IES DR LLUÍS SIMARRO

ENSENYAMENT PÚBLIC

XÀTIVA

CFGS Desarrollo de aplicaciones web

Módulo profesional: Programación

GENERALITAT VALENCIANA

Conselleria d'Educació, Investigació, Cultura i Esport

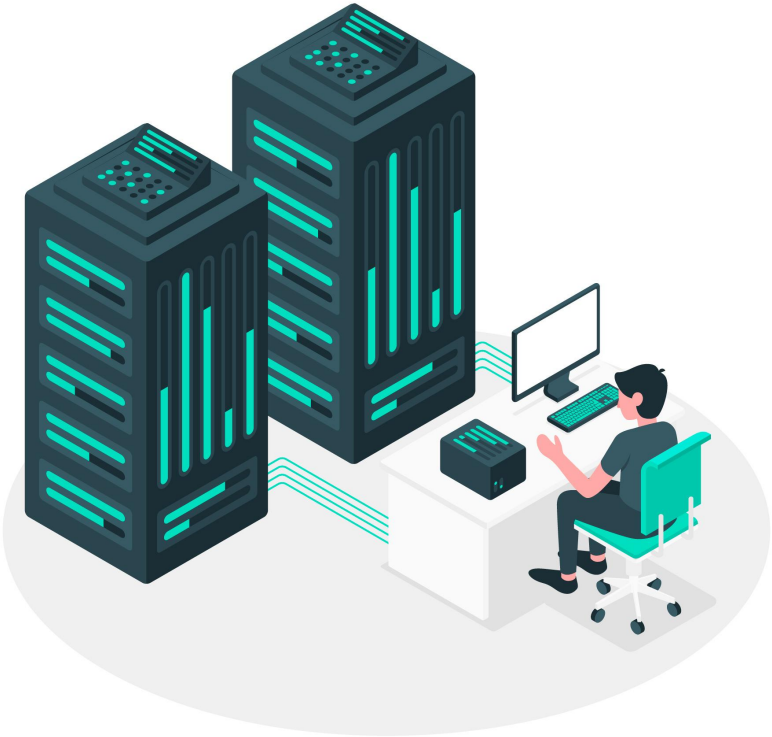
GOBIERNO DE ESPAÑA

MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Unió Europea

Fons Social Europeu

L'FSE inverteix en el teu futur



Material elaborado por:

Anna Sanchis

Carlos Cacho

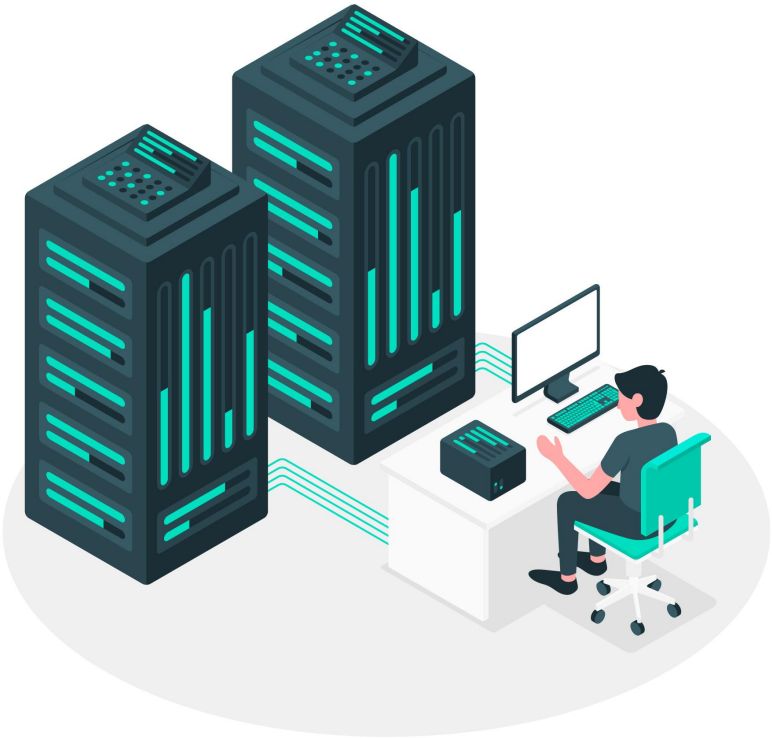
Raquel Torres

Lionel Tarazón

Fco. Javier Valero

Revisado y editado por:

Edu Torregrosa Llácer



Datos profesor

Edu Torregrosa Llácer

eduardotorregrosa@iesIluissimarro.org

Web del módulo: https://moodle.aulaenlanube.com/

Métodos en JAVA

1. Introducción.
2. Declaración de métodos
3. Uso de métodos
4. Recursión

Introducción

- En muchas ocasiones, se tiene que usar un mismo segmento de código para resolver un mismo (sub)problema con datos **diferentes**
- Escribir el mismo código varias veces es un trabajo repetitivo, improductivo, fuertemente relacionado con determinadas variables y que, por todo lo anterior, presenta una tendencia al error nada despreciable
- La mayor parte de los lenguajes de programación ofrecen mecanismos que permiten reutilizar un mismo segmento de código sobre datos diferentes (**subprogramación**)

Introducción

- Un **subprograma** es un segmento de código que .....
  - → Se usa (se **invoca** su ejecución) desde algún punto de un (sub)programa para resolver un (sub)problema dado
  - → Emplea los datos (**parámetros**) que se le proporcionan al invocarlo para resolver dicho (sub)problema
  - → Devuelve (**return**) los resultados (**valores de retorno**) del (sub)problema para los datos proporcionados al invocarlo

Introducción

La mejor forma de crear y mantener un programa grande es construirlo a partir de piezas más pequeñas. Cada una de estas piezas es más manejable que el programa en su totalidad.

> Las **funciones** (subprogramas) son utilizadas para evitar la repetición de código en un programa. Se pueden ejecutar desde varios puntos de un programa con solo invocarlos.

El concepto de función es una forma de encapsular un conjunto de instrucciones dentro de una declaración específica (llamada generalmente SUBPROGRAMA o FUNCIÓN), permitiendo la descomposición funcional y la diferenciación de tareas.

Declaración de métodos

El método es el mecanismo de subprogramación que ofrece el lenguaje Java

Un **método** se declara definiendo su **cabecera** y su **cuerpo**, dentro de una clase Java.

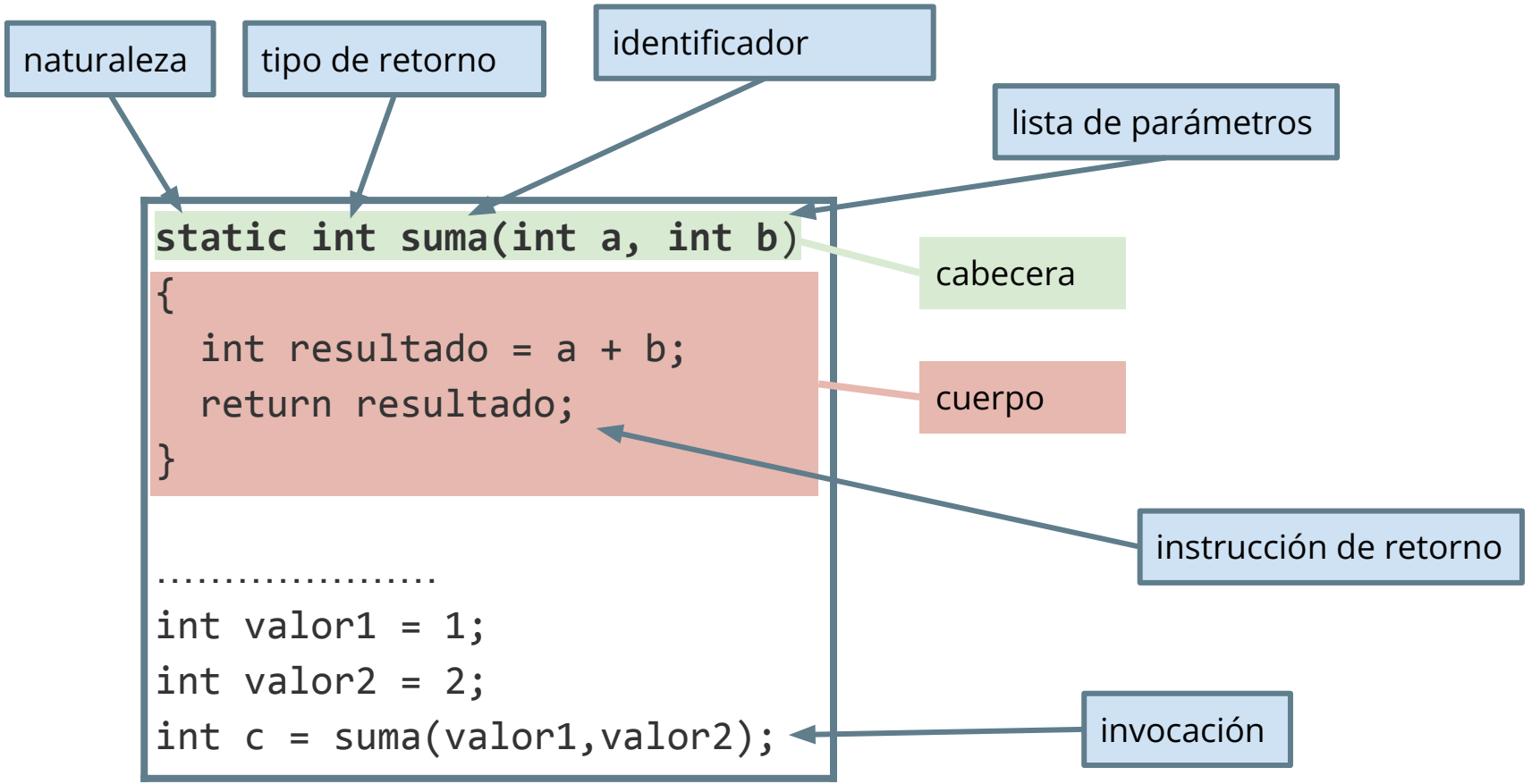
La **cabecera** de un método se define, en este orden:

- su **naturaleza** (modificadores como public, static...)
- su **tipo de retorno**, o tipo de sus resultados (valores de retorno)
- su **nombre** (identificador)
- su **lista de parámetros**, o los tipos y nombres de sus datos (parámetros)

El **cuerpo** es la secuencia de instrucciones que son ejecutadas cuando es invocado, y puede incluir:

- Declaraciones de variables.
- Cualquier tipo de instrucción o bloque de instrucciones: asignación, condicional, bucle, e incluso, **llamadas o invocaciones a otros métodos.**

Declaración de métodos



Declaración de métodos

- Identificación de los componentes de la **cabecera** de un método:

naturaleza	tipo de retorno	identificador	lista de parámetros
static	boolean	posibleTriangulo	(double a, double b, double c)

- Por ahora, normalmente se usará el modificador static para describir la naturaleza de un método; en temas posteriores se darán más detalles sobre ellos.
- El nombre de un método y el de sus parámetros siguen las reglas habituales de los identificadores
- En JAVA se debe indicar el tipo de dato de retorno del método antes del identifcador

```
static int suma(int a, int b) //declaramos un método con identificador suma y que retorna un valor entero(int)
```

Declaración de métodos - RETURN

- Como resultado de su invocación, un método devuelve unos resultados, los denominados valores de retorno.
- El tipo de retorno de un método puede ser cualquier tipo de datos Java, primitivo o de referencia (int, double, char, String, boolean, etc.)
- Para devolver un valor de retorno de un método se usa la instrucción **return** . En concreto, el valor de retorno aparece tras la palabra reservada return.
    - **La ejecución de un método finaliza tras la ejecución de cualquiera de los return que aparecen en su cuerpo**
    - **Si un método no devuelve explícitamente ningún valor vía return, entonces el tipo de su resultado es void. En este caso, implícitamente, se ejecuta “return” como última instrucción del método**

Declaración de métodos - RETURN

Ejemplos de métodos con y sin valores de retorno:

```
static double areaCuadrado(double lado)
{
  return lado*lado;
}
static double perimetroCuadrado(double lado)
{
  return lado*4;
}

static void mostrarMenu()
{
  System.out.println("Esto es un método que pinta un menú");
  System.out.println("1 - Obtener área");
  System.out.println("2 - Obtener perímetro");
}
```

Declaración de métodos - Parámetros

- Es al invocar a un método cuando se le proporcionan los valores de los datos (parámetros actuales) para los que debe ejecutar su código
- Los parámetros definidos en la cabecera de un método, al declararlo, se denominan parámetros formales pues no tienen asociado un valor real
    - **Sintaxis: tipo1 nomParF1, tipo2 nomParF2 … tipok nomParFk**
- Los parámetros, a través de sus identificadores pueden usarse como variables en el cuerpo del método, **NO deben volver a declararse**.
- Se pueden declarar métodos sin parámetros, en este caso se utilizarán paréntesis vacíos después del identificador.

Ejemplo: `static int obtenerNum()`

Uso de métodos - Invocaciones

- Un método no se ejecuta por sí mismo, sino que ha de ser invocado para ejecutarse.
- Para invocar o llamar a un método se necesita escribir:
    - Su identificador
    - Los datos con los que trabajará, es decir, la lista de sus parámetros actuales.

Por ejemplo, declaradas e inicializadas s1,s2,s3. Las siguientes son invocaciones correctas:

```
posibleTriangulo(s1, s2, s3) //invocamos utilizando las variables s1, s2 y s3
areaCuadrado(s1) //invocamos utilizando la variable s1
tablonDeAnuncios(2*1) //invocamos utilizando un valor numérico como parámetro
saludo(“Nombre”) //invocamos utilizando la palabra nombre como parámetro
saludo(nombre) //invocamos utilizando la variable nombre como parámetro
```

Uso de métodos - Invocaciones

- Un parámetro actual es cualquier expresión que se evalúe a un tipo de datos compatible con el del correspondiente parámetro formal de la cabecera del método
- Al evaluar un parámetro actual, el correspondiente parámetro formal se inicializa al valor resultante, o toma dicho valor.
- La llamada a un método se evalúa al resultado que éste devuelve.
- **Se puede realizar la invocación a un método en cualquier expresión que tenga un tipo compatible con el retorno del método.**

El método main

- El **main** es el método que aparecerá en la mayoría de las clases de Java que se presenten en este módulo
- Cuando se pide la ejecución de una clase, **main es el método que la JVM** invoca por defecto
- Cabecera: **public static void main (String[] args)**
- Su tipo de retorno es **void**, i.e. no retorna ningún valor de retorno.
- Su único parámetro es una **lista de objetos String**. Esta lista recibe como parámetros la lista de argumentos que acompañan al comando java en la línea de comandos:

```
public class Eco {
     public static void main(String[] args){
       for ( int i=0; i<args.length; i++ )
      System.out.println(args[i]);
       }
  }
    Al ejecutar el comando "java Eco ¡Hola mundo!"
  se escribirá en pantalla  ¡Hola mundo! porque
  args[0]="¡Hola" y args[1]="mundo!"
```

Variables locales y globales

- **En el cuerpo de un método** se pueden declarar variables, que se denominan **variables locales**.
- Las variables locales **sólo se pueden usar en el método en el que se declaran**, por lo que su ámbito es dicho método.
- Los parámetros de un método se pueden considerar variables locales del mismo.
- En cambio, en una clase se pueden declarar variables globales en cualquier punto fuera del cuerpo de un método.
- Cualquier método de una clase puede usar cualquiera de las variables globales de ésta.

Variables locales y globales

- Si bien las variables locales de un método y las variables globales de una clase no pueden tener el mismo nombre, una variable local (o un parámetro) puede sobrescribir a una variable global usando su nombre.

```
public class Ambitos {
  static int x; // variable Global
  static int cuadrado(int z){
        int x; // variable Local del método cuadrado
        x = z * z; // x es la variable Local, no la Global
        return x;
   }
   public static void main(String args[]){
       int p; // variable Local del método cuadrado
       p = 10; //  Correcto: p es Local en este ámbito
       q = 100; // Incorrecto: q no es de este ámbito
       x = cuadrado(p); // Correcto: x es Global
    }
}
```

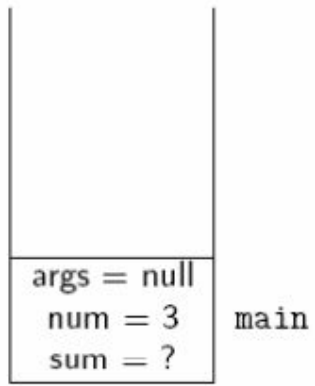
La pila de llamadas

- Para implementar la ejecución de la llamada a un método **la JVM utiliza la denominada pila de llamadas**
- La pila de llamadas se aloja en la memoria RAM y está compuesta por los denominados registros de activación. Cada uno de estos registros:
    - tiene asociada una llamada dada a un método, la del main incluida.
    - contiene las variables locales del método y sus parámetros, junto con el valor que se les ha asignado a éstos últimos al realizar la llamada (parámetros actuales)
- Un método solo puede usar:
    - la información contenida en su correspondiente registro de activación de la pila de llamadas
    - los datos contenidos en las variables globales

La pila de llamadas

- Ejemplo de funcionamiento de la pila de llamadas:

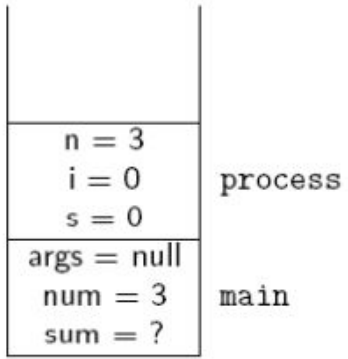
```
...
static int process(int n){
        int i, s = 0;
        for ( i=0; i<n; i++ )
                s=s+i;
        return s;
}
public static void main(String args[]){
      int num, sum, j;
      num = 3;
      sum = process(num);          // <--
}
```



La pila de llamadas

- Ejemplo de funcionamiento de la pila de llamadas:

```
...
static int process(int n){
        int i, s = 0;
        for ( i=0; i<n; i++ )        // <---
                s=s+i;
        return s;
}
public static void main(String args[]){
      int num, sum, j;
      num = 3;
      sum = process(num);
}
```

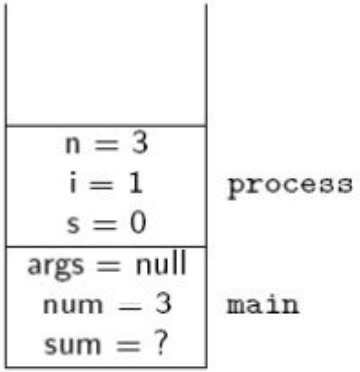


La pila de llamadas

- Ejemplo de funcionamiento de la pila de llamadas:

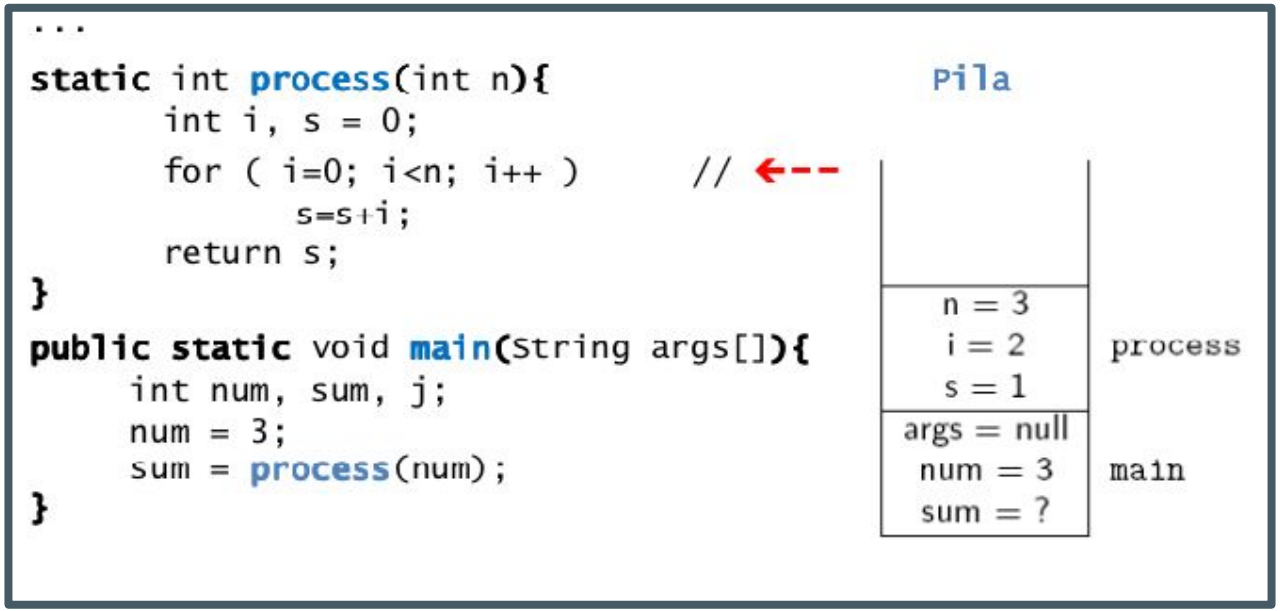
```
...
static int process(int n){
        int i, s = 0;
        for ( i=0; i<n; i++ )        // <---
                s=s+i;
        return s;
}
public static void main(String args[]){
      int num, sum, j;
      num = 3;
      sum = process(num);
}
```

Pila



La pila de llamadas

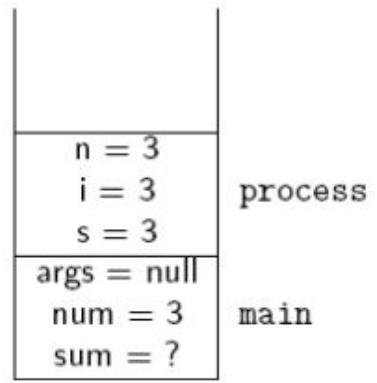
- Ejemplo de funcionamiento de la pila de llamadas:



La pila de llamadas

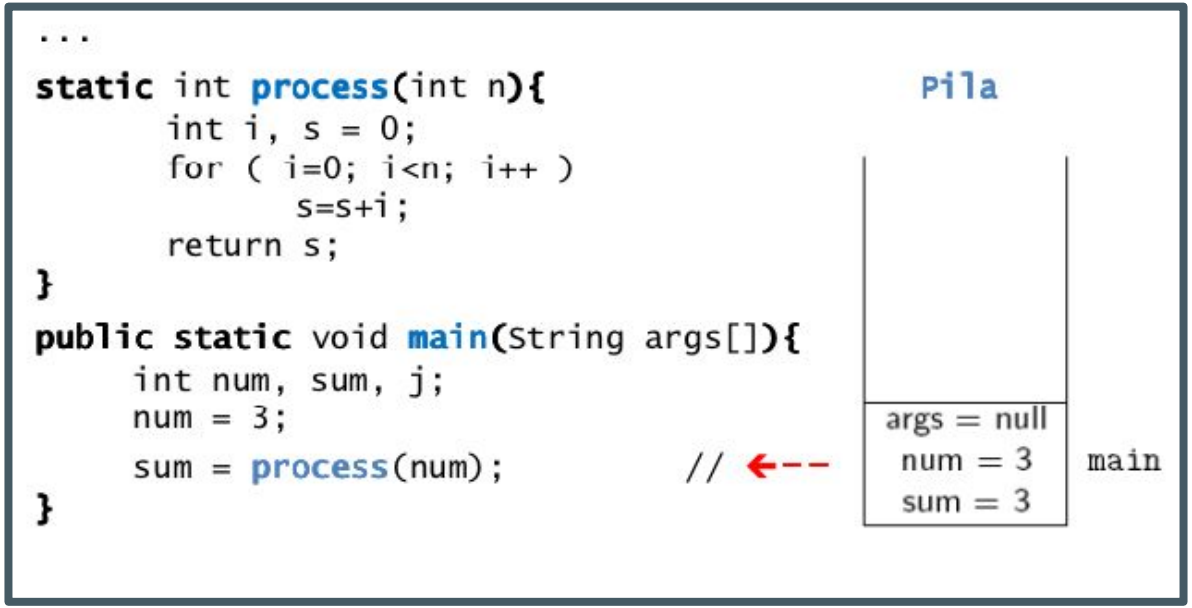
- Ejemplo de funcionamiento de la pila de llamadas:

```
...
static int process(int n){
       int i, s = 0;
       for ( i=0; i<n; i++ )      // <---
              s=s+i;
       return s;
}
public static void main(String args[]){
      int num, sum, j;
      num = 3;
      sum = process(num);
}
```



La pila de llamadas

- Ejemplo de funcionamiento de la pila de llamadas:



Ejemplos

1. Implementa un programa que dado un número calcule el cubo, la declaración del método será: public static double cubo (double x)
2. Implementa un programa que dados dos números los multiplique, la declaración del método será: public static int producto (int a, int b)
3. Utiliza el método anterior para que, dado un número calcule la tabla de multiplicación.
4. Implementa un programa que dado un número diga si éste es positivo, la declaración del método será: public static boolean esPositivo(int x)

Recursión

- Se denomina recursivo a cualquier ente (definición, proceso, estructura, etc.) que se define en función de sí mismo.
- Un algoritmo es recursivo si obtiene la solución de un problema en base a los resultados que él mismo proporciona para casos más sencillos del mismo problema. Es decir, es un método que se invoca a sí mismo:
- El caso base de un problema es el que se puede resolver trivialmente.
- El caso general de un problema es el que resuelve recursivamente, empleando el mismo algoritmo para casos más sencillos
- Las funciones matemáticas recursivas se implementan fácilmente usando algoritmos recursivos, como por ejemplo la función factorial:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n > 0 \end{cases}$$

Recursión

- Los algoritmos recursivos se implementan fácilmente usando métodos recursivos, como por ejemplo el método fact:

```
static int fact(int n){
    if ( n == 0 )        return 1;
        caso base        instrucciones para resolver el caso base
    /*else n>0 */        return n * fact(n-1);
}       !caso base,      instrucciones para resolver el caso general
        i.e. caso general
```

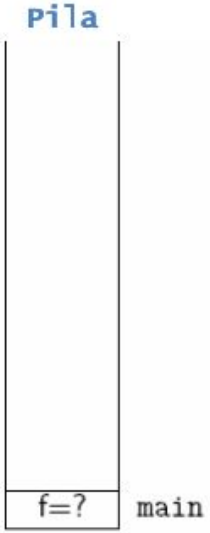
Observa: el cuerpo de un método recursivo sólo puede ser una instrucción condicional

- La JVM emplea intensivamente la pila de llamadas, también denominada la pila de recursión, para ejecutar un método recursivo.

Recursión

- Ejemplo: ejecución de un método recursivo usando la pila de llamadas:

```
...
static int fact(int n){
  int r;
       if ( n == 0 ) r = 1;
  else          r = n * fact(n-1);
  return r;
}
public static void main(String args[]){
  int f;
  f = fact(4);                    // <---
}
```



Recursión

- Ejemplo: ejecución de un método recursivo usando la pila de llamadas:

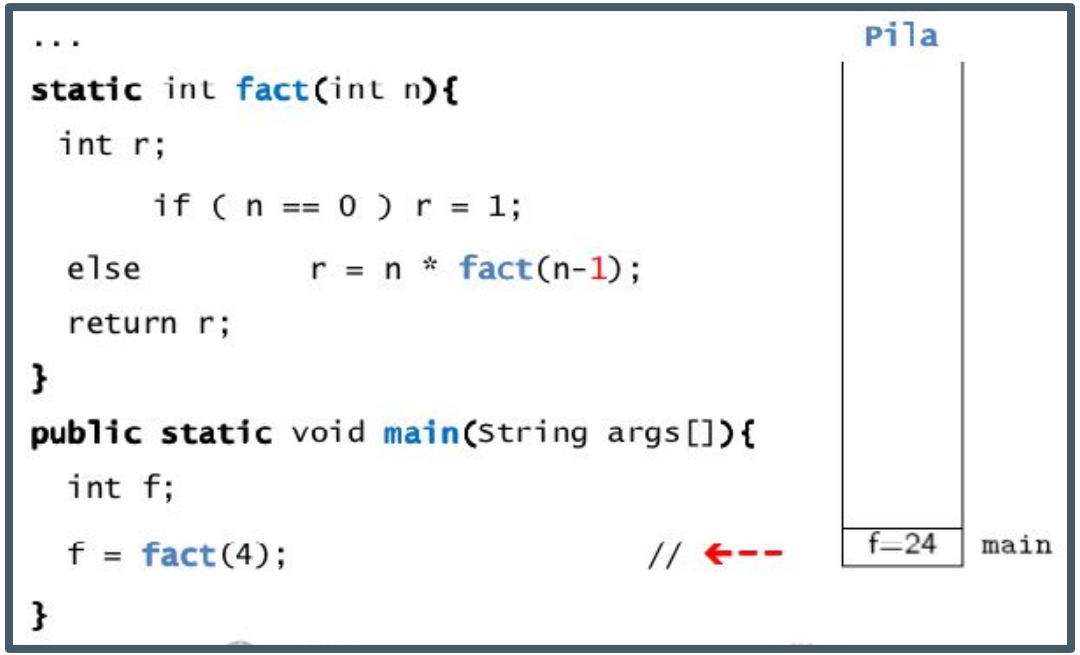
```
...
static int fact(int n){
  int r;
       if ( n == 0 ) r = 1;
  else          r = n * fact(n-1);
  return r;                          // <---
}
public static void main(String args[]){
  int f;
  f = fact(4);
}
```

Pila

n = 0 r = 1	fact
n = 1 r = ?	fact
n = 2 r = ?	fact
n = 3 r = ?	fact
n = 4 r = ?	fact
f=?	main

Recursión

- Ejemplo: ejecución de un método recursivo usando la pila de llamadas:



Recursión

- El caso base de un método recursivo permite detener la recursión (no se hace ninguna llamada en él), por lo que resulta imprescindible.
- De hecho, en las llamadas recursivas del caso general alguno de los parámetros debe disminuir su valor para que finalmente - tras un nº finito de llamadas- se alcance el caso base
- En general, los métodos recursivos son menos eficientes que los que solucionan el mismo problema con un bucle (iterativos): la recursión hace un uso intensivo de la pila de llamadas.
- Casi siempre, los métodos recursivos permiten expresar la resolución de ciertos problemas complejos de forma mucho más sencilla que sus equivalentes iterativos. De hecho, formular un algoritmo recursivo es el primer paso que hay para resolver muchos problemas interesantes con técnicas algorítmicas avanzadas.

Preguntas