Asignación 10 - Depuración de Recursividad

Nombre:
Gibran Alonso Ibarra Palomares

Id:
00000267883

Fecha:

22/10/2025

Materia:
Estructura de Datos

Maestro:
Francisco Antonio Mejía Domínguez

INSTITUTO TECNOLÓGICO DE SONORA

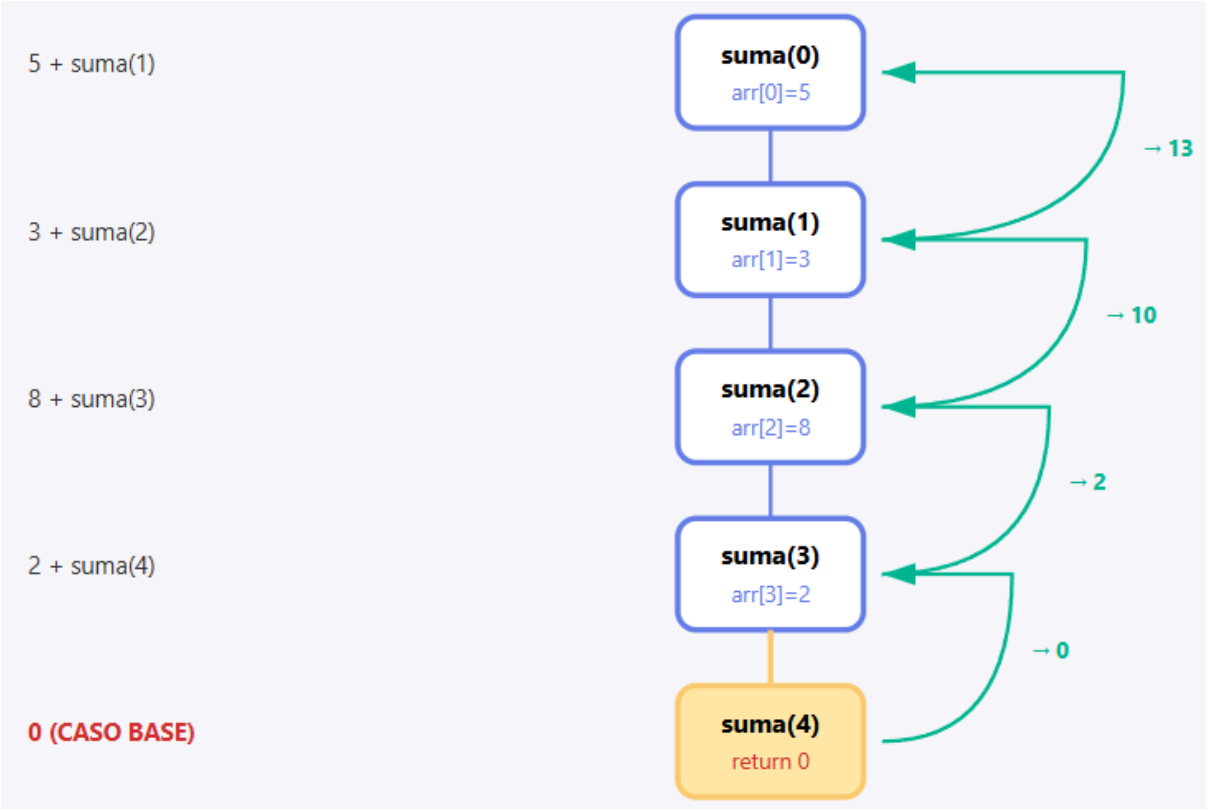
Ejercicio 1 — Suma de elementos de un arreglo

El código

```java
// ==========================================
// EJERCICIO 1: Suma de elementos de un arreglo
// ==========================================

/**
 * Version ITERATIVA: Suma de elementos de un arreglo.
 * Recorre el arreglo con un bucle acumulando la suma de todos los elementos.
 *
 * @param arr El arreglo de enteros a sumar.
 * @return La suma total de los elementos del arreglo.
 * @example sumaIterativa([1, 2, 3]) retorna 6
 */
public static int sumaIterativa(int[] arr) {
    int suma = 0;
    for (int i = 0; i < arr.length; i++) {
        suma += arr[i];
    }
    return suma;
}

/**
 * Version RECURSIVA: Suma de elementos de un arreglo.
 * CASO BASE: Si el indice esta fuera del rango del arreglo (index >= arr.length), retorna 0.
 * CASO RECURSIVO: Suma el elemento actual (arr[index]) con la suma del resto del arreglo.
 *
 * @param arr El arreglo de enteros a sumar.
 * @param index El indice actual desde donde empezar la suma.
 * @return La suma total de los elementos desde el indice dado hasta el final.
 * @example sumaRecursiva([1, 2, 3], 0) retorna 6
 */
public static int sumaRecursiva(int[] arr, int index) {
    // CASO BASE: si el indice esta fuera de rango
    if (index >= arr.length) {
        return 0;
    }
    // CASO RECURSIVO: elemento actual + suma del resto
    return arr[index] + sumaRecursiva(arr, index + 1);
}

/**
 * Sobrecarga para facilitar la llamada a sumaRecursiva, empezando desde el indice 0.
 *
 * @param arr El arreglo de enteros a sumar.
 * @return La suma total de todos los elementos del arreglo.
 */
public static int sumaRecursiva(int[] arr) {
    return sumaRecursiva(arr, 0);
}
```

Árbol de recursión para sumaRecursiva([5, 3, 8, 2], 0)

Cuando llamamos sumaRecursiva([5, 3, 8, 2], 0), la computadora genera este árbol de llamadas:



Traza de ejecución paso a paso

Lee el siguiente flujo de arriba hacia abajo:

FASE 1: Bajando (haciendo llamadas)

Paso 1: Llamada → sumaRecursiva(arr, 0)
Paso 2: Llamada → sumaRecursiva(arr, 1)
Paso 3: Llamada → sumaRecursiva(arr, 2)
Paso 4: Llamada → sumaRecursiva(arr, 3)
Paso 5: Llamada → sumaRecursiva(arr, 4) (caso base)

FASE 2: Retornando (obteniendo resultados)

Paso 6: sumaRecursiva(arr, 4) retorna 0
Paso 7: sumaRecursiva(arr, 3) calcula: 2 + 0 = 2 y retorna
Paso 8: sumaRecursiva(arr, 2) calcula: 8 + 2 = 10 y retorna
Paso 9: sumaRecursiva(arr, 1) calcula: 3 + 10 = 13 y retorna
Paso 10: sumaRecursiva(arr, 0) calcula: 5 + 13 = 18 y retorna

Resultado final: 18

Pila de ejecución (Call Stack)

La pila es como un apilador de platos: cada llamada nueva va al tope, y cuando una función termina, se quita de la pila.

Momento 1: Máxima profundidad (antes del primer caso base)

Pila:

`suma(4)` ← Tope
`suma(3)`
`suma(2)`
`suma(1)`
`suma(0)` ← Base

Momento 2: Después de resolver suma(4) = 0

Pila:

`suma(3)` ← Tope
`suma(2)`
`suma(1)`
`suma(0)`

suma(4) se eliminó

Resultado: 0

Momento 3: Finalmente, todo resuelto

Pila:

[Pila vacía]

Resultado obtenido: 18

Cada vez que una función termina, se elimina de la pila. Si la pila se llena demasiado, obtenemos un error llamado **StackOverflow**

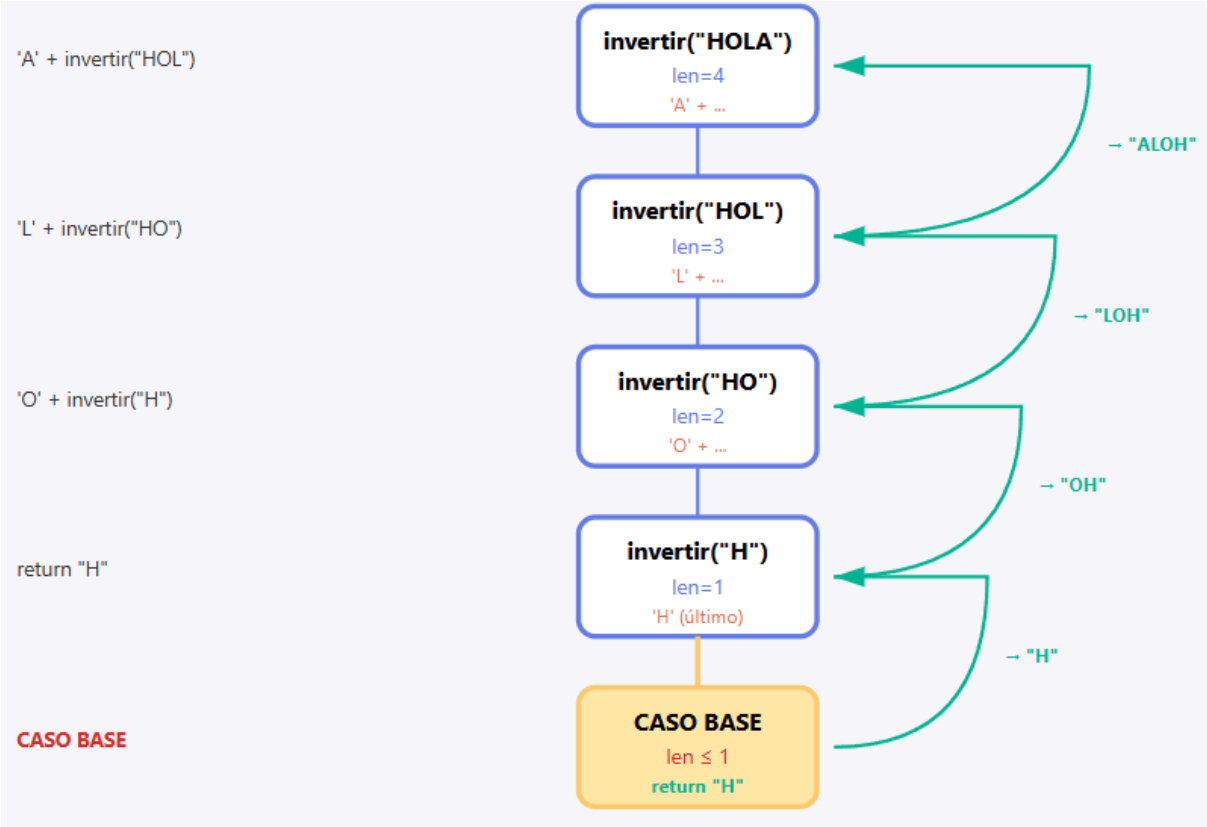
Ejercicio 2 — Invertir una cadena

El código

```java
// ==========================================
// EJERCICIO 2: Contar ocurrencias
// ==========================================

/**
 * Version ITERATIVA: Contar ocurrencias de un valor x en un arreglo.
 * Recorre el arreglo con un bucle contando cuantas veces aparece x.
 *
 * @param arr El arreglo de enteros donde buscar.
 * @param x El valor a contar.
 * @return El numero de veces que x aparece en el arreglo.
 * @example contarIterativo([1, 2, 2, 3], 2) retorna 2
 */
public static int contarIterativo(int[] arr, int x) {
    int count = 0;
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] == x) {
            count++;
        }
    }
    return count;
}

/**
 * Version RECURSIVA: Contar ocurrencias de un valor x en un arreglo.
 * CASO BASE: Si el indice esta fuera del rango (index >= arr.length), retorna 0.
 * CASO RECURSIVO: Si el elemento actual es igual a x, suma 1; de lo contrario, suma 0;
 *                 mas el conteo del resto del arreglo.
 *
 * @param arr El arreglo de enteros donde buscar.
 * @param x El valor a contar.
 * @param index El indice actual desde donde empezar el conteo.
 * @return El numero de veces que x aparece desde el indice dado hasta el final.
 * @example contarRecursivo([1, 2, 2, 3], 2, 0) retorna 2
 */
public static int contarRecursivo(int[] arr, int x, int index) {
    // CASO BASE: indice fuera de rango
    if (index >= arr.length) {
        return 0;
    }
    // CASO RECURSIVO: suma 1 si coincide, 0 si no + recursion en el resto
    int coincide = (arr[index] == x) ? 1 : 0;
    return coincide + contarRecursivo(arr, x, index + 1);
}

/**
 * Sobrecarga para facilitar la llamada a contarRecursivo, empezando desde el indice 0.
 *
 * @param arr El arreglo de enteros donde buscar.
 * @param x El valor a contar.
 * @return El numero total de veces que x aparece en el arreglo.
 */
public static int contarRecursivo(int[] arr, int x) {
    return contarRecursivo(arr, x, 0);
}
```

Árbol de recursión para invertirRecursivo("HOLA")

Cuando llamamos **invertirRecursivo("HOLA")**, la computadora genera este árbol de llamadas:



Traza de ejecución paso a paso

Lee el siguiente flujo de arriba hacia abajo:

FASE 1: Bajando (haciendo llamadas)

Paso 1: Llamada → invertirRecursivo("HOLA")
Paso 2: Llamada → invertirRecursivo("HOL")
Paso 3: Llamada → invertirRecursivo("HO")
Paso 4: Llamada → invertirRecursivo("H") (caso base)

FASE 2: Retornando (obteniendo resultados)

Paso 5: invertirRecursivo("H") retorna "H"
Paso 6: invertirRecursivo("HO") calcula: 'O' + "H" = "OH" y retorna
Paso 7: invertirRecursivo("HOL") calcula: 'L' + "OH" = "LOH" y retorna
Paso 8: invertirRecursivo("HOLA") calcula: 'A' + "LOH" = "ALOH" y retorna

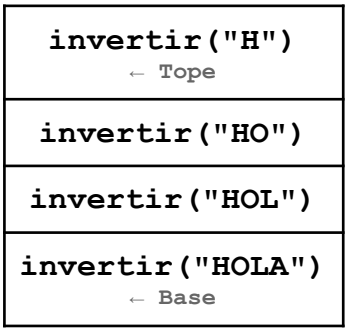
Resultado final: "ALOH"

Pila de ejecución (Call Stack)

La pila es como un apilador de platos: cada llamada nueva va al tope, y cuando una función termina, se quita de la pila.

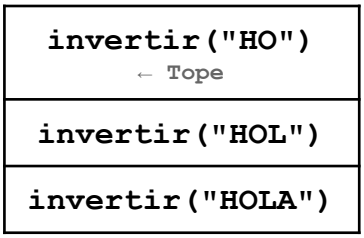
Momento 1: Máxima profundidad (antes del primer caso base)

Pila:



Momento 2: Después de resolver invertir("H") = "H"

Pila:



invertir("H") se eliminó

Resultado: "H**"**

Momento 3: Finalmente, todo resuelto

Pila:

[Pila vacía]

Resultado obtenido: "ALOH"

Cada vez que una función termina, se elimina de la pila. Si la pila se llena demasiado, obtenemos un error llamado **StackOverflow**.