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.

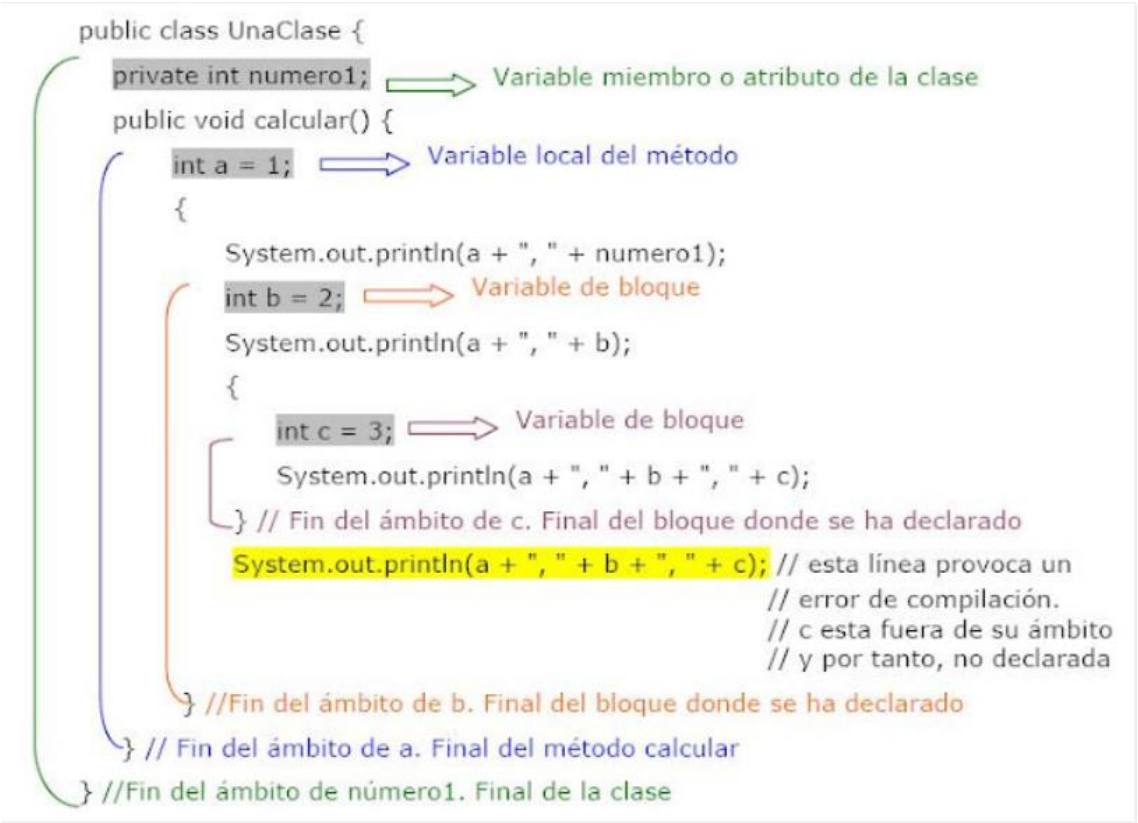
Distintos bloques pueden contener variables con el mismo nombre. Se trata de variables distintas.

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
```