Capítulo V Recursión

Los errores están colgando en tus manos

Primero quisiera mostrarle un montaje introspectivo de la captura de pantalla para introducirnos al concepto de recursión. Tú puedes simular este fenómeno visual con un objeto puesto entre dos espejos reflejándose entre sí.

Objetivo

- Introducirte al concepto de recursión y a familiarizarte a ella a través de simples implementaciones

¿Qué es recursión?

¿Es un paradigma computacional? ¿Es recursión una estrategia algorítmica?, ¿un truco de todo programador@?, ¿una habilidad computacional?, es de todo un poco.

Recursión es cuando una función se ejecuta a sí misma, observemos el siguiente ejemplo:

Problema ejemplar

Digamos que quiero que implementes una multiplicación, pero no puedes usar el símbolo de multiplicación *. Puedes imaginar que tu mascota se comió esta tecla de multiplicar (tal vez se cansó de tragarse tu tarea). Sería tedioso, pero se puede implementar a través de una función recursiva.

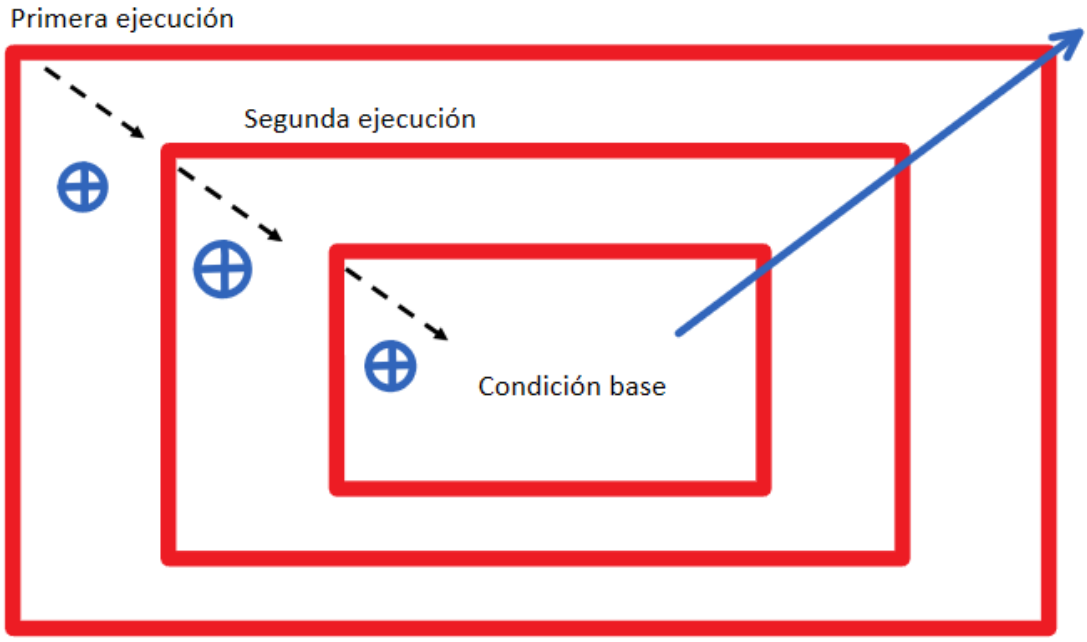
```
public static int Multiplicar(int a, int b){
        // condición base especificada; de lo contrario
        // la recursión se vuelve infinita (desborde de pila)

        if (a == 1){
            return b;
        }
        System.out.print(b + " + ");
        // empiezo de la recursión con una entrada
        // que decrece hacia la condición base.
        return b + Multiplicar(a - 1, b);
    }
```

Esquematización

Una función recursiva es definida de acuerdo al siguiente templete:

1. Especificación de una condición base
2. Empiezo de la recursión en donde el valor de la entrada decrece hacia la condición base

Debajo es una ilustración de recursión.



En teoría, toda función que hayas definido hasta ahora que tenga bucles (por o mientras) puede ser definida de manera recursiva.

Entre las razones para preferir recursión incluye:

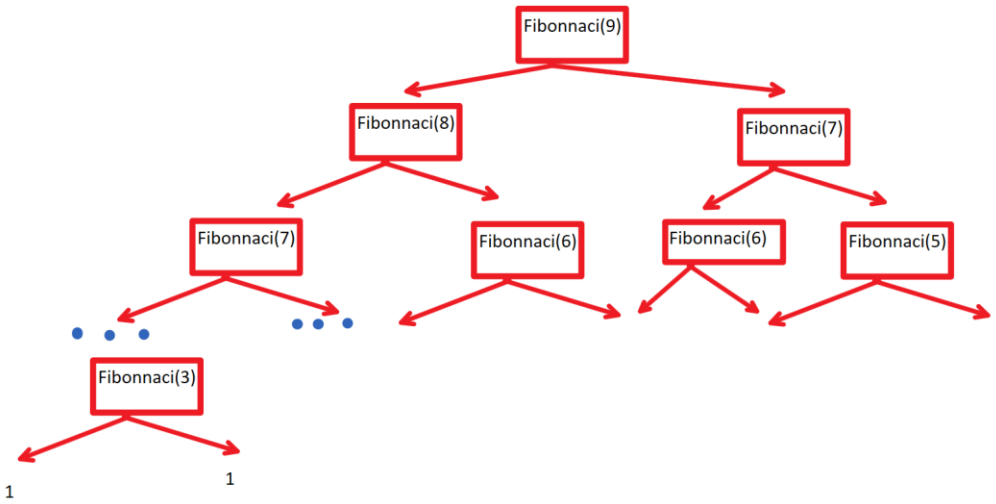
- Principalmente permite que tu código se simplifique y parezca más “elegante” (para que le saques pica a algun@ que otr@ colega”
- Permite el desarrollo del pensamiento computacional (hablaremos de esto en más profundidad en el interludio sobre la Paradoja de Russell y Teorema de Incompletitud de Gödel)
- Te ayuda a aprender nuevos lenguajes de programación como Haskell. Este lenguaje de programación NO TIENE BUCLES.
-
-

Desventajas de recursión

Aparte de ser difícil implementarla, recursión puede causar código redundante:

Considera la implementación recursiva de hallar el último número de una secuencia Fibonnaci.

```
public static int SecuenciaFibonnaci(int n){
        // condición base especificada; de lo contrario
        // la recursión se vuelve infinita (desborde de
pila)
        if (n == 1){
            System.out.print(n + ", ");
            return 1;
        }

        if (n == 2){
            System.out.print(n + ", ");
            return 1;
        }

        System.out.print(n + ", ");
        // empiezo de la recursión con una entrada
        // que decrece hacia la condición base.
                return   SecuenciaFibonnaci(n   -   1)   +
SecuenciaFibonnaci(n - 2);
    }
```



Puedes memoizar (no me tragué una r)

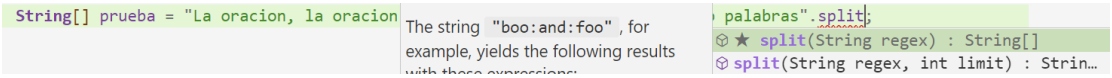
Memoizar es un estrategia computacional la cual mantienes un registro de qué funciones han sido ejecutadas para prevenir redundancia.

Práctica con lista vinculada

Regresa a la implementación de la lista vincular singular y considera implementar una versión recursiva de cualquier función que te interese. Por ejemplo, podemos crear una versión recursiva de la función de Acceder a un Nodo.

```
private int AccederRecursiva(int indice, Nodo siguiente){

      // ¿cómo podemos empezar refiriéndonos a la cabeza,
y al mismo tiempo
        // poder actualizarlo después de cada paso de
ejecución de la función
      // recursiva?

      if (indice == 0){
         return siguiente.valor;
      }

      // trasversar = trasversar.punteroSiguiente;
      // System.out.println(indice);


               return    AccederRecursiva(indice    -    1,
siguiente.punteroSiguiente);   //   AccederRecursiva   (--
i) != (i --)
     }
```

Observamos que si no especificamos un símbolo de referencia, el valor de límite es de manera predeterminada cero. ¿Cómo podemos hacer lo mismo con nuestra implementación de acceso recursivo?




```
String[] prueba = "La oracion, la oracion tiene cinco palabras, tiene cinco palabras".split(",");
```

Mantén presionado CTRL (o la tecla correspondiente de tu sistema operativo) y accede a la implementación de esta función.

```
public String[] split(String regex) {
        return split(regex, 0);
    }
```

Por ende, podemos hacer lo mismo con nuestra función recursiva:

```
public int AccederRecursiva(int indice){
      return AccederRecursiva(indice, this.cabeza);
     }
```

Con el fin de prevenir de que el usuario no pueda decidir desde que nodo empezar la recursión, podemos cambiar la función **AccederRecursiva** original de público a privado.

Conclusión

Bienvenidos al mundo de computación, en donde simples multiplicaciones nos ilustrarán varios bastantes principios informáticos.