Programación de sistemas

Recursión

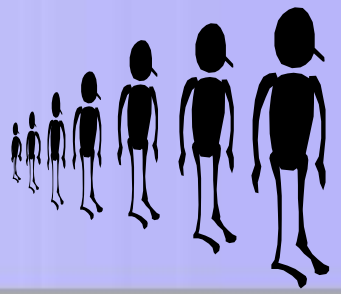
Julio Villena Román
<jvillena@it.uc3m.es>

MATERIALES BASADOS EN EL TRABAJO DE DIFERENTES AUTORES:
Carlos Delgado Kloos, Carlos Alario Hoyos

Contenidos

- Recursión
- Tipos de recursión
- Recursión vs. iteración

Recursión



- Proceso por el cual un **método se llama a sí mismo** de forma continuada hasta que se llegue a un determinado estado
  - ✓ **Caso recursivo:** El método se invoca a sí mismo
  - ✓ **Caso base:** Permite salir de la recursión
- Un método recursivo debe tener un caso recursivo y al menos un caso base
  - ➢ El caso recursivo debe aproximarse al caso base
- Los métodos recursivos utilizan sentencias más sencillas para simplificar problemas complejos

Ejemplo 1

- **Sumatorio de 1 a n (con n > 0)**

```
public static long s (int n){
    if (n==1){
        return 1;
    }
    else{
        return s(n-1)+n;
    }
}
```

Caso base

Caso recursivo

Ejemplo 1

- **Sumatorio de 1 a n (con n > 0)**

```
s(3) =
s(2)+3 =
(s(1)+2)+3 =
(1+2)+3 =
(3+3)=
6
```

```
public static long s (int n){
    if (n==1){
        return 1;
    }
    else{
        return s(n-1)+n;
    }
}
```

- El caso recursivo sucede dos veces => s(3) y s(2)
- El caso base sucede una vez => s(1)

Ejemplo 2

*(si n > 1)

- **Recursión mal diseñada***

```
public static long s (int n){
    if (n==1){
        return 1;
    } else {
        return s(n+1)+n;
    }
}
```

Caso base

Caso recursivo

Ejemplo 2

*(si n > 1)

- **Recursión mal diseñada***
  - `s(3) =`
  - `s(4)+3 =`
  - `(s(5)+4)+3 =`
  - `((s(6)+5)+4)+3 =`
  - `(((s(7)+6)+5)+4)+3 =`
  - …
  - (no termina!!)

```
public static long s (int n){
   if (n==1){
      return 1;
   } else {
      return s(n+1)+n;
   }
}
```

- El caso recursivo sucede “infinitas” veces
- El caso base no sucede nunca
- **StackOverflowError** (desbordamiento de la pila de memoria)

Ejemplo 3

- **Sumatorio de n a 1000**

```
public static long s (int n){
    if (n==1000){
        return 1000;
    }
    else if (n>1000){
        return 0;
    }
    else{
        return s(n+1)+n;
    }
}
```

Casos base

Caso recursivo

Ejemplo 4

- **Cuenta atrás (con contador > 0)**

```
public static void cuentaAtras(int contador){
    if(contador == 0) {
        return;
    }
    else {
        System.out.println(""+contador);
        cuentaAtras(--contador);
        return;
    }
}
```

Caso base

Caso recursivo

Ejercicio 1

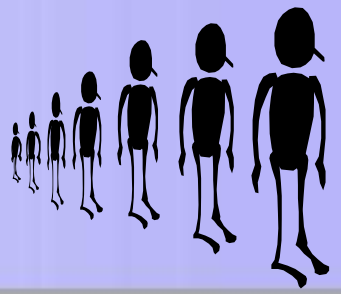
- Implementa el método recursivo `square(int n)`, el cual permite calcular el **cuadrado de n** (cuando n > 1). Para ello puedes tener en cuenta la siguiente fórmula:

$$(N-1)^2 = N^2 - 2N + 1$$

Ejercicio 2

- Implementa el método recursivo `riddle(int p, int q)` con p y q > 0. Cada vez que se invoca este método, el valor de `q` incrementa en una unidad, mientras que el valor de `p` decrementa en una unidad. Al llegar `p` a cero, el método devuelve el valor de `q`.
- ¿Cuántas veces se ejecuta el caso recursivo y el caso base al llamar a `riddle(3,5)` y cuál es el valor de retorno?

Tipos de recursión



- **Recursión lineal**
  - Máximo una llamada recursiva por caso recursivo
  - "Recursión no por la cola" (*non-tail recursion*)
    - Hay que llegar al caso base para obtener el primer resultado. Entonces comienza el calculo del resultado final
    - Computacionalmente costoso
  - "Recursión por la cola" (*tail recursion*)
    - El resultado final se obtiene al llegar al caso base (uso de un acumulador)
    - Menor coste computacional

Recursión lineal no por la cola

- **Cálculo del factorial**

fac(n) = n! = n*(n-1)*(n-2)…2*1 (con n >= 0)

```
public static long fac(int n){
    if (n<=1){
        return 1;
    }
    else{
        return n*fac(n-1);
    }
}
```

fac (4) =
4*fac(3) =
4*3*fac(2) =
4*3*2*fac(1) =
4*3*(2*1) =
4*(3*2) =
4*6 =
24

```
public static void main (String[] args){
    System.out.println(fac(4));
}
```

Recursión lineal por la cola

- **Cálculo del factorial**

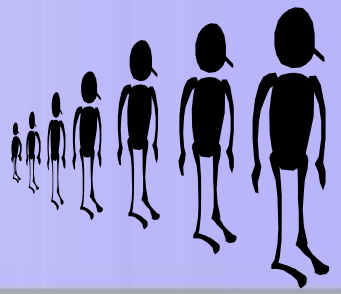
fac(n) = n! = n*(n-1)*(n-2)…2*1 (con n >= 0)

```
public static long fac(int n, int m){
    if (n<=1){
        return m;
    }
    else{
        return fac(n-1, n*m);
    }
}
```

fac (4) =
fac(4,1) =
fac(3,4) =
fac(2,12) =
fac(1,24) =
24

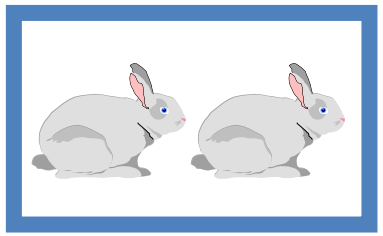
```
public static void main (String[] args){
    System.out.println(fac(4,1));
}
```

Tipos de recursión

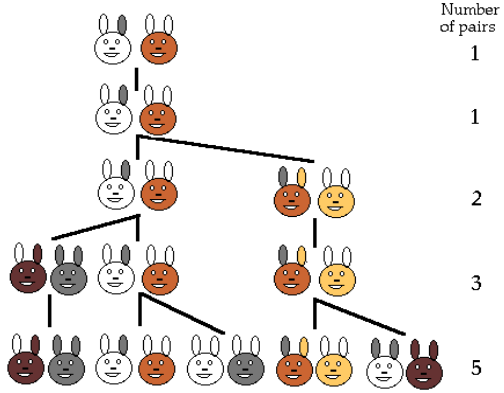


- **Recursión no lineal**
  - Varias llamadas recursivas por caso recursivo
  - Recursión en cascada
    - P.ej. `return método(n) + método(n-1);`
  - Recursión anidada
    - P.ej. `return método(n, método(n-1));`

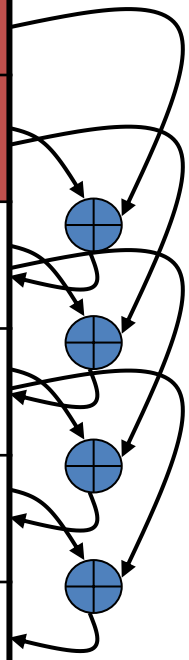
Recursión en cascada: Fibonacci



```
fib(5)=
fib(4)+fib(3)= 5+3
```



n	fib(n)
0	1
1	1
2	2
3	3
4	5
5	8
...	...



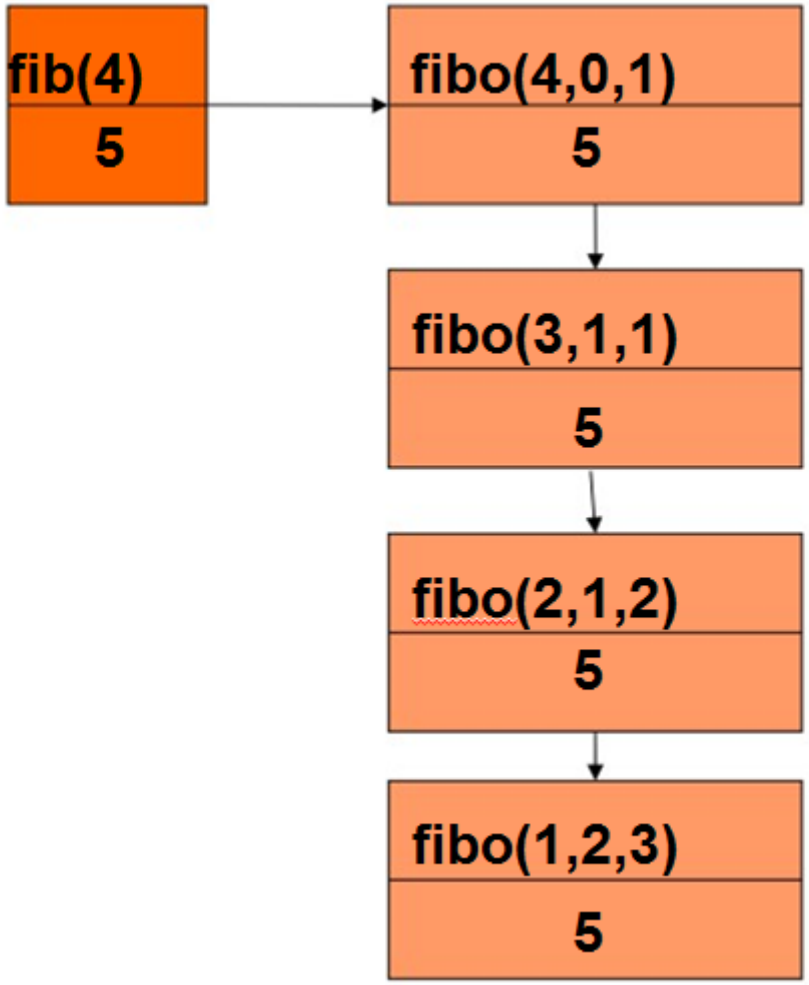
Recursión en cascada: Fibonacci

- **Serie de Fibonacci (con n > 0)**

```
public static long fib(int n){
   if (n<=1){
      return 1;
   }
   else{
      return fib(n-1)+fib(n-2);
   }
}
```

Recursión en cascada: Fibonacci

- **Conversión a recursión lineal de Fibonacci**

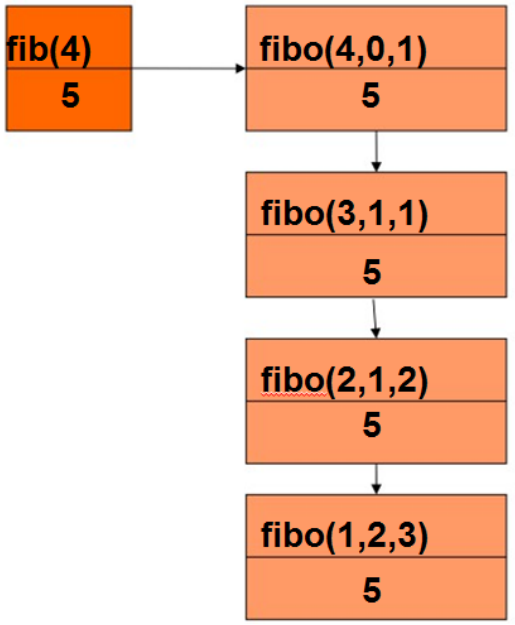


Recursión en cascada: Fibonacci

- **Conversión a recursión lineal de Fibonacci**

```
public static long fibo(int n, int x, int y){
   if (n<=1){
      return x+y;
   }
   else {
      return fibo(n-1, y, x+y);
   }
}

public static long fib (int n){
   return fibo(n,0,1);
}
```



Fibonacci no recursivo

$$fib(n) = \frac{(1+\sqrt{5})^{n+1} - (1-\sqrt{5})^{n+1}}{(2^{n+1} \cdot \sqrt{5})}$$

Recursión anidada: Morris

```
public static int mor(int n, int m){
   if (n==m){
      return (m+1);
   }
   else{
      return(mor(n, mor(n-1, m+1)));
   }
}
```

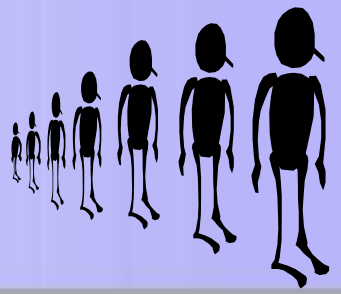
- mor(4,0) =
- mor(4, mor(3,1)) =
- mor(4, mor(3, mor(2,2)) =
- mor(4, mor(3,3)) =
- mor(4,4) =
- 5

Ejercicio 3

- Implementa el método recursivo **`a(int m, int n)`** que permite calcular la función de *Ackermann-Peter* tal y como se define a continuación:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1,A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases}$$

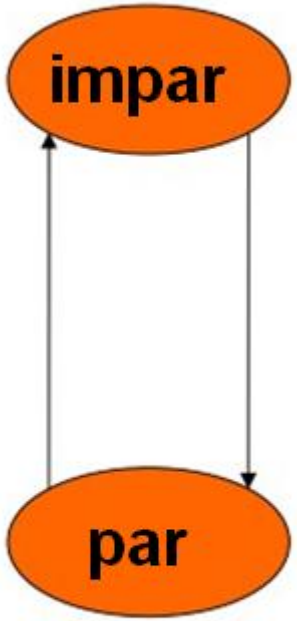
Tipos de recursión



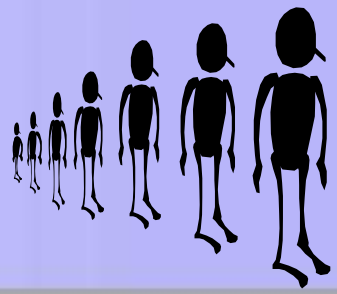
- **Recursión mutua**

```
public static boolean par(int n){
    if (n==0){
        return true;
    }
    else{
        return impar(n-1);
    }
}

public static boolean impar(int n){
    if (n==0){
        return false;
    }
    else{
        return par(n-1);
    }
}
```



Recursión vs. Iteración



- La recursión lineal por la cola se puede convertir de forma inmediata en iteración (bucle).
- Para otras formas de recursión se requieren técnicas de transformación de programas y posiblemente estructuras de datos más complejas.

"The transformation from recursion to iteration is one of the most fundamental concepts of computer science." D. Knuth, 1974

Recursión vs. Iteración

- **Cálculo del factorial**

fac(n) = n! = n*(n-1)*(n-2)…2*1 (con n > 0 y m = 1)

```
public static long fac(int n, int m){
    if (n<=1){
       return m;
    }
    else{
       return fac(n-1, n*m);
    }
}
```

Recursión

```
public static long fac(int n, int m){
    private int a = n;
    private int b = m;
    while (! a <= 1){
       b = a * b;
       a = a - 1;
    }
    return b;
}
```

Iteración

Ejercicio 4

- Implementa el método recursivo `mcd(int a, int b)` el cual permite calcular el máximo común divisor de dos números enteros.

$$\gcd(x,y) = \begin{cases} x & \text{if } y = 0 \\ \gcd(y, \text{remainder}(x,y)) & \text{if } y > 0 \end{cases}$$

Ejercicio 5

- Implementa el método recursivo `decToBin(int decimal, String binary)` el cual recibe un número decimal y devuelve una cadena con el equivalente en binario de dicho número decimal.