# LỜI CẢM ƠN

Trong thời đại công nghệ thông tin ngày nay, việc ứng dụng Công nghệ thông tin vào cuộc sống cũng như các ngành khoa học ngày càng trở nên quan trọng. Quá trình làm đề tài thực tập chuyên ngành này là bước đầu tiên đi vào thực tiễn và cũng chính là bước đầu thực hành và đúc rút chứng minh cho những môn học trên ghế nhà trường nói chung và môn học chuyên nghành nói riêng. Nó cũng chính là quá trình nhận xét đánh giá và rút ra ưu, nhược điểm, để từ đó tìm ra phương án tối ưu nhất cho việc học và đi ra thực tế sau này của bản thân.

Em xin chân thành cảm ơn sự hướng dẫn của các thầy giáo, cô giáo trong Khoa Công nghệ thông tin – Trường Đại học Vinh. Em đặc biệt bày tỏ lòng biết ơn thầy giáo **TS.Nguyễn Quang Ninh** đã tạo mọi điều kiện và luôn giúp đỡ, hướng dẫn em tận tình để em hoàn thành đề tài thực tập chuyên ngành này. Em chân thành cảm ơn các thầy cô luôn sẵn sàng giúp đỡ và tạo mọi điều kiện tốt nhất cho em. Cảm ơn gia đình, bạn bè luôn quan tâm động viên giúp đỡ để em có được như ngày hôm nay.

Mặc dù có nhiều cố gắng bằng toàn bộ kiến thức để hoàn thành công việc, song thời gian và kinh nghiệm của bản thân chưa được trau dồi nhiều nên việc trình bày, phân tích, xây dựng chương trình còn nhiều thiếu sót cần được bổ sung. Vì vậy em rất mong nhận được ý kiến đóng góp của thầy cô để sản phẩm này có thể hoàn thiện, được ứng dụng vào thực tiễn.

Sinh viên thực hiện

**Phạm Ngọc Hiếu**

[1.GIỚI THIỆU 3](#_Toc419072220)

[1.1 Đơn giản 3](#_Toc419072221)

[1.2 Hướng đối tượng 3](#_Toc419072222)

[1.3 Độc lập phần cứng và hệ điều hành 4](#_Toc419072223)

[1.4 Mạnh mẽ 5](#_Toc419072224)

[1.5 Bảo mật 5](#_Toc419072225)

[1.6 Phân tán 6](#_Toc419072226)

[1.7 Đa luồng 6](#_Toc419072227)

[1.8 Linh động 6](#_Toc419072228)

[2.LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA 6](#_Toc419072229)

[2.1 Trừu tượng hoá (Abstraction) 7](#_Toc419072230)

[2.2 Tính kế thừa (Inheritance) 8](#_Toc419072231)

[2.3 Tính đa hình (Polymorphism) 8](#_Toc419072232)

[3.ĐỐI TƯỢNG ,LỚP 10](#_Toc419072233)

[3.1 Khai báo đối tượng 10](#_Toc419072234)

[3.2 Cách truy xuất thành phần của lớp 11](#_Toc419072235)

[3.4 Phạm vi truy xuất thành phần của lớp 13](#_Toc419072236)

[3.5 Phương thức main() 13](#_Toc419072237)

[3.6 Hàm khởi tạo (Constructor) 14](#_Toc419072238)

[3.7 Hàm hủy 16](#_Toc419072239)

[3.8 Từ khoá this 16](#_Toc419072240)

[3.10 Nạp chồng hàm (Overloaded Methods) 17](#_Toc419072241)

[3.11 Truyền tham đối 19](#_Toc419072242)

[3.12 Khai báo kế thừa 20](#_Toc419072243)

[3.14 Từ khoá super 21](#_Toc419072244)

[3.15 Sử dụng từ khoá final cấm sự chồng lắp 24](#_Toc419072245)

[3.16 Sử dụng từ khoá final cấm sự kế thừa 24](#_Toc419072246)

[4.SỬ DỤNG JAVA DEMO THƯ VIỆN BIỂU THỨC CHÍNH QUY 24](#_Toc419072247)

[4.1 Tổng quan 24](#_Toc419072248)

[4.2 Hỗ trợ các ngôn ngữ 24](#_Toc419072249)

[4.3 Các ký tự đặc biệt trong Java Regex (Special characters) 26](#_Toc419072250)

[4.4 Sử dụng String.matches(String) 26](#_Toc419072251)

[4.5 Sử dụng Pattern và Matcher 31](#_Toc419072252)

[4.6 Nhóm (Group) 34](#_Toc419072253)

[4.7 Sử dụng Pattern, Matcher, Group và \*? 36](#_Toc419072254)

[4.8 Demo chương trình sử dụng biểu thức chính quy quy định cách đặt tên biến và khai báo biến trong ngôn ngữ lập trình C 37](#_Toc419072255)

[5.SỬ DỤNG JAVA DEMO CÀI ĐẶT MÃ HÓA DES 40](#_Toc419072256)

[4.1. Sơ đồ khối 45](#_Toc419072257)

[4.5. Thuật toán 47](#_Toc419072258)

[5.6. Lập mã DES 60](#_Toc419072259)

[5.7 Chương trình DEMO 66](#_Toc419072260)

[6.SỬ DỤNG JAVA DEMO CÀI ĐẶT THUẬT TOÁN TÌM MỌI KHÓA 71](#_Toc419072261)

[6.1 Một vài khái niệm ban đầu 71](#_Toc419072262)

[6.2 Thuật toán 72](#_Toc419072263)

[6.3 Chương trình DEMO 73](#_Toc419072264)

[7.SỬ DỤNG JAVA DEMO PHẦN MỀM QUẢN LÝ NHÂN SỰ 76](#_Toc419072265)

[7.1 Mục đích 76](#_Toc419072266)

[7.2 Các chức năng chính của chương trình 76](#_Toc419072267)

# 1.GIỚI THIỆU

Java là một ngôn ngữ lập trình được Sun Microsystems giới thiệu vào tháng 6 năm 1995. Từ đó, nó đã trở thành một công cụ lập trình của các lập trình viên chuyên nghiệp. Java được xây dựng trên nền tảng của C và C++, do vậy nó sử dụng các cú pháp của C và các đặc trưng hướng đối tượng của C++.

Vào năm 1991, một nhóm các kỹ sư của Sun Microsystems có ý định thiết kế một ngôn ngữ lập trình để điều khiển các thiết bị điện tử như tivi, máy giặt, lò nướng,… Mặc dù C và C++ có khả năng làm việc này nhưng trình biên dịch lại phụ thuộc vào từng loại CPU.

Trình biên dịch thường phải tốn nhiều thời gian để xây dựng nên rất đắt, vì vậy để mỗi loại CPU có một trình biên dịch riêng là rất tốn kém. Do đó nhu cầu thực tế đòi hỏi một ngôn ngữ chạy nhanh, gọn, hiệu quả và độc lập thiết bị tức là có thể chạy trên nhiều loại CPU khác nhau, dưới các môi trường khác nhau. “Oak” đã ra đời và vào năm 1995 được đổi tên thành Java. Mặc dù mục tiêu ban đầu không phải cho Internet nhưng do đặc trưng không phụ thuộc thiết bị nên Java đã trở thành ngôn ngữ lập trình cho Internet.

Java là ngôn ngữ lập trình được phát triển từ ngôn ngữ lập trình C/C++. Nó kế thừa, phát huy các thế mạnh của ngôn ngữ C/C++ và lược bỏ đi các cú pháp phức tạp của C/C++. Ngôn ngữ lập trình Java có một số đặc trưng tiêu biểu: đơn giản, hướng đối tượng, độc lập phần cứng và hệ điều hành, mạnh mẽ, bảo mật, phân tán, đa luồng và linh động.

## 1.1 Đơn giản

Những người thiết kế mong muốn phát triển một ngôn ngữ dễ học và quen thuộc với đa số người lập trình. Do vậy Java loại bỏ các đặc trưng phức tạp của C và C++ như:

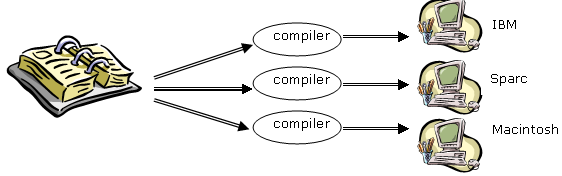
* Loại bỏ thao tác con trỏ, thao tác định nghĩa chồng toán tử
* Không cho phép đa kế thừa mà sử dụng các giao diện
* Không sử dụng lệnh “goto” cũng như file header (.h)
* Loại bỏ cấu trúc “struct” và “union”

## 1.2 Hướng đối tượng

Java là ngôn ngữ lập trình thuần hướng đối tượng, mọi chương trình viết trên Java đều phải được xây dựng trên các đối tượng. Nếu trong C/C++ ta có thể tạo ra các hàm (chương trình con không gắn với đối tượng nào) thì trong Java ta chỉ có thể tạo ra các phương thức (chương trình con gắn liền với một lớp cụ thể). Trong Java không cho phép các đối tượng có tính năng đa kế thừa mà được thay thế bằng các giao diện (interface)

## 1.3 Độc lập phần cứng và hệ điều hành

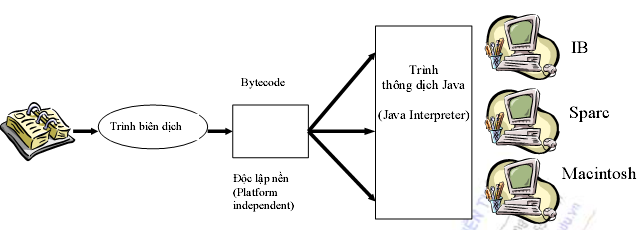
Đối với các ngôn ngữ lập trình truyền thống như C/C++, phương pháp biên dịch được thực hiện như sau :



Hình 1.1: Các biên dịch chương trình hệ thống

Với mỗi nền phần cứng khác nhau, có một trình biên dịch khác nhau để biên dịch mã nguồn chương trình cho phù hợp với nền phần cứng ấy. Do vậy, khi chạy trên một nền phần cứng khác bắt buộc phải biên dịch lại mã nguồn.

Đối với các chương trình viết bằng Java, trình biên dịch Javac sẽ biên dịch mã nguồn thành dạng bytecode. Sau đó, khi chạy chương trình trên các nền phần cứng khác nhau, máy ảo Java dùng trình thông dịch Java để chuyển mã bytecode thành dạng chạy được trên các nền phần cứng tương ứng. Do vậy, khi thay đổi nền phần cứng, không phải biên dịch lại mã nguồn Java.



Hình 1.2: Biên dịch hệ thống java

## 1.4 Mạnh mẽ

Java là ngôn ngữ yêu cầu chặt chẽ về kiểu dữ liệu.

* Kiểu dữ liệu phải khai báo tường minh.
* Java không sử dụng con trỏ và các phép toán con trỏ.
* Java kiểm tra tất cả các truy nhập đến mảng, chuỗi khi thực thi để đảm bảo rằng các truy nhập đó không ra ngoài giới hạn kích thước
* Trong các môi trường lập trình truyền thống, lập trình viên phải tự mình cấp phát bộ nhớ, trước khi chương trình kết thúc thì phải tự giải phóng bộ nhớ đã cấp. Vấn đề có thể nảy sinh khi lập trình viên quên giải phóng bộ nhớ đã xin cấp trước đó. Trong chương trình Java, lập trình viên không phải bận tâm đến việc cấp phát bộ nhớ. Quá trình cấp phát, giải phóng được thực hiện tự động, nhờ dịch vụ thu nhặt những đối tượng không còn sử dụng nữa (garbage collection).
* Cơ chế bẫy lỗi của Java giúp đơn giản hóa qúa trình xử lý lỗi và hồi phục sau lỗi.

## 1.5 Bảo mật

Java cung cấp một môi trường quản lý thực thi chương trình với nhiều mức để kiểm soát tính an toàn:

* Ở mức thứ nhất, dữ liệu và các phương thức được đóng gói bên trong lớp. Chúng chỉ được truy xuất thông qua các giao diện mà lớp cung cấp.
* Ở mức thứ hai, trình biên dịch kiểm soát để đảm bảo mã là an toàn, và tuân theo các nguyên tắc của Java.
* Mức thứ ba được đảm bảo bởi trình thông dịch; chúng kiểm soát xem bytecode có đảm bảo các quy tắc an toàn trước khi thực thi không.
* Mức thứ tư kiểm soát việc nạp các lớp vào bộ nhớ để giám sát việc vi phạm giới hạn truy xuất trước khi nạp vào hệ thống.

## 1.6 Phân tán

Java được thiết kế để hỗ trợ các ứng dụng chạy trên mạng bằng các lớp mạng (java.net). Hơn nữa, Java hỗ trợ nhiều nền chạy khác nhau nên chúng được sử dụng rộng rãi như là công cụ phát triển trên Internet - nơi sử dụng nhiều nền khác nhau.

## 1.7 Đa luồng

Chương trình Java cung cấp giải pháp đa luồng (Multithreading) để thực thi các công việc đồng thời. Chúng cũng cung cấp giải pháp đồng bộ giữa các luồng. Đặc tính hỗ trợ đa luồng này cho phép xây dựng các ứng dụng trên mạng chạy hiệu quả.

## 1.8 Linh động

Java được thiết kế như một ngôn ngữ động để đáp ứng cho những môi trường mở. Các chương trình Java chứa rất nhiều thông tin thực thi nhằm kiểm soát và truy nhập đối tượng lúc chạỵ. Điều này cho phép khả năng liên kết mã động.

# 

# 2.LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG TRONG JAVA

Java là một ngôn ngữ lập trình hướng đối tượng. Nếu bạn chưa bao giờ dùng một ngôn ngữ lập trình hướng đối tượng trước đây, bạn cần phải hiểu các khái niệm sau : lập trình hướng đối tượng (Object Oriented Programming)là gì ? đối tượng (Object), lớp (class) là gì, mối quan hệ giữa đối tượng và lớp, gởi thông điệp (Messages) đến các đối tượng là gì ? Mỗi một chương trình máy tính đều gồm có 2 phần : phần mã lệnh và phần dữ liệu. Một số chương trình đặt trọng tâm ở phần mã lệnh, số khác đặt trọng tâm ở phần dữ liệu. Từ đó dẫn đến 2 mô hình quyết định nên cấu trúc của chương trình : một trả lời cho câu hỏi “Điều gì đang xảy ra”, và một cho “Cái gì đang chịu tác động”. Mô hình 1 gọi là mô hình hướng xử lý, nó mô tả như là một chương trình bao gồm một chuỗi các bước thực hiện (mã lệnh). Nhưng khi chương trình càng ngày càng lớn và phức tạp thì khó khăn để sử dụng mô hình thứ nhất.

Vì vậy mô hình thứ 2 được đưa ra, đó là mô hình hướng đối tượng. Chương trình của bạn sẽ xây dựng dựa vào dữ liệu và phần giao diện được định nghĩa cho phần dữ liệu đó. Mô hình này được mô tả như là dữ liệu điều khiển truy xuất đối với mã lệnh.

Ngôn ngữ lập trình hướng đối tượng có các khả năng sau :

- Mô phỏng thế giới thực một cách tự nhiên bởi các đối tượng và mối quan hệ giữa chúng, thuận tiện cho việc thiết kế hệ thống phức tạp

- Thừa kế mã có sẵn một cách dễ dàng, giúp tiết kiệm công sức và nâng cao năng suất của người lập trình, dễ bảo trì, dễ nâng cấp, mở rộng

## 2.1 Trừu tượng hoá (Abstraction)

Con người đã đơn giản hoá các vấn đề phức tạp thông qua sự trừu tượng hoá. Ví dụ, người sử dụng máy tính không nhìn máy tính một cách phức tạp. Nhờ sự trừu tượng hoá mà người ta có thể sử dụng máy tính mà không quan tâm đến cấu trúc chi tiết bên trong máy tính. Họ chỉ sử dụng chúng như là một thực thể

Cách tốt nhất để nắm vững kỹ thuật trừu tượng là dùng hệ thống phân cấp. Điều này cho phép bạn phân lớp các thành phần có ý nghĩa của cả hệ thống phức tạp, chia nhỏ chúng thành những phần đơn giản có thể quản lý được. Nhìn bên ngoài máy tính là một đối tượng, nếu nhìn sâu hơn một cấp, máy tính bao gồm một số bộ phận : hộp điều khiển, màn hình, bàn phím, chuột..., các bộ phận này lại bao gồm các bộ phận nhỏ hơn, ví dụ như hộp điều khiển có bảng mạch chính chứa CPU, các mạch giao tiếp gắn trên bảng mạch chính, đĩa cứng, ổ đĩa mềm… Nhờ sự trừu tượng hoá mà bạn không quan tâm đến chi tiết từng bảng mạch, mà chỉ quan tâm mối quan hệ, giao tiếp giữa các bộ phận. Một mạch giao tiếp dù có chức năng ly kỳ thế nào đi nữa, bạn có thể sử dụng không mấy khó khăn nếu được ấn vừa vặn vào khe cắm trên bảng mạch chính.

Sự phân cấp trừu tượng một hệ thống phức tạp có thể áp dụng cho các chương trình máy tính. Phần dữ liệu từ một chương trình hướng xử lý kinh điển có thể trừu tượng hoá thành các đối tượng thành phần. Dãy các xử lý trở thành các thông điệp giữa các đối tượng. Vì thế các đối tượng cần có hoạt động đặc trưng riêng. Bạn có thể coi các đối tượng này như những thực thể độc lập tiếp nhận các yêu cầu từ bên ngoài. Đây là phần cốt lõi của lập trình hướng đối tượng.Tất cả các ngôn ngữ lập trình hướng đối tượng đều có các cơ chế cho phép bạn triển khai các mô hình hướng đối tượng. Đó là tính đóng gói, kế thừa, và tính đa hình.2.2 Tính đóng gói (Encapsulation)Đây là cơ chế dùng một vỏ bọc kết hợp phần dữ liệu và các thao tác trên dữ liệu đó (phần mã lệnh) thành một thể thống nhất, tạo nên sự an toàn, tránh việc sử dụng không đúng thiết kế, bảo vệ cho mã lệnh và dữ liệu chống việc truy xuất từ những đoạn mã lệnh bên ngoài.Trong Java tính đóng gói thể hiện qua khái niệm lớp (Class). Lớp là hạt nhân của Java, tạo nền tảng cho lập trình hướng đối tượng trong Java. Nó định nghĩa dữ liệu và các hành vi của nó (dữ liệu và mã lệnh), gọi là các thành viên của lớp, dùng chung cho các đối tượng cùng loại. Từ sự phân tích hệ thống, người ta trừu tượng nên các lớp. Sau đó các đối tượng được tạo ra theo khuôn mẫu của lớp. Mỗi đối tượng thuộc một lớp có dữ liệu và hành vi định nghĩa cho lớp đó, giống như là sinh ra từ một khuôn đúc của lớp đó. Vì vậy mà lớp là khuôn mẫu của đối tượng, đối tượng là thể hiện của một lớp. Lớp là cấu trúc logic, còn đối tượng là cấu trúc vật lý. Dữ liệu định nghĩa trong lớp gọi là biến, mã lệnh gọi là phương thức. Phương thức định nghĩa cho việc sử dụng dữ liệu như thế nào. Điều này có nghĩa là hoạt động của lớp được định nghĩa thông qua phương thức.

Các đặc trưng của lớp gồm có hai phần chính : thuộc tính (Attribute) và hành vi (Behavior). Giả sử bạn phải tạo ra giao diện với người dùng và cần có những nút nhấn (Button). Thế thì trước hết bạn xây dựng lớp Button với các thuộc tính như nhãn ghi trên nút, chiều rộng, chiều cao, màu của nút, đồng thời quy định hành vi của nút nhấn, nghĩa là nút nhấn cần phản ứng như thế nào khi được chọn, phát yêu cầu gì, có đổi màu hay nhấp nháy chi không. Với lớp Button như vậy, bạn có thể tạo ra nhanh chóng những nút nhấn cụ thể phục vụ cho các mục đích khác nhauGói là kỹ thuật của Java, dùng để phân hoạch không gian tên lớp, giao diện thành những vùng dễ quản lý hơn, thể hiện tính đóng gói của Java.

## 2.2 Tính kế thừa (Inheritance)

Tính kế thừa là khả năng xây dựng các lớp mới từ các lớp đã có. Tính đóng gói cũng tác động đến tính kế thừa. Khi lớp đóng gói một số dữ liệu và phương thức, lớp mới sẽ kế thừa mọi cấu trúc dữ liệu và các phương thức của lớp mà nó kế thừa. Ngoài ra nó có thể bổ sung các dữ liệu và các phương thức của riêng mình.

Nó rất quan trọng vì nó ứng dụng cho khái niệm cây phân cấp (mô hình TopDown). Không sử dụng cây phân lớp, mỗi lớp phải định nghĩa tất cả các dữ liệu và phương thức của mình một cách rõ ràng. Nếu sử dụng sự kế thừa, mỗi lớp chỉ cần định nghĩa thêm những đặc trưng của mình.

*Ví dụ :*  Xe có thể xem như một lớp và các xe Pergout, BWM, Dream là các đối tượng của lớp xe. Các xe đều có thể lái đi, dừng lại... Từ lớp xe ở trên, ta có thể xây dựng các lớp xe đạp, xe ôtô. Xe ôtô có thêm máy và có thể tự khởi động…

## 2.3 Tính đa hình (Polymorphism)

Khi một lớp được kế thừa từ các lớp tổ tiên thì nó có thể thay đổi cách thức làm việc của lớp tổ tiên trong một số phương thức nào đó (nhưng tên, kiểu trả về, danh sách tham đối của phương thức thì vẫn giữ nguyên). Điều này gọi là viết chồng. Như vậy với một tên phương thức, chương trình có thể có các hành động khác nhau tùy thuộc vào lớp của đối tượng gọi phương thức. Đó là tính đa hình

*Ví dụ :* với một phương thức chạy, xe ôtô, xe máy có thể tăng ga, còn xe đạp thì phải đạp…

Tính đa hình còn thể hiện ở việc một giao diện có thể sử dụng cho các hoạt động của một lớp tổng quát, hay còn gọi là “một giao diện, nhiều phương thức”. Có nghĩa là có thể thiết kế một giao diện tổng quát cho một nhóm các hành vi liên quan. Điều này giảm thiểu sự phức tạp bằng cách cho phép một giao diện có thể sử dụng cho các hoạt động của một lớp tổng quát. Trình biên dịch sẽ xác định hoạt động cụ thể nào sẽ được thi hành tùy theo điều kiện. Bạn chỉ cần nhớ các giao diện của lớp tổng quát và sử dụng nó.

Sự kết hợp đúng đắn giữa : đa hình, đóng gói và kế thừa tạo nên một môi trường lập trình có khả năng phát triển tốt hơn rất nhiều so với môi trường không hỗ trợ hướng đối tượng. Một cây phân cấp lớp thiết kế tốt là điều căn bản cho việc sử dụng lại những đoạn mã lệnh mà bạn đã tốn công sức nhiều cho việc phát triển và kiểm tra. Tính đóng gói cho phép bạn sử dụng các đối tượng và ra lệnh thi hành tới chúng mà không phá vỡ cấu trúc các đoạn mã lệnh đã bảo vệ bởi giao diện của các lớp. Sự đa hình cho phép bạn tạo ra những đoạn mã lệnh gọn gàng, dễ đọc, dễ hiểu và có tính ổn định.Java là ngôn ngữ lập trình hướng đối tượng nên có đầy đủ các tính năng trên, thư viện lớp Java được cung cấp khá đầy đủ cho người lập trình để bắt đầu một dự án mở.

# 3.ĐỐI TƯỢNG ,LỚP

Khi định nghĩa một lớp, bạn chỉ ra thuộc tính mà nó chứa được thể hiện bằng biến (Member Variable) và hành vi được thể hiện bởi hàm (Method)

Các biến định nghĩa bên trong một lớp gọi là các biến thành viên (Member Variables). Mã lệnh chứa trong các phương thức (Method). Các phương thức và biến định nghĩa trong lớp gọi chung là thành phần của lớp. Trong hầu hết các lớp, các biến thể hiện được truy cập bởi các phương thức định nghĩa trong lớp đó. Vì vậy, chính các phương thức quyết định dữ liệu của lớp có thể dùng như thế nào. Lớp định nghĩa một kiểu dữ liệu mới, dùng để tạo các đối tượng thuộc kiểu đó.

Dạng đầy đủ của một định nghĩa lớp như sau :

|  |  |
| --- | --- |
| **[public]** | Lớp được truy xuất chung cho các Package khác, mặc định chỉ có các đoạn mã trong cùng một gói mới có quyền truy xuất nó |
| **[abstract]** | Lớp trừu tượng, không thể khởi tạo |
| **[final]** | Lớp hằng không có lớp con, không kế thừa |
| **class *ClassName*** | Tên lớp |
| **[extends *SuperClass*]** | Kế thừa lớp cha SuperClass |
| **[implements *Interfaces*]** | Giao diện được cài đặt bởi Class |
| **{ //Member Variables Declarations** | Khai báo các biến |
| **// Methods Declarations** | Khai báo các phương thức |
| **}** |  |

## 3.1 Khai báo đối tượng

Để có được các đối tượng của một lớp phải qua hai giai đoạn :

**⬩ *ClassName ObjectName;****Ví dụ :* Box myBox

Khai báo biến myBox có kiểu lớp Box. Khai báo này thực ra không cấp phát ký ức đủ chứa đối tượng thuộc lớp Box, mà chỉ tạo ra quy chiếu trỏ đến đối tượng Box. Sau câu lệnh này, quy chiếu myBox xuất hiện trên ký ức chứa giá trị null chỉ ra rằng nó chưa trỏ đến một đối tượng thực tế nào Khác với câu lệnh khai báo biến kiểu sơ cấp là dành chỗ trên ký ức đủ chứa một trị thuộc kiểu đó :*Ví dụ :* int i;

Sau câu lệnh này, biến nguyên i hình thành. ⬩ Sau đó, để thực sự tạo ra một đối tượng và gán địa chỉ của đối tượng cho biến này, dùng toán tử new

***ObjectName* = new *ClassName();****Ví dụ :* myBox = new Box();

⬩ Có thể kết hợp cả hai bước trên vào một câu lệnh :

***ClassName ObjectName =* new *ClassName();****Ví dụ :* Box myBox = new Box();

Box myBox2 = myBox; myBox2 tham chiếu đến cùng đối tượng mà myBox tham chiếu

width

height

depth

myBox

mybox

Box Object

myBox2

mybox

## 3.2 Cách truy xuất thành phần của lớp

⬩ Biến khai báo trong định nghĩa lớp gồm có hai loại :

- Biến đối tượng (Instance Variable hay Object Variable) : chỉ thuộc tính đối tượng, khi truy xuất phải khởi tạo đối tượng

+ Cách khai báo biến đối tượng :

***Type* *InstanceVar;***

+ Cách truy cập biến đối tượng :

***ObjectName*.*InstanceVar***

- Biến lớp (Class Variable) : về bản chất là biến toàn cục, là biến tĩnh được tạo lập một lần cùng với lớp, dùng chung cho mọi đối tượng thuộc lớp, khi truy xuất không cần khởi tạo đối tượng, để trao đổi thông tin của các đối tượng cùng lớp

+ Cách khai báo biến lớp :

**static *Type* *ClassVar;***

+ Cách truy cập biến lớp :

***ClassName*.*ClassVar***

⬩ Hàm khai báo trong định nghĩa lớp gồm có hai loại :

- Hàm đối tượng (Object Method) : cách truy xuất hàm đối tượng như biến đối tượng

***ObjectName*.*ObjectMethod*(*Parameter-List*)**- Hàm lớp (Class Method) : thông thường một thành phần của lớp chỉ truy xuất trong sự liên kết với một đối tượng thuộc lớp của nó. Tuy nhiên, có thể tạo ra một thành phần mà có thể dùng một độc lập một mình, không cần tham chiếu đến một đối tượng cụ thể, có thể được truy xuất trước khi bất kỳ đối tượng nào của lớp đó được tạo ra, bằng cách đặt trước khai báo của nó từ khoá static. Cách truy xuất hàm lớp :

***ClassName*.*ClassMethod*(*Parameter-List*)**

Các hàm toán học của lớp Math trong Package Java.Lang là hàm lớp nên khi gọi không cần phải khởi tạo đối tượng

*Ví dụ :* double a = Math.sqrt(453.28);3.3 Khai báo phương thức (hàm)

Dạng tổng quát của một phương thức như sau :

|  |  |
| --- | --- |
| **[*acess*]** | **điều khiển truy xuất** |
| **[static]** | **hàm lớp** |
| **[abstract]** | **hàm trừu tượng** |
| **[final]** | **hàm hằng** |
| **[*Type*] *MethodName*(*Parameter-List*) throws *exceptions* {**  // Body of method} |  |

- Type : Kiểu dữ liệu do hàm trả về, có thể là kiểu bất kỳ, kể cả các kiểu lớp do bạn tạo ra. Nếu hàm không trả về giá trị nào, kiểu trả về của nó phải là void.

- Các hàm có kiểu trả về không phải là void sẽ trả về một giá trị cho chương trình gọi nó dùng dạng câu lệnh return như sau : return biểu thức;

Giá trị của biểu thức được tính và trả về cho hàm

- Tất cả thông tin bạn muốn truyền được gởi thông qua tham số nằm trong hai dấu ( ) ngay sau tên hàm. Nếu không có tham số vẫn phải có ( )

Parameter-List : Danh sách tham đối phân cách bởi các dấu phẩy, mỗi tham đối phải được khai báo kiểu, có thể là kiểu bất kỳ, có dạng : *Type Parameter1*, *Type Parameter2* ...

## 3.4 Phạm vi truy xuất thành phần của lớp

Các điều khiển truy xuất của Java là public, private và protected. protected chỉ áp dụng khi có liên quan đến kế thừa sẽ xét đến sau

Khi bổ sung tiền tố cho một thành phần của lớp (biến và hàm) là :- Từ khoá public : chỉ ra rằng thành phần này có thể được truy xuất bởi bất kỳ dòng lệnh nào dù ở trong hay ngoài lớp mà nó khai báo

- private : chỉ có thể được truy xuất trong lớp của nó, mọi đoạn mã nằm ngoài lớp, kể cả những lớp con đều không có quyền truy xuất- Khi không có điều khiển truy xuất nào được dùng, mặc nhiên là public nhưng chỉ trong gói của nó, không thể truy xuất từ bên ngoài gói của nó

## 3.5 Phương thức main()

Khi chạy ứng dụng độc lập, bạn chỉ tên Class muốn chạy, Java tìm gọi hàm main() trước tiên trong Class đó, phương thức main sẽ điều khiển chạy các phương thức khác.

Dạng tổng quát của phương thức main()

**public static void main(String args[]) {**

**// Body of Method**

**}**

- Một chương trình chỉ cần một lớp có phương thức main() gọi là lớp ứng dụng độc lập Primary Class.

- Từ khoá static cho phép hàm main() được gọi khi không cần khởi tạo đối tượng. Vì main() được trình thông dịch của Java gọi trước khi bất kỳ lớp nào được khởi tạo

- Từ khoá void cho biết hàm main() không trả về giá trị

- Từ khoá public chỉ ra rằng hàm này được gọi bởi dòng lệnh bên ngoài lớp khi chương trình khởi động.

- Tham đối String args[ ] khai báo tham số tên args thuộc lớp String, chứa chuỗi ký tự. Tham đối này giữ các tham đối dòng lệnh dùng khi thi hành chương trình. *Ví dụ 1 :*

class ViDu {

public static void main (String args[]) {

for (int i=0; i < args.length; i++) {

System.out.println(“Tham doi thu “+i+”: “+args[i]);

}

}

}

Khi chạy chương trình :

C:\>java ViDu Thu tham doi dong lenh ↵

Tham doi thu 0 : Thu

Tham doi thu 1 : tham ....

C:>java ViDu Thu “tham doi” “dong lenh” ↵

Tham doi thu 0 : Thu

Tham doi thu 1 : tham doi

Tham doi thu 2 : dong lenh

## 3.6 Hàm khởi tạo (Constructor)

Có những thao tác cần thực hiện mỗi khi đối tượng lần đầu tiên được tạo như khởi tạo giá trị cho các biến. Các công việc này có thể làm tự động bằng cách dùng hàm khởi tạo.

Hàm khởi tạo có cùng tên với lớp mà nó thuộc về, chỉ được tự động gọi bởi toán tử new khi đối tượng thuộc lớp được tạo. Hàm khởi tạo không có giá trị trả về, khi định nghĩa hàm có thể ghi void hay không ghi. *Ví dụ :* - kích thước hộp được khởi tạo tự động khi đối tượng được tạo.

class Box {

double width;

double height;

double depth;

double volume() {

return width \* height \* depth;}Box(double w, double h, double d) { width = w; height = h;

depth = d;

}}class BoxDemo {public static void main (String args[ ]) { Box myBox1 = new Box(10,20,15);

Box myBox2 = new Box(3,6,9);double vol;vol = myBox1.volume();System.out.println(“Thể tích là : “+vol);vol = myBox2.volume();System.out.println(“Thể tích là : “+vol);}

}- Khi bạn không định nghĩa tường minh hàm khởi tạo cho một lớp, Java sẽ tạo hàm khởi tạo mặc nhiên cho lớp đó. Vì vậy các chương trình trước đó vẫn làm việc bình thường. Hàm khởi tạo mặc nhiên không có danh sách tham đối, tự động khởi tạo tất cả các biến của đối tượng về trị rỗng theo các quy ước mặc định của Java, trị 0 cho kiểu số, ký tự ‘\0’ cho kiểu ký tự char, trị false cho kiểu boolean, trị null cho các đối tượng

- Hàm khởi tạo cũng có thể được nạp chồng như hàm bình thường ̣(sẽ nói rõ ở phần sau) nghĩa là ta được phép định nghĩa nhiều hàm khởi tạo khác nhau ở danh sách tham đối hay kiểu tham đối

## 3.7 Hàm hủy

Các đối tượng cấp phát động bằng toán tử new, khi không tồn tại tham chiếu nào đến đối tượng, đối tượng đó xem như không còn cần đến nữa và bộ nhớ cho nó có thể được tự động giải phóng bởi bộ thu gom rác (garbage collector). Trình thu gom rác hoạt động trong một tuyến đoạn (Thread) độc lập với chương trình của bạn. Bạn không phải bận tâm gì đối với công việc này. Sau này bạn sẽ hiểu rõ tuyến đoạn là thế nào

Tuy nhiên, Java cũng cho phép ta viết hàm hủy, có thể cũng cần thiết cho những trường hợp nào đó. Hàm hủy trong Java chỉ được gọi bởi trình thu gom rác, do vậy bạn khó đoán trước vào lúc nào hàm hủy sẽ được gọi

Dạng hàm hủy như sau :

**protected void finalize() {**

**// Body of Method**

**}**

## 3.8 Từ khoá this

Nếu biến được định nghĩa trong thân hàm, đó là biến cục bộ chỉ tồn tại khi hàm được gọi. Nếu biến cục bộ như vậy được đặt tên trùng với biến đối tượng hoặc biến lớp, nó sẽ che khuất biến đối tượng hay biến lớp trong thân hàm :

Từ khoá this có thể dùng bên trong bất cứ phương thức nào để tham chiếu đến đối tượng hiện hành, khi biến đối tượng trùng tên với biến cục bộ.

## 3.10 Nạp chồng hàm (Overloaded Methods)

Trong cùng một lớp, Java cho phép bạn định nghĩa nhiều hàm trùng tên với điều kiện các hàm như vậy phải có danh sách tham đối khác nhau, nghĩa là khác nhau về số tham đối hoặc kiểu của các tham đối. Khả năng như vậy gọi là sự nạp chồng hàm. Java chỉ phân biệt hàm này với hàm khác dựa vào số tham đối và kiểu của các tham đối, bất chấp tên hàm và kiểu của kết quả trả về.

*Ví dụ :*

// MyRect.java

import java.awt.Point;

class MyRect {

int x1 = 0;

int y1 = 0;

int x2 = 0;

int y2 = 0;

MyRect buildRect(int x1, int y1, int x2, int y2) {

this.x1 = x1;

this.y1 = y1;

this.x2 = x2;

this.y2 = y2;

return this;

}

MyRect buildRect(Point topLeft, Point bottomRight) {

x1 = topLeft.x;

y1 = topLeft.y;

x2 = bottomRight.x;

y2 = bottomRight.y;

return this;

}

MyRect buildRect(Point topLeft, int w, int h) {

x1 = topLeft.x;

y1 = topLeft.y;

x2 = x1+w;

y2 = y1 + h;

return this;

}

void display() {

System.out.print(“Doi tuong MyRect : <” + x1 + “, “+y1);

System.out.println(“, “+x2+”, “+y2+”>”);

}

}

Thật ra, trong gói awt có sẵn lớp Rectangle chuyên dùng để biểu diễn hình chữ nhật. Lớp MyRect của ta chỉ dùng để minh hoạ cho khái niệm nạp chồng hàm. Trong lớp MyRect có những hàm giúp bạn tạo ra đối tượng MyRect với những yếu tố cho trước khác nhau :

- Cho trước toạ độ góc trên trái x1, y1 và toạ độ góc dưới phải x2, y2

- Cho trước góc trên trái và góc dưới phải của hình chữ nhật dưới dạng đối tượng Point

- Cho trước toạ độ góc trên trái của hình chữ nhật dạng đối tượng Point cùng chiều rộng, chiều cao

Nhờ khả năng nạp chồng hàm, bạn chỉ cần nhớ một tên hàm cho các hàm khác nhau cùng chức năng

Chương trình sử dụng lớp MyRect xây dựng ở trên :

import java.awt.Point;

class UngDung {

public static void main(String args[]) {

MyRect rect = new MyRect();

rect.buildRect(25,25,50,50);

rect.display();

rect.buildRect(new Point(10,10), new Point(20,20));

rect.display();

rect.buildRect(new Point(10,10), 50, 50);

rect.display();

}

}

## 3.11 Truyền tham đối

Java dùng cả hai cách truyền tham đối : truyền bằng giá trị và truyền bằng tham chiếu, tùy vào cái gì được truyền- Khi ta truyền một kiểu sơ cấp cho phương thức, nó sẽ truyền bằng giá trị. Vì vậy những gì xảy ra với tham đối trong phương thức, khi ra khỏi phương thức sẽ hết tác dụng- Khi ta truyền một đối tượng (kiểu phức hợp) cho phương thức, nó sẽ truyền bằng tham chiếu. Vì vậy, thay đổi ở đối tượng bên trong phương thức ảnh hưởng đến đối tượng dùng làm tham đối.

*Ví dụ 1 :*

class ViDu {

void tinhToan(int i, int j) {

i \*= 2;

j /= 2;

}

}

class UngDung {

public static void main(String args) {

ViDu o = new ViDu();

int a = 15, b = 20;

System.out.println(“a và b trước khi gọi : “+a+ “ ”+b);

o.tinhToan(a, b);

System.out.println(“a và b sau khi gọi : “+a+” “+b);

}

}

Kết quả của chương trình :

a và b trước khi gọi : 15 20

a và b sau khi gọi : 15 20

## 3.12 Khai báo kế thừa

Ta có thể sử dụng tính kế thừa tạo lớp tổng quát có những đặc tính chung đại diện cho một tập hợp các đối tượng có cùng mối quan hệ. Sau đó, lớp này có thể được kế thừa bởi một hay nhiều lớp khác và những đặc tính này trở thành những thành những đặc tính của lớp kế thừa

- Lớp được kế thừa gọi là lớp cha (SuperClass : là lớp cha trực tiếp)- Lớp kế thừa gọi là lớp con (SubClass) Lớp con kế thừa tất cả các biến và hàm định nghĩa trong lớp cha

class *ClassName* extends *SuperClass*

{ //Member Variables Declarations, Methods

}

- Mặc dù vậy, lớp con không thể truy xuất các thành phần được khai báo private trong lớp cha- Một biến tham chiếu của lớp cha có thể gán để tham chiếu đến một lớp con bất kỳ dẫn xuất từ lớp cha. Khi một tham chiếu đến một lớp con được gán cho biến tham chiếu kiểu lớp cha, ta chỉ có quyền truy xuất những phần được định nghĩa bởi lớp cha. 3.13 Viết chồng hàm hay che khuất hàm (Overriding Methods)

Trong phân cấp lớp, khi một hàm của lớp con có cùng tên, và giống nhau về số lượng và kiểu tham đối cũng như kiểu trả về với một hàm ở lớp cha, thì hàm ở lớp con được gọi là viết chồng hàm trong lớp cha. Khi đó hàm của lớp con sẽ che khuất hàm thừa kế từ lớp cha

Tuy nhiên lớp con không được viết chồng hàm hằng (có khai báo final) và hàm lớp trong lớp cha.

*Ví dụ :* Tất cả các lớp là hậu duệ của lớp Object. Lớp Object chứa phương thức toString, mà trả về một đối tượng String chứa tên lớp của đối tượng. Hầu hết các lớp con viết chồng phương thức này và in ra một vài điều gì đó có nghĩa cho lớp đó

## 3.14 Từ khoá super

Đôi khi bạn không muốn thực hiện viết chồng một phương thức mà chỉ muốn thêm chức năng vào phương thức. Để làm được điều này, bạn gọi phương thức được viết chồng dùng từ khoá super. Từ khoá super dùng khi lớp con cần tham chiếu lớp cha trực tiếp của nó. Super có hai dạng cú pháp :- Dạng 1 : Hàm khởi tạo lớp cha phải được gọi trước hàm khởi tạo của lớp con. Nếu trong định nghĩa hàm khởi tạo ở lớp con không có câu lệnh gọi hàm khởi tạo lớp cha, trình biên dịch Java sẽ tự động đưa vào câu lệnh gọi hàm khởi tạo mặc định của lớp cha có dạng : *classname*()

Bạn có thể tự thêm lệnh gọi hàm khởi tạo ở lớp cha có dạng như sau :

super(*Parameter-List*)Parameter-List là danh sách các tham đối cần thiết cho hàm khởi tạo của lớp cha. super() phải luôn luôn là phát biểu đầu tiên được thực hiện trong hàm khởi tạo của lớp conVí dụ :

class MyPoint {

int x, y;

MyPoint(int x, int y) {

this.x = x;

this.y = y;

}

void display() {

System.out.print(“x = “+x+”, y = “+y+”\n”);

}

}

class MyPoint2 extends MyPoint {

int z;

String name;

MyPoint2(int x, int y, int z, String name) {

super(x,y); // Khởi tạo 2 biến x, y bằng cách gọi

this.z = z; // hàm dựng của lớp cha

this.name = name;

}

void display() { // Viết chồng hàm kế thừa từ lớp cha

System.out.print(“x = “+x+”, y = “+y+”, z = “+z+” “+”name :”+name+”\n”);

}

}

- Dạng 2 : dùng để hàm lớp con truy xuất hàm kế thừa từ lớp cha :

super.*Member*

Member có thể là phương thức hay biến của đối tượngVí dụ : Viết lại hàm display() trong class MyPoint2, có gọi hàm kế thừa từ lớp cha :

void display() {

super.display();

System.out.print(”, z = “+z+” “+”name :”+name+”\n”);

}

Trong trường hợp chúng ta muốn định nghĩa một lớp cha theo một cấu trúc trừu tượng cho trước mà không cần hiện thực đầy đủ các phương thức. Tức là ta muốn tạo một lớp cha có dạng chung cho tất cả các lớp con và để các lớp con hiện thực chi tiết. Khi đó, bạn muốn chắc chắn lớp con có chồng lắp phương thức. Những phương thức phải được chồng lắp trong lớp con gọi là phương thức trừu tượng, được khai báo abstract và không có phần thân phương thức

abstract [*Type*] *MethodName*(*Parameter-List*) ;

Bất kỳ lớp nào chứa một hay nhiều phương thức trừu tượng cũng phải khai báo trừu tượng, sử dụng từ khoá abstract trước từ khoá class. Không thể khởi tạo đối tượng kiểu lớp trừu tượng, vì lớp trừu tượng không được định nghĩa đầy đủ. Do đó, bạn cũng không thể khai báo hàm khởi tạo. Bất kỳ lớp con nào cũng phải hoặc là viết chồng tất cả các phương thức trừu tượng hoặc chính nó lại được khai báo abstract

*Ví dụ :* Trong các ứng dụng, bạn có thể vẽ đường tròn, hình chữ nhật, đoạn thẳng, đường cong… Mỗi một đối tượng đồ hoạ này đều chứa các thuộc tính (vị trí, nét viền) và hành vi (di chuyển, thay kích thước, vẽ). Bạn có thể khai báo chúng kế thừa lớp Graphic. Tuy nhiên vẽ một đường tròn là hoàn toàn khác với vẽ một hình chữ nhật, nên lớp Graphic được khai báo là lớp trừu tường, chứa các phương thức đã được hiện thực như moveTo, và phương thức trừu tượng như draw

abstract class GraphicObject {

int x, y;

. . .

void moveTo(int newX, int newY) {

. . .

}

abstract void draw();

}

Mỗi một lớp con không trừu tượng của lớp Graphic như Circle, Rectangle sẽ phải cài đặt đầy đủ cho phương thức draw

class Circle extends GraphicObject {

void draw() {

. . .

}

}

class Rectangle extends GraphicObject {

void draw() {

. . .

}

}

## 3.15 Sử dụng từ khoá final cấm sự chồng lắp

Mặc dù chồng lắp phương thức là một trong những đặc điểm mạnh nhất của Java, tuy nhiên trong vài trường hợp bạn muốn cấm điều này. Để cấm một phương thức lớp con viết chồng phương thức ở lớp cha, bạn đưa từ khoá final vào đầu khai báo

Ví dụ : class Box {

double width;double height;double depth;…

final double volume() { return width \* height \* depth;} . . .

}

## 3.16 Sử dụng từ khoá final cấm sự kế thừa

Muốn khai báo một lớp mà không có lớp con kế thừa, bạn sử dụng từ khoá final. Với một lớp final, thì tất cả các phương thức của nó sẽ là final.

Ta không thể khai báo một lớp vừa abstract và final vì một lớp trừu tượng là một lớp chưa hoàn chỉnh và phải có lớp con để hiện thực đầy đủ

*Ví dụ :* final class Box { . . .

}

# 

# 4.SỬ DỤNG JAVA DEMO THƯ VIỆN BIỂU THỨC CHÍNH QUY

## 4.1 Tổng quan

Một biểu thức chính quy (Regular expressions) định nghĩa một khuôn mẫu (pattern) tìm kiếm chuỗi. Nó có thể được sử dụng tìm kiếm, sửa đổi, và thao tác trên văn bản. Khuôn mẫu được định nghĩa bởi biểu thức chính quy có thể khớp một hoặc một vài lần, hoặc không khớp với một văn bản  
Viết tắt của biểu thức chính quy là regex

## 4.2 Hỗ trợ các ngôn ngữ

Biểu thức chính quy (Regular expression) được hỗ trợ bởi hầu hết các ngôn ngữ lập trình, ví dụ, Java, Perl, Groovy, vv Thật không may mỗi ngôn ngữ hỗ trợ biểu thức thông thường hơi khác nhau.

Quy tắc viết biểu thức chính quy

|  |  |  |
| --- | --- | --- |
| No | Regular Expression | Mô tả |
| 1 | . | Khớp (match) với bất kỳ ký tự nào |
| 2 | ^regex | Biểu thức chính quy phải  khớp tại điểm bắt đầu |
| 3 | regex$ | Biểu thức chính quy phải khớp ở cuối dòng. |
| 4 | [abc] | Thiết lập định nghĩa, có thể khớp với a hoặc b hoặc c. |
| 5 | [abc][vz] | Thiết lập định nghĩa, có thể khớp với a hoặc b hoặc c theo sau là v hay z. |
| 6 | [^abc] | Khi dấu ^ xuất hiện như là nhân vật đầu tiên trong dấu ngoặc vuông, nó phủ nhận mô hình. Điều này có thể khớp với bất kỳ ký tự nào ngoại trừ a hoặc b hoặc c. |
| 7 | [a-d1-7] | Phạm vi: phù hợp với một chuỗi giữa a và điểm d và con số từ 1 đến 7. |
| 8 | X|Z | Tìm X hoặc Z. |
| 9 | XZ | Tìm X và theo sau là Z. |
| 10 | $ | Kiểm tra kết thúc dòng. |
|  | | |
| 11 | \d | Số bất kỳ, viết ngắn gọn cho [0-9] |
| 12 | \D | Ký tự không phải là số, viết ngắn gon cho [^0-9] |
| 13 | \s | Ký tự khoảng trắng, viết ngắn gọn cho [ \t\n\x0b\r\f] |
| 14 | \S | Ký tự không phải khoản trắng, viết ngắn gọn cho [^\s] |
| 15 | \w | Ký tự chữ, viết ngắn gọn cho [a-zA-Z\_0-9] |
| 16 | \W | Ký tự không phải chữ, viết ngắn gọn cho [^\w] |
| 17 | \S+ | Một số ký tự không phải khoảng trắng (Một hoặc nhiều) |
| 18 | \b | Ký tự thuộc a-z hoặc A-Z hoặc 0-9 hoặc \_, viết ngắn gọn cho [a-zA-Z0-9\_]. |
|  | | |
| 19 | \* | Xuất hiện 0 hoặc nhiều lần, viết ngắn gọn cho {0,} |
| 20 | + | Xuất hiện 1 hoặc nhiều lần, viết ngắn gọn cho {1,} |
| 21 | ? | Xuất hiện 0 hoặc 1 lần, ? viết ngắn gọn cho {0,1}. |
| 22 | {X} | Xuất hiện X lần, {} |
| 23 | {X,Y} | Xuất hiện trong khoảng X tới Y lần. |
| 24 | \*? | \* có nghĩa là xuất hiện 0 hoặc nhiều lần, thêm ? phía sau nghĩa là tìm kiếm khớp nhỏ nhất. |

## 4.3 Các ký tự đặc biệt trong Java Regex (Special characters)

Một số ký tự đặc biệt trong Java Regex:

\.[{(\*+?^$|

Những ký tự liệt kê ở trên là các ký tự đặc biệt. Trong Java Regex bạn muốn nó hiểu các ký tự đó theo cách thông thường bạn cần thêm dấu \ ở phía trước.  
  
Chẳng hạn ký tự chấm . java regex đang hiểu là một ký tự bất kỳ, nếu bạn muốn nó hiểu là một ký tự chấm thông thường, cần phải có dấu \ phía trước.

[?](http://o7planning.org/web/fe/default/vi/document/13982/huong-dan-su-dung-bieu-thuc-chinh-quy-java-regex)

// Mẫu regex mô tả một ký tự bất kỳ.

String regex = ".";

// Mẫu regex mô tả  ký tự dấu chấm.

String regex = "\\.";

## 4.4 Sử dụng String.matches(String)

Sử dụng  method String.matches(String regex) cho phép bạn kiểm tra toàn bộ String có khớp với regex không. Đây là một cách thông dụng nhất. Hãy xem các ví dụ:

public class StringMatches {

 public static void main(String[] args) {

  String s1 = "a";

     System.out.println("s1=" + s1);

     // Kiểm tra toàn bộ s1

     // Khớp với bất kỳ ký tự nào

     // Quy tắc: .

     // ==> true

     boolean match = s1.matches(".");

     System.out.println("-Match . " + match);

     s1 = "abc";

     System.out.println("s1=" + s1);

     // Kiểm tra toàn bộ s1

     // Khớp với bất kỳ ký tự nào.

     // ==> false  (Rõ ràng, chuỗi 3 ký tự sao khớp với 1 ký tự bất kỳ?)

     match = s1.matches(".");

     System.out.println("-Match . " + match);

     // Kiểm tra toàn bộ s1

     // Khớp với bất kỳ ký tự nào 0 hoặc nhiều lần

     // Kết hợp các quy tắc: . và \*

     // ==> true

     match = s1.matches(".\*");

     System.out.println("-Match .\* " + match);

     String s2 = "m";

     System.out.println("s2=" + s2);

     // Kiểm tra toàn bộ s2

     // Bắt đầu bởi m

     // Quy tắc ^

     // true

     match = s2.matches("^m");

     System.out.println("-Match ^m " + match);

     s2 = "mnp";

     System.out.println("s2=" + s2);

     // Kiểm tra toàn bộ s2

     // Bắt đầu bởi m

     // Quy tắc ^

     // ==> false  (Rõ ràng, chuỗi 3 ký tự sao khớp với 1 ký tự bất kỳ bắt đầu bởi m)

     match = s2.matches("^m");

     System.out.println("-Match ^m " + match);

     // Bắt đầu bởi m

     // Sau đó là ký tự bất kỳ, xuất hiện 1 hoặc nhiều lần.

     // Quy tắc ^ và . và +

     // true

     match = s2.matches("^m.+");

     System.out.println("-Match ^m.+ " + match);

     String s3 = "p";

     System.out.println("s3=" + s3);

     // Kiểm tra s3 kết thúc bằng p

     // Quy tắc $

     // true

     match = s3.matches("p$");

     System.out.println("-Match p$ " + match);

     s3 = "2nnp";

     System.out.println("s3=" + s3);

     // Kiểm tra toàn bộ s3

     // Kết thúc bằng p

     // ==> false  (Rõ ràng, chuỗi 4 ký tự sao khớp với 1 ký tự p cuối cùng)

     match = s3.matches("p$");

     System.out.println("-Match p$ " + match);

     // Kiểm tra toàn bộ s3

     // Ký tự bất kỳ xuất hiện 1 lần: .

     // tiếp theo là n, xuất hiện 1 hoặc tối đa 3 lần.

     // Kết thúc bởi p: p$

     // Kết hợp các quy tắc: . , {X,Y}, $

     // true

     match = s3.matches(".n{1,3}p$");

     System.out.println("-Match .n{1,3}p$ " + match);

     String s4 = "2ybcd";

     System.out.println("s4=" + s4);

     // Bắt đầu là 2

     // Tiếp theo x hoặc y hoặc z

     // Tiếp theo bất kỳ 1 hoặc nhiểu lần.

     // Kết hợp các quy tắc: [abc] , . , +

     // true

     match = s4.matches("2[xyz].+");

     System.out.println("-Match 2[xyz].+ " + match);

     String s5 = "2bkbv";

     // Bắt đầu là bất kỳ, 1 hoặc nhiểu lần

     // Tiếp theo a hoặc b, hoặc c: [abc]

     // Tiếp theo z hoặc v: [zv]

     // Tiếp theo bất kỳ

     // true

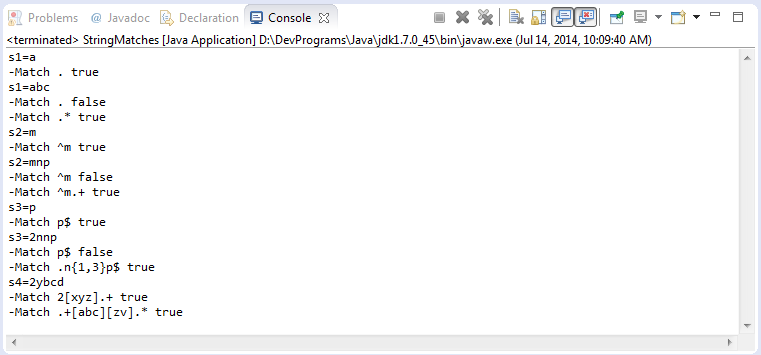
     match = s5.matches(".+[abc][zv].\*");

     System.out.println("-Match .+[abc][zv].\* " + match);

 }

}

Kết quả chạy ví dụ:



public class SplitWithRegex {

   public static final String TEXT = "This is my text";

   public static void main(String[] args) {

       System.out.println("TEXT=" + TEXT);

       // Khoảng trắng xuất hiện 1 hoặc nhiều lần.

       // Các ký tự khoảng trắng: \t\n\x0b\r\f

       // Kết hợp quy tắc: \s và +

       String regex = "\\s+";

       String[] splitString = TEXT.split(regex);

       // 4

       System.out.println(splitString.length);

       for (String string : splitString) {

           System.out.println(string);

       }

       // Thay thế tất cả các khoảng trắng với ký tự tab.

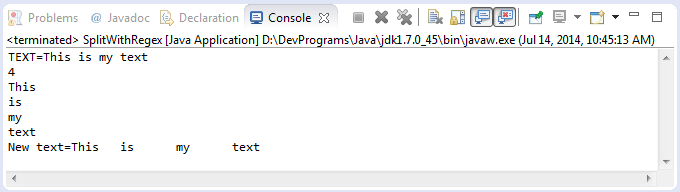
       String newText = TEXT.replaceAll("\\s+", "\t");

       System.out.println("New text=" + newText);

   }

}

Kết quả chạy ví dụ:



public class EitherOrCheck {

   public static void main(String[] args) {

       String s = "The film Tom and Jerry!";

       // Kiểm tra toàn bộ s

       // Bắt đầu bởi ký tự bất kỳ xuất hiện 0 hoặc nhiều lần

       // Tiếp theo là từ Tom hoặc Jerry

       // Kết thúc bởi ký tự bất kỳ xuất hiện 0 hoặc nhiều lần

       // Kết hợp các quy tắc: ., \*, X|Z

       // true

       boolean match = s.matches(".\*(Tom|Jerry).\*");

       System.out.println("s=" + s);

       System.out.println("-Match .\*(Tom|Jerry).\* " + match);

       s = "The cat";

       // false

       match = s.matches(".\*(Tom|Jerry).\*");

       System.out.println("s=" + s);

       System.out.println("-Match .\*(Tom|Jerry).\* " + match);

       s = "The Tom cat";

       // true

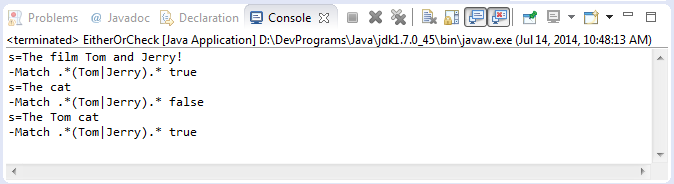
       match = s.matches(".\*(Tom|Jerry).\*");

       System.out.println("s=" + s);

       System.out.println("-Match .\*(Tom|Jerry).\* " + match);

   }

}



## 4.5 Sử dụng Pattern và Matcher

1. Pattern là một đối tượng mẫu, một phiên bản biên dịch của biểu thức chính quy. Nó không có cấu tử public, và chúng ta sẽ sử dụng method tĩnh compile(String) để tạo đối tượng, với tham số là biểu thức chính quy.  
2. Matcher là một phương tiện để khớp với String dữ liệu vào với đối tượng Pattern đã tạo trước đó. Class này không có cấu tử public, và chúng ta lấy đối tượng này thông qua method matcher(String) của đối tượng pattern. Với tham số String là văn bản cần kiểm tra.  
3. PatternSyntaxException sẽ bị ném ra nếu biểu thức chính quy có ngữ pháp không chính xác.

String regex= ".xx.";

// Tạo đối tượng Pattern thông qua method tĩnh.

Pattern pattern = Pattern.compile(regex);

// Lấy ra đối tượng Matcher

Matcher matcher = pattern.matcher("MxxY");

boolean match = matcher.matches();

System.out.println("Match "+ match);

* **Class Patten:**

public static Pattern compile(String regex, int flags) ;

public static Pattern compile(String regex);

public Matcher matcher(CharSequence input);

public static boolean matches(String regex, CharSequence input);

* **Class Matcher:**

public int start()

public int start(int group)

public int end()

public int end(int group)

public String group()

public String group(int group)

public String group(String name)

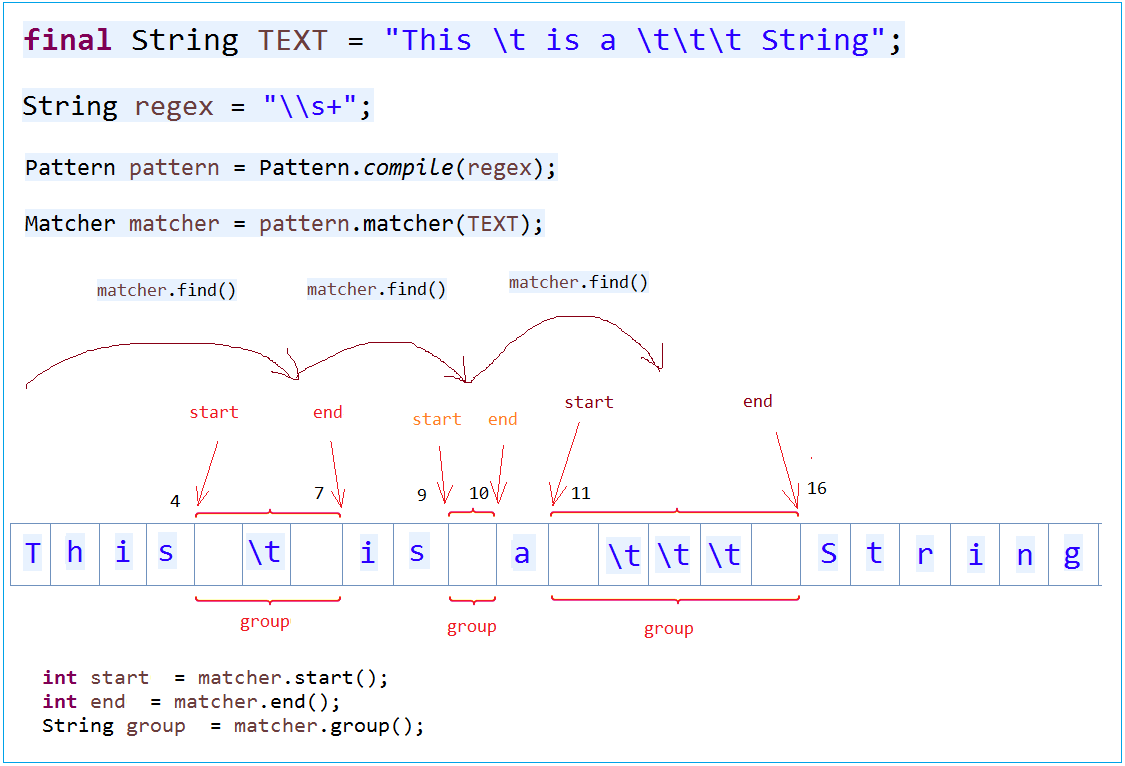
public int groupCount()

public boolean matches()

public boolean lookingAt()

public boolean find()

Đây là một ví dụ sử dụng Matcher và method find() để tìm kiếm các chuỗi con khớp với biểu thức chính quy.



import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class MatcherFind {

   public static void main(String[] args) {

       final String TEXT = "This \t is a \t\t\t String";

       // Khoảng trắng xuất hiện 1 hoặc nhiều lần.

       String regex = "\\s+";

       Pattern pattern = Pattern.compile(regex);

       Matcher matcher = pattern.matcher(TEXT);

       int i = 0;

       while (matcher.find()) {

           System.out.print("start" + i + " = " + matcher.start());

           System.out.print(" end" + i + " = " + matcher.end());

           System.out.println(" group" + i + " = " + matcher.group());

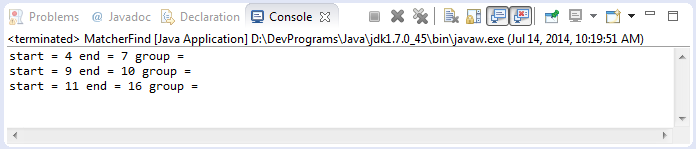
           i++;

       }

   }

}

Kết quả chạy ví dụ:



import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class MatcherLookingAt {

   public static void main(String[] args) {

       String country1 = "iran";

       String country2 = "Iraq";

       // Bắt đầu bởi I tiếp theo là ký tự bất kỳ.

       // Tiếp theo là ký tự a hoặc e.

       String regex = "^I.[ae]";

       Pattern pattern = Pattern.compile(regex, Pattern.CASE\_INSENSITIVE);

       Matcher matcher = pattern.matcher(country1);

       // lookingAt() tìm kiếm khớp phần đầu.

       System.out.println("lookingAt = " + matcher.lookingAt());

       // Trong khi matches() phải khớp toàn bộ

       System.out.println("matches = " + matcher.matches());

       // Reset matcher với text mới, country2

       matcher.reset(country2);

       System.out.println("lookingAt = " + matcher.lookingAt());

       System.out.println("matches = " + matcher.matches());

   }

}

## 4.6 Nhóm (Group)

Một biểu thức chính quy bạn có thể tách ra thành các nhóm (group):

// Một biểu thức chính quy

String regex = "\\s+=\\d+";

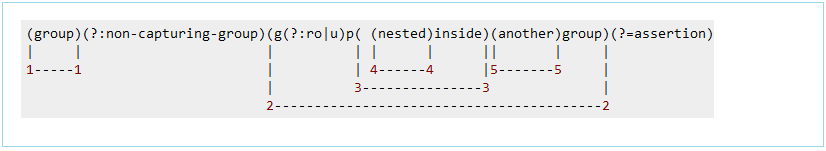
// Viết dưới dạng group, bởi dấu ()

String regex2 = "(\\s+)(=)(\\d+)";

// Một cách khác.

String regex3 = "(\\s+)(=\\d+)";

Các group có thể lồng nhau, và như vậy cần một quy tắc đánh chỉ số các group.  Toàn bộ pattern được định nghĩa là group số 0. Còn lại được mô tả giống hình minh họa dưới đây:



Chú ý: Sử dụng (?:pattern) để thông báo với Java không xem đây là một group (None-capturing group)

Từ Java 7, bạn có thể xác định một group có tên (?<name>pattern), Và bạn có thể truy cập các nội dung khớp với Matcher.group (String name). Điều này làm Regex dài hơn, nhưng mã này là có ý nghĩa hơn, dễ hơn.  
  
Nhóm bắt theo tên cũng có thể được truy cập thông qua Matcher.group (int group) với các đề án đánh số tương tự.  
  
Nội bộ, Java chỉ lập bản đồ từ tên đến số nhóm. Do đó, bạn không thể sử dụng cùng tên để bắt 2 nhóm khác nhau.

Hãy xem một ví dụ sử dụng đánh tên cho nhóm (group) (Java >=7)

package org.o7planning.tutorial.regex;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class NamedGroup {

   public static void main(String[] args) {

       final String TEXT = " int a = 100;float b= 130;float c= 110 ; ";

       // Sử dụng (?<groupName>pattern) để định nghĩa một Group có tên: groupName

       // Định nghĩa group có tên declare: sử dụng (?<declare> ...)

       // Và một group có tên value: sử dụng: (?<value> ..)

       String regex = "(?<declare>\\s\*(int|float)\\s+[a-z]\\s\*)=(?<value>\\s\*\\d+\\s\*);";

       Pattern pattern = Pattern.compile(regex);

       Matcher matcher = pattern.matcher(TEXT);

       while (matcher.find()) {

           String group = matcher.group();

           System.out.println(group);

           System.out.println("declare: " + matcher.group("declare"));

           System.out.println("value: " + matcher.group("value"));

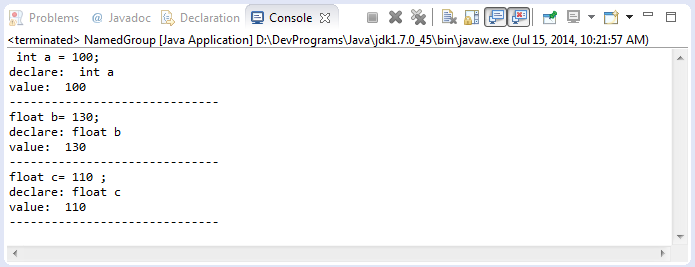
           System.out.println("------------------------------");

       }

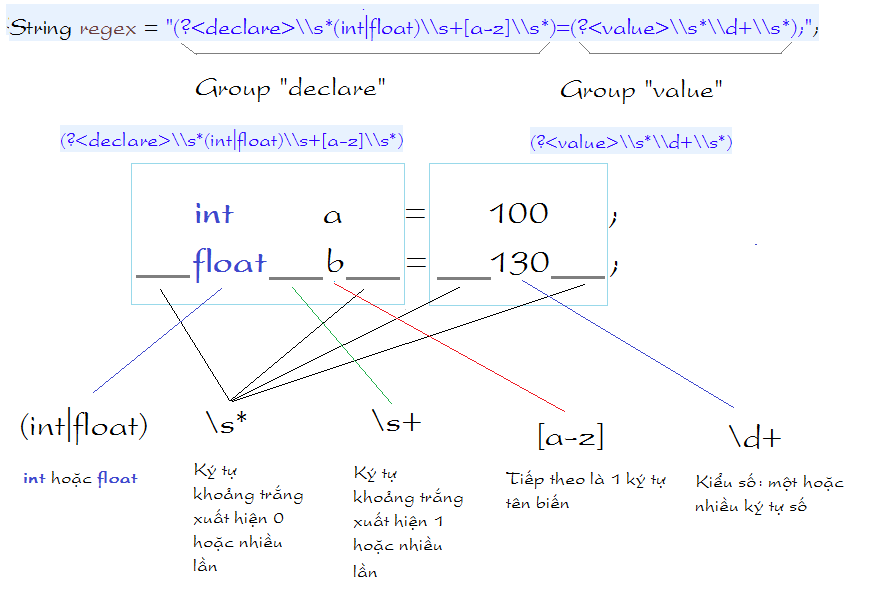
   }

}

Kết quả chạy ví dụ:



Để dễ hiểu bạn có thể xem hình minh họa dưới đây:



## 4.7 Sử dụng Pattern, Matcher, Group và \*?

Trong một số tình huống \*? rất quan trọng, hãy xem một ví dụ sau:

// Đây là một regex

// Bắt gặp ký tự bất kỳ 0 hoặc nhiều lần,

// sau đó tới ký tự ' và tiếp theo là >

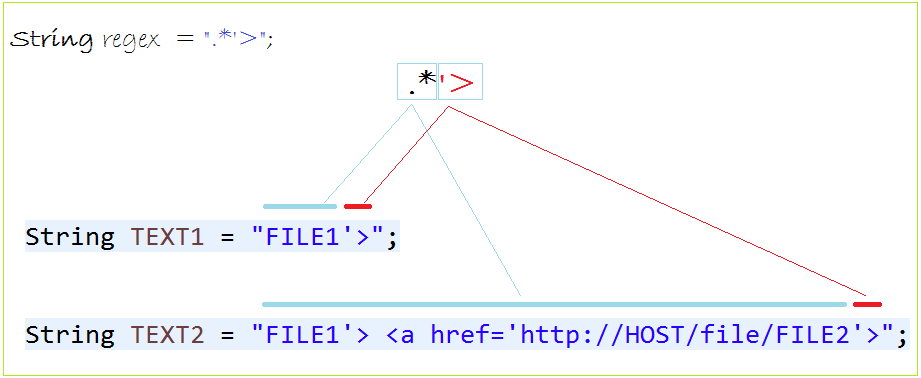
String regex = ".\*'>";

// Đoạn TEXT1 sau đây có vẻ hợp với regex nói trên.

String TEXT1 = "FILE1'>";

// Đoạn TEXT2 sau cũng hợp với regex nói trên.

String TEXT2 = "FILE1'> <a href='[http://HOST/file/FILE2](http://host/file/FILE2)'>";



\*? sẽ tìm ra một phù hợp nhỏ nhất. Chúng ta xem ví dụ sau:

import java.util.regex.Matcher;

import java.util.regex.Pattern;

public class NamedGroup2 {

  public static void main(String[] args) {

      String TEXT = "<a href='[http://HOST/file/FILE1](http://host/file/FILE1)'>File 1</a>"

              + "<a href='[http://HOST/file/FILE2](http://host/file/FILE2)'>File 2</a>";

      // Java >= 7.

      // Định nghĩa một group có tên fileName

      // \*? ==> Nó sẽ tìm một phù hợp nhỏ nhất.

      String regex = "/file/(?<fileName>.\*?)'>";

      Pattern pattern = Pattern.compile(regex);

      Matcher matcher = pattern.matcher(TEXT);

      while (matcher.find()) {

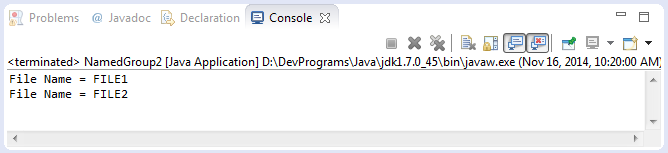
          System.out.println("File Name = " + matcher.group("fileName"));

      }

  }

}

Kết quả chạy ví dụ:



## 4.8 Demo chương trình sử dụng biểu thức chính quy quy định cách đặt tên biến và khai báo biến trong ngôn ngữ lập trình C

Để hoàn thành được chương trình này thì điều đầu tiên chúng ta cần phải tìm hiểu lại cách khai báo biến, quy định cách đặt tên biến trong ngôn ngữ C.

Trong ngôn ngữ C có rất nhiều cách khai báo biến hợp lệ như sau :

Int x,y;

Int x=5,y=6,z;

Float x[6],y[8]; // khai báo mảng

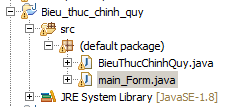
Char \*s; // khai báo con trỏ

Int \*z[6][7];//con trỏ và mảng đa chiều

Việc khai báo tên biến cũng phải tuân theo những quy tắc như :

* Không được phép trùng tên với từ khóa
* Không được phép bắt đầu bằng số
* Có thể bắt đầu bằng ký tự \_

Mã nguồn chương trình có cấu trúc như sau :



ở đây lớp main\_Form.java chỉ có tác dụng tạo giao diện và hiện thị kết quả demo.Còn lớp BieuThucChinhQuy.java mới là lớp quan trọng nhất.Nhiệm vụ của nó là nhận vào một chuỗi khai báo biến và kết trả là trả về xem khai báo biến đó là đúng hay là sai,hinh thức các hàm của lớp này như sau :

public class BieuThucChinhQuy {

public boolean kiemtra(String y)

{

}

public boolean kiemTraKhaiBaoBienKieu1(String x)

{

}

public boolean kiemTraKhaiBaoBienKieu2(String x)

{

}

public boolean kiemTraKhaiBaoBienKieu3(String x)//kieu nay la kieu khac

{

}

public boolean kiemTraKhaiBaoBienKieu4(String x)//kiem tra khai bao mang

{

}

public boolean kiemTraKhaiBaoBienKieu5(String x)

{

}

public boolean kiemTraKhaiBaoBienKieu6(String x)

{

}

public boolean kiemTraKhaiBaoBienKieu7(String x)

{

}

}

ở đây chúng ta phải hiểu là các phương thức kiemTraKhaiBaoBienKieu là những phương thức dùng để kiểm tra xem cái cách khai báo biến x của chúng ta có đúng với một kiểu khai báo hợp lệ của ngôn ngữ C hay không.Chẳng hạn

public boolean kiemTraKhaiBaoBienKieu1(String x)

{

boolean match = x.matches("(\\s\*(int|float|double|char|)\\s\*(\_|[a-zA-Z])[a-zA-Z0-9\_]\*;\\s\*)+");

return match;

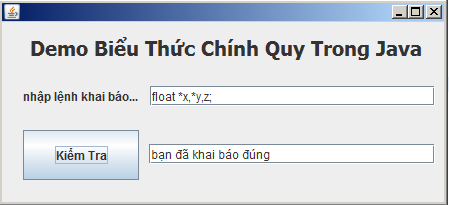
}

Kiểu khai báo ở đây nghĩa là đầu tiên người sử dụng có thể khai báo bao nhiêu dấu cách tùy ý,sau đó phải khai báo một trong 4 giá trị int,float,double hoặc char tiếp theo đó là bao nhiêu dấu cách tùy ý và tên biến thì có thể bắt đầu bằng dấu \_ hoặc một chữ cái bất kỳ (không được phép là số),kết thúc việc khai báo là dấu ;

Rõ ràng khi chúng ta đưa chuỗi khai báo x vào phương thức này thì nó sẽ kiểm tra cho chúng ta xem x có đúng với chuẩn đó hay không , nếu đúng thì trả về true còn không thì trả về flase.

Tất nhiên thì trên đây chỉ là định nghĩa cho một trường hợp trong rất nhiều cách khai báo biến trong ngôn ngữ C,đó là lý do tại sao chúng ta phải định nghĩa những trường hợp khai báo hợp lệ khác nữa trong các phương thức kiemTraKhaiBaoBienKieu2, kiemTraKhaiBaoBienKieu3, kiemTraKhaiBaoBienKieu4…..Mỗi trường hợp sẽ định nghĩa một kiểu khai báo ,và người dùng chỉ cần khai báo đúng ít nhất một kiểu trong số đó là chương trình sẽ xác ðịnh ðó là một khai báo hợp lệ.

Chýõng trình demo của chúng ta sẽ như sau :



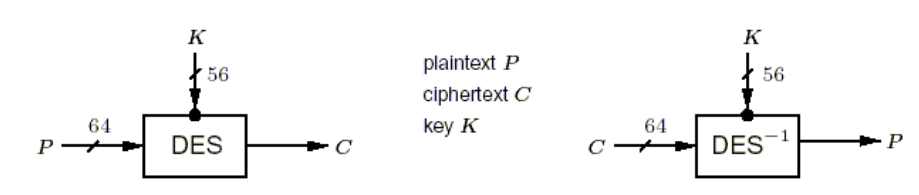
# 5.SỬ DỤNG JAVA DEMO CÀI ĐẶT MÃ HÓA DES

**5.1. LỊCH SỬ**

Vào thập niên 60, hệ mã Lucifer đã được đưa ra bởi Horst Feistel. Hệ mã này gắn liền với hãng IBM nổi tiếng. Sau đó Ủy ban tiêu chuẩn Hoa kỳ đã dàn xếp với IBM để thuật toán mã hóa này thành miễn phí và phát triển nó thành chuẩn mã hóa dữ liệu và công bố vào ngày 15/02/1977

**5.2. PHƯƠNG PHÁP BẢO MẬT**

DESlà thuật toán mã hóa với input là khối 64 bit, output cũng là khối 64 bit. Khóa mã hóa có độ dài 56 bit, thực ra chính xác hơn phải là 64 bit với các bit ở vị trí chia hết cho 8 có thể sử dụng là các bit kiểm tra tính chẵn lẻ. Số khóa của không gian khóa K là 256



**Hình 3.1.** Chuẩn mã dữ liệu DES

Thuật toán thực hiện 16 vòng. Từ khóa input K, 16 khóa con 48 bit Ki sẽ được sinh ra, mỗi khóa cho mỗi vòng thực hiện trong quá trình mã hóa. Trong mỗi vòng, 8 ánh xạ thay thế 6 bit thành 4 bit Si ( còn gọi là hộp Si) được chọn lựa kỹ càng và cố định, ký hiệu chung là S sẽ được sử dụng. Bản rõ 64 bit sẽ được sử dụng chia thành 2 nữa L0 và R0. Các vòng có chức năng giống nhau, nhận input là Li-1 và Ri-1 từ vòng truớc và sinh ra output là các xâu 32 bit Li và Ri như sau:

Li=Ri-1;

Ri=Li-1 ⊕ f(Ri-1) trong đó f(Ri-1, Ki)=P(S(E(Ri-1)⊕Ki));

Trong đó:

- ⊕ là ký hiệu của phép tuyển loại trừ (XOR) của hai xâu bit theo modulo 2.

- Hàm f là một hàm phi tuyến

- E là hoán vị mở rộng ánh xạ R­i-1 từ 32 bit thành 48 bit (đôi khi tất cả các bit sẽ được sử dụng hoặc một bit sẽ được sử dụng hai lần)

- P là hoán vị cố định khác của 32 bit

Một hoán vị khởi đầu (IP) được sử dụng cho vòng đầu tiên, sau vòng cuối cùng nửa trái và phải sẽ được đổi cho nhau và xâu cuối cùng kết quả sẽ được hoán vị lần cuối bởi hoán vị ngược của IP (IP-1).

Quá trình giải mã diễn ra tương tự nhưng với các khóa con ứng dụng vào các vòng theo thứ tự ngược lại

Có thể hình dung đơn giản là phần bên phải trong mỗi vòng (sau khi mở rộng input 32 bit thành 8 ký tự 6 bit – xâu 48 bit) sẽ thực hiện một tính toán thay thế phụ thuộc khóa trên mỗi ký tự trong xâu 48 bit, và sau đó sử dụng một phép chuyển bit cố định để phân bố lại các bit của các ký tự kết quả hình thành nên output 32 bit.

Các khóa con Ki (chứa 48 bit của K) được tính bằng cách sử dụng các bảng PC1 và PC2 (Permutation Choice 1 và 2). Trước tiên 8 bit ( K8, K16, …, K64) của K bị bỏ đi (áp dụng PC1). 56 bit còn lại được hoán vị và gán cho hai biến 28 bit C và D sẽ được quay 1 hoặc 2 bit, và các khóa con 48 bit Ki được chọn từ kết quả của việc ghép hai xâu với nhau.

Như vậy, ta có thể mô tả toàn bộ thuật toán sinh mã DES dưới dạng công thức như sau:

**Y = IP-1 • f16 • T • f15 • T • ... • f2 • T • f1 • IP(X)**

Trong đó :

- T mô tả phép hoán vị của các khối Li, RI (1 ≤ i ≤ 15).

- fi mô tả việc dùng hàm f với khóa Ki (1 ≤ i≤ 16)

**4.3. ƯU NHƯỢC ĐIỂM**

**4.3.1. Ưu điểm:**

- Có tính bảo mật cao

- Công khai, dễ hiểu

- Nó có thể triển khai trên thiết bị điện tử có kích thước nhỏ

**4.3.2. Các yếu điểm của DES:**

**4.3.2.1. Tính bù**

Nếu ta ký hiệu  là phần bù của u (ví dụ : 0100101 là phần bù của 1011010) thì des có tính chất sau

y = DES (x,k) → = DES (  ,)

Cho nên nếu ta biết mã y được mã hóa từ thông tin x với khóa K thì ta suy được bản mã  được mã hóa từ bản rõ  với khóa . Tính chất này là một yếu điểm của DES bởi vì qua đó đối phương có thể loại bỏ đi một số khóa phải thử khi tiến hành thử giải mã theo kiểu vét cạn

**4.3.2.2. khóa yếu**

Khóa yếu là các khóa mà theo thuật toán sinh khóa con thì tất cả 16 khóa con đều như nhau : K1=K2=... =K16

Điều đó khiến cho việc mã hóa và giải mã đối với khóa yếu là giống hệt nhau

|  |  |  |
| --- | --- | --- |
| Khóa yếu (Hex) | C0 | D0 |
| 0101 0101 0101 0101  FEFE FEFE FEFE FEFE  1F1F 1F1F 0E0E 0E0E  E0E0 E0E0 F1F1 F1F1 | {0}28  {1}28  {0}28  {1}28 | {0}28  {1}28  {1}28  {0}28 |

**Bảng 3.1**.Các khóa yếu của DES

Đồng thời còn có 6 cặp khóa nửa yếu (semi-weak key) khác với thuộc tính như sau :

y= DES(x,k1) và y=DES(x,k2)

Nghĩa là với 2 khóa khác nhau nhưng mã hóa cùng một bản mã từ cùng một bản rõ :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| C0 | D0 | Semi-weak key(Hex) | | C0 | D0 |
| {01}14  {01}14  {01}14  {01}14  {0}28  {1}28 | {01}14  {10}14  {0}28  {1}28  {01}14  {01}14 | 01FE 01FE 01FE 01FE  1FE0 1FE0 1FE0 1FE0  01E0 01E0 01F1 01F1  1FFE 1FFE 0EFE 0EFE  011F 011F 010E 010E  E0FE E0FE F1FE F1FE | FE01 FE01 FE01 FE01  E01F E01F E01F E01F  E001 E001 F101 F101  FE1F FE1F FE0E FE0E  1F01 1F01 0E01 0E01  FEE0 FEE0 FEF1 EF1 | {10}14  {10}14  {10}14  {10}14  {0}28  {1}28 | {10}14  {01}14  {0}28  {1}28  {10}14  {10}14 |

**Bảng 3.2.**Các khóa nửa yếu của DES

**4.3.3.3. DES có cấu trúc đại số**

Với 64 bit khối bản rõ có thể được ánh xạ lên tất cả các vị trí của khối 64 bit khối bản mã trong 264 cách. Trong thuật toán DES, với 56 bit khóa có thể cho chúng ta 256 (khoảng 1017 ) vị trí ánh xạ. Với việc đa mã hóa thì không gian ánh xạ còn lớn hơn. Tuy nhiên điều này chỉ đúng nếu việc mã hóa DES là không cấu trúc

Với DES có cấu trúc đại số thì việc đa mã hóa sẽ được xem ngang bằng với việc đơn mã hóa. Ví dụ như có hai khóa bất kỳ K1 và K2 thì sẽ luôn được khóa K3 như sau :

EK2(EK1(X))=EK3(X)

Nói một cách khác, việc mã hóa DES mang tính chất “nhóm”, đầu tiên mã hóa bản rõ bằng khóa K1 sau đó là khóa K2 sẽ giống với việc mã hóa ở khóa K3. Điều này thực sự quan trọng nếu sử dụng DES trong đa mã hóa. Nếu một “nhóm” được phát với cấu trúc hàm quá nhỏ thì tính an toàn sẽ giảm.

**4.3.3.4. Không gian khóa K**

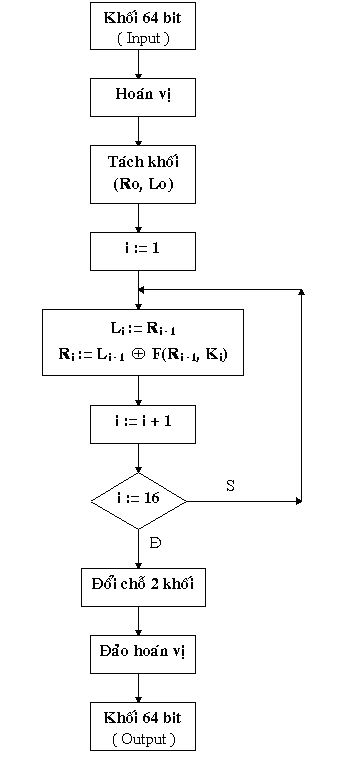
DES có 256 = 1017 khóa. Nếu chúng ta biết được một cặp “tin/mã” thì chúng ta có thể thử tất cả 1017 khả năng này để tìm ra khóa cho kết quả khớp nhất. Giả sử như một phép thử mất 10-6s, thì chúng sẽ mất 1011s, tức 7300 năm. Nhưng với các máy tính được chế tạo theo xử lý song song. Chẳng hạn với 107 con chip mã DES chạy song song thì bây giờ mỗi một con chipset chỉ phải chịu trách nhiệm tính toán với 1010 phép thử. Chipset mã DES ngày nay có thể xử lý tốc độ 4.5x107 bit/s tức có thể làm được hơn 105 phép mã DES trong một giây.

Vào năm 1976 và 1977, Dieffie và Hellman đã ước lượng rằng có thể chế tạo được một máy tính chuyên dụng để vét cạn không gian khóa DES trong ½ ngày với cái giá 20 triệu đô la. Năm 1984, chipset mã hóa DES với tốc độ mã hóa 256000 lần/giây. Năm 1987, đã tăng lên 512000 lần/giây. Vào năm 1993, Michael Wiener đã thiết kế một máy tính chuyên dụng với giá 1 triệu đô la sử dụng phương pháp vét cạn để giải mã DES trung bình trong vòng 3,5 giờ (và chậm nhất là 7 giờ).

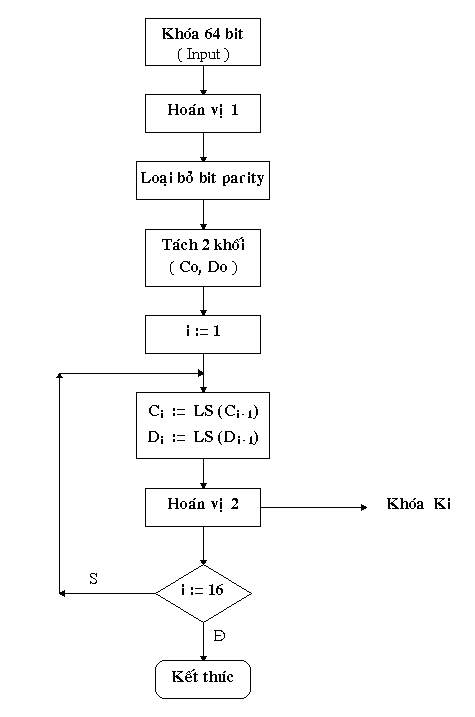
Đến năm 1990, hai nhà toán học người Do Thái – Biham và Shamir – đã phát minh ra phương pháp mã hóa vi sai (diferential cryptanalyis), đây là một kỹ thuật sử dụng những phỏng đoán khác nhau trong bản rõ để đưa ra những thông tin trong bản mã. Với phương pháp này, Biham và Shamir đã chứng minh rằng nó hiệu quả hơn cả phương pháp vét cạn.

Phá mã vi sai là thuật toán xem xét những cặp mã khóa khác nhau, đây là những cặp mã hóa mà bản mã của chúng là khác biệt. Người ta sẽ phân tích tiến trình biến đổi của những cặp mã này thông qua các vòng của DES khi chúng được mã hóa với cùng một khóa K. Sau đó sẽ chọn hai bản rõ khác nhau một cách ngẫu nhiên hợp lý nhất. Sử dụng sự khác nhau của kết quả bản mã và gán cho những khóa khác nhau một cách phù hợp nhất. Khi phân tích nhiều hơn những cặp bản mã, chúng ta sẽ tìm ra một khóa được xem là đúng nhất.

## 4.1. Sơ đồ khối



**Hình 3.2.**  Sơ đồ khối chương trình DES



**Hình 3.3.** Sơ đồ khối quá trình sinh khóa

## 4.5. Thuật toán

DES là thuật toán mã hóa khối, nó xử lý từng khối thông tin của bản rõ có độ dài xác định là 64 bit. Trước khi đi vào 16 chu trình chính, khối dữ liệu cần bảo mật được “bẻ” ra từng khối 64 bit, và từng khối 64 bit này sẽ được lần lượt đưa vào 16 vòng mã hóa DES để thực hiện

**Input:** bản rõ M = m1m2 … m64, là một khối 64 bit, khóa 64 bit K = k1k2 . . . k64 (bao gồm cả 8 bit chẵn lẻ, việc thêm bit chẵn lẻ sao cho các đoạn khóa 8 bit có số bit 1 là lẻ)

**Output:** bản mã 64 bit C = c1 c2 … c64

1. Sinh khóa con. Tính các khóa con theo thuật toán sinh khóa con
2. (L0,R0) ← IP (m1 m2 . . . m64) (sử dụng bản hoán vị IP để hoán vị các bit, kết quả nhận được chia thành 2 nửa là L0 = m58 m50 . . . m8, R0 = m57 m49 . . . m7)
3. Với i chạy từ i=1 đến 16 thực hiện:

Tính các Li và Ri theo công thức:

Li=Ri-1;

Ri=Li-1 ⊕ f(Ri-1) trong đó f ( Ri-1, Ki )=P ( S ( E ( Ri-1 ) ⊕ Ki ) );

Việc tính f ( Ri-1 ) = P ( S ( E ( Ri-1 ) ⊕ Ki ) ) được thực hiện như sau:

🟆 Mở rộng Ri-1 = r1r2 . .. r32 từ 32 bit thành 48 bit bằng cách sử dụng hoán vị mở rộng E.

T ← E ( Ri-1 ) . ( Vì thế T = r32 r1 r2 . . . r32 r1)

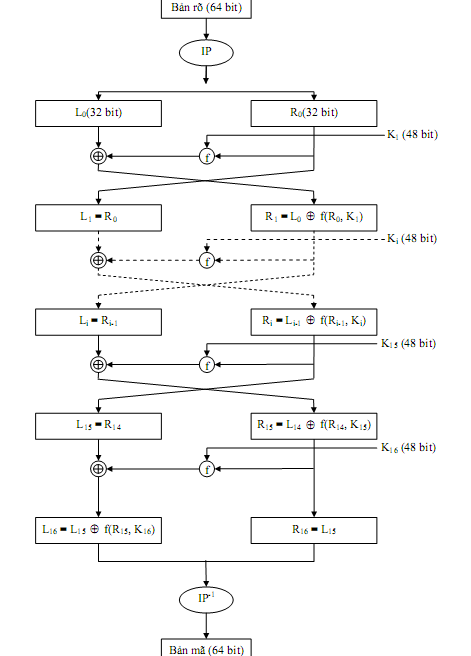
🟆 T’ ← T ⊕ Ki. Biểu diễn T’ như là các xâu gồm 8 ký tự 6 bit T’ = ( B1, . . . ,B8 )

🟆 T’’ ← ( S1 ( B1 ) , S2 ( B2 ) , . . . , S8 ( B8 ) ). Trong đó Si ( Bi ) ánh xạ b1b2 . . . b6 thành các xâu 4 bit của phần tử thuộc hàng r và cột c của các bảng Si (S box) trong đó r = 2 \* b1 + b6 và c = b2 b3 b4 b5 là một số nhị phân từ 0 tới 15. Chẳng hạn S1 ( 011011) sẽ cho r = 1 và c = 3 và kết quả là 5 biểu diễn dưới dạng nhị phân là 0101.

🟆 T’’’ ← P ( T’’) trong đó P là hoán vị cố định để hoán vị 32 bit của

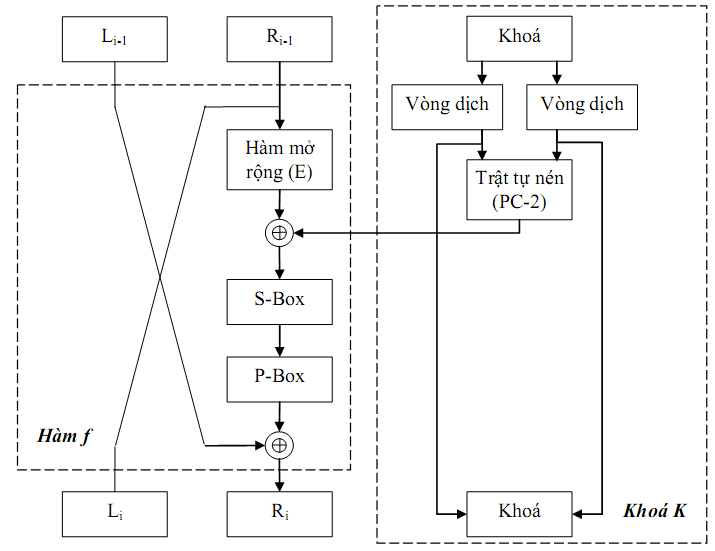
T’’ = t1 t2 . . . t32 sinh ra t16 t7 . . . t25

1. Khối từ b1 b2 . . . b64 ← ( R16, L16) ( đổi vị trí các khối cuối cùng L16, R16)
2. C ← IP-1 ( b1 b2 . . . b64) ( Biến đổi sử dụng IP-1, C = b40 b8 . . . b25).



**Hình 3.4**. Sơ đồ mã hóa DES

### 4.5.1 Quá trình mã hóa:



**Hình 3.5.** Sơ đồ một vòng DES

Chia làm 3 giai đoạn:

**4.5.1.1.Giai đoạn 1:**

Với bản rõ cho trước x, 1 xâu x' sẽ được tạo ra bằng cách hoán vị các bit của x theo hoán vị ban đầu IP:

x'=IP(x)=L0 R0

L0:32 bit đầu

R0:32 bit cuối

**IP:**

|  |
| --- |
| 58 50 42 34 26 18 10 2 60 52 44 36 28 20 12 4 62 54 46 38 30 22 14 6 64 56 48 40 32 24 16 8 57 49 41 33 25 17 9 1 59 51 43 35 27 19 11 3 61 53 45 37 29 21 13 5 63 55 47 39 31 23 15 7 |

**Bảng 3.3.** Hoán vị IP

**4.5.1.2.Giai đoạn 2:**

Tính toán 16 lần lập theo 1 hàm xác định. Ta sẽ tính LiRi (1≤ i ≤ 16) theo quy tắc:

Li=Ri-1  
 Ri = Li-1⊕ f (Ri-1, Ki)

⊕ là toán tử Xor

k1,k2,k3.....k16 là xâu bit độ dài 48 bit được tính qua hàm khoá K (thực tế thì Ki là 1 phép hoán vị bit trong K)

**4.5.1.3.Giai đoạn 3:**

Áp dụng hoán vị ngược IP-1 cho xâu bit R16 L16 ta thu được bản mã y:  
y = IP-1 (R16 L16)

+ Chú ý vị trí của R16 và L16.

**IP-1**

|  |
| --- |
| 40 8 48 16 56 24 64 32 39 7 47 15 55 23 63 31 38 6 46 14 54 22 62 30 37 5 45 13 53 21 61 29 36 4 44 12 52 20 60 28 35 3 43 11 51 19 59 27 34 2 42 10 50 18 58 26 33 1 41 9 49 17 57 25 |

**Bảng 3.4.** Hoán vị IP-1

**3.5.2. Quá trình giải mã:**

Do là 1 thuật toán đối xứng nên quá trình giải mã và mã hóa cũng gần giống nhau chỉ khác ở:

Li=Ri-1  
 Ri = Li-1⊕ f (Ri-1, K16-i)

Khóa K của hàm F sẽ đi từ 16 ->0

**3.5.3. Hàm F**

Đầu vào hàm f có 2 biến:

biến 1: R là xâu bit có độ dài 32 bit, biến 2:K là xâu bit có độ dài 48 bit. Đầu ra của f là xâu bit có độ dài 32 bit.

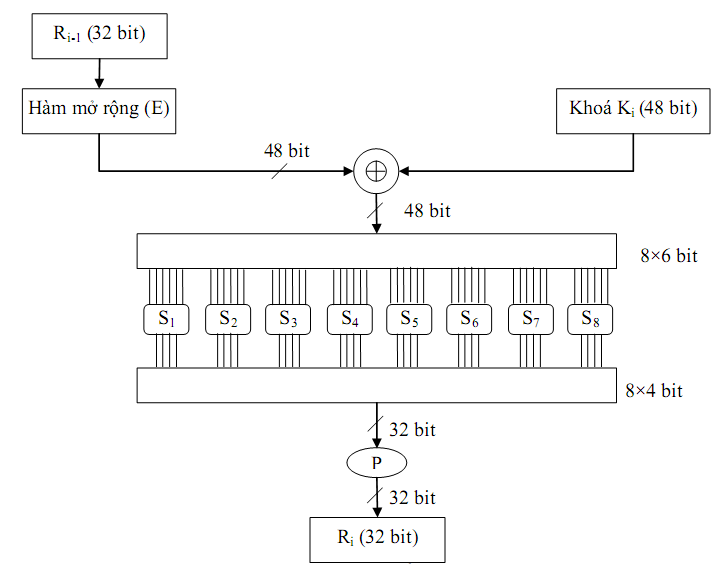
- Biến thứ nhất Ri-1 được mở rộng thành một xâu bit có độ dài 48 bit theo một hàm mở rộng cố đinh E. Thực chất hàm mở rộng E ( Ri-1) là một hoán vị có lặp trong đó lặp lại 16 bit của Ri-1

- Tính E ( Ri-1 ) ⊕ Ki và viết kết quả thành 8 xâu 6 bit B1B2B3B4B5B6B7B8

- Đưa khối 8 bit Bi vào 8 bảng S1, S2, … .S8 ( được gọi là các hộp S-Box). Mỗi hộp S-Box là một bảng 4\*16 cố định có các cột từ 0 đến 15 và các hàng từ 0 đến 3. Với mỗi xâu 6 bit Bi = b1b2b3b4b5b6, ta tính được Si (B i) như sau: hai bit b1b6  xác định hàng r trong trong hộp Si, bốn bit b2b3b4b5 xác định cột c trong hộp S­i. Khi đó, Si (Bi) sẽ xác định phần tử Ci=Si ( r,c), phần tử này viết dưới dạng nhị phân 4 bit. Như vậy, 8 khối 6 bit Bi ( 1 ≤ i ≤ 8 ) sẽ cho ra 8 khối 4 bit Ci với ( 1 ≤ i ≤ 8 )

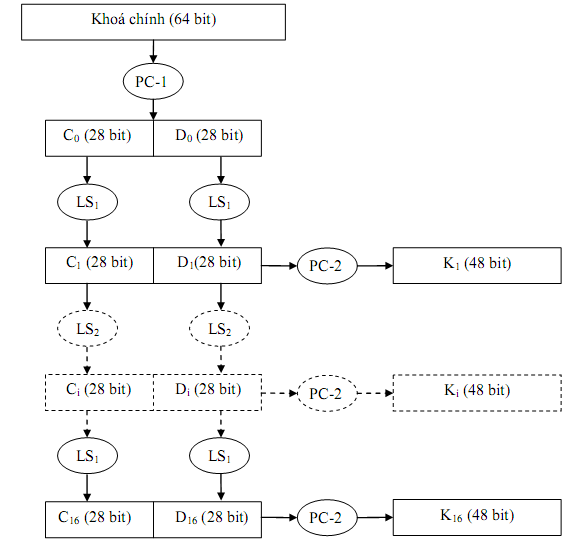
- Xâu bit C = C1C2C3C4C5C6C7C8 có độ dài 32 bit được hoán vị theo phép toán hoán vị P (hộp P-Box). Kết quả P(C) sẽ là kết quả của hàm f( Ri-1, K­i), và cũng chính Ri cho vòng sau

**Hình 3.6.** Sơ đồ hàm F



**3.5.4. Quá trình tạo khóa con**

- Mười sáu vòng lặp DES chạy cùng thuật toán như nhau nhưng với 16 khóa con khác nhau. Các khóa con đều được sinh ra từ khóa chính của DES bằng một thuật toán sinh khóa con.



**Hình 3.7.** Sơ đồ tạo khóa con

K là xâu có độ dài 64 bit, một bit trong 8 bit của byte sẽ được lấy ra dùng để kiểm tra phát hiện lỗi( thường thì các bit này ở vị trí 8, 16, 24, ...,64) tạo ra chuỗi 56 bit. Sau khi bỏ các bit kiểm tra ta sẽ hoán vị chuối 56 bit, 2 bước trên được thực hiện thông qua hoá vị ma trận PC1.

**PC-1**

|  |
| --- |
| 57 49 41 33 25 17 9 1 58 50 42 34 26 18 10 2 59 51 43 35 27 19 11 3 60 52 44 36 63 55 47 39 31 23 15 7 62 54 46 38 30 22 14 6 61 53 45 37 29 21 13 5 28 20 12 4 |

**Bảng 3.5 .** Hoán vị PC-1

Ta chia PC-1 thành 2 phần:

C0: 28 bit đầu

D0: 28 bit cuối

Mỗi phần sẽ được xử lý 1 cách độc lập.

Ci=LSi(Ci-1)  
 Di = LSi(Ci-1) với 1≤ i ≤ 16

+ LSi biểu diễn phép dịch bit vòng(cyclic shift) sang trái 1 hoặc 2 vị trí tuỳ thuộc vào i. Cyclic shift sang trái 1 bit nếu i=1, 2 , 9, 16 hoặc sang trái 2 bit nếu i thuộc các vị trí còn lại.

Ki=PC-2(CiDi).

Số bit dịch của các vòng:



**Bảng 3.6.** Bảng dịch bit tại các vòng lặp của DES

+ PC-2 là hoán vị cố định sẽ hoán vị chuỗi CiDi 56 bit thành chuỗi 48 bit.

**PC-2**

|  |
| --- |
| 14 17 11 24 1 5 3 28 15 6 21 10 23 19 12 4 26 8 16 7 27 20 13 2 41 52 31 37 47 55 30 40 51 45 33 48 44 49 39 56 34 53 46 42 50 36 29 32 |

**Bảng 3.7.** Hoán vị PC-2

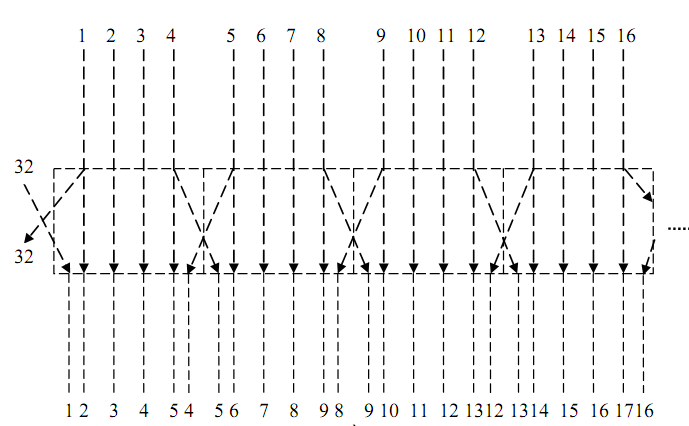
**3.5.5. Hàm (ánh xạ) mở rộng (E)**

Hàm mở rộng (E) sẽ tăng độ dài từ Ri từ 32 bit lên 48 bit bằng cách thay đổi các thứ tự của các bit cũng như lặp lại các bit. Việc thực hiện này nhằm hai mục đích:

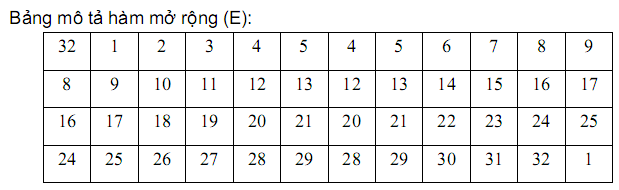
- Làm độ dài của Ri cùng cỡ với khóa K để thực hiện việc cộng modulo XOR.

- Cho kết quả dài hơn để có thể được nén trong suốt quá trình thay thế

Tuy nhiên, cả hai mục đích này nhằm một mục tiêu chính là bảo mật dữ liệu. Bằng cách cho phép 1 bit có thể chèn vào hai vị trí thay thế, sự phụ thuộc của các bit đầu ra với các bit đầu vào sẽ trải rộng ra. DES được thiết kế với điều kiện là mỗi bit của bản mã phụ thuộc vào mỗi bit của bản rõ và khóa.



**Hình 3.8.** Sơ đồ của hàm mở rộng



**Bảng 3.8.** Hàm mở rộng E

Đôi khi nó được gọi là hàm E-Box, mỗi 4 bit của khối vào, bit thứ nhất và bit thứ tư tương ứng với 2 bit của đầu ra, trong khi bit thứ hai và ba tương ứng với 1 bit ở đầu ra

**3.5.6. Hộp S – Box**

- Mỗi hàng trong mỗi hộp S là hoán vị của các số nguyên từ 0 đến 15

- Không có hộp S nào là hàm Affine hay tuyến tính đối với các đầu vào của nó

- Sự thay đổi của một bit đầu vào sẽ dẫn đến sự thay đổi ít nhất hai bit đầu ra

- Đối với hộp S bất kỳ và với đầu vào x ( một xâu bit có độ dài bằng 6 bit) bất kỳ, thì S(x) và S (x ⊕ 001100) phải khác nhau ít nhất là 2 bit

Sau khi cộng modulo với khóa K, kết quả thu được chuỗi 48 bit chia làm 8 khối đưa vào 8 hộp S-Box. Mỗi hộp S-Box có 6 bit đầu vào và 4 bit đầu ra ( tổng bộ nhớ yêu cầu cho 8 hộp S-Box chuẩn DES là 256 bytes). Kết quả thu được là một chuỗi 32 bit tiếp tục vào hộp P-Box

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S1** | | | | | | | | | | | | | | | |
| 14  0  4  15 | 4  15  1  12 | 13  7  14  8 | 1  4  8  2 | 2  14  13  4 | 15  2  6  9 | 11  13  2  1 | 8  1  11  7 | 3  10  15  5 | 10  6  12  11 | 6  12  9  3 | 12  11  7  14 | 5  9  3  10 | 9  5  10  0 | 0  3  5  6 | 7  8  0  13 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S2** | | | | | | | | | | | | | | | |
| 15  3  0  13 | 1  13  14  8 | 8  4  7  10 | 14  7  11  1 | 6  15  10  3 | 11  2  4  15 | 3  8  13  4 | 4  14  1  2 | 9  12  5  11 | 7  0  8  6 | 2  1  12  7 | 13  10  6  12 | 12  6  9  0 | 0  9  3  5 | 5  11  2  14 | 10  5  15  9 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S3** | | | | | | | | | | | | | | | |
| 10  13  13  1 | 0  7  6  10 | 9  0  4  13 | 14  9  9  0 | 6  3  8  6 | 3  4  15  9 | 15  6  3  8 | 5  10  0  7 | 1  2  11  4 | 13  8  1  15 | 12  5  2  14 | 7  14  12  3 | 11  12  5  11 | 4  11  10  5 | 2  15  14  2 | 8  1  7  12 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S4** | | | | | | | | | | | | | | | |
| 7  13  10  3 | 13  8  6  15 | 14  11  9  0 | 3  5  0  6 | 0  6  12  10 | 6  15  11  1 | 9  0  7  13 | 10  3  13  8 | 1  4  15  9 | 2  7  1  4 | 8  2  3  5 | 5  12  14  11 | 11  1  5  12 | 12  10  2  7 | 4  14  8  2 | 15  9  4  14 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S5** | | | | | | | | | | | | | | | |
| 2  14  4  11 | 12  11  2  8 | 4  2  1  12 | 1  12  11  7 | 7  4  10  0 | 10  7  13  14 | 11  13  7  2 | 6  1  8  13 | 8  5  15  6 | 5  0  9  15 | 3  15  12  0 | 15  10  5  9 | 13  3  6  10 | 0  9  3  4 | 14  8  0  5 | 9  6  14  3 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S6** | | | | | | | | | | | | | | | |
| 12  10  9  4 | 1  15  14  3 | 10  4  15  2 | 15  2  5  12 | 9  7  2  9 | 2  12  8  5 | 6  9  12  15 | 8  5  3  10 | 0  6  7  11 | 13  1  0  14 | 3  13  4  1 | 4  14  10  7 | 14  0  1  6 | 7  11  13  0 | 5  3  11  8 | 11  8  6  13 |

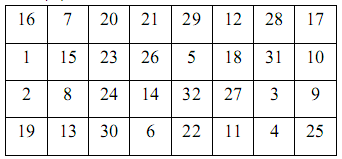
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S7** | | | | | | | | | | | | | | | |
| 4  13  1  6 | 11  0  4  11 | 2  11  11  13 | 14  7  13  8 | 15  4  12  1 | 0  9  3  4 | 8  1  7  10 | 13  10  14  7 | 3  14  10  9 | 12  3  15  5 | 9  5  6  0 | 7  12  8  15 | 5  2  0  14 | 10  15  5  2 | 6  8  9  3 | 1  6  2  12 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **S8** | | | | | | | | | | | | | | | |
| 13  1  7  2 | 2  15  11  1 | 8  13  4  14 | 4  8  1  7 | 6  10  9  4 | 15  3  12  10 | 11  7  14  8 | 1  4  2  13 | 10  12  0  15 | 9  5  6  12 | 3  6  10  9 | 14  11  13  0 | 5  0  15  3 | 0  14  3  5 | 12  9  5  6 | 7  2  8  11 |

**Bảng 3.9.** 8 hộp S-Box

**3.5.7. Hộp P-Box**

Việc hoán vị này mang tính đơn ánh, nghĩa là một bit đầu vào sẽ cho một bit ở đầu ra, không bit nào được sử dụng 2 lần hay bị bỏ qua. Hộp P-Box thực chất chỉ là chức năng sắp xếp đơn thuần theo bảng sau:



**Bảng 3.10.** Bảng hoán vị P

## 5.6. Lập mã DES

Đây là ví dụ về việc sử dụng DES. Giả sử ta mã hóa bản rõ sau trong dạng thập lục phân(Hexadecimal)

0123456789ABCDEF

sử dụng khóa thập lục phân :

133457799BBCDFF1

Khóa trong dạng nhị phân không có các bit kiểm tra sẽ là :

00010010011010010101101111001001101101111011011111111000.

Áp dụng IP , ta nhận được L0 và R0 (trong dạng nhị phân)



|  |  |  |
| --- | --- | --- |
| **L0**  **L1 = R0** | **=**  **=** | **11001100000000001100110011111111**  **11110000101010101111000010101010** |

16 vòng lặp mã được thể hiện như sau:

|  |  |  |
| --- | --- | --- |
| **E(R0)**  **K1**  **E(R0) ⊕ K1**  **S-box output**  **f(R0,K1)**  **L2 = R1** | **=**  **=**  **=**  **=**  **=**  **=** | **011110100001010101010101011110100001010101010101**  **000110110000001011101111111111000111000001110010**  **011000010001011110111010100001100110010100100111**  **01011100100000101011010110010111**  **00100011010010101010100110111011**  **11101111010010100110010101000100** |

|  |  |  |
| --- | --- | --- |
| **E(R1)**  **K2**  **E(R1) ⊕ K2**  **S-box output**  **f(R1, K2)**  **L3 = R2** | **=**  **=**  **=**  **=**  **=**  **=** | **011101011110101001010100001100001010101000001001**  **011110011010111011011001110110111100100111100101**  **000011000100010010001101111010110110001111101100**  **11111000110100000011101010101110**  **00111100101010111000011110100011**  **11001100000000010111011100001001** |

|  |  |  |
| --- | --- | --- |
| **E(R2)**  **K3**  **E(R2) ⊕ K3**  **S-box output**  **f(R2, K3)**  **L4 = R3** | **=**  **=**  **=**  **=**  **=**  **=** | **111001011000000000000010101110101110100001010011**  **010101011111110010001010010000101100111110011001**  **101100000111110010001000111110000010011111001010**  **00100111000100001110000101101111**  **01001101000101100110111010110000**  **10100010010111000000101111110100** |

|  |  |  |
| --- | --- | --- |
| **E(R3)**  **K4**  **E(R3) ⊕ K4**  **S-box output**  **f(R3, K4)**  **L5 = R4** | **=**  **=**  **=**  **=**  **=**  **=** | **010100000100001011111000000001010111111110101001**  **011100101010110111010110110110110011010100011101**  **001000101110111100101110110111100100101010110100**  **00100001111011011001111100111010**  **10111011001000110111011101001100**  **011101110** |

|  |  |  |
| --- | --- | --- |
| **E(R4)**  **K5**  **E(R4) ⊕ K5**  **S-box output**  **f(R4, K5)**  **L6 = R5** | **=**  **=**  **=**  **=**  **=**  **=** | **101110101110100100000100000000000000001000001010**  **011111001110110000000111111010110101001110101000**  **110001100000010100000011111010110101000110100010**  **01010000110010000011000111101011**  **00101000000100111010110111000011**  **10001010010011111010011000110111** |

|  |  |  |
| --- | --- | --- |
| **E(R5)**  **K6**  **E(R5) ⊕ K6**  **S-box output**  **f(R5, K6)**  **L7 = R6** | **=**  **=**  **=**  **=**  **=**  **=** | **110001010100001001011111110100001100000110101111**  **011000111010010100111110010100000111101100101111**  **101001101110011101100001100000001011101010000000**  **01000001111100110100110000111101**  **10011110010001011100110100101100**  **11101001011001111100110101101001** |

|  |  |  |
| --- | --- | --- |
| **E(R6)**  **K7**  **E(R6) ⊕ K7**  **S-box output**  **f(R6, K7)**  **L8 = R7** | **=**  **=**  **=**  **=**  **=**  **=** | **111101010010101100001111111001011010101101010011**  **111011001000010010110111111101100001100010111100**  **000110011010111110111000000100111011001111101111**  **00010000011101010100000010101101**  **10001100000001010001110000100111**  **00000110010010101011101000010000** |

|  |  |  |
| --- | --- | --- |
| **E(R7)**  **K8**  **E(R7) ⊕ K8**  **S-box output**  **f(R7, K8)**  **L9 = R8** | **=**  **=**  **=**  **=**  **=**  **=** | **000000001100001001010101010111110100000010100000**  **111101111000101000111010110000010011101111111011**  **111101110100100001101111100111100111101101011011**  **01101100000110000111110010101110**  **00111100000011101000011011111001**  **11010101011010010100101110010000** |

|  |  |  |
| --- | --- | --- |
| **E(R8)**  **K9**  **E(R8) ⊕ K9**  **S-box output**  **f(R8, K9)**  **L10 = R9** | **=**  **=**  **=**  **=**  **=**  **=** | **011010101010101101010010101001010111110010100001**  **111000001101101111101011111011011110011110000001**  **100010100111000010111001010010001001101100100000**  **00010001000011000101011101110111**  **00100010001101100111110001101010**  **00100100011111001100011001111010** |

|  |  |  |
| --- | --- | --- |
| **E(R9)**  **K10**  **E(R9) ⊕ K10**  **S-box output**  **f(R9, K10)**  **L11 = R10** | **=**  **=**  **=**  **=**  **=**  **=** | **000100001000001111111001011000001100001111110100**  **101100011111001101000111101110100100011001001111**  **101000010111000010111110110110101000010110111011**  **11011010000001000101001001110101**  **01100010101111001001110000100010**  **10110111110101011101011110110010** |

|  |  |  |
| --- | --- | --- |
| **E(R10)**  **K11**  **E(R10) ⊕ K11**  **S-box output**  **f(R10, K11)**  **L12 = R11** | **=**  **=**  **=**  **=**  **=**  **=** | **010110101111111010101011111010101111110110100101**  **001000010101111111010011110111101101001110000110**  **011110111010000101111000001101000010111000100011**  **01110011000001011101000100000001**  **11100001000001001111101000000010**  **11000101011110000011110001111000** |

|  |  |  |
| --- | --- | --- |
| **E(R11)**  **K12**  **E(R11) ⊕ K12**  **S-box output**  **f(R11, K12)**  **L13 = R12** |  | **011000001010101111110000000111111000001111110001**  **011101010111000111110101100101000110011111101001**  **000101011101101000000101100010111110010000011000**  **01111011100010110010011000110101**  **11000010011010001100111111101010**  **01110101101111010001100001011000** |

|  |  |  |
| --- | --- | --- |
| **E(R12)**  **K13**  **E(R12)⊕ K13**  **S-box output**  **f(R12, K13)**  **L14 = R13** | **=**  **=**  **=**  **=**  **=**  **=** | **001110101011110111111010100011110000001011110000**  **100101111100010111010001111110101011101001000001**  **101011010111100000101011011101011011100010110001**  **10011010110100011000101101001111**  **11011101101110110010100100100010**  **00011000110000110001010101011010** |

|  |  |  |
| --- | --- | --- |
| **E(R13)**  **K14**  **E(R13)⊕ K14**  **S-box output**  **f(R13, K14)**  **L15 = R14** | **=**  **=**  **=**  **=**  **=**  **=** | **000011110001011000000110100010101010101011110100**  **010111110100001110110111111100101110011100111010**  **010100000101010110110001011110000100110111001110**  **01100100011110011001101011110001**  **10110111001100011000111001010101**  **11000010100011001001011000001101** |

|  |  |  |
| --- | --- | --- |
| **E(R14)**  **K15**  **E(R14)⊕ K15**  **S-box output**  **f(R14, K15)**  **L16 = R15** | **=**  **=**  **=**  **=**  **=**  **=** | **111000000101010001011001010010101100000001011011**  **101111111001000110001101001111010011111100001010**  **010111111100010111010100011101111111111101010001**  **10110010111010001000110100111100**  **01011011100000010010011101101110**  **01000011010000100011001000110100** |

|  |  |  |
| --- | --- | --- |
| **E(R15)**  **K16**  **E(R15)⊕ K16**  **S-box output**  **f(R15, K16)**  **R16** | **=**  **=**  **=**  **=**  **=**  **=** | **001000000110101000000100000110100100000110101000**  **110010110011110110001011000011100001011111110101**  **111010110101011110001111000101000101011001011101**  **10100111100000110010010000101001**  **11001000110000000100111110011000**  **00001010010011001101100110010101** |

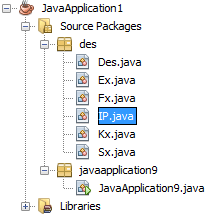
Cuối cùng , áp dụng IP-1 cho ta nhận được bản mã trong dạng thập lục phân như sau :



85E813540F0AB405

## 5.7 Chương trình DEMO

Chương trình demo của chúng ta như sau :



Lớp IP được dùng để xử lý khối dữ liệu 64 bit đầu tiên trong quá toàn bộ chu trình của mã hóa DES.Đây là lớp mà chúng ta sẽ xây dựng đâu tiên trong toàn bộ chương trình.

public class IP

{

private int a[] = {….};

private int b[] = {….};

public void hoanvi\_IP(int x[])

{

…….

}

public void hoanvi\_IP\_am1(int x[])

{

…….

}

}

Rõ ràng là nó có 2 thuộc tính là hai mảng kiểu int, nên nhớ là hai mảng này chúng ta có thể tự tạo ra cho mình hoặc nếu không thì có thể lấy luôn những bảng sẵn có.

Hai phương thức hoanvi\_IP (dùng cho quá trình mã hóa ) và hoanvi\_IP\_am1 (hoán vị cho ra bản mã),khối bản rõ 64 bit của chúng ta trước khi tách thành 2 nửa 32bit thì nó sẽ đi qua khối IP này để hoán vị các vị trí của bit.

Lớp Fx lớp này chính là đại diện cho hàm f trong quá trình mã hóa DES của chúng ta.Cõ lẽ phức tạp nhất chính là việc thiết kế hàm f này .Để thiết kế được nó chúng ta phải thiết kế một số lớp thành phần khác – những thành phần mà tạo nên Fx.Nó bao gồm :

Hộp E(chính là lớp Ex) có chức năng mở rộng 32 bit đầu vào thành 48 bit

Lớp Kx có chức năng tạo ra 16 khóa con tương ứng với mỗi chu trình trong quá trình mã hóa DES , mỗi khóa con dài 48 bit được tạo ra từ 64 bit(thực ra là từ 56 bit) của khóa chính

Cuối cùng là lớp Sx,lớp này có nhiệm vụ từ 48 bit đầu vào nó đi qua tổng cộng là 8 hộp S để cho 32 bit đầu ra,sau đó 32 bit đầu ra này sẽ đi qua thanh hoán vị P một lần nữa để cho 32 bit kết quả đầu ra.

Thứ nhất chúng ta xem hộp E được xây dựng chi tiết thế nào, nó là thứ đơn giản nhất trong F nên chúng ta nên đi xem trước

public class Ex

{

private int E[] = {....};

public void tao\_48\_bit(int x[],int y[]);

}

Đầu vào của chúng ta ở đây là x 48 bit(để lưu trữ dữ liệu 48 bit đầu ra) và y 32 bít(dữ liệu đầu vào của Ex).Lớp này có một thuộc tính là một mảng kiểu int E gồm 48 phần tử , tuy nhiên thì các phần từ có giá trị nằm trong khoảng từ 1 đến 32.

Thứ hai chúng ta xem lớp Kx được xây dựng thế nào.

public class Kx

{

private int PC1[] ={....};

private int PC2[] ={....};

private int LS[][] ={{.....},{.....}};// cai nay khong can lam

private int C[] = new int[28];

private int D[] = new int[28];

private int PC56[] = new int[56];

private int PC48[] = new int[48];

public int K48[] = new int[48];

public void chuyenthanh56bit(int x[],int y[]) // x la 64 bit,y la 56 bit

public void tach\_2\_nua\_28bit(int x[])// x o day 56 bit

public void quay(int x[],int y[],int j) // x, y la 28 bit, i la chi so vong lap

public void xac\_dinh\_khoa\_k(int x[],int y[])//x,y la dau vao 28 bit

public void khoa(int i,int y[]) // i chi so vong lap. ,y la khoa ban dau 64 bit

{

chuyenthanh56bit(y,PC56); // 56 bit da nam trong PC56

tach\_2\_nua\_28bit(PC56);// hai nua 28 bit da nam trong C va D

for(int j=0;j<=i;j++)

{

quay(C,D,j); // x, y la 28 bit, i la chi so vong lap

}

xac\_dinh\_khoa\_k(C,D);//khoa 48 bit da duoc luu vao K48

}

}

Lớp này có chức năng là xác định khóa con 48 bít cho mỗi vòng lặp,nó có nhiều vòng lặp như vậy nhưng thực ra thì phương thức cơ bản của nó là phương thức khoa(int i,int y[]),phương thức này có hai tham số đầu vào là int i là vòng lặp thứ i, và int y[] là khóa ban đầu 64 bit.Những phương thức còn lại của lớp Kx được xây dựng với mục đích cuối cùng là để xây dựng nên phương thức khoa này.

Cuối cùng chúng ta sẽ đi xem xem lớp Sx được xây dựng thế nào , về mặt ý nghĩa thì hộp S-Box này đóng một vai trò quan trọng cho độ an toàn của DES,nếu không có S-Box thì quá trình sẽ là tuyến tính và việc thám mã sẽ rất đơn giản.

public class Sx // lop nay nhan dau vao 48 bit di qua 8 hop S de tao thanh 32 bit

{

private int S1[][] = {{....},{....},{....},{....}};

private int S2[][] = {{....},{....},{....},{....}};

private int S3[][] = {{....},{....},{....},{....}};

private int S4[][] = {{....},{....},{....},{....}};

private int S5[][] = {{....},{....},{....},{....}};

private int S6[][] = {{....},{....},{....},{....}};

private int S7[][] = {{....},{....},{....},{....}};

private int S8[][] = {{....},{....},{....},{....}};

private int P[] ={....};

public void chuyen\_thanh\_nhiphan4bit(int x,int bit4[])

public void bit6\_chuyen\_thanh\_4bit(int x[],int y[],int z[][]) // x la 6 bit,y la 4 bit va z la cac S1,S2...S8

public void bit48\_thanh\_32bit(int x[],int y[]) // x la 48 bit,y la 32 bit

public void di\_qua\_hopP(int x[],int y[])// x,y deu la 32 bit

}

Đến đây chúng ta chú ý cách để chuyển từ 6 bit về 4 bit thông qua hộp S là như thế này :

public void bit6\_chuyen\_thanh\_4bit(int x[],int y[],int z[][])

{

int a,b,c;

a = x[5] + 2\*x[0]; // a sẽ có giá trị từ 0 đến 3

b = 8\*x[1]+4\*x[2]+2\*x[3]+x[4];// b sẽ có giá trị từ 0 đến 15

c= z[a][b];

chuyen\_thanh\_nhiphan4bit(c,y);

}

Sau khi đã có 3 lớp Kx,Sx và Ex chúng ta sẽ đi xây dựng lớp Fx như sau :

public class Fx // lop nay chinh la ham f

{

public int a[] = new int[48];

private int b[] = new int[48];

private int c[] = new int[32];

private int d[] = new int[32];

private Ex e1 = new Ex();

private Kx k1 = new Kx();

private Sx s1 = new Sx();

public void hien\_thi(String a,int x[])

public void xor\_48bit(int x[],int y[])// xor hai x,y 48 bit voi nhau

public void xac\_dinh\_fx\_tai\_vongi\_step\_step(int x[],int y[],int k[],int i)

}

Với lớp Fx này thì phương thức trọng tâm của nó là phương thức xac\_dinh\_fx\_tai\_vongi\_step\_step(int x[],int y[],int k[],int i)

Nó có nhiệm vụ tạo 32 bit đầu ra y với tham số đầu vào 32 bit x , khóa k 64 bít tại vòng lặp thứ i.

# 6.SỬ DỤNG JAVA DEMO CÀI ĐẶT THUẬT TOÁN TÌM MỌI KHÓA

## 6.1 Một vài khái niệm ban đầu

**Đầu tiên, chúng ta cần hiểu một vài khái niệm :**

Ta gọi :

* Q là tập cơ sở dữ liệu
* F là tập phụ thuộc hàm
* L(left) : là các thuộc tính chỉ xuất hiện bên trái
* R(right) : là các thuộc tính chỉ xuất hiện ở vế phải
* S(supperkey) : là tập các siêu khóa
* K(key) : là tập các khóa

**Tập thuộc tính nguồn (TN)** : bao gồm các thuộc tính chỉ xuất hiện ở vế trái, không xuất hiện ở vế phải của F( tập phụ thuộc hàm) và các thuộc tính không xuất hiện ở cả vế trái và vế phải của F.

**Vậy TN = Q \ R**

Nghĩa là ta lấy Q trừ cho R để tìm thuộc tính chỉ xuất hiện ở L và các thuộc tính không xuất hiện ở cả L và R

Ví dụ : Cho tập cơ sở dữ liệu Q = {A,B,C,D,E}    L = {A,B}    R = {B,C,E}

TN = Q \ R = {A,D}

**Tập thuộc tính đích (TĐ)** : Bao gồm các thuộc tính chỉ xuất hiện ở R, không xuất hiện ở L.

**Vậy TĐ = R \ L**

Ví dụ : Cho L = {A,B,C,D,E}     R = {E,F,G,H}

TĐ = {F,G,H}

**Tập thuộc tính trung gian (TG)** : Chứa các thuộc tính xuất hiện ở cả L và R

Vậy TG = L Giao R (Giao của 2 tập hợp để lấy thuộc tính chung của 2 Tập hợp đó)

Ví dụ : Cho L = {A,B,C,D,E}   R = {D,E,F,G}

Vậy TG = L /cap R = {D,E}

## 6.2 Thuật toán

**Bước 1 :**

Tìm tập thuộc tính nguồn TN và Tập thuộc tính trung gian TG, bằng các ví dụ ở trên thì các bạn có thể dễ dàng tìm thấy 2 tập thuộc tính này.

**Bước 2 :**

Nếu TG = 0

Thì K(Key) = TN, và kết thúc thuật toán, xuất ra K của tập cơ sở dữ liệu <Q,F>

Ngược lại, nếu TG # 0

Thì qua bước 3

**Bước 3 :**

Tìm tất cả các tập con Xicủa TG

**Bước 4 :**

Tìm Siêu khóa(Si) bằng cách với mọi Xi, nếu (TN U Xi)+= Q thì khi đó Si = TN U Xi

**Bước 5 :**

Tìm Khóa(Ki) bằng cách loại bỏ các siêu khóa không tối thiểu

Với mọi SiSj thuộc S

Nếu Sichứa trong Sjthì loại bỏ Sjra khỏi tập siêu khóa. Khi đó, tập S còn lại chính là tập khóa cần tìm

Ví dụ :

Ta có S = {AB, ABC, ED, EDF}

Ta thấy AB chứa trong ABC, ED chứa trong EDF vậy chúng ta cần phải loại bỏ ABC và EDF.

Vậy S = {AB,ED} chính là tập khóa cần tìm

*Chúng ta có một ví dụ mẫu như sau :*

Ví dụ : Cho một tập cơ sỡ dữ liệu R = <Q, F>

Với Q = {ABC}     F = {AB –> C, C -> A}. Tìm tất cả các khóa thuộc tập cơ sở dữ liệu trên.

Bài làm:

L = {ABC}      R = {CA}

TN = {B}        TG = {AC} # 0 nên ta làm tiếp bước 3

Ta có tập con Xicủa tập TG = {0, A,C,AC}

Ta lấy từng thuộc tính thuộc tập con Xi của tập TG hợp với TN ta có các thuộc tính sau :

S1= TN U 0 = B Ta có B+= B # Q nên S1 = A không là siêu khóa

S2= TN U A = AB Ta có AB+= ABC = Q nên S2 = AB là siêu khóa

S3 = TN U C = BC Ta có BC+= ABC = Q nên S3 = BC là siêu khóa

S4 = TN U AC = ABC Ta có ABC+= ABC = Q nên S4 = ABC  là siêu khóa

Vậy ta có tập siêu khóa S = {AB,BC,ABC}.

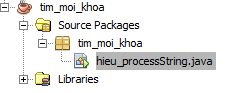
Tuy nhiên, vì AB chứa trong ABC và BC chứa trong ABC nên loại bỏ siêu khóa ABC ra khỏi tập siêu khóa

Vậy ta có, tập khóa K = {AB,BC} là khóa của lượt đồ quan hệ

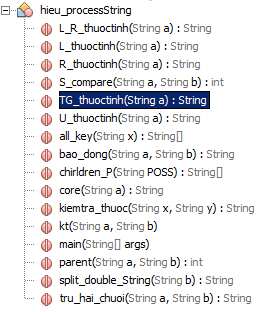
## 6.3 Chương trình DEMO

Chương trinh demo của chúng ta như sau :

Project của chúng ta chỉ bao gồm một lớp trong đó bao gồm cả hàm mail :



Trong lớp hieu\_processString.java này có những phương thức như sau :



Trong những phương thức trên chúng ta chú ý đến hai phương thức là main và all\_key, phương thức main là nơi để chạy chương trình,rõ ràng thì mã lệnh của chương trình , thuật toán của chương trình sẽ không được viết ở hàm này.Phương thức all\_key là phương thức thể hiện thuật toán của chương trình, những phương thức còn lại được xây dựng với một mục đích cuối cùng là để xây dựng nên phương thức này.

Code nguồn của phương thức all\_key được thể hiện như sau :

**public** **static** String[] all\_key(String x)

{

//đây là phương thức dung để giải quyết toan bộ bài toán

// những phương thức còn lại được để xây dựng nên phương thức này

// đầu vào là chuỗi a ,bao gồm các quan hệ,ví dụ x=ab.c,c.d,ad.e/g

// đầu ra là một tập các khóa ví dụ :abg,bc....

//giải thuật để giải quyết bài toán này đã có đầy rẫy trên mạng

//bước 1 : Tìm nhân của mọi khóa CORE

String core1 = core(x);

//bước 2 : Tìm bao đóng của core

String luutru[] = **new** String[100];

String luutru1[] = **new** String[100];

**for**(**int** i=0;i<100;i++)

{

luutru[i]=".";

luutru1[i]=".";

}

String bao\_dong\_core1 = bao\_dong(x,core1);

String S="";

**if**(S\_compare(bao\_dong\_core1,U\_thuoctinh(x))==1)

{

luutru1[0] = core1;

**return** luutru1;

}

**else**

{

String Poss =L\_R\_thuoctinh(x);

String chirld[]=chirldren\_P(Poss);

**int** j=0;

**for**(**int** i=0;i<chirld.length;i++)

{

String tg = chirld[i];

tg = tg + core1;

String baodong\_tg =bao\_dong(x,tg);

**if**(S\_compare(baodong\_tg,U\_thuoctinh(x))==1)

{

luutru[j] =tg;

j=j+1;

}

}

// xong vòng for này ta được mảng luutru đang luu trữ những siêu khóa của quan hệ

//từ tập siêu khóa này ta loại bỏ để thu được kết quả cuối cùng là tập các khóa

**for**(**int** i=0;i<luutru.length;i++)

{

**for**(**int** j1=0;j1<luutru.length;j1++)

{

**if**(parent(luutru[i],luutru[j1])==1)

luutru[i]=".";

}

}

**int** z=0;

**for**(**int** i=0;i<luutru.length;i++)

{

**if**(luutru[i]==".") **continue**;

**else**

{

luutru1[z]=luutru[i];

z=z+1;

}

} // bây giờ luutru1 chính là mảng chứa các khóa của quan hệ

**return** luutru1;

}

}

Trong hàm main chúng ta chỉ khởi tạo một đối tượng của lớp all\_key và tiến hành xử lý quan hệ là được :

**public** **static** **void** main(String[] args)

{

String x[]=all\_key("ae.c,cg.a,bd.g,ga.e/") ;//A->B, B->C, C->A

String a = "";

System.out.println("đầu vào là :ae.c,cg.a,bd.g,ga.e/");

**for**(**int** i=0;i<x.length;i++)

{

**if**(x[i]==".")

**continue**;

**else**

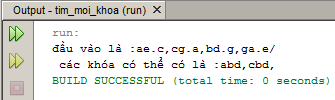
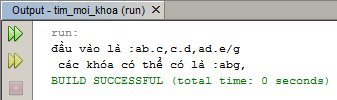
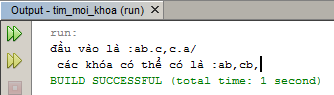
a =a + x[i]+",";

}

System.out.println(" các khóa có thể có là :"+a);

}

Bây giờ chúng ta thử tiến hành chạy thử một vài ví dụ và kết quả hiện thị như sau :



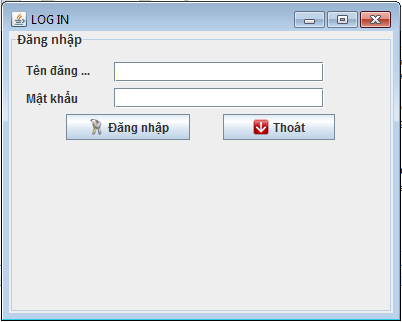
# 7.SỬ DỤNG JAVA DEMO PHẦN MỀM QUẢN LÝ NHÂN SỰ

## 7.1 Mục đích

* Chương trình được xây dựng bằng ngôn ngữ Java,với mục đích hỗ trợ công tác quản lý nhân sự trong một công ty,tiết kiệm thời gian tìm kiếm thông tin nhân viên cho những người quản lý
* Nhân sự được quản lý thông qua các thông tin trong hồ sơ của nhân viên bao gồm: mã nhân viên, tên đầy đủ, địa chỉ, phòng ban, dự án
* Người quản lý sẽ quản lý với tư cách là admin khi đăng nhập vào chương trình với các thông tin Username,Password.

## 7.2 Các chức năng chính của chương trình

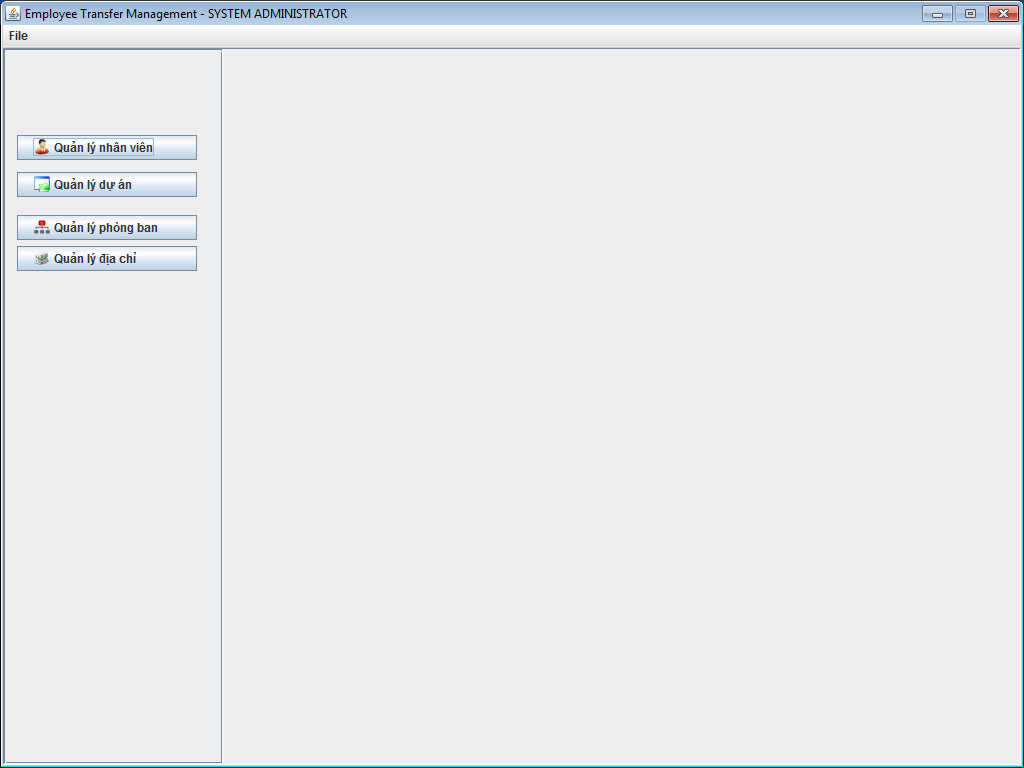
**Đăng nhập**



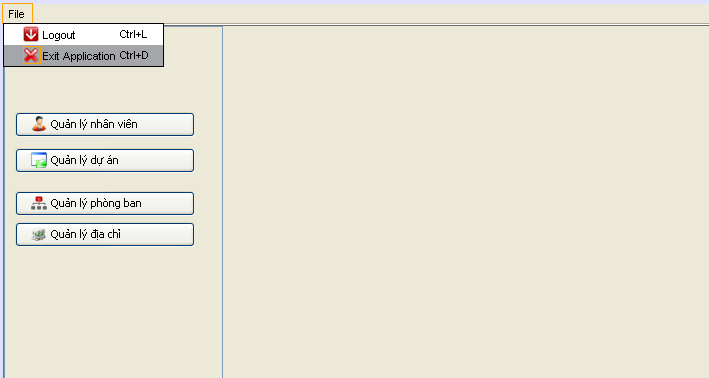
Tại đây người quản lý đăng nhập vào chương trình với tư cách là admin,từ đó cập nhật các thông tin liên quan đến nhân viên ở trong đó.

**Giao diện chính của chương trình**

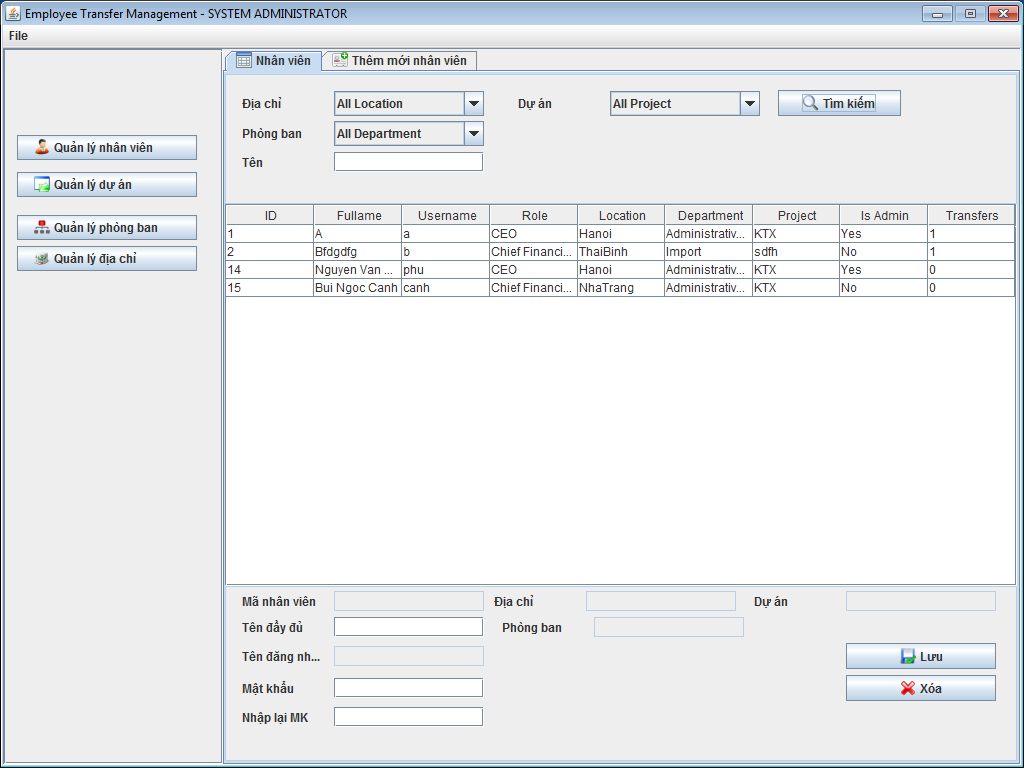
Sau khi đăng nhập thì 1 giao diện mới xuất hiện, đây là công cụ giúp các nhà quản lý thực hiện công tác quản lý nhân viên của mình, giao diện gồm các phần các chức năng riêng gồm có quản lý nhân viên, quản lý phòng ban, quản lý dự án, quản lý địa chỉ



Bạn có thể thoát khỏi chương trình khi vào File như hình dưới

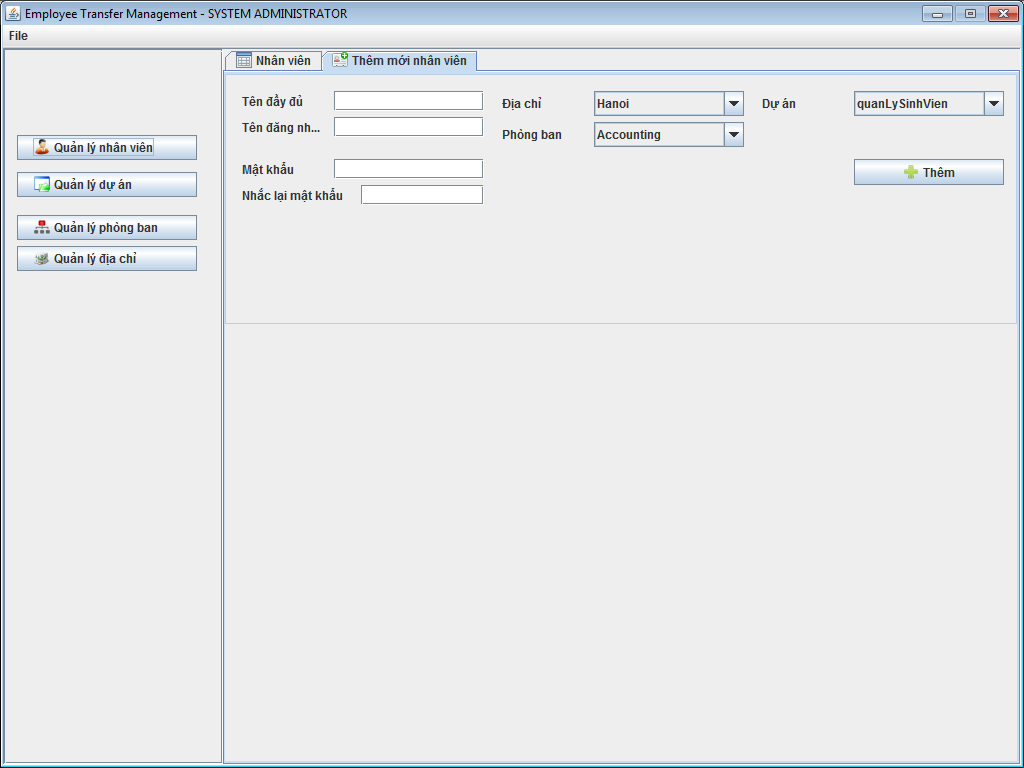


**Bảng nhân viên**

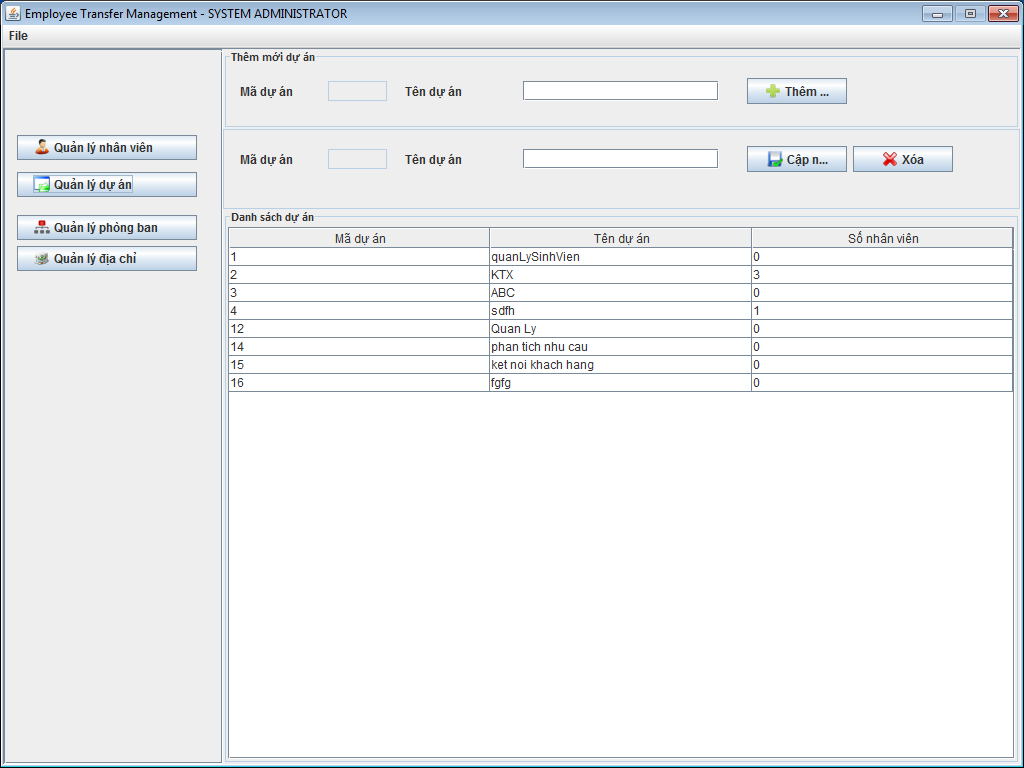


Đây là khi người quản lý muốn truy suất vào chương trình với mục đích tìm kiếm thông tin của nhân viên

Nếu người quản lý muốn cập nhật thông tin của 1 nhân viên mơi thì sẽ mở tab bên cạnh với giao diện giúp người dùng cập nhật thông tin của nhân viên mới

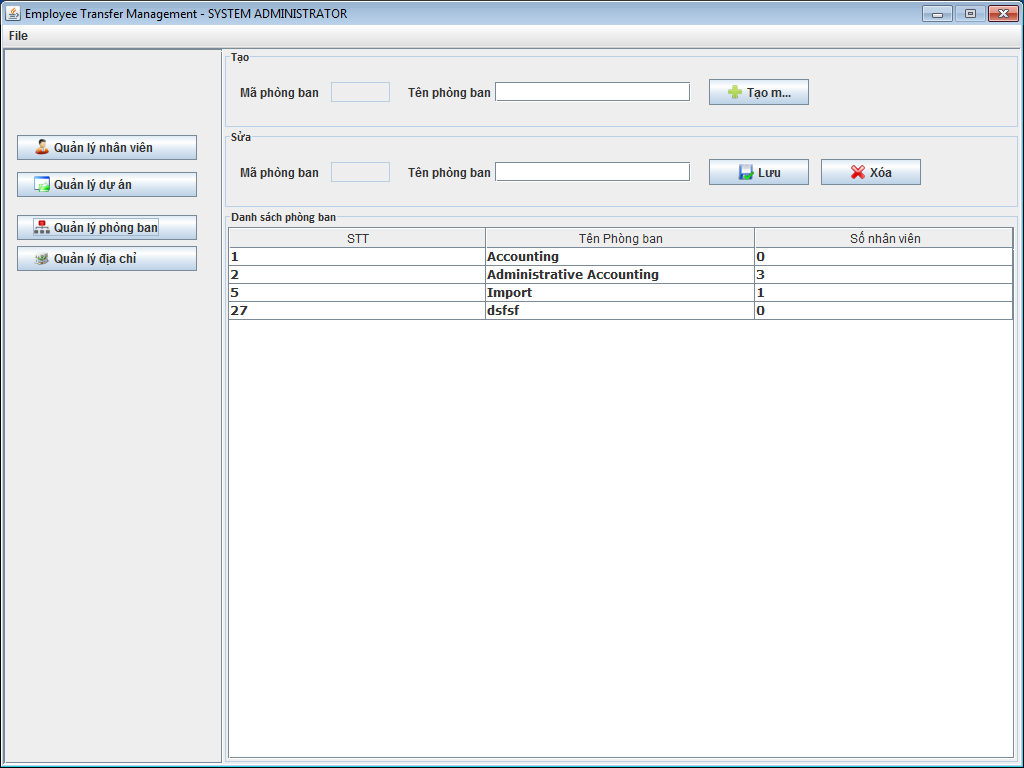


**Dự án**



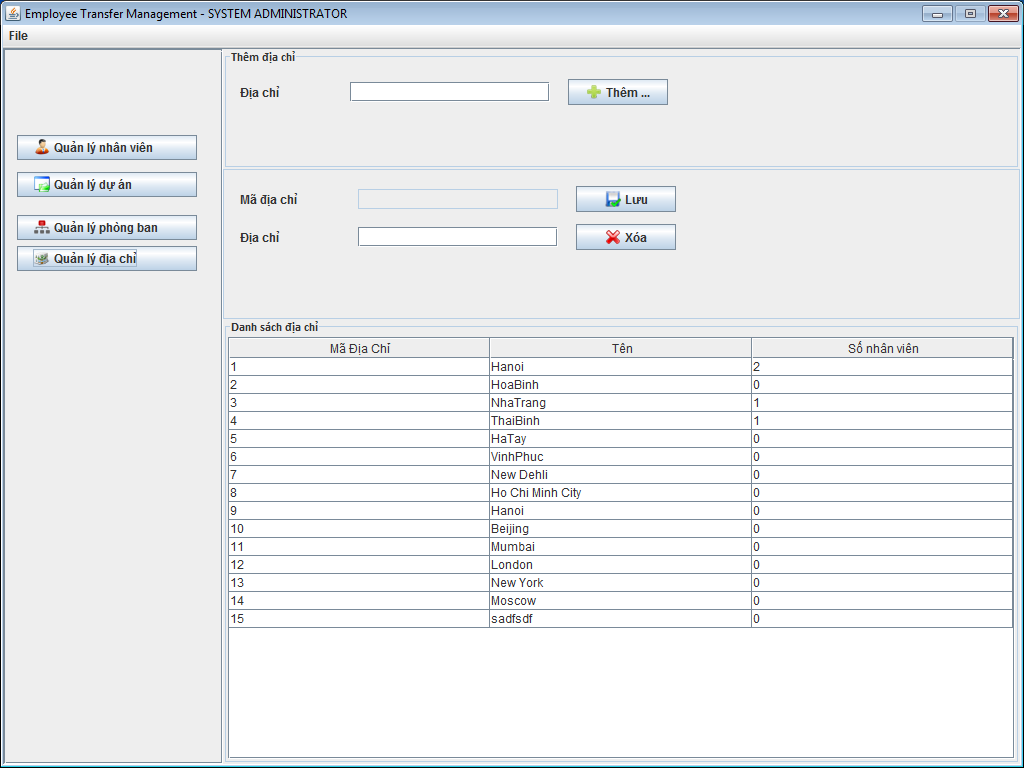
Trong 1 công ty số dự án có thể rất nhiều,vì thế việc quản lý nhân viên tham gia trong dự án đó cũng rất khó khăn và rắc rối,khi dùng chương trình này để quản lý người quản lý tiết kiệm rất nhiều thời gian trong việc truy suất thông tin của nhân viên, và có thể biết được dự án đó có bao nhiêu nhân viên đã tham gia

**Bảng phòng ban**



Khi truy suất đến bảng phòng ban,các thông tin về mã phòng tên phòng ban sẽ hiện ra dưới dạng nhập nhắm cập nhật thêm thông tin đồng thời xóa các thông tin không cần thiết

**Bảng địa chỉ**



Ở đây địa chỉ của nhân viên sẽ được cập nhật,giúp cho người quản lý nắm bắt cụ thể hơn thông tin của nhân viên