A gửi SYN cho B
 - Chỉ ra giá trị khởi tạo seq # của A
 - không có dữ liệu
- Bước 2: B nhận SYN, trả lời bằng SYNACK
 - B khởi tạo vùng đệm
 - Chỉ ra giá trị khởi tạo seq. # của B
- Bước 3: A nhận SYNACK, trả lời ACK, có thể kèm theo dữ liệu

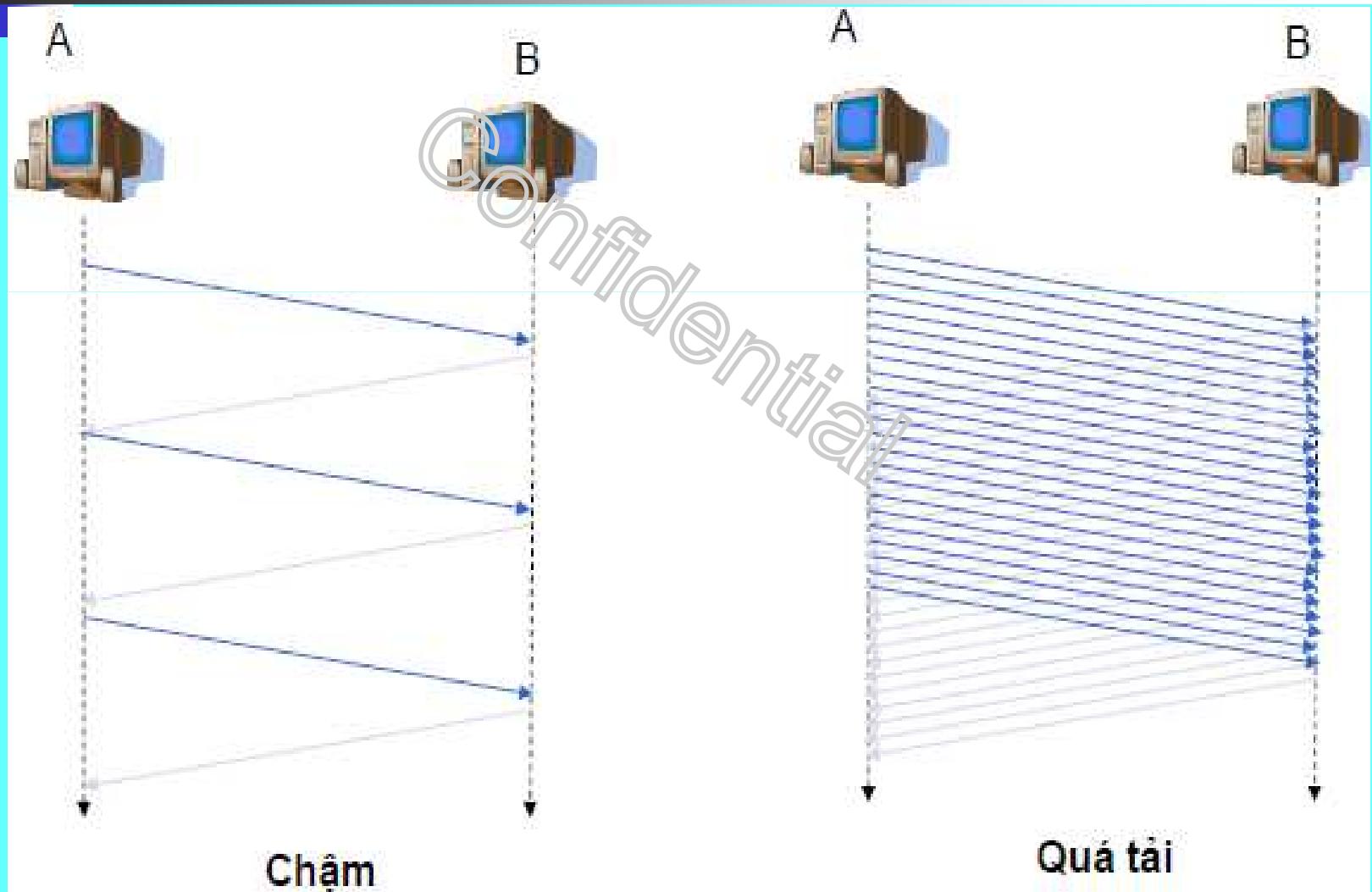


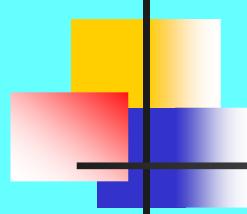
Đóng liên kết

- Bước 1: Gửi FIN cho B
- Bước 2: B nhận được FIN, trả lời ACK, đóng thời đóng liên kết và gửi FIN
- Bước 3: A nhận FIN, trả lời ACK, vào trạng thái chờ
- Bước 4 : B nhận ACK, đóng liên kết



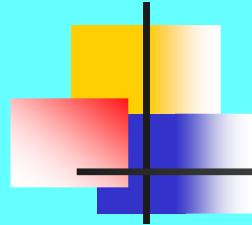
Kiểm soát luồng





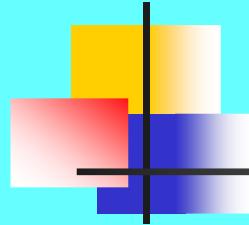
Kiểm soát luồng

- Điều khiển lượng dữ liệu được gửi đi
 - Bảo đảm hiệu quả là tốt
 - Không làm quá tải các bên
- Các bên sẽ có cửa sổ kiểm soát
 - Rwnd : cửa sổ nhận
 - Cwnd : cửa sổ kiểm soát tắc nghẽn
- Lượng dữ liệu gửi đi phải nhỏ hơn $\min(rwnd, cwnd)$



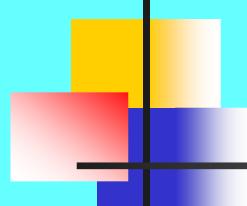
Kiểm soát tắc nghẽn

- Khi nào tắc nghẽn xảy ra :
 - Quá nhiều ~~cặp~~ gửi nhận trên mạng
 - Truyền quá nhiều làm mạng quá tải
- Hậu quả của nghẽn mạng:
 - Mất gói tin
 - Thông lượng giảm, độ trễ tăng
 - Mạng sẽ trở nên tồi tệ hơn



Kiểm soát tắc nghẽn

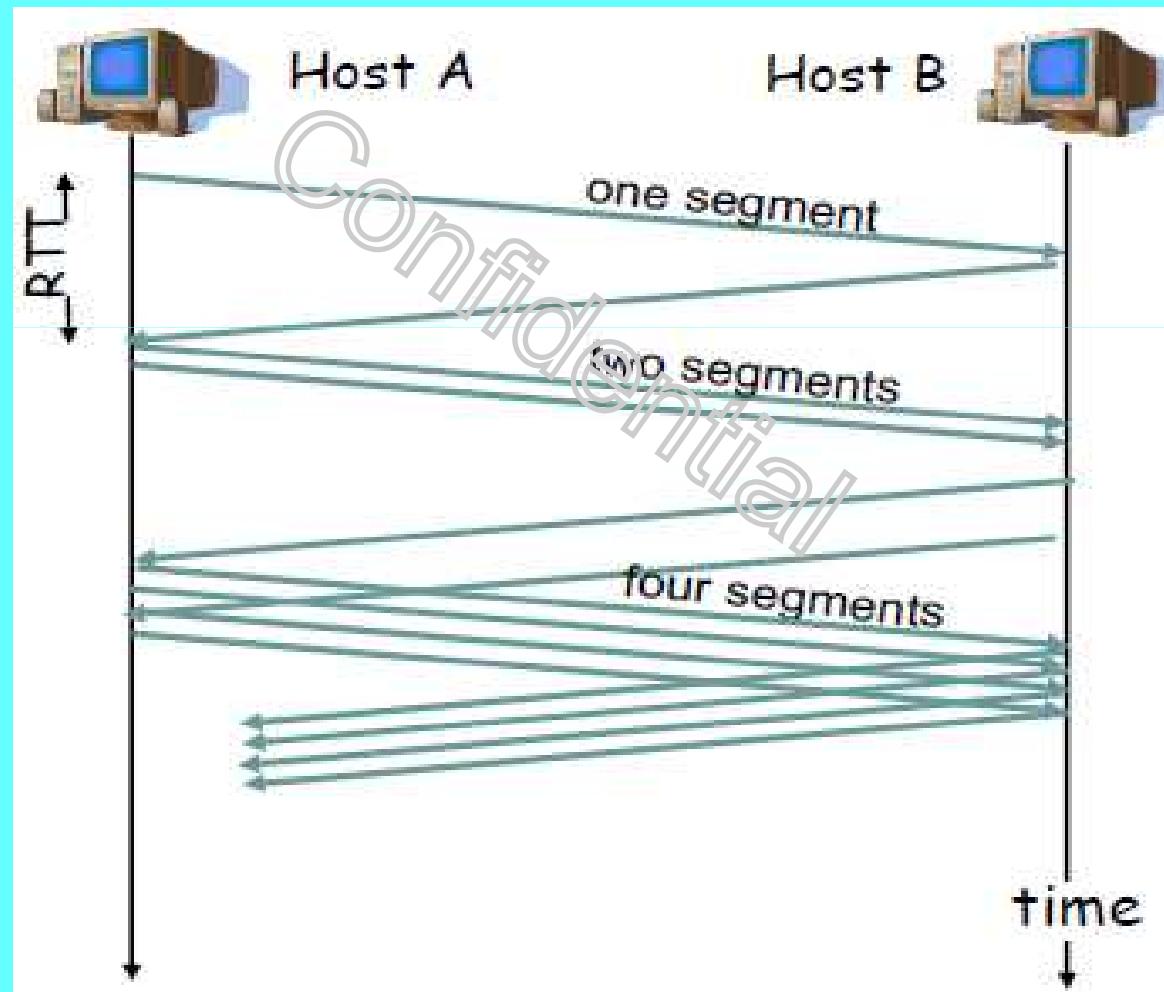
- Nguyên lý kiểm soát tắc nghẽn :
 - Slow-start
 - Tăng tốc độ theo hàm số mũ
 - Tiếp tục tăng đến một ngưỡng nào đó
 - Tránh tắc nghẽn
 - Tăng dần tốc độ theo hàm tuyến tính cho đến khi phát hiện tắc nghẽn



Slow Start

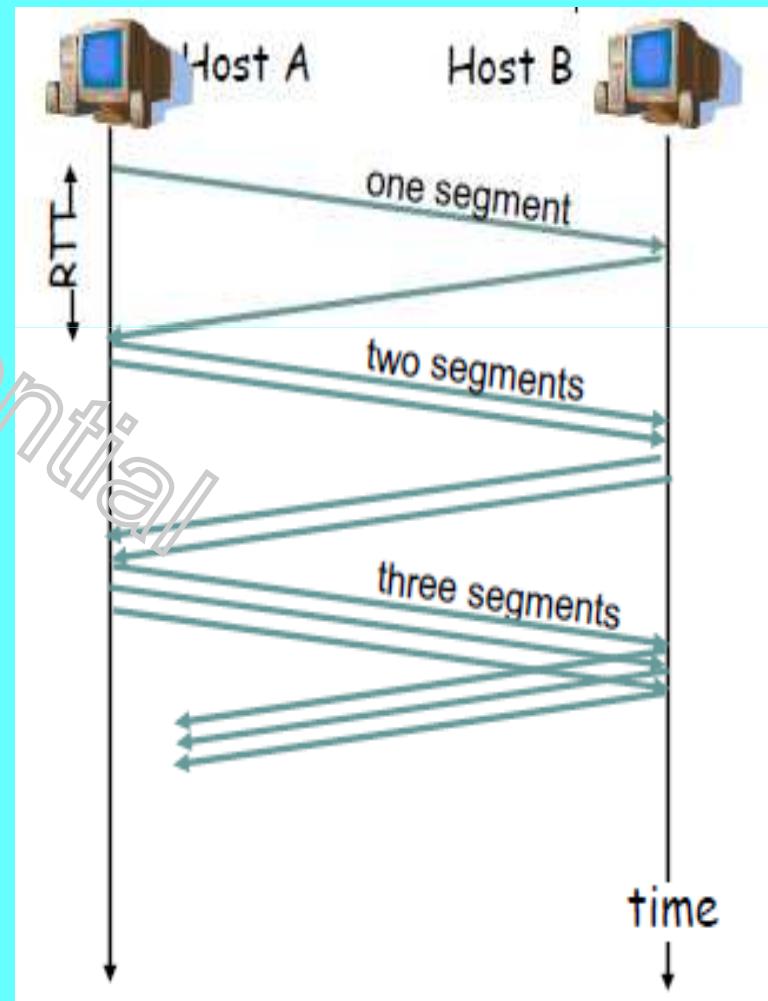
- Đặt cwnd bằng 1 MMS(Maximum segment size)
- Tăng cwnd lên gấp đôi khi nhận được ACK
- Bắt đầu chậm nhưng tăng theo hàm mũ
- Tăng cho đến một ngưỡng : ssthresh
 - Sau đó TCP chuyển sang trạng thái tránh tắc nghẽn

Slow Start



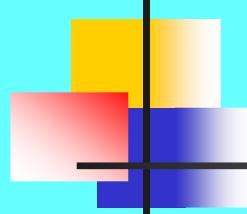
Tránh tắc nghẽn

- Tăng cwnd theo cấp số cộng sau khi nó đạt đến ssthresh
- Khi bên gửi nhận được ACK
 - Tăng cwnd thêm 1 MMS



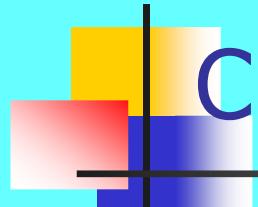
Nhắc lại kiến trúc phân tầng

Application (HTTP, Mail, ...)	Hỗ trợ các ứng dụng trên mạng
Transport (UDP, TCP ...)	Truyền dữ liệu giữa các ứng dụng
Network (IP, ICMP...)	Chọn đường và chuyển tiếp gói tin giữa các máy, các mạng
Datalink (Ethernet, ADSL...)	Hỗ trợ việc truyền thông cho các thành phần kế tiếp trên cùng 1 mạng
Physical (bits...)	Truyền và nhận dòng bit trên đường truyền vật lý



Mối quan hệ giữa mô hình OSI và TCP/IP

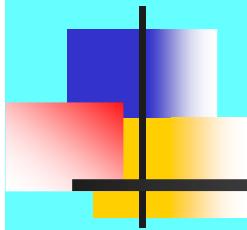
OSI	TCP/IP	
APPLICATION PRESENTATION	APPLICATION	Telnet, FTP, DNS
SESSION TRANSPORT	TRANSPORT	TCP, UDP
NETWORK	INTERNET	IP
DATA PHYSICAL	NETWORK INTERFACE	WANs, TokenRing



Chương 4 : Ngôn ngữ lập trình Java

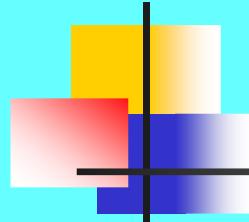
- Sơ lược lập trình hướng đối tượng
- 4.1 : Giới thiệu về Java
- 4.2 : Các thành phần cơ bản -
Cấu trúc chương trình Java
- 4.3 : Interface & Package
- 4.4 : Xử lý biệt lệ
- 4.5 : Lập trình giao diện
- 4.6 : Applets
- 4.7 : Lập trình đa tuyến
- 4.8 : Các luồng vào ra

CHƯƠNG 4 : NGÔN NGỮ LẬP TRÌNH JAVA



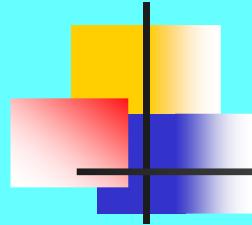
Configurable

**GIỚI THIỆU SƠ LƯỢC LẬP TRÌNH
HƯỚNG ĐỐI TƯỢNG**



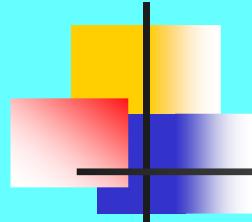
Mục Tiêu Bài Học

- Thể nào là lập trình hướng đối tượng
- Tìm hiểu về trừu tượng dữ liệu
- Định nghĩa lớp và đối tượng
- Constructor và Destructor
- Tìm hiểu về tính lưu trữ, bao bọc dữ liệu, tính kế thừa và đa hình
- Các ưu điểm của phương pháp lập trình hướng đối tượng



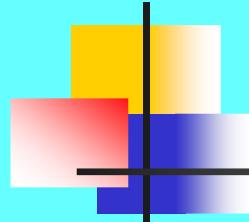
Lập Trình Hướng Đối Tượng

- Lấy đối tượng làm nền tảng cơ sở của phương pháp lập trình
- Phương pháp thiết kế và thực hiện bằng các hệ phần mềm



Trùu Tượng Dữ Liệu

- Là tiến trình xác định và tập hợp các **tính chất** và các **hành động** của một thực thể có liên quan đến ứng dụng
- Lợi ích :
 - Tập trung vào vấn đề
 - Xác định những tính chất và hành động thiết yếu
 - Loại trừ những chi tiết không cần thiết



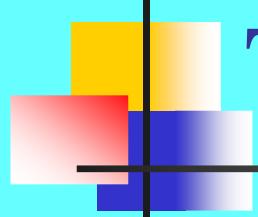
Trùu Tượng Dữ Liệu

**Các tính chất
của một đối
tượng Người
Người**

Tên
Địa chỉ
Tuổi
Chiều cao
Màu tóc

**Các tính chất của
một đối tượng
Khách hàng**

Tên
Địa chỉ



Trùu Tượng Dữ Liệu (tiếp theo)

Các thuộc tính	Các hành động
Tên của khách hàng	Nhập tên của khách hàng
Địa chỉ của khách hàng	Nhập địa chỉ của khách hàng
Đời xe hơi đã mua	Nhập đời xe hơi mua được
Người bán xe hơi	Nhập tên người bán xe hơi
	Lập hóa đơn

Lớp

Lớp là một nhóm các đối tượng có chung những tính chất và hành động

Lớp Khách hàng

Tên khách hàng

Địa chỉ khách hàng

Đời xe hơi đã mua

Tên người bán xe hơi

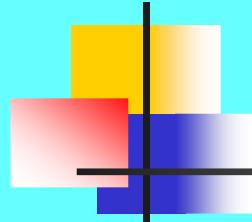
Nhập tên

Nhập địa chỉ

Nhập đời của xe hơi mua được

Nhập tên của người bán xe hơi

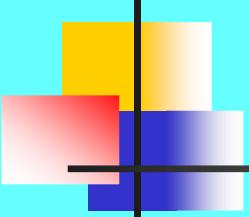
Lập hóa đơn



Đối Tượng

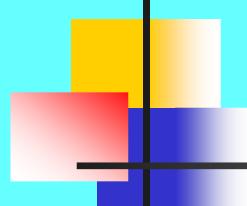
- Đối tượng là một thể hiện của lớp
 - Toàn
 - Anh
 - Tuấn

Confidential



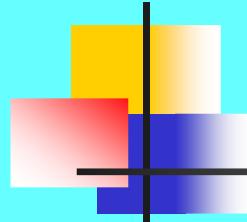
Đối Tượng (tiếp theo)

- Thuộc tính
 - Tính chất mô tả một đối tượng
- Hành động
 - Dịch vụ mà đối tượng có thể đáp ứng
- Phương thức
 - Đặc tả cách đáp ứng bằng hành động khi được yêu cầu
- Thông điệp
 - Yêu cầu một hành động
- Biến cố
 - Sự kích thích từ đối tượng này gửi sang đối tượng khác



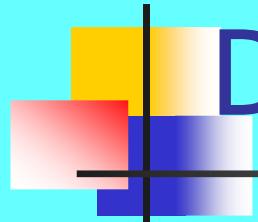
Lớp Và Đối Tượng

- Lớp là một thực thể, còn đối tượng là một thực thể thực tế
- Lớp là một mô hình ý niệm định rõ các tính chất và các hành động được quy định bởi một đối tượng, còn đối tượng là một mô hình thực sự
- Lớp là khuôn mẫu từ đó đối tượng được tạo ra
- Tất cả các đối tượng trong cùng một lớp có các tính chất và các hành động như nhau



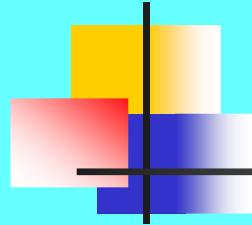
Constructor

- Tiến trình tạo ra một đối tượng được gọi là Constructor
- Một Constructor:
 - Cấp phát vùng nhớ
 - Khởi gán những thuộc tính (nếu có)
 - Cho phép truy cập những thuộc tính và phương thức



Destructor

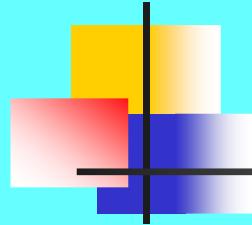
- Tiến trình hủy một đối tượng gọi là Destructor
- Một Destructor
 - Giải phóng bộ nhớ
 - Cấm truy cập thuộc tính và phương thức



Tính Lưu Trữ

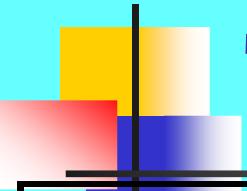
- Tính lưu trữ là khả năng của đối tượng có thể lưu lại dữ liệu của nó sau khi đã bị hủy

Confidential



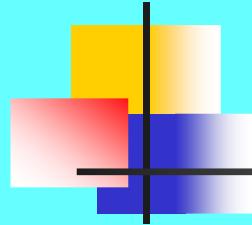
Tính Bao Bọc Dữ Liệu

- Tiến trình **che dấu** những chi tiết hiện **thực** một ~~đối~~ tượng được gọi là **tính bao bọc**
- **Ưu điểm:**
 - Tất cả những thuộc tính và phương thức cần thiết đều được tạo
 - Một lớp có thể có nhiều tính chất và phương thức nhưng chỉ một số trong đó được hiển thị cho người dùng



Tính Kế Thừa

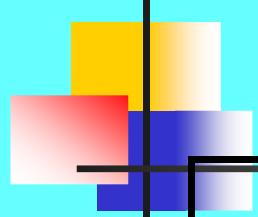
LỚP SINH VIÊN	LỚP NHÂN VIÊN	LỚP KHÁCH HÀNG
Tên	Tên	Tên
Địa chỉ	Địa chỉ	Địa chỉ
Điểm môn 1	Lương	Sản phẩm mua được
Điểm môn 2	Chức vụ	Nhập tên
Nhập tên	Nhập tên	Nhập địa chỉ
Nhập địa chỉ	Nhập địa chỉ	Nhập mã sản phẩm
Nhập điểm	Nhập lương	Lập hóa đơn
Tính tổng số điểm	Tính lương	



Tính Ké Thừa (tiếp theo)

Lớp Người
Tên
Địa chỉ
Nhập tên
Nhập địa chỉ

Confidential



Tính Ké Thừa (tiếp theo)

LỚP NGƯỜI

Tên

Địa chỉ

Nhập tên

Nhập địa chỉ

+

=

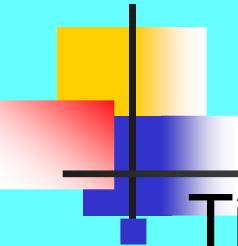
Thêm các thuộc tính và hành động cần thiết vào lớp khách hàng

Nhập mã sản phẩm đã mua

Lập hóa đơn

Lớp Khách Hàng

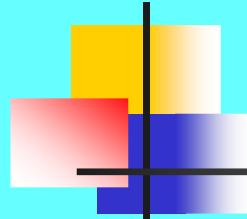
Confidential



Tính Ké Thừa (tiếp theo)

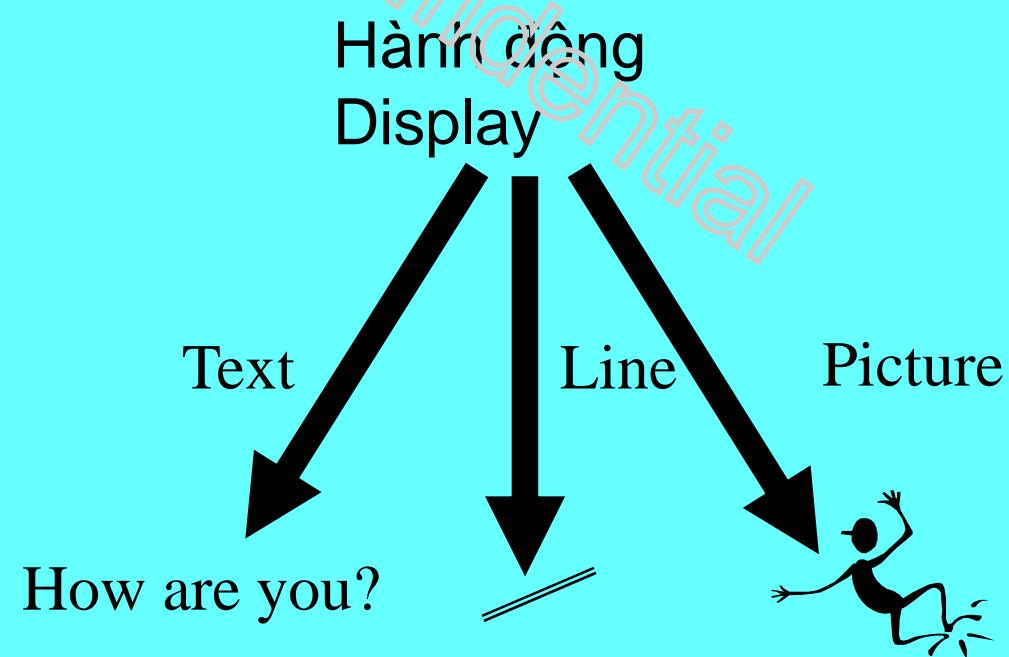
Tính Thừa kế

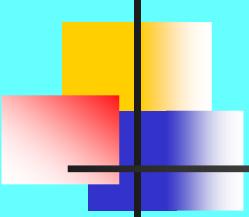
- Là cơ chế cho phép một lớp chia sẻ những thuộc tính và những hành động đã định nghĩa trong một hoặc nhiều lớp khác
- Lớp con
 - Là lớp thừa kế từ lớp khác
- Lớp cha
 - Là lớp từ đó một lớp khác thừa kế các ứng xử của nó
- Đa thừa kế
 - Khi một lớp con thừa kế từ hai hoặc nhiều lớp



Tính Đa Hình

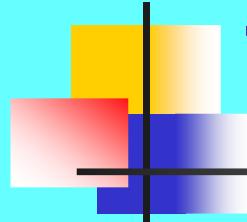
- Tính đa hình là thuộc tính cho phép một hành động ứng xử khác nhau trên các lớp khác nhau





Ưu điểm của phương pháp hướng đối tượng

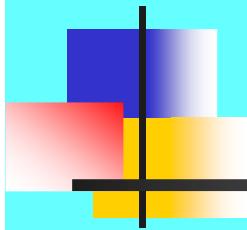
- Chia sẻ trong phạm vi một ứng dụng
- Đẩy mạnh sự dùng lại của các đối tượng khi hiện thực những ứng dụng mới
- Về lâu dài, chi phí ~~giảm~~ đáng kể
- Giảm lỗi và rắc rối trong bảo trì
- Điều chỉnh nhanh hơn



TỔNG KẾT

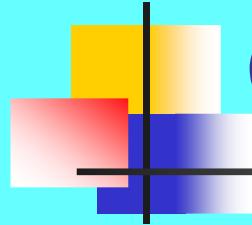
- Tiếp cận hướng đối tượng đưa ra một giải pháp toàn diện cho một bài toán cụ thể
- Trừu tượng dữ liệu là một tiến trình xác định và tập hợp các tính chất và các hành động có quan hệ với một thực thể cụ thể
- Lớp mô tả một thực thể, còn đối tượng là một thực thể thực tế
- Constructor và Destructor
- Tính lưu trữ, bao bọc dữ liệu, tính kế thừa và đa hình

CHƯƠNG 4 : NGÔN NGỮ LẬP TRÌNH JAVA



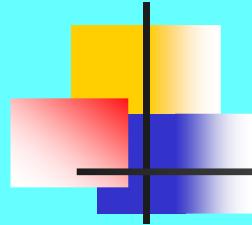
Confidential

Giới thiệu ngôn ngữ lập trình Java



Giới thiệu

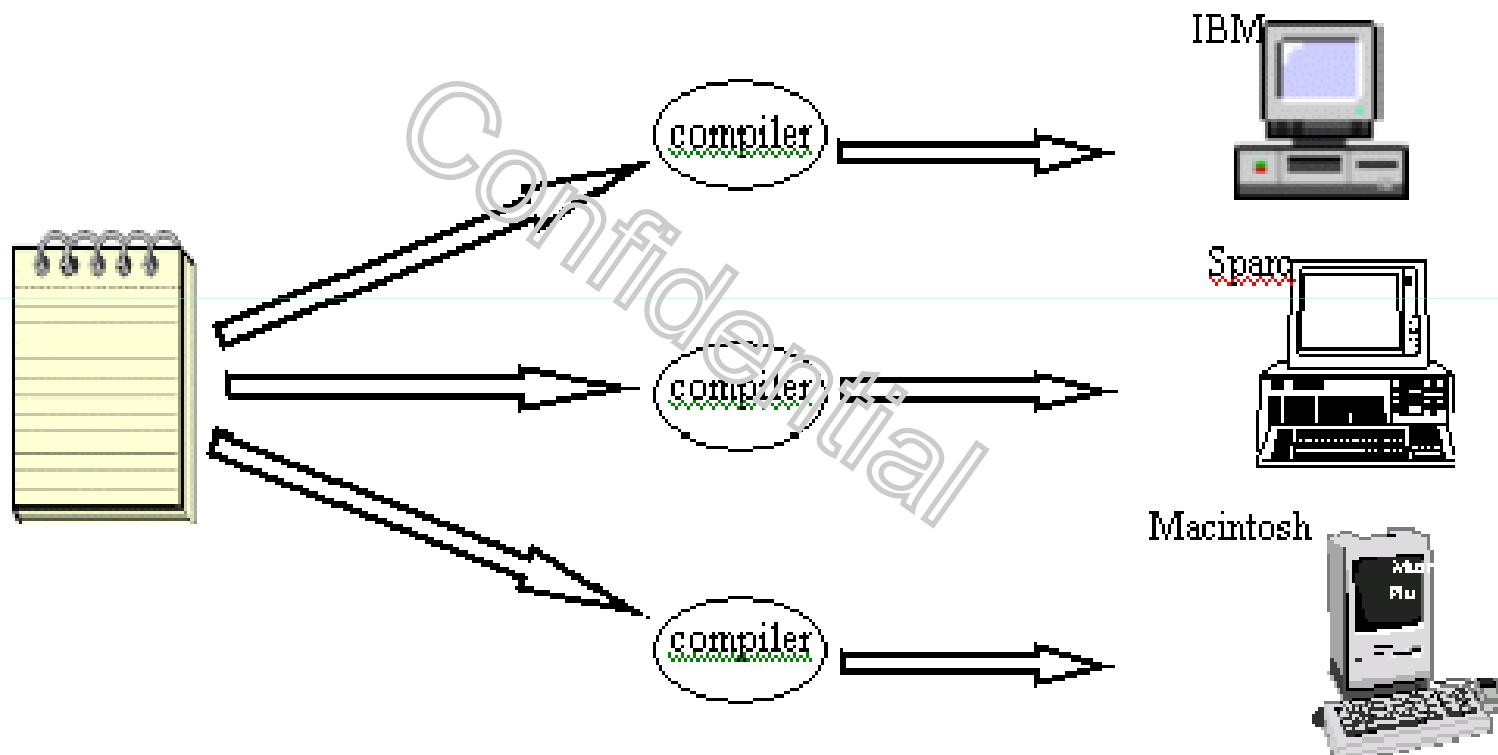
- Sự phát triển của Java
- Hướng tới người dùng
- Giống với C / C++



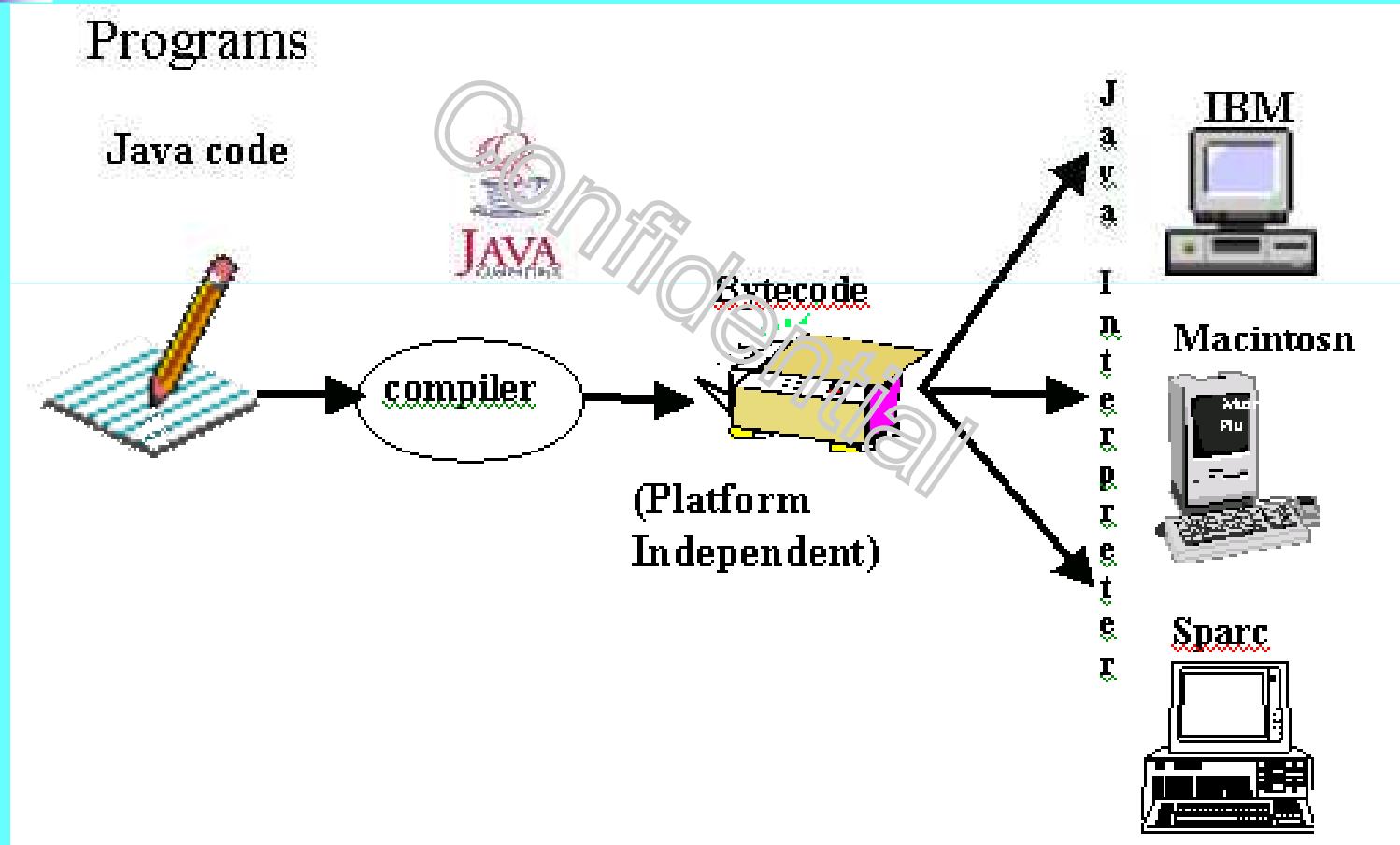
Các đặc trưng của Java

- Đơn giản
- Hướng đối ~~ứng~~ *Confidential*
- Độc lập phần cứng
- Mạnh
- Bảo mật
- Phân tán
- Đa luồng
- Động

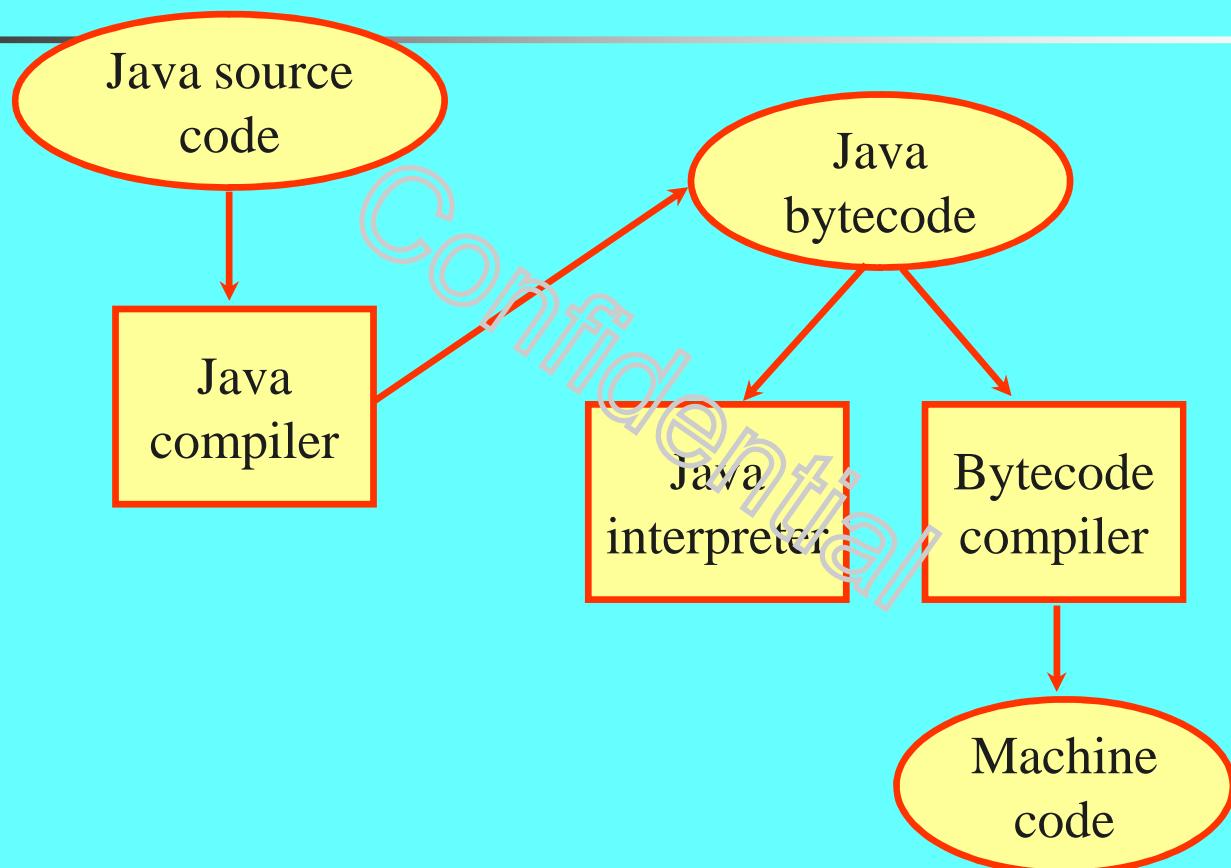
Các chương trình dịch truyền thống

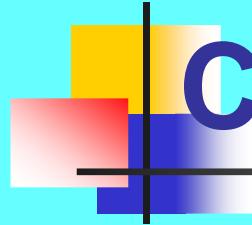


Chương trình dịch Java



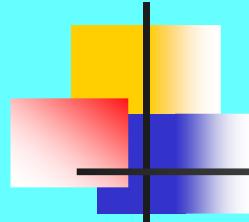
Java Translation and Execution





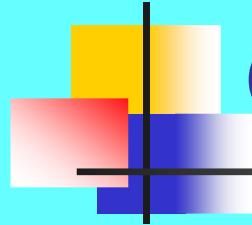
Các loại chương trình Java

- Applets
- Ứng dụng độc lập (console Application)
- Ứng dụng giao diện (GUI Application)
- Servlet
- Ứng dụng cơ sở dữ liệu



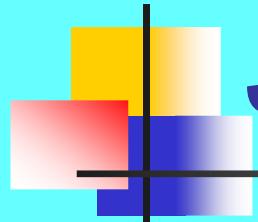
Máy ảo Java

- Là một phần mềm dựa trên cơ sở máy tính ảo
- Là tập hợp các lệnh logic để xác định hoạt động của máy tính
- Được xem như là một hệ điều hành thu nhỏ
- Nó thiết lập lớp trừu tượng cho:
 - Phần cứng bên dưới
 - Hệ điều hành
 - Mã đã biên dịch



Quá trình dịch chương trình Java

- Trình biên dịch chuyển mã nguồn thành tập các lệnh **không** phụ thuộc vào phần cứng
- Trình thông dịch trên mỗi máy chuyển tập lệnh này thành chương trình thực thi
- Máy ảo tạo ra một môi trường để thực thi các lệnh bằng cách:
 - Nạp các file .class
 - Quản lý bộ nhớ
 - Dọn “rác”

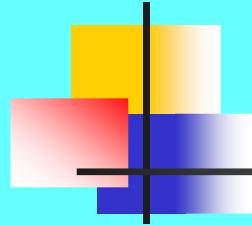


Trình dịch Java

Java Development Kit

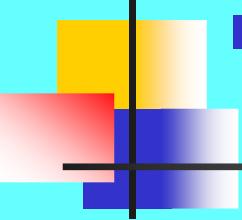
- Java 1.0
- Java 1.1
- Java 2
-

Confidential



Bộ công cụ JDK

- Trình biên dịch, 'javac'
 - **javac [options] sourcecodename.java**
- Trình thông dịch, 'java'
 - **java [options] classname**
- Trình dịch ngược, 'javap'
 - **javap [options] classname**
- Công cụ sinh tài liệu, 'javadoc'
 - **javadoc [options] sourcecodename.java**



- Chương trình tìm lỗi - Debug, 'jdb'

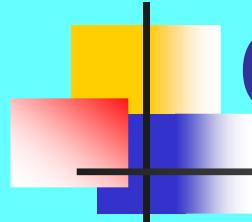
- **jdb [options] sourcecodename.java**

OR

- **jdb -host -password [options]**
sourcecodename.java

- Chương trình xem applet ,
'appletviewer'

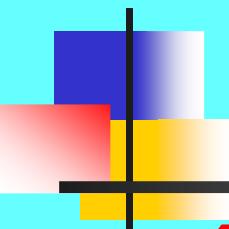
- **appletviewer [options]**
sourcecodename.java / url



Các gói chuẩn của Java

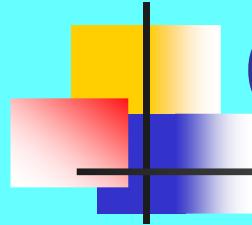
- **java.lang – General support**
- **java.applet - Creating applets for the web**
- **java.awt - Graphics and graphical user interfaces**
- **java.io**
- **java.util – Utilities**
- **java.net - Network communication**
- **java.awt.event**
- **java.rmi**
- **java.security**
- **java.sql**

CHƯƠNG 4 : NGÔN NGỮ LẬP TRÌNH JAVA



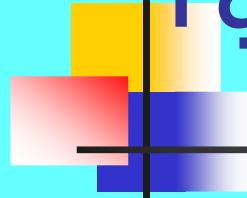
Config
Initial

4.2 Các kiến thức cơ bản – Cấu trúc chương trình của ngôn ngữ Java



Các thành phần cơ bản

- * TẬP KÝ TỰ CỦA JAVA
- * TỪ KHOÁ
- * TÊN, LỜI CHÚ THÍCH
- * CÂU LỆNH VÀ DẤU CHỐM CÂU
- * CẤU TRÚC CỦA CHƯƠNG TRÌNH JAVA
- * MỘT SỐ CHƯƠNG TRÌNH JAVA MẪU



Tập kí tự của java

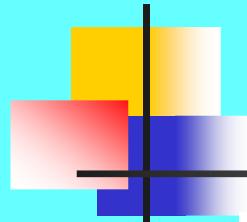
Mỗi ngôn ngữ đều được xây dựng trên **bộ kí tự**.

Bộ kí tự của Java như sau:

- + Các chữ cái in hoa: A, B, C, ... Y, Z
- + Các chữ cái in thường: a, b, c, ... y, z
- + Các chữ số: 0, 1, 2, ..., 9
- + Các dấu câu: , ; / ? . [] { }! ' " ~ @ # \$ % ^ & * () - + = < >
- + Các dấu ngăn cách không nhìn thấy: dấu cách, dấu tab, dấu xuống hàng enter
- + Dấu gạch nối dưới _

Chú ý:

- + Java phân biệt chữ in hoa và chữ in thường.
- + Có thể dùng các kí tự khác như Φ, θ, ∞, σ, ε, ... hay cả tiếng Việt trong chương trình Java



Bộ từ khóa của Java

abstract	boolean	break	byte
case	catch	char	class
const	continue	default	do
double	else	extends	final
finally	float	for	goto
if	implements	import	instanceof
int	interface	long	native
new	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	try
void	volatile	while	

Tên và Lời chú thích

* *Tên :*

- Tên dùng để xác định các đại lượng khác nhau trong một chương trình.
- Mọi tên được sử dụng trong chương trình đều phải được khai báo.
- Tên hợp lệ là dãy kí tự liên nhau, có thể gồm các chữ cái (a..,z, A..,Z), các chữ số (0..9), kí tự gạch dưới (_).
- Tên phải bắt đầu bằng chữ cái hoặc dấu gạch dưới.
- Tên không được trùng với từ khoá.

Ví dụ : **Nghiem_x, hoan_doi, y_1, y2,..** là tên hợp lệ.

Tên và Lời chú thích

* **Lời chú thích :**

- Không có tác dụng tạo ra mã của chương trình, giúp dễ dàng trong việc tìm kiếm, kiểm tra, sửa chương trình
- Lời chú thích có thể được đặt ở bất kỳ đâu trong chương trình
- Cách ghi lời chú thích:

Cách 1 : Chú thích cho nhiều dòng

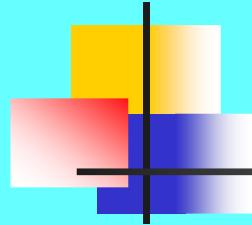
/* lời chú thích

*/

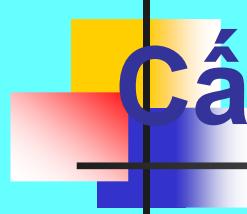
Cách 2 : Chú thích cho một dòng

// lời chú thích

Câu lệnh và Dấu chấm câu



- Một câu lệnh thường gồm phần mô tả dữ liệu và phần lệnh theo quy định của Java.
- Có hai loại:
 - + Câu lệnh đơn giản: là những lệnh không chứa các lệnh khác : lệnh gán, lệnh gọi phương thức, ...
 - + Câu lệnh cấu trúc: gồm khôi lệnh, lệnh rẽ nhánh, lệnh lặp
- Khôi lệnh là gồm nhiều lệnh được đặt trong dấu { }
- Các câu lệnh cách nhau bởi dấu ";"
- Dấu chấm phẩy ";" dùng để ngăn cách các lệnh, nghĩa là cuối mỗi lệnh, mỗi khai báo đều có chấm phẩy



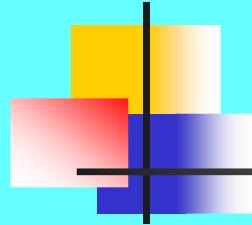
Cấu trúc một chương trình Java

- Xác lập thông tin môi trường
- Khai báo lớp đối tượng (Class)
- Các thành phần (Tokens):
 - Định danh
 - Từ khóa / từ dự phòng
 - Ký tự phân cách
 - Hằng (Literals)
 - Toán tử

Java Program Structure

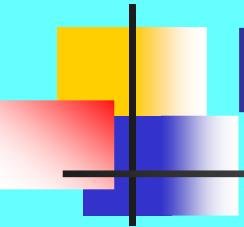
```
// comments about the class  
public class MyProgram  
{  
    // comments about the method  
    public static void main (String[ ] args)  
    {  
        } } }  
    } } } } } } } } } } } } } } } } } } }
```

The code illustrates the structure of a Java program. It starts with class-level comments, followed by the class definition with its name, MyProgram. Inside the class, there is another set of comments preceding the main method. The main method is defined with its signature: `public static void main (String[] args)`. The entire body of the main method is enclosed in curly braces. Labels with arrows point to specific parts: 'method header' points to the `main` method declaration, and 'method body' points to the opening brace of the main method's block.



Chương trình Java mẫu

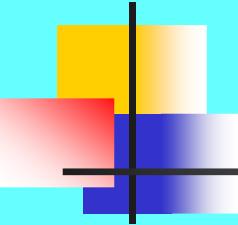
```
// This is a simple program called “Ex1.java”
// import java.lang.*;
class Ex1
{
    public static void main(String args[])
    {
        System.out.println(“My first program in Java”);
    }
}
```



Biên dịch chương trình java

- ..\jdk\bin>**javac Ex1.java**
- ..\jdk\bin>**java Ex1**

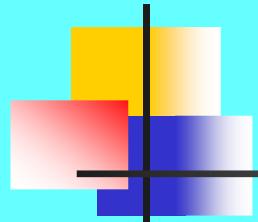
- Kết quả:
My first program in Java



Truyền đối số dòng lệnh

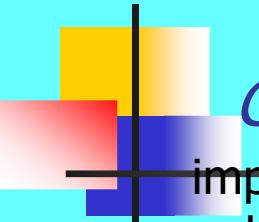
```
class Ex2
{
    public static void main(String s[])
    {
        System.out.println(s[0]);
        System.out.println(s[1]);
        System.out.println(s[2]);
    }
}
```

Truyền đối số trong dòng lệnh (Tiếp theo...)



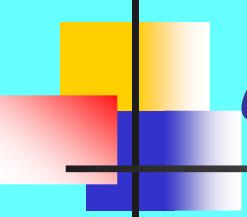
```
D:\WINDOWS\System32\cmd.exe
C:\>java Ex2 c'est la vie
c'est
la
vie
C:\>
```

Confidential



Chương trình tính căn bậc hai của m

```
import java.io.*;
public class canbachai
{
    public static void Tinhcb2(int m)
    {
        double n=0.0;
        n=Math.sqrt(m);
        System.out.println("Căn bậc hai của " +m+ " là " +n);
    }
    public static void main(String args[])
    {
        // tạo đối tượng để gọi
        canbachai t=new canbachai();
        t.Tinhcb2(9);
        t.Tinhcb2(16);
    }
}
```



Chương trình tính n^{10}

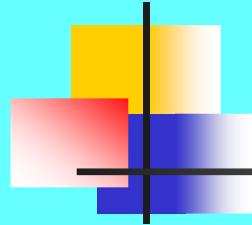
```
import java.io.*;
class Tinh_n_mu10{
    public static long Tinh(int n)
    {
        long tg=1;
        for(int i=1;i<=10;i++) tg*=n;
        return tg;
    }
    public static void main(String [] args)
    {
        // tao doi tuong dt
        Tinh_n_mu10  dt=new  Tinh_n_mu10 ();
        System.out.println("Luy thua 10 cua so la :" +dt.Tinh(2));
    }
}
```

Chương trình kiểm tra một số có phải bội của 5 không

```
import java.io.*;
public class Boi5
{
    public static void Kiemtraboi5(int m)
    {
        if(m%5!=0)
            System.out.println(m+"Khong la boi so cua 5");
        else
            System.out.println(m+"La boi cua 5");
    }
    public static void main(String[] args)
    {
        Boi5 dt1=new Boi5();// tao doi tuong dt1
        dt1.Kiemtraboi5(9);
        Boi5 dt2=new Boi5();// tao doi tuong dt2
        dt2.Kiemtraboi5(10);
    }
}
```

Chương trình nhận các ký tự gõ vào từ bàn phím và chỉ nhận các ký tự số.

```
class Convert
{
    public static void main (String [] args) throws java.io.IOException
    {
        System.out.print ("Please enter a number: ");
        int num = 0;
        int ch;
        while ((ch = System.in.read ()) != '\n')
            if (ch >= '0' && ch <= '9') {
                num *= 10;
                num += ch - '0';
            }
            else
                break;
        System.out.println ("num = " + num + "num squared = " + num *
num);
    }
}//class
```



Kiểu dữ liệu

- Kiểu dữ liệu cơ sở (Primitive Data Types)
- Kiểu dữ liệu tham chiếu (Reference data types)

Kiểu dữ liệu cơ sở

- 4 kiểu integers:

- byte, short, int, long
-

- 2 kiểu số thực:

- float, double

- 1 kiểu characters:

- char

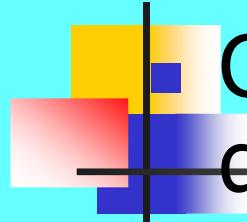
- 1 kiểu boolean :

- boolean

Kiểu dữ liệu cơ sở

Type	Storage	Min Value	Max Value
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	< -9 x 10 ¹⁸	> 9 x 10 ¹⁸
float	32 bits	+/- 3.4 x 10 ³⁸ with 7 significant digits	
double	64 bits	+/- 1.7 x 10 ³⁰⁸ with 15 significant digits	

Characters & Boolean

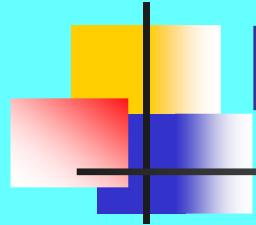


Character literals are delimited by single quotes:

'a' 'x' '7' '\$' ',' '\n'

- A boolean value represents a true or false condition
- The reserved words true and false are the only valid values for a boolean type

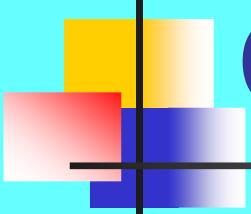
```
boolean done = false;
```



Kiểu dữ liệu tham chiếu

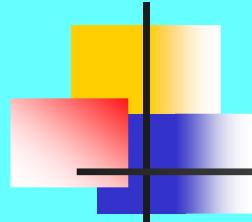
- Mảng (Array)
- Lớp (Class)
- Interface

Confidential



Các toán tử

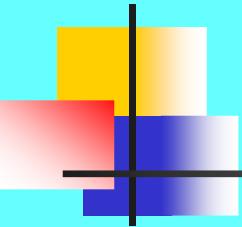
- Các loại toán tử:
 - Toán tử số học (Arithmetic operators)
 - Toán tử dạng Bit (Bitwise operators)
 - Toán tử so sánh (Relational operators)
 - Toán tử logic (Logical operators)
 - Toán tử điều kiện (Conditional operator)
 - Toán tử gán (Assignment operator)



Toán tử số học

Arithmetic Operators

+	Addition (Phép cộng)
-	Subtraction (Phép trừ)
*	Multiplication (Phép nhân)
/	Division (Phép chia)
%	Modulus (Lấy số dư)
++	Increment (Tăng dần)
--	Decrement (Giảm dần)



`+=`

Phép cộng và gán

`-=`

Phép trừ và gán

`*=`

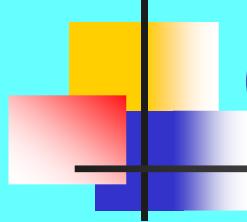
Phép nhân và gán

`/=`

Phép chia và gán

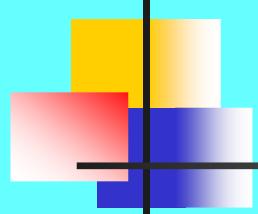
`%=`

Phép lấy số dư và gán



Toán tử so sánh (Relational Operators)

<code>==</code>	So sánh bằng
<code>!=</code>	So sánh khác
<code><</code>	Nhỏ hơn
<code>></code>	Lớn hơn
<code><=</code>	Nhỏ hơn hoặc bằng
<code>>=</code>	Lớn hơn hoặc bằng



Toán tử Bit (Bitwise Operators)

~

Phủ định (NOT)

&

Và (AND)

|

Hết (OR)

^

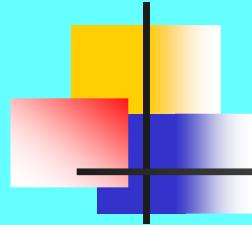
Exclusive OR (XOR)

>>

Dịch sang phải (Shift right)

<<

Dịch sang trái (Shift left)



Toán tử Logic (Logical Operators)

&&

Logical AND

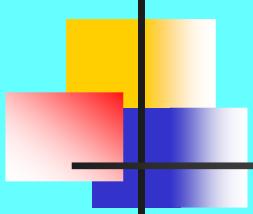
||

Logical OR

!

Logical unary NOT

Confidential



Toán tử điều kiện (Conditional Operator)

- Cú pháp

Biểu thức 1 ? Biểu thức 2 : Biểu thức 3;

- **Biểu thức 1**

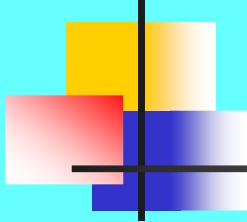
Điều kiện kiểu Boolean trả về giá trị True hoặc False

- **Biểu thức 2**

Trả về giá trị nếu kết quả của mệnh đề 1 là True

- **Biểu thức 3**

Trả về giá trị nếu kết quả của mệnh đề 1 là False



Toán tử gán (Assignment Operator)

= Assignment (Phép gán)

Giá trị có thể ~~đ~~ được gán cho nhiều biến số

- Ví dụ

a = b = c = d = 90;

Thứ tự ưu tiên của các toán tử

Thứ tự	Toán tử
1.	trong ngoặc tính trước
2.	Các toán tử đơn như <code>+, -, ++, --</code>
3.	Các toán tử số học và các toán tử dịch như <code>*, /, +, -, <, >, <<, >></code>
4.	Các toán tử quan hệ như <code>>, <, >=, <=, =, !=</code>
5.	Các toán tử logic và Bit như <code>&&, , &, I, ^</code>
6.	Các toán tử gán như <code>=, *=, /=, +=, -=</code>

Operator Precedence

a + b + c + d + e
1 2 3 4 5

a - b / c + d * e
3 1 4 2

a / (b + c) - d % e
2 1 4 5

a / (b * (c + (d - e))))
4 3 2 1

Confidential

Assignment Revisited

The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

answer = sum / 4 + MAX * lowest;

4 1 3 2



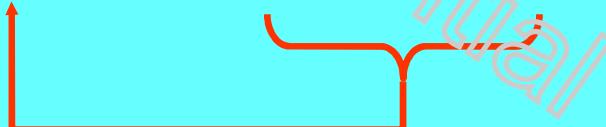
Then the result is stored in the variable on the left hand side

Assignment Revisited

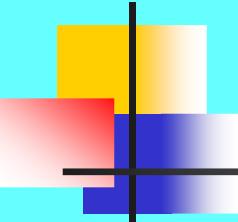
The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of count

```
count = count + 1;
```



Then the result is stored back into count
(overwriting the original value)



Ép kiểu (Type Casting)

- Kiểu dữ liệu này được chuyển đổi sang một kiểu dữ liệu khác

Cú pháp: **(kiểu_mới)biểu_thức;**

Ví dụ :

nguyên;

thực;

Ví dụ :

int m=3, n=2;

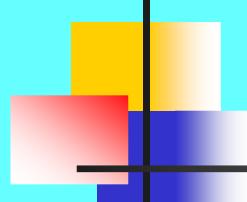
float x1,x2;

x1 = m/n; // x1 nhận giá trị là số

x2 = m/float(n); // x2 nhận giá trị là số

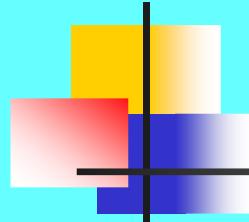
float c = 34.29675;

int b = (int)c + 10;



Các kí tự định dạng xuất dữ liệu (Escape Sequences)

Escape Sequence	Mô tả
\n	Xuống dòng mới
\r	Chuyển con trỏ đến đầu dòng hiện hành
\t	Chuyển con trỏ đến vị trí dừng Tab kế tiếp (ký tự Tab)
\\	In dấu \
\'	In dấu nháy đơn (')



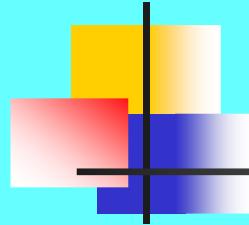
Hằng, Biến, Biểu thức

* **Hằng :**

- Hằng là đại lượng có giá trị không thay đổi trong suốt quá trình thực thi của chương trình

* **Biến :**

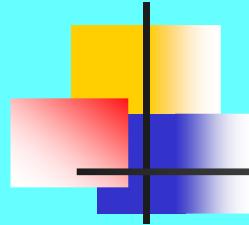
- Biến là đại lượng có giá trị thay đổi trong suốt quá trình thực thi của chương trình
- Khai báo biến để chương trình dành riêng vùng nhớ thích hợp cho biến đó.
- Mọi lệnh truy cập đến biến là truy cập đến giá trị của nó.



Hằng, Biến, Biểu thức

* *Biến* :

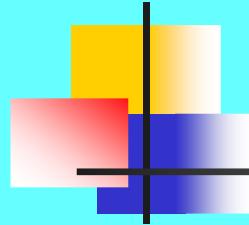
- Cú pháp khai báo biến :
datatype identifier [=value][, identifier [=value]...];
- Khai báo biến số gồm 3 thành phần:
 - Kiểu dữ liệu của biến số
 - Tên biến
 - Giá trị ban đầu của biến (không bắt buộc)



Hằng, Biến, Biểu thức

* **Biểu thức :**

- Là một dãy các toán tử và toán hạng
 - + Toán hạng: có thể là hằng, hàm, biến, ...
 - + Toán tử : +, -, *, /, \$, %, & ...
- Mỗi biểu thức sẽ có giá trị thuộc một kiểu dữ liệu nào đó
- Ưu tiên cao nhất cho biểu thức con trong cặp ngoặc đơn



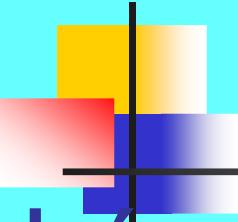
Khai báo mảng

* Mảng một chiều:

- **datatype identifier [];**
- **datatype identifier [] = new datatype[size];**
- **datatype identifier [] = {value1,value2,...,valueN};**

* Mảng hai chiều :

- **datatype identifier [][];**
- **Datatype [][] identifier;**



Lớp và phương thức (Classes & Methods)

Confidential

Lớp trong Java

```
// comments about the class
```

```
public class MyProgram
```

```
{
```

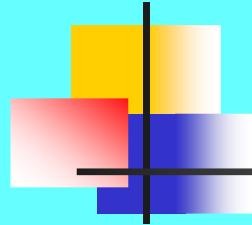
class body

```
}
```

class header

Confidential

Comments can be added almost anywhere

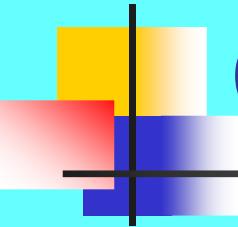


Lớp trong Java

- Cú pháp khai báo lớp (Class)

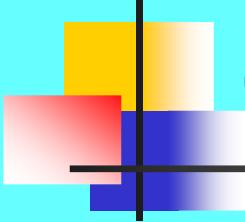
```
class tênlớp  
{  
    kiểu tênbiến;  
    :  
    met_datatype tênphươngthúc([dsthamsô])  
    :  
}
```

Confidential



Các lớp nội (Nested Classes)

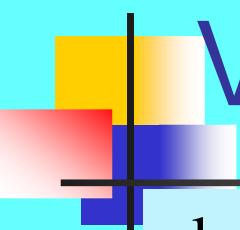
- Một lớp được định nghĩa bên trong một lớp khác. Lớp đó ~~đ~~ được gọi là “lớp nội” (Nesting)
- Các kiểu lớp nội:
 - Static
 - Non-static



Phương thức (Methods in Classes)

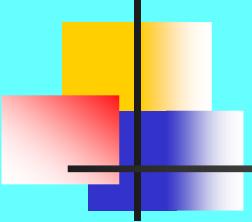
- Phương thức được định nghĩa như là một hành động hoặc một tác vụ thật sự của đối tượng
- Cú pháp

```
accessSpecifier modifier datatype methodName(parameterList)
{
    //body of method
}
```



Ví dụ về sử dụng phương thức

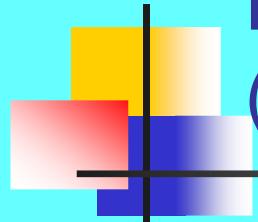
```
class Ex3
{  int x = 10;          // variable
   public void show( )
   {
      System.out.println(x);
   }
   public static void main(String args[ ])
   {
      Ex3 t = new Ex3( );        // tao doi tuong t
      t.show( );                // method call
      Ex3 t1 = new Ex3( );       // tao doi tuong t1
      t1.x = 20;
      t1.show( );
   }
}
```



Các chỉ định truy xuất của phương thức (Access specifiers)

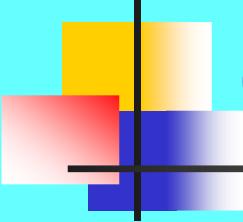
- Công cộng (public)
- Riêng tư (private)
- Bảo vệ (protected)

Confidential



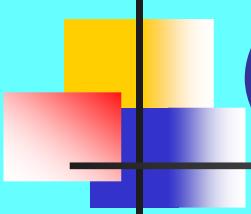
Những phương thức được nạp chồng : (Methods Overloading)

- Những phương thức được nạp chồng :
 - Cùng ở trong một lớp
 - Có cùng tên
 - Khác nhau về danh sách tham số
- Những phương thức được nạp chồng là một hình thức đa hình trong quá trình biên dịch (compile time)



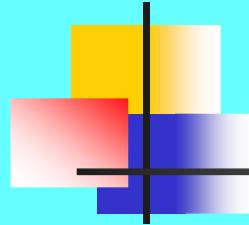
Ghi đè phương thức (Methods Overriding)

- Những phương thức được ghi đè:
 - Có mặt trong lớp cha (superclass) cũng như lớp kế thừa (subclass)
 - Được định nghĩa lại trong lớp kế thừa (subclass)
- Những phương thức được ghi đè là một hình thức đa hình trong quá trình thực thi (Runtime)



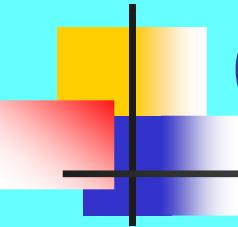
Phương thức khởi tạo (Class Constructors)

- Là một phương thức đặc biệt dùng để khởi tạo giá trị ~~cho~~ các biến thành viên của lớp đối tượng
- Có cùng tên với ~~tên~~ lớp và không có giá trị trả về
- Được gọi khi đối tượng ~~được~~ được tạo ra
- Có 2 loại:
 - Tường minh (Explicit constructors)
 - Ngầm định (Implicit constructors)



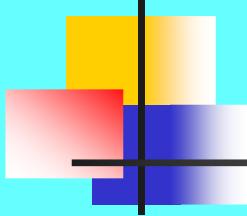
Phương thức khởi tạo của lớp dẫn xuất (Derived class constructors)

- Có cùng tên với lớp dẫn xuất (subclass)
- Nếu ở lớp dẫn xuất muốn gọi hàm constructor của lớp cơ sở thì câu lệnh gọi hàm constructor của lớp cơ sở phải là câu lệnh đầu tiên trong hàm constructor của lớp dẫn xuất
- Dùng từ khoá *super*



Các lệnh điều khiển

- Điều khiển rẽ nhánh:
 - Mệnh đề **if-else**
 - Mệnh đề **switch-case**
- Vòng lặp (Loops):
 - Vòng lặp **while**
 - Vòng lặp **do-while**
 - Vòng lặp **for**



Lệnh if-else

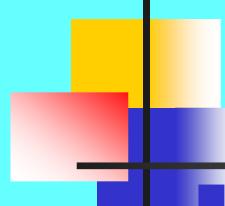
- Cú pháp

if (đk) lệnh1 ;

else

lệnh2;

Confidential



Lệnh switch-case

Cú pháp

switch (bt)

{

case 'value1': lệnh1;

break;

case 'value2': lệnh2;

break;

:

:

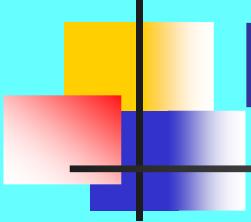
case 'valueN': lệnhN;

break;

default: lệnh(N+1);

}

Confidential



Lệnh lặp while

- Cú pháp

```
while(đk)
```

```
{
```

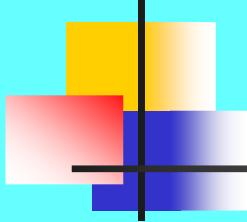
```
    lệnh;
```

```
:
```

```
:
```

```
}
```

Confidential

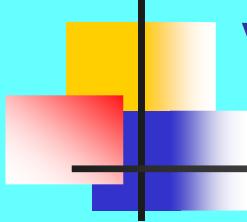


Lệnh lặp do-while

- Cú pháp

```
do  
{  
    action statements;  
    :  
    :  
}  
while(condition);
```

Confidential



Vòng lặp for

- Cú pháp

```
for(dskt ; btdk ; btkc)
```

```
{
```

```
    lệnh;
```

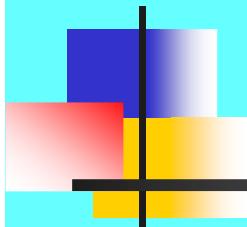
```
:
```

```
:
```

```
}
```

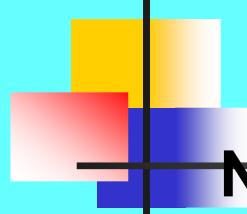
Confidential

CHƯƠNG 4 : NGÔN NGỮ LẬP TRÌNH JAVA



Confidential

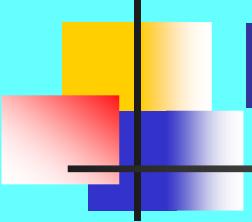
4.3 Gói & Interface (Packages & Interfaces)



Giới thiệu

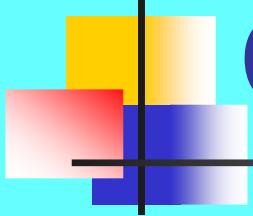
Những thành phần cơ bản của chương trình Java:

- Gói (Packages)
- Giao diện (Interfaces)
- **Những phần của một chương trình Java:**
 - Lệnh khai báo gói(package)
 - Lệnh chỉ định gói được dùng (Lệnh import)
 - Khai báo lớp public (một file java chỉ chứa 1 lớp public class)
 - Các lớp khác (classes private to the package)
- Tập tin nguồn Java có thể chứa tất cả hoặc một vài trong số các phần trên.



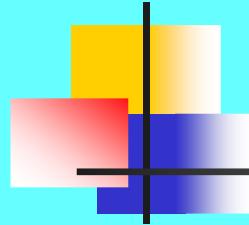
Interfaces

- Interface chính là lớp abstract thuần tuý
- Khai báo những phương thức mà lớp sử dụng
- Trong Java 1 lớp chỉ có thể dẫn xuất từ 1 lớp duy nhất tại cùng một thời điểm, nhưng có thể dẫn xuất cùng lúc nhiều Interface
- Trong Interface không chứa những phương thức cụ thể (concrete methods)
- interface cần phải được implements.



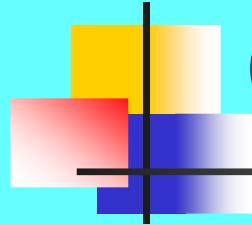
Các bước tạo interface

- Định nghĩa Interface
 - Biên dịch Interface
 - Implements Interface
-
- Tính chất của interface:
 - Tất cả phương thức trong interface phải là **public**.
 - Các phương thức phải được định nghĩa trong lớp dẫn xuất interface đó.



Sử dụng Interface

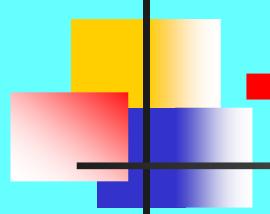
- Interface không thể dẫn xuất từ lớp khác, nhưng có thể dẫn xuất từ những interface khác
- Nếu một lớp dẫn xuất từ một interface mà interface đó dẫn xuất từ các interface khác thì lớp đó phải định nghĩa tất cả các phương thức có trong các interface đó. Nếu không lớp đó trở thành lớp abstract
- Khi định nghĩa một interface mới có nghĩa là một kiểu dữ liệu tham chiếu mới được tạo ra.



Gói (Packages)

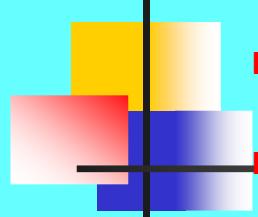
- Gói được xem tương tự như thư mục lưu trữ những lớp, những interfaces và các gói con khác.

Q Confidential



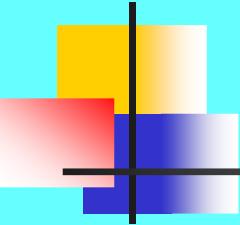
Những ưu điểm khi dùng gói (Package):

- Cho phép tổ chức các lớp vào những đơn vị nhỏ hơn
- Giúp tránh được tình trạng trùng lặp khi đặt tên lớp, tên interfaces .
- Cho phép bảo vệ các lớp.
- Tên gói (Package) có thể được dùng để nhận dạng chức năng của các lớp.

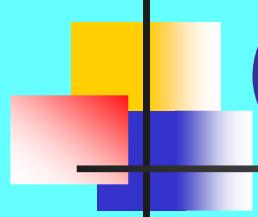


■ Những lưu ý khi tạo gói:

- Mã nguồn phải bắt đầu bằng lệnh ‘package’
- Mã nguồn phải nằm trong cùng thư mục mang tên của gói
- Tên gói nên ~~bắt~~ đầu bằng ký tự thường (lower case) để phân biệt giữa tên lớp đối tượng và tên gói
- Những lệnh khác phải viết phía dưới dòng khai báo gói là mệnh đề **import**, kế đến là các lệnh định nghĩa lớp đối tượng
- Những lớp đối tượng trong gói cần phải được biên dịch.
- Để chương trình Java có thể sử dụng những gói này, ta phải **import** gói vào trong mã nguồn

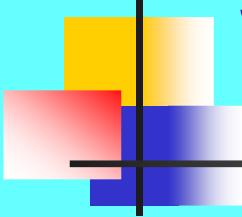


- Import gói (Importing packages):
 - Có 2 cách:
 - Xác định lớp hoặc interface của gói cần được import.
 - Hoặc có thể import toàn bộ gói
- vd : import java.util.Vector;
import java.util.*;



Các bước tạo ra gói (Package)

- Khai báo gói
- Import những gói chuẩn cần thiết
- Khai báo và định nghĩa các lớp đối tượng có trong gói
- Lưu thành tập tin **.java**, và biên dịch những lớp đối tượng đã được định nghĩa trong gói.



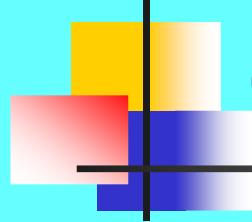
Sử dụng những gói do người dùng định nghĩa (user-defined packages)

- Mã nguồn của những chương trình này phải ở cùng thư mục của gói do người dùng định nghĩa. Nếu không ta phải thiết lập đường dẫn.
- Để những chương trình Java khác sử dụng những gói này, import gói vào trong mã nguồn
- Import những lớp đối tượng cần dùng
- Import toàn bộ gói
- Tạo tham chiếu đến những thành viên của gói



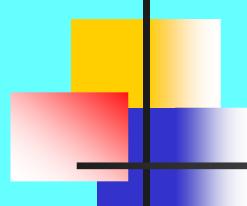
Thiết lập đường dẫn - xác lập CLASSPATH

- Là danh sách các thư mục, giúp cho việc tìm kiếm các tập tin lớp đối tượng tương ứng
- Nên xác lập CLASSPATH trong lúc thực thi (runtime), vì như vậy nó sẽ xác lập đường dẫn cho quá trình thực thi hiện hành



Gói và điều khiển truy xuất (Packages & Access Control)

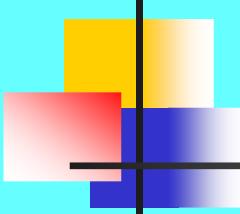
	public	protected	No modifier	private
Same class	Yes	Yes	Yes	Yes
Same package subclass	Yes	Yes	Yes	No
Same package non-subclass	Yes	Yes	Yes	No
Different package subclass	Yes	Yes	No	No
Different package non-subclass	Yes	No	No	No



Gói java.lang

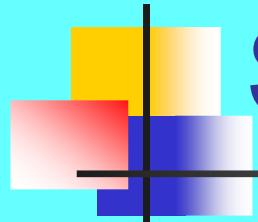
- Gói **java.lang** được import mặc định.
- Những lớp **Wrapper** cho các kiểu dữ liệu nguyên thủy:

Data type	Wrapper class
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short



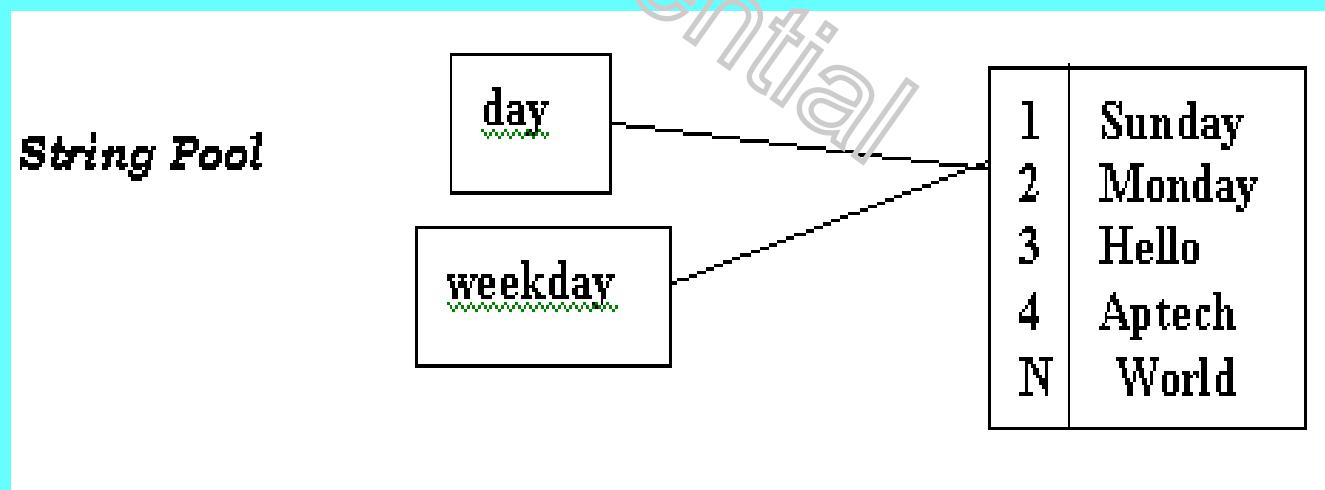
Lớp String

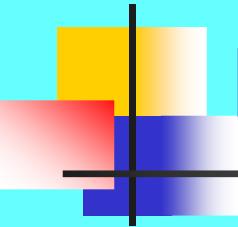
- Phương thức khởi tạo (Constructor):
 - **String str1 = new String();**
 - **String str2 = new String("Hello World");**
 - **char ch[] = {"A","B","C","D","E"};**
 - **String str3 = new String(ch);**
 - **String str4 = new String(ch,0,2);**



String Pool

- ‘String Pool’ đại diện cho tất cả các ký tự được tạo ra trong chương trình
- Khái niệm ‘String Pool’



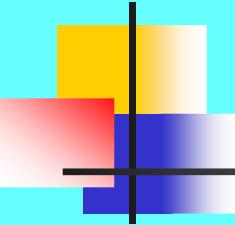


Những phương thức của lớp String

- **charAt()//**
- **startsWith()**
- **endsWith()**
- **copyValueOf()**
- **toCharArray()**
- **indexOf()**
- **toUpperCase()**
- **toLowerCase()**
- **trim()**
- **equals()**

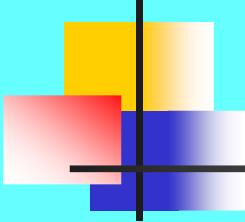
Lớp StringBuffer

- Cung cấp những phương thức khác nhau để thao tác trên đối tượng string (chuỗi ký tự)
- Những đối tượng của lớp này khá linh hoạt
- Cung cấp những phương thức khởi tạo (constructor) đã được nạp chèng (overloaded)
- Những phương thức của lớp **StringBuffer**:
 - **append()**
 - **insert()**
 - **charAt()**
 - **setCharAt()**
 - **setLength()**
 - **getChars()**
 - **reverse()**



Lớp `java.lang.Math`

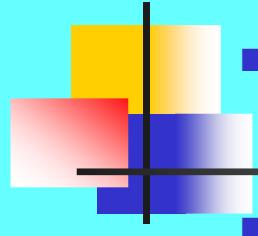
- `abs()`
- `ceil()// lam tron tren`
- `floor()// lam tron duoi`
- `max()`
- `min()`
- `round()`
- `random()`
- `sqrt()`
- `sin()`
- `cos()`
- `tan()`

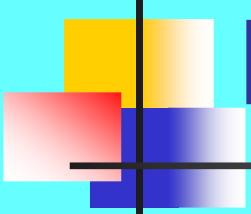


Lớp Runtime

- Đóng gói (Encapsulates) môi trường thực thi
- Dùng để quản lý bộ nhớ, và thi hành những tiến trình cộng nhêm
- Phương thức:
 - **exit(int)**
 - **freeMemory()**
 - **getRuntime()**
 - **gc()**
 - **totalMemory()**
 - **exec(String)**

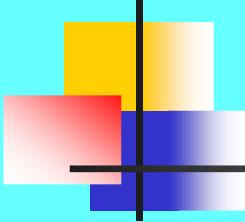
Lớp System

- 
- Cung cấp những luồng chuẩn như nhập (Input), xuất (Output) và các luồng lỗi (Error Streams)
 - Cung cấp khả năng truy xuất đến những thuộc tính của hệ thống thực thi Java, và những thuộc tính môi trường như phiên bản, đường dẫn, nhà cung cấp...
 - Phương thức:
 - **exit(int)**
 - **gc()**
 - **getProperties()**
 - **setProperties()**
 - **currentTimeMillis()**
 - **arrayCopy(Object, int, Object, int, int)**



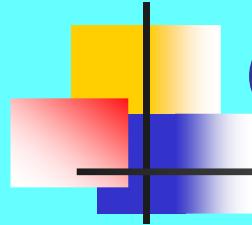
Lớp Class

- Đối tượng của lớp này che giấu trạng thái thực thi của đối tượng trong một ứng dụng Java.
- Đối tượng của lớp này có thể tạo ra bằng 1 trong 3 cách sau:
 - Sử dụng phương thức **getClass()** của đối tượng
 - Sử dụng phương thức tĩnh **forName()** của lớp để tạo ra một đối tượng của lớp đó trong lúc đặt tên cho lớp
 - Sử dụng đối tượng ClassLoader để nạp một lớp mới



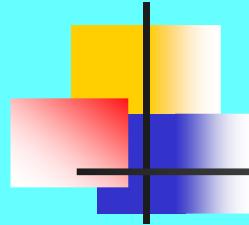
Lớp Object

- Là lớp cha tối cao của tất cả các lớp
- Phương thức:
 - **equals(Object)**
 - **finalize()**
 - **notify()**
 - **notifyAll()**
 - **toString()**
 - **wait()**



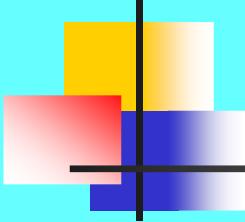
Gói java.util

- Cung cấp phần lớn những lớp Java hữu dụng và thường xuyên cần đến trong hầu hết các ứng dụng
- Giới thiệu những lớp trừu tượng sau:
 - Hashtable
 - Random
 - Vector
 - StringTokenizer



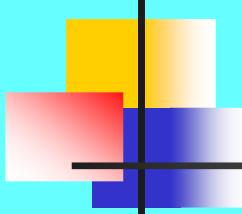
Lớp Hashtable

- Dẫn xuất từ lớp trừu tượng Dictionary
- Dùng để ~~hỗ trợ~~ kết những khóa vào những giá trị cụ thể
- Phương thức khởi tạo Hashtable:
 - **Hashtable(int)**
 - **Hashtable(int, float)**
 - **Hashtable()**



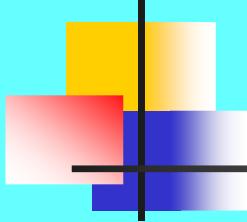
Những phương thức của lớp Hashtable

- **clear()**
- **done()**
- **contains(Object)**
- **containsKey(Object)**
- **elements()**
- **get(Object key)**
- **isEmpty()**
- **keys()**
- **put(Object, Object)**
- **rehash()**
- **remove(Object key)**
- **size()**
- **toString()**



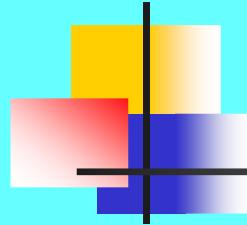
Lớp Random

- Những phương thức nhận giá trị ngẫu nhiên:
 - **nextDouble()**
 - **nextFloat()**
 - **nextGaussian()**
 - **nextInt()**
 - **nextLong()**
- Phương thức khởi tạo (Constructors):
 - **random()**
 - **random(long)**



Những phương thức của lớp Random

- **nextDouble()**
- **nextFloat()**
- **nextGaussian()**
- **nextInt()**
- **nextLong()**
- **setSeed(long)**

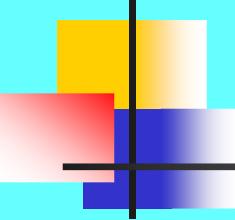


Lớp Vector

- Cung cấp khả năng co giãn cho mảng khi thêm phần tử vào mảng
- Lưu trữ những thành phần của kiểu Object
- Một Vector riêng rẽ có thể lưu trữ những phần tử khác nhau, đó là những instance của những lớp khác nhau
- Phương thức khởi tạo (Constructors):
 - **Vector(int)**
 - **Vector(int, int)**
 - **Vector()**

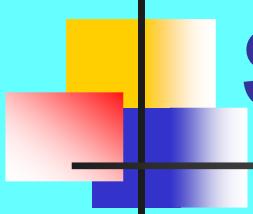
Những phương thức của lớp Vector

- **addElement(Object)**
- **capacity()**
- **clone()**
- **contains(Object)**
- **copyInto(Object [])**
- **elementAt(int)**
- **elements()**
- **ensureCapacity(int)**
- **firstElement()**
- **indexOf(Object)**
- **indexOf(Object, int)**
- **insertElementAt(Object, int)**
- **isEmpty()**
- **lastElement()**
- **lastIndexOf(Object)**
- **lastIndexOf(Object, int)**
- **removeAllElements()**
- **removeElement(Object)**
- **removeElementAt(int)**
- **setElementAt(Object, int)**
- **setSize(int)**
- **size()**
- **toString()**
- **trimToSize()**



Lớp StringTokenizer

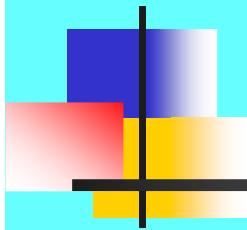
- Có thể được dùng để tách một chuỗi thành những thành phần câu thành của nó (constituent tokens)
- Ký tự phân cách có thể được chỉ định khi một đối tượng **StringTokenizer** được khởi tạo
- Phương thức khởi tạo (Constructors):
 - **StringTokenizer(String)**
 - **StringTokenizer(String, String)**
 - **StringTokenizer(String, String, Boolean)**
- Lớp **StringTokenizer** sử dụng giao diện liệt kê (enumeration interface)



Những phương thức của lớp StringTokenizer

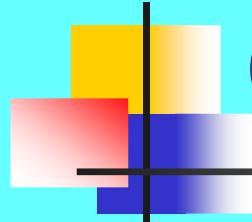
- **countTokens()**
- **hasMoreElements()**
- **hasMoreTokens()**
- **nextElement()**
- **nextToken()**
- **nextToken(String)**

CHƯƠNG 4 : NGÔN NGỮ LẬP TRÌNH JAVA



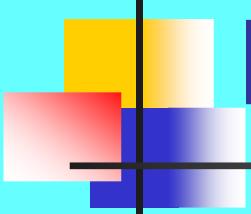
Confidential

4.4 Xử lý biệt lệ



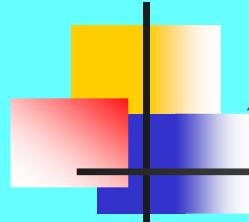
Giới thiệu về biệt lệ

- Là một kiểu lỗi đặc biệt
- Nó xảy ra ~~trong~~ trong thời gian thực thi đoạn lệnh
- Thông thường các điều kiện thực thi chương trình gây ra ~~biệt~~ biệt lệ
- Nếu các điều kiện này không được xử lý, thì việc thực thi có thể kết thúc đột ngột



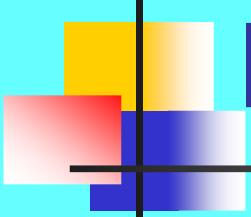
Mục đích của việc xử lý biệt lệ

- Giảm thiểu việc kết thúc bất thường của hệ thống và của chương trình.
- Ví dụ, thao tác xuất/nhập trong một tập tin, nếu việc chuyển đổi kiểu dữ liệu không thực hiện đúng, một biệt lệ sẽ xảy ra và chương trình bị hủy mà không đóng tập tin. Lúc đó tập tin sẽ bị hư hại và các nguồn tài nguyên được cấp phát cho tập tin không được thu hồi lại cho hệ thống.



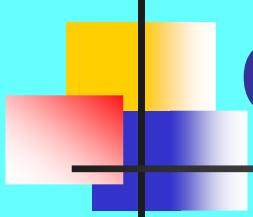
Xử lý biệt lệ

- Khi một biệt lệ xảy ra, đối tượng tương ứng với biệt lệ đó sẽ được tạo ra.
- Đối tượng này sau đó được truyền tới phương thức nơi mà biệt lệ xảy ra.
- Đối tượng này chứa các thông tin chi tiết về biệt lệ. Thông tin này có thể nhận được và xử lý.
- Lớp 'throwable' mà Java cung cấp là lớp trên nhất của lớp biệt lệ.



Mô hình xử lý biệt lệ

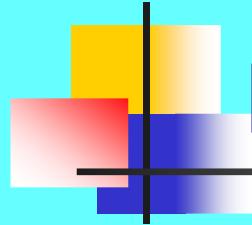
- Mô hình được biết đến là mô hình ‘catch and throw’
Confidential
- Khi một lỗi xảy ra, biệt lệ sẽ được chặn và được vào một khối
- Từ khóa để xử lý biệt lệ
 - try
 - catch
 - throw
 - throws
 - finally



Cấu trúc của mô hình xử lý biệt lệ

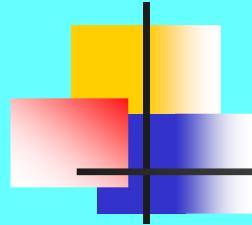
- Cú pháp

```
try { .... }  
catch(Exception e1) { .... }  
catch(Exception e2) { .... }  
catch(Exception eN) { .... }  
finally { .... }
```



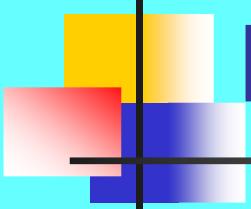
Mô hình ‘Catch and Throw’ nâng cao

- Người lập trình chỉ quan tâm tới các lỗi khi cần thiết.
- Một thông báo lỗi có thể được cung cấp trong exception-handler.



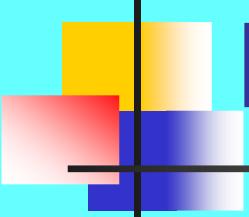
Khối ‘try’ và ‘catch’

- Được sử dụng để thực hiện trong mô hình ‘catch and throw’ của xử lý biệt lệ.
- Khối lệnh ‘try’ gồm tập hợp các lệnh thực thi
- Một hoặc nhiều khối lệnh ‘catch’ có thể tiếp theo sau một khối lệnh ‘try’
- Các khối lệnh ‘catch’ này bắt biệt lệ trong khối lệnh ‘try’.



Khối lệnh ‘try’ và ‘catch’ (tt)

- Để bắt bất kỳ loại biệt lệ nào, ta có thể chỉ ra kiểu biệt lệ là ‘Exception’
catch(Exception e)
- Khi biệt lệ bị bắt không biết thuộc kiểu nào, chúng ta có thể sử dụng lớp ‘Exception’ để bắt biệt lệ đó.
- Lỗi sẽ được truyền thông qua khối lệnh ‘try catch’ cho tới khi chúng bắt gặp một ‘catch’ tham chiếu tới nó, nếu không chương trình sẽ bị kết thúc



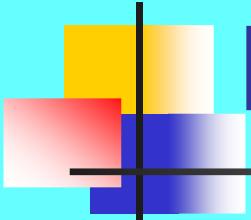
Khối lệnh chứa nhiều Catch

- Các khối chứa nhiều ‘catch()’ xử lý các kiểu biệt lệ khác nhau một cách độc lập.
- Ví dụ

```
try
{ doFileProcessing();
  displayResults();
} catch(LookupException e)
{ handleLookupException(e);
}
catch(Exception e)
{ System.err.println("Error:"+e.printStackTrace()); }
```

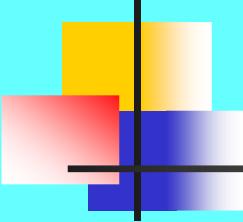
Khối lệnh chứa nhiều Catch (tt)

- Khi sử dụng các ‘try’ lồng nhau, khối ‘try’ bên trong ~~được~~ thực thi trước.
- Bất kỳ biến nào bị chặn trong khối lệnh ‘try’ sẽ bị bắt trong khối lệnh ‘catch’ tiếp ngay sau.
- Nếu khối lệnh ‘catch’ thích hợp không được tìm thấy, thì các khối ‘catch’ của khối ‘try’ bên ngoài sẽ được xem xét
- Ngược lại, Java Runtime Environment sẽ xử lý biệt lập.



Khối ‘finally’

- Thực hiện tất cả các việc thu dọn khi biệt lệ xảy ra
- Có thể sử dụng kết hợp với khối ‘try’
- Chứa các câu lệnh thu hồi tài nguyên về cho hệ thống hay lệnh in ra các câu thông báo:
 - Đóng tập tin
 - Đóng lại bộ kết quả (được sử dụng trong chương trình cơ sở dữ liệu)
 - Đóng lại các kết nối được tạo trong cơ sở dữ liệu.



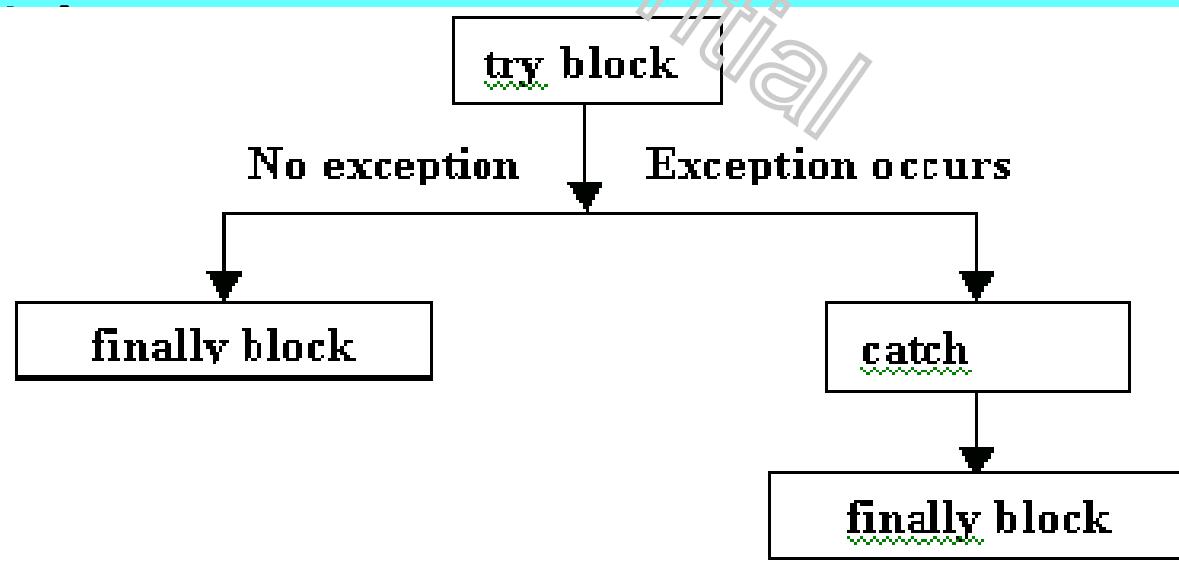
Khối 'finally' (tt)

- Ví dụ

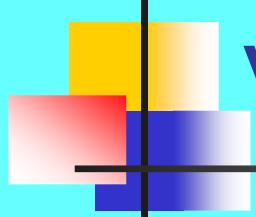
```
try
{
    doSomethingThatMightThrowAnException();
}
finally
{
    cleanup();
}
```

Khối ‘finally’ (tt)

- Khối này tùy chọn có hoặc không có
- Được đặt sau khối ‘catch’ cuối cùng.
- Khối ‘finally’ bảo đảm lúc nào cũng được thực hiện bất chấp biệt lệ có xảy ra hay không



Flow of the ‘try’, ‘catch’ and ‘finally’ blocks



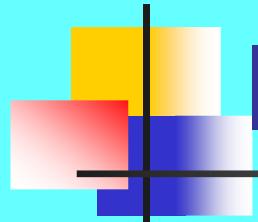
Các biệt lệ được định nghĩa với lệnh ‘throw’ và ‘throws’

- Các biệt lệ ném ra với sự trợ giúp của từ khoá throw.

- Throw là một đối tượng của một lớp biệt lệ.

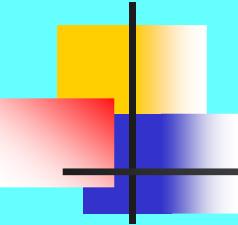
- Ví dụ của lệnh ‘throw’

```
try{  
    if (flag < 0)  
    {  
        throw new MyException( );// user-defined  
    }  
}
```



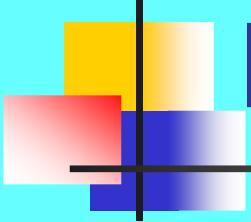
Các biệt lệ được định nghĩa với lệnh ‘throw’ và ‘throws’(tt)

- Lớp ‘Exception’ implements interface ‘Throwable’ và cung cấp các tính năng hữu dụng để phân phối cho các biệt lệ.
- Một lớp con của lớp Exception là một biệt lệ mới.



Danh sách các biệt lệ

- RuntimeException
- ArithmeticException
- IllegalAccessException
- IllegalArgumentException
- ArrayIndexOutOfBoundsException
- NullPointerException
- SecurityException
- ClassNotFoundException



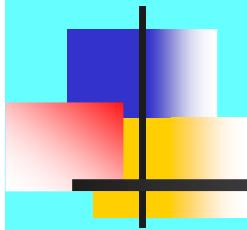
Danh sách các biệt lệ (tt)

- NumberFormatException
- AWTException
- IOException
- FileNotFoundException
- EOFException
- NoSuchMethodException
- InterruptedException

Danh sách các biệt lệ (tt)

Tên lớp ngoại lệ	Ý nghĩa
Throwable	Đây là lớp cha của mọi lớp ngoại lệ trong Java
Exception	Đây là lớp con trực tiếp của lớp Throwable, nó mô tả một ngoại lệ tổng quát có thể xảy ra trong ứng dụng
RuntimeException	Lớp cơ sở cho nhiều ngoại lệ java.lang
ArthmeticException	Lỗi về số học, ví dụ như ‘chia cho 0’.
IllegalAccessException	Lớp không thể truy cập.
IllegalArgumentException	Đối số không hợp lệ.
ArrayIndexOutOfBoundsException	Lỗi truy cập ra ngoài mảng.
NullPointerException	Khi truy cập đối tượng null.
SecurityException	Cơ chế bảo mật không cho phép thực hiện.
ClassNotFoundException	Không thể nạp lớp yêu cầu.
NumberFormatException	Việc chuyển đổi từ chuỗi sang số không thành công.
AWTException	Ngoại lệ về AWT
IOException	Lớp cha của các lớp ngoại lệ I/O
FileNotFoundException	Không thể định vị tập tin
EOFException	Kết thúc một tập tin.
NoSuchMethodException	Phương thức yêu cầu không tồn tại.
InterruptedException	Khi một luồng bị ngắt.

CHƯƠNG 4 : NGÔN NGỮ LẬP TRÌNH JAVA

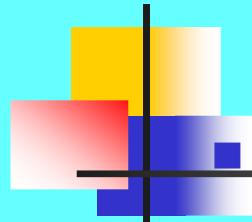


Confidential

4.5 LẬP TRÌNH GIAO DIỆN VỚI AWT-SWING

GIỚI THIỆU VỀ AWT

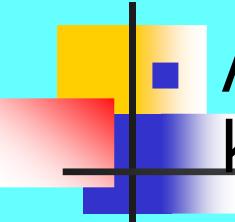
- AWT viết tắt của **Abstract Windowing Toolkit**



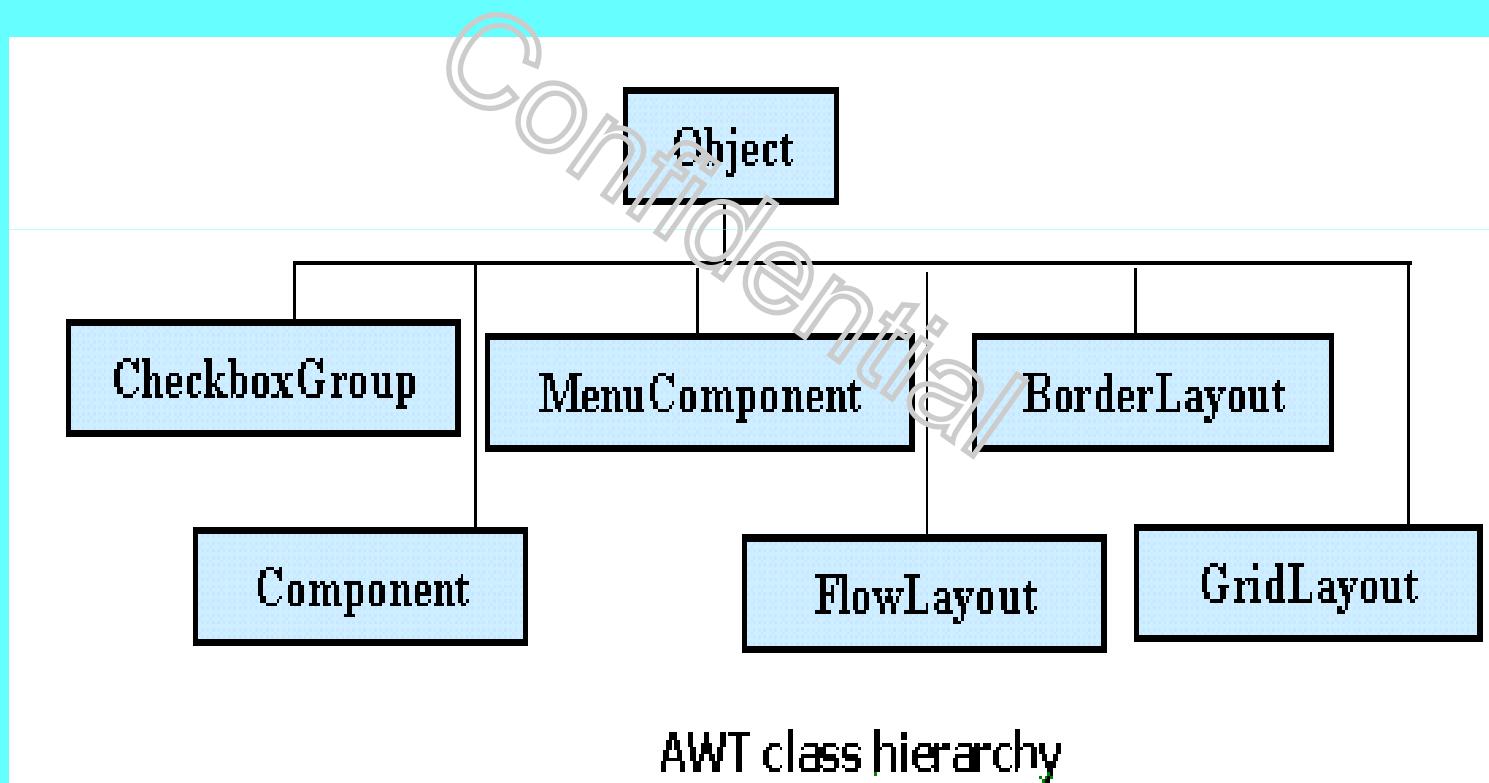
AWT là tập hợp các lớp Java cho phép chúng ta tạo một GUI

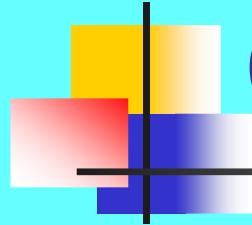
- Cung cấp ~~các~~ thành phần khác nhau để tạo chương trình GUI như:
 - Containers
 - Components
 - Layout managers
 - Graphics và drawing capabilities
 - Fonts
 - Events

Confidential



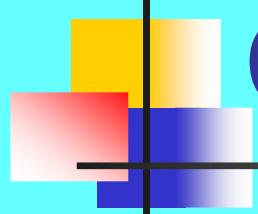
AWT bao gồm các lớp, interfaces và các gói khác





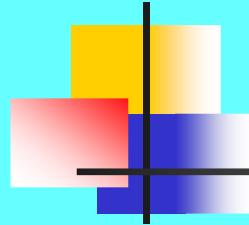
Components

- Tất cả các thành phần cấu tạo nên chương trình GUI được gọi là component.
- **Ví dụ**
 - Containers,
 - textfields, labels, checkboxes, textareas
 - scrollbars, scrollpanes, dialog



Containers

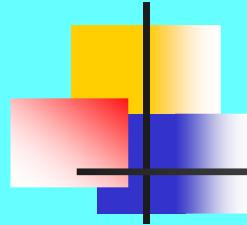
- Là thành phần mà có thể chứa các thành phần khác
- Có các frames, panels, latches, hooks
- Java.awt chứa một lớp có tên là Container. Lớp này ~~được~~ 2 lớp dẫn xuất trực tiếp và không trực tiếp gồm:
 - Frames
 - Panels



Frames

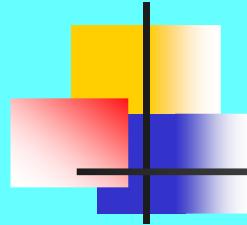
- Là các cửa sổ
- Là lớp con của lớp Windows
- Được hiển thị trong một cửa sổ và có đường viền

Confidential



Panels

- Là các vùng chứa trong một cửa sổ.
- Hiển thị trong một cửa sổ mà trình duyệt hoặc appletviewer cung cấp và không có đường viền.
- Được sử dụng để nhóm một số các thành phần
- Một panel không thể nhìn thấy vì thế chúng ta cần phải thêm nó vào frame.



Dialog

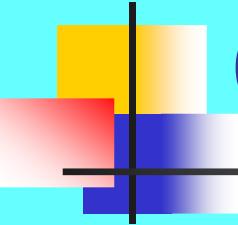
- Là một lớp con của lớp Window
- Đối tượng dialog được cấu trúc như sau :

Frame myframe = new Frame("My frame");

String title = "Title";

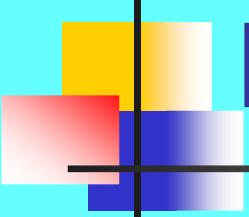
boolean modal = true;

Dialog dlg = new Dialog(myframe, title, modal);



Các Components khác

- Ví dụ
 - textfields, labels, checkboxes, textareas
 - scrollbars, scrollpanes, dialog

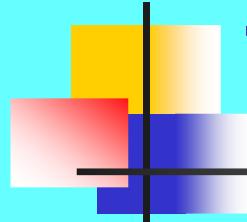


Label

- Được dùng để hiển thị chuỗi (String)
- Các hàm tạo dựng:
 - **Label()**
 - **Label(String labelText)**
 - **Label(String labelText, int alignment)**
- Các phương thức:
 - **setFont(Font f)**
 - **setText(String s)**
 - **getText()**

TextField

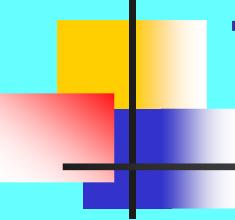
- Là điều khiển text, cho phép hiển thị text hoặc cho user nhập dữ liệu vào.
- Các hàm dựng:
 - **TextField()**
 - **TextField(int columns)**
 - **TextField(String s)**
 - **TextField(String s, int columns)**
- Các phương thức:
 - **setEchoChar(char)**
 - **setText(String s)**
 - **getText()**
 - **setEditable(boolean)**
 - **isEditable()**



TextArea

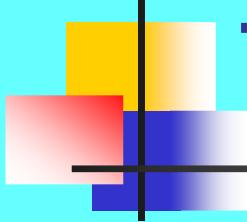
- Được dùng khi chuỗi hiển thị có từ hai dòng trở ~~lên~~ lên.

Confidential



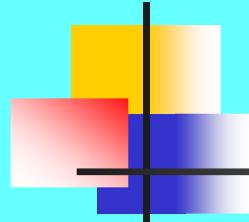
TextArea (tt...)

- Các hàm dựng:
 - **TextArea()**
 - **TextArea(int rows, int cols)**
 - **TextArea(String text)**
 - **TextArea(String text,int rows, int cols)**



Các phương thức của TextArea

- **setText(String)**
- **getText()**
- **setEditable(boolean)**
- **isEditable()**
- **insertText(String, int)**
- **replaceText(String, int, int)**



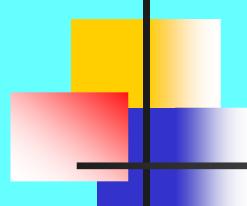
Button

- Các nút ấn là cách dễ nhất để lấy các sự kiện của người dùng.
- Các hàm tạo dựng:
 - **Button()**
 - **Button(String text)**

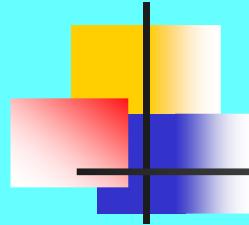
Checkboxes và RadioButtons

- Checkboxes cho phép người dùng đưa ra nhiều chọn lựa
- Radiobuttons chỉ cho phép người dùng duy nhất một chọn lựa
- Các hàm dựng để tạo checkbox:
 - **Checkbox()**
 - **Checkbox(String text)**
- Để tạo radiobutton, ta phải tạo đối tượng CheckBoxGroup trước khi tạo button

Choice

- 
- Lớp ‘Choice’ cho phép ta tạo danh sách có nhiều chọn lựa
 - Ví dụ

```
Choice colors=new Choice( );
colors.addItem("Red");
colors.addItem("Green");
```



Layout Manager

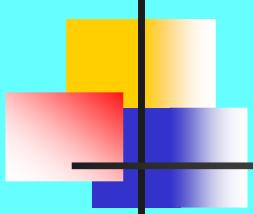
- Các loại layout manager khác nhau:
 - Flow Layout
 - Border Layout
 - Card Layout
 - Grid Layout
 - GridBagConstraints
 - GridBag Layout
- Layout manager được thiết lập bằng cách gọi phương thức ‘setLayout()’



FlowLayout

- FlowLayout là layout manager mặc định cho các applet và các panel
- Với FlowLayout các component sẽ được sắp xếp từ góc trái trên đến góc phải dưới của màn hình theo từng hàng.
- Các constructor:

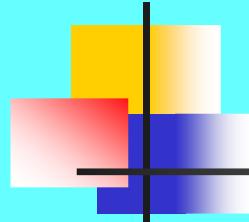
FlowLayout mylayout = new FlowLayout();
FlowLayout exLayout = new
flowLayout(FlowLayout.RIGHT);



BorderLayout

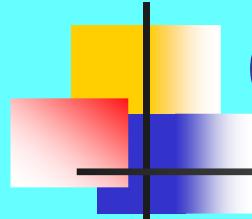
- BorderLayout mặc định cho Window, Frame và Dialog
- Trình quản lý này có thể bố trí container thành 5 vùng, NORTH, EAST, SOUTH, WEST và CENTER.
- **Ví dụ:** Để thêm một thành phần vào vùng North của container

```
Button b1= new Button("North Button");
setLayout(new BorderLayout());
add(b1, BorderLayout.NORTH);
```



CardLayout

- Có thể lưu trữ một danh sách các kiểu layout khác nhau
- Mỗi layout được xem như một thẻ (card)
- Thẻ thường là đối tượng Panel
- Một thành phần độc lập như button sẽ điều khiển các thẻ được đặt ở phía trên nhất
- Các bước để tạo CardLayout:
 - Bố trí layout của panel chính là CardLayout
 - Lần lượt thêm các panel khác vào panel chính



GridLayout

- Hỗ trợ việc chia container thành một lưới
- Các thành phần được bố trí trong các ô của lưới.
- Một ô lưới nên chứa ít nhất một thành phần
- Kiểu layout này được sử dụng khi tất cả các components có cùng kích thước
- Hàm constructor

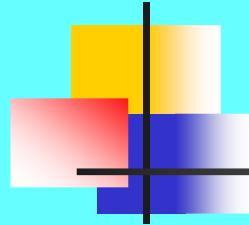
GridLayout gl = new GridLayout(no. of rows, no. of columns);



GridLayout

Bố trí các thành phần một cách chính xác

- Các thành phần không cần có cùng kích thước
- Các thành phần được sắp xếp trong một lưới chứa các dòng và các cột
- Thứ tự đặt các thành phần không tuân theo hướng từ trái-sang-phải và trên-xuống-dưới
- Hàm constructor
GridLayout gb = new GridLayout();



GridLayout

- Để sử dụng layout này, bạn cần phải biết thông tin về kích cỡ và cách bố trí của các thành phần
- Lớp ‘`GridBagLayout`.`Constraints`’ lưu trữ tất cả các thông tin mà lớp GridLayout yêu cầu: Vị trí và kích thước mỗi thành phần



Các lớp quản lý cách tổ chức các thành phần giao diện

Tên lớp	Mô tả
FlowLayout	Xếp các thành phần giao diện trước tiên theo hàng từ trái qua phải, sau đó theo cột từ trên xuống dưới. Cách sắp xếp này là mặc định đối với Panel và Applet.
GridLayout	Các thành phần giao diện được sắp xếp trong các ô lưới hình chữ nhật lần lượt theo hàng từ trái qua phải và theo cột từ trên xuống dưới trong một phần tử chung. Mỗi thành phần giao diện chứa trong một ô.
BorderLayout	Các thành phần giao diện (ít hơn 5) được đặt vào các vị trí theo các hướng: north (bắc), south (nam), west (tây), east (đông) và center (trung tâm). Cách sắp xếp này là mặc định đối với lớp Window, Frame và Dialog.
GridBagLayout	Cho phép đặt các thành phần giao diện vào lưới hình chữ nhật, nhưng một thành phần có thể chiếm nhiều ô.

Các phương pháp thiết kế *layout*

Để biết *layout* hay để đặt lại *layout* cho chương trình ứng dụng, chúng ta có thể sử dụng hai hàm của lớp *Container*:

LayoutManager getLayout();

void setLayout(LayoutManager mgr);

Các thành phần giao diện sau khi đã được tạo ra thì phải được đưa vào một phần tử chứa nào đó. Hàm *add()* của lớp *Container* được nạp chồng để thực hiện nhiệm vụ đưa các thành phần vào phần tử chứa.

Component add(Component comp)

Component add(Component comp, int index)

Component add(Component comp, Object constraints)

Component add(Component comp, Object constraints, int index)

Trong đó, đối số *index* được sử dụng để chỉ ra vị trí của ô cần đặt thành phần giao diện *comp* vào. Đối số *constraints* xác định các hướng để đưa *comp* vào phần tử chứa.

<Tiếp theo>

Ngược lại, khi cần loại ra khỏi phần tử chứa một thành phần giao diện thì sử dụng các hàm sau:

void remove(int index)

void remove(Component comp)

void removeAll()

Lớp FlowLayout

Lớp *FlowLayout* có các hàm tạo lập để sắp hàng các thành phần giao diện:

FlowLayout()

FlowLayout(int aligment)

FlowLayout(int aligment, int horizongap, int verticalgap)

public static final int LEFT

public static final int CENTER

public static final int RIGHT

<Tiếp theo>

Lớp GridLayout

Lớp *GridLayout* cung cấp các hàm tạo lập để sắp hàng các thành phần giao diện:

GridLayout()

GridLayout(int rows, int columns)

GridLayout(int rows, int columns, int horizongap, int verticalgap)

Tạo ra một lưới hình chữ nhật có *rows* × *columns* ô có khoảng cách giữa các hàng các cột là *horizongap*, *verticalgap*. Một trong hai đối số *rows* hoặc *columns* có thể là 0, nhưng không thể cả hai, *GridLayout(1, 0)* là tạo ra lưới có một hàng.

<Tiếp theo>

Ví dụ: Mô tả cách sử dụng *GridLayout*

```
import java.awt.*;
import java.applet.*;

public class GridLayoutApplet extends Applet {
    public void init() {
        //Create a list of colors
        Label xLabel = new Label("X coordinate: ");
        Label yLabel = new Label("Y coordinate: ");
        TextField xInput = new TextField(5);
        TextField yInput = new TextField(5);
```

<Tiếp theo>

```
// Tạo ra lưới hình chữ nhật có 4 ô và đặt layout để sắp xếp các thành phần //
xLabel, xInput, yLabel, yInput
setLayout(new GridLayout(2,2));
add(xLabel); // Đặt xlabel vào ô thứ nhất
add(xInput); // Đặt xInput vào ô thứ hai
add(yLabel); // Đặt ylabel vào ô thứ ba
add(yInput); // Đặt yInput vào ô thứ tư
}
}
```

X coordinate:

Y coordinate:

Lớp BorderLayout

Lớp ***BorderLayout*** cho phép đặt một thành phần giao diện vào một trong bốn hướng: *bắc* (*NORTH*), *nam* (*SOUTH*), *đông* (*EAST*), *tây* (*WEST*) và *ở giữa* (*CENTER*).

BorderLayout()

BorderLayout(int horizongap, int verticalgap)

Tạo ra một *layout* mặc định hoặc có khoảng cách giữa các thành phần (tính bằng *pixel*) là *horizongap* theo hàng và *verticalgap* theo cột.

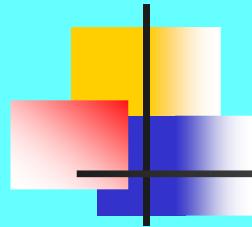
Trường hợp mặc định là *CENTER*, ngược lại, có thể chỉ định hướng để đặt các thành phần *comp* vào phần tử chứa theo *constraint* là một trong các hằng trên.

<Tiếp theo>

Ví dụ: Đặt một nút *Button* có tên “Vẽ” ở trên (NORTH), trường văn bản “Hiển thị thông báo” ở dưới (SOUTH) và hai thanh trượt (SCROLLBAR) ở hai bên cạnh (WEST và EAST).

```
import java.awt.*;
import java.applet.*;
public class BorderLayoutApplet extends Applet {
    public void init() {
        TextField msg = new TextField("Hiển thị thông báo");
        msg.setEditable(false);
        Button nutVe = new Button("Vẽ");
        Canvas vungVe = new Canvas();
        vungVe.setSize(150, 150); // Đặt kích thước cho vungVe vẽ tranh
        vungVe.setBackground(Color.white); // Đặt màu nền là trắng
```

<Tiếp theo>



```
Scrollbar sb1=new Scrollbar(Scrollbar.VERTICAL,0,10,-50,100);
Scrollbar sb2=new Scrollbar(Scrollbar.VERTICAL,0,10,-50,100);

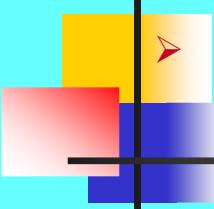
setLayout(new BorderLayout());

add(nutVe, BorderLayout.NORTH); // Đặt nutVe ở trên (NORTH)
add(msg, BorderLayout.SOUTH); // Đặt msg ở dưới (SOUTH)
add(vungVe, BorderLayout.CENTER); // Đặt vungVe ở giữa (CENTER)
add(sb1, BorderLayout.WEST); // Đặt sb1 ở bên trái (WEST)
add(sb2, BorderLayout.EAST); // Đặt sb2 ở bên phải (EAST)

}
```

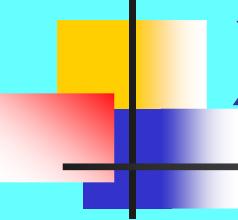


Xử lý sự kiện



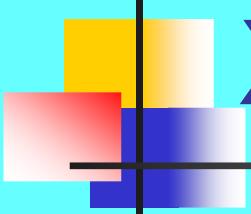
Các ứng dụng với GUI thường được hướng dẫn bởi các sự kiện (*event*). Việc nhấn một nút, mở, đóng các Window hay gõ các ký tự từ bàn phím, v.v. đều tạo ra các sự kiện (*event*) và được gửi tới cho chương trình ứng dụng.

- Các sự kiện (Events) được xử lý bằng các công cụ sau:
 - Abstract Windowing Toolkit
 - Trình duyệt.
 - Các trình xử lý sự kiện do người dùng tạo riêng.
- Các ứng dụng cần đăng ký trình xử lý sự kiện với đối tượng
- Các trình xử lý này được gọi khi có một sự kiện tương ứng xảy ra



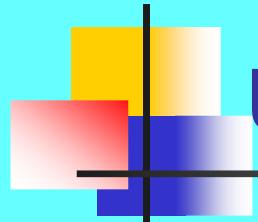
Xử lý các sự kiện

- Event Listener sẽ lắng nghe một sự kiện cụ thể mà một đối tượng tạo ra
- Mỗi event listener cung cấp các phương thức để xử lý các sự kiện này
- Lớp có cài đặt listener cần định nghĩa những phương thức này



Xử lý các sự kiện

- Các bước cần tuân thủ để sử dụng mô hình Event Listener:
 ■ Cài đặt Listener tương ứng
 ■ Nhận diện được tất cả các *thành phần tạo sự kiện*
 ■ Nhận diện được tất cả các *sự kiện được xử lý*
 ■ Cài đặt các phương thức của listener, và viết các đoạn mã để xử lý sự kiện trong các phương thức đó
- Interface định nghĩa các phương thức khác nhau để xử lý mỗi sự kiện



Các sự kiện và Listener tương ứng

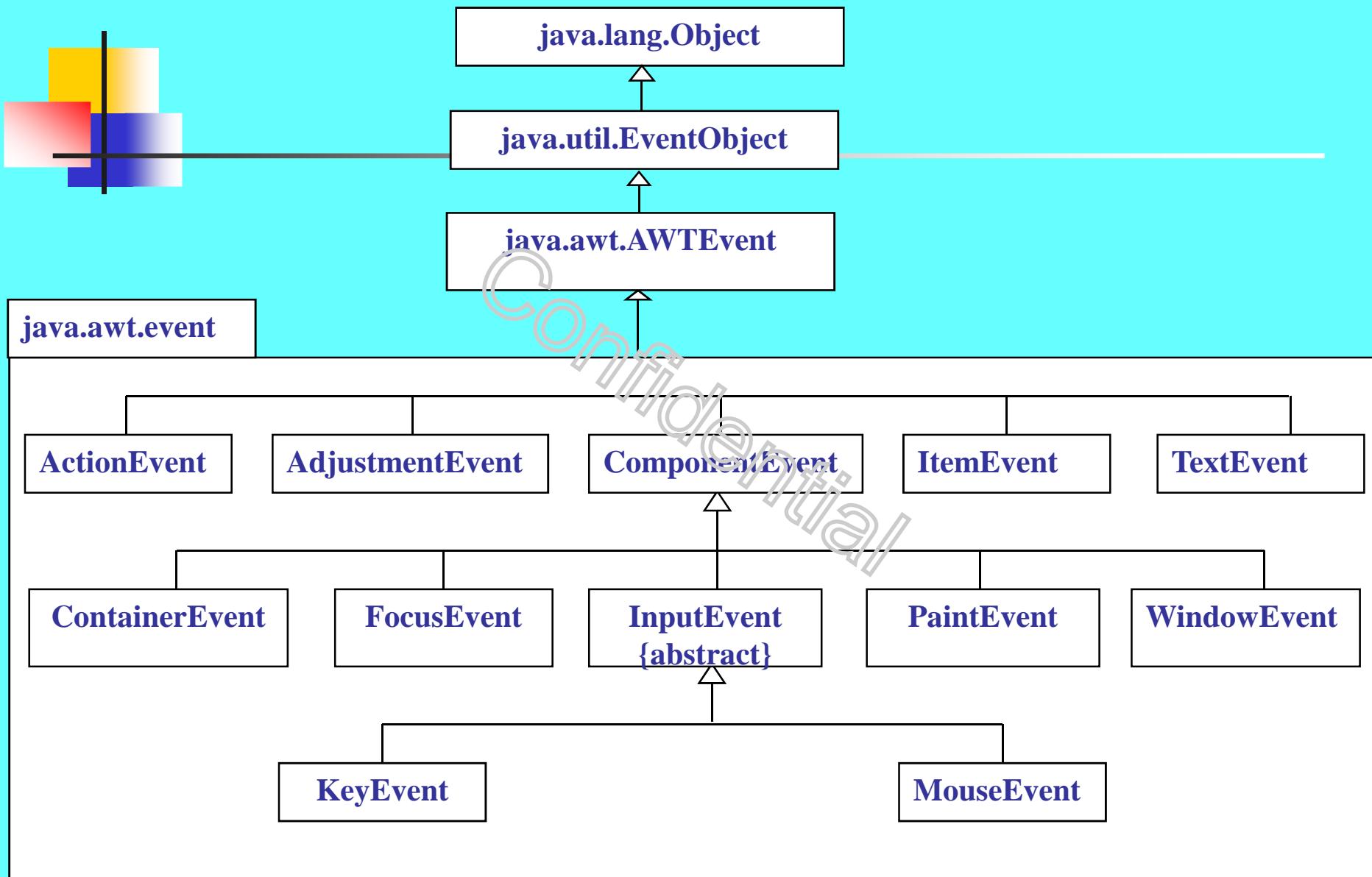
- ActionEvent
 - AdjustmentEvent
 - ComponentEvent
 - FocusEvent
 - ItemEvent
 - WindowEvent
 - TextEvent
 - MouseEvent
 - KeyEvent
- ActionListener
 - AdjustmentListener
 - ComponentListener
 - FocusListener
 - ItemListener
 - WindowListener
 - TextListener
 - MouseListener
 - MouseMotionListener
 - KeyListener

Xử lý các sự kiện

Trong Java các sự kiện được thể hiện bằng các đối tượng. Lớp cơ sở nhất, lớp cha của tất cả các lớp con của các sự kiện là lớp *java.util.EventObject*.

- Các lớp con của *AWTEvent* được chia thành hai nhóm:
1. Các lớp mô tả về ngữ nghĩa của các sự kiện
 2. Các lớp sự kiện ở mức thấp

Các lớp xử lý sự kiện



1. Các lớp ngữ nghĩa

a/ ActionEvent

Sự kiện này được phát sinh bởi những hoạt động thực hiện trên các thành phần của GUI. Các thành phần gây ra các sự kiện hành động bao gồm:

Button - khi một nút button được kích hoạt

List - khi một mục trong danh sách được kích hoạt đúp

MenuItem - khi một mục trong thực đơn được chọn

TextField - khi gõ phím ENTER trong trường văn bản (**text**).

Lớp ActionEvent có các hàm:

String getActionCommand()

Cho biết tên của lệnh tương ứng với sự kiện xảy ra, là tên của nút, mục hay text.

b/ AdjustmentEvent

Thành phần gây ra sự kiện căn chỉnh (adjustment):

Scrollbar - khi thực hiện một lần căn chỉnh trong thanh trượt

Scrollbar.

Lớp này có hàm:

int getValue()

Cho lại giá trị hiện thời được xác định bởi lần căn chỉnh sau cùng.

c/ ItemEvent

Các thành phần của GUI gây ra các sự kiện về các mục gồm có:
Checkbox - khi trạng thái của hộp kiểm tra *Checkbox* thay đổi

CheckboxMenuItem - khi trạng thái của hộp kiểm tra *Checkbox* ứng với mục của thực đơn thay đổi,

Choice - khi một mục trong danh sách được chọn hoặc bị loại bỏ

List - khi một mục trong danh sách được chọn hoặc bị loại bỏ.

Lớp *ItemEvent* có hàm:

Object getItem()

Cho lại đối tượng được chọn hay vừa bị loại bỏ.

d/ TextEvent

Các thành phần của GUI gây ra các sự kiện về *text* gồm có:

TextArea - khi kết thúc bằng nhấn nút ENTER,

TextField - khi kết thúc bằng nhấn nút ENTER.

<Tiếp theo>

Kiểu sự kiện	Nguồn gây ra sự kiện	Các hàm đón nhận và điều khiển các sự kiện	Giao diện Listener tương ứng
ActionEvent	Button List TextField Scrollbar	addComponentListener remove ActionListener	ActionListener
AdjustmentEvent		addAdjustmentListener removeAdjustmentListener	AdjustmentListener
ItemEvent	Choice Checkbox CheckboxMen uItem List	addItemListener removeItemListener	ItemListener
TextEvent	List TextArea TextField	addTextListener removeTextListener	TextListener

2. Các sự kiện ở mức thấp

a/ **ComponentEvent**

Sự kiện này xuất hiện khi một thành phần bị che giấu, hiển thị hay thay đổi lại kích thước. Lớp *ComponentEvent* có hàm:

Component getComponent()

Cho lại đối tượng tham chiếu kiểu *Component*.

b/ **ContainerEvent**

Sự kiện này xuất hiện khi một thành phần được bổ sung hay bị loại bỏ khỏi phần tử chứa (*Container*).

c/ **FocusEvent**

Sự kiện này xuất hiện khi một thành phần nhận được một nút từ bàn phím.

d/ **KeyEvent**

Lớp *KeyEvent* là lớp con của lớp trừu tượng *InputEvent* được sử dụng để xử lý các sự kiện liên quan đến các phím của bàn phím.

int getKeyCode()

Đối với các sự kiện KEY_PRESSED hoặc KEY_RELEASED, hàm này được sử dụng để nhận lại giá trị nguyên tương ứng với mã của phím trên bàn phím.

char getKeyChar()

Đối với các sự kiện KEY_PRESSED, hàm này được sử dụng để nhận lại giá trị nguyên mã Unicode tương ứng với ký tự của bàn

<Tiếp theo>

e/ MouseEvent

Lớp **MouseEvent** là lớp con của lớp trừu tượng **InputEvent** được sử dụng để xử lý các tín hiệu của chuột. Lớp này có các hàm:

int getX()

int getY()

Point getPoint()

Các hàm này được sử dụng để nhận lại tọa độ *x, y* của vị trí liên quan đến sự kiện do chuột gây ra.

void translatePoint(int dx, int dy)

Hàm **translate()** được sử dụng để chuyển tọa độ của sự kiện do chuột gây ra đến (*dx, dy*).

int getClickCount()

Hàm **getClickCount()** đếm số lần kích chuột.

f/ PaintEvent

Sự kiện này xuất hiện khi một thành phần gọi hàm **paint()**/**update()** để vẽ.

g/ WindowEvent

Sự kiện loại này xuất hiện khi thao tác với các Window. Lớp này có hàm:

Window getWindow()

Hàm này cho lại đối tượng của lớp Window ứng với sự kiện liên quan đến Window đã xảy ra.

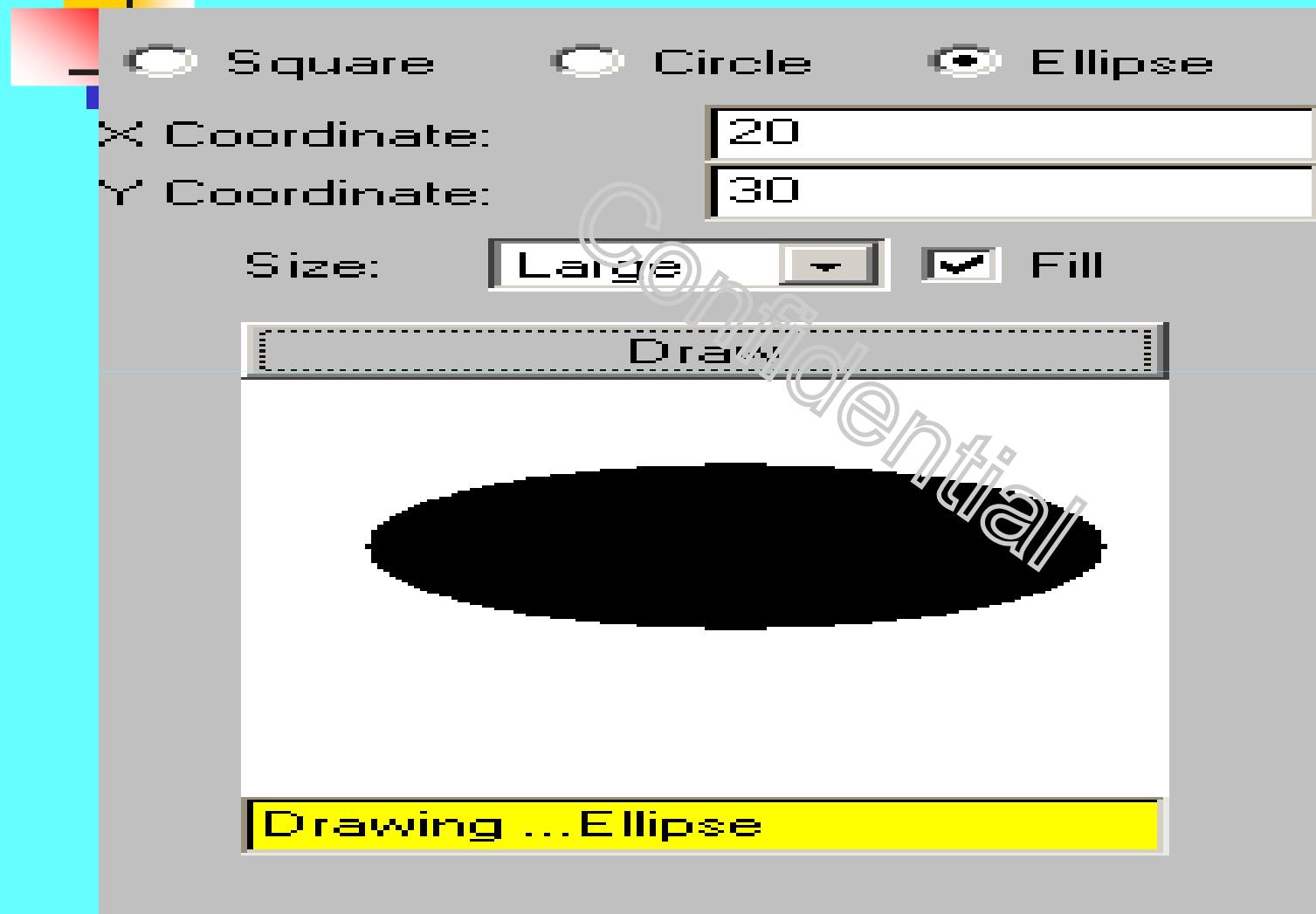
<Tiếp theo>

Các hàm xử lý các sự kiện ở mức thấp

Kiểu sự kiện	Nguồn gây ra sự kiện	Các hàm đón nhận và di dời các sự kiện	Giao diện Listener tương ứng
ComponentEvent	Component	addComponentListener removeComponentListener	ComponentListener
ContainerEvent	Container	addContainerListener removeContainerListener	ContainerListener
FocusEvent	Component	addFocusListener removeFocusListener	FocusListener
KeyEvent	Component	addKeyListener removeKeyListener	KeyListener
MouseEvent	Component	addMouseListener removeMouseListener addMouseMotionListener removeMouseMotionListener	MouseMotionListener
WindowEvent	Window	addWindowListener removeWindowListener	WindowListener

<Tiếp theo>

Ví dụ: Xây dựng ứng dụng có giao diện đồ họa GUI



<Tiếp theo>

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

interface IGeometryConstants { // Định nghĩa Interface gồm các hằng số trước
    int SQUARE = 0;
    int CIRCLE = 1;
    int ELLIPSE = 2;
    String[] shapeNames = {"Square", "Circle", "Ellipse"};
    int SMALL = 0;
    int MEDIUM = 1;
    int LARGE = 2;
    String[] sizeNames = {"Small", "Medium", "Large"};
}

public class DemoApplet extends Applet implements
    IGeometryConstants {
    // Khai báo một Panel để chứa các Checkbox tương ứng với các loại hình shape
    Panel shapePanel; // Khai báo một Panel
    CheckboxGroup shapeCBG; // Khai báo một CheckboxGroup
    Checkbox squareCB; // Khai báo một Checkbox ứng với hình vuông
    Checkbox circleCB; // Khai báo một Checkbox ứng với hình tròn
    Checkbox ellipseCB; // Khai báo một Checkbox ứng với hình ellipse
    // Khai báo một Panel để chứa các Label và các TextField
    Panel xyPanel; // Khai báo một Panel xyPanel
    Label xLabel; // Khai báo một Label xLabel
    TextField xInput; // Khai báo một TextField xInput
    Label yLabel; // Khai báo một Label yLabel
    TextField yInput; // Khai báo một TextField yInput
```

<Tiếp theo>

```
// Khai báo một Panel để chứa các Label , Choice và Checkbox
Panel          sizePanel;
Label          sizeLabel;
Choice         sizeChoices;
Checkbox       fillCB;

// Khai báo một Panel để chứa shape, coordinates, size and fill Panel leftPanel
Panel          leftPanel;
// Khai báo một Panel để chứa Massage display, draw button and canvas
Panel          rightPanel;
Button         drawButton;
DrawRegion    drawRegion;
TextField      messageDisplay;
// Định nghĩa lại hàm init()
public void init() {
    makeShapePanel();
    makeXYPanel();
    makeSizePanel();
    makeLeftPanel();
    makeRightPanel();
    // Gọi hàm xử lý các sự kiện
    addListeners();
    // Đưa các Panel vào hệ thống
    add(leftPanel);
    add(rightPanel);
}
```

<Tiếp theo>

```
// Đưa các thành phần về các shape vào shapePanel
void makeShapePanel() {
    shapePanel = new Panel();
    shapeCBG = new CheckboxGroup();
    squareCB = new Checkbox(shapeNames[SQUARE], shapeCBG, true);
    circleCB = new Checkbox(shapeNames[CIRCLE], shapeCBG, false);
    ellipseCB = new Checkbox(shapeNames[ELLIPSE], shapeCBG, false);
    shapePanel.setLayout(new FlowLayout());
    shapePanel.add(squareCB);
    shapePanel.add(circleCB);
    shapePanel.add(ellipseCB);
}
// Đưa các thành phần về các x,y coordinates vào xyPanel
void makeXYPanel() {
    xyPanel = new Panel();
    xLabel = new Label("X Coordinate:");
    yLabel = new Label("Y Coordinate:");
    xInput = new TextField(5);
    yInput = new TextField(5);
    xyPanel.setLayout(new GridLayout(2,2));
    xyPanel.add(xLabel);
    xyPanel.add(xInput);
    xyPanel.add(yLabel);
    xyPanel.add(yInput);
}
```

<Tiếp theo>

```
// Đưa các thành phần về size và fill vào sizePanel
void makeSizePanel() {
    sizePanel = new Panel();
    sizeLabel = new Label("Size:");
    sizeChoices = new Choice();
    sizeChoices.add(sizeNames[0]);
    sizeChoices.add(sizeNames[1]);
    sizeChoices.add(sizeNames[2]);
    fillCB = new Checkbox("Fill", false);
    sizePanel.setLayout(new FlowLayout());
    sizePanel.add(sizeLabel);
    sizePanel.add(sizeChoices);
    sizePanel.add(fillCB);
}
// Đưa các thành phần vào leftPanel
void makeLeftPanel(){
    leftPanel = new Panel();
    leftPanel.setLayout(new BorderLayout());
    leftPanel.add(shapePanel, "North");
    leftPanel.add(xyPanel, "Center");
    leftPanel.add(sizePanel, "South");
}
```

<Tiếp theo>

```
// Đưa các thành phần vào rightPanel
void makeRightPanel(){
    rightPanel = new Panel();
    messageDisplay = new TextField("MESSAGE DISPLAY");
    messageDisplay.setEditable(false);
    messageDisplay.setBackground(Color.yellow);

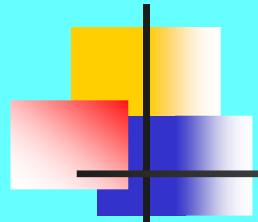
    drawButton = new Button("Draw");
    drawButton.setBackground(Color.lightGray);

    drawRegion = new DrawRegion();
    drawRegion.setSize(150, 150);
    drawRegion.setBackground(Color.white);
    rightPanel.setLayout(new BorderLayout());
    rightPanel.add(drawButton, BorderLayout.NORTH);
    rightPanel.add(messageDisplay, BorderLayout.SOUTH);
    rightPanel.add(drawRegion, BorderLayout.CENTER);
}
```

<Tiếp theo>

```
// Xử lý các sự kiện tương ứng với các việc thực hiện tròn các thành phần đồ họa
void addListeners(){
    drawButton.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent evt){
            int shape, xCoord, yCoord, width;
            messageDisplay.setText("");
            if(squareCB.getState())
                shape = SQUARE;
            else if (circleCB.getState())
                shape = CIRCLE;
            else if (ellipseCB.getState())
                shape = ELLIPSE;
            else {
                messageDisplay.setText("Unknown shape");
                return;
            }
            try{
                xCoord = Integer.parseInt(xInput.getText());
                yCoord = Integer.parseInt(yInput.getText());
            }catch(NumberFormatException e){
                messageDisplay.setText("Illegal coordinate");
                return;
            }
        }
    });
}
```

<Tiếp theo>



```
switch(sizeChoices.getSelectedIndex()){
    case SMALL: width = 30; break;
    case MEDIUM: width = 60; break;
    case LARGE: width = 120; break;
    default:
        messageDisplay.setText("Unknown size");
        return;
}
messageDisplay.setText("Drawing..."+shapeNames[shape]);
drawRegion.doDraw(shape,xCoord,yCoord,
    fillCB.getState(),width);
});
xInput.addTextListener(new TextListener(){
    public void textValueChanged(TextEvent evt){
        checkTF(xInput);
    }
});
yInput.addTextListener(new TextListener(){
    public void textValueChanged(TextEvent evt){
        checkTF(yInput);
    }
});
}// Kết thúc hàm addListener()
```

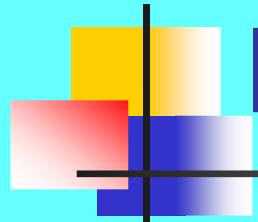
A large, semi-transparent watermark reading "Confidential" diagonally across the slide.

<Tiếp theo>

```
void checkTF(TextField tf){
    messageDisplay.setText("");
    try{
        Integer.parseInt(tf.getText());
    }catch(NumberFormatException e){
        messageDisplay.setText("Illegal coordinate ");
    }
}
// Định nghĩa lớp DrawRegion mở rộng lớp Canvas và sử dụng các hàng ở giao diện
class DrawRegion extends Canvas implements IGeometryConstants{
    private int shape,xCoord, yCoord;
    private boolean fillFlag;
    private int width;
    public DrawRegion(){
        setSize(150, 150);
        setBackground(Color.white);
    }
}
// Định nghĩa hàm doDraw() để vẽ
public void doDraw(int h, int x, int y, boolean f, int w){
    this.shape = h;
    this.xCoord = x;
    this.yCoord = y;
    this.fillFlag = f;
    this.width = w;
    repaint();
}
```

<Tiếp theo>

```
public void paint(Graphics g){  
    switch(shape){  
        case SQUARE:  
            if(fillFlag) g.fillRect(xCoord, yCoord, width,width);  
            else g.drawRect(xCoord, yCoord, width,width);  
            break;  
        case CIRCLE:  
            if(fillFlag) g.fillOval(xCoord, yCoord, width,width);  
            else g.drawOval(xCoord, yCoord, width,width);  
            break;  
        case ELLIPSE:  
            if(fillFlag) g.fillOval(xCoord,yCoord,width,width/2);  
            else g.drawOval(xCoord, yCoord, width,width/2);  
            break;  
    }  
}
```



Menus

- Các loại menu :
 - Pull-down
 - Pop-up menu
- Chỉ có thể đặt các thanh menubar vào trong các Frame mà thôi
- Các thành phần của menu:
 - Menubar
 - Menultems

Thành phần Menu

Lớp *abstract class MenuComponent* là lớp cơ sở cho tất cả các lớp thực hiện những vấn đề liên quan đến thực đơn (*menu*).

Lớp *MenuBar* cài đặt thanh thực đơn và trong đó có thể chứa các thực đơn *pull-down*.

Lớp *MenuItem* định nghĩa từng mục của thực đơn.

Lớp *Menu* cài đặt các thực đơn *pull-down* để có thể đưa vào một thực đơn bất kỳ.

Lớp *PopupMenu* biểu diễn cho thực đơn pop-up.

Lớp *CheckboxMenuItem* chứa các mục được chọn để kiểm tra trong các mục.

Việc tạo lập một thanh thực đơn cho một *frame* được thực hiện như sau:

1. *Tạo ra một thanh thực đơn,*

```
MenuBar thanhThDon = newMenuBar();
```

2. *Tạo ra một thực đơn,*

```
Menu thucDon = new Menu("Cac loai banh");
```

3. *Tạo ra các mục trong thực đơn và đưa vào thực đơn,*

```
MenuItem muc = new MenuItem("Banh day");
```

```
thucDon.add(muc); // Đưa muc vào thucDon
```

4. *Đưa các thực đơn vào thanh thực đơn,*

```
thanhThDon.add(thucDon); // Đưa thucDon vào thanhThDon
```

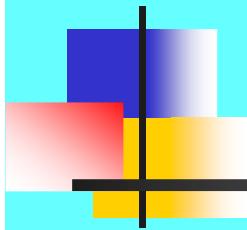
5. *Tạo ra một frame và đưa thanh thực đơn vào frame đó.*

Thành phần *Menu*

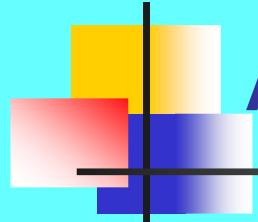
Ví dụ: Tạo lập và sử dụng các thực đơn.

```
import java.awt.*;
import java.applet.*;
public class MenuDemo extends Applet{
    public static void main(String args[]){
        MenuBar thanhThDon = new MenuBar();
        Menu thucDon = new Menu("Cac loai banh");
        MenuItem b1 = new MenuItem("Banh day");
        thucDon.add(b1);
        MenuItem b3 = new MenuItem("Banh khoai");
        thucDon.add(b3);
        thucDon.addSeparator();// Tạo ra một thanh phôn cách
        MenuItem b4 = new MenuItem("Banh gio");
        thucDon.add(b4);
        thucDon.add(new CheckboxMenuItem("Banh ran"));
        thanhThDon.add(thucDon);
        Frame frame = new Frame("Cac mon an");
        frame.setMenuBar(thanhThDon);
        frame.pack(); frame.setVisible(true);}
}
```

CHƯƠNG 4 : NGÔN NGỮ LẬP TRÌNH JAVA

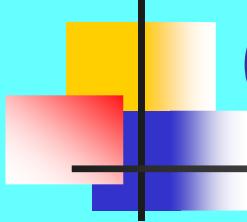


4.6 Applets



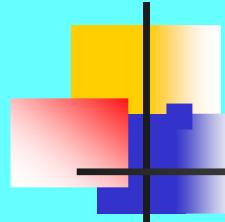
Applets

- Là một chương trình Java mà chạy với sự hỗ trợ của trình duyệt web hoặc appletviewer
- Tất cả các applets là lớp con của lớp ‘Applet’
- Để tạo một applet, bạn cần import gói sau:
 - **java.applet**



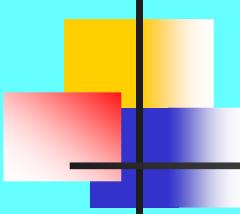
Cấu trúc applet

- Định nghĩa một applet từ bốn sự kiện xảy ra trong quá trình thực thi
- Đối với mỗi sự kiện được định nghĩa bởi một phương thức tương ứng.
- Các phương thức:
 - **init()**
 - **start()**
 - **stop()**
 - **destroy()**



Các phương thức khác:

- **paint()**
- **repaint()**
- **showStatus()**
- **getAppletInfo()**
- Các phương thức **init()**, **start()**, **stop()**, **destroy()**, and **paint()** được thừa kế từ applet.
- Mỗi phương thức này mặc định là rỗng. Vì thế các phương thức này phải được nạp chồng.

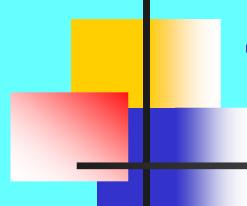


Biên dịch và thực thi applet

- Một applet thì được biên dịch theo cú pháp sau
javac Applet1.java
- Để thực thi một applet, tạo một tập tin HTML có sử dụng thẻ applet
 - Thẻ applet có hai thuộc tính:
 - Width
 - Height
 - Để truyền tham số tới applet, sử dụng thẻ ‘param’, và tiếp theo là thẻ ‘value’
- Applet có thể được thực thi bằng applet viewer

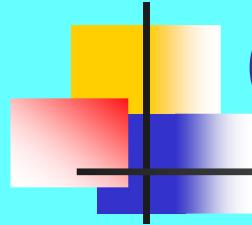
Điểm khác biệt giữa applet và một ứng dụng

- Các ứng dụng khi thực thi phải sử dụng trình biên dịch Java, trong khi các applets thực thi được trên bất kỳ trình duyệt nào mà hỗ trợ Java, hoặc sử dụng ‘AppletViewer’ trong JDK.
- Một ứng dụng bắt đầu với phương thức ‘main()’. Còn đối với applet thì không sử dụng phương thức này
- Một ứng dụng sử dụng ‘System.out.println()’ để hiển thị, trong khi một applet thì sử dụng các phương thức của lớp Graphics.

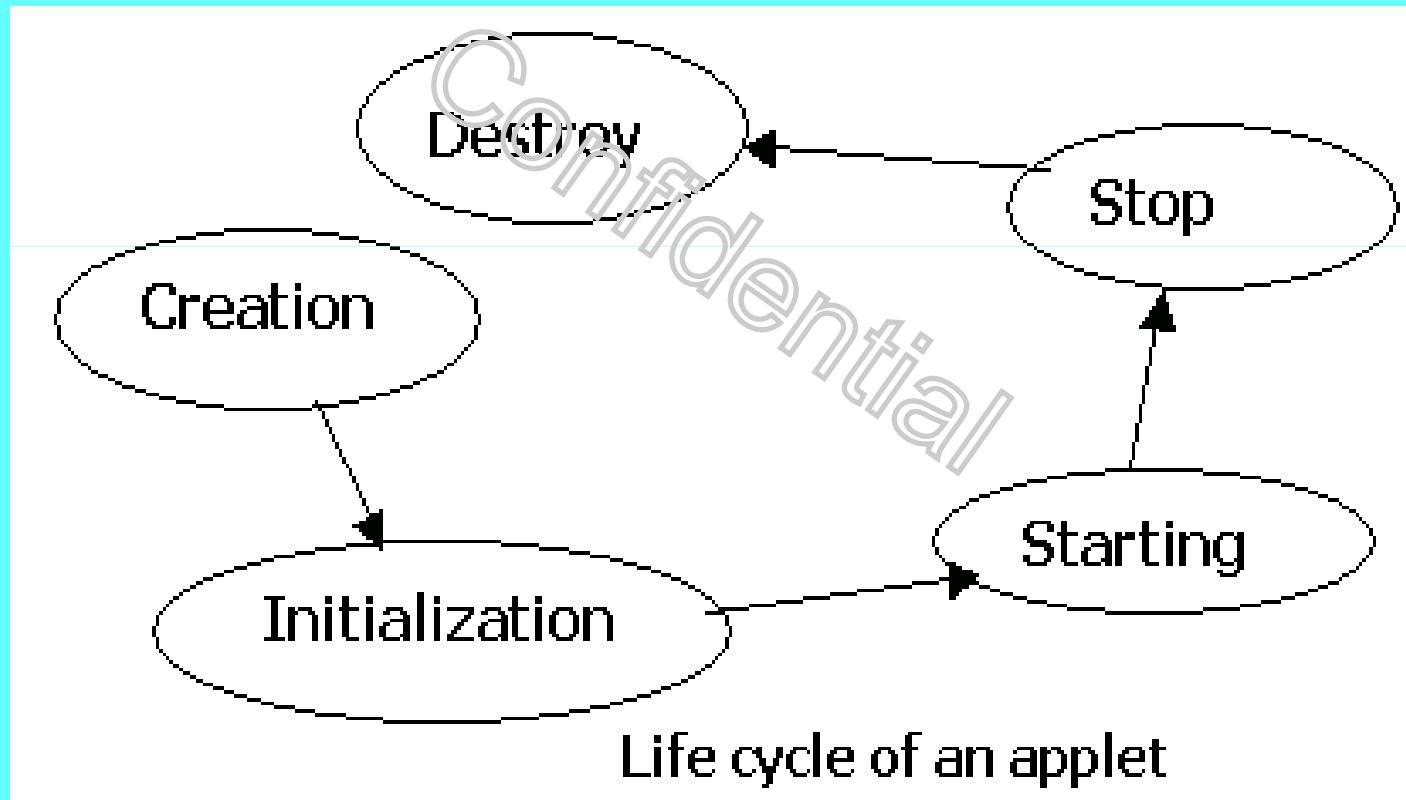


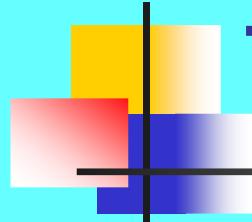
Những hạn chế về bảo mật trong applet

- Không thể đọc hoặc viết các tập tin trên hệ thống tập tin của người sử dụng
- Không thể giao tiếp với một site trên internet. Mà chỉ giao tiếp với một dịch vụ trên trang web có applet.
- Không thể chạy bất kỳ chương trình nào trên hệ thống của người đọc
- Không thể load bất kỳ chương trình nào được lưu trên hệ thống của người sử dụng



Chu trình sống của applet

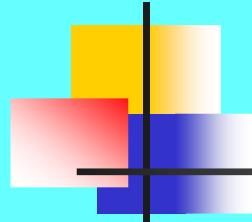




Truyền tham số tới một applet

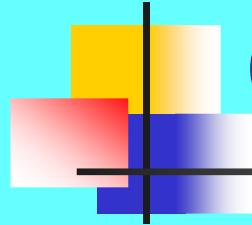
- Để truyền tham số, sử dụng PARAM trong thẻ ~~HTML~~
- Ví dụ

```
<applet code = "Mybutton1" width = "100" height = "100">
<PARAM NAME = "Mybutton" value = "Display Dialog">
</applet>
```



Lớp đồ họa

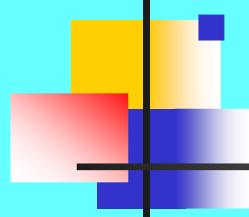
- Được cung cấp bởi gói AWT
- Cung cấp ~~một~~ một tập hợp các phương thức để vẽ như sau:
 - Oval
 - Rectangle
 - Square
 - Circle
 - Lines
 - Text in different fonts



Graphical Background

- Các phương thức để vẽ nền :
 - **getGraphics()**
 - **repaint()**
 - **update(Graphics g)**
 - **paint(Graphics g)**

Hiển thị chuỗi, ký tự và bytes



- Phương thức để vẽ hoặc hiển thị một chuỗi trên frame

Cú pháp

- **drawString(String str, int xCoor, int yCoor);**
- Phương thức để vẽ hoặc hiển thị các ký tự trên frame

Cú pháp

- **drawChars(char array[], int offset, int length, int xCoor, int yCoor);**
- Phương thức để vẽ hoặc hiển thị bytes trên frame

Cú pháp

- **drawBytes(byte array[], int offset, int length, int xCoor, int yCoor);**

Vẽ các hình thê

Phương thức được sử dụng để vẽ đường thẳng như sau

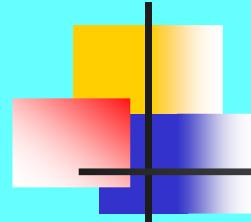
Cú pháp

- **drawLine(int x1, int y1, int x2, int y2);**
- Các phương thức được sử dụng để vẽ đường tròn như sau

Cú pháp

- **drawOval(int xCoor, int yCoor, int width, int height);**
- **setColor(Color c);**
- **fillOval(int xCoor, int yCoor, int width, int height);**

- Phương thức sử dụng để vẽ hình vuông:

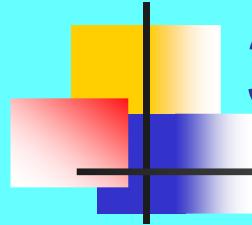


Cú pháp

- `drawRect(int xCoor, int yCoor, int width, int height);`
- `fillRect(int xCoor, int yCoor, int width, int height);`
- Các phương thức được sử dụng để vẽ hình vuông có góc tròn

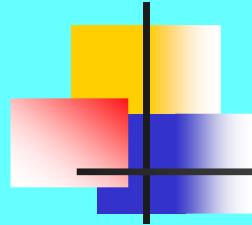
Cú pháp

- `drawRoundRect(int xCoor, int yCoor, int width, int height, int arcWidth, int arcHeight);`
- `fillRoundRect (int xCoor, int yCoor, int width, int height, int arcWidth, int arcHeight);`



3D Rectangles & Arcs

- Các phương thức được sử dụng để vẽ hình 3D Cú pháp
 - **draw3DRect(int xCoord, int yCoord, int width, int height, boolean raised);**
 - **drawArc(int xCoord, int yCoord, int width, int height, int arcwidth, int archeight);**
 - **fillArc(int xCoord, int yCoord, int width, int height, int arcwidth, int archeight);**

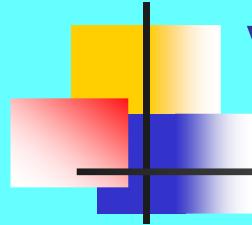


Drawing PolyLines

- Các phương thức được sử dụng để vẽ nhiều đoạn thẳng

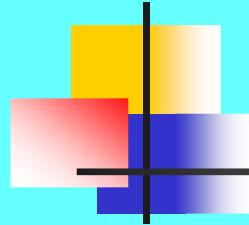
Cú pháp

- **drawPolyline(int xArray[], int yArray[], int totalPoints);**
- **g.setFont(new Font("Times Roman", Font.BOLD,15));**



Vẽ và tô các hình đa giác

- Các phương thức để vẽ và tô các hình đa giác
- Cú pháp**
- **drawPolygon(int x[], int y[], int numPoints);**
 - **fillPolygon(int x[], int y[], int numPoints);**



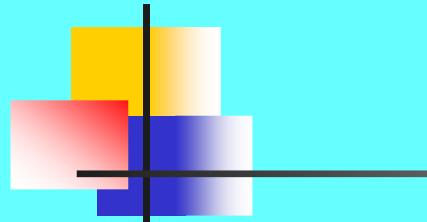
Màu

- Java sử dụng màu RGB
- Bảng các giá trị màu

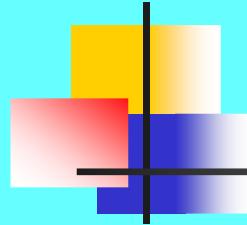
Element	Range
Red	0-255
Green	0-255
Blue	0-255

- Cú pháp của hàm dựng để tạo một màu
`color(int red, int green, int blue);`

- Bảng trình bày các giá trị màu RGB thông thường

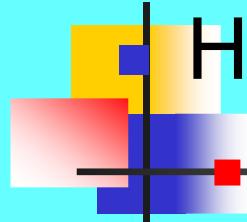


Color	Red	Green	Blue
White	255	255	255
Light Gray	192	192	192
Gray	128	128	128
Dark Gray	64	64	64
Black	0	0	0
Pink	255	175	175
Orange	255	200	0
Yellow	255	255	0
Magenta	255	0	255



Font

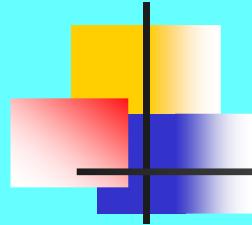
- Gói java.awt package cung cấp bởi lớp ‘Font’
- Các phương thức của lớp Font:
 - **getAllFont()**
 - **getLocalGraphicsEnvironment()**
 - **getFont()**
 - **getFontList()**



Hàm dựng Font nhận 3 tham số

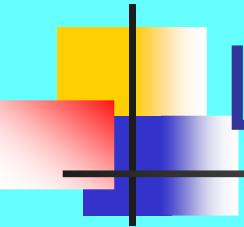
- Tên font trong chuỗi định dạng; tên này có trong phương thức getFontList().
- Kiểu của font.
- Kích thước của font.
- Ví dụ

**Font f1 = new Font("SansSerif", Font.ITALIC, 16);
g.setFont(f1);**



Lớp FontMetrics

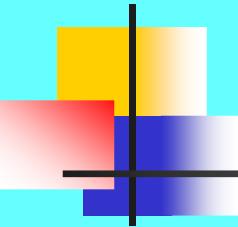
- Đo các ký tự khác nhau hiển thị trong các font khác nhau.
- Việc đo bao gồm ‘height’, ‘baseline’, ‘ascent’, ‘descent’ và ‘leading’ của font.
- Nó không cụ thể vì nó là một lớp trừu tượng



Lớp FontMetrics (tiếp theo...)

- Phương thức:

- **getFontMetrics(f1)**
- **getHeight()**
- **getAscent()**
- **getDescent()**
- **getLeading()**
- **getName()**

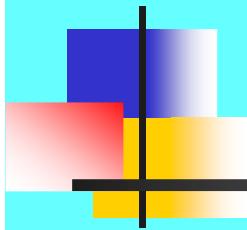


Kiểu vẽ

- Có 2 kiểu vẽ.
- XOR
 - setXORMode(Color c)**
- Paint
 - setPaintMode()**

Confidential

CHƯƠNG 4 : NGÔN NGỮ LẬP TRÌNH JAVA



Confidential

4.7 Lập trình đa tuyến

Lập trình đa tuyến

Đa nhiệm, tiến trình và luồng

- Xử lý đa luồng trong Java
- Mức ưu tiên của luồng
- Vấn đề đồng bộ hóa và bài toán tắc nghẽn

Tiến trình, đa nhiệm và đa luồng

- **Tiến trình** : là một chương trình chạy trên hệ điều hành và được quản lý thông qua các thẻ
- **Tiểu trình** : là một đơn vị xử lý cơ bản của hệ thống. Một tiến trình sở hữu nhiều tiểu trình
- **Đơn nhiệm** : tại một thời điểm chỉ có một tiến trình
- **Đa nhiệm**: ở cùng một thời điểm có nhiều hơn một tiến trình thực hiện đồng thời trên cùng một máy tính. Có hai kỹ thuật đa nhiệm:
 - + Đa nhiệm dựa trên các *tiến trình*
 - + Đa nhiệm dựa trên các *luồng*

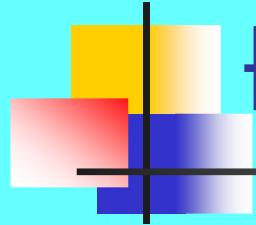
Tiến trình, đa nhiệm và đa luồng

- Một tiến trình có thể bao gồm nhiều luồng.
- *Các luồng của một tiến trình có thể chia sẻ với nhau về không gian địa chỉ chương trình, các đoạn dữ liệu và môi trường xử lý, đồng thời cũng có vùng dữ liệu riêng để thao tác.*
- Trong môi trường đơn luồng, ở mỗi thời điểm chỉ cho phép một tác vụ thực thi.
- Kỹ thuật đa nhiệm cho phép tận dụng được những thời gian rỗi của CPU để thực hiện những tác vụ khác.

<Tiếp theo>

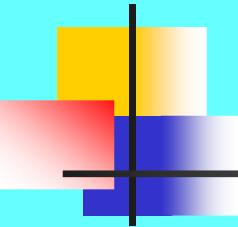
- **Đa nhiệm có thể thực hiện được theo hai cách:**

- + Phụ thuộc vào hệ điều hành, nó có thể cho tạm ngừng chương trình mà không cần tham khảo các chương trình đó.
- + Các chương trình chỉ bị dừng lại khi chúng tự nguyện nhường điều khiển cho chương trình khác.
- Nhiều hệ điều hành hiện nay đã hỗ trợ đa luồng, Java hỗ trợ đa nhiệm dựa trên các luồng và cung cấp các đặc tính ở mức cao cho lập trình đa luồng.



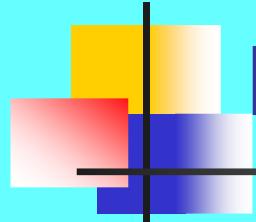
Đa luồng (đa tuyến)

- Là khả năng làm việc với nhiều luồng
- Đa luồng chuyên sử dụng cho việc thực thi nhiều công việc đồng thời
- Đa luồng giảm thời gian rỗi của hệ thống đến mức thấp nhất.



Tạo và quản lý luồng

- Khi chương trình Java thực thi hàm main() tức là luồng main được thực thi. Tuyến này được tạo ra một cách tự động, tại đây :
 - Các luồng con sẽ được tạo ra từ đó
 - Nó là luồng cuối cùng kết thúc việc thực thi. Ngay khi luồng main() ngừng thực thi, chương trình bị chấm dứt



Phân chia thời gian giữa các luồng

- CPU thực thi chỉ một luồng tại một thời điểm nhất định.
- Các luồng có độ ưu tiên bằng nhau thì được phân chia thời gian sử dụng bộ vi xử lý.

Lập trình đa luồng Java

- **Nói Java ta có thể xây dựng các chương trình đa luồng**
- **Một ứng dụng có thể bao gồm nhiều luồng, mỗi luồng được gắn công việc cụ thể và được thực thi đồng thời với các luồng khác**
- **Java cung cấp hai giải pháp tạo lập luồng:**
 - 1. Thiết lập lớp con của Thread**
 - 2. Cài đặt lớp xử lý luồng từ giao diện Runnable.**

Lập trình đa luồng Java

- **Cách thứ nhất:** Tạo ra một lớp kế thừa từ lớp Thread và ghi đè phương thức run của lớp Thread như sau:

```
class MyClass extends Thread
```

```
{  
    // Một số thuộc tính  
    public void run()  
    {  
        // Các lệnh cần thực hiện theo luồng  
    }  
    // Một số hàm khác được viết đè hay được bổ  
    // sung  
}
```

Khi chương trình chạy nó sẽ gọi một hàm đặc biệt đã được khai báo trong Thread đó là start() để bắt đầu một luồng đã được tạo ra.

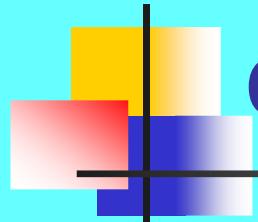
Lập trình đa luồng Java

Cách thứ hai:

+ Java giải quyết hạn chế trên bằng cách xây dựng lớp để tạo ra các luồng thực hiện trên cơ sở cài đặt giao diện hỗ trợ luồng.

+ Tạo ra một lớp triển khai từ giao diện Runnable, cài đặt phương thức run

```
class MyClass implements Runnable
{
    // Các thuộc tính
    // Nạp chồng hay viết đè một số hàm
    public void run()
    {
        . . .
    }
}
```



Trạng thái và các phương thức của lớp Thread

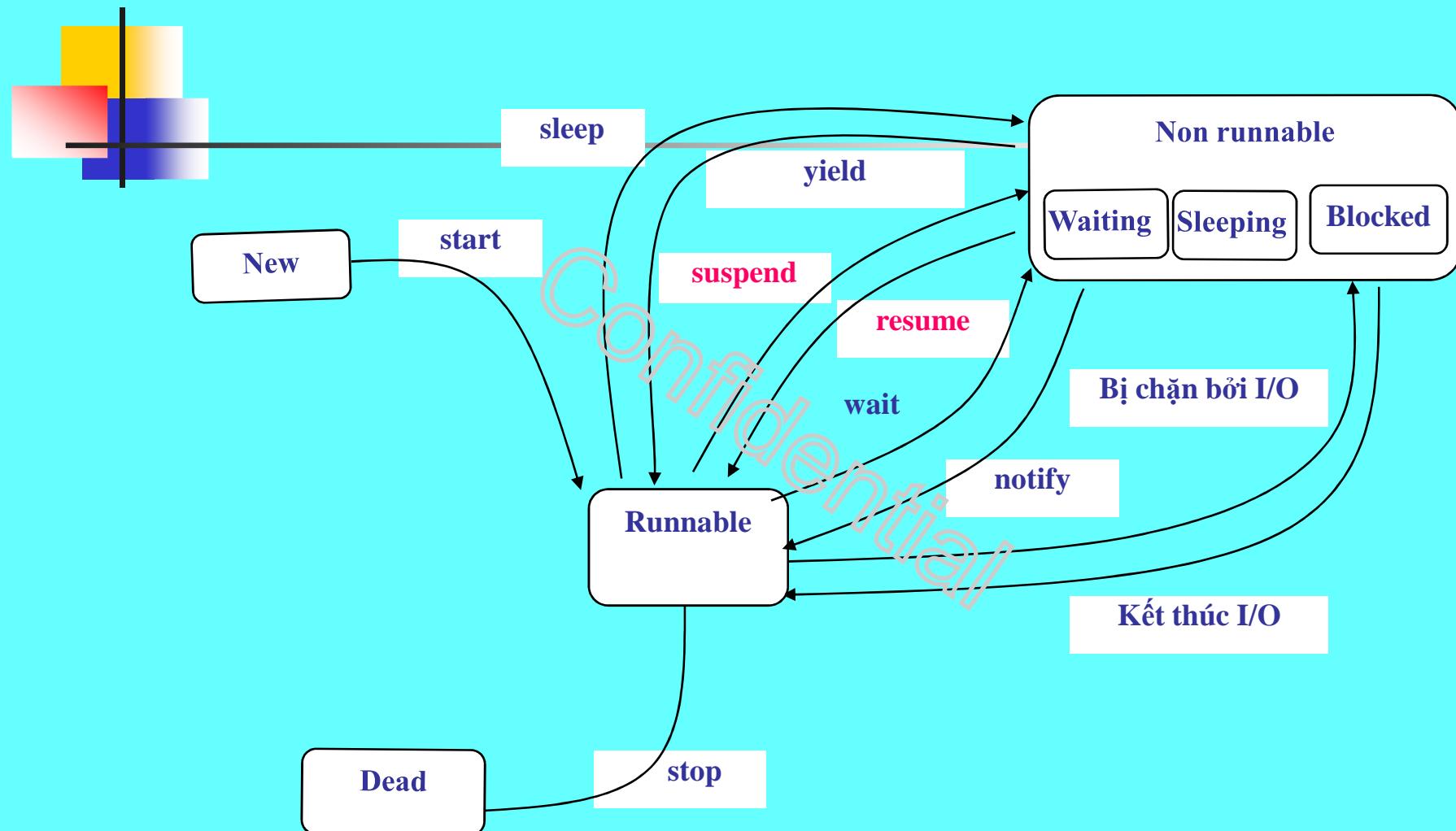
- **Trạng thái:**
 - born
 - ready to run
 - running
 - sleeping
 - waiting
 - ready
 - blocked
 - dead
- **Phương thức:**
 - start()
 - sleep()
 - wait()
 - notify()
 - run()
 - stop()

Các trạng thái của Thread

Một luồng có thể ở một trong các trạng thái sau:

- + **New**: Khi một luồng mới được tạo ra với toán tử `new()` và sẵn sàng hoạt động.
- + **Runnable**: Trạng thái mà luồng đang chiếm CPU để thực hiện, khi bắt đầu thì nó gọi hàm `start()`.
 - Bộ lập lịch phân luồng của hệ điều hành sẽ quyết định luồng nào sẽ được chuyển về trạng thái Runnable và hoạt động.
 - Cũng cần lưu ý rằng ở một thời điểm, một luồng ở trạng thái Runnable có thể hoặc không thể thực hiện.
- + **Non runnable (blocked)**: Từ trạng thái runnable chuyển sang trạng thái ngừng thực hiện (“bị chặn”) khi gọi một trong các hàm: `sleep()`, `suspend()`, `wait()`, hay *bị chặn lại ở Input/output*.

<Tiếp theo>

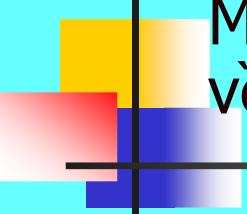


Các trạng thái của Thread

Một luồng có thể ở một trong các trạng thái sau:

- + **Waiting**: khi ở trạng thái Runnable, một luồng thực hiện hàm wait() thì nó sẽ chuyển sang trạng thái chờ đợi (Waiting).
- + **Sleeping**: khi ở trạng thái Runnable, một luồng thực hiện hàm sleep() thì nó sẽ chuyển sang trạng thái ngủ (Sleeping).
- + **Blocked**: khi ở trạng thái Runnable, một luồng bị chặn lại bởi những yêu cầu về tài nguyên, như yêu cầu vào/ra (*I/O*), thì nó sẽ chuyển sang trạng bị chặn (Blocked).

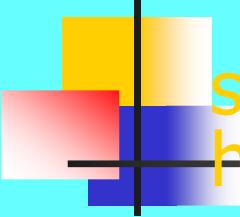
Các trạng thái của Thread



Mỗi luồng phải thoát ra khỏi trạng thái Blocked để quay về trạng thái Runnable, khi

- + Nếu một luồng đã được cho đi “ngủ” (sleep) sau khoảng thời gian bằng số micro giây n đã được truyền vào tham số của hàm sleep(n).
- + Nếu một luồng bị chặn lại vì vào/ra và quá trình này đã kết thúc.
- + Nếu luồng bị chặn lại khi gọi hàm wait(), sau đó được thông báo tiếp tục bằng cách gọi hàm notify() hoặc notifyAll().
- + Nếu một luồng bị chặn lại để chờ monitor của đối tượng đang bị chiếm giữ bởi luồng khác, khi monitor đó được giải phóng thì luồng bị chặn này có thể tiếp tục thực hiện (khái niệm monitor được đề cập ở phần sau).

Các trạng thái của Thread



+ Nếu một luồng bị chặn lại bởi lời gọi hàm `suspend()`, muốn thực hiện thì trước đó phải gọi hàm `resume()`.

+ Hàm `suspend()` có tác dụng tạm ngừng tuyễn, ít được dùng do không nhả tài nguyên của hệ thống, dễ dẫn đến deadlock.

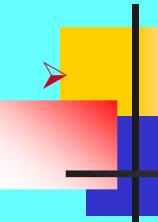
Nếu ta gọi các hàm không phù hợp đối với các luồng thì JVM sẽ phát sinh ra ngoại lệ `IllegalThreadStateException`.

+ Dead: Luồng chuyển sang trạng thái “chết” khi nó kết thúc hoạt động bình thường, hoặc gặp phải ngoại lệ không thực hiện tiếp được. Trong trường hợp đặc biệt, bạn có thể gọi hàm `stop()` để kết thúc (“giết chết”) một luồng.

Mức ưu tiên của các luồng

- Trong Java, mỗi luồng có một mức ưu tiên thực hiện nhất định.
- Khi chương trình chính thực hiện sẽ tạo ra luồng chính, luồng cha. Luồng này sẽ tạo ra các luồng con, và cứ thế tiếp tục.
- Theo mặc định, ~~một~~ luồng sẽ kế thừa mức ưu tiên của luồng cha của nó. Bạn có thể tăng hay giảm mức ưu tiên của luồng bằng cách sử dụng hàm `setPriority()`.
- Mức ưu tiên của các luồng có thể đặt lại trong khoảng từ `MIN_PRIORITY` (Trong lớp `Thread` được mặc định bằng 1) và `MAX_PRIORITY` (mặc định bằng 10), hoặc `NORM_PRIORITY` (mặc định là 5).
- Luồng có mức ưu tiên cao nhất tiếp tục thực hiện cho đến khi:
 - + Nó nhường quyền điều khiển cho luồng khác bằng cách gọi hàm `yield()`
 - + Nó dừng thực hiện (bị “dead” hoặc chuyển sang trạng thái bị chặn).
- có một luồng với mức ưu tiên cao hơn vào trạng thái

Mức ưu tiên của các luồng



Vấn đề này sinh là chọn luồng nào để thực hiện khi có nhiều hơn một luồng sẵn sàng thực hiện và có cùng một mức ưu tiên cao nhất? Nói chung, một số cơ sở sử dụng bộ lập lịch lựa chọn ngẫu nhiên, hoặc lựa chọn chúng để thực hiện theo thứ tự xuất hiện.

Ví dụ:

Chúng ta hãy xét chương trình hiển thị các quả bóng màu xanh hoặc đỏ này (chuyển) theo những đường nhất định.

Mỗi khi nhấn nút “Blue ball” thì có 5 luồng được tạo ra với mức ưu tiên thông thường (mức 5) để hiển thị và di chuyển các quả bóng xanh.

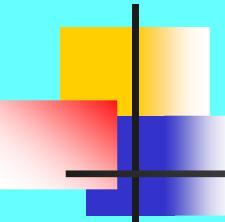
Khi nhấn nút “Red ball” thì cũng có 5 luồng được tạo ra với mức ưu tiên (mức 7) cao hơn mức thông thường để hiển thị và di chuyển các quả bóng đỏ.

Để kết thúc trò chơi bạn nhấn nút “Close”.

Mức ưu tiên của các luồng

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Bounce{
    public static void main(String arg[]){
        JFrame fr = new BounceFrame();
        fr.show();
    }
}
class BounceFrame extends JFrame{
    public BounceFrame(){
        setSize(300, 200);
        setTitle("Bong chuyen");
        addWindowListener(new WindowAdapter(){
            public void windowClosing(WindowEvent e){
                System.exit(0);
            }
        });
    }
}
```

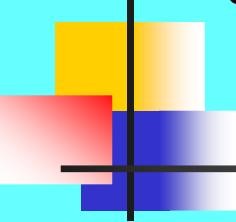
Mức ưu tiên của các luồng



```
Container contentPane = getContentPane();
canvas = new JPanel();
contentPane.add(canvas, "Center");
JPanel p = new JPanel();
addButton(p, "Blue ball", new ActionListener(){
    public void actionPerformed(ActionEvent evt){
        for(int i = 0; i < 5; i++){
            Ball b = new Ball(canvas, Color.blue);
            b.setPriority(Thread.NORM_PRIORITY);
            b.start();        }
    }
});
addButton(p, "Red ball", new ActionListener(){
    public void actionPerformed(ActionEvent evt){
        for(int i = 0; i < 5; i++){
            Ball b = new Ball(canvas, Color.red);
            b.setPriority(Thread.NORM_PRIORITY + 2);
            b.start();
        }
    }
});
```

A large, semi-transparent watermark reading "CONFIDENTIAL" diagonally across the slide content.

Mức ưu tiên của các luồng



```
addButton(p, "Close", new ActionListener(){
    public void actionPerformed(ActionEvent evt){
        canvas.setVisible(false);
        System.exit(0);
    }
});
contentPane.add(p, "South");
}
public void addButton(Container c, String title, ActionListener a){
    JButton b = new JButton(title);
    c.add(b);
    b.addActionListener(a);
}
private JPanel canvas;
}
```

A large, diagonal watermark reading "Confidential" is overlaid across the code area.

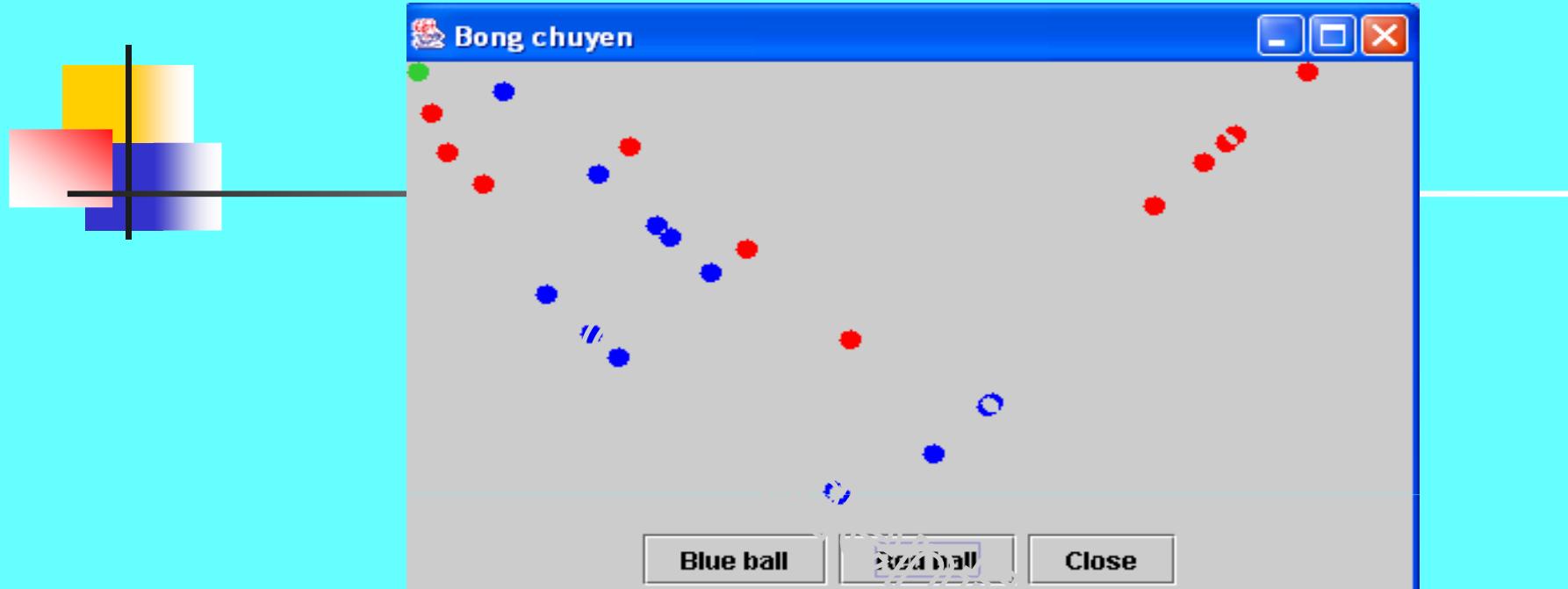
Mức ưu tiên của các luồng

```
class Ball extends Thread{  
    public Ball(JPanel b, Color c){  
        box = b; color = c;  
    }  
    public void draw(){  
        Graphics g = box.getGraphics(); g.setColor(color);  
        g.fillOval(x, y, XSIZE, YSIZE); g.dispose();  
    }  
    public void move(){  
        if(!box.isVisible()) return;  
        Graphics g = box.getGraphics();  
        g.setXORMode(box.getBackground()); g.setColor(color);  
        g.fillOval(x, y, XSIZE, YSIZE);  
        x += dx; y += dy;  
        Dimension d = box.getSize();  
        if(x < 0){  
            x = 0; dx = -dx;  
        }  
        if(x + XSIZE >= d.width){  
            x = d.width - XSIZE; dx = -dx;  
        }  
        if(y < 0){  
            y = 0; dy = -dy; }  
    }  
}
```

Mức ưu tiên của các luồng

```
if(y + YSIZE >= d.height){  
    y = d.height - YSIZE; dy = -dy;  
}  
g.fillOval(x, y, XSIZE, YSIZE);  
g.dispose();  
}  
public void run(){  
    try{  
        for(int i = 1; i <= 1000; i++){  
            move(); sleep(5);  
        }  
    }catch(InterruptedException e){  
    }  
}  
private JPanel box;  
private static final int XSIZE = 10;  
private static final int YSIZE = 10;  
private int x = 0;  
private int y = 0;  
private int dx = 2;  
private int dy = 2;  
private Color color;  
}
```

Mức ưu tiên của các luồng

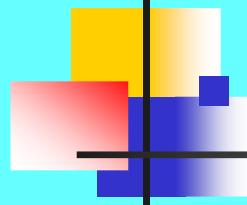


- Chạy chương trình trên chúng ta nhận thấy hình như những quả bóng đỏ nảy nhanh hơn vì các luồng thực hiện chúng có mức ưu tiên cao hơn.

Lưu ý: Các luồng có mức ưu tiên thấp hơn sẽ không có cơ hội thực hiện nếu những luồng cao hơn nhường, hoặc nhường bằng hàm `yield()`. Nếu có những luồng đang ở trạng thái Runnable mà có mức ưu tiên ít nhất bằng mức ưu tiên của luồng vừa nhường thì một trong số chúng được xếp lịch để thực hiện.

- + Bộ lập lịch thường xuyên tính lại mức ưu tiên của các luồng đang thực hiện
- + Tìm luồng có mức ưu tiên cao nhất để thực hiện.

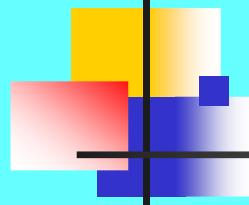
Đa tuyến với Applets



Trong Java ta có thể tạo ra các tuyến thi hành song song bằng cách triển khai giao diện Runnable

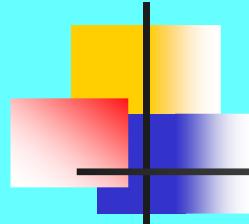
- Java không hỗ trợ kế thừa bội
- Muốn kế thừa từ một lớp nào đó mà lại muốn đa tuyến thì bắt buộc sử dụng giao diện Runnable
- Các chương trình Java dựa trên Applet thường sử dụng nhiều hơn một tuyến

Đa tuyến với Applets



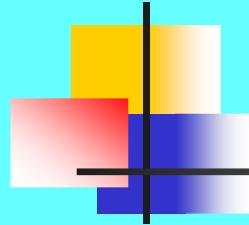
Trong đa tuyến với Applets, Lớp ‘`java.applet.Applet`’ là lớp con được tạo ra một Applet người sử dụng đã định nghĩa

- Lớp con của Applet không thể dẫn xuất được trực tiếp từ lớp Thread.
- Cách để lớp con Applet là tuyến:
 - implements Runnable
 - Truyền đối tượng Runnable vào hàm constructor của Thread.



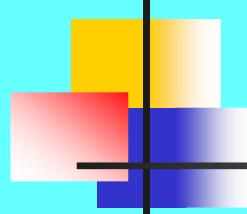
Sự đồng bộ

- Khi nhiều tuyến truy cập tài nguyên dùng chung
 - Tài nguyên không thể chia xẻ, khi đó tài nguyên có thể bị phá hỏng
- Ví dụ : Một luồng đọc dữ liệu, trong khi luồng khác lại thay đổi
- Cần cho phép một luồng hoàn thành tác vụ của nó, rồi cho phép luồng kế tiếp thực thi



Sự đồng bộ

- Thêm nhập các tài nguyên/dữ liệu bởi nhiều tuyến
- Sự đồng bộ (Synchronization)
- Sự quan sát (Monitoring)



Sự đồng bộ

- Để thâm nhập sự quan sát của một đối tượng, lập trình viên sử dụng từ khóa ‘**synchronized**’ khi khai báo phương thức.
- Mỗi một đối tượng sẽ có một bộ quản lý khóa, ~~chỉ~~ cho một phương thức “**synchronized**” của đối tượng đó chạy tại một thời điểm
- Khi một tuyến đang ~~đang~~ được thực thi trong phạm vi một phương thức đồng bộ (**synchronized**), bất kỳ tuyến khác hoặc phương thức đồng bộ khác mà cố gắng gọi nó trong thời gian đó sẽ phải đợi

Đồng bộ hóa

- Các luồng chia sẻ với nhau cùng một không gian bộ nhớ, nghĩa là chúng có thể chia sẻ với nhau các tài nguyên.
- Khi có nhiều hơn một luồng cùng muốn sử dụng một tài nguyên sẽ xuất hiện tình trạng căng thẳng, ở đó chỉ cho phép một luồng được quyền truy cập.
- Để cho các luồng chia sẻ với nhau được các tài nguyên và hoạt động hiệu quả, luôn đảm bảo nhất quán dữ liệu thì phải có cơ chế đồng bộ chúng.

Đồng bộ hóa

- Mẫu chốt của sự đồng bộ là khái niệm “monitor” (giám sát) hay còn gọi là “semaphore” (cờ hiệu)
- Khái niệm “semaphore” thường được sử dụng để điều khiển đồng bộ các hoạt động truy cập vào những tài nguyên dùng chung.
- Một luồng muốn truy cập ~~vào~~ một tài nguyên dùng chung (như biến dữ liệu) thì trước tiên nó phải yêu cầu để có được monitor riêng.
- Khi có được monitor thì luồng như có được “chìa khoá” để “mở cửa” vào miền “tranh chấp” để sử dụng những tài nguyên đó.

Đồng bộ hóa

-  Cơ chế monitor thực hiện hai nguyên tắc đồng bộ chính:
 - + Không một luồng nào khác được phân monitor khi có một luồng đã yêu cầu và đang chiếm giữ. Những luồng có yêu cầu monitor sẽ phải chờ cho đến khi monitor được giải phóng.
 - + Khi có một luồng giải phóng (ra khỏi) monitor, một luồng đang chờ *monitor* có thể truy cập vào tài nguyên dùng chung tương ứng với monitor đó.
- Mọi đối tượng trong Java đều có monitor, mỗi đối tượng có thể được sử dụng như một khóa loại trừ nhau, cung cấp khả năng để đồng bộ truy cập vào những tài nguyên chia sẻ.

Trong lập trình có hai cách để thực hiện đồng bộ:

- + Các hàm được đồng bộ
- + Các khối được đồng bộ

Đồng bộ hóa

Các hàm đồng bộ

- Hàm của một lớp chỉ cho phép một luồng được thực hiện ở một thời điểm thì nó phải khai báo **synchronized**, được gọi là *hàm đồng bộ*.
- Một luồng muốn thực hiện hàm đồng bộ thì nó phải chờ để có được monitor của đối tượng có hàm đó.
- Trong khi một luồng đang thực hiện hàm đồng bộ thì tất cả các luồng khác muốn thực hiện hàm này của cùng một đối tượng, đều phải chờ cho đến khi luồng đó thực hiện xong và được giải phóng.
- Bằng cách đó, những hàm được đồng bộ sẽ không bao giờ bị tắc nghẽn.

Đồng bộ hóa

Các hàm đồng bộ

- Những hàm không được đồng bộ của đối tượng có thể được gọi thực hiện mọi lúc bởi bất kỳ đối tượng nào.
- Khi chạy, chương trình sẽ thi hành tuần tự các lệnh cho đến khi kết thúc chương trình.
- Trong Java, hàm đồng bộ ~~có thể khai báo static~~. Các lớp cũng có thể có các monitor tương tự như đối với các đối tượng.
- Một luồng yêu cầu monitor của lớp trước khi nó có thể thực hiện với một hàm được đồng bộ tĩnh (static) nào đó trong lớp, đồng thời các luồng khác muốn thực hiện những hàm như thế của cùng một lớp thì bị chặn lại.

Đồng bộ hóa

Ví dụ: Hệ thống ngân hàng có 10 tài khoản, trong đó có các giao dịch chuyển tiền giữa các tài khoản với nhau một cách ngẫu nhiên. Chương trình tạo ra 10 luồng cho 10 tài khoản.

Mỗi giao dịch được một luồng phục vụ sẽ chuyển một lượng tiền ngẫu nhiên từ một tài khoản sang tài khoản khác.

- Nếu chương trình thực hiện với 10 luồng hoạt động không đồng bộ để chuyển tiền giữa các tài khoản trong ngân hàng.
- Vấn đề sẽ nảy sinh khi có hai luồng đồng thời muốn chuyển tiền vào cùng một tài khoản. Giả sử hai luồng cùng thực hiện:

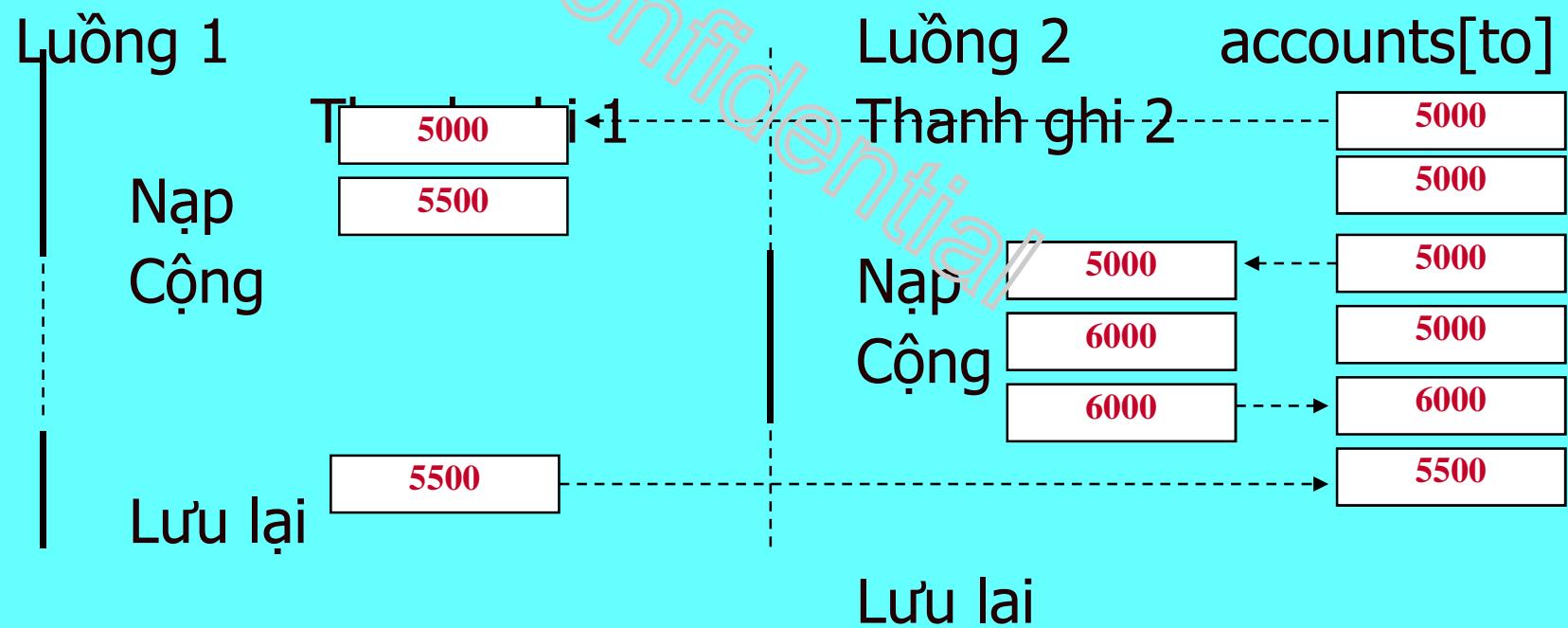
accounts[to] += amount;

Câu lệnh này được thực hiện như sau:

1. Nạp accounts[to] vào thanh ghi.
2. Cộng số tiền trong tài khoản accounts với amount
3. Lưu lại kết quả cho accounts[to]

Đồng bộ hóa

Chúng ta có thể giả thiết luồng thứ nhất thực hiện bước 1 và 2 với amount = 500, sau đó nó bị ngắt. Luồng thứ hai có thể thực hiện trọn vẹn cả ba bước trên với amount = 1000, sau đó luồng thứ nhất kết thúc việc cập nhật bằng cách thực hiện nốt bước 3. Quá trình này được mô tả như ở hình sau :



Kết thúc luồng thứ nhất, accounts[to] có 6000, nhưng ngay sau đó luồng thứ hai kết thúc thì cũng chính tài khoản đó chưa chuyển tiền đi ⁵⁰⁹

Đồng bộ hóa

Java sử dụng cơ chế đồng bộ khá hiệu quả là monitor.

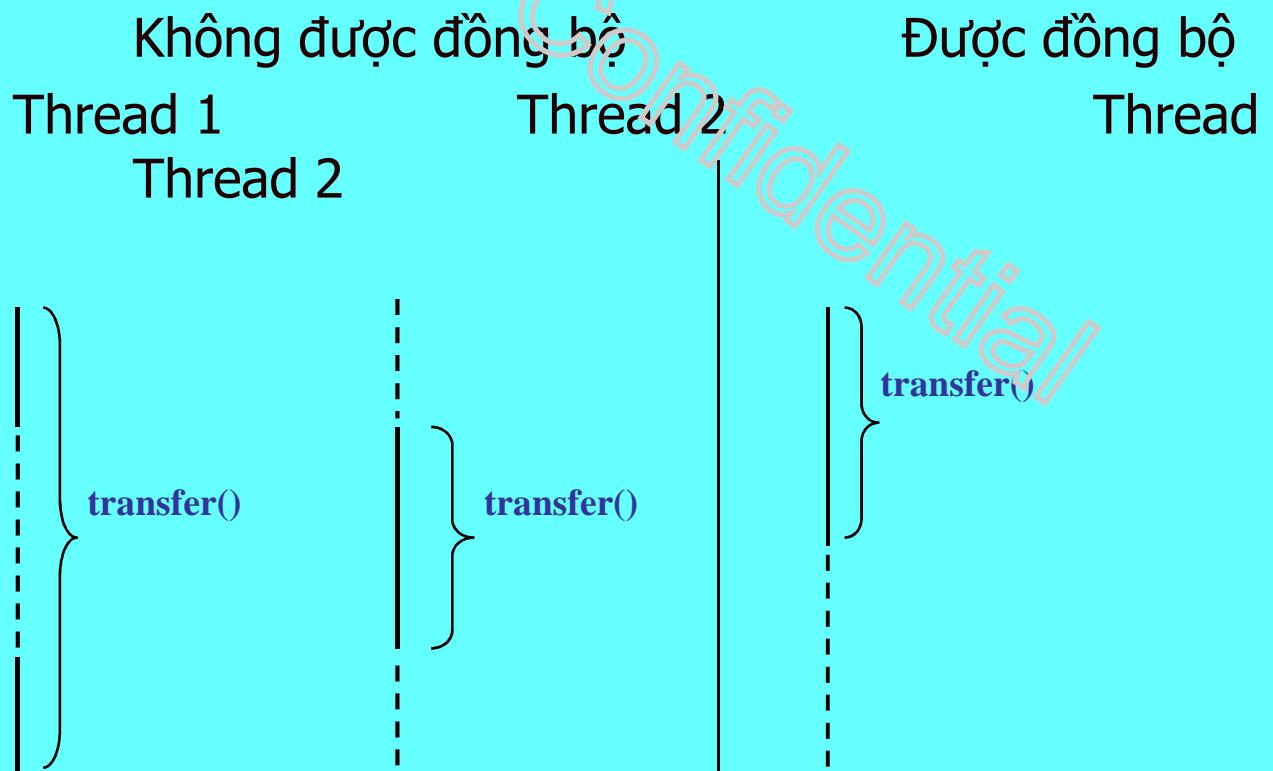
Một hàm sẽ không bị ngắt nếu bạn khai báo nó là synchronized.

```
public synchronized void transfer(int from, int to, int amount){  
    if(accounts[from] < amount) return;  
    accounts[from] -= amount;  
    accounts[to] += amount;  
    numTransacts++;  
    if(numTransacts % NTEST == 0) test();  
}  
public synchronized void test(){  
    int sum = 0;  
    for(int i = 0; i < accounts.length; i++)  
        sum += accounts[i];  
    System.out.println("Giao dich: " + numTransacts  
        + " tong so: " + sum);  
}
```

Đồng bộ hóa

Khi có một luồng gọi hàm được đồng bộ thì nó được đảm bảo rằng hàm này phải thực hiện xong thì luồng khác mới được sử dụng đối với cùng một đối tượng.

Hoạt động của các luồng không đồng bộ và đồng bộ của hai luồng thực hiện gọi hàm transfer()



Các khoá đối tượng

Khi một luồng gọi một hàm được đồng bộ thì đối tượng của nó bị “khoá”, giống như khoá cửa phòng.

- Như vậy, khi một luồng khác muốn gọi hàm được đồng bộ của cùng đối tượng đó thì sẽ không mở được.
- Sau khi thực hiện xong, luồng ở bên trong giải phóng hàm được đồng bộ và sử dụng, ra khỏi đối tượng và đưa chìa khoá ra ngoài bậc cửa để những luồng khác có thể tiếp tục công việc của mình.

Các khoá đối tượng

Một luồng có thể giữ nhiều khoá đối tượng ở cùng một thời điểm, như trong khi đang thực hiện một lời gọi hàm đồng bộ của một đối tượng, nó lại gọi tiếp hàm đồng bộ của đối tượng khác.

- Nhưng, tại mỗi thời điểm, mỗi khoá đối tượng chỉ được một luồng sở hữu.

Chúng ta hãy phân tích chi tiết hơn hoạt động của hệ thống ngân hàng. Một giao dịch chuyển tiền sẽ không thực hiện được nếu không còn đủ tiền. Nó phải chờ cho đến các tài khoản khác chuyển tiền đến và khi có đủ thì mới thực hiện được giao dịch đó.

```
public synchronized void transfer(int from, int to, int amount){  
    while(accounts[from] < amount)  
        wait();  
    // Chuyển tiền  
}
```

Các khoá đối tượng



Chúng ta sẽ làm gì khi trong tài khoản không có đủ tiền? Tất nhiên là phải chờ cho đến khi có đủ tiền trong tài khoản. Nhưng transfer() là hàm được đồng bộ. Do đó, khi một luồng đã chiếm được khoá đối tượng thì luồng khác sẽ không có cơ hội sở hữu trừ nó, cho đến khi luồng trước giải phóng khoá đó.

- Khi wait() được gọi ở trong hàm được đồng bộ (như ở transfer()), luồng hiện thời sẽ bị chặn lại và trao lại khoá đối tượng cho luồng khác.
- Có sự khác nhau thực sự giữa luồng đang chờ để sử dụng hàm đồng bộ với hàm bị chặn lại bởi hàm wait().
 - Khi một luồng gọi wait() thì nó được đưa vào danh sách hàng đợi.
 - Cho đến khi các luồng chưa được đưa ra khỏi danh sách hàng đợi thì bộ lập lịch sẽ bỏ qua và do vậy chúng không thể tiếp tục được.
 - Để đưa một luồng ra khỏi danh sách hàng đợi thì phải có một luồng khác gọi notify() hoặc notifyAll() trên *cùng một đối tượng*.

notify() đưa một luồng bất kỳ ra khỏi danh sách hàng đợi.

notifyAll() đưa tất cả các luồng ra khỏi danh sách hàng đợi.

Các khoá đối tượng

Những luồng đưa ra khỏi danh sách hàng đợi sẽ được bộ lập lịch kích hoạt chúng. Ngay tức khắc luồng nào chiếm được khoá đối tượng thì sẽ bắt đầu thực hiện. Như vậy, trong hàm transfer() chúng ta gọi notifyAll() khi kết thúc việc chuyển tiền để một trong các luồng có thể được tiếp tục thực hiện và tránh bế tắc.

Cuối cùng chương trình sử dụng cơ chế đồng bộ được viết lại như sau.

```
public class SynBankTransfer{  
    public static void main(String arg[]){  
        Bank b = new Bank(NACCOUNTS,INI_BALANCE);  
        for(int i = 0; i < NACCOUNTS; i++){  
            TransferThread t = new TransferThread(b, i, INI_BALANCE);  
            t.setPriority(Thread.NORM_PRIORITY + i % 2);  
            t.start();  
        }  
    }  
    public static final int NACCOUNTS = 10;  
    public static final int INI_BALANCE = 10000;  
}
```

Các khoá đối tượng

```
class Bank{  
    public static final int NTEST = 1000;  
    private int[] accounts;  
    private long numTransacts = 0;  
    public Bank(int n, int initBalance){  
        accounts = new int[n];  
        for(int i = 0; i < accounts.length; i++)  
            accounts[i] = initBalance;  
        numTransacts = 0;  
    }  
    public void transfer(int from, int to, int amount){  
        while(accounts[from] < amount) wait();  
        accounts[from] -= amount;  
        accounts[to] += amount;  
        numTransacts++;  
        notifyAll();  
        if(numTransacts % NTEST == 0) test();  
    }  
}
```

Các khoá đối tượng

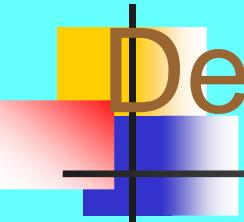
```
public synchronized void test(){  
    int sum = 0;  
    for(int i = 0; i < accounts.length; i++)  
        sum += accounts[i];  
    System.out.println("Giao dich: " + numTransacts + " tong so: " + sum);  
}  
  
public int size(){  
    return accounts.length;  
}  
}  
  
class TransferThread extends Thread{  
    private Bank bank;  
    private int fromAcc;  
    private int maxAmount;  
    public TransferThread(Bank b, int from, int max){  
        bank = b;  
        fromAcc = from; maxAmount = max;  
    }  
}
```

Confidential

Các khoá đối tượng

```
public void run(){  
    try{  
        while(!interrupted()){  
            int toAcc = (int)(bank.size() * Math.random());  
            int amount = (int)(maxAmount * Math.random());  
            bank.transfer(fromAcc, toAcc, amount);  
            sleep(1);  
        }  
    }catch(InterruptedException e){  
    }  
}
```

Nếu bạn chạy chương trình với các hàm transfer(), test() được đồng bộ thì mọi việc sẽ thực hiện chính xác đúng theo yêu cầu. Tuy nhiên, bạn cũng có thể nhận thấy chương trình sẽ chạy chậm hơn chút ít bởi vì phải trả giá cho cơ chế đồng bộ nhằm đảm bảo cho hệ thống hoạt động chính xác, đảm bảo nhất quán dữ liệu, hoặc tránh gây ra tắc nghẽn.



Deadlock

- Một “deadlock” xảy ra khi hai tuyến có một phụ thuộc vòng quanh trên một cặp đối tượng đồng bộ

Q Confidential

Deadlock

 Cơ chế đồng bộ trong Java là rất tiện lợi, khá mạnh, nhưng không giải quyết được mọi vấn đề này sinh trong quá trình xử lý đa luồng.

Ví dụ : ở Account 1 có 2000\$, Account 2 có 3000\$ và Thread 1 cần chuyển 3000\$ từ Account 1 sang Account 2, ngược lại Thread 2 cần chuyển 3500\$ từ Account 2 sang Account 1.

Khi đó, Thread 1 và Thread 2 rơi vào tình trạng *chết tắc, hoặc tắc nghẽn* vì chúng chặn lẫn nhau.

Một hệ thống mà tất cả các luồng (tiến trình) bị chặn lại để chờ lẫn nhau và không một luồng (tiến trình) nào thực hiện tiếp thì được gọi là hệ thống bị chết tắc (tắc nghẽn).

Trong tình huống ở trên, cả hai luồng đều phải gọi `wait()` xử lý hai tài khoản đều không đủ số tiền để

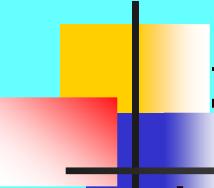
Deadlock

Trong chương trình SynBankTransfer.java, hiện tượng tắc nghẽn không xuất hiện bởi một lý do đơn giản. Mỗi giao dịch chuyển tiền nhiều nhất là 10000\$.

Có 10 tài khoản với tổng số tiền là 100000\$. Do đó, ở mọi thời điểm đều có ít nhất một tài khoản có không ít hơn 10000\$, nghĩa là không phụ trách tài khoản đó được phép thực hiện.

Tuy nhiên, khi lập trình ta có thể gây ra tình huống khác có thể làm xuất hiện tắc nghẽn

Deadlock

 Ví dụ :

Trong SynBankTransfer.java thay vì gọi `notifyAll()` ta gọi `notify()`.

Như ở trên đã phân tích, `notifyAll()` thông báo cho tất cả các luồng đang chờ để có đủ tiền chuyển đi có thể tiếp tục thực hiện, còn `notify()` chỉ báo cho một luồng được tiếp tục.

Khi đó, nếu luồng được thông báo lại không thể thực hiện, vì không đủ tiền để chuyển chặng hạn, thì tất cả các luồng khác cũng sẽ bị chặn lại.

Deadlock

Chúng ta hãy xét kịch bản sau:

+ Account 1: 19000\$

+ Tất cả các Account còn lại đều có 9000\$

+ Thread 1: chuyển 9500\$ từ Account 1 sang Account 2

+ Tất luồng khác đều chuyển sang tài khoản khác một lượng tiền là 9100\$.

Chỉ có Thread 1 đủ tiền để chuyển còn các luồng khác bị chặn lại. Thread 1 thực hiện chuyển tiền xong ta có:

+ Account 1: 9500\$

+ Account 2: 18500\$

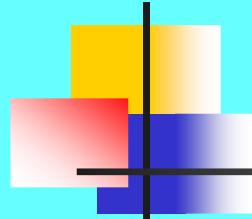
+ Tất cả các Account còn lại đều có 9000\$

Giả sử Thread 1 gọi `notify()`. Hàm này chỉ thông báo cho một luồng ngẫu nhiên để nó có thể tiếp tục thực hiện. Giả sử đó là Thread 3. Nhưng luồng này cũng không chuyển được vì không đủ tiền ở tài khoản Account 3, nên phải chờ (gọi `wait()`). Thread 1 vẫn tiếp tục thực hiện. Một giao dịch mới ngẫu nhiên lại được tạo ra. Ví dụ

Thread 1: chuyển 9600\$ từ Account 1 sang Account 2.

Bây giờ Thread lại gọi `wait()`, và như vậy tất cả các luồng đều rơi vào tình trạng tắc nghẽn.

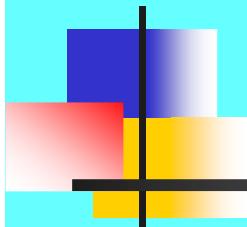
Qua ví dụ trên cho thấy, một ngôn ngữ lập trình có cơ chế hỗ trợ đồng bộ là chưa đủ để giải quyết vấn đề tắc nghẽn. Quan trọng là khi thiết kế chương trình, ta phải đảm bảo rằng ở mọi thời điểm có ít nhất một luồng (tiến trình) tiếp tục thực hiện.



Phương thức finalize()

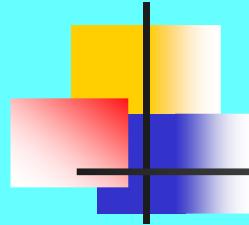
- Java cung cấp một cách để làm sạch một tiến trình trước khi điều khiển trở lại hệ điều hành
- Phương thức `finalize()`, nếu hiện diện, sẽ được thực thi trên mỗi đối tượng, trước khi sự dọn rác
- Câu lệnh của phương thức `finalize()` như sau:
 - **`protected void finalize() throws Throwable`**
- Tham chiếu không phải là sự dọn rác; chỉ các đối tượng mới được dọn rác

CHƯƠNG 4 : NGÔN NGỮ LẬP TRÌNH JAVA



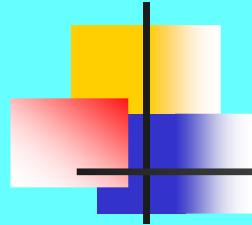
Confidential

4.8 Các luồng I/O



Các luồng

- Các luồng là những “đường ống” để gửi và nhận thông tin trong các chương trình java.
- Khi một luồng đọc hoặc ghi, các luồng khác bị khoá.
- Nếu lỗi xảy ra trong khi đọc hoặc ghi luồng, một biệt lệ sẽ được tạo ra.
- Lớp ‘java.lang.System’ định nghĩa luồng nhập và xuất chuẩn.



Các lớp luồng I/O

- Lớp System.out.
- Lớp System.in.
- Lớp System.err.

Confidential

Các lớp luồng I/O

Để xử lý mọi loại dữ liệu, java chia luồng thành 2 loại : luồng byte (byte stream) và luồng ký tự (character stream)

- Lớp **InputStream** và **OutputStream** là hai lớp cơ sở cho mọi luồng nhập xuất hướng byte
- Lớp **Reader/ Writer** hai lớp cơ sở cho việc đọc ghi hướng ký tự

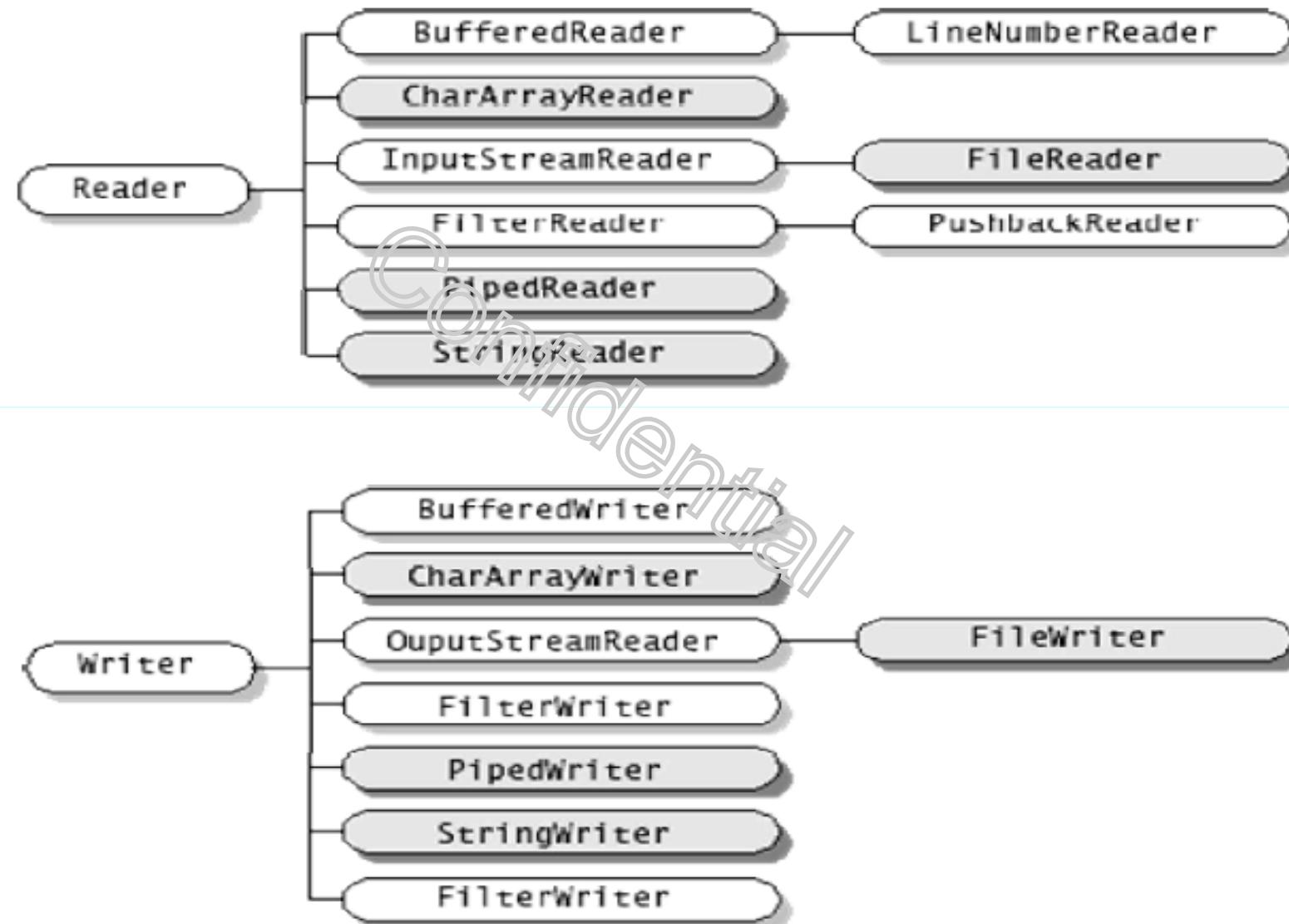


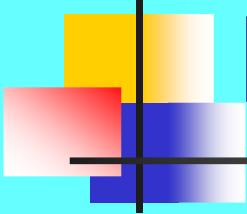
Lớp nhập xuất hướng ký tự

Reader và Writer là hai lớp cơ sở trừu tượng cho luồng hướng ký tự

- Cung cấp ~~một~~ giao diện chung cho tất cả các lớp đọc/ghi hướng ký tự
- Mỗi lần đọc/ghi ra luồng thì đọc 2 byte tương ứng với một ký tự

Mô hình phân cấp đọc/ghi hướng ký





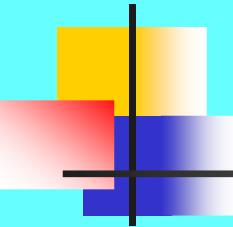
Lớp Reader

- Lớp Reader và InputStream có một giao diện giống nhau, chúng chỉ khác nhau về kiểu dữ liệu đầu vào
- Lớp Reader có các phương thức đọc một ký tự hoặc mảng các ký tự
- Các phương thức:
 - **int read()**
 - **int read(char cbuf[])**
 - **int read(char cbuf[], int offset, int length)**

Lớp Writer

Lớp Writer và OutputStream có một giao diện giống nhau, chúng chỉ khác nhau về kiểu dữ liệu đầu vào

- Lớp Writer định nghĩa các phương thức để ghi một ký tự hoặc mảng các ký tự ra luồng
- Các phương thức:
 - `int write(int c)`
 - `int write(char cbuf[])`
 - `int write(char cbuf[], int offset, int length)`



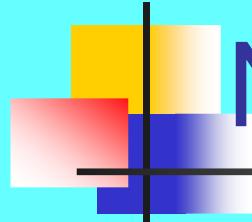
Lớp Writer

- Hỗ trợ các phương thức sau :
 - **flush()**
 - **close()**

Confidential

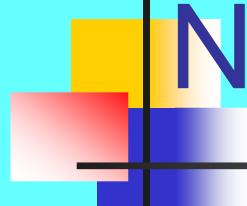
Nhập/xuất chuỗi và mảng ký tự

- Hỗ trợ nhập và xuất từ các vùng đệm bộ nhớ
- Hỗ trợ 8 bit ký tự nhập và kết xuất
- Lớp ‘CharArrayReader’ không bổ sung phương thức mới vào các phương thức mà lớp ‘Reader’ cung cấp.



Nhập/xuất chuỗi và mảng ký tự

- Lớp ‘CharArrayWriter’ bổ sung phương thức sau đây vào phương thức của lớp ‘Writer’ cung cấp:
 - **reset()**
 - **size()**
 - **toCharArray()**
 - **toString()**
 - **writeTo()**



Nhập/xuất chuỗi và mảng ký tự

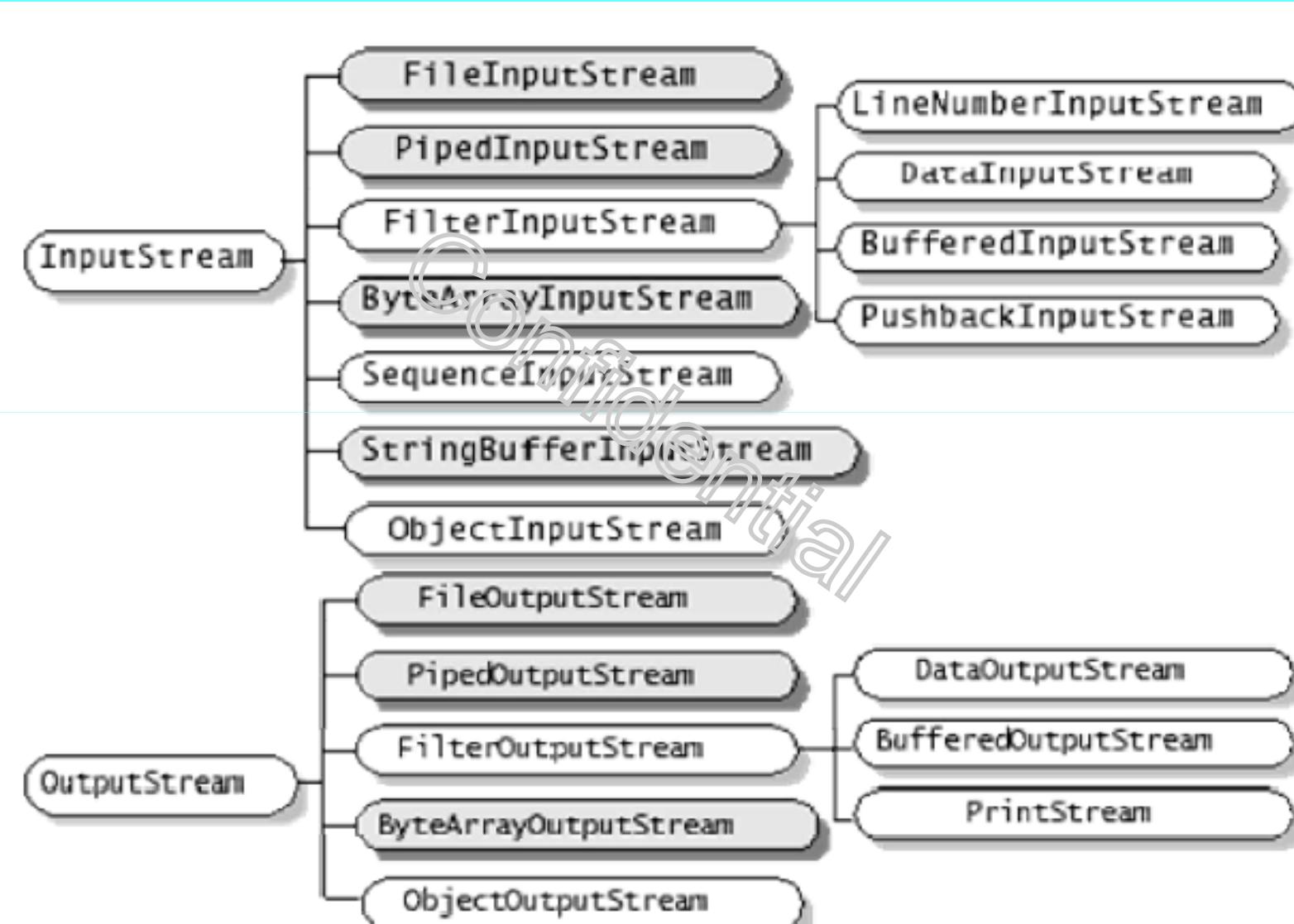
- Lớp ‘StringReader’ trợ giúp đọc các ký tự đầu vào từ sâu chuỗi.
- Nó không bổ sung bất kỳ phương thức nào mà lớp Reader cung cấp.
- Lớp ‘StringWriter’ trợ giúp để ghi luồng kết xuất ký tự ra một đối tượng ‘StringBuffer’.
- Lớp này bổ sung thêm các phương thức sau:
 - **getBuffer()**
 - **toString()**

Lớp hướng byte

Để có thể đọc ghi 1 byte, ta sử dụng luồng hướng byte

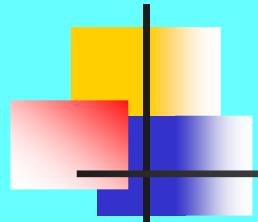
- Hai lớp `InputStream` và `OutputStream` là hai lớp cơ sở trừu tượng cho các luồng hướng byte
- Mỗi lần đọc/ghi ra luồng/thì đọc 8 bits ra luồng

Mô hình phân cấp đọc/ghi hướng byte



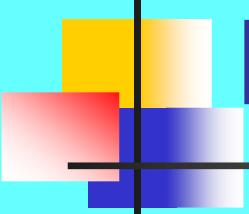
Lớp InputStream

- Là lớp trừu tượng
- Định nghĩa cách nhận dữ liệu
- Cung cấp số phương thức dùng để đọc và các luồng dữ liệu làm đầu vào.
- Trong lớp InputStream có các phương thức cho việc đọc một byte hoặc mảng các byte
- Các phương thức:
 - **int read()**
 - **int read(byte cbuf[])**
 - **int read(byte cbuf[], int offset, int len)**



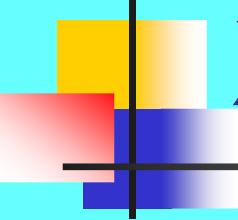
Lớp OutputStream

- Là lớp trừu tượng.
- Định nghĩa cách ghi dữ liệu vào luồng.
- Cung cấp tập các phương thức trợ giúp trong việc tạo, ghi và xử lý các luồng xuất.
- Lớp OutputStream có các phương thức để ghi một byte hoặc mảng các byte ra luồng
- Các phương thức:
 - **write(int c)**
 - **write(byte cbuf[])**
 - **write(byte[], int offset, int length)**



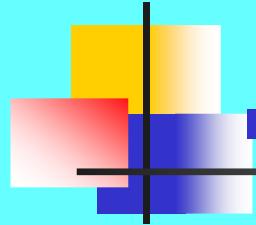
Nhập mảng các Byte

- Sử dụng các đệm bộ nhớ
- Lớp **ByteArrayInputStream**
- Tạo ra một luồng nhập từ đệm bộ nhớ về mảng các byte.
 - Không hỗ trợ các phương thức mới
 - Các phương thức nộp chồng của lớp InputStream, giống như ‘read()’, ‘skip()’, ‘available()’ và ‘reset()’.



Xuất mảng các Byte

- sử dụng các vùng đệm bộ nhớ
- Lớp **ByteArrayOutputStream**
 - Tạo ra một luồng kết xuất trên mảng byte
 - Cung cấp các khả năng bổ sung cho mảng kết xuất tăng cường nhằm chừa chỗ cho dữ liệu mới ghi vào.
 - Cũng cung cấp các phương thức để chuyển đổi luồng tới mảng byte, hay đối tượng String.

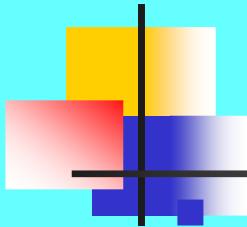


Phương thức của lớp **ByteArrayOutputStream** :

- **reset()**
- **size()**
- **writeTo()**

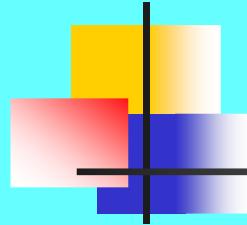
Confidential

Các lớp nhập/xuất File



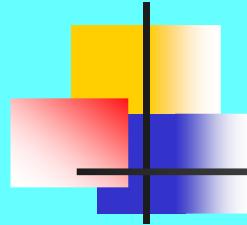
Các lớp này trợ giúp trong Java để hỗ trợ các thao tác nhập và xuất:

- File
- FileDescriptor
- FileInputStream
- FileOutputStream
- FileReader
- FileWriter
- Các lớp File, FileDescriptor, và RandomAccessFile được sử dụng hỗ trợ trực tiếp hoặc truy cập nhập/xuất ngẫu nhiên.



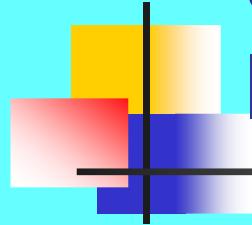
Lớp tập tin

- Được sử dụng truy cập các đối tượng tập tin và thư mục
- Những tập tin có tên được đặt tên theo qui ước của hệ điều hành.
- Lớp này cung cấp phương thức khởi tạo để tạo ra các thư mục và tập tin
- Tất cả các thao tác thư mục và tập tin đều được sử dụng các phương thức truy cập và các phương thức thư mục mà các lớp tập tin cung cấp



Lớp tệp tin

- Để xử lý tệp tin ngoại trú, ta sử dụng các luồng liên quan đến tệp tin như : FileInputStream, FileOutputStream, FileReader, FileWriter
- FileInputStream và FileOutputStream phục vụ cho việc đọc ghi tệp tin hướng Byte
- FileReader và FileWriter phục vụ cho việc đọc ghi tệp tin hướng ký tự



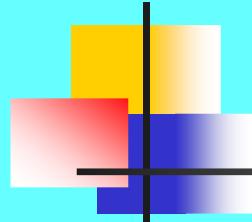
Các hàm tạo của các lớp tương ứng để liên kết luồng với một tệp tin cụ thể

- **public void** FileInputStream (String FileName)
- **public void** FileInputStream (File file)
- **public void** FileOutputStream (String FileName)
- **public void** FileOutputStream (File file)
- **public void** FileWriter (String FileName)
- **public void** FileWriter (File file)
- **public void** FileReader (String FileName)
- **public void** FileReader (File file)

Nhập / xuất lọc

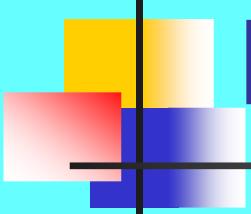
Lọc:

- Về cơ bản được sử dụng để thích ứng các luồng theo các nhu cầu của chương trình cụ thể.
- Bộ lọc nằm giữa luồng nhập và luồng xuất.
- Thực hiện một số tiến trình đặc biệt trên các byte được chuyển giao từ đầu vào đến kết xuất.
- Có thể phối hợp để thực hiện một dãy các tùy chọn lọc.



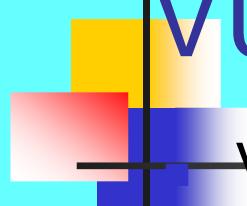
Lớp FilterInputStream

- Là lớp trừu tượng.
- Là cha của tất cả các lớp luồng nhập đã lọc.
- Cung cấp khả năng tạo ra một luồng từ luồng khác.
- Một luồng có thể đọc và cung cấp dưới dạng kết xuất cho luồng khác.
- Duy trì một dãy các đối tượng của lớp ‘InputStream’
- Cho phép tạo ra nhiều bộ lọc kết xích



Lớp FilterOutputStream

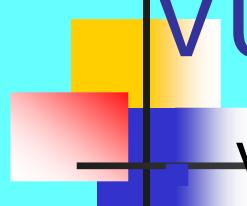
- Là dạng bổ trợ cho lớp ‘FilterInputStream’.
- Là cha của ~~tất~~ cả các lớp luồng kết xuất.
- Duy trì đối tượng của lớp ‘OutputStream’ như là một biến ‘out’.
- Dữ liệu ghi ra lớp này có thể sửa đổi để thực hiện các thao tác lọc, và sau đó phản hồi đến đối tượng ‘OutputStream’.



Vùng đệm nhập/xuất

Vì các thao tác với ổ cứng, mạng thường lâu hơn so với thao tác bộ nhớ trong

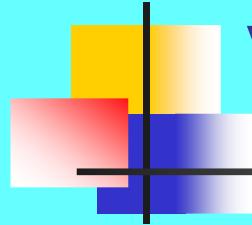
- Kỹ thuật sử dụng vùng đệm nhằm tăng tốc độ đọc/ghi
- Với kỹ thuật vùng đệm sẽ giảm được số lần đọc/ghi luồng
- Trong Java ta có thể tạo ra vùng đệm của các lớp :
 - `BufferInputStream`
 - `BufferOutputStream`
 - `BufferedReader`
 - `BufferWriter`



Vùng đệm nhập/xuất

Vùng đệm:

- Là kho lưu trữ dữ liệu.
- Có thể cung cấp dữ liệu thay vì quay trở lại nguồn dữ liệu gốc ban đầu.
- Java sử dụng vùng đệm nhập và kết xuất để tạm thời lập cache dữ liệu được đọc hoặc ghi vào một luồng.
- Trong khi thực hiện vùng đệm nhập:
 - Số lượng byte lớn được đọc cùng thời điểm và lưu trữ trong một vùng đệm nhập.
 - Khi chương trình đọc luồng nhập, các byte nhập được đọc vào vùng đệm nhập.



Vùng đệm nhập/xuất (tt...)

- Trong trường hợp vùng đệm kết xuất, một chương trình ghi ra một luồng.
- Dữ liệu kết xuất được lưu trữ trong một vùng đệm kết xuất.
- Dữ liệu được lưu trữ cho đến khi vùng đệm trỏ nên đầy, hay luồng kết xuất được xả trống.
- Kết thúc, vùng đệm kết xuất được chuyển gửi đến đích của luồng xuất.



Vùng đệm nhập/xuất

Các phương thức tạo dựng luồng đệm :

public BufferInputStream(InputStream)

public BufferInputStream (InputStream in, int bufferSize)

public BufferOutputStream (OutputStream out)

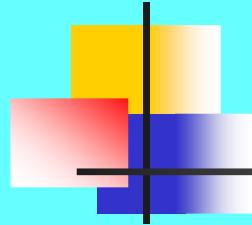
public BufferOutputStream (OutputStream out, int bufferSize)

public BufferedReader (Reader in)

public BufferedReader (Reader in, int bufferSize)

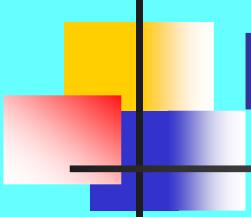
public BufferedWriter (Writer out)

public BufferedWriter (Writer out, int bufferSize)



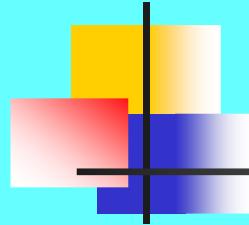
Lớp BufferedInputStream

- Tự động tạo ra và duy trì vùng đệm để hỗ trợ vùng đệm nhập.
- Lớp ‘BufferedInputStream’ là một bộ đệm, nó có thể áp dụng cho một số các đối tượng nhất định của lớp ‘InputStream’.
- Cũng có thể phối hợp các tập tin đầu vào khác.
- Sử dụng vài biến để triển khai vùng đệm nhập.



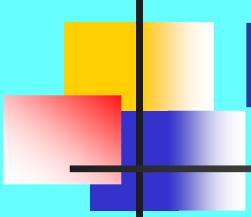
Lớp BufferedInputStream (tt...)

- Định nghĩa hai phương thức thiết lập:
 - Một cho phép chỉ định kích thước của vùng đệm nhập.
 - Phương thức kia thì không.
- Cả hai phương thức thiết lập đều tiếp nhận một đối tượng của lớp 'InputStream' như một tham số.
- Nạp chồng các phương thức truy cập mà InputStream cung cấp, và không đưa vào bất kỳ phương thức mới nào.



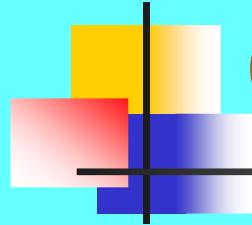
Lớp BufferedOutputStream

- Thực hiện vùng đệm kết xuất theo cách tương ứng với lớp ‘BufferedInputStream’.
- Định nghĩa hai phương thức thiết lập. Nó cho phép chúng ta ấn định kích thước của vùng đệm xuất trong một phương thức thiết lập, cũng giống như cung cấp kích thước vùng đệm mặc định.
- Nạp chòng tắt cả phương thức của lớp ‘OutputStream’ và không đưa vào bất kỳ phương thức nào.



Lớp PrintWriter

- Thực hiện một kết xuất.
- Lớp này có ~~phương~~ phương thức bổ sung , trợ giúp in các kiểu dữ liệu cơ bản .
- Lớp PrintWriter thay thế lớp ‘PrintStream’
- Thực tế cải thiện lớp ‘PrintStream’; lớp này dùng một dấu tách dòng phụ thuộc nền tảng điểm các dòng thay vì ký tự ‘\n’.
- Cung cấp phần hỗ trợ cho các ký tự unicode so với PrintStream.
- Các phương thức:
 - **checkError()**
 - **setError()**

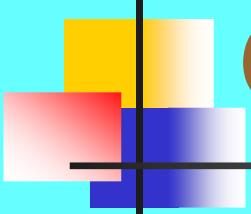


Giao diện DataInput

- Được sử dụng để đọc các byte từ luồng nhị phân
- Cho phép chúng ta chuyển đổi dữ liệu từ từ khuôn dạng UTF-8 được sửa đổi Java đến dạng chuỗi.
- Định nghĩa số phương thức, bao gồm các phương thức để đọc các kiểu dữ liệu nguyên thuỷ.

Giao diện DataInput

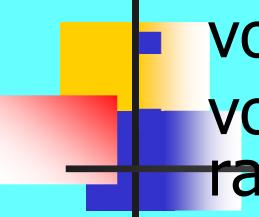
- boolean readBoolean() : Đọc giá trị logic từ luồng
- byte readByte(): Đọc một byte từ luồng
- char readChar() : Đọc một kí tự từ luồng
- double readDouble() : Đọc một số double từ luồng
- float readFloat() : Đọc một số thực từ luồng
- void readFully(byte []b) : Đọc một mảng byte từ luồng và ghi vào mảng
- void readFully(byte []b, int off, int len) : Đọc len byte từ luồng và ghi vào mảng từ vị trí off
- int readInt() : Đọc một số nguyên
- String readLine() : Đọc một xâu kí tự cho đến khi gặp kí tự xuống dòng và bỏ qua kí tự xuống dòng
- long readLong() : Đọc một số long
- short readShort() : Đọc một số short
- int readUnsignedByte() : Đọc một số nguyên không dấu (0..255)
- int readUnsignedShort() : Đọc số nguyên không dấu (0..65535)
- String readUTF() : Đọc một xâu kí tự Unicode
- int skipBytes(int n) : Bỏ qua n byte từ luồng



Giao diện DataOutput

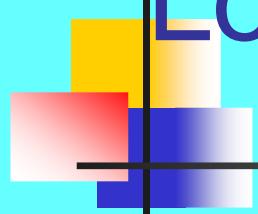
- Được sử dụng để xây dựng lại dữ liệu một số kiểu dữ liệu nguyên thuỷ vào trong dãy các byte
- Ghi các byte dữ liệu vào luồng nhị phân
- Cho phép chúng ta chuyển đổi một chuỗi vào khuôn dạng UTF-8 được sửa đổi Java và viết nó vào trong một dãy.
- Định nghĩa một số phương thức và tất cả phương thức kích hoạt IOException trong trường hợp lỗi.

Giao diện DataOutput



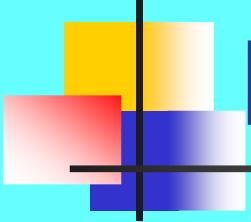
void write(byte[] b) : Ghi một mảng byte ra luồng
void write(byte []b, int off, int len) : Ghi một mảng byte ra luồng tại vị trí off, len byte

- void write(int b) : Ghi một byte ra luồng
- void writeBoolean(boolean b) : Ghi một giá trị logic ra luồng
- void writeByte(int b) : Ghi ra luồng phần thấp của b
- void writeBytes(String s) : Ghi một xâu ra luồng
- void writeChar(int b) : Ghi một kí tự ra luồng
- void writeChars(String s) : Ghi một xâu kí tự ra luồng
- void writeDouble(double b) : Ghi một số double ra luồng
- void writeFloat(float b) : Ghi một số thực ra luồng
- void writeInt(int b) : Ghi một số nguyên ra luồng
- void writeLong(long b) : Ghi một số long ra luồng
- void writeShort(int b) : Ghi một số short ra luồng
- void writeUTF(String s) : Ghi một xâu kí tự Unicode ra



Lớp RandomAccessFile

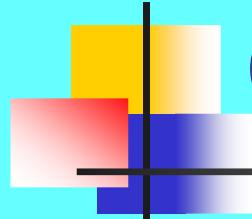
- Cung cấp khả năng thực hiện I/O theo các vị trí cụ thể bên trong một tập tin.
- dữ liệu có thể đọc hoặc ghi ngẫu nhiên ở những vị trí bên trong tập tin thay vì một kho lưu trữ thông tin liên tục.
- phương thức ‘seek()’ hỗ trợ truy cập ngẫu nhiên.
- Thực hiện cả đọc và ghi dữ liệu.
- Hỗ trợ các cấp phép đọc và ghi tập tin cơ bản.
- Kế thừa các phương thức từ các lớp ‘DataInput’ và ‘DataOutput’



Các phương thức của lớp RandomAccessFile

- **seek()**
- **getFilePointer()**
- **length()**

Confidential



Gói java.awt.print

- Gồm có các interface

- Pageable:

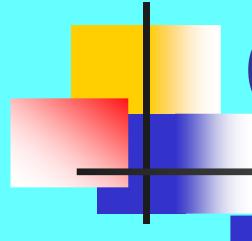
- Định nghĩa các phương thức dùng để các đối tượng biểu thị các trang sẽ được in.
 - Chỉ định số trang đã được in, và trang hiện tại hay là trang giới trang đã được in

- Printable:

- Chi định phương thức ‘print()’ sử dụng để in một trang trên đối tượng ‘Graphics’

- PrinterGraphics:

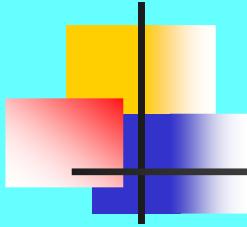
- Cung cấp khả năng truy cập đối tượng ‘PrinterJob’



Gói java.awt.print

- interface ‘PrinterGraphics’ cung cấp các lớp sau:
 - Paper
 - Book
 - PageFormat
 - PrinterJob
- Gói ‘java.awt.print’ kích hoạt các ngoại lệ:
 - PrinterException
 - PrinterIOException
 - PrinterAbortException

Confidential



KẾT THÚC BÀI HỌC

Confidential