Recursión

ALGORITMOS Y ESTRUCTURAS DE DATOS I

¿Qué es la recursión según el diccionario?

- Volver una cosa al sitio dónde salió, retornar, repetirse, reaparecer.

¿En programación?

- Es una herramienta muy potente para la resolución de problemas.
- Muchos algoritmos se pueden expresar más fácilmente utilizando una formulación recursiva.
- Un método recursivo es un método que, directa o indirectamente, se hace una llamada a sí mismo.

Características de una función recursiva

- Una función recursiva si invoca a si mismo con instancias diferentes.
- Los sucesivos puntos de retorno que se van generando en cada invocación recursiva se almacenan en una pila (o stack)
- La función debe tener al menos un caso base (escenario en el cual dejará de invocarse a si mismo)

Pila de llamadas

- Es una **estructura lineal dinámica** donde se agregan y quitan elementos sólo en uno de sus extremos.
- El tope de la pila es el punto donde debe retornarse el control tras la terminación del subprograma corriente.
- Así mismo, si el subprograma corriente llama a otro, el correspondiente punto de retorno debe colocarse como nuevo tope de la pila.
- Al finalizar un subprograma, se usa el tope de la pila como dirección de retorno y se lo remueve de la pila.

Ejemplo: Potencia

Se escribe a^n y se lee: «a elevado a n».

Definiciones:

- Cuando el exponente es un número natural, equivale a multiplicar un número por sí mismo varias veces: el exponente determina la cantidad de veces.

$$2^4 = 2 \cdot 2 \cdot 2 \cdot 2 = 16$$

- Cuando el exponente es un número entero negativo, equivale a la fracción inversa de la base pero con exponente positivo
- Cualquier número elevado a 0, distinto de 0, es igual a 1
- Toda potencia de exponente 1 es igual a la base

Ejemplo: Potencia

Resumen:

n > 0, a<>0 $\longrightarrow$ $a^n = \underbrace{a \times \cdots \times a}_{n},$

n = 0, a<>0 $\longrightarrow$ $a^0 = 1\ (a \neq 0)$

n < 0, a<>0 $\longrightarrow$ $a^{-p} = \frac{1}{a^p}$

n = 1, a<>0 $\longrightarrow$ $a^1 = a$

a = 0 $\longrightarrow$ $a^n = 0$

(indefinido para n=0, suponemos 0)

Pila de llamadas:

```
public double potencia(int a, int n) {
        if (a == 0){
                return 0;
        }else{
                if  (n == 0){
                        return 1;
                }else{
                        if (n<0){
                            return 1/potencia (a, -1*n);
                         }else{
                            return a * potencia (a, n-1);
                         }
                  }
        }
  }
```

Definición de casos base

Pila de llamadas: funcionamiento

VARIABLES a = 2, n = 3

```
   double potencia(int a, int n) {
1    if (a == 0){
2      return 0;
3    }else{
4      if  (n == 0){
5        return 1;
6      }else{
7        if (n<0){
8          return 1/potencia (a, -1*n);
9        }else{
10           return a * potencia (a, n-1);
   } } } }
```

P(2,2)
P(2,3) L10

stack

Pila de llamadas: funcionamiento

VARIABLES a = 2, n = 2

```
   double potencia(int a, int n) {
1    if (a == 0){
2      return 0;
3    }else{
4      if  (n == 0){
5        return 1;
6      }else{
7        if (n<0){
8          return 1/potencia (a, -1*n);
9        }else{
10         return a * potencia (a, n-1);
   } } } }
```

P(2,1)
P(2,2) L10
P(2,3) L10

stack

Pila de llamadas: funcionamiento

VARIABLES a = 2, n = 1

```
   double potencia(int a, int n) {
1    if (a == 0){
2      return 0;
3    }else{
4      if  (n == 0){
5        return 1;
6      }else{
7        if (n<0){
8          return 1/potencia (a, -1*n);
9        }else{
10         return a * potencia (a, n-1);
   } } } }
```

P(2,0)
P(2,1) L10
P(2,2) L10
P(2,3) L10

stack

Pila de llamadas: funcionamiento

VARIABLES a = 2, n = 0

```
   double potencia(int a, int n) {
1    if (a == 0){
2      return 0;
3    }else{
4      if   (n == 0){
5        return 1;
6      }else{
7        if (n<0){
8          return 1/potencia (a, -1*n);
9        }else{
10         return a * potencia (a, n-1);
   } } } }
```

P(2,1) L10
P(2,2) L10
P(2,3) L10

stack

Pila de llamadas: funcionamiento

VARIABLES a = 2, n = 0

```
double potencia(int a, int n) {
1   if (a == 0){
2     return 0;
3   }else{
4     if  (n == 0){
5       return 1;
6     }else{
7       if (n<0){
8         return 1/potencia (a, -1*n);
9       }else{
10        return a * potencia (a, n-1);
} } } }
```

2

1

2

P(2,2) L10

P(2,3) L10

stack

Pila de llamadas: funcionamiento

VARIABLES a = 2, n = 0

```
double potencia(int a, int n) {
1   if (a == 0){
2     return 0;
3   }else{
4     if  (n == 0){
5       return 1;
6     }else{
7       if (n<0){
8         return 1/potencia (a, -1*n);
9       }else{
10        return a * potencia (a, n-1);
} } } }
```

2

2

4

P(2,3) L10

stack

Pila de llamadas: funcionamiento

VARIABLES a = 2, n = 0

```
double potencia(int a, int n) {
1   if (a == 0){
2     return 0;
3   }else{
4     if  (n == 0){
5       return 1;
6     }else{
7       if (n<0){
8         return 1/potencia (a, -1*n);
9       }else{
10        return a * potencia (a, n-1);
} } } }
```

2

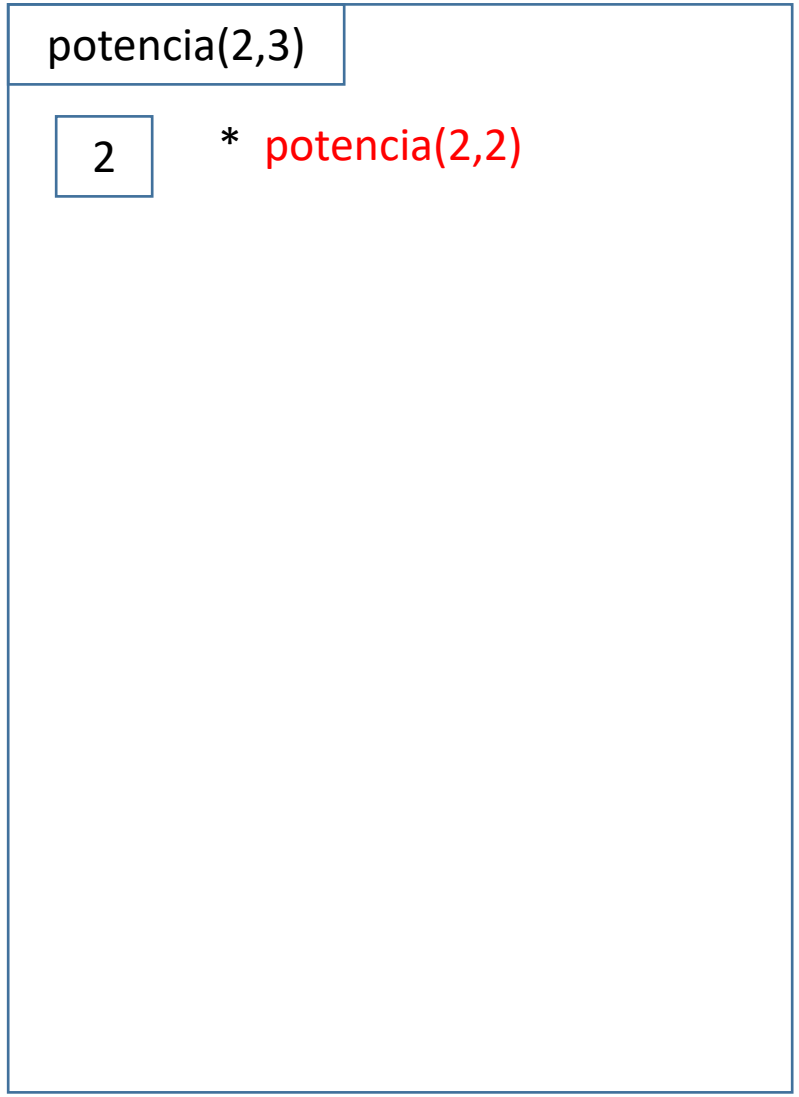
4

8

stack

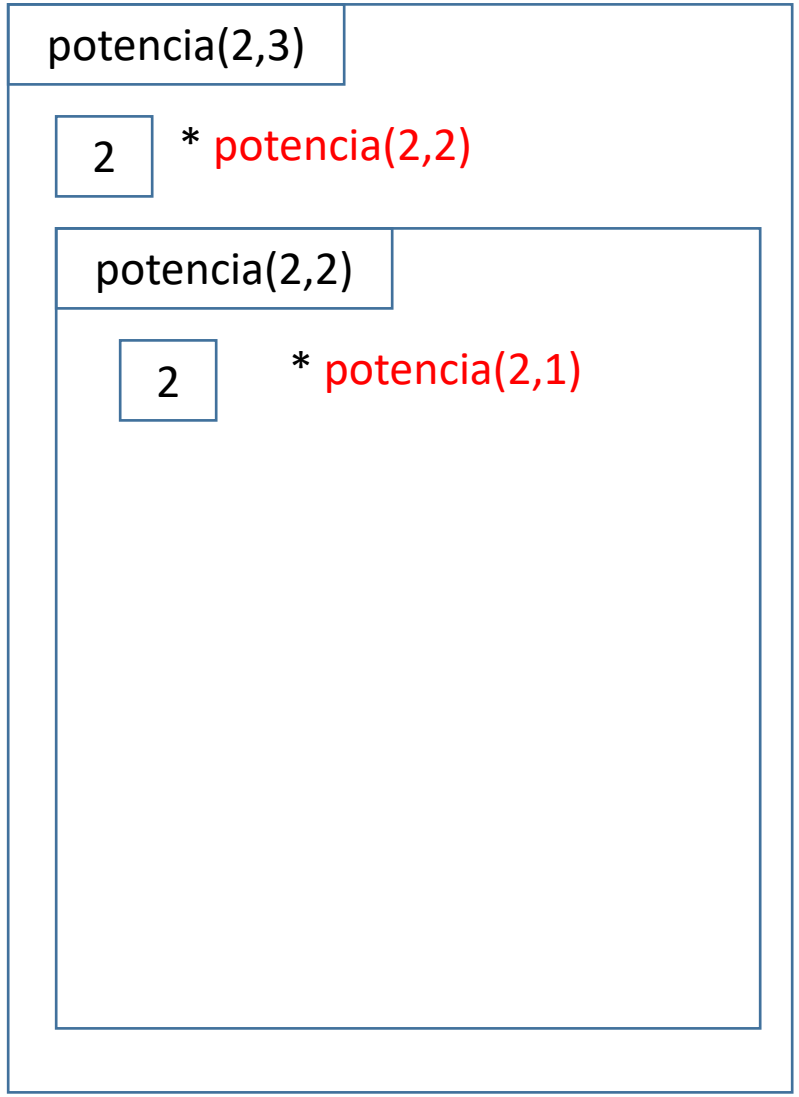
Pila de llamadas: método de verificación

```
double potencia(int a, int n) {
  if (a == 0){
    return 0;
  }else{
    if  (n == 0){
      return 1;
    }else{
      if (n<0){
         return 1/potencia (a, -1*n);
      }else{
         return a * potencia (a, n-1);
} } } }
```



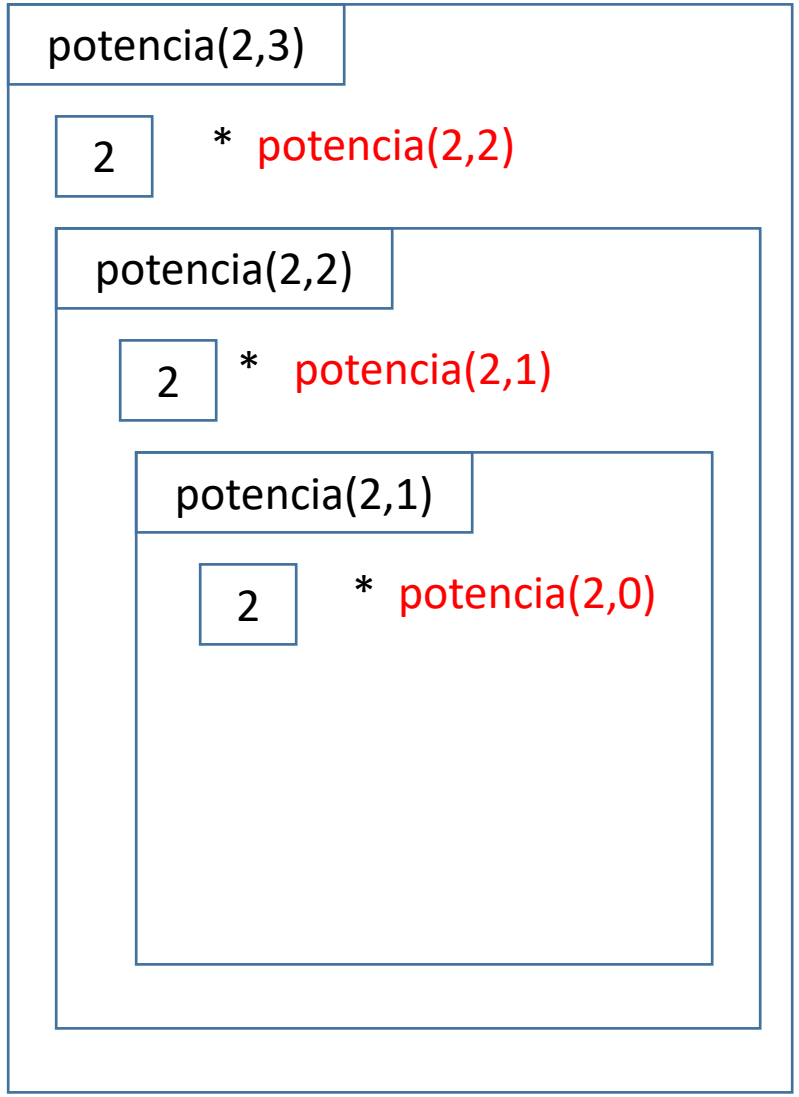
Pila de llamadas: método de verificación

```
double potencia(int a, int n) {
  if (a == 0){
    return 0;
  }else{
    if  (n == 0){
      return 1;
    }else{
      if (n<0){
         return 1/potencia (a, -1*n);
      }else{
         return a * potencia (a, n-1);
} } } }
```



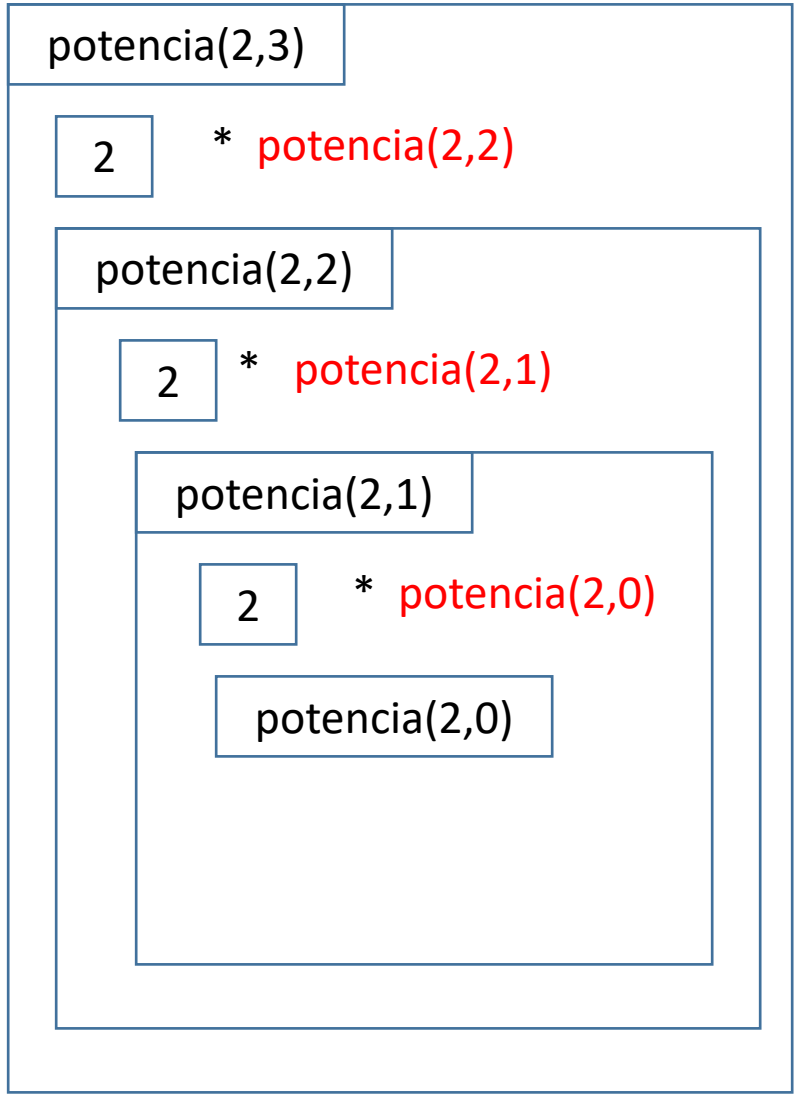
Pila de llamadas: método de verificación

```
double potencia(int a, int n) {
  if (a == 0){
    return 0;
  }else{
    if  (n == 0){
      return 1;
    }else{
      if (n<0){
         return 1/potencia (a, -1*n);
      }else{
         return a * potencia (a, n-1);
} } } }
```



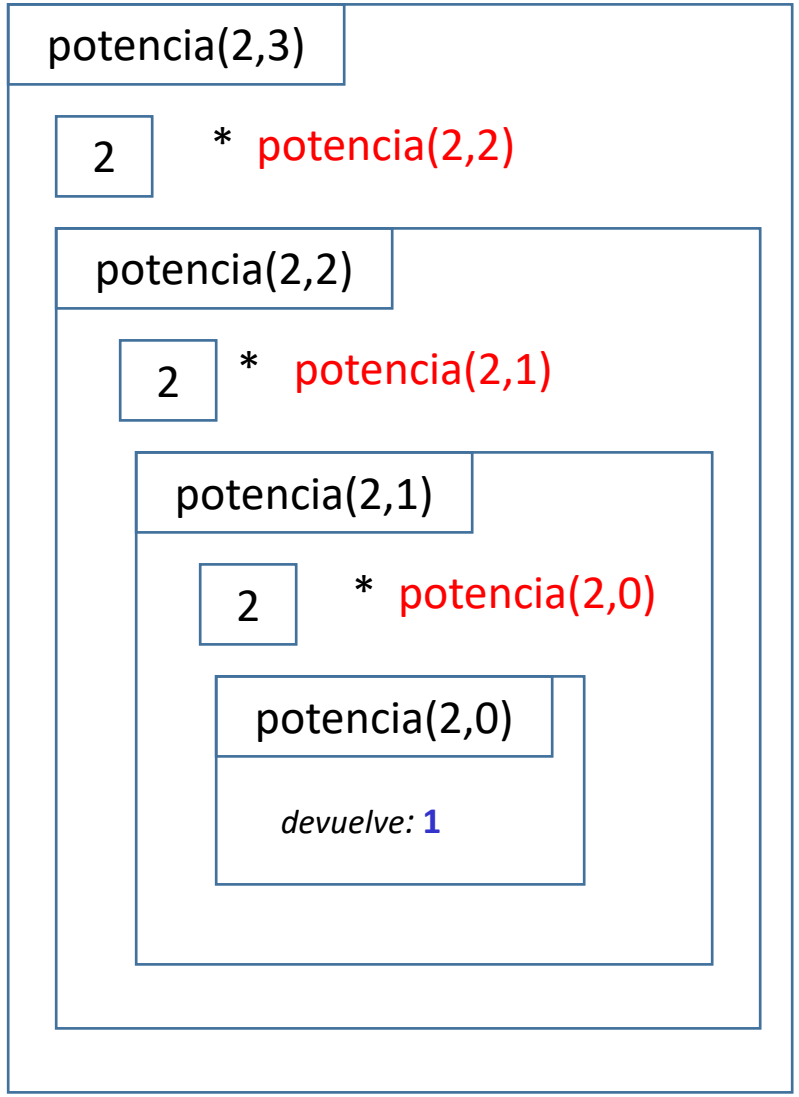
Pila de llamadas: método de verificación

```
double potencia(int a, int n) {
  if (a == 0){
    return 0;
  }else{
    if  (n == 0){
      return 1;
    }else{
      if (n<0){
         return 1/potencia (a, -1*n);
      }else{
         return a * potencia (a, n-1);
} } } }
```



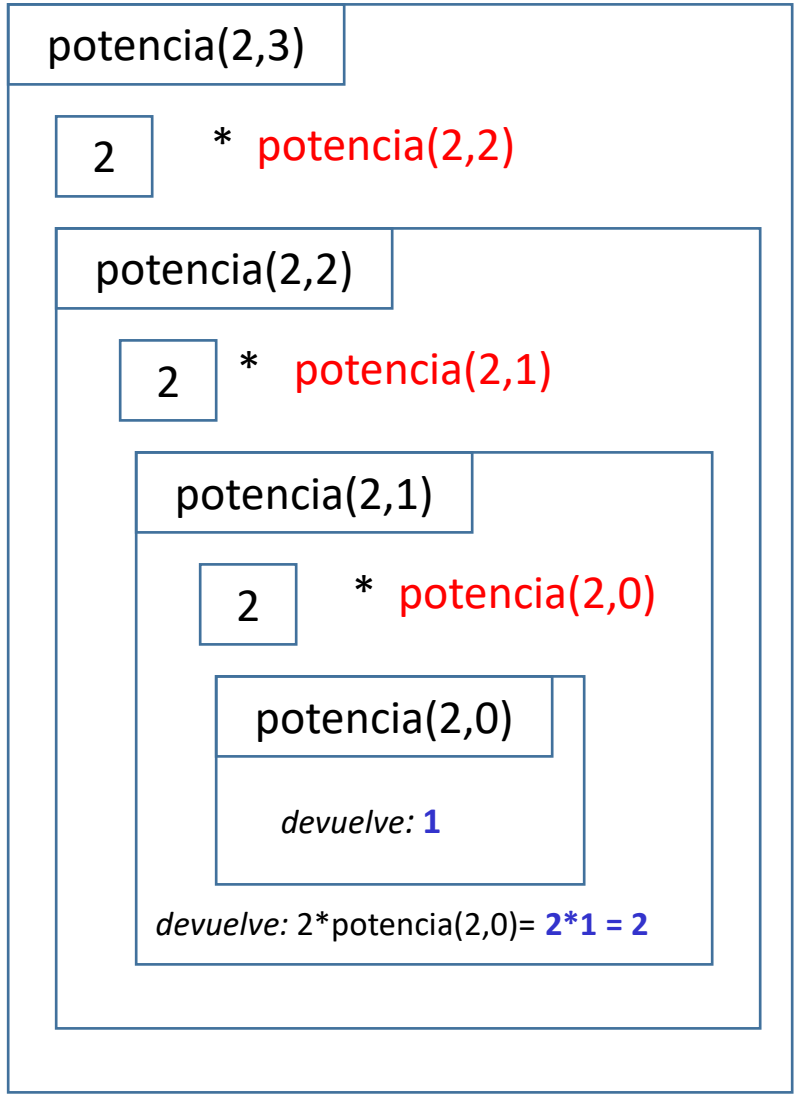
Pila de llamadas: método de verificación

```
double potencia(int a, int n) {
  if (a == 0){
    return 0;
  }else{
    if  (n == 0){
      return 1;
    }else{
      if (n<0){
         return 1/potencia (a, -1*n);
      }else{
         return a * potencia (a, n-1);
} } } }
```



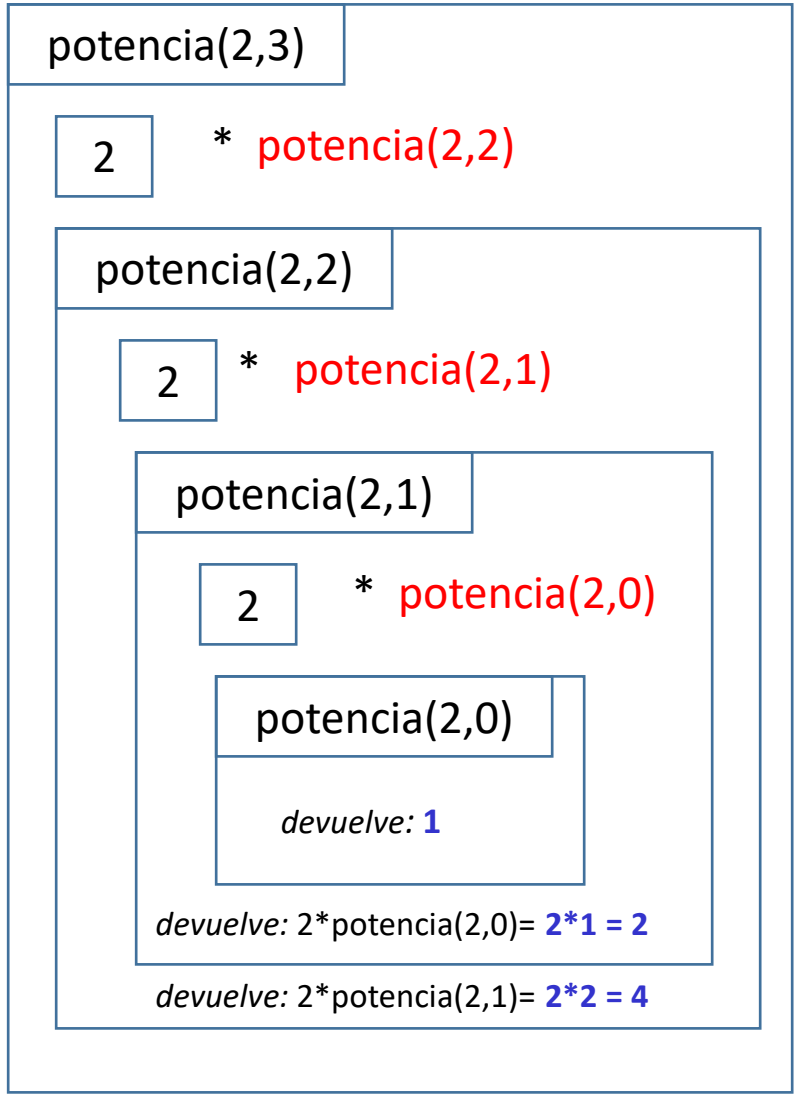
Pila de llamadas: método de verificación

```
double potencia(int a, int n) {
  if (a == 0){
    return 0;
  }else{
    if  (n == 0){
      return 1;
    }else{
      if (n<0){
         return 1/potencia (a, -1*n);
      }else{
         return a * potencia (a, n-1);
} } } }
```



Pila de llamadas: método de verificación

```
double potencia(int a, int n) {
  if (a == 0){
    return 0;
  }else{
    if  (n == 0){
      return 1;
    }else{
      if (n<0){
         return 1/potencia (a, -1*n);
      }else{
         return a * potencia (a, n-1);
} } } }
```



Pila de llamadas: método de verificación

```
double potencia(int a, int n) {
  if (a == 0){
    return 0;
  }else{
    if  (n == 0){
      return 1;
    }else{
      if (n<0){
         return 1/potencia (a, -1*n);
      }else{
         return a * potencia (a, n-1);
} } } }
```