Ejercicios de Recursividad en Java

A continuación se presentan 6 ejercicios típicos de recursividad para que los alumnos practiquen. Cada ejercicio incluye enunciado, firma del método, explicación clara de la recursividad, ejemplos y pistas.

Ejercicio 1 — Suma recursiva de elementos de un array

Firma sugerida:

```java
public static int sumaArray(int[] a, int i)
```

Enunciado:\
Implementa un método recursivo que calcule la suma de todos los elementos de un array de enteros, empezando desde el índice `i`.

Explicación:

- Caso base: si `i` es igual o mayor que la longitud del array, no quedan elementos por sumar, devuelve 0.
- Caso recursivo: suma el elemento actual `a[i]` más la suma de los elementos desde `i+1`.
- Cada llamada avanza el índice `i` en 1, garantizando que la recursión termina.

Ejemplo:\
Entrada: `a = [1, 2, 3, 4]`, llamada inicial `sumaArray(a, 0)` \
Salida: `10`

Ejercicio 2 — Potencia rápida (pow) usando recursividad

Firma sugerida:

```java
public static long potencia(long x, long n)
```

Enunciado:\
Implementa un método recursivo que calcule `x` elevado a la potencia `n` (con `n` entero no negativo) usando la técnica de división y conquista para lograr eficiencia O(log n).

Explicación:

- Caso base: si `n == 0`, devuelve 1.
- Caso recursivo: calcula `potencia(x, n/2)` y usa ese resultado para calcular la potencia completa.
- Si `n` es par, el resultado es `half * half`.
- Si `n` es impar, el resultado es `x * half * half`.
- Cada llamada reduce el exponente a la mitad, garantizando que la recursión termina rápido.

Ejemplo:\
Entrada: `x=2, n=10` \
Salida: `1024`

Ejercicio 3 — Búsqueda binaria recursiva

Firma sugerida:

```
public static int busquedaBinaria(int[] a, int izq, int der, int clave)
```

Enunciado:\
Implementa un método recursivo que busque una clave en un array ordenado y devuelva su índice o -1 si no está.

Explicación:

- Caso base: si `izq > der`, la clave no está, devuelve -1.
- Caso recursivo: calcula el índice medio `mid`.
  - Si `a[mid] == clave`, devuelve `mid`.
  - Si `clave < a[mid]`, busca en la mitad izquierda.
  - Si `clave > a[mid]`, busca en la mitad derecha.
- Cada llamada reduce el rango de búsqueda a la mitad, garantizando que la recursión termina.

Ejemplo:\
Entrada: `a = [1,3,5,7,9], clave=7`\
Salida: `3`

Ejercicio 4 — Fibonacci: recursiva y memoización

Firmas sugeridas:

```
public static long fibRec(int n)
public static long fibMemo(int n, long[] memo)
```

Enunciado:

1. Implementa la versión recursiva simple de Fibonacci.
2. Implementa la versión con memoización para evitar cálculos repetidos.
3. Compara el número de llamadas recursivas entre ambas versiones.

Explicación:

- Caso base: `fib(0) = 0`, `fib(1) = 1`.
- Caso recursivo: `fib(n) = fib(n-1) + fib(n-2)`.
- La versión simple recalcula muchas veces los mismos valores.
- La versión memoizada guarda resultados ya calculados para evitar redundancia.
- La recursión termina porque `n` decrece hasta llegar a 0 o 1.

Ejemplo:\
Entrada: `n=5`\
Salida: `fibRec(5) = 5`, `fibMemo(5) = 5`

Ejercicio 5 — Permutaciones de una cadena (backtracking)

Firma sugerida:

```
public static void permutar(String s, int l, int r)
```

Enunciado:\
Genera todas las permutaciones posibles de una cadena de caracteres usando recursividad y backtracking.

Explicación:

- Caso base: si `l == r`, se ha generado una permutación completa, imprimirla.
- Caso recursivo: para cada índice `i` entre `l` y `r`, intercambia caracteres en posiciones `l` e `i`, llama recursivamente con `l+1`, y luego deshace el intercambio (backtrack).
- La recursión termina porque `l` avanza hasta `r`.

Ejemplo:\
Entrada: `"ABC"`\
Salida:

```
ABC
ACB
BAC
BCA
CAB
CBA
```

Ejercicio 6 — Torres de Hanoi (imprimir movimientos)

Firma sugerida:

```
public static void hanoi(int n, char origen, char auxiliar, char destino)
```

Enunciado:\
Imprime la secuencia de movimientos para resolver las Torres de Hanoi con `n` discos, moviéndolos de la torre `origen` a la torre `destino` usando la torre `auxiliar`.

Explicación:

- Caso base: si `n == 1`, mueve el disco directamente de `origen` a `destino`.
- Caso recursivo:
    1. Mueve `n-1` discos de `origen` a `auxiliar`.
    2. Mueve el disco más grande de `origen` a `destino`.

3. Mueve `n-1` discos de `auxiliar` a `destino`.

- La recursión termina porque `n` decrece hasta 1.

Ejemplo:\
Entrada: `n=2`\
Salida:

```
A -> B
A -> C
B -> C
Total movimientos: 3
```

Recomendaciones para la entrega

- Implementa cada ejercicio en un método estático en Java.
- Añade comentarios explicando el caso base y el caso recursivo.
- Prueba con varios casos, incluyendo casos límite (arrays vacíos, n=0, etc.).
- No uses bucles para resolver los ejercicios que piden recursividad.
- Sube el código fuente java con las soluciones.
