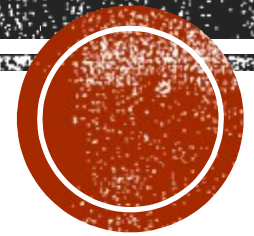
LECTURA DE DATOS E JAVA



SCANNER

- La lectura de datos en Java es realizada mediante esta clase.
- Para ello, tendremos que realizar los siguientes pasos:
  1) Escribir la directiva import
     - Al inicio de la clase añadimos:

       ```
       import java.util.Scanner
       ```
  2) Creamos un objeto Scanner y le transferimos “el control del teclado” mediante System.in

     ```
     Scanner sc = new Scanner(System.in);
     ```
  3) Al finalizar el uso del objeto, debemos cerrar “el control del teclado”, para ello añadiremos:

     ```
     sc.close();
     ```

Métodos, funciones y procedimientos en Java.

INDICE

- Concepto.
- Funciones, métodos o procedimientos.
- Ejemplo
- Cohesión y acoplamiento. Niveles.
- Ejercicios

CONCEPTO

- Los métodos son una herramienta indispensable para programar.
- Ya que nos permiten automatizar tareas que requiramos con frecuencia, uso de parámetros y hacer uso de otras librerías (funciones matemáticas, aritméticas, de archivos, de fechas, etc).
- Aprender a crear métodos y usarlos correctamente es de gran importancia, separar nuestro código en módulos (según su utilidad o tarea que realiza, ejemplo: abrir, cerrar, cargar, ...).
- Lo más adecuado es tener un método asociado a una tarea o funcionalidad básica, es decir, descomponer un problema en subproblemas y solucionarlos en pequeñas funciones.

FUNCIONES, MÉTODOS O PROCEDIMIENTOS.

- **Funciones**:

Las funciones son un conjunto de líneas de código (instrucciones), encapsulados en un bloque, *usualmente reciben parámetros*, cuyos valores utilizan para efectuar operaciones y adicionalmente *devuelven/retornan* un valor. En otras palabras, una función puede recibir parámetros o argumentos (algunas no reciben nada), hace uso de dichos valores recibidos como sea necesario y devuelve un valor usando la instrucción *return*. Si no devuelve algo, entonces le llamamos **procedimiento**.

FUNCIONES, MÉTODOS O PROCEDIMIENTOS.

- **Métodos**:

Los métodos realizan las mismas tareas que las funciones, es decir, son funcionalmente idénticos. Su diferencia radica en la manera en que hacemos uso de uno u otro, es decir, un método también puede recibir valores, efectuar operaciones con estos y retornar valores, sin embargo un método está **siempre** asociado a un **objeto**.

¡ojO! En Java se debe hablar de métodos y no de funciones, pues en Java estamos siempre obligados a crear un objeto para usar el método. Para que sea una función/procedimiento, esta debe ser **static**, para que no requiera de un objeto para ser llamada.

FUNCIONES, MÉTODOS O PROCEDIMIENTOS.

- **Procedimientos**:

Los procedimientos son un conjunto de instrucciones que se ejecutan sin retornar ningún valor, pueden recibir o no argumentos. En Java, un procedimiento es un método cuyo tipo de retorno es void, que no nos obliga a utilizar una sentencia return.

EJEMPLO

[**acceso**] [**modificador**] **tipo** nombreFuncion([tipo nombreArgumento,[tipo nombreArgumento],...)

```
{
        /*
                Bloque de instrucciones
        */

        return valor;
}
```

1) **Modificador de acceso** => Puede ser public, private o protected. En caso de no indicarlo, se asume el modificador de acceso por defecto.
2) **Modificador** => Puede ser final o static (o ambas), también es opcional.
3) Un método o función siempre **retorna** algo, por lo tanto es obligatorio declararle un **tipo** (int, boolean,...). Si no es función, pondremos **void**.
4) **Nombre** de dicha función, para poder identificarla y llamarla (invocarla) durante la ejecución.
5) En el interior del paréntesis, indicaremos el conjunto de parámetros.

Hasta este punto es lo que denominaremos **cabecera o prototipo** del método.

A continuación, entre llaves, definiremos el **cuerpo** del método, conjunto de sentencias a ejecuta.

COHESIÓN Y ACOPLAMIENTO

- **Cohesión**:
  1) Se puede definir como la relación entre los componentes de un ente/conjunto/unidad.
  2) Se refiere a que los módulos tienen una sola responsabilidad/función

- **Acoplamiento**:
  1) Es la manera en la que se relacionan los componentes entre ellos.
  2) Relación y grado de dependencia de los módulos entre ellos.

COHESIÓN NIVELES : FUERTE

- **Funcional**
  - Un módulo realiza una única acción.
- **Secuencial**
  - Las acciones dentro de un módulo han de realizarse en un orden concreto, debido a los datos que requiere.
- **De comunicación**
  - El módulo contiene un conjunto de operaciones que son realizadas sobre los mismos datos.
- **Temporal**
  - Las operaciones dentro del módulo tienen que realizarse al mismo tiempo, por ejemplo, inicialización.

COHESIÓN NIVELES : DÉBIL

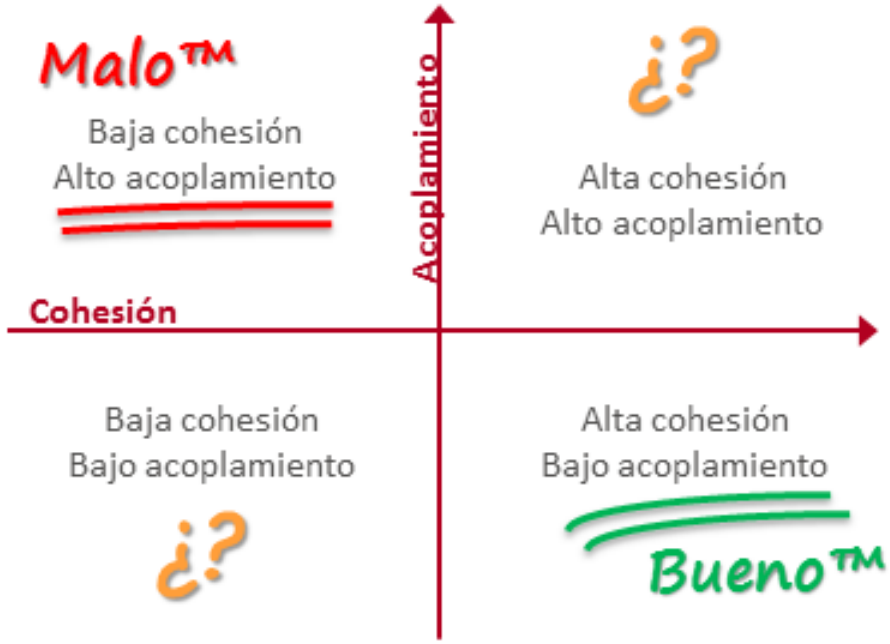
- **Procedural**
  - Las operaciones del módulo se realizan en un orden concreto aunque sean independientes.
- **Lógica**
  - El módulo depende un parámetro para la ejecución de las operaciones, es decir, el nexo de unión entre es las operaciones del módulo es el flujo de ejecución.
- **Coincidental**
  - No existe relación observable entre las operaciones realizadas dentro de un módulo.

ACOPLAMIENTO NIVELES : DE MEJOR A PEOR

- **De datos (normal)**
  - Todo lo que se comparte entre módulos es especificado en una lista de parámetros por el módulo invocado.
- **De control**
  - Un módulo le pasa datos a otro para indicarle que hacer, el primer módulo conoce el funcionamiento interno del segundo.
- **Externo**
  - Cuando dos o más módulos utilizan los mismo datos globales.
  - Generalmente, este tipo de acoplamiento, no es recomendable.
- **Patológico o de contenido**
  - Cuando un módulo utiliza el código de otro o altera sus datos locales.

COHESIÓN Y ACOPLAMIENTO

- Objetivo:



- Beneficios:

Al obtener una alta cohesión y bajo acoplamiento, obtenemos, los siguientes beneficios:

1. Aumenta la **legibilidad** del código.
2. **Mantenibilidad**, los cambios a realizar serán mínimos (módulo específico).
3. **Reusabilidad**, si los módulos están bien definidos, podrán ser reutilizados en otros “programas”.
4. La realización de **pruebas** estará bien acotado y serán mínimas por módulo utilizado.

JAVA.UTILS

INDICE

- Strings
- Math
- Random

STRING

Definición

Crea una cadena vacía:

String nombreCadena = new String ();

Crea una cadena con el valor indicado:

String nombreCadena = new String ("valor");

String nombreCadena = "valor";

Llamada a métodos:

Variable = **nombreCadena.nombreMétodo (parámetros);**

//el punto detrás del objeto, nos permite acceder a los métodos de una clase

La Variable será del mismo tipo que el tipo de retorno del método.

STRING

Método	Descripción
char charAt(int index)	Devuelve el carácter en la posición indicada por parámetro
int length()	Devuelve la longitud de la cadena
static String format(String format, Object... args)	Devuelve la cadena formateada.
String substring(int beginIndex)	Devuelve una subcadena a partir del índice indicado.
String substring(int beginIndex, int endIndex)	Devuelve una subcadena a partir del índice indicado y hasta la posición dada.
boolean contains(CharSequence s)	Devuelve verdadero o falso si la cadena dada por parámetro está contenida en la principal.
static String join(CharSequence delimiter, CharSequence... elements)	Devuelve la unión de las cadenas indicadas.
boolean isEmpty()	Indica si la cadena está vacía.
String concat(String str)	Concatena la cadena dada por parámetro.
String replace(char old, char new)	Reemplaza todas ocurrencias del character indicado por le Nuevo valor.
String replace(CharSequence old, CharSequence new)	Reemplaza la secuencia de caracteres indicada por la secuenia de caracteres dados.

STRING

Método	Descripción
static String equalsIgnoreCase(String another)	Compara una cadena con otra, sin tener en cuenta el case sensitive (distinción entre mayúsculas y minúsculas).
String[] split(String regex)	Divide la cadena que encaja con el parámetro dado.
int indexOf(int ch)	Devuelve la posición del caracter indicado por parámetros.
int indexOf(int ch, int fromIndex)	Devuelve la posición de un character desde la posición dada.
String toLowerCase()	Devuelve la cadena en minúscula.
String toUpperCase()	Devuelve la cadena en mayúscula.
String trim()	Elimina los espacios iniciales y finales de la cadena.

MATH

Import:

En caso de *requerir importación*: import java.lang.Math;

Utilidad:

Esta clase contiene métodos para realizar operaciones numéricas básicas como: exponentes, logarítmicas, raíces cuadradas y funciones trigonométricas.

MATH

Método	Descripción	Parámetros	Tipo de dato devuelto
abs	Devuelve el valor absoluto de un numero.	Un parametro que puede ser un int, double, float o long	El mismo que introduces.
arcos	Devuelve el arco coseno de un angulo en radianes.	Double	Double
asin	Devuelve el arco seno de un ángulo en radianes.	Double	Double
atan	Devuelve el arco tangente entre -PI/2 y PI/2.	Double	Double
atan2	Devuelve el arco tangente entre -PI y PI.	Double	Double
ceil	Devuelve el entero más cercano por arriba.	Double	Double
floor	Devuelve el entero más cercano por debajo.	Double	Double
round	Devuelve el entero más cercano.	Double o float	long (si introduces un double) o int (si introduces un float)
cos	Devuelve el coseno de un ángulo.	Double	Double
sin	Devuelve el seno de un ángulo.	Double	Double
tan	Devuelve la tangente de un ángulo.	Double	Double
exp	Devuelve el exponencial de un número.	Double	Double
log	Devuelve el logaritmo natural en base e de un número.	Double	Double
max	Devuelve el mayor de dos entre dos valores.	Dos parametros que pueden ser dos int, double, float o long	El mismo tipo que introduces.
min	Devuelve el menor de dos entre dos valores.	Dos parametros que pueden ser dos int, double, float o long	El mismo tipo que introduces.
random	Devuelve un número aleatorio entre 0 y 1. Se pueden cambiar el rango de generación.	Ninguno	Double
sqlrt	Devuelve la raíz cuadrada de un número.	Double	Double
pow	Devuelve un número elevado a un exponente.	Dos parámetros double (base y exponente)	Double

Random

Import:

Tenemos que importa la clase: java.util.Random

Utilidad:

Nos permite generar números de forma aleatoria

Método	Descripción
nextBoolean()	Generará de forma distribuida valores booleanos.
nextDouble()	Generará un doble uniformemente distribuido entre 0.0 y 1.0.
nextFloat()	Generará un Returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence.
nextGaussian()	Returns the next pseudorandom, Gaussian ("normally") distributed double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.
nextInt()	Genera un número entero de forma aleatoria Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence.
nextInt(int n)	Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.
nextLong()	Returns the next pseudorandom, uniformly distributed long value from this random number generator's sequence.
setSeed(long seed)	Sets the seed of this random number generator using a single long seed.

VARIABLES, TIPOS DE DATOS CONVERSIONES

INDICE

- Variables
- Tipos de datos
- Conversiones

VARIABLES

Una variable es una zona en la memoria del computador con un valor que puede ser almacenado para ser usado más tarde en el programa.

La variables vienen dadas por:

- **Un nombre (identificador)**, que permite al programa acceder al valor que contiene en memoria. Debe ser un identificador válido, es decir, **el primer carácter de la secuencia debe ser una letra, un símbolo de subrayado (_) o el símbolo dólar ($)**.
- **Un tipo de dato**, que especifica qué clase de información guarda la variable en esa zona de memoria.
- **Un rango de valores** que puede admitir dicha variable.

TIPOS DE VARIABLES

En un programa, podemos encontrar distintos tipos de variables. Las diferencias entre una variable y otra dependerán de varios factores, por ejemplo, el tipo de datos que representan, si su valor cambia o no a lo largo de todo el programa, o cuál es el papel que llevan a cabo en el programa.

Según el párrafo anterior, en Java, se definen los siguientes tipos de variables:

a) **Variables de tipos primitivos y variables referencia**, según el tipo de información que contengan. En función de a qué grupo pertenezca la variable, tipos primitivos o tipos referenciados, podrá tomar unos valores u otros, y se podrán definir sobre ella unas operaciones u otras.

b) **Variables y constantes**, dependiendo de si su valor cambia o no durante la ejecución del programa. La definición de cada tipo sería:

- *Variables*: Sirven para almacenar los datos durante la ejecución del programa, pueden estar formadas por cualquier tipo de dato primitivo o referencia. Su valor puede cambiar varias veces a lo largo de todo el programa.
- *Constantes o variables finales*: Son aquellas variables cuyo valor no cambia a lo largo de todo el programa.

c) **Variables miembro y variables locales**, en función del lugar donde aparezcan en el programa. La definición concreta sería:

- **Variables miembro:** Son las variables que se crean dentro de una clase, fuera de los método de esta. Pueden ser de tipos primitivos o referencias, variables o constantes.
- **Variables locales:** Son las variables que se crean y usan ***dentro de un método o, en general, dentro de cualquier bloque de código***. La variable deja de existir cuando la ejecución del bloque de código o el método finaliza. Al igual que las variables miembro, las variables locales también pueden ser de tipos primitivos o referencias.

TIPOS DE DATOS

- En los lenguajes fuertemente tipados, a todo dato (constante, variable o expresión) le corresponde un tipo que es conocido antes de que se ejecute el programa.
- El tipo de dato, limita el valor de la variable o expresión, las operaciones que se pueden hacer sobre esos valores, y el significado de esas operaciones.

Los tipos de datos en Java se dividen principalmente en dos categorías:

- **Tipos de datos sencillos o primitivos**: Representan valores simples que vienen predefinidos en el lenguaje; contienen valores únicos, como por ejemplo un carácter o un número.
- **Tipos de datos referencia**: Se definen con un nombre o referencia (puntero) que contiene la dirección en memoria de un valor o grupo de valores. Dentro de este tipo tenemos por ejemplo los vectores o arrays, que son una serie de elementos del mismo tipo, o las clases, que son los modelos o plantillas a partir de los cuales se crean los objetos.

TIPOS DE DATOS PRIMITIVOS

Los tipos primitivos son aquéllos datos *sencillos* que constituyen los tipos de información más habituales: *números, caracteres y valores lógicos o booleanos*. Al contrario que en otros lenguajes de programación orientados a objetos, en Java, **no son objetos**.

TIPOS DE DATOS PRIMITIVOS				
Tipo	**Descripción**	**Bytes**	**Rango**	**Valor por default**
byte	Entero muy corto	1	-128 a 127	0
short	Entero corto	2	-32,768 a 32,767	0
int	Entero	4	-2,147,483,648 a 2,147,483,647	0
long	Entero largo	8	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807	0L
float	Numero con punto flotante de precisión individual con hasta 7 dígitos significativos	4	+/-1.4E-45 (+/-1.4 times 10^{-45}) a +/-3.4E38 (+/-3.4 times 10^{38})	0.0f
double	Numero con punto flotante de precisión doble con hasta 16 dígitos significativos	8	+/-4.9E-324 (+/-4.9 times 10^{-324}) a +/-1.7E308 (+/-1.7 times 10^{308})	0.0d
char	Carácter Unicode 2	`\u0000` a `\uFFFF`	'\u0000'	
boolean	Valor Verdadero o Falso	1	true o false	false

Restricciones de identificadores

Identificador	Convención	Ejemplo
nombre de variable	Comienza por letra minúscula, y si tienen más de una palabra se colocan juntas y el resto comenzando por mayúsculas	numAlumnos, suma
nombre de constante	En letras mayúsculas, separando las palabras con el guión bajo, por convenio el guión bajo no se utiliza en ningún otro sitio.	TAM_MAX, PI
nombre de una clase	Comienza por letra mayúscula	String, MiTipo
nombre de función	Comienza con letra minúscula	modifica_Valor, obtiene_Valor

VARIABLES

- Un **identificador** es una secuencia ilimitada de caracteres **Unicode**.
- **Unicode** es un código de caracteres o sistema de codificación, un alfabeto que recoge los caracteres de prácticamente todos los idiomas importantes del mundo. Las líneas de código en los programas se escriben usando ese código de caracteres, esto nos permite que aplicar nuestro idioma local para facilitar el trabajo al programador. El estándar Unicode originalmente utilizaba 16 bits, pudiendo representar hasta 65.536 caracteres distintos, dos elevado a la potencia dieciséis. Actualmente Unicode puede utilizar más o menos bits, dependiendo del formato que se utilice: UTF-8, UTF-16 o UTF32.
- A cada carácter le corresponde **unívocamente** un número entero perteneciente al intervalo de 0 a 2 elevado a n, siendo n el número de bits utilizados para representar los caracteres. Por ejemplo, la letra ñ es el entero164.
- Unicode es “compatible” con el código ASCII, por lo que la conversión entre ellos es inmediata.

CONVERSIONES DE TIPO

¿Qué pasará si dividimos un número entre otro, tendrá decimales el resultado de esa división?

- Siempre que el denominador no sea divisible entre el divisor, tendremos un resultado con decimales, pero no es así.
- Si el denominador y el divisor son variables de tipo entero, el resultado será entero y no tendrá decimales. Para que el resultado tenga decimales necesitaremos hacer una **conversión de tipo**.

Las conversiones de tipo se realizan para hacer que el resultado de una expresión sea del tipo que nosotros deseamos y/o esperamos. Existen dos tipos de conversiones:

- **Conversiones automáticas**: Cuando a una variable de un tipo se le asigna un valor de otro tipo numérico *con menos bits para su representación,* se realiza una conversión automática. En ese caso, el valor se dice que es promocionado al tipo más grande, para poder hacer la asignación. También se realizan conversiones automáticas en las operaciones aritméticas, cuando estamos utilizando valores de distinto tipo, el valor más pequeño se promociona al valor más grande, ya que *el tipo mayor siempre podrá representar cualquier valor del tipo menor* (por ejemplo, de int a long o de float a double).

CONVERSIONES DE TIPO

- **Conversiones explícitas**: Cuando hacemos una conversión de un tipo con más bits a un tipo con menos bits. En estos casos debemos indicar que queremos hacer la conversión de manera expresa, ya que *se puede producir una pérdida de datos y hemos de ser conscientes de ello*. Este tipo de conversiones se realiza con el **operador cast**. El operador cast es un operador unario que se forma colocando delante del valor a convertir el tipo de dato entre paréntesis. Tiene la misma precedencia que el resto de operadores unarios y se asocia de izquierda a derecha. Debemos tener en cuenta que *un valor numérico nunca puede ser asignado a una variable de un tipo menor en rango*, si no es con una conversión explícita.

```
int a ;
byte b;
a = 12 ;
b = 12;
```

Conversiones de una cadena

Java nos facilita varios métodos para convertir una cadena (String) en alguno de los tipos de datos primitivos:

- **De String a Float**: La función parseFloat() analiza una cadena y devuelve un número de punto flotante. Ejemlo:

  Float.parseFloat(String str)

- **De String a Int**: La función parseInt() analiza una cadena y devuelve un entero. Ejemplo:

  Integer.parseInteger(String str)

  Integer.valueOf(String str);

- **De String a Double**: La función parseDouble() analiza una cadena y devuelve un número decimal doble.

  Double.parseDouble(String str)

Conversiones de una cadena

- **De String a Boolean**: La función parseBoolean() analiza una cadena y devuelve un booleano. Ejemplo:

Boolean.parseBoolean(String str)

Boolean.valueOf(String str);

- **De String a char**: La función charAt() analiza una cadena y devuelve un carácter. Ejemplo:

cadena.charAt(posicion)

Todos estos tipos de datos poseen un proceso similar de conversión a String:

String.valueOf(valorTipoDato);