# **GUÍA 1: INTRODUCCIÓN A JAVA**

Java es un lenguaje de programación de alto nivel, estos nos permiten escribir código mediante idiomas que conocemos y luego, para se ejecutados, se traduce a lenguaje de máquina mediante traductores o compiladores. Jva es un lenguaje de alto nivel donde sus palabras reservadas están en **INGLÉS.**

# ESTRUCTURA DE UN PROGRAMA JAVA

Primero debemos ir a **File🡺New Proyect**.

Se abre una ventana emergente donde seleccionamos en **categories:🡺Java** y en **Proyects:🡺Java Application**.

Luego le damos a **Next**.

La ventana cambia y le ponemos un nombre a nuestro proyecto en **Proyect Name**.

Verificamos que SIEMPRE esté tildado la casilla de **Create Main Class (acá aparecerá el nombre que le hayamos puesto a nuestro proyecto dos veces seguidas separados por una coma)**.

# ¿CÓMO SE VE UN PROGRAMA EN JAVA?

## ¿Cuáles son los comentarios?

Los comentarios son ignorados por el compilador y solo son útiles para el programador. Los comentarios ayudan a explicar aspectos relevantes de un programa y lo hacen más legible.

El delimitador de inicio de un comentario es **/\*** y el delimitador de fin de comentarios es **/\***.

Si el comentario es breve podemos usar **//** igual que en PSInt.

## ¿Cuál es el paquete?

Los paquetes son contenedores de clases y su función es la de organizar la distribución de las clases. A cada paquete que creamos es una carpeta que contiene archivos que son las clases.

**Paquete = Una carpeta en el Navegador de Archivos de Windows.**

**Clase = Los archivos que se van a crear dentro de la carpeta (paquete).**

En Java se le provee la sentencia **package** como la forma de agrupar clases relacionadas.

**La sentencia package tiene la siguiente forma:**

**package** <nombre\_paq\_sup>[.<nombre\_sub\_paq>]\*;

La declaración package debe estar al principio del archivo fuente y se pueden declarar sólo de a UN paquete. Los de los paquetes los pondrá el programador al crear el programa y son jerárquicos (al igual que su organización de directorio en disco) además, están separados por puntos.

## ¿Cuál es la clase?

En el mundo de orientación a objetos, todos los programas se definen en término de objetos y sus relaciones. Las clases sirven para modelar los objetos que serán utilizados por nuestros programas.

Una clase está formada por una parte que corresponde a la declaración de la clase, y otra que corresponde al cuerpo de la misma.

Declaracion de la clase {

Cuerpo de la clase

}

La definición, o cuerpo, de una clase comienza con una llave abierta (**{**) y termina con una llave cerrada (**}**). El nombre de la clase lo define el programador.

## ¿Cuál es el Método?

Después de la definición de una clase se escribe ella definición del método **main()**. Un método es una secuencia de sentencias ejecutables. La sentencia de un método queda delimitada por llaves, para abrir uso **{**  y para cerrar uso **}**. Los métodos son de vital importancia para los objetos y las clases.

### Método main()

El main() sirve para que un programa se pueda ejecutar, este método, vendría a reemplazar al Algoritmo / FinAlgoritmo de PSInt, también inicia con una llave abierta **{** y termina con una llave cerrada **}** y tiene la siguiente declaración:

public static void main () (String[ ] args)[ {

}

* **public**: es un tipo de acceso que indica que el método main() es público y, que por lo tanto, puede ser llamado desde otras clases. Todo método main() debe ser público para poder ejecutarse desde el intérprete de Java (JVM).
* **static**: es un modificador que indica que la clase no necesita ser instanciada para poder utilizar el método. También indica que el método es el mismo para todas las instancias que pudieran crearse.
* **void**: indica que la función o método main() no devuelve ningún valor.

El método main() debe aceptar siempre, como parámetro, un vector de strings, que contendrá los posibles argumentos que se le pasen al programa en la línea de comandos, aunque como es nuestro caso, no se utilice.

# SENTENCIAS

Son las unidades ejecutables más pequeñas de un programa, en otras palabras, una línea de código escrita es una sentencia. Especifican y controlan el flujo de ejecución de un programa. Una sentencia consta de palabras clave o reservadas como expresiones, declaraciones de variables, o llamadas a funciones.

Los textos siempre se escriben entre comillas dobles para diferenciarlos de otros elementos del lenguaje. **Todas las sentencias de un programa de Java deben terminar con el símbolo punto y coma (;).** Este símbolo indica al compilador que ha finalizado una sentencia.

# ¿CUÁLES SON LOS ELEMNTOS DE UN PROGRAMA?

Los elementos de un programa son los componentes que vana a conformar las sentencias que podamos escribir en nuestro programa. Estos elementos siempre estarán dentro de un programa/algoritmo**, es decir, dentro del método main():**

public static void main () (String[ ] args)[ {

ACÁ VAN LAS SENTENCIAS Y LOS ELEMENTOS DEL PROGRAMA

}

Los elementos de un programa son: **identificadores, variables, constantes, operadores, palabras reservadas.**

# ¿QUÉ SON LAS PALABRAS RESERVADAS?

Palabras que dentro del lenguaje significan la ejecución de una instrucción determinada. En Java todas las palabras están en INGLÉS.

# ¿QUÉ SON LOS IDENTIFICADORES?

Los identificadores son los nombre que se usan para identificar cada uno de los elementos del lenguaje (son los que les ponemos nosotros a las variables, subprogramas, funciones, vectores, matrices, etc.).

Es aconsejable poner nombres que tengan sentido y fáciles de interpretar

Para saber si un nombre es bueno me debo preguntar:

* ¿Refleja este nombre la información que aloja o la función que cumple?

# VARIABLES Y CONSTANTES

Las **CONSTANTES** son las que tienen información que **NO** va a cambiar a lo largo de la ejecución del programa.

Las **VARIABLES** son las que tienen información que **SI** va a cambiar a lo largo de la ejecución del programa.

## Declaración de variables en Java

Normalmente las definiciones de las variables y constantes con nombre deben ser declaradas antes de ser utilizadas en el programa.

<tipo\_de\_dato> <nombre\_de\_la\_variable>

# TIPOS DE DATOS EN JAVA

Java es un lenguaje del tipo estático por lo que las variables deben ser definidas antes de ser utilizadas y no podemos cambiar el tipo de variable (string, int, boolean, etc.), a menos que se haga a través de una conversión.

## Tipos de datos primitivos

1. **byte 🡺** Número que va desde el -128 al 127 (inclusive).
2. **short 🡺** Número que va desde el -32.768 al 32.767 (inclusive).
3. **int 🡺** Número que va desde el -2.147.483.648 al 2.147.483.647 (inclusive) (ESTE USAMOS PARA LOS ESNTEROS)
4. **float 🡺** Numero decimal de precisión simple.
5. **double 🡺**  Número decimal de precisión doble (ESTE USAMOS COMO REAL)
6. **boolean 🡺** Soporta dos tipos de valores: Verdadero o Falso (ESTE USAMOS COMO LÓGICA)
7. **char 🡺** Datos del tipo carácter
8. **String 🡺** Deben ir entre comillas y una vez definidas son imutables (ESTE USAMOS PARA LAS CADENAS)

# INSTRUCIONES PRIMITIVAS

Las instrucciones primitivas van a ser las de **asignación, lectura y escritura**.

## Asignación

La instrucción de asignación permite almacenar un valor en una variable previamente definida:

*<variable> = <expresión (o valor o expresión o verdadero o falso)> 🡺 Depende el tipo de variable la asignación*

En Java podemos definir una variable al mismo tiempo que le asignamos un valor

<tipo\_de\_dato (int o String o boolean)> <nombre\_variable (num1 o nombre o etc.> = <expresión>;

Al ejecutarse la asignación, primero se evalúa la expresión de la derecha y luego se le asigna el resultado a la variable de la izquierda. El tipo de variable debe coincidir con la expresión si no arroja error (si defino como *int* la debo igualar a un número, así con todo).

## Valores por defecto

* **short = 0**
* **int = 0**
* **long = 0**
* **double = 0.0**
* **boolean = false**
* **char = ´\u000´**
* **String = Null**
* **Objetos = Null**

En nuestro caso usamos todas variables locales, por lo que no se le asigna ningún valor. Para que el programa no arroje error debemos inicializarlas en todos los casos (igual que PSItn).

# OPERADORES

Los operadores son símbolos especiales de la plataforma que permiten especificar operaciones en uno, dos o tres operados y retornar un resultado.

Los operadores con mayor precedencia se evalúan siempre primero.

Orden de precedencia:

1. Operadores unitarios
2. Operadores aritméticos
3. Operadores de bits
4. Operadores relacionales
5. Operadores booleanos
6. Operadores por asignación

Cuando dos operadores tienen el mismo orden de prioridad se evalúan de izquierda a derecha.

## Operadores Aritméticos

* + 🡺 Operador de suma
* - 🡺 Operador de resta
* \* 🡺 Operador de multiplicación
* / 🡺 Operador de división
* % 🡺 Operador de módulo

## Operadores Unitarios

* + 🡺 Operador unitario de suma, indica que el valor es positivo
* - 🡺 Operador unitario de resta, indica que el valor es negativo
* ++ 🡺 Operador de incremento
* -- 🡺 Operador de decremento

## Operadores de Igualdad o Relacionales

* == 🡺 Igual
* != 🡺 Distinto
* > 🡺 Mayor que
* >= 🡺 Mayor o igual que
* < 🡺 Menor que
* <= 🡺 Menor o igual que

# TIPOS DE INSTRUCCIONES

* **Instrucción de inicio/fin:** se utiliza para delimitar bloques de código.
* **Instrucción de asignación:** se utiliza para asignar el resultado de la evaluación de una expresión a una variable.
* **Instrucción de lectura:** se utiliza para leer datos de un dispositivo de entrada y se asigna a las variables.
* **Instrucción de escritura:** se utiliza para escribir o mostrar mensajes o el contenido de las variables en un dispositivo de salida.
* **Instrucción de bifurcación:** mediante estas instrucciones el desarrollo lineal de un programa se interrumpe.

# ENTRADAS Y SALIDAS DE INFORMACIÓN

Las operaciones de entrada permiten leer determinados valores y asignarlos a las variables, mientras que las operaciones de salida permiten escribir o mostrar resultados de variables o de sus operaciones, o bien simplemente mostrar mensajes por pantalla.

## Escritura en Java

Usamos la instrucción **System.out.print()** para mostrar el mensaje en el **Output**. Todo lo que queremos mostrar como un mensaje de texto debe ir entre comillas.

Si quisiéramos mostrar mensajes y la impresión de una variable deberíamos usar el símbolo más (+) para que se muestren juntos

*System.out.println(“Hola Mundo soy” + nombre) 🡺* En este caso la variable se llama **nombre**

# CLASES DE UTILIDAD

Las **clases de utilidad** son clases que definen un conjunto de métodos/funciones que realizan operaciones, normalmente muy utilizadas o necesarias. Lo que hacen es que no tengamos que escribir dichas operaciones manualmente, por ejemplo si queremos sacar una raíz cuadrada no tendremos que hacer el cálculo, solo poner la clase <*sqrt(double a)>* y el programa nos devolverá el resultado de la variable, en este caso, double que hayamos puesto dentro de la clase de utilidad *sqrt*.

Entre las clases de utilidad de Java más utilizadas y conocidas están las siguientes: Arrays, String, Interger, Math, Date, Calendar y GregorianCalendar.

Por ahora solo vamos a ver **Math** y **String**

## Clase String (funciones para Strings/Cadenas)

* **charAt(int index):** retorna el carácter que esté en la posición indicada en ***index* (index=índice)**
* **equals(String srt):** Compara dos cadenas y retorna Verdadero si son iguales o Falso si son distintas
* **equalsIgnoreCase(String str):** Compara dos cadenas ignorando las mayúsculas y minúsculas
* **compareTo(String otraCadena):** Compara dos cadenas y las ordena alfabéticamente, **si son iguales retorna 0** si la **primera está antes que la segunda retorna entero negativo** y **entero positivo si la segunda está antes**.
* **concat(String str):** Concatena la primer cadena (String) con la segunda (str)
* **contains(CharterSequence s):** Retorna verdadero si la cadena contiene la secuencia del tipo char del parámetro
* **endsWhit(String suffix):** Retorna verdadero si la cadena es igual al objeto del parámetro
* **indexOf(String str):** Retorna el ìndeice de la primera ocurrencia de la cadena del parámetro
* **isEmpty():** Retorna verdadero si la longitud de la cadena es 0
* **length():** Retorna la longitud de la cadena
* **replace(char oldChar, charNewChar):** Retorna una nueva cadena reemplazando los caracteres del primer parámetro (oldChar) con el del segundo parámetro (newChar)
* **split(String regex**): Retorna un arreglo de cadenas separadas por la cadena del parámetro
* **startsWith(String prefix):** Retorna verdadero si la cadena comienza igual al prefijo del parámetro (prefix)
* **substring(int beginIndex):** Retorna una cadena desde el carácter indicado (begininIndex).
* **Substring(int begininIndex, int endIndex):** Retorna una cadena que va desde el primer carácter indicado (bigininIndex) hasta el segundo carácter indicado (endIndex)
* **toCharArray():** Retorna un conjunto de caracteres de cadena
* **toLowerCase():**  Retorna la cadena en minúsculas
* **toUpperCase():** Retorna la cadena en mayúsculas

Podemos pasar un número a cadena y una cadena a número de la siguiente manera

* **De número a cadena:**
  + int *numEntero = 4;*

String *numCadena* = String.valueOf(*numeEntero*);

* **De cadena a nùmero:**
  + String *numCadena = 4;*

int *numEntero* = Integer.parseInt(*numeCadena*);

## Clase Math (funciones para Enteros y Reales)

* **abs(double a):** Devuelve el valor absoluto de un valor double introducido como parámetro
* **abs(int a):** Devuelve el valor absoluto de un valor entero introducido como parámetro
* **abs(long a):** Devuelve el valor absoluto de un valor long introducido como parámetro
* **max(double a, double b):** Devuleve el mayor de los valores doubles introducidos como parámetros
* **max(int a, int b):** Devuleve el mayor de los valores enteros introducidos como parámetros
* **max(long a, long b):** Devuleve el mayor de los valores long introducidos como parámetros
* **min(double a, double b):** Devuleve el menor de los valores doubles introducidos como parámetros
* **min(int a, int b):** Devuleve el menor de los valores enteros introducidos como parámetros
* **min(long a, long b):** Devuleve el menor de los valores long introducidos como parámetros.
* **pow(double a, double b):** Devuelve el valor del primer argumento elevado a la potencia del segundo argumento
* **random():** Devuelve un double con un signo positivo, mayor o igual que 0.0 y menor que 1.0
* **raund(double a):** Devuelve el long redondeado más cercano al double introducido
* **sqrt(double a):** Devuelve la raíz cuadrada positiva correctamente redondeada de un double
* **floor(double a):** Devuelve el entero más cercano por debajo

### Método random() de la clase Math

Sirve para crear números al azar. El margen con el que trabaja el método random oscila entre 0.0 y 1.0, este último no incluido, por lo que si queremos generar un número entero entre 0 y 9, hay que escribir la siguiente sentencia:

int *numero =* ( int ) (Math.random() \* 10);

# **GUÍA 1: RESUMEN VIDEO**

## Crear un programa

Creamos el archivo como lo indica la guía.

Cuando creamos un nuevo proyecto, lo que hacemos es crear la carpeta donde vamos a alojar todos los archivos de nuestro programa, es decir un paquete (package).

## Clases

Una clase (palabra reservada *class*) es una las unidades básicas y más importantes de java como lenguaje de tipo orientado a objetos

Dentro de una clase se colocan datos y códigos del programa que vamos a codificar.

En los primeros programas que vamos a hacer solo usamos una clase porque son muy sencillos.

La extensión que tienen los programas de tipo Java es ***.java.***

## Como queda armada la estructura del programa de afuera hacia adentro

1. **package:** nombre que le pongo cuando creo el archivo.
2. **class:** por defecto cuando lo creamos le pone el mismo nombre que el package dos veces separados por una coma.
3. **main:** es un subprograma del tipo rama maestra que permite ejecutar el programa (como Algoritmo/FinAlgoritmo en PSInt). Es lo primero que se ejecuta cuando ejecuto un programa.

Main es un método, como cualquier función o procedimiento, solo que se los llama método cuando están dentro de una clase, pero son lo mismo que una función.

1. **Sentencias:** es donde escribo las líneas de código. Todas la líneas terminan con punto y coma (;) esto se llama delimitador de línea.
2. **La primer sentencia, Scanner:** La primer línea es Scanner *leer = new Scanner(System.in);*

Que es lo que hice en este simple paso

* 1. Cree una variable
  2. Declaré el tipo de variable (Scanner)
  3. Declarè el nombre de la variable (leer)
  4. Asignamos a la variable un objeto ( new Scanner(System.in))

1. **Escribo las sentencias que van a hacer al funcionamiento del programa:**
   1. Declaro la variable String y la nombro, en este caso con *nombre.*
   2. Escribo el mensaje por pantalla con <System.out.println(“Mensaje que quiero mostrar por pantalla.”);>
   3. Declaro la variable <*nombre = leer*.next();>
   4. Escribo el mensaje por pantalla con <System.out.println(“Mensaje que quiero mostrar por pantalla.” + *nombre*);> 🡺 Pongo + para que sin saltar línea se escriba el valor de la variable

# **GUÍA 2: ESTRUCTURAS DE CONTROL**

# INSTRUCCIONES DE BIFURCACIÓN

Mediante estas instrucciones el desarrollo lineal de un programa se interrumpe.

Para esto también vamos a usar los operadores lógicos o condicionales, estos son os mismos que en PSInt pero se escriben de distinta forma:

## Operadores condicionales

* && 🡺 AND (y)
* ││ 🡺 OR (o)
* ! 🡺 Operador Lógico de Negación

# ESTRUCTURAS DE CONTROL

Las estructuras de control son construcciones hechas a partir de palabras reservadas del lenguaje que permiten modificar el flujo de ejecución de un programa. De Este modo, pueden crearse construcciones de alternativas mediante sentencias condicionales y bucles de repetición de bloques de instrucciones. Un bucle de instrucciones se encontrará encerrado mediante llaves {…..} si existe más de una instrucción.

## Estructuras condicionales

Los condicionales son estructuras de control que cambian el flujo de ejecución de un programa de acuerdo con si se cumple o no una condición. Existen dos tipos de estructuras condicionales, las estructuras *if/elese*  y la estructura *switch.*

### Estructura if/else

Esta estructura le indica al programa que ejecute cierta parte del código sólo si la condición de evaluación es verdadera (<true>). La forma más simple de esta estructura es la siguiente:

If(*condición*) {

*<sentencia>*

}

Condición tiene que ser booleano para que se evalúe como verdadera o falsa. La <*sentencia*> *se ejecutará si, y solo si, la*  <*condición*> evaluada es verdadera.

En caso de querer ejecutar otra acción en caso de que <*condición*>se evalúe como falsa (<false>) debemos escribir la estructura *if/else* de la siguiente manera:

if(*condición*) {

*<sentencia 1>*

}else {

*<sentencia 2>*

}

En muchas ocasiones, se anidan estructuras if/else, de fomra que se pregunte por una condición si anteriormente no se ha cumplido otra y así sucesivamente.

if(*condición 1*) {

*<sentencia 1>*

}else if(*condición 2*){

*<sentencia 2>*

} else {

*<sentencia 3>*

}

### Estructura switch

El bloque *<switch>* evalúa qué valor tiene la variable, y de acuerdo con el valor que posee ejecuta las sentencias del bloque *<case>* correspondiente, es decir, del bloque*<case>* que cumpla con el valor de la variable que se está ecaluando dentro del *<switch>*.

switch*(<variable>*) {

case *<valor1>:*

<sentencia1>

break;

case *<valor2>:*

<sentencia2>

break;

default:

<sentencia3>

}

El caso de la sentencia <*break> que va detrás de cada* <case> termina la sentencia *<switch>* que la envuelve, es decir que el flujo de control del programa continúa con la primera sentencia que se encuentra a continuación del cierre del bloque *<switch>*.

La instrucción dentro del bloque *<default>* se ejecuta cuando la variable que se está evaluando no coincide con ninguno de los valores *<case>*.

## Estructuras repetitivas

Las estructuras que repiten una secuencia de acciones un número determinado de veces se denominan *bucles*, y se denomina *iteración* al hecho de repetir la ejecución de una secuencia de acciones.

Todo bucle tiene que llevar asociada una condición, que es la que se va a evaluar para determinar cuando se repite el *bucle* y cuando deja de repetirse.

## Sentencia de Salto

En Java existen dos formas de realizar un salto incondicional en el flujo “normal” de un programa. A saber, las instrucciones, *break* y *continue*.

### Break

La instrucción *break* sirve apara abandonar una estructura de control, tanto de las condiciones (if-else y switch) como de las repetitivas (for, do-while y while). En el momento en el que se ejecuta la instrucción break, el control del programa sale de la estructura en la que se encuentra contenida y continúa con el programa.

### Continue

La sentencia *continue* corta la iteración en donde se encuentra el *continue*, pero en lugar de salir del bucle, continúa con la siguiente iteración.

**La instrucción *continue* transfiere el control del programa desde la instrucción *continue* directamente a la cabecera del bucle (for, do-while o while) donde se encuentra.**

# While

La estructura *while* ejecuta un bloque de acciones mientras se cumple una condición. La condición se comprueba antes de empezar a ejecutar por primera vez el bucle, por lo tanto, si la condición se evalúa *<false>* en la primera iteración, entonces el bloque de instrucciones no se ejecutará ninguna vez.

while(*<condición>*) {

*<sentencias>*

}

# Do/While

En este tipo de bloques, el bloque se ejecuta siempre al menos una vez. El bucle de instrucciones se ejecutará mientras la condición se evalúe *<true>*. Por lo tanto, entre las instrucciones que se repiten deberá existir alguna, que en algún momento, haga que la condición se evalúe a *<false>*, de lo contrario el bucle será infinito.

do {

*<sentencias>*

} while(*<condición>*);

# For

La estructura *for* proporciona una forma compacta de recorrer un rango de valores cuando la cantidad de veces que se debe iterar un bloque de código es conocida. La forma general de la estructura *for* se puede expresar del siguiente modo:

for (*<inicialización>; <terminación>; <incremento>*) {

*<sentencias>*

}

La expresión *<inicialización>* inicializa el bucle y se ejecuta una sola vez al iniciar el bucle. El bucle se termina cuando al evaluar la expresión *<terminación>* el resultado que se obtiene es <*false>*. La expresión *<incremento>* se invoca después de cada iteración que realiza el bucle (para hacer un incremento de uno en uno pongo *i++*); esta expresión incrementa o decrementa un valor hasta que se cumpla la condición de *<terminación>* del bucle.

**Como regla general puede decirse que se utilizará el bucle *for* cuando se conozca la cantidad de veces que ha de repetirse el bucle de instrucciones. Se utilizará del bucle *do-while* cunado no se conoce exactamente el numero de veces que se ejecutará el bucle, pero se sabe que por lo menos se ha de ejecutar una vez. Se utilizará el bucle *while* cunado es posible que no se deba ejecutar el bucle ni una sola vez.**

# **GUÍA 2: RESUMEN VIDEO 1**

## Palabras reservadas en Java para las estructuras de control

* **Condicional Si:** if
* **Condicional SiNo:** else
* **Condicional Según:** switch
* **Condicional else if:** SiNo Si, este lo puedo concatenar (que es distinto a anidar en la forma de ejecución)

## Opciones de ejecución

Si tengo varios programas dentro del mismo paquete (*pakage*) y yo quiero ejecutar la que estoy viendo lo hago con **shift+F6.**

## Shitch

Tengo que poner break al terminar cada *case* para que corte la secuencia de acciones, si no lo pongo sigue con el siguiente *case* en vez de terminar la secuencia.

## Ejecución paso a paso o prueba de escritorio: Debug o Debugger

En Java, y en otros lenguajes, la prueba de escritorio y el paso a paso se llama **depurar o, en inglés, DEBUG o DEBURGGER** y lo ejecuto con **ctrl+F5.**

Tenemos que identificar un punto de **interrupción o, en inglés, BREAK POINT**. Para marcarlo debemos tocar el número de línea con el mouse. Y lo que hace es que va a ejecutar el programa hasta llagar al *break point*.

Los depuradores nos permiten ver los valores que van tomando las variables. Si lo quiero hacer me tengo que parar en la variable, dentro del *debug* y con secundario todo *enter New watch*.

Si quiero avanzar paso a paso lo hago con F8.

Si, una vez que llegó al *break point*, quiero que ejecute el programa hasta el final toco Play o F5.

# **GUÍA 2: RESUMEN VIDEO 2**

## Palabras reservadas en Java Bucles

* **while:** mientras
* **do while:** hacer mientras
* **for:** Para 🡺 1° Inicialización de la variable de control, 2° Lavariable de control es utilizada dentro de la expresión de terminación mientras la expresión de terminación es verdadera el bucle se mantienen en ejecución (es una variable booleana que opera para el funcionamiento del bucle, cuando da falso se termina el bucle), 3° Indicar el paso de la variable de control, con *i++* hago el incremento +1

## Sentencias de salto

* **break:** aborta el bucle y salta a la siguiente sentencia (deja de ejecutar el bucle)
* **continue:** no aborta el bucle, solo salta la iteración actual y vuelve a entrar al bucle, pero en la siguiente iteración saltando todas las líneas que estén por debajo del *continue*

## BreakPoint condicionado

Introduzco el breakpoint y toco secundario y toco propiedades. Luego en ejecución y pongo que llegue hasta la última o penúltima iteración, esto lohago escribiendo *i == num* (para el caso de la última iteración) o *i == (num-1)* (para el caso de la penúltima iteración). Luego toco Play o F5 para que ejecute el programa hasta el final.

# **GUÍA 3: SUBPROGRAMA**

Un método muy útil para solucionar problemas complejos es dividirlos en problemas más pequeños y sencillos, estos serán nuestros subprogramas.

El problema principal se resuelve en el programa o algoritmo principal, mientras que la solución de los subproblemas será a través de subprogramas, conocidos como procedimientos o funciones. Un subprograma recibe los datos necesarios para realizar una tarea desde el programa principal y devuelve los resultados de esa tarea.

# FUNCIONES

Una función según sus parámetros puede recibir argumentos (algunas no reciben nada), hace uso de dichos valores recibidos y retorna un valor usando la instrucción *return*, si no retorna es otro tipo de función. Los tipos que pueden usarse en la función son: int, double, long, boolean, String, char.

A estas funciones les vamos a asignar un tipo de acceso y modificador. Por ahora siempre vamos a crear las funciones con el acceso *public* y el modificador *static*.

[acceso] [modificador] [tipo] nombreDeLaFuncion([tipo] nombreDelArgumento, …) {

/\*

Bloque de instrucciones

(*secuencia de acciones y sentencias*)

\*/

return valor;

}

* [acceso] 🡺 public
* [modificador] 🡺 static
* [tipo] 🡺 como las variables (int, double, etc.)

## Consejos acerca del return

Cualquier instrucción que se encuentre después del *return* **NO SERÁ EJECUTADA**.

El tipo de valor que se retorna en una función debe coincidir con el tipo declarado a la función.

# PROCEDIMIENTOS (Funciones sin retorno)

En el contexto de Java un procedimiento es básicamente una función cuyo retorno es void, los que indica que devuelven ningún resultado.

[acceso] [modificador] void nombreDeLaFuncion([tipo] nombreDelArgumento, …) {

/\*

Bluque de instrucciones

(*secuencia de acciones y sentencias*)

\*/

return valor;

}

* [acceso] 🡺 public
* [modificador] 🡺 static
* void 🡺 como en el método main
* [tipo] 🡺 como las variables (int, double, etc.)

# ACERCA DE LOS ARGUMENTOS Y LOS PARÁMETROS

Una función, un método o un procedimiento pueden tener una cantidad infinita de parámetros, aunque habitualmente no suelen tener más de 4 o 5.

Si una función tiene más de un parámetro cada uno de ellos debe ir separado por una coma.

Los argumentos de una función también tienen un tipo y un nombre que los identifica. El tipo del argumento puede ser cualquiera y no tiene relación con el tipo del método.

Al recibir un argumento nada nos obliga a hacer uso de éste al interior del método.

En Java los argumentos que sean variables de tipos primitivos (int, double, char, etc.) se van a pasar por valor, mientras que los objetos (String, Interger, etc.) y los arreglos se van a pasar por referencia.

* Por valor 🡺 No se modifica el valor fuera del programa
* Por referencia 🡺 Sí se modifica el valor fuera del programa
* Parámetro 🡺 Son los datos que mando desde el programa principal al subprograma
* Argumento 🡺 Son los datos que recibe el subprograma que provienen del programa principal
* **Los parámetros y argumentos deben estar en el mismo orden tanto cuando “salen” del programa principal, como cuando “llegan” al subprograma.**

# **GUÍA 3: RESUMEN VIDEO 3**

# FUNICONES DE JAVA

Todas las funciones tienen el denominador de accesibilidad público (*public*), y el otro denominador va a ser *static*. EL retorno de la función va a ser *void*.

Tengo que alcarar el tipo de retorno (*int*, *String*, *boolean*, etc.).

Tengo que darle un nombre sí o sí y un tipo (*int*, *String*, *boolean*, etc.). Cuando la quiero usar tengo que hacerlo a trvés de este nombre.

Tengo que pasarle *parámetros* desde el algoritmo principal, cuando llegan a la función se pasan a llamar *argumentos* que son los valores con los que va trabajar la función.

Las funciones también tienen un cuerpo contenido entre llaves (**{ }**).

Puedo pasar parámetros directamente con la función de ingresar por teclado si al llamar la función pongo en los argumentos:

*nombreFuncion (****leer.next()****);*

# **GUÍA 4: AREREGLOS Y VECTORES**

Un arreglo es un contenedor de objetos que tienen un número fijo de valores del mismo tipo. El tamaño del arreglo es establecido cuando el arreglo es creado y luego de su creación su tamaño es fijo, esto significa que no puede cambiar. Cada uno de los espacios de un arreglo es llamado elemento y cada elemento puede ser accedido por un índice numérico que arranca desde el 0 hasta el tamaño menos uno. Por ejemplo, si tenemos un vector de 5 elementos los índices serían: 0, 1, 2, 3 y 4.

Al igual que la declaración de otros tipos de variables, la declaración de un arreglo tiene dos componentes: el tipo de dato del arreglo (int, String, boolean, etc.) y su nombre (el que le define el programador). El tipo de dato del arreglo se define como *tipo[]*, en donde, *tipo* es el tipo de dato (int, String, boolean, etc.) de cada elemento contenido en él. Los corchetes sirven para indicar que esa variable va a ser un arreglo. El tamaño del arreglo no es parte de su tipo (es por esto que los corchetes están vacíos).

Una vez declarado un arreglo hay que crearlo/dimensionarlo, es decir, hay que asignar al arreglo un tamaño para almacenar los valores. La creación de un arreglo se hace con el operador *new*. Recordemos que las matrices son bidimensionales por lo que tienen dos tamaños, uno para las filas y otro para las columnas.

* **Declaración de un vector:**

tipo[] nombreDelVector = new tipo [Tamaño];

* **Declaración de una matriz:**

tipo[] [] nombreDeLaMatriz = new tipo [Filas] [Columnas];

# ASIGNAR ELEMENTOS A UN ARREGLO

Cuando queremos ingresar un elemento a un arreglo vamos a tener que elegir el subíndice donde lo queremos guardar. Una vez tenemos decidido el subíndice tenemos que invocar nuestro vector/matriz por su nombre y entre corchetes el subíndice donde queremos guardar el elemento.

Después, pondremos el signo igual (que es el operador de asignación) seguido del elemento a guardar. En las matrices vamos a necesitar dos subíndices y dos corchetes para representar la posición de la fila y la columna donde queremos guardar el elemento.

* **Asignación de un vector:**

nombreDelVector [índice] = elemnto

Ej.: edades [0] = 5

* **Asignación de una matriz:**

nombreDeLaMatriz [fila] [columna] = elemnto

Ej.: medidas [0] [3] = 11

Esta forma de asignación implica cargar todos los valores de nuestro arreglo uno por uno, lo que no es práctico. Para poder asignar varios valores a nuestro arreglo y no hacerlo de uno en uno usamos el bucle **Para** de la siguiente forma, pondremos el valor inicial de nuestro arreglo y su valor final en las respectivas partes del Para. Nosotros usamos la variable creada en el Para, y la pasaríamos a nuestro arreglo para representar todos los subíndices del arreglo, de esa manera, recorriendo todas las posiciones y asignándole a cada posición un elemento.

Para poder asignar varios elementos a una matriz usamos dos bucles Para anidados, ya que un Para recorrerá las filas (variable i) y el otro las columnas (variable j).

* **Asignación de un vector:**

for ( int i = 0; i < 5; i++) {

nombreDelVector [i] = 5

}

* **Asignación de una matriz:**

for ( int i = 0; i < 5; i++) {

for ( int j = 0; j < 5; j++) {

nombreDeLaMatriz [i] [j] = 6

}

}

# Vectores y matrices en subprogramas

Los arreglos se pueden pasar como parámetros a un subprograma poniendo el tipo de dato delante del nombre del vector o matriz, pero debemos sumarle las llaves para respetar que es un vector o matriz, **los arreglos siempre se pasan por referencia.**+

En Java podemos devolver un vector o una matriz en una función para usarla en otro momento. Lo que tenemos que hacer es poner como tipo de dato de la función, el tipo de dato que tendrá el vector y así poder devolverlo.

* **Paso de un vector:**

public static void llenarVector (int [] nombreDelVector ) {

for ( int i = 0; i < n; i++) {

nombreDelVector [i] = num

}

}

* **Paso de una matriz:**

public static void llenarMatriz (int [] [] nombreDeLaMatriz ) {

for ( int i = 0; i < 5; i++) {

for ( int j = 0; j < 5; j++) {

nombreDeLaMatriz [i] [j] = 6

}

}

# GUÍA 4: RESUMEN VIDEO 4

# ARREGLOS

Son estructuras que nos permiten guardar múltiples valores del mismo tipo en memoria.

Son estructuras multidimensionales, cuando tienen una dimensión son vectores y si tienen más son matrices.

# DECLARACIÓN DE ARREGLOS

1° Declaro el tipo.

2° Dimensiono con la cantidad de pares de corchetes [].

3° Nombro la variable.

4° Construcción en memoria del arreglo, pongo el mismo tipo que en la declaración del vector y luego entre corchetes la cardinalidad (cantidad de elementos por dimensión del arreglo).

(1°tipo) **int** (2°dimensión) **[] []** (3°nombre) **arreglo** = (4° Construcción) (tipo) **int** (cardinalidad) **[3] [4];**

# FUNCIÓN: LARGO DE UNA ARREGLO *arreglo.length*

La función *.length* nos devuelve la cardinalidad de cada una de las dimensiones del arreglo.

Primero declaro el vector (por ejemplo):

int [] *vector* = new int [3];

Segundo muestro la longitud del vector:

System.out.println(“El vector es “+vector.length+”:”);

**OJO: ¡El largo no es el TAMAÑO del arreglo!**

**Tamaño: Cantidad de elementos que puede albergar el arreglo.**

**Largo: Cardinalidad, cantidad de elementos de cada dimensión.**

## RECORRER ARREGLOS

Para recorrer los arreglos no valemos de bucles, en este caso el bucle **for**.

for ( int i = 0; i < vector.length; i++) {

Vector [i] = leer.nextImt();

}

En los arreglos podemos usar el bucle **for mejorado** que nos permite recorrer arreglos sin necesidad de identificar el subíndice recorriéndolo elemento por elemento.

* **Para vectores:**

for (int elemento: vector) {

aux = aux + “ “ + elemento;

}

System.out.println(aux);

* **Para matrices:**

for (int [] fila : matriz) {

aux = “ “;

for ( int elemento: fila) {

aux += aux + “ “ + elemento;

}

}

System.out.println(aux);