UTEC
Universidad Tecnológica

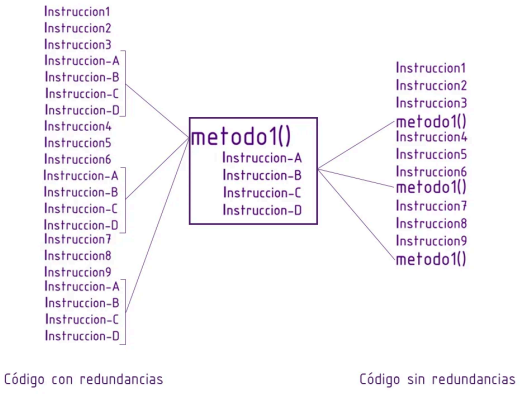
Modularidad

Cuando un programa comienza a ser largo y complejo (la mayoría de los problemas reales se solucionan con programas de este tipo), no es apropiado tener un único texto con sentencias una tras otra. La razón es que no se comprende bien qué hace el programa debido a que se intenta abarcar toda la solución a la vez. Asimismo el programa se vuelve monolítico y difícil de modificar, además suelen aparecer trozos de código muy similares entre sí repetidos a lo largo de todo el programa.

Para solucionar estos problemas y proporcionar otras ventajas adicionales a la programación, los lenguajes de alto nivel suelen disponer de una herramienta que permite estructurar el programa principal como compuesto de subprogramas (métodos) que resuelven problemas parciales del problema principal. A su vez, cada uno de estos subprogramas puede estar resuelto por otra conjunción de problemas parciales, etc. Los procedimientos y funciones son mecanismos de estructuración que permiten ocultar los detalles de la solución de un problema y resolver una parte de dicho problema en otro lugar del código.

Método

Secuencia ordenada de pasos muy precisos que conducen a la solución de un problema, en un tiempo finito.



UTEC
Universidad Tecnológica

Vamos a ver como “encapsular” código y abstraernos de cómo está implementado determinado algoritmo. Lo que en pseudo-código vimos como sub rutina, en Java le llamaremos método.

Sintaxis de definición de método:

[especificadores] tipoRetorno nombre([parámetros de entrada]) {
// instrucciones dentro del bloque del método
[return valor;]
}
Nota: los elementos que aparecen entre [ ] son opcionales.

¿Qué es la firma de un método ?
Ejemplo:

```
public static int sumar(int a, int b) {
  -esto es el cuerpo del método
}
```

Esta primer linea o cabezal que es la definición del método, esta marcado con color amarillo es lo que se conoce como la firma del método.

Vamos a explicar la sintaxis del método:

● nombre: para poder referenciar desde otro bloque de código es conveniente que el nombre sea representativo de lo que hace y puede seguir la convención de CamelCase.
● parámetros de entrada (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. Se debe especificar para cada parámetro de entrada su tipo y su nombre (con el cual lo podemos utilizar dentro del bloque del método). Un método puede no recibir ningún parámetro de entrada y los paréntesis quedarían vacíos.

● tipoRetorno: indica el tipo del valor que retorna el método (es obligatorio especificarlo). Si el método no devuelve ningún valor este tipo será **void**. El tipo de retorno puede ser primitivo (int, double, boolean, etc.) o complejo (String, array, listas, clases, etc.) estos los veremos más adelante.

● especificadores: determinan el tipo de acceso al método.

● return: se utiliza para retornar un valor. El tipo de dato del valor retornado debe ser igual a tipoRetorno. Luego de return viene el valor a devolver, esto puede ser: un valor especificado, una variable o una expresión que al evaluarla sea de tipoRetorno. La instrucción return puede aparecer en cualquier lugar dentro del método, no tiene que estar necesariamente al final, pero al ejecutarse esta instrucción inmediatamente se sale de la ejecución del método.

● Bloque {}: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.

IMPORTANTE:

Por ahora nos vamos a manejar siempre con especificadores public static.

Un método public puede ser accedido desde cualquier bloque de código.

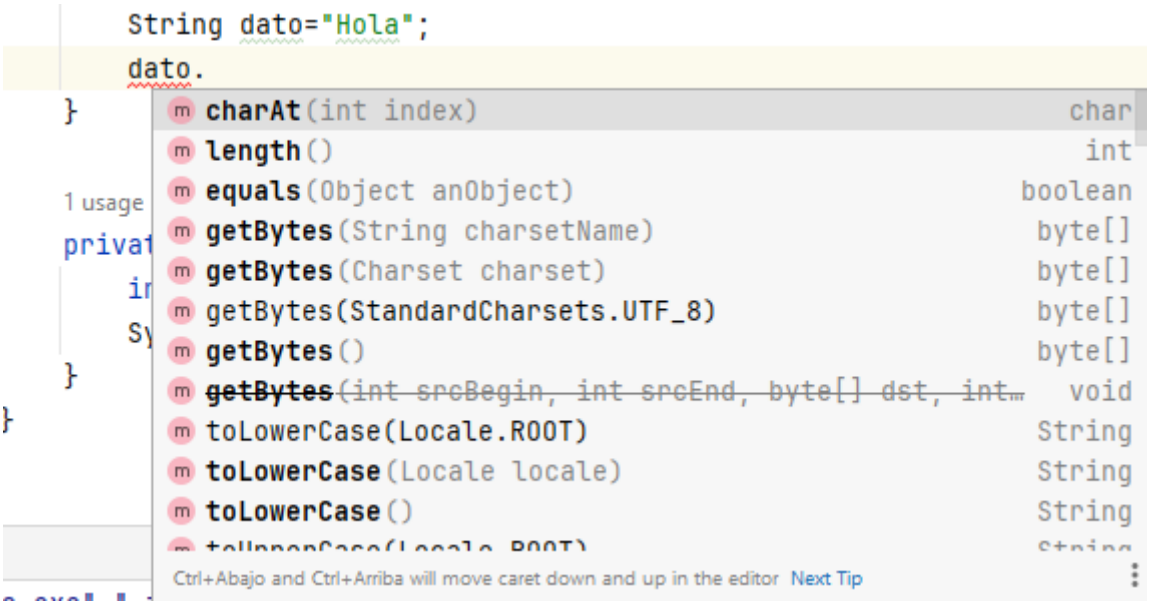
Un método static pertenece a una clase (y no a su objeto o instancia), y un método static sólo puede hacer llamadas a otro método static. Por esto último es que si se utiliza un método **dentro del bloque main**, este método **debe ser static**, sino, no compila el programa.

Igualmente luego veremos en mayor detalle qué significa cada especificación.

Cuando un método tiene un retorno se dice que es una función. Cuando no tiene retorno se dice que es un método de tipo procedimiento.

También veremos que muchas clases ya definidas en Java tienen sus propios métodos.

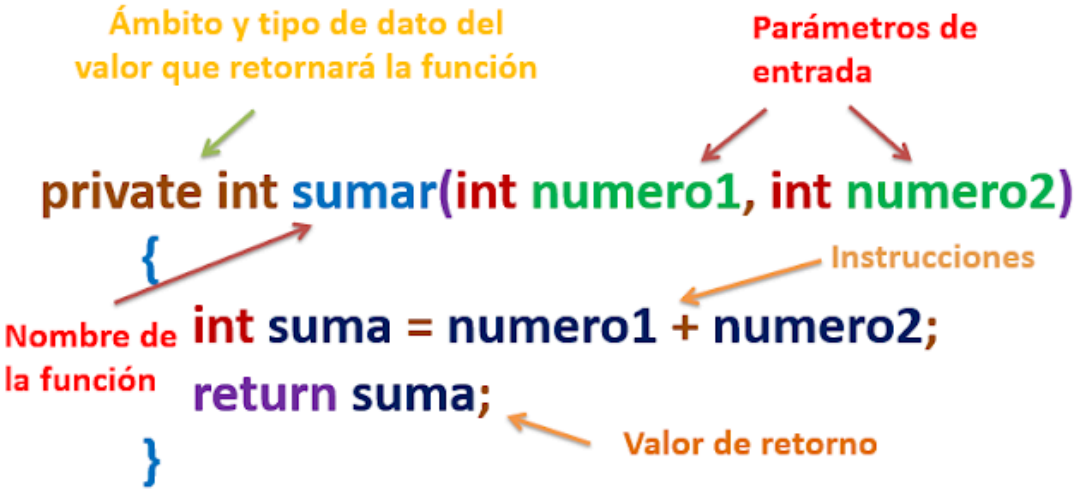
Por ejemplo la clase String, tiene sus métodos propios como vemos en la siguiente imagen:



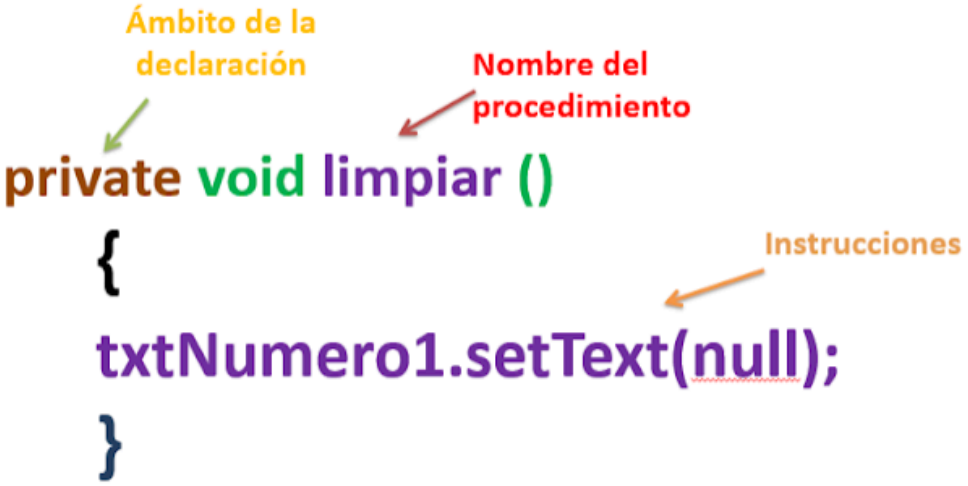
Intellij nos ayuda a ver los métodos de la clase String, ven que en un círculo rosado están los métodos que tiene la clase String, y dentro de paréntesis de cada uno los parámetros necesarios.

Para que se imaginen lo que hace una función, algo que se repite en diferentes partes del código, y lo podemos comparar con lo que hace un calefón. El calefón recibe agua fría (parámetros o argumentos), hace un proceso interno para calentar el agua y nos devuelve agua caliente.

Ahora veremos algunos ejemplos de Funciones y Procedimientos.



En este caso tenemos una función, ya que esta retorna un valor entero, lo vemos en la firma de la misma (int) y para que funcione correctamente en el return devuelve suma que es una variable entera (int) correspondiente a la suma de los 2 números pasados como parámetros.



Aca tenemos un procedimiento, ya que tiene void en la firma del método lo que hace que no devuelva nada, solo poner un dato en null (que veremos más adelante)

Ejemplo 1

Supongamos que tenemos el siguiente código:

```
package utec;

public class Principal {
    public static void main(String[] args) {
        int a=5,b=6;
        int suma=a+b;
        System.out.println("la suma es: "+suma);
    }
}
```

Es simplemente un programa que suma dos variables dadas e imprime el resultado de las mismas. Pero se podría tener el código que suma dos números encapsulados aparte en un método.

```
1   package utec;
2
3   public class Principal {
4       public static void main(String[] args) {
5           int a=5,b=6;
6           int suma=sumar(a,b);
7           System.out.println("la suma es: "+suma);
8       }
9
        1 usage
10      private static int sumar(int num1, int num2) {
11          int sumatoria = num1+num2;
12          return sumatoria;
13      }
14  }
15
```

Si nos fijamos, ahora no todo el código de nuestro programa está contenido dentro del bloque que tiene el método “main”, sino que está fuera del bloque debajo.

Este bloque corresponde al método de nombre sumar, que es una función, como sabemos esto, mirando la firma del método que está en la línea 10 la tercera palabra en este caso dice int que es lo que debe devolver nuestra función.

Recibe dos parámetros que van a llamarse num1 y num2 del tipo int, esta función se llama desde la línea 6, con el nombre de la función, y los dos parámetros o argumentos.

sumar(a,b) ---> sumar(num1,num2), donde a se mete en num1, y b en num2, deben respetar la posición de llamada con la que se recibe.

Como el retorno es int, el tipo de retorno es int.

Como son bloques diferentes aplica el tema del scope de las variables para cada uno de ellos.

El resultado de ejecutar este programa es :

```
"C:\Program Files\J
la suma es: 11
```

Ejemplo 2:

```
1   package utec;
2
3   public class Principal2 {
4       public static void main(String[] args) {
5           String a="Marcela";
6           String b="Perez";
7           String concatenado = concatenar(a,b);
8           System.out.println(concatenado);
9       }
        1 usage
10  @   private static String concatenar(String nombre, String apellido) {
11          return "Hola "+nombre+" "+apellido;
12      }
13  }
14
```

En este caso tenemos una función que se llama concatenar que lo que hace es devolver un String concatenado, con dos variables String y que se arma un texto de saludo que retorna para luego ser mostrado en la consola.

Desde donde se llama, dentro del main desde la línea 7 del código, se le pasan dos parámetros, primero a y luego b, en ese orden se reciben y se guarda la a en el primer parámetro que es nombre, y la variable b con el apellido.

Este sería el resultado :

```
"C:\Program Files\Jav
Hola Marcela Perez
```

¿Qué pasa si pasamos los parámetros al revés ?

```
1   package utec;
2
3   public class Principal2 {
4       public static void main(String[] args) {
5           String a="Marcela";
6           String b="Perez";
7           String concatenado = concatenar(b,a);
8           System.out.println(concatenado);
9       }
        1 usage
10  @   private static String concatenar(String nombre, String apellido) {
11          return "Hola "+nombre+" "+apellido;
12      }
13  }
14
```

El resultado será el siguiente:

```
"C:\Program Files\Java\jdk-17.0
Hola Perez Marcela
```

Como los parámetros son posicionales la variable b (Peres) va para el nombre y la variable a (Marcela) va para el apellido.

Se podría llamar más de una vez esta función veremos que pasa si al código anterior le agregamos lo siguiente:

```
6        String b="Perez";
7        String concatenado = concatenar(a,b);
8        System.out.println(concatenado);
9        String res1=concatenar( nombre: "Raul", apellido: "Ortega");
10       String res2=concatenar( nombre: "", apellido: "Algo");
11       String res3=concatenar( nombre: 3, apellido: "Garcia");
```

Vemos que en la línea 11 nos marca un error, y nuestro programa no compila, es debido que los parámetros son del tipo String, y le estoy pasando un int.

Para corregirlo esto lo puedo corregir colocando el número entre comillas dobles y lo soluciono.

```
8        System.out.println(concatenado);
9        String res1=concatenar( nombre: "Raul", apellido: "Ortega");
10       String res2=concatenar( nombre: "", apellido: "Algo");
11       String res3=concatenar( nombre: "3", apellido: "Garcia");
12   }
```

Ejemplo 3:

```java
package utec;

public class Main2 {
    public static void main(String[] args) {
        String mensaje="Hola como estas?";
        imprime(mensaje);
    }

    1 usage
    private static void imprime(String mensaje) { // Firma del método
        System.out.println(mensaje); // Esto es un procedimiento
    }                                // por tener void (no devuelve nada)
}                                    // no tiene return
```

Esto es un procedimiento, ya que en la firma del método tiene un void (no devuelve nada) por lo que no tiene return.

El resultado de ejecutar este programa usando el procedimiento es:

```
"C:\Program Files\Ja
Hola como estas?
```

IMPORTANTE:

Cabe destacar que las funciones y procedimientos se utilizan generalmente cuando en nuestro programa tenemos código repetido. ¿Por qué?

Porque de esa forma encapsulamos el código repetido dentro de una función y desde el programa principal lo único que debo hacer es llamarla la cantidad de veces que sea necesario, sin tener que repetir explícitamente dicho código.

UTEC
Universidad Tecnológica

La Clase Math en Java

En la clase Math de Java hay muchas funciones matemáticas disponibles para su uso.

Para consultar la lista completa de funciones nos podemos fijar en la API de Java, según la versión que estemos usando.

Para la versión 17, podemos consultar en el siguiente link :

https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/lang/Math.html

Función matemática	Significado	Ejemplo	Resultado
round	redondeo	double x = Math.round(4.5)	x=5
ceil	Redondeo al entero mayor	double x = Math.ceil(5.4)	x=6
floor	Redondeo al entero menor	double x = Math.floor(5.8)	x=5
random	Número aleatorio	double x= Math.random()	x=0.234324765
pow	Potencia	x=Math.pow(2,3);	x=8
abs	Valor Absoluto	Int x=(int)Math.abs(6.3);	x=6
cos	Coseno	double x=Math.cos(0.8)	x=0.6967
sqrt	Raíz cuadrada	double x=Math.sqrt(144)	x=12

La función random() es una función random que me genera cada vez que la invoco un número aleatorio entre 0 y 1.

UTEC
Universidad Tecnológica

```java
package utec;

public class ClaseMath {
    public static void main(String[] args) {
        // Ejemplos de la clase Math de Java, viene incorporadas en la librerias
        // Standard de Java, al igual que la clase String

        // Redondeo hacia arriba
        double x= Math.ceil(5.4);
        System.out.println(x);

        // Redondeo hacia abajo
        x = Math.floor(5.8);
        System.out.println(x);

        // Tenemos que hacer valor absoluto hay que castear el resultado ya que la funcion
        // devuelve un double
        int y=(int)Math.abs(6.8);
        System.out.println(y);
        // Potencia de 2 elevado a la 3
        x = Math.pow(2,3);
        System.out.println(x);


        for (int i=0;i<10;i++)  // random -- aleatorio entre 0 y 1
            System.out.println(Math.random());
    }
}
```

La salida del programa seria:

UTEC
Universidad Tecnológica

ClaseMath

```
"C:\Program Files\Java\jdk-17.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.3
6.0
5.0
6
8.0
0.287718543123069
0.7796155230313869
0.6394049316251542
0.9151543928578357
0.787987208901223
0.39033500050882264
0.30829411856246236
0.13175415972449855
0.787473176270776
0.548125262867353

Process finished with exit code 0
```