**Định nghĩa**

* Đệ quy là chủ đề phức tạp sẽ được trình bày cụ thể trong môn học cấu trúc dữ liệu và giải thuật. Ở đây ta chỉ xem xét đệ quy ở mức độ cơ bản nhất.
* Một phương thức đệ quy là phương thức gọi lại chính nó một cách trực tiếp hoặc gián tiếp thông qua các phương thức khác
* Mở rộng ra, một cách giải quyết gọi là đệ quy nếu trong các bước thực hiện để giải quyết vấn đề lại chứa chính bản thân nó.
* Ví dụ:

*// phương thức đệ quy tính n!*

public static long factorial(int n) {

    if (n <= 1) { *// trường hợp cơ sở*

return 1; *// trả về giá trị đơn giản nhất của bài toán*

} else { *// thực hiện lời gọi đệ quy*

return n \* *factorial*(n - 1);

    }

}

**Đặc điểm**

* Đệ quy thường áp dụng với các phương thức. Phương thức đệ quy thông thường chỉ biết cách giải quyết vấn đề ở trường hợp cơ bản nhất.
* Trường hợp này còn có tên gọi là trường hợp cơ sở, điểm dừng.
* Với các lời gọi trùng với trường hợp cơ sở thì phương thức đệ quy sẽ trả về giá trị của trường hợp cơ sở tương ứng.
* Khi phương thức đệ quy được gọi với trường hợp phức tạp hơn, phương thức đệ quy thường chia vấn đề thành hai phần: phần thứ nhất là phần mà phương thức này biết cách giải quyết; phần thứ hai phương thức này không biết cách giải quyết.
* Ở phần thứ hai này vấn đề cần xử lý có vẻ giống hệt vấn đề gốc ban đầu nhưng ở quy mô nhỏ hơn. Và để xử lý vấn đề thì phương thức đệ quy thực hiện lời gọi đến chính nó.
* Hành động gọi lại chính nó được gọi là lời gọi đệ quy, bước đệ quy.
* Phương thức chứa lời gọi đệ quy(thường đến chính nó) là phương thức đệ quy
* Khi thực hiện lời gọi đệ quy, chương trình sẽ phải chờ đến khi tất cả các bước đệ quy trong lời gọi trả về kết quả thì nó mới có kết quả để trả về cho nơi đã gọi nó.
* Để đảm bảo phương thức đệ quy có thể kết thúc được, sau mỗi lời gọi đệ quy vấn đề phải có xu hướng thu nhỏ lại, tiến dần đến trường hợp cơ sở, điểm dừng.
* Đệ quy cũng có thể lặp vô hạn nếu vấn đề không nhỏ hơn sau mỗi bước đệ quy.

**Ý nghĩa sử dụng**

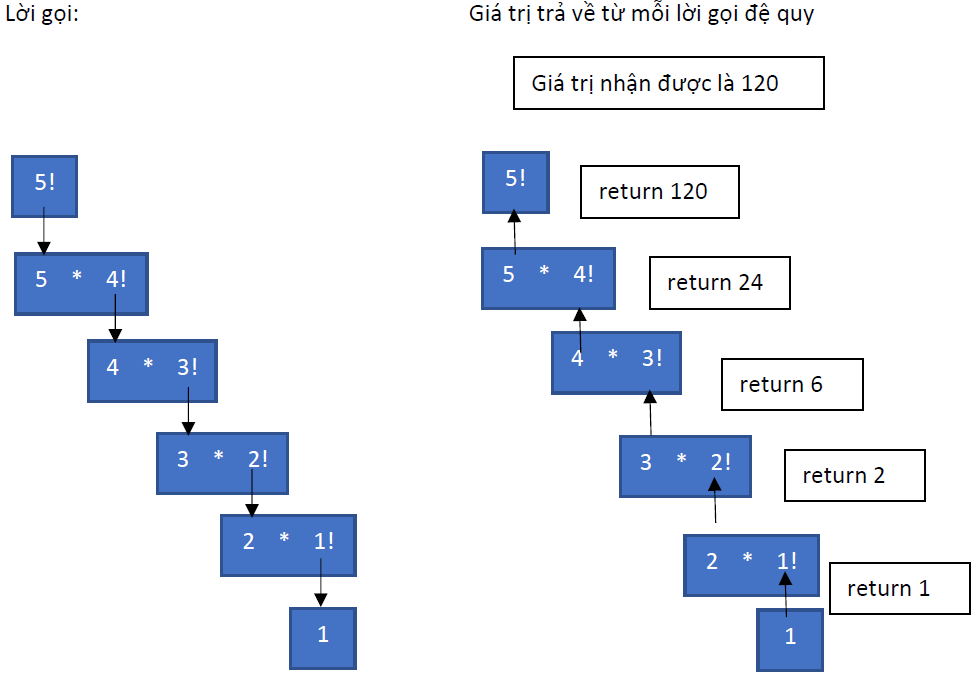
* Thường dùng phương thức đệ quy giải quyết vấn đề mang tính chất truy hồi.
* Vấn đề nào đó có thể giải quyết bằng đệ quy thì có thể giải quyết bằng vòng lặp.
* Cách giải quyết vấn đề bằng đệ quy ngắn gọn, tường minh hơn sử dụng vòng lặp
* Mỗi cách giải quyết vấn đề luôn có ưu điểm và nhược điểm riêng. Trong đó nhược điểm của đệ quy là tốn tài nguyên CPU, bộ nhớ và thời gian thực hiện.
* Lưu ý luôn đảm bảo vấn đề sẽ trở nên nhỏ hơn sau mỗi lời gọi đệ quy nếu không thì vấn đề sẽ không thể giải quyết được bằng đệ quy.

**Ví dụ minh họa**

**Ví dụ 1: Tính n!**

* Ví dụ tính 5! = 5 \* 4 \* 3 \* 2 \* 1 = 120.
* Định nghĩa đệ quy của n! là n! = n \* (n – 1)!, cụ thể:
  + 5! = 5 \* 4 \* 3 \* 2 \* 1
  + 5! = 5 \* (4 \* 3 \* 2 \* 1)
  + 5! = 5 \* (4!)
* Suy ra n! = n\*(n-1)!

Phân tích chi tiết:

[](https://braniumacademy.net/wp-content/uploads/2020/08/v%C3%AD-d%E1%BB%A5-%C4%91%E1%BB%87-quy.png)

import java.util.Scanner;

public class Example {

    public static void main(String[] args) {

        var input = new Scanner(System.*in*);

        System.*out*.println("Nhập n không quá 23: ");

        var n = input.nextInt();

        if(n < 0) {

            System.*out*.println("Không thể tính giai thừa của số âm");

        } else {

            var result = *factorial*(n);

            System.*out*.println(result);

        }

    }

*// phương thức đệ quy tính n!*

public static long factorial(int n) {

        if (n <= 1) { *// trường hợp cơ sở*

return 1; *// trả về giá trị đơn giản nhất của bài toán*

} else { *// thực hiện lời gọi đệ quy*

return n \* *factorial*(n - 1);

        }

    }

}

Kết quả

Nhập n không quá 23:

20

2432902008176640000

**Ví dụ 2:**

Tìm số fibonacci thứ n. Số Fibonacci được định nghĩa như sau: Số fibonacci là số cho theo công thức sau:  , từ n >= 2 thì  =  + .

* Phân tích: như vậy công thức tính số Fibonacci có thể đệ quy vì nó truy hồi dựa vào kết quả đã biết: Fn = fn-1 + fn-2. Điểm dừng là n = 0 hoặc n = 1.
* Code mẫu:

import java.util.Scanner;

public class Example {

    public static void main(String[] args) {

        var input = new Scanner(System.*in*);

        System.*out*.println("Nhập n >= 0 không quá 90: ");

        var n = input.nextInt();

        if (n < 0) {

            System.*out*.println("Không thể tính Fn với n âm");

        } else {

            var result = *fibonacci*(n);

            System.*out*.println("F" + n + " = " + result);

        }

    }

*// tìm số fibonacci dùng vòng lặp*

public static long fiboLoop(int n) {

   long f0 = 0;

   long f1 = 1;

   long fn = n;

   for (int i = 2; i <= n; i++) {

       fn = f0 + f1;

       f0 = f1;

       f1 = fn;

   }

   return fn;

}

}

Kết quả:

Nhập n >= 0 không quá 94:

40

F40 = 102334155