Erick Antillón

# Introducción a la programación y computación 1

Clase 2 - Programación en Java

# Agenda

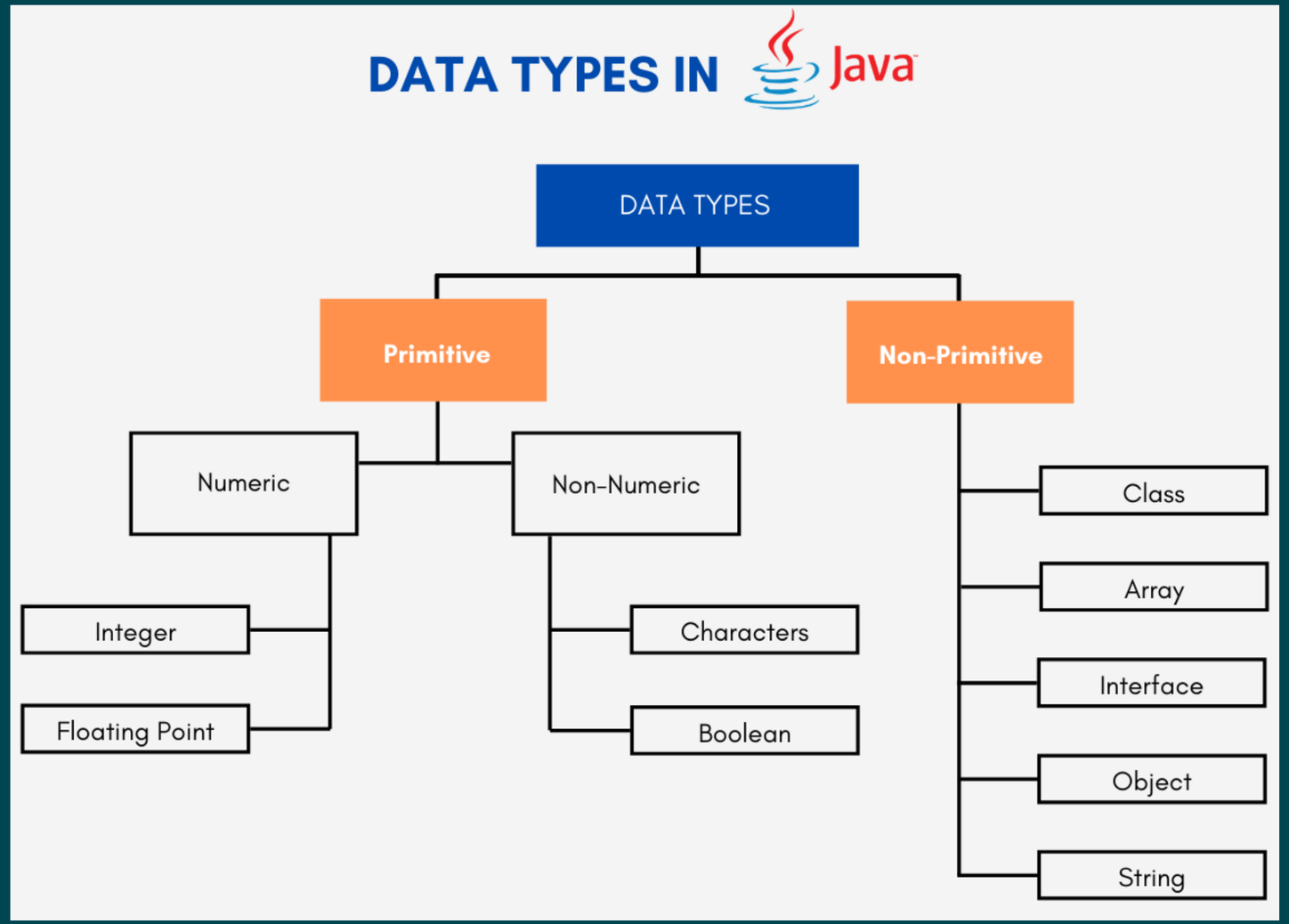
- Avisos
- Tipos de datos
- Operadores y jerarquía
- Control de flujo
- Arreglos y ordenamiento
- Funciones y recursividad
- Ejemplo práctico
- Asistencia

# Avisos

- Asignación DTT
- Curso de Python SAE/SAP
- Práctica 1 - 12:20
  https://meet.google.com/ggw-wtkm-gaf

# Tipos de datos en Java

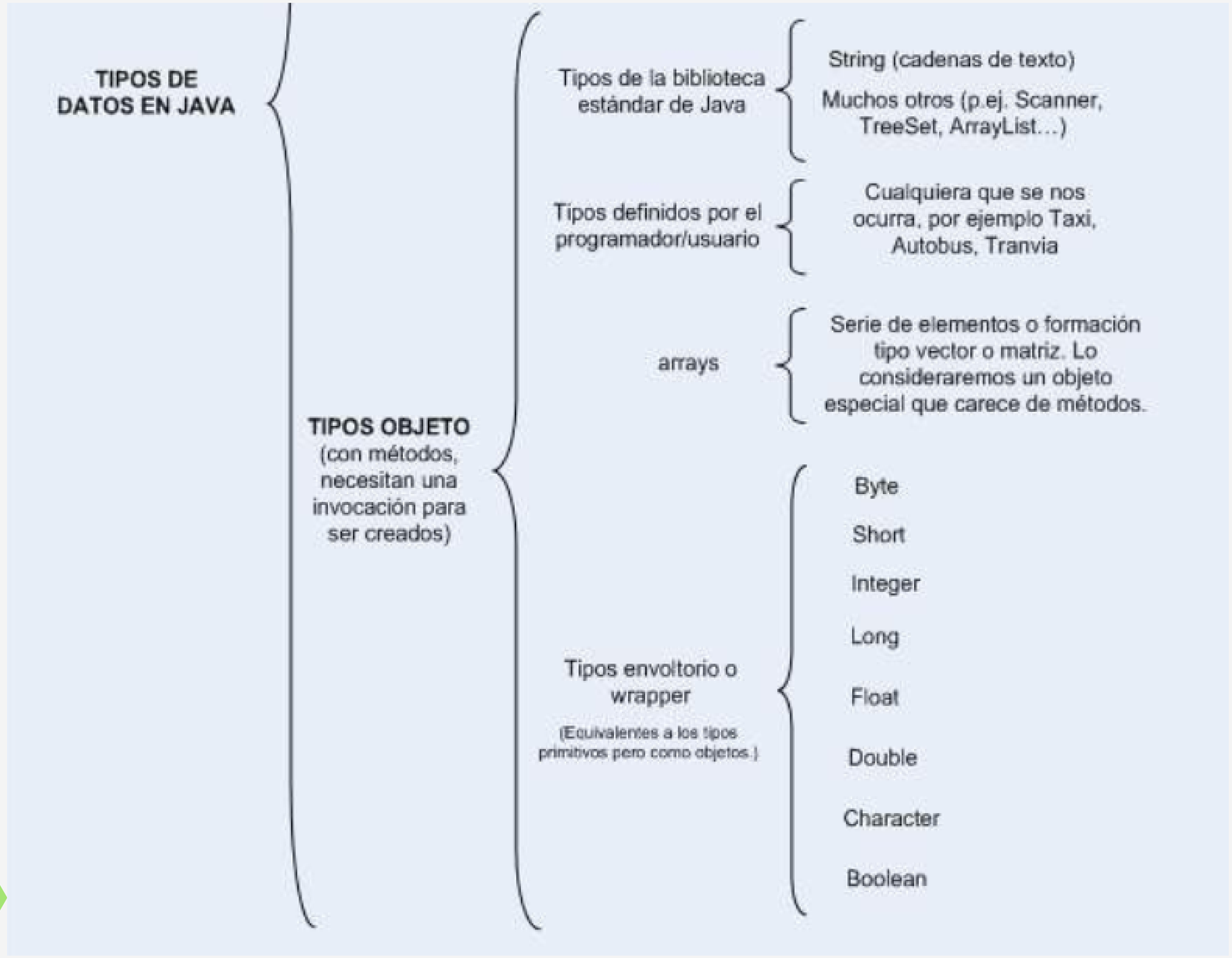
## Tipos primitivos y no primitivos



# Primitivos

| VARIABLES DE TIPOS PRIMITIVOS. | | | | | |
|---|---|---|---|---|---|
| **Nombre** | **Tipo** | **Tamaño** | **Valor por defecto** | **Forma de inicializar** | **Rango** |
| Boolean | Lógico | 1 bit | False | Boolean a=true | True-false |
| Char | Carácter | 16 bits | Null | Char a='Z' | Unicode |
| Byte | Numero entero | 8 bits | 0 | Byte a =0 | -128 a 127 |
| Short | Numero entero | 16 bits | 0 | Short a =12 | -32.768 a 32.767 |
| Int | Numero entero | 32 bit | 0 | Int a= 1250 | -2.147.483.648 a 2.147.483.649 |
| Long | Numero entero | 64 bits | 0 | Long a= 125000 | -9*10^18 a 9*10^18 |
| Float | Numero real | 32 bits | 0 | Float a =3.1 | -3,4*10^38 a 3,4*10^38 |
| Double | Numero real | 64 bits | 0 | Double a = 125.2333 | -1,79*10^308 a 1,79*10^308 |

# No Primitivos



# Operadores Logicos

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| == | igual que | `7 == 38` | `false` |
| != | distinto que | `'a' != 'k'` | `true` |
| < | menor que | `'G' < 'B'` | `false` |
| > | mayor que | `'b' > 'a'` | `true` |
| <= | menor o igual que | `7.5 <= 7.38` | `false` |
| >= | mayor o igual que | `38 >= 7` | `true` |

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| ! | Negación - NOT (unario) | `!false`<br>`!(5==5)` | `true`<br>`false` |
| \| | Suma lógica – OR (binario) | `true \| false`<br>`(5==5)\|(5<4)` | `true`<br>`true` |
| ^ | Suma lógica exclusiva – XOR (binario) | `true ^ false`<br>`(5==5)\|(5<4)` | `true`<br>`true` |
| & | Producto lógico – AND (binario) | `true & false`<br>`(5==5)&(5<4)` | `false`<br>`false` |
| \|\| | Suma lógica con cortocircuito: si el primer operando es `true` entonces el segundo se salta y el resultado es `true` | `true \|\| false`<br>`(5==5)\|\|(5<4)` | `true`<br>`true` |
| && | Producto lógico con cortocircuito: si el primer operando es `false` entonces el segundo se salta y el resultado es `false` | `false && true`<br>`(5==5)&&(5<4)` | `false`<br>`false` |

https://www.arkaitzgarro.com/java/capitulo-4.html

# Operadores Aritemeticos

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| - | operador unario de cambio de signo | `-4` | `-4` |
| + | Suma | `2.5 + 7.1` | `9.6` |
| - | Resta | `235.6 - 103.5` | `132.1` |
| * | Producto | `1.2 * 1.1` | `1.32` |
| / | División (tanto entera como real) | `0.050 / 0.2`<br>`7 / 2` | `0.25`<br>`3` |
| % | Resto de la división entera | `20 % 7` | `6` |

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| ++ | Incremento<br>`i++` primero se utiliza la variable y luego se incrementa su valor<br>`++i` primero se incrementa el valor de la variable y luego se utiliza | `4++`<br>`a=5;`<br>`b=a++;`<br>`a=5;`<br>`b=++a;` | `5`<br><br>`a vale 6 y b vale 5`<br><br>`a vale 6 y b vale 6` |
| -- | decremento | `4--` | `3` |

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| += | Suma combinada | `a+=b` | `a=a+b` |
| -= | Resta combinada | `a-=b` | `a=a-b` |
| *= | Producto combinado | `a*=b` | `a=a*b` |
| /= | División combinada | `a/=b` | `a=a/b` |
| %= | Resto combinado | `a%=b` | `a=a%b` |

https://www.arkaitzgarro.com/java/capitulo-4.html

# Jerarquia de Operaciones



| Prior. | Operador | Tipo de operador | Operación |
|---|---|---|---|
| 1 | ++<br>--<br>+, -<br>~<br>¡ | Aritmético<br>Aritmético<br>Aritmético<br>Integral<br>Booleano | Incremento previo o posterior (unario)<br>Incremento previo o posterior (unario)<br>Suma unaria, Resta unaria<br>Cambio de bits (unario)<br>Negación (unario) |
| 2 | (tipo) | Cualquiera | |
| 3 | *, /, % | Aritmético | Multiplicación, división, resto |
| 4 | +, -<br>+ | Aritmético<br>Cadena | Suma, resta<br>Concatenación de cadenas |
| 5 | <<<br>>><br>>>> | Integral<br>Integral<br>Integral | Desplazamiento de bits a izquierda<br>Desplazamiento de bits a derecha con inclusión de signo<br>Desplazamiento de bits a derecha con inclusión de cero |
| 6 | <, <=<br>>, >=<br>instanceof | Aritmético<br>Aritmético<br>Objeto, tipo | Menor que, Menor o igual que<br>Mayor que, Mayor o igual que<br>Comparación de tipos |
| 7 | ==<br>¡=<br>==<br>¡= | Primitivo<br>Primitivo<br>Objeto<br>Objeto | Igual (valores idénticos)<br>Desigual (valores diferentes)<br>Igual (referencia al mismo objeto)<br>Desigual (referencia a distintos objetos) |
| 8 | &<br>& | Integral<br>Booleano | Cambio de bits AND<br>Producto booleano |
| 9 | ^<br>^ | Integral<br>Booleano | Cambio de bits XOR<br>Suma exclusiva booleana |
| 10 | \|<br>\| | Integral<br>Booleano | Cambio de bits OR<br>Suma booleana |
| 11 | && | Booleano | AND condicional |
| 12 | \|\| | Booleano | OR condicional |
| 13 | ? : | Booleano, cualquiera, cualquiera | Operador condicional (ternario) |
| 14 | =<br>*=, /=, %=<br>+=, -=<br><<=, >>=<br>>>>=<br>&=, ^=, \|= | Variable, cualquiera | Asignación<br>Asignación con operación |



https://www.arkaitzgarro.com/java/capitulo-4.html

# Control de flujo

## Tipos de control de flujo

El control de flujo se refiere a la forma en que se ejecutan las instrucciones en un programa. La estructura de control de flujo permite que un programa tome decisiones y repita acciones.

```
// Estructuras de selección

if (expresionLogica) {
    sentencia_1;
}

if (expresionLogica) {
    sentencia_1;
} else {
    sentencia_2;
}
```

```
switch (expression) {
  case value1:
    // secuencia de sentencias.
    break;
  case value2:
    // secuencia de sentencias.
    break;
  case valueN :
    // secuencia de sentencias.
    break;
  default:
}
```

# Control de flujo

## Estructuras de bucle

Permiten que un programa repita acciones.

```
for (inicio; termino; iteracion) {
    sentencia_1;
    sentencia_2;
    sentencia_n;
}

while (expresionLogica) {
    sentencias;
}

do {
    sentencias;
    [iteracion;]
} while (expresionLogica);
```

# Control de flujo

## Jump statements

Permiten controlar la ejecución de un programa mediante la transferencia de control a otra parte del programa.

---

## Estructuras de control de excepción

Permiten manejar excepciones, que son eventos anómalos que ocurren durante la ejecución del programa. El más comun es "**try-catch**"

---

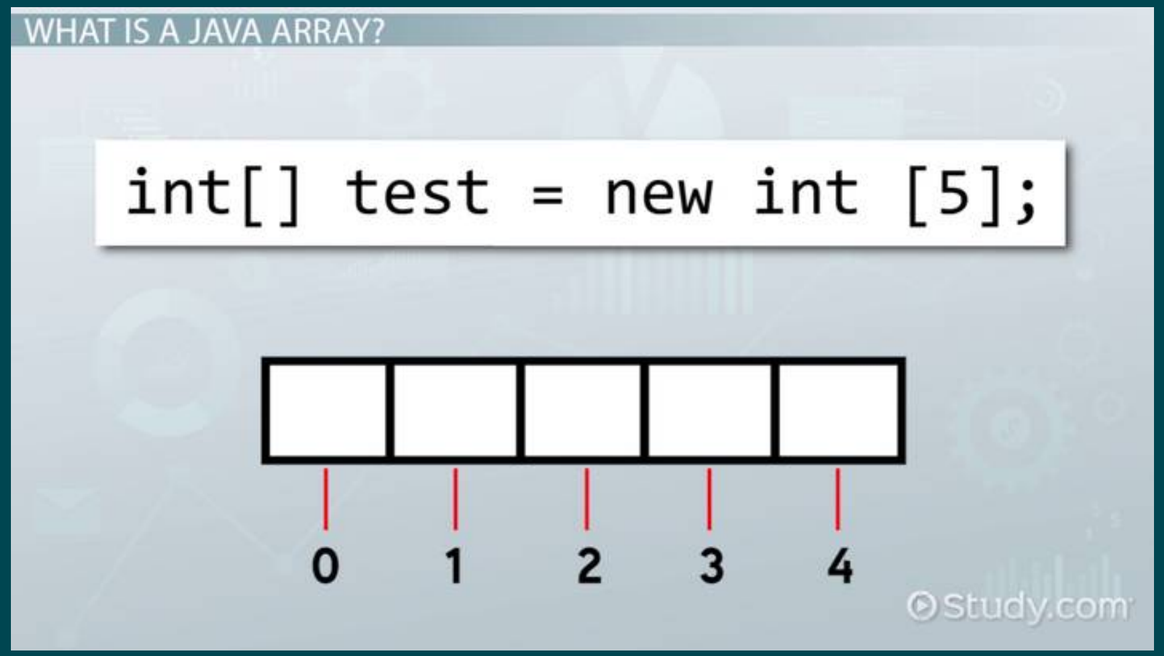
## Estructura "for each"

Permite iterar sobre un arreglo o una colección de datos de una manera más sencilla, es utilizada para recorrer un arreglo o una colección, no se requiere un contador o una variable de control.



12

# Arreglos estáticos




- En Java, los arreglos son estructuras de datos que permiten almacenar un conjunto de elementos del mismo tipo.

```java
// Declarar sin inicializar
int[] myArray1;

// Con inicialización
int[] myArray2 = new int[5];

// Con inicialización de valores
int[] myArray3 = {1,2,3,4,5};
```

- Se utiliza el índice del elemento para acceder al valor almacenado en esa posición. Ejemplo: int value = myArray[2];

# Funciones

En Java, una función se conoce como método. Los métodos son bloques de código que realizan una tarea específica y pueden ser reutilizados en varios lugares del programa.

La estructura básica de un método en Java es la siguiente:

```
modificadorDeAcceso tipoDeRetorno nombreDelMétodo(listaDeParámetros) {
    // cuerpo del método
}

// ejemplo
public void imprimirMensaje(String mensaje) {
    System.out.println(mensaje);
}
```

Donde:

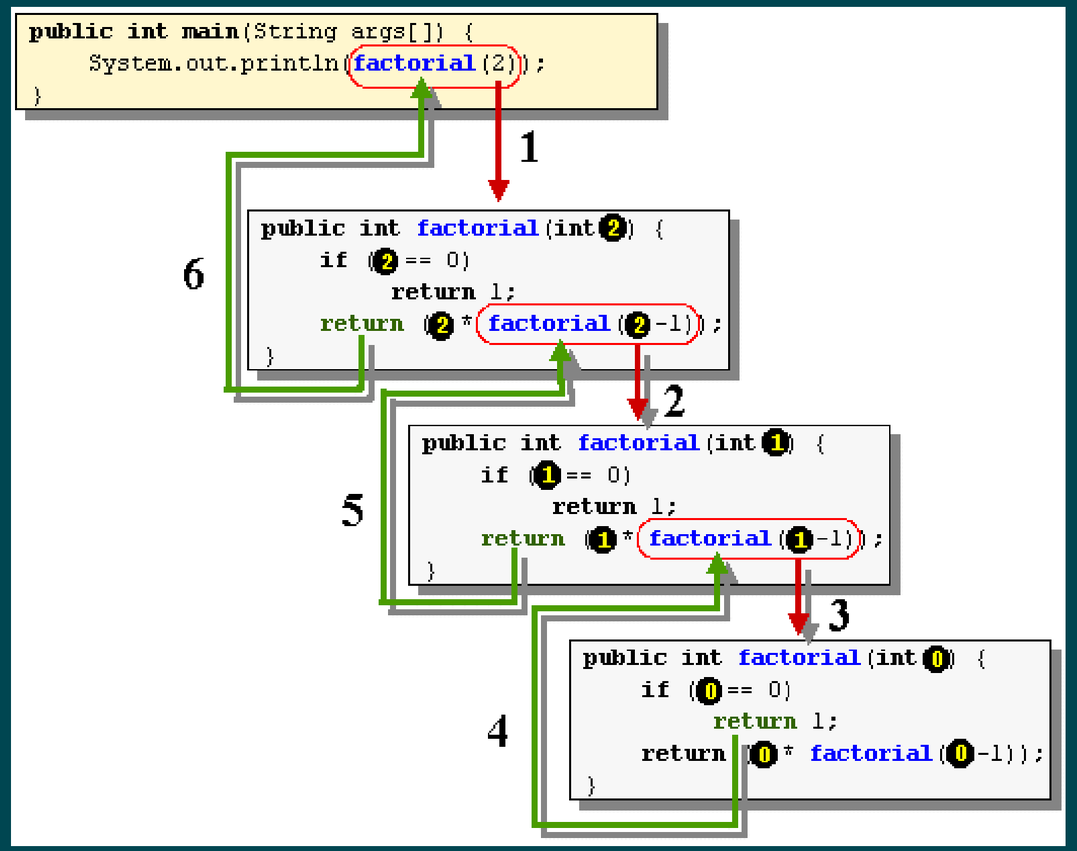
- **modificadorDeAcceso:** es opcional y puede ser ***public, private, protected***, entre otros.
- **tipoDeRetorno:** es el tipo de dato que devuelve el método, puede ser cualquier tipo de dato primitivo o una clase. Si el método no devuelve ningún valor, se utiliza la palabra reservada ***void***.
- **nombreDelMétodo:** es un identificador que se utiliza para llamar al método.
- **listaDeParámetros:** es opcional y es una lista de variables que se pasan al método.

```java
// funcion sin "return" se coloca void
public void imprimirMensaje(String mensaje) {
    System.out.println(mensaje);
}

// funcion que devuelve un valor se coloca el tipo y lleva
"return"
public double areaCirculo(double radio) {
    double area = 3.14 * radio * radio;
    return area;
}

double areaCalculada = areaCirculo(5) // 78.54
```

# Recursividad

La recursión es una técnica de programación en la que un método llama a sí mismo para resolver un problema. Un método recursivo tiene dos partes: la base del caso y el caso recursivo. La base del caso es una condición específica que detiene la recursión, mientras que el caso recursivo es el proceso de llamada al método dentro del propio método.

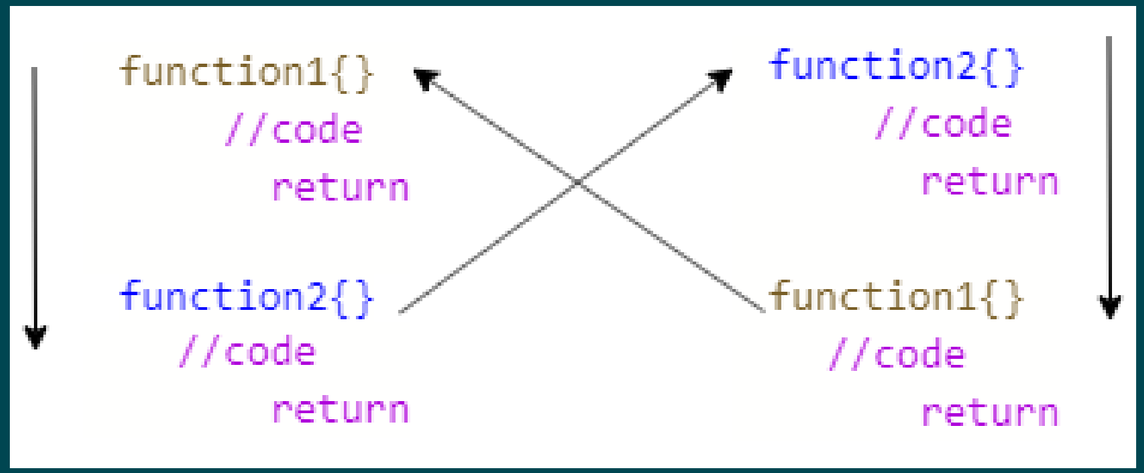
```java
// Calcular factorial de un numero entero positivo
public static int factorial(int n) {
    if (n == 0) {
        return 1;
    }
    return n * factorial(n - 1);
}
```



# Recursividad

Existen dos tipos principales de recursión en programación:

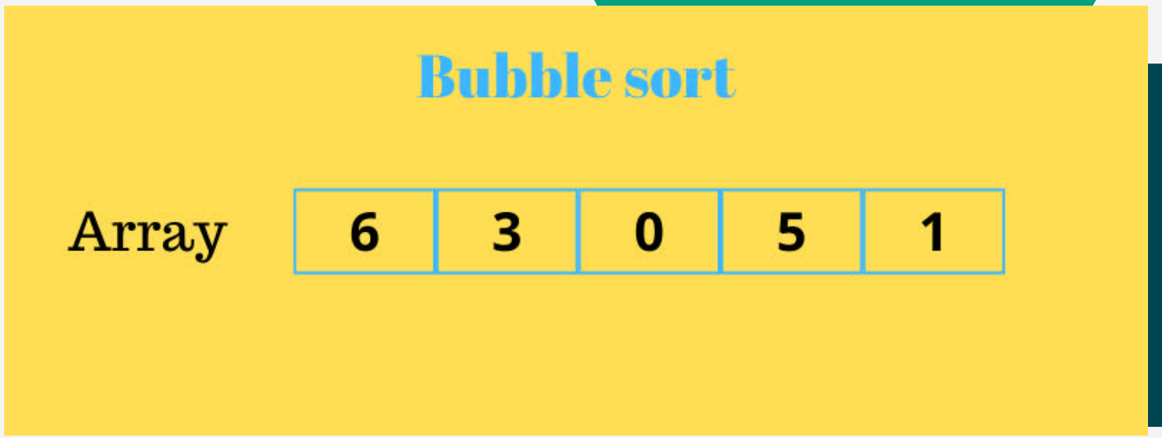
- **Recursión directa**: Es cuando un método se llama a sí mismo de manera directa, como en los ejemplos que mencioné anteriormente.
- **Recursión indirecta**: Es cuando un método llama a otro método que a su vez llama al primer método, generando un ciclo de llamadas recursivas. Un ejemplo de esto sería un método "A" que llama a un método "B" y este a su vez llama al método "A" de nuevo.



# Métodos de ordenamiento

## Ordenamiento de burbuja

El ordenamiento de burbuja es un algoritmo de ordenamiento simple que se basa en la comparación y el intercambio de elementos adyacentes en un arreglo. El algoritmo se repite hasta que no se realizan más intercambios, lo que indica que el arreglo está ordenado.




Se recorre el arreglo de la primera posición hasta la penúltima posición

En cada iteración, se comparan dos elementos adyacentes, si el primer elemento es mayor que el segundo, se intercambian los valores.

Se repite este proceso hasta que no se realizan más intercambios.

Este proceso se repite varias veces, cada vez que se recorre el arreglo, el elemento más grande se va moviendo hacia la última posición, y se va ordenando el arreglo.

# Ejemplo Práctico

Creación, manejo de arreglos, y ordenamiento. Funciones.

```
60
61          // llamada de una funcion
62          imprimirTipoNumero(numeros);
63
64          // Ordenamiento burbuja
65          for (int i = 0; i < numeros.length-1; i++) {
66              for (int j = 0; j < numeros.length-i-1; j++) {
67                  if (numeros[j] > numeros[j+1]) {
68                      int temp = numeros[j];
69                      numeros[j] = numeros[j+1];
70                      numeros[j+1] = temp;
71                  }
72              }
73          }
74
75          //
76          System.out.println("\nNúmeros ordenados:");
77          System.out.println(Arrays.toString(numeros));
78
```

# ¿Dudas o preguntas?

# Asistencia

- Formulario de Asistencia: https://forms.gle/9YetgXxkbVbJ9uAk9