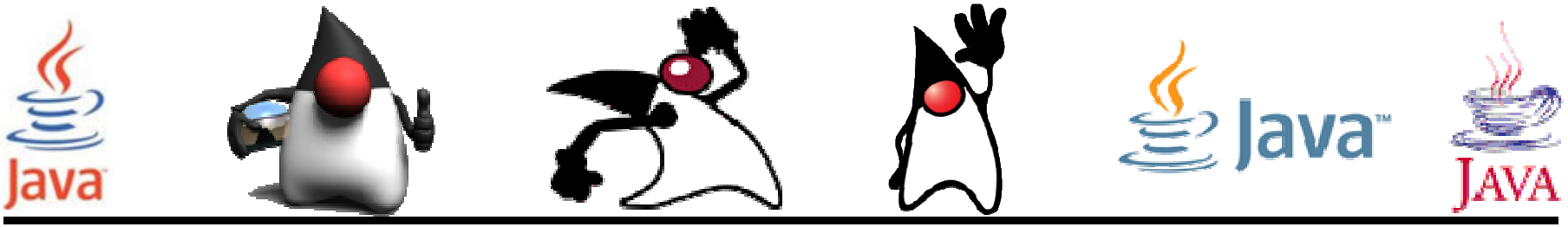


Métodos

Pedro Corcuera

Dpto. Matemática Aplicada y
Ciencias de la Computación

Universidad de Cantabria

corcuerp@unican.es

Objetivos

- Conocer la implementación de métodos.
- Familiarizarse con el concepto de paso de parámetros.
- Desarrollar estrategias para la descomposición de tareas complejas en otras más simples.
- Determinar el alcance de las variables.
- Diseñar e implementar algoritmos recursivos

Índice

- Métodos como cajas negras
- Implementación de métodos
- Paso de parámetros
- Valores de retorno
- Métodos sin valor de retorno
- Refinamiento sucesivo
- Alcance de variables
- Métodos recursivos

Métodos como cajas negras

- Un método es una secuencia de instrucciones con un nombre.
- El método se declara mediante un nombre para el bloque de código.

```
public static void main(String[] args)
{
  double z = Math.pow(2, 3);
  . . .
}
```

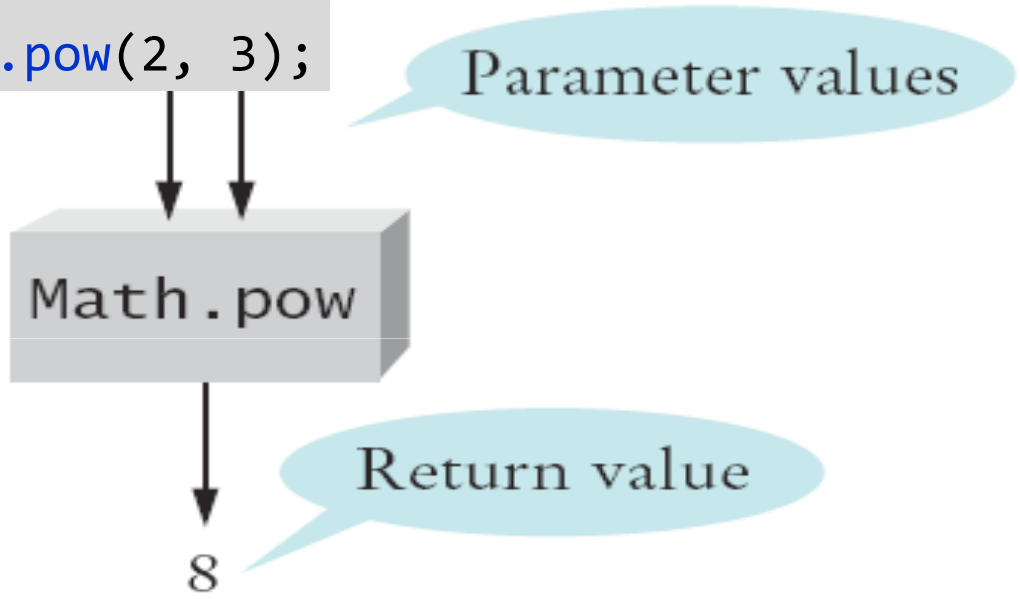
- Un método se invoca cuando se requiere para ejecutar sus instrucciones. Facilita la reutilización.

Qué es un método?

- Algunos métodos ya se han venido usando:
  - Math.pow()
  - String.length()
  - Character.isDigit()
  - Scanner.nextInt()
  - main()
- Todos tienen:
  - Un nombre. Sigue las mismas reglas que las variables.
  - Paréntesis ( ) para indicar los parámetros de entrada.

Parámetros y valores retornados

```
public static void main(String[] args)
{
   double z = Math.pow(2, 3);
   . . .
}
```



- main 'pasa' dos parámetros a `Math.pow`
- calcula y retorna un valor de 8 a main
- main almacena el valor devuelto en la variable 'z'

Implementación de métodos

- Se requiere un método para calcular el volumen de un cubo
  - Qué se necesita para que cumpla su trabajo?
  - Qué responderá?
- Para desarrollar este método:
  - Poner un nombre al método (`cubeVolume`)
  - Dar un tipo y un nombre para cada parámetro (`double sideLength`)
  - Especificar el tipo y el valor devuelto (`double`)
  - Agregar modificadores `public static`

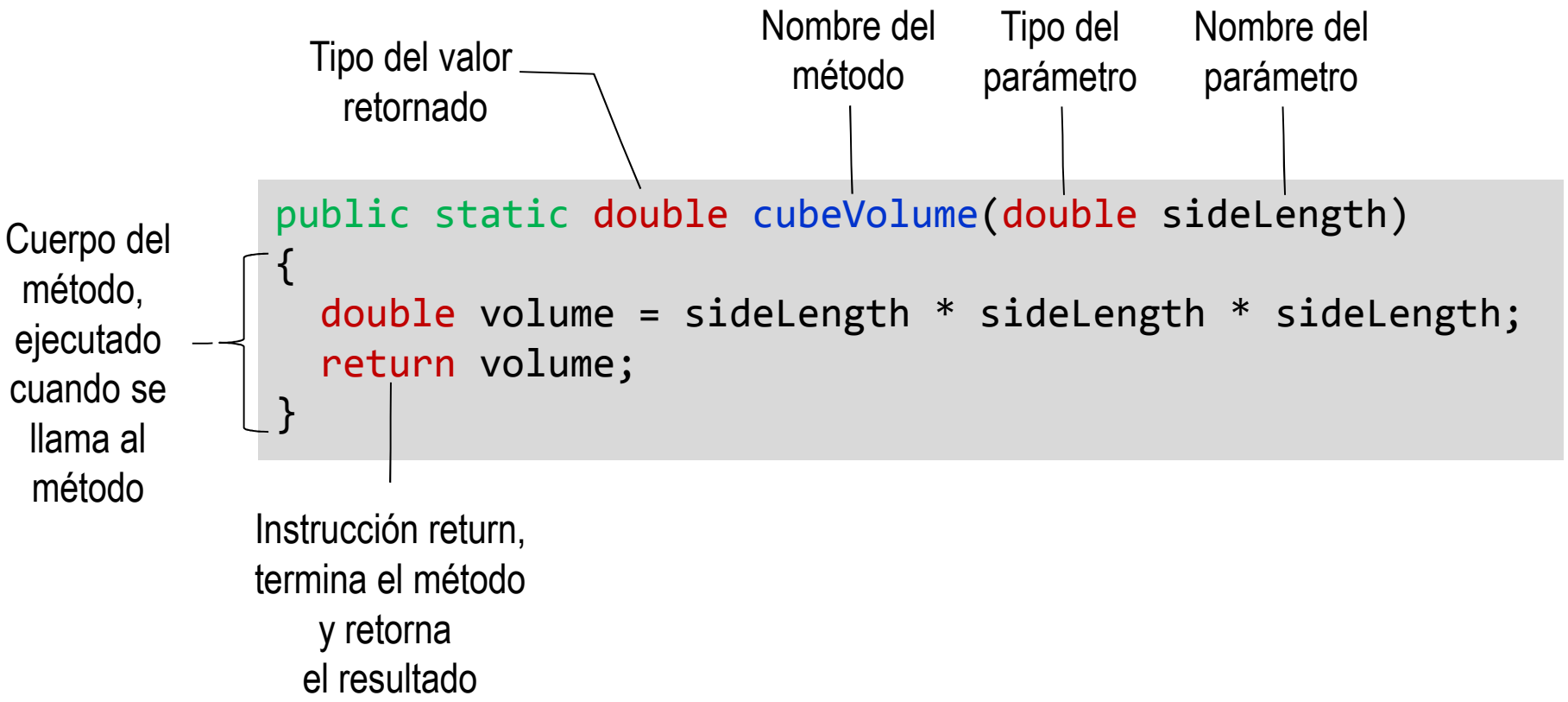
```
public static double cubeVolume(double sideLength)
```

Dentro del método

- Desarrollo del cuerpo del método
  - El cuerpo está encerrado entre llaves { }
  - El cuerpo contiene las declaraciones de variables e instrucciones que se ejecutan cuando se invoca el método
  - Retorna la respuesta calculada

```
public static double cubeVolume(double sideLength)
{
   double volume = sideLength * sideLength * sideLength;
   return volume;
}
```

Dentro del método



Invocación del método

- El valor retornado por `cubeVolume` es almacenado en variables locales en `main`
- Los resultados se imprimen

```
public static void main(String[] args)
{
  double result1 = cubeVolume(2);
  double result2 = cubeVolume(10);
  System.out.println("Un cubo de lado 2 tiene un volumen "
   + result1);
  System.out.println("Un cubo de lado 10 tiene un volumen "
   + result2);
}
```

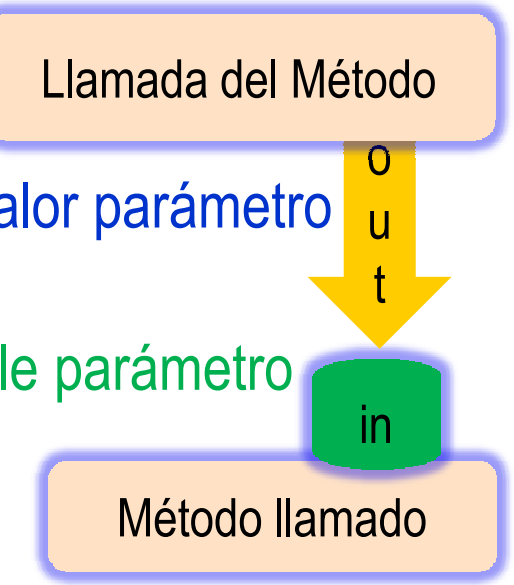
Programa completo

```
/** Programa que calcula el volumen de dos cubos. */
public class Cubos
{
   public static void main(String[] args)
   {
      double result1 = cubeVolume(2);
      double result2 = cubeVolume(10);
      System.out.println("Un cubo de lado 2 tiene un volumen " + result1);
      System.out.println("Un cubo de lado 10 tiene un volumen " + result2);
   }

   /** Calcula el volumen de un cubo.
      @param sideLength es la longitud del lado del cubo
      @return volumen del cubo
   */
   public static double cubeVolume(double sideLength)
   {
      double volume = sideLength * sideLength * sideLength;
      return volume;
   }
}
```

Paso de parámetros

- Las variables parámetro mantienen los valores de los parámetros suministrados en la llamada del método
  - Ambos deben ser del mismo tipo
- El valor del parámetro puede ser:
  - El contenido de una variable
  - Un valor literal
  - Llamado también 'parámetro actual'
- La variable parámetro es:
  - Nombrada en la declaración del método llamado
  - Usada como una variable en el método llamado
  - Llamado también 'parámetro formal'



Paso de parámetros

```
public static void main(String[] args)
{
   double result1 = cubeVolume(2);
   . . .
}
```

result1 = 8

```
public static double cubeVolume(double sideLength)
{
   double volume = sideLength * sideLength * sideLength;
   return volume;
}
```

sideLength = 2

volume = 8

Error común

- Intentar modificar los parámetros
  - Se pasa una copia de los valores de los parámetros
  - El método llamado puede modificar su copia local pero no el valor original en la invocación del método

```
public static void main(String[] args)
{
  double total = 10;
  addTax(total,   7.5);
}
```

copia

total 10.0

```
public static int addTax(double price, double rate)
{
  double tax = price * rate / 100;
  price = price + tax; // No tiene efecto fuera del metodo
  return tax;
}
```

price 10.75

Métodos con valores de retorno

- Los métodos pueden (opcionalmente) retornar un valor
  - El tipo del return se especifica en la declaración del método
  - La instrucción return termina el método inmediatamente y retorna el valor especificado al método invocante

Tipo return

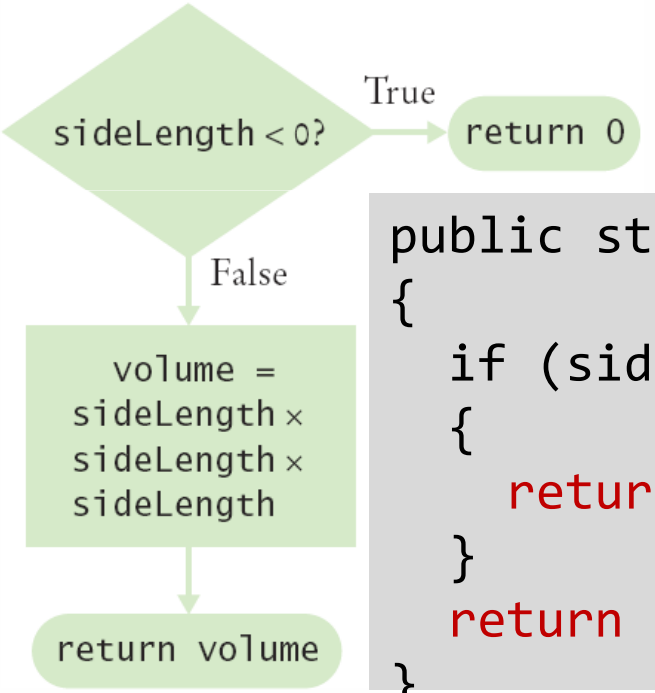
```
public static double addTax(double price, double rate)
{
  price += price * rate / 100;
  return price;
}
```

Valor return

- El valor return puede ser un valor, una variable o una expresión de cálculo
- El tipo debe coincidir con el tipo return

Métodos con returns múltiples

- Un método puede usar múltiple instrucciones return
  - **Cada** ramificación debe tener una instrucción return




```
public static double cubeVolume(double sideLength)
{
   if (sideLength < 0)
   {
      return 0;
   }
   return sideLength * sideLength * sideLength;
}
```

Pasos para implementar métodos

- Describir lo que el método debe hacer
- Determinar las entradas del método
- Determinar los tipo de las entradas y del valor retornado
- Escribir el pseudocódigo para el resultado deseado
- Implementar el cuerpo del método
- Prueba del método
  - Diseñar los casos de prueba y códificarlos

Métodos sin valores de retorno

- En caso de tener métodos que no retornan valores se usa `void` como tipo del método

```
...
boxString("Hello");
...
```

```
-------
!Hello!
-------
```

```
public static void boxString(String str)
{
  int n = str.length();
  for (int i = 0; i < n + 2; i++)
    { System.out.print("-"); }
  System.out.println();
  System.out.println("!" + str + "!");
  for (int i = 0; i < n + 2; i++)
    { System.out.print("-"); }
  System.out.println();
}
```

Metodología de diseño de métodos

- Para resolver una tarea compleja se divide la tarea en tareas más simples
- La subdivisión en tareas más sencillas continúa hasta que se pueda resolverlas fácilmente
- Este proceso se conoce como Refinamiento Sucesivo (Stepwise Refinement)

Stepwise Refinement: ejemplo

- Se requiere tener un programa que convierta un número en texto
  - Definimos rango de números a convertir (< 1000)
  - Se toma cada digito de izquierda a derecha
  - Manejo del primer dígito (cientos)
    - Si hay dígito entre 1 – 9 se imprime nombre del dígito ("uno",…, "nueve")
    - Añadir "cientos"
  - Segundo dígito (decenas)
    - Obtener segundo dígito (entero entre 0 - 9)
    - Si es cero pasa a obtener dígito de unidades

Stepwise Refinement: ejemplo

- Si segundo dígito es 0
  - Obtener tercer dígito (entero entre 0 - 9)
  - Obtener nombre del dígito ("uno",…, "nueve")
- Si el segundo dígito es 1
  - Obtener tercer dígito (entero entre 0 - 9)
  - Obtener nombre ("diez", "once",…, "diecinueve")
- Si el segundo dígito es 2 -9
  - Obtener nombre "veinte", "treinta",…, "noventa"
  - Obtener tercer dígito (entero entre 0 - 9)
  - Obtener nombre del dígito ("uno",…, "nueve")

Stepwise Refinement: ejemplo

- digitName
  - Toma un número de 0 - 9
  - Retorna una cadena ("", "uno",…, "nueve")
- tensName (segundo dígito >= 20)
  - Toma un entero entre 0 - 9
  - Retorna una cadena ("veinte", "treinta",…, "noventa") y
    - digitName(tercer digito)
- teenName
  - Toma un entero entre 0 - 9
  - Retorna una cadena ("diez", "once",…, "diecinueve")

Stepwise Refinement: pseudocódigo

```
part = number (parte que aún necesitaser convertido)
name = "" (nombre del número)
if part >= 100
  name = digitName(part / 100) + " ciento"
  elimina cientos de part
if part >= 20
  añade tensName(part) a nombre
  elimina decenas de part
else if part >= 10
  añade teenName(part) a nombre
  part = 0
if (part > 0)
  añade digitName(part) a nombre
```

Stepwise Refinement: código

```
import java.util.Scanner;

public class NombreEntero {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      System.out.print("Ingresar un entero positivo < 1000: ");
      int input = in.nextInt(); System.out.println(intName(input));
   }

   public static String intName(int number) { // convierte entero a Español
      int part = number; String name = "";

      if (part >= 100) {
         name = digitName(part / 100) + " cientos"; part = part % 100;
      } if (part >= 20) {
         name = name + " " + tensName(part); part = part % 10;
      } else if (part >= 10) {
         name = name + " " + teenName(part); part = 0;
      } if (part > 0) {
         name = name + " " + digitName(part);
      } return name;
   }
```

Stepwise Refinement: código

```
/** Convierte un digito en su nombre.
    @param digit entero entre 1 y 9
    @return el nombre del digito ("uno" ... "nueve")
*/
public static String digitName(int digit)
{
  if (digit == 1) { return "uno"; }
  if (digit == 2) { return "dos"; }
  if (digit == 3) { return "tres"; }
  if (digit == 4) { return "cuatro"; }
  if (digit == 5) { return "cinco"; }
  if (digit == 6) { return "seis"; }
  if (digit == 7) { return "siete"; }
  if (digit == 8) { return "ocho"; }
  if (digit == 9) { return "nueve"; }
  return "";
}
```

Stepwise Refinement: código

```
/**
      Convierte un digito entre 10 y 19 en su nombre.
      @param number entero entre 10 y 19
      @return el nombre del numero ("diez" ... "diecinueve")
   */
   public static String teenName(int number)
   {
      if (number == 10) { return "diez"; }
      if (number == 11) { return "once"; }
      if (number == 12) { return "doce"; }
      if (number == 13) { return "trece"; }
      if (number == 14) { return "catorce"; }
      if (number == 15) { return "quince"; }
      if (number == 16) { return "dieciseis"; }
      if (number == 17) { return "diecisiete"; }
      if (number == 18) { return "dieciocho"; }
      if (number == 19) { return "diecinueve"; }
      return "";
   }
```

Stepwise Refinement: código

```
/**
      Obtiene el nombre de la parte decenas de un numero entre 20 y 99.
      @param number entero entre 20 y 99
      @return el nombre de las decenas de un numero ("veinte" ...
   "noventa")
   */
   public static String tensName(int number)
   {
      if (number >= 90) { return "noventa"; }
      if (number >= 80) { return "ochenta"; }
      if (number >= 70) { return "setenta"; }
      if (number >= 60) { return "sesenta"; }
      if (number >= 50) { return "cincuenta"; }
      if (number >= 40) { return "cuarenta"; }
      if (number >= 30) { return "treinta"; }
      if (number >= 20) { return "veinte"; }
      return "";
   }
}
```

Alcance de las variables

- Es la parte del programa en la cual es visible
- Las variables se pueden declarar:
  - Dentro de un método
    - Se conocen como “variables locales”
    - Disponible sólo dentro del método
    - Las variables parámetro son variables locales
  - Dentro de un bloque de código { }
    - Llamadas “alcance de bloque”
    - El alcance termina al final del bloque
  - Fuera del método
    - Llamadas “variables globales”
    - Se pueden usar y modificar por cualquier método

Alcance de las variables: ejemplo

- `sum` es una variable local en main
- `square` sólo es visible dentro del bloque del ciclo for
- `i` sólo es visible dentro del bloque del ciclo for

```
public static void main(String[] args)
{
  int sum = 0;
  for (int i = 1; i <= 10; i++)
  {
    int square = i * i;
    sum = sum + square;
  }
  System.out.println(sum);
}
```

sum

i

square

Variables locales de métodos

- Las variables locales dentro de un método no son visibles en otros métodos
  - `sideLength` es local a main
  - Causa un error de compilación

```
public static void main(String[] args)
{
  double sideLength = 10;
  int result = cubeVolume();
  System.out.println(result);
}

public static double cubeVolume()
{
  return sideLength * sideLength * sideLength; // ERROR
}
```

Reusando nombres de variables locales

- Las variables declaradas dentro de un método no son visibles en otros métodos
  - `result` es local a square y `result` es local a main
  - Son dos variables diferentes y no se solapan

```
public static int square(int n)
{
  int result = n * n;
  return result;
}

public static void main(String[] args)
{
  int result = square(3) + square(4);
  System.out.println(result);
}
```

`result`

`result`

Reusando nombres de variables de bloque

- Las variables declaradas dentro de un bloque no son visibles en otros bloques
  - `i` está en el primer bloque for e `i` está en el segundo
  - Son dos variables diferentes y no se solapan

```
public static void main(String[] args) {
  int sum = 0;
  for (int i = 1; i <= 10; i++) {
    sum = sum + i;
  }
  for (int i = 1; i <= 10; i++) {
    sum = sum + i * i;
  }
  System.out.println(sum);
}
```

`i`

`i`

Solape de alcance

- Las variables, incluyendo los parámetros, deben tener un nombres únicos dentro de su alcance
  - n tiene alcance local y n está es un bloque dentro del alcance
  - El compilador dará error cuando se declara n

```
public static int sumOfSquares(int n) {
  int sum = 0;
  for (int i = 1; i <= n; i++) {
    int n = i * i; // ERROR
    sum = sum + n;
  }
  return sum;
}
```

alcance bloque n

Local n

Solape de variables globales y locales

- Las variables globales y locales (método) se pueden solapar
  - La variable local `same` se usará dentro de su alcance
  - No hay acceso a la v. global `same` cuando la local `same` está en su alcance

```
public class Scoper {
  public static int same;   // global
  public static void main(String[] args) {
    int same = 0;      // local
    for (int i = 1; i <= 10; i++) {
      int square = i * i; same = same + square;
    }
    System.out.println(same);
  }
}
```

same

same

Algoritmos recursivos

- Son algoritmos que expresan la solución de un problema en términos de una llamada a sí mismo (llamada recursiva o recurrente)
- Ejemplo típico: Factorial (n!) de un número

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

- Son más ineficientes que los iterativos pero más simples y elegantes
- Todo algoritmo recursivo tiene su equivalente iterativo

Métodos recursivos – definición y diseño

- Un método recursivo es un método que se llama a sí mismo dentro del cuerpo del método.
- Para diseñar correctamente un algoritmo recursivo, es necesario:
  - Establecer correctamente la ley de recurrencia.
  - Definir el procedimiento de finalización del algoritmo recursivo (normalmente con el valor o valores iniciales).

Métodos recursivos - Verificación

- Para verificar funciones recursivas se aplica el método de las tres preguntas:
  - *pregunta Caso-Base*: Hay una salida no recursiva de la función, y la rutina funciona correctamente para este caso "base"?
  - *pregunta Llamador-Más Pequeño*: Cada llamada recursiva a la función se refiere a un caso más pequeño del problema original?
  - *pregunta Caso-General*: Suponiendo que las llamadas recursivas funcionan correctamente, funciona correctamente toda la función?

Métodos recursivos: ejemplo factorial

```
import java.util.Scanner;

public class Factorialr {
   public static void main(String[] args) {
     Scanner in = new Scanner( System.in );
     System.out.print("Ingresar numero para calcular factorial (>0): ");
     int n = in.nextInt();
     if (n >= 0)
       System.out.print("El factorial de " + n + " es: " + factorialR(n));
     else
       System.out.print("Error en el numero. Debe ser >= 0");
   }

   public static long factorialR(long n)
   {
     if (n == 0)
       return 1;
     else
       return n*factorialR(n-1);
   }
}
```

Métodos recursivos: imprime triangulo

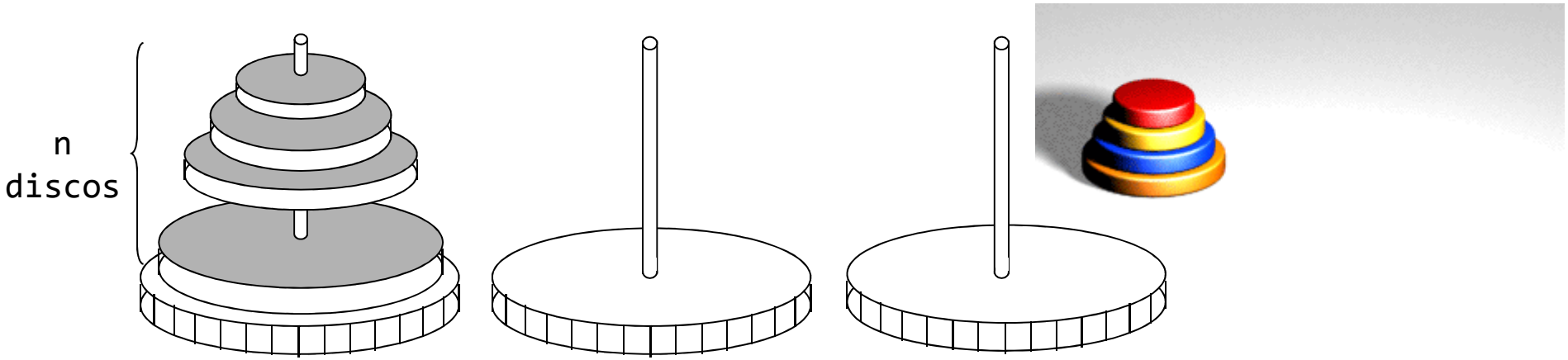
```
public class ImprimeTriangulo
{
   public static void main(String[] args) {
      printTriangle(4);
   }

   /**
      Imprime un triangulo con una longitud de lado dada
      @param sideLength longitud de la fila inferior
   */
   public static void printTriangle(int sideLength)
   {
      if (sideLength < 1) { return; }
      printTriangle(sideLength - 1);

      // Imprime la fila inferior
      for (int i = 0; i < sideLength; i++) {
         System.out.print("[]");
      }
      System.out.println();
   }
}
```

Métodos recursivos – torres de Hanoi

- Juego consistente en tres pivotes y un número de discos de diferentes tamaños apilados.
  - Consiste en mover los discos de un pivote a otro.
  - Sólo se puede mover un disco cada vez
  - Un disco de mayor diámetro nunca puede estar encima de uno de menor diámetro



Métodos recursivos – torres de Hanoi

- Se considera un pivote origen y otro como destino. El otro pivote se usa para almacenamiento temporal.
- El algoritmo para n discos (>0), numerados del más pequeño al más grande, y que los nombres de los pivotes son detorre, atorre y auxtorre es:
  - Mover los n-1 discos superiores del pivote detorre al pivote auxtorre usando el pivote atorre como temporal.
  - Mover el disco n al pivote atorre.
  - Mover los n-1 discos del pivote auxtorre al pivote atorre usando detorre como temporal.

Métodos recursivos: torres de hanoi

```
import java.util.Scanner;

public class TorresHanoi {
   public static void main(String[] args) {
     Scanner in = new Scanner( System.in );
     System.out.print("Torres de Hanoi. Numero de discos: ");
     int n = in.nextInt();
     torresHanoi(n,'A','C','B');
   }

   public static void torresHanoi(int n, char detorre, char atorre,
                                  char auxtorre) {
     if (n > 0)
     {
       torresHanoi(n-1,detorre,auxtorre,atorre);
       System.out.printf("mover disco %d de torre %c a torre %c\n",
                         n, detorre, atorre);
       torresHanoi(n-1,auxtorre,atorre,detorre);
     }
     return;
   }
}
```

Métodos recursivos: cambio de base

- Se requiere un programa para cambiar un número entero (base 10) en otra base (2 - 16)
- El algoritmo para cambiar de base es:
  - dividir sucesivamente hasta que el cociente sea menor que la base.
  - los dígitos del número resultante se forman agrupando, de derecha a izquierda, el último cociente y los restos obtenidos durante la división desde el último al primero.
  - Si los dígitos superan la base 10 se utilizan letras.
- Ejemplo: $17_{10} = 10001_2$

Métodos recursivos: cambio de base

```
import java.util.Scanner;

public class CambiaBase
{
   public static void main(String[] args) {
     String TablaDigitos = "0123456789abcdef";
     Scanner in = new Scanner( System.in );
     System.out.print("Ingresar numero a cambiar base (>0): ");
     int n = in.nextInt();
     System.out.print("Ingresar base (2 - 16): ");
     int b = in.nextInt();
     System.out.print("El numero " + n + " en base " + b + " es: ");
     cambiaBase(n, b, TablaDigitos);
   }

   public static void cambiaBase( int N, int base, String TablaD) {
     if( N >= base )
        cambiaBase( N / base, base, TablaD  );
     System.out.print( TablaD.charAt( N % base ) );
   }
}
```