CURSO DE PROGRAMACIÓN FULL STACK

INTRODUCCIÓN A JAVA

FUNDAMENTOS DEL LENGUAJE

INTRODUCCIÓN A JAVA

Hasta el momento hemos aprendido los diferentes tipos de estructuras de control comunes a todos los lenguajes de programación, dentro del paradigma de programación imperativa, haciendo uso del pseudo intérprete PSeInt. A partir de esta guía comenzaremos a introducir cada uno de los conceptos vistos hasta el momento, pero haciendo uso de un lenguaje de programación de propósito general como lo es Java.

JAVA

Java es un tipo de lenguaje de programación y una plataforma informática, creada y comercializada por Sun Microsystems en el año 1995 y desde entonces se ha vuelto muy popular, gracias a su fácil portabilidad a todos los sistemas operativos existentes.

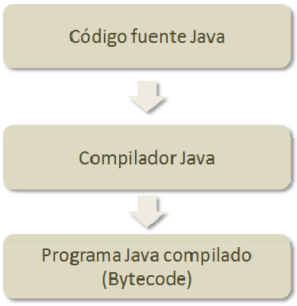
Java es un lenguaje de programación de alto nivel, estos, permiten escribir código mediante idiomas que conocemos (ingles, español, etc.) y luego, para ser ejecutados, se traduce al lenguaje de máquina mediante traductores o compiladores. Java es un lenguaje de alto nivel donde sus palabras reservadas están en ingles.

COMPILADOR EN JAVA

Permite traducir todo un programa de una sola vez, haciendo una ejecución más rápida y puede almacenarse para usarse luego sin volver a hacer la traducción. Los programas de Java se compilan a un lenguaje intermedio, denominado ByteCode. Este código es interpretado por la maquina virtual de Java (JVM) del entorno de ejecución (JRE) y así se consigue la portabilidad en distintas plataformas.

EJECUCIÓN DE UNA APLICACIÓN

En Java, todo el código fuente se escribe en archivos de texto plano cuya extensión es .java, en el ejemplo: MyProgram.java. Posteriormente, al compilar el código fuente, el compilador (javac) realiza un análisis de sintaxis del código escrito en los archivos fuente de Java (con extensión *.java). Si no encuentra errores en el código genera los archivos compilados con extensión .class. En caso de encontrar errores muestra la línea o líneas erróneas en el código. Los archivos .class no contienen código nativo del procesador, sino que contienen un conjunto de instrucciones que se denominan bytecodes, el lenguaje máquina de la JVM. Las instrucciones en bytecodes son independientes del tipo de computadora. Luego, el lanzador de aplicaciones java corre su aplicación con una instancia de la JVM en una computadora específica (Windows, Unix, MacOS, etc).



ARCHIVO FUENTE EN JAVA

Un archivo fuente de la tecnología Java tiene la siguiente forma:

```
[<declaración_paquete>]
<declaración_importacion>*
<modificador>* <declaración_clase>{
      <atributos>*
      <constructores>*
      <metodos>*
}
```

El orden de estos puntos es importante. Cualquier sentencia de importación debe preceder todas las declaraciones de clases. Si usa una declaración de paquetes, el mismo debe preceder tanto a las clases como a las importaciones. El nombre del archivo fuente tiene que ser el mismo que el nombre de la declaración de la clase pública en ese archivo. Un archivo fuente puede incluir más de una declaración de clase, pero *sólo una* puede ser declarada pública. Si un archivo fuente no contiene clases públicas declaradas, el nombre del archivo fuente no está restringido. Sin embargo, es una buena práctica tener un archivo fuente para cada clase declarada y que el nombre del archivo sea idéntico al nombre de la clase.

ESTRUCTURA DE UN PROGRAMA JAVA

Un programa describe cómo un ordenador debe interpretar las ordenes del programador para que ejecute y realice las instrucciones dadas tal como están escritas. Un programador utiliza los elementos que ofrece un lenguaje de programación para diseñar programas que resuelvan problemas concretos o realicen acciones bien definidas.

El siguiente programa Java muestra un mensaje en la consola con el texto “Hola Mundo”.

```
/*
* Este programa escribe el texto "Hola Mundo" en la consola * utilizando el método System.out.println()
*/
```

```
package primerprograma;
public class HolaMundo {
public static void main (String[] args) {
System.out.println("Hola Mundo");
    }
}
```

En este programa se pueden identificar los siguientes elementos del lenguaje Java: comentarios, paquete, definiciones de clase, definiciones de método y sentencias.

COMENTARIO

El programa comienza con un comentario. El delimitador de inicio de un comentario es **/*** y el delimitador de fin de comentario es ***/**.

El texto del primer comentario de este ejemplo seria: 'Este programa escribe el texto "Hola Mundo" en la consola utilizando el método System.out.println()'. 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. En un comentario se puede escribir todo lo que se desee, el texto puede ser de una o más líneas.

PAQUETES

Después del comentario viene está escrito el nombre del paquete. Los paquetes son contenedores de clases y su función es la de organizar la distribución de las clases. Los paquetes y las clases son análogos a las carpetas y archivos utilizadas por el sistema operativo, respectivamente.

El lenguaje de programación de la tecnología Java 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, en caso de existir, debe estar al principio del archivo fuente y sólo la declaración de un paquete está permitida. Los nombres de los paquetes los pondrá el programador al crear el programa y son jerárquicos (al igual que una organización de directorios en disco) además, están separados por puntos. Es usual que sean escritos completamente en minúscula.

CLASES

La primera línea del programa, después del package. Define una clase que se llama HolaMundo. 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. Los objetos, las clases y los paquetes son conceptos que serán abordados con profundidad más adelante en el curso.

Una clase está formada por una parte correspondiente a la declaración de la clase, y otra correspondiente al cuerpo de la misma:

```
Declaración de clase {

Cuerpo de clase

}
```

En la plantilla de ejemplo se ha simplificado el aspecto de la Declaración de clase, pero sí que puede asegurarse que la misma contendrá, como mínimo, la palabra reservada *class* y el nombre que recibe la clase. La definición de la clase o cuerpo de las comienza con una llave abierta ({) y termina con una llave cerrada (}). El nombre de la clase lo define el programador

MÉTODOS

Después de la definición de clase se escribe la definición del método main(). Pero que es un método?. Dentro del cuerpo de la clase se declaran los atributos y los métodos de la clase. Un método es una secuencia de sentencias ejecutables. Las sentencias de un método quedan delimitadas por los caracteres { y } que indican el inicio y el fin del método, respectivamente. Si bien es un tema sobre el que se profundizará más adelante en el curso, los métodos son de vital importancia para los objetos y las clases. En un principio, para dar los primeros pasos en Java nos alcanza con esta definición.

MÉTODO MAIN()

Ahora sabemos lo que es un método, pero en el ejemplo podemos ver el método main(). El main() sirve para que un programa se pueda ejecutar, este método, vendría a representar el Algoritmo / FinAlgoritmo de pseint y tiene la siguiente declaración:

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

A continuación, describirnos cada uno de los modificadores y componentes que se utilizan *siempre* en la declaración del método *main():*

public: es un tipo de acceso que indica que el método *main()* es público y, por tanto, puede ser llamado desde otras clases. Todo método *main()* debe ser público para poder ejecutarse desde el intérprete Java (JVM).

static: es un modificador el cual 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.

Luego, al indicarle a la máquina virtual que ejecute una aplicación el primer método que ejecutará es el método *main()*. Si indicamos a la máquina virtual que corra una clase que no contiene este método, se lanzará un mensaje advirtiendo que la clase que se quiere ejecutar no contiene un método *main()*, es decir que dicha clase no es ejecutable.

Si no se han comprendido hasta el momento muy bien todos estos conceptos, los mismos se irán comprendiendo a lo largo del curso.

SENTENCIA

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

En nuestro ejemplo, del método main() se incluye una sentencia para mostrar un texto por la consola. Los textos siempre se escriben entre comillas dobles para diferenciarlos de otros elementos del lenguaje. Todas las sentencias de un programa Java deben terminar con el símbolo punto y coma. Este símbolo indica al compilador que ha finalizado una sentencia.

Una vez que el programa se ha editado, es necesario compilarlo y ejecutarlo para comprobar si es correcto. Al finalizar el proceso de compilación, el compilador indica si hay errores en el código Java, donde se encuentran y el tipo de error que ha detectado: léxico, sintáctico o semántico.

ELEMENTOS DE UN PROGRAMA

Los conceptos vistos previamente, son la estructura de un programa, pero también existen los elementos de un programa. Estos son, básicamente, los componentes que van a conformar las sentencias que podamos escribir en nuestro programa. Recordemos que toda sentencia en nuestro programa debe terminar con el símbolo punto y coma Nos van a ayudar para crear nuestro programa y resolver sus problemas. Estos elementos siempre estarán dentro de un programa/algoritmo.

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

PALABRAS RESERVADAS

Palabras que dentro del lenguaje significan la ejecución de una instrucción determinada, por lo que no pueden ser utilizadas con otro fin. En Java, al ser un lenguaje que está creado en ingles, todas nuestra palabras reservadas van a estar en ese idioma.

IDENTIFICADOR

Los identificadores son los nombres que se usan para identificar cada uno de los elementos del lenguaje, como ser, los nombres de las variables, nombres de las clases, interfaces, atributos y métodos de un programa. Si bien Java permite nombres de identificadores tan largos que se desee, es aconsejable crearlos de forma que tengan sentido y faciliten su interpretación. El nombre ideal para un identificador es aquel que no se excede en longitud (lo más corto posible) y que califique claramente el concepto que intenta representar en el contexto del problema que se está resolviendo.

VARIABLES Y CONSTANTES

Recordemos que en Pseint dijimos que los programas de computadora necesitan información para la resolución de problemas. Está información puede ser un numero, un nombre, etc. Para utilizar la información, vamos a guardarla en algo llamado, variables y constantes. Las variables y constantes vendrían a ser como pequeñas cajas, que guardan algo en su interior, en este caso información. Estas, van a contar como previamente habíamos mencionado, con un identificador, un nombre que facilitara distinguir unas de otras y nos ayudara a saber que variable o constante es la contiene la información que necesitamos.

Dentro de toda la información que vamos a manejar, a veces, necesitaremos información que no cambie. Tales valores son las constantes. De igual forma, existen otros valores que necesitaremos que cambien durante la ejecución del programa; esas van a ser nuestras variables.

DECLARACIÓN DE VARIABLES EN JAVA

Normalmente los identificadores de las variables y de las constantes con nombre deben ser declaradas en los programas antes de ser utilizadas. La sintaxis de la declaración de una variable en java suele ser:

```
<tipo_de_dato> <nombre_variable>;
```

TIPOS DE DATO EN JAVA

Java es un lenguaje de *tipado estático*, esto significa que todas las variables *deben ser declaradas* antes que ellas puedan ser utilizadas y que no podemos cambiar el tipo de dato de una variable, a menos que, sea a traves de una conversión.

TIPOS DE DATOS PRIMITIVOS

Primitivos: Son predefinidos por el lenguaje. La biblioteca Java proporciona clases asociadas a estos tipos que proporcionan métodos que facilitan su manejo.

byte	Es un entero con signo de 8 bits, el mínimo valor que se puede almacenar es -128 y el máximo valor es de 127 (inclusive).
short	Es un entero con signo de 16 bits. El valor mínimo es -32,768 y el valor máximo 32,767 (inclusive).
int	Es un entero con signo de 32 bits. El valor mínimo es -2,147,483,648 y el valor máximo es 2,147,483,64 (inclusive). Generalmente es la opción por defecto.
long	Es un entero con signo de 64 bits, el valor mínimo que puede almacenar este tipo de dato es -9,223,372,036,854,775,808 y el máximo valor es 9,223,372,036,854,775,807 (inclusive).
float	Es un número decimal de precisión simple de 32 bits (IEEE 754 Punto Flotante).
double	Es un número decimal de precisión doble de 64 bits (IEEE 754 Punto Flotante).
boolean	Este tipo de dato sólo soporta dos posibles valores: verdadero o falso y el dato es representado con tan solo un bit de información.
char	El tipo de dato carácter es un simple carácter unicode de 16 bits. Su valor mínimo es de '\u0000' (En entero es 0) y su valor máximo es de '\uffff' (En entero es 65,535). Nota: un dato de tipo carácter se puede escribir entre comillas simples, por ejemplo 'a', o también indicando su valor Unicode, por ejemplo '\u0061'.
String	Además de los tipos de datos primitivos el lenguaje de programación Java provee también un soporte especial para cadena de caracteres a través de la clase String. Encerrando la cadena de caracteres con comillas dobles se creará de manera automática una nueva instancia de un objeto tipo String. String cadena = “Sebastián”; Los objetos String son inmutables, esto significa que una vez creados, sus valores no pueden ser cambiados. Si bien esta clase no es técnicamente un tipo de dato primitivo, el lenguaje le da un soporte especial y hace parecer como si lo fuera.

VALORES POR DEFECTO

En Java no siempre es necesario asignar valores cuando nuevos atributos son declarados. Cuando los atributos son declarados, pero no inicializados, el compilador les asignará un valor por defecto. A grandes rasgos el valor por defecto será cero o null dependiendo del tipo de dato. La siguiente tabla resume los valores por defecto dependiendo del tipo de dato.

byte	0
short	0
int	0
long	0
float	0.0
double	0.0
boolean	false
char	'\u0000'
String	null
Objetos	null

Las variables locales son ligeramente diferentes; el compilador no asigna un valor predeterminado a una variable local no inicializada. Las variables locales son aquellas que se declaran dentro de un método. Si una variable local no se inicializa al momento de declararla, se debe asignar un valor antes de intentar usarla. El acceso a una variable local no inicializada dará lugar a un error en tiempo de compilación.

OPERADORES

Los operadores son símbolos especiales de la plataforma que permiten especificar operaciones en uno, dos o tres operandos y retornar un resultado. También aprenderemos qué operadores poseen mayor orden de precedencia. Los operadores con mayor orden de precedencia se evalúan siempre primero.

Primeramente, proceden los operadores unarios, luego los aritméticos, después los de bits, posteriormente los relacionales, detrás vienen los booleanos y por último el operador de asignación. La regla de precedencia establece que los operadores de mayor nivel se ejecuten primero. Cuando dos operadores poseen el mismo nivel de prioridad los mismos se evalúan de izquierda a derecha.

OPERADOR DE ASIGNACIÓN	
=	Operador de Asignación Simple
OPERADORES ARITMÉTICOS	
+	Operador de Suma
-	Operador de Resta
*	Operador de Multiplicación
/	Operador de División
%	Operador de Módulo
OPERADORES UNARIOS	
+	Operador Unario de Suma; indica que el valor es positivo.
-	Operador Unario de Resta; indica que el valor es negativo.
++	Operador de Incremento.
--	Operador de Decremento.
OPERADORES DE IGUALDAD Y RELACIÓN	
==	Igual
!=	Distinto

>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que
OPERADORES CONDICIONALES	
&&	AND
ll	OR
!	Operador Lógico de Negación.

TIPOS DE INSTRUCCIONES

Además de los elementos de un programa/algoritmo, tenemos las instrucciones que pueden componer un programa. Las instrucciones —acciones— básicas que se pueden implementar de modo general en un algoritmo y que esencialmente soportan todos los lenguajes son las siguientes:

✓ **Instrucciones de inicio/fin,** son utilizadas para delimitar bloques de código.

✓ **Instrucciones de asignación,** se utilizan para asignar el resultado de la evaluación de una expresión a una variable. El valor (dato) que se obtiene al evaluar la expresión es almacenado en la variable que se indique.

```
<nombre de la variable> ← <expresión>
```

expresión es igual a una expresión matemática o lógica, a una variable o constante.

✓ **Instrucciones de lectura**, se utilizan para leer datos de un dispositivo de entrada y se asignan a las variables.

✓ **Instrucciones de escritura**, se utilizan para escribir o mostrar mensajes o contenidos de las variables en un dispositivo de salida.

✓ **Instrucciones de bifurcación**, mediante estas instrucciones el desarrollo lineal de un programa se interrumpe. Las bifurcaciones o al flujo de un programa puede ser según el punto del programa en el que se ejecuta la instrucción hacia adelante o hacia atrás.

INSTRUCCIONES PRIMITIVAS

Dentro de las instrucciones previamente vistas, existe un subdivisión que son las instrucciones primitivas, las instrucciones primitivas van a ser las instrucciones de asignación, lectura y escritura.

ASIGNACIÓN

La instrucción de asignación permite almacenar un valor en una variable (previamente definida). Esta es nuestra manera de guardar información en una variable, para utilizar ese valor en otro momento.

```
<variable> = <expresión>
```

En Java, podemos definir una variable y al mismo tiempo podemos asignarle un valor a diferencia de Pseint:

```
<tipo_de_dato> <nombre_variable> = expresion;
```

Al ejecutarse la asignación, primero se evalúa la expresión de la derecha y luego se asigna el resultado a la variable de la izquierda. El tipo de la variable y el de la expresión deben coincidir.

ENTRADA Y SALIDA DE INFORMACIÓN

Los cálculos que realizan las computadoras requieren, para ser útiles la entrada de los datos necesarios para ejecutar las operaciones que posteriormente se convertirán en resultados, es decir, salida.

Las operaciones de entrada permiten leer determinados valores y asignarlos a determinadas variables y las operaciones de salida permiten escribir o mostrar resultados de determinadas variables y las operaciones, o simplemente mostrar mensajes.

ESCRITURA EN JAVA

En nuestro ejemplo de código al principio de la guía, usábamos la instrucción System.out.println() para mostrar el mensaje Hola Mundo. Esta instrucción permite mostrar valores en el Output, que es la interfaz grafica de Java. Todo lo que quisiéramos mostrar en nuestra interfaz grafica, deberá ir entre comillas dobles y dentro del paréntesis.

```
System.out.println(“Hola Mundo”);
```

Si quisiéramos que concatenar un mensaje y la impresión de una variable deberíamos usar el símbolo más para poder lograrlo.

```
System.out.println(“La variable tiene el valor de: ” + variable);
```

Si quisiéramos escribir sin saltos de línea, deberíamos quitarle el ln a nuestro System.out.println.

```
System.out.print("Hola");
System.out.print(“Mundo”);
```

LECTURA O ENTRADA EN JAVA

En Java hay muchas maneras de ingresar información en el Output por teclado en nuestro programa Java, en el curso vamos a usar la clase Scanner.

Scanner es una clase en el paquete **java.util** utilizada para obtener la entrada de los tipos primitivos como int, double etc. y también String. Es **la forma más fácil de leer datos** en un programa Java.

Ejemplo definición clase Scanner:

```
Scanner leer = new Scanner(System.in);
```

- Este objeto Scanner vamos a tener que importarlo para poder usarlo, ya que es una herramienta que nos provee Java. Para importarlo vamos a utilizar la palabra clave **import**, seguido de la declaración de la librería donde se encuentra el Scanner. Esta sentencia, va debajo de la sentencia package. La sentencia se ve así: **import java.util.Scanner;**
- Para crear un objeto de clase Scanner, normalmente pasamos el objeto predefinido System.in, que representa el flujo de entrada estándar.
- Se le puso el nombre leer, pero se le puede el nombre que nosotros quisiéramos.
- Para utilizar las funciones del objeto Scanner, vamos a utilizar el nombre que le hemos asignado y después del nombre ponemos un punto(.), de esa manera podremos llamar a las funciones del Scanner.
- Para leer valores numéricos de un determinado tipo de datos, la función que se utilizará es **nextT().** Por ejemplo, para leer un valor de tipo int (entero), podemos usar nextInt(), para leer un valor de tipo double(real), usamos nextDouble() y etc. Para leer un String (cadenas), usamos nextLine().

Podemos usarlo cuando definimos la variable:

```
int numero = leer.nextInt();
```

Podemos usarlo con una variable pre definida:

```
int numero;
numero = leer.nextInt();
```

Nota: pueden encontrar un ejemplo lectura y entrada en Java en Moodle.

INSTRUCCIONES DE BIFURCACIÓN

Mediante estas instrucciones el desarrollo lineal de un programa se interrumpe. Las bifurcaciones o al flujo de un programa puede ser según el punto del programa en el que se ejecuta la instrucción hacia adelante o hacia atrás. De esto se encargan las estructuras de control.

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. Hay que señalar que un bloque 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 a si se cumple o no una condición. Cuando el flujo de control del programa llega al condicional, el programa evalúa la condición y determina el camino a seguir. Existen dos tipos de estructuras condicionales, las estructuras *if / else* y la estructura *switch*.

IF/ELSE

La estructura *if* es la más básica de las estructuras de control de flujo. Esta estructura le indica al programa que ejecute cierta parte del código sólo si la condición evaluada es verdadera («true»). La forma más simple de esta estructura es la siguiente:

```
if(<condición>){
  <sentencias>
}
```

En donde, <*condición*> es una expresión condicional cuyo resultado luego de la evaluación es un dato booleano(lógico) verdadero o falso. El bloque de instrucciones <sentencias> se ejecuta si, y sólo si, la expresión (que debe ser lógica) se evalúa a true, es decir, se cumple la condición.

Luego, en caso de que la condición no se cumpla y se quiera ejecutar otro bloque de código, otra forma de expresar esta estructura es la siguiente:

```
if(<condición>){
  <sentencias A>
} else {
   <sentencias B>
}
```

El flujo de control del programa funciona de la misma manera, cuando se ejecuta la estructura if, se evalúa la expresión condicional, si el resultado de la condición es verdadero se ejecutan las sentencias que se encuentran contenidas dentro del bloque de código if (<sentencias A>). Contrariamente, se ejecutan las sentencias contenidas dentro del bloque else (<sentencias B>).

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

```
if (<condicion1>) {
<sentencias A>
} else if(<condicion2>){
<sentencias B>
} else {
<sentencias C>
}
```

Al contrario de la estructura if / else, la estructura *switch* permite cualquier cantidad de rutas de ejecución posibles. Un switch funciona con los datos primitivos byte, short, char e int. También funciona con Enumeraciones, tema que se verá más adelante en el curso, y con unas cuantas clases especiales que «envuelven» a ciertos tipos primitivos: Character, Byte, Short, e Integer (tema que trataremos cuando se profundice en Orientación a Objetos).

SWITCH

El bloque *switch* evalúa qué valor tiene la variable, y de acuerdo al 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 evaluando dentro del switch.

```
switch(<variable>) {
case <valor1>:
<sentencias1>
break;
case <valor2>:
<sentencias2>
break;
default:
<sentencias3>
}
```

El uso de la sentencia break que va detrás de cada case termina la sentencia switch que la envuelve, es decir que el control de flujo del programa continúa con la primera sentencia que se encuentra a continuación del cierre del bloque switch. Si el programa comprueba que se cumple el primer valor (valor1) se ejecutará el bloque de instrucciones <sentencias1>, pero si no se encuentra inmediatamente la sentencia break también se ejecutarían las instrucciones <sentencias2>, y así sucesivamente hasta encontrarse con la palabra reservada break o llegar al final de la estructura.

Las instrucciones dentro del bloque default se ejecutan cuando la variable que se está evaluando no coincide con ninguno de los valores case. Esta sentencia equivale al else de la estructura if-else.

Nota: pueden encontrar un ejemplo de estructuras condicionales en Moodle.

ESTRUCTURAS REPETITIVAS

Durante el proceso de creación de programas, es muy común, encontrarse con que una operación o conjunto de operaciones deben repetirse muchas veces. Para ello es importante conocer las estructuras de algoritmos que permiten repetir una o varias acciones, un número determinado de veces.

Las estructuras que repiten una secuencia de instrucciones 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 va a determinar cuándo se repite el bucle y cuando deja de repetirse.

WHILE

La estructura *while* ejecuta un bloque de instrucciones 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 a «false» en la primera iteración, entonces el bloque de instrucciones no se ejecutará ninguna vez.

```
while (<condición>) {
<sentencias>
}
```

DO / WHILE

En este tipo de bucle, el bloque instrucciones se ejecuta siempre al menos una vez. El bloque de instrucciones se ejecutará mientras la condición se evalúe a «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>);
```

La diferencia entre *do-while* y *while* es que do-while evalúa su condición al final del bloque en lugar de hacerlo al inicio. Por lo tanto, el bloque de sentencia después del "do" siempre se ejecutan al menos una vez.

FOR

La estructura *for* proporciona una forma compacta de recorrer un rango de valores cuando la cantidad de veces que se deber 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 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; esta expresión incrementa o decrementa un valor hasta que se cumpla la condición de <terminación> del bucle.

La estructura for también ha sido mejorada para iterar de manera más compacta las colecciones y los arreglos, tema que se verá más adelante en este curso. Esta versión mejorada se conoce como for-each.

Como regla general puede decirse que se utilizará el bucle for cuando se conozca de antemano el número exacto de veces que ha de repetirse un determinado bloque de instrucciones. Se utilizará el bucle do-while cuando no se conoce exactamente el número de veces que se ejecutará el 3bucle, pero se sabe que por lo menos se ha de ejecutar una. Se utilizará el bucle while cuando es posible que no deba ejecutarse ninguna vez.

Nota: pueden encontrar un ejemplo de **estructuras repetitivas** en Moodle.

SENTENCIAS 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 para abandonar una estructura de control, tanto de las condicionales (if-else y switch) como de las repetitivas (for, do-while y while). En el momento que se ejecuta la instrucción break, el control del programa sale de la estructura en la que se encuentra contenida y continua 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.

Nota: pueden encontrar un ejemplo de **sentencias de salto** en Moodle.

SUBPROGRAMAS

Un método muy útil para solucionar un problema complejo es dividirlo en subproblemas —problemas más sencillos— y a continuación dividir estos subproblemas en otros más simples, hasta que los problemas más pequeños sean fáciles de resolver. Esta técnica de dividir el problema principal en subproblemas se suele denominar "divide y vencerás".

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

FUNCIONES

Las funciones o métodos son un conjunto de líneas de código (instrucciones), encapsulados en un bloque, usualmente según los parámetros definidos en la función, esta recibe argumentos, cuyos valores se utilizan para efectuar operaciones y adicionalmente retornan un valor. En otras palabras, una función según sus parámetros, puede recibir argumentos (algunas no reciben nada), hace uso de dichos valores recibidos como sea necesario 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, doble, long, boolean, String y char.

A estas funciones les vamos a asignar un tipo de acceso y un modificador. Estos dos conceptos los vamos a ver mejor más adelante, pero por ahora siempre vamos a crear las funciones con el acceso public y el modificador static. **Para saber más sobre estos dos temas, leer la explicación del método main.**

```
[acceso][modificador][tipo] nombreFuncion([tipo] nombreArgumento, …….){
    /*
        * Bloque de instrucciones
    */
    return valor;
}
```

PROCEDIMIENTOS (FUNCIONES SIN RETORNO)

Los procedimientos son básicamente un conjunto de instrucciones que se ejecutan sin retornar ningún valor, hay quienes dicen que un procedimiento no recibe valores o argumentos, sin embargo, en la definición no hay nada que se lo impida. En el contexto de Java un procedimiento es básicamente una **función** cuyo tipo de retorno es void, los que indica que devuelven ningún resultado.

```
[acceso][modificador] void nombreFuncion([tipo] nombreArgumento){
    /*
        * Bloque de instrucciones
    */
}
```

Acerca de los argumentos o parámetros:

Hay algunos detalles respecto a los argumentos de un método, veamos:

- Una función, un método o un procedimiento pueden tener una cantidad infinita de parámetros, es decir pueden tener cero, uno, tres, diez, cien o más 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, sin embargo, para que recibirlo si no lo vamos a usar.
- 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, Integer, etc.) y los arreglos se van a pasar por referencia. Nota: el concepto de objetos lo vamos a ver más adelante.

Consejos acerca de return:

- Cualquier instrucción que se encuentre después de la ejecución de return NO será ejecutada. Es común encontrar funciones con múltiples sentencias return al interior de condicionales, pero una vez que el código ejecuta una sentencia return lo que haya de allí hacia abajo no se ejecutará.
- El tipo de valor que se retorna en una función debe coincidir con el del tipo declarado a la función, es decir si se declara int, el valor retornado debe ser un número entero.
- En el caso de los procedimientos (void) podemos usar la sentencia return pero sin ningún tipo de valor, sólo la usaríamos como una manera de terminar la ejecución del procedimiento.

ARREGLOS: VECTORES Y MATRICES

Un arreglo es un contenedor de objetos que tiene 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 una de los espacios de un arreglo es llamada elemento y cada elemento puede ser accedido por un índice numérico que arranca desde 0 hasta el tamaño menos uno. Por ejemplo, si tenemos un vector de 5 elementos mis índices serian: 0-1-2-3-4

Al igual que la declaración de otros tipos de variables, la declaración de un arreglo tiene dos componentes: el tipo de dato del arreglo y su nombre. El tipo de dato del arreglo se escribe como *tipo[]*, en donde, tipo es el tipo de dato 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 de la matriz.

Declaración y creación de un Vector

```
tipo[] arregloV = new tipo[Tamaño];
```

Declaración y creación de una Matriz

```
tipo[][] arregloM = new tipo[Filas][Columnas];
```

ASIGNAR ELEMENTOS A UN ARREGLO

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

Después, pondremos el signo de 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

```
vector[0] = 5;
```

Asignación de una Matriz

```
matriz[0][0] = 6;
```

Esta forma de asignación implica asignar todos los valores de nuestro arreglo de uno en uno, esto va a conllevar un trabajo bastante grande dependiendo del tamaño de nuestro arreglo.

Entonces, para poder asignar varios valores a nuestro arreglo y no hacerlo de uno en uno usamos un bucle Para. El bucle Para, al poder asignarle un valor inicial y un valor final a una variable, podemos adaptarlo fácilmente a nuestros arreglos. Ya que, pondríamos el valor inicial de nuestro arreglo y su valor final en las respectivas partes del Para. Nosotros, usaríamos 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 nuestra matriz, usaríamos dos bucles Para anidados., ya que un Para recorrerá las filas *(variable i)* y otro las columnas *(variable j).*

Asignación de un Vector

```
for (int i = 0; i < 5; i++) {
  vector[i] = 5;
}
```

Asignación de una Matriz

```
for (int i = 0; i < 3; i++) {
for (int j = 0; j < 3; j++) {
  matriz[i][j] = 6;
  }
}
```

Nota: pueden encontrar un ejemplo de **vectores y matrices** en Moodle.

VECTORES Y MATRICES EN SUBPROGRAMAS

Los arreglos se pueden pasar como parámetros a un subprograma (función o procedimiento) del mismo modo que pasamos variables, poniendo el tipo de dato delante del nombre del vector o matriz, pero deberemos sumarle las llaves para representar que es un vector o matriz. Sin embargo, hay que tener en cuenta que la diferencia entre los arreglos y las variables, es que los arreglos siempre se pasan por referencia.

```
public static void llenarVector(int[] vector){
}
public static void mostrarMatriz(int[][] matriz){
}
```

A diferencia de Pseint, en Java si podemos devolver un vector o una matriz en una función para usarla en otro momento. Lo que hacemos es poner como tipo de dato de la función, el tipo de dato que tendra el vector y asi poder devolverlo.

```
public static int devolverVector(){
int[] vector = new int[5];
return vector;
```

CLASES DE UTILIDAD

Dentro del API de Java existe una gran colección de clases que son muy utilizadas en el desarrollo de aplicaciones. Las clases de utilidad son clases que definen un conjunto de métodos que realizan funciones, normalmente muy reutilizadas. Estas nos van a ayudar junto con las estructuras de control, a lograr resolver problemas de manera más sencilla.

Entre las clases de utilidad de Java más utilizadas y conocidas están las siguientes: Arrays, String, Integer, Math, Date, Calendar y GregorianCalendar. En esta guía solo vamos a ver la **Math**, **String** para hacer algunos ejercicios y después veremos el resto en mayor profundidad.

CLASE STRING

La clase String está orientada al manejo de cadenas de caracteres y pertenece al paquete java.lang del API de Java. Los objetos que son instancias de la clase String, se pueden crear a partir de cadenas constantes también llamadas literales, las cuales deben estar contenidas entre comillas dobles. Una instancia de la clase String es inmutable, es decir, una vez que se ha creado y se le ha asignado un valor, éste no puede modificarse (añadiendo, eliminando o cambiando caracteres).

Al ser un objeto, una instancia de la clase String no sigue las normas de manipulación de los datos de tipo primitivo con excepción del operador concatenación. El operador + realiza una concatenación cuando, al menos, un operando es un String. El otro operando puede ser de un tipo primitivo. El resultado es una nueva instancia de tipo String.

Método	Descripción.
charAt(int index)	Retorna el carácter especificado en la posición index
equals(String str)	Sirve para comparar si dos cadenas son iguales. Devuelve true si son iguales y false si no.
equalsIgnoreCase(String str)	Sirve para comparar si dos cadenas son iguales, ignorando la grafía de la palabra. Devuelve true si son iguales y false si no.
compareTo(String otraCadena)	Compara dos cadenas de caracteres alfabéticamente. Retorna 0 si son iguales, entero negativo si la primera es menor o entero positivo si la primera es mayor.
concat(String str)	Concatena la cadena del parámetro al final de la primera cadena.
contains(CharSequence s)	Retorna true si la cadena contiene la secuencia tipo char del parámetro.
endsWith(String suffix)	Retorna verdadero si la cadena es igual al objeto del parámetro
indexOf(String str)	Retorna el índice 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, char newChar)	Retorna una nueva cadena reemplazando los caracteres del primer parámetro con el carácter del segundo parámetro
split(String regex)	Retorna un arreglo de cadenas separadas por la cadena del parámetro
startsWith(String prefix)	Retorna verdadero si el comienzo de la cadena es igual al prefijo del parámetro
substring(int beginIndex)	Retorna la sub cadena desde el carácter del parámetro
substring(int beginIndex, int endIndex)	Retorna la sub cadena desde el carácter del primer parámetro hasta el carácter del segundo parámetro
toCharArray()	Retorna el conjunto de caracteres de la cadena
toLowerCase()	Retorna la cadena en minúsculas
toUpperCase()	Retorna la cadena en mayúsculas

Java al ser un lenguaje de tipado estático, requiere que para pasar una variable de un tipo de dato a otro necesitemos usar un conversor. Por lo que, para convertir cualquier tipo de dato a un String, utilicemos la función valueOf(n).

Ejemplo:

```
int numEntero = 4;
String numCadena = String.valueOf(numEntero);
```

Si quisiéramos hacerlo al revés, de String a int se usa el método de la clase Integer, parseInt().

Ejemplo:

```
String numCadena = "1";
int numEntero = Integer.parseInt(numCadena);
```

CLASE MATH

En ocasiones nos vemos en la necesidad de incluir **cálculos, operaciones, matemáticas, estadísticas**, etc en nuestro programas Java.

Es cierto que muchos cálculos se pueden hacer simplemente utilizando los operadores aritméticos que java pone a nuestra disposición, pero existe una opción mucho más sencilla de utilizar, sobre todo para *cálculos complicados*. Esta opción es la **clase Math** del paquete **java.lang**.

La clase Math nos ofrece numerosos y valiosos métodos y constantes estáticos, que podemos utilizar tan sólo anteponiendo el nombre de la clase.

Método	Descripción.
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)	Devuelve el mayor de dos valores double
max(int a, int b)	Devuelve el mayor de dos valores Enteros.
max(long a, long b)	Devuelve el mayor de dos valores long.
min(double a, double b)	Devuelve el menor de dos valores double.
min(int a, int b)	Devuelve el menor de dos valores enteros.
min(long a, long b)	Devuelve el menor de dos valores long.
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.
round(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

El **método random** podemos utilizarlo para generar **números al azar**. El rango o margen con el que trabaja el método random oscila entre 0.0 y 1.0 (Este último no incluido)

Por lo tanto, para generar un número entero entre 0 y 9, hay que escribir la siguiente sentencia:

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

Nota: pueden encontrar un ejemplo de las **funciones de la clase String y Math** en Moodle.