Píldora Formativa: Recurrencia en Programación

1. ¿Qué es la Recurrencia?

La recurrencia es un concepto clave en programación que ocurre cuando una función se llama a sí misma para resolver un problema dividiéndolo en subproblemas más pequeños. Es como una tarea repetitiva donde cada paso depende del resultado del paso anterior.

Ejemplo cotidiano: Imagínate que quieres abrir una serie de cajas apiladas. Para abrir la última, primero necesitas abrir la caja de arriba. Este proceso se repite hasta llegar a la última caja (el caso base).

2. Estructura Básica de una Función Recursiva

Una función recursiva tiene dos partes fundamentales:

1. **Caso base:** La condición que detiene la recursión.
2. **Llamada recursiva:** La parte donde la función se llama a sí misma con un problema más pequeño.

Plantilla de código general:

```
public void funcionRecursiva(int parametro) {
    if (parametro <= 0) { // Caso base
        System.out.println("Fin de la recursión");

    }else{
    System.out.println("Llamada recursiva con: " + parametro);
    funcionRecursiva(parametro - 1); // Llamada recursiva

        }
}
```

3. Ejemplo Sencillo: Factorial de un Número

El factorial de un número (n!) se define como el producto de todos los enteros positivos desde 1 hasta n.

Fórmula recursiva:

- Caso base: 1! = 1
- Relación recursiva: n! = n × (n-1)!

Código en Java:

```
public int factorial(int n) {
```

```
    if (n == 0) { // Caso base
        return 1;
    }
    return n * factorial(n - 1); // Llamada recursiva
}
```

Trazado para factorial(3):

1. factorial(3) = 3 × factorial(2)
2. factorial(2) = 2 × factorial(1)
3. factorial(1) = 1 × factorial(0)
4. factorial(0) = 1 (caso base)

Resultado final: 3 × 2 × 1 = 6.

4. ¿Por qué es útil la Recurrencia?

- **Simplificación de problemas:** Permite resolver problemas complejos dividiéndolos en partes más pequeñas y manejables.
- **Casos típicos:** Cálculo de factoriales, Fibonacci, recorrer estructuras como árboles o grafos, resolver rompecabezas como Torres de Hanoi.

5. Ejemplo Básico Adicional: Suma de un Arreglo

Problema: Diseñar una función recursiva que sume los elementos de un arreglo.

Código en Java:

```
public int suma(int[] arr, int n) {
    if (n == 0) { // Caso base
        return 0;
    }
    return arr[n - 1] + suma(arr, n - 1); // Llamada recursiva
}
```

Trazado para arr = {1, 2, 3}, n = 3:

1. suma({1, 2, 3}, 3) = 3 + suma({1, 2, 3}, 2)
2. suma({1, 2, 3}, 2) = 2 + suma({1, 2, 3}, 1)
3. suma({1, 2, 3}, 1) = 1 + suma({1, 2, 3}, 0)
4. suma({1, 2, 3}, 0) = 0 (caso base)

Resultado final: 1 + 2 + 3 = 6.

6. Errores Comunes al Usar Recurrencia

1. **Olvidar el caso base:** Esto genera una recurrencia infinita y eventualmente un error de desbordamiento de pila (StackOverflowError).
2. **No reducir el problema:** Si el tamaño del problema no se reduce, la función nunca llegará al caso base.

Ejemplo de error:

```
public void errorRecursivo(int n) {
    // No hay caso base
    errorRecursivo(n);
}
```

7. Ejemplo

Diseña una función recursiva que realice una cuenta regresiva desde un número dado hasta 0, imprimiendo cada paso.

Ejemplo de ejecución: Entrada: 5 Salida:

```
5
4
3
2
1
0
```

Código inicial para resolver:

```
public void cuentaRegresiva(int n) {
    if (n < 0) { // Caso base
        System.out.println("Fin de la cuenta regresiva");

    }else{
    System.out.println(n);
    cuentaRegresiva(n - 1); // Llamada recursiva
        }
}
```