Métodos en Java

Un **método en Java** es un conjunto de instrucciones definidas dentro de una clase, que realizan una determinada tarea y a las que podemos invocar mediante un nombre.

Algunos métodos que hemos utilizado hasta ahora:
– Math.pow()
– Math.sqrt()
– Character.isDigit()
- System.out.println();

Cuando se llama a un método, la ejecución del programa pasa al método y cuando éste acaba, la ejecución continúa a partir del punto donde se produjo la llamada.

Utilizando métodos:

- Podemos construir programas modulares.
- Se consigue la reutilización de código. En lugar de escribir el mismo código repetido cuando se necesite, por ejemplo para validar una fecha, se hace una llamada al método que lo realiza.

En Java un método siempre pertenece a una clase.

Todo programa java tiene un método llamado main. Este método es el punto de entrada al programa y también el punto de salida.

1. ESTRUCTURA GENERAL DE UN MÉTODO JAVA

La estructura general de un método Java es la siguiente:

```
[especificadores] tipoDevuelto nombreMetodo([lista parámetros]) [throws listaExcepciones]
{
  // instrucciones
  [return valor;]
}
```

Los elementos que aparecen entre corchetes son opcionales.

especificadores (opcional): determinan el tipo de acceso al método. Se verán en detalle más adelante.

tipoDevuelto: indica el tipo del valor que devuelve el método. En Java es imprescindible que en la declaración de un método, se indique el tipo de dato que ha de devolver. El dato se

devuelve mediante la instrucción return. Si el método no devuelve ningún valor este tipo será void.

nombreMetodo: es el nombre que se le da al método. Para crearlo hay que seguir las mismas normas que para crear nombres de variables.

Lista de parámetros (opcional): después del nombre del método y siempre entre paréntesis puede aparecer una lista de parámetros (también llamados argumentos) separados por comas. Estos parámetros son los datos de entrada que recibe el método para operar con ellos. Un método puede recibir cero o más argumentos. Se debe especificar para cada argumento su tipo. **Los paréntesis son obligatorios** aunque estén vacíos.

throws listaExcepciones (opcional): indica las excepciones que puede generar y manipular el método.

return: se utiliza para devolver un valor. La palabra clave return va seguida de una expresión que será evaluada para saber el valor de retorno. Esta expresión puede ser compleja o puede ser simplemente el nombre de un objeto, una variable de tipo primitivo o una constante.

El tipo del valor de retorno debe coincidir con el tipoDevuelto que se ha indicado en la declaración del método.

Si el método no devuelve nada (tipoDevuelto = void) la instrucción return es opcional.

Un método puede devolver un tipo primitivo, un array, un String o un objeto.

Un método tiene un único punto de inicio, representado por la llave de inicio {. La ejecución de un método termina cuando se llega a la llave final } o cuando se ejecuta la instrucción return.

La instrucción return puede aparecer en cualquier lugar dentro del método, no tiene que estar necesariamente al final.

2. IMPLEMENTACIÓN DE MÉTODOS EN JAVA

Pasos para implementar un método:

1. Describir lo que el método debe hacer
2. Determinar las entradas del método
3. Determinar los tipos de las entradas
4. Determinar el tipo del valor retornado
5. Escribir las instrucciones que forman el cuerpo del método
6. Prueba del método: diseñar distintos casos de prueba

Ejemplo de método: método que suma dos números enteros.

```
import java.util.*;
public class Metodos1 {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int numero1, numero2, resultado;
    System.out.print("Introduce primer número: ");
    numero1 = sc.nextInt();
    System.out.print("Introduce segundo número: ");
    numero2 = sc.nextInt();
    resultado = sumar(numero1, numero2);
    System.out.println("Suma: " + resultado);
  }
  public static int sumar(int a, int b){
      int c;
      c = a + b;
      return c;
  }
}
```

El método se llama sumar y recibe dos números enteros a y b. En la llamada al método **los valores de las variables numero1 y numero2 se copian en las variables a y b.** El método suma los dos números y guarda el resultado en c. Finalmente devuelve mediante la instrucción return la suma calculada.

Ejemplo de programa Java que contiene un método con varios return:

Programa que lee por teclado un año y calcula y muestra si es bisiesto. Para realizar el cálculo utiliza un método llamado esBisiesto.

```
package bisiesto;
import java.util.*;
public class Bisiesto {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int año;
    System.out.print("Introduce año: ");
    año = sc.nextInt();
    if(esBisiesto(año))  //llamada al método
      System.out.println("Bisiesto");
    else
      System.out.println("No es bisiesto");
```

```
  }
  /**
   * método que calcula si un año es o no bisiesto
   */
  public static boolean esBisiesto(int a){
    if(a%4==0 && a%100!=0 || a%400==0)
      return true;
    else
      return false;
  }
}
```

En la llamada al método bisiesto, el valor de la variable año se copia en la variable a y el método trabaja con esta variable. El valor true ó false devuelto por return pasa a ser el valor de la condición.

Ejemplo de programa Java: Método que no devuelve ningún valor. El método cajaTexto recibe un String y lo muestra rodeado con un borde.

El tipo devuelto es void y no es necesario escribir la sentencia return. El método acaba cuando se alcanza la llave final.

```
import java.util.*;
public class MetodoVoid {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String cadena;
    System.out.print("Introduce cadena de texto: ");
    cadena = sc.nextLine();
    cajaTexto(cadena); //llamada al método
  }
 /**
   * método que muestra un String rodeado por un borde
   */
  public static void cajaTexto(String str){
       int n = str.length();
       for (int i = 0; i < n + 4; i++){
          System.out.print("#");
       }
       System.out.println();
       System.out.println("# " + str + " #");
       for (int i = 0; i < n + 4; i++){
          System.out.print("#");
       }
```

```
        System.out.println();
    }
}
```

Paso de parámetros en Java y ámbito de las variables

1. PARÁMETROS ACTUALES Y FORMALES

Parámetros actuales: Son los argumentos que aparecen en la llamada a un método. Contienen los valores que se le pasan al método. Un parámetro actual puede ser una variable, un objeto, un valor literal válido, etc.

Parámetros formales: Son los argumentos que aparecen en la cabecera del método. Reciben los valores que se envían en la llamada al método. Se utilizan como variables normales dentro del método.

Los parámetros actuales y los formales **deben coincidir en número, orden y tipo**. Si el tipo de un parámetro actual no coincide con su correspondiente parámetro formal, el sistema lo convertirá al tipo de este último, siempre que se trate de tipos compatibles. Si no es posible la conversión, el compilador dará los mensajes de error correspondientes.

Si el método devuelve un valor, la llamada al método puede estar incluida en una expresión que recoja el valor devuelto.

2. ÁMBITO DE UNA VARIABLE

El ámbito o alcance de una variable es la zona del programa donde la variable es accesible.

El ámbito lo determina el lugar donde se declara la variable.

En Java las variables se pueden clasificar según su ámbito en:

- Variables miembro de una clase o atributos de una clase
- Variables locales
- Variables de bloque

Variables miembro o atributos de una clase

Son las declaradas dentro de una clase y fuera de cualquier método.

Aunque suelen declararse al principio de la clase, se pueden declarar en cualquier lugar siempre que sea fuera de un método.

Son accesibles en cualquier método de la clase.

Pueden ser inicializadas.

Si no se les asigna un valor inicial, el compilador les asigna uno por defecto:

- 0 para las numéricas
- '\0' para las de tipo char
- null para String y resto de referencias a objetos.

Variables locales

Son las declaradas dentro de un método.

Su ámbito comienza en el punto donde se declara la variable.

Están disponibles desde su declaración hasta el final del método donde se declaran.

No son visibles desde otros métodos.

Distintos métodos de la clase pueden contener variables con el mismo nombre. Se trata de variables distintas.

El nombre de una variable local debe ser único dentro de su ámbito.

Si se declara una variable local con el mismo nombre que una variable miembro de la clase, la variable local oculta a la miembro. La variable miembro queda inaccesible en el ámbito de la variable local con el mismo nombre.

Se crean en memoria cuando se declaran y se destruyen cuando acaba la ejecución del método.

No tienen un valor inicial por defecto. El programador es el encargado de asignarles valores iniciales válidos.

Los parámetros formales son variables locales al método.

Variables de bloque

Son las declaradas dentro de un bloque de instrucciones delimitado por llaves { }.

Su ámbito comienza en el punto donde se declara la variable.

Están disponibles desde su declaración hasta el final del bloque donde se declaran.

No son visibles desde otros bloques.

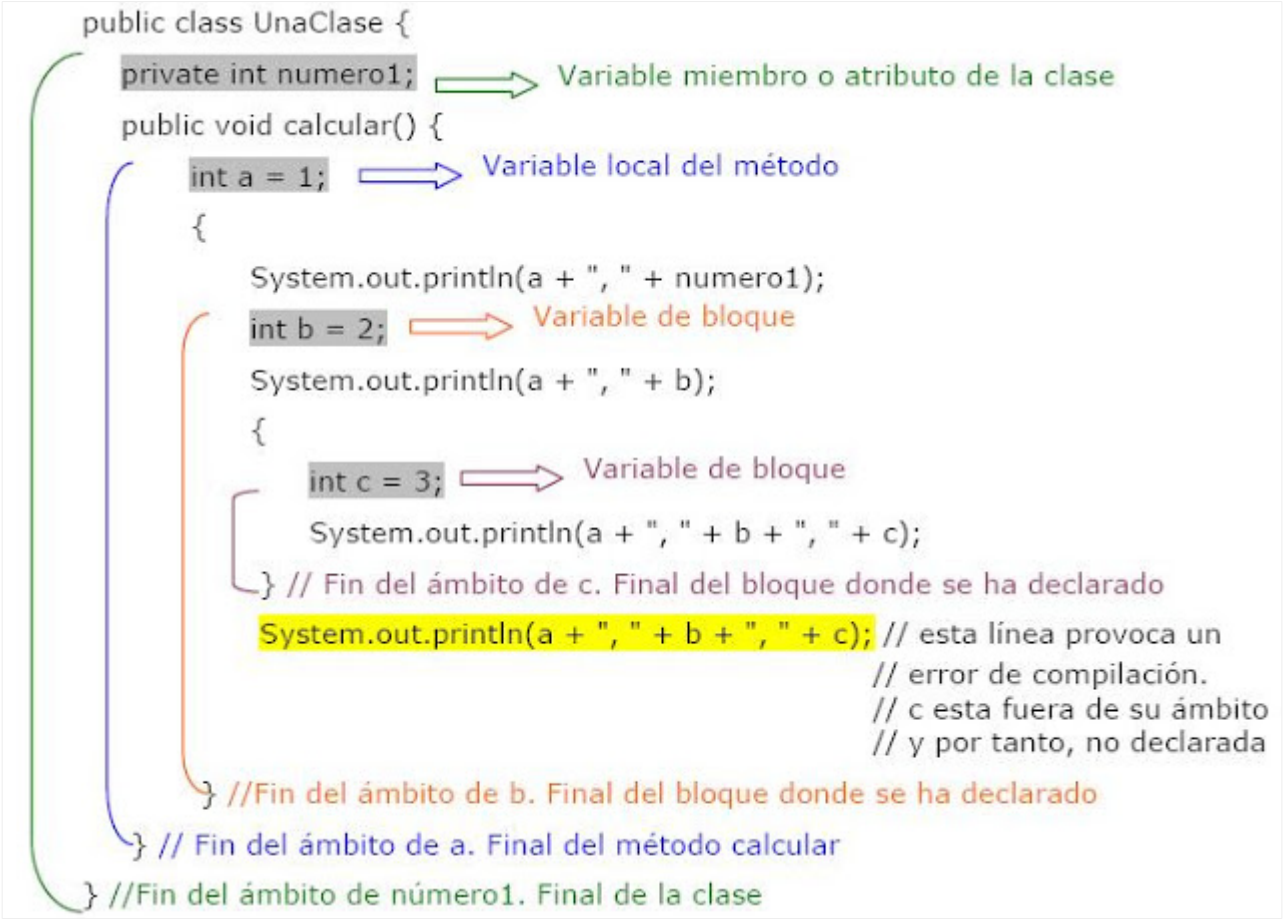
Distintos bloques pueden contener variables con el mismo nombre. Se trata de variables distintas.

Si un bloque de instrucciones contiene dentro otro bloque de instrucciones, en el bloque interior no se puede declarar una variable con el mismo nombre que otra del bloque exterior.

Se crean en memoria cuando se declaran y se destruyen cuando acaba la ejecución del bloque.

No tienen un valor inicial por defecto. El programador es el encargado de asignarles valores iniciales válidos.

Ejemplo de variables de bloque en java:



Un ejemplo típico de declaración de variables dentro de un bloque es hacerlo en las instrucciones for:

```
for(int i = 1; i<=20; i+=2){
    System.out.println(i);
}
```

La variable i se ha declarado dentro del for y solo es accesible dentro de él.

Si a continuación de la llave final del for escribimos:

```
System.out.println(i);
```

se producirá un error de compilación: la variable i no existe fuera del for y el compilador nos dirá que no está declarada.

En el siguiente ejemplo se declara la variable x en cada bloque for:

```
public static void main(String[] args) {
   int suma = 0;
   for (int x = 1; x <= 10; x++) {
       suma = suma + x;
   }
   for (int x = 1; x <= 10; x++) {
      suma = suma + x * x;
```

```
    }
    System.out.println(suma);
}
```

Las variables locales deben tener nombres únicos dentro de su alcance.

Ejemplo de solapamiento de variables locales y de bloque que producen un error de compilación:

```
public static int sumaCuadrados(int n) {
    int suma = 0;
    for (int i = 1; i <= n; i++) {
        int n = i * i; // ERROR n ya declarada
        suma = suma + n;
     }
     return suma;
}
```

3. PASO DE PARÁMETROS EN JAVA

En programación hay dos formas de paso de parámetros a un método:

Paso de parámetros por valor

Cuando se invoca al método se crea una nueva variable (el parámetro formal) y se le copia el valor del parámetro actual.

El parámetro actual y el formal son dos variables distintas aunque tengan el mismo nombre.

El método trabaja con la copia de la variable por lo que cualquier modificación que se realice sobre ella dentro del método no afectará al valor de la variable fuera.

En definitiva, el método no puede modificar el valor de la variable original.

Paso de parámetros por referencia

Cuando se invoca al método se crea una nueva variable (el parámetro formal) a la que se le asigna la dirección de memoria donde se encuentra el parámetro actual.

En este caso el método trabaja con la variable original por lo que puede modificar su valor.

Lenguajes como C, C++, C#, php, VisualBasic, etc. soportan ambas formas de paso de parámetros.

Pero, cómo se realiza el paso de parámetros en Java?

En Java todos los parámetros se pasan por valor

Cuando se realiza la llamada a un método, los parámetros formales reservan un espacio en memoria y reciben los valores de los parámetros actuales.

Cuando el **argumento** es de **tipo primitivo** (int, double, char, boolean, float, short, byte), el paso por valor significa que cuando se invoca al método se reserva un nuevo espacio en memoria para el parámetro formal. **El método no puede modificar el parámetro actual.**

Cuando el **argumento** es una **referencia a un objeto** (por ejemplo, un *array* o cualquier otro objeto) el paso por valor significa que el método recibe una copia de la dirección de memoria

donde se encuentra el objeto. La referencia no puede modificarse pero **sí se pueden modificar los contenidos de los objetos** durante la ejecución del método.

Tipos primitivos: no se pueden modificar

Arrays y objetos: se pueden modificar

Ejemplo de paso de parámetros por valor. La variable n no se modifica.

```
public static void main(String[] args) {
   int n = 1;
  System.out.println("Valor de n en main antes de llamar al método:" + n);
 incrementar(n);
 System.out.println("Valor de n en main después de llamar al método:" + n);
}

public static void incrementar(int n){
      n++;
     System.out.println("Valor de n en el método después incrementar:" + n);
}
```

La salida de este programa es:

Valor de n en main antes de llamar al método: 1

Valor de n en el método después incrementar: 2

Valor de n en main después de llamar al método: 1

Si es necesario que un método modifique un tipo primitivo que recibe como parámetro podemos utilizar un array auxiliar.

Cuando se pasa un array a un método se pasa la dirección de memoria donde se encuentra por lo tanto el método puede modificarlo.

```
public static void main(String[] args) {
 int n = 1;
 int [] aux = new int[1]; //aux: array auxiliar para el paso de parámetros
 System.out.println("Valor de n en main antes de llamar al método: " + n);
   aux[0]=n;
   incrementar(aux);
   n = aux[0];
   System.out.println("Valor de n en main después de llamar al método: "
                                                                  + n);
}
public static void incrementar(int [] x){
  x[0]++;
 System.out.println("Valor de n en el método después incrementar: "
                                                               + x[0]);
}
```

La salida de este programa es:

Valor de n en main antes de llamar al método: 1

Valor de n en el método después incrementar: 2

Valor de n en main después de llamar al método: 2