Recursividad en Java

En Java, un método puede llamarse a sí mismo. Este proceso se llama **recursividad** y se dice que **un método que se llama a sí mismo es recursivo**.

1. Qué es Recursividad

En general, la recursividad es el proceso de definir algo en términos de sí mismo y es algo similar a una definición circular. El componente clave de un método recursivo es una declaración que ejecuta una llamada a sí mismo. La recursividad es un poderoso mecanismo de control.

2. Ejemplo de recursividad

El ejemplo clásico de recursividad es el cálculo del factorial de un número. El factorial de un número N es el producto de todos los números enteros que se encuentren entre 1 y N. Por ejemplo, el factorial de 3 es 1 × 2 × 3, es decir, es 6.

El siguiente programa muestra una forma recursiva de calcular el factorial de un número. Para propósitos de comparación, también se incluye un equivalente no recursivo (iterativo).

```
// Un simple programa de recursividad
class Factorial
{
    // Esto es un método recursivo
    int facR (int n){
        int resultado;
        if (n==1) return 1;
        resultado=facR(n-1)*n;
        return resultado;
    }

    // Esto es un equivalente iterativo
    int facI (int n){
        int t, resultado;

        resultado=1;
        for (t=1; t<=n; t++) resultado *=t;
        return resultado;
    }
}

class Recursividad{
    public static void main(String[] args) {

        Factorial f= new Factorial();
```

```
        System.out.println("Factorial utilizando un método recursivo:");
        System.out.println("El factorial de 3 es: "+f.facR(3));
        System.out.println("El factorial de 6 es: "+f.facR(6));
        System.out.println();

        System.out.println("Factorial utilizando un método iterativo:");
        System.out.println("El factorial de 3 es: "+f.facI(3));
        System.out.println("El factorial de 6 es: "+f.facI(6));
        System.out.println();
    }
}
```

Salida:

```
Factorial utilizando un método recursivo:
El factorial de 3 es: 6
El factorial de 6 es: 720

Factorial utilizando un método iterativo:
El factorial de 3 es: 6
El factorial de 6 es: 720
```

3. Entendiendo la recursividad

La operación del método no recursivo **factI()** debe ser clara. Utiliza un bucle que comienza en 1 y multiplica progresivamente cada número por el nuevo producto.

La operación del **factR()** es un poco más compleja. Cuando se llama a **factR()** con un argumento de 1, el método devuelve 1; de lo contrario, devuelve el producto de **FacR(n-1)*n**.

Para evaluar esta expresión, se llama a **factR()** con *n-1*. Este proceso se repite hasta que n es igual a 1 y las llamadas al método comienzan a devolver. Por ejemplo, cuando se calcula el factorial de 2, la primera llamada a**factR()** hará que se realice una segunda llamada con un argumento de 1. Esta llamada devolverá 1, que luego se multiplicará por 2 (el valor original de *n*). La respuesta es 2. Puede encontrar interesante insertar instrucciones *println()* en **factR()** para mostrar a qué nivel está cada llamada y cuáles son los resultados intermedios.

Cuando un método se llama a sí mismo, a las nuevas variables y parámetros locales se les asigna almacenamiento en la pila (stack), y el código del método se ejecuta con estas nuevas variables desde el principio. **Una llamada recursiva no hace una nueva copia del método**. Solo los argumentos son nuevos. A medida que retorna o devuelve cada llamada recursiva, las viejas variables y parámetros locales se eliminan de la pila, y la ejecución se reanuda en el punto de la llamada dentro del método. Se podría decir que los métodos recursivos se “desplazan” hacia afuera y hacia atrás.

4. Desbordamiento de pila (stack overflow)

Las versiones recursivas de muchas rutinas pueden ejecutarse un poco más lentamente que sus equivalentes iterativos debido a la sobrecarga adicional de las llamadas a métodos adicionales. Demasiadas llamadas recursivas a un método podrían causar un **desbordamiento de la pila**.

Como el almacenamiento para los parámetros y las variables locales está en la pila y cada llamada nueva crea una nueva copia de estas variables, es posible que la pila se haya agotado. Si esto ocurre, el sistema de tiempo de ejecución (run-time) de Java causará una excepción. Sin embargo, probablemente no tendrás que preocuparte por esto a menos que una rutina recursiva se vuelva loca.

La principal ventaja de la recursividad es que algunos tipos de algoritmos se pueden implementar de forma más clara y más recursiva de lo que pueden ser iterativamente. Por ejemplo, el algoritmo de clasificación Quicksort es bastante difícil de implementar de forma iterativa. Además, algunos problemas, especialmente los relacionados con la IA, parecen prestarse a **soluciones recursivas**.

```
int fact(int n)
{
    // condición base equivocada (esto causa
    // stack overflow).
    if (n == 100)
        return 1;

    else
        return n*fact(n-1);
}
```

×**En el ejemplo anterior**: Si se llama a fact(10), llamará a fac (9), fact(8), fact(7), etc., pero el número nunca llegará a 100. Por lo tanto, no se alcanza la condición base. Si la memoria se agota con estos métodos en la pila, provocará un error de desbordamiento de pila.

Al escribir métodos recursivos, debe tener una instrucción condicional, como un if, en algún lugar para forzar el retorno del método sin que se ejecute la llamada recursiva. Si no lo hace, una vez que llame al método, nunca retornará. Este tipo de error es muy común cuando se trabaja con recursividad. Use las declaraciones *println()* generosamente para que pueda ver lo que está pasando y abortar la ejecución si ve que ha cometido un error. ¡Hasta pronto!