**INTRODUCCIÓN A JAVA**

**Qué es Java**

**Introducción**

JAVA es una tecnología pensada para desarrollo de aplicaciones de gran envergadura, altamente escalables, de gran integración con otras tecnologías y sumamente robustas.

Sus principales características son presentadas a continuación:

**Lenguaje orientado a objetos**

Respeta el paradigma de orientación a objetos, permitiendo utilizar los fundamentos del mismo:

* Herencia
* Polimorfismo
* Abstracción
* Encapsulamiento

Todos estos conceptos serán presentados más adelante.

**Sintaxis basada en C/C++**

Aporta gran simplicidad ya que es una de las formas de escribir código más reconocidas y difundidas, y permite incorporar rápidamente a los programadores que conocen este lenguaje.

**Multiplataforma**

Significa que su código es portable, es decir se puede transportar por distintas plataformas. De esta manera es posible codificar una única vez una aplicación, y luego ejecutarla sobre cualquier plataforma y/o sistema operativo.

**Manejo automático de memoria**

No hay que preocuparse por liberar memoria manualmente ya que un proceso propio de la tecnología se encarga de monitorear, y por consiguiente eliminar el espacio ocupado que no está siendo utilizado. El proceso encargado de realizar este trabajo se denomina Garbage Collector.

**Evolución permanente**

La tecnología está en constante evolución debido a la gran cantidad de “consumidores” que poseen, JAVA es uno de los lenguajes más utilizados en el mundo, y SUN pretende estar a la altura de la situación ofreciendo constantemente nuevas entregas.

O**rganización**

La tecnología está organizada en tres grandes áreas bien definidas:

**JME (Mobile / Wireless)**

Esta área tiene como objetivo el desarrollo de aplicaciones móviles, tales como GPS, Handhelds (por ejemplo la conocida Palm), celulares y otros dispositivos móviles programables. JME significa Java Micro Edition.

**JSE (Core / Desktop)**

Esta área tiene como objetivo el desarrollo de aplicaciones de escritorio, similares a las aplicaciones tipo ventanas creadas con Visual Basic o Delphi. Incluye la funcionalidad básica del lenguaje como manejo de clases, colecciones, entrada/salida, acceso a base de datos, manejo de sockets, hilos de ejecución, etc. JSE significa Java Standard Edition.

**JEE (Enterprise / Server)**

Esta área tiene como objetivo el desarrollo de aplicaciones empresariales, de gran envergadura. Contempla ambientes web, como los ambientes manejados por servidores de aplicación. Las tecnologías principales incluidas en esta área son Servlets, JSP y EJB, entre otras. JEE significa Java Enterprise Edition.

**La historia**

**El comienzo**

En el año 1990 nace Java, bajo el diseño y la implementación de la empresa Sun Microsystems. El padre-fundador de la tecnología es James Gosling, a través de una filial dentro de Sun llamada First Person Inc.

Gosling tuvo la visión inicial de construir un lenguaje de programación capaz de ejecutar su código sobre cualquier set de instrucciones, de distintos procesadores. Inicialmente el proyecto apuntó a la programación unificada de distintos electrodomésticos, es decir programar una sola vez y que el programa generado fuera útil para cualquier dispositivo.

El proyecto inicial de Java fue técnicamente un éxito, aunque comercialmente no tuvo el rendimiento esperado, y debió ser relegado unos años.

**Aparición de Internet**

En el año 1993, Internet da el gran salto, y se convierte de una interfaz textual a una interfaz gráfica.

Java ve una oportunidad y entra fuertemente a internet con los Applets, pequeños programitas construidos en Java – con todos sus beneficios – capaces de ejecutarse dentro de un navegador. Es aquí donde Java comienza a dar sus primeros pasos firmes como lenguaje a difundir masivamente. En el año 1995, el navegador Netscape Navigator comienza formalmente a soportar los Applets Java.

Adicionalmente, el lenguaje podía adaptarse fácilmente a las múltiples plataformas, con lo cual surge una de las primeras aplicaciones multiplataformas más conocidas: WebRunner (hoy HotJava), un navegador multiplataforma construido en Java.

**Por qué el nombre JAVA**

Inicialmente la intención fue nombrar al lenguaje de programación con el nombre de Oak, pero este ya estaba registrado. La leyenda cuenta que una visita a la cafetería le dio rápida solución al problema.

En las confiterías norteamericanas hay un café denominado Java, en el cual está inspirado el nombre del lenguaje de programación. El logotipo de Java es justamente una taza café.

**DESARROLLO, COMPILACIÓN Y EJECUCIÓN**

**Organización**

**El Java Development Kit (JDK)**

El Java Development Kit es el kit de desarrollo propuesto por Sun Microsystems para realizar desarrollos en JAVA. Se puede bajar de forma gratuita de la página [http://www.java.sun.com](https://www.google.com/url?q=http://www.java.sun.com/&sa=D&ust=1503402017508000&usg=AFQjCNGcSK1Ux3lFJggp52ro-1sGN6Sx3Q)

El kit incluye herramientas de desarrollo tales como un compilador, un debugger, un documentador para documentar en forma casi automática una aplicación, un empaquetador para crear archivos de distribución, y otras herramientas más.

El kit no incluye un entorno de desarrollo interactivo (o IDE) como pueden ser Netbeans, Jdeveloper o Eclipse.

**El compilador**

El compilador viene incluido como una herramienta dentro de la JDK, en el sistema operativo Windows viene presentado como javac.exe

El compilador transforma los archivos de código fuente de java, es decir los archivos de texto con extensión .java, en archivo compilados, también denominados bytecode. Los archivos compilados tienen la extensión .class, y son archivos binarios.

**El Java Runtime Environment (JRE)**

Java Runtime Environment es el ambiente de ejecución de Java, y también está incluido en la JDK. Tiene como componentes más importantes a la Java Virtual Machine y a las class libraries, que son las que contienen las clases base del lenguaje de programación JAVA.

El JRE se distribuye también en forma independiente, es decir sin la JDK, ya que cuando es necesario desplegar una aplicacion hecha en JAVA en el cliente, no es necesario instalarle herramientas que son propias del proceso de desarrollo, como ser el compilador, empaquetador, documentador, y otros.

Sin una JRE instalada no es posible ejecutar una aplicacion construida en JAVA.

En Windows, el comando para invocarlo es el java.exe

**La Java Virtual Machine (JVM)**

**Qué es**

La Java Virtual Machine viene incluida dentro de la Java Runtime Environment, y tiene como principal objetivo la ejecución de código JAVA compilado, es decir de los archivos .class

La JVM se encarga de interpretar el bytecode y convertirlo a código nativo en tiempo de ejecución, lo cual hace que la ejecución sea un poco más lenta, pero garantiza la portabilidad, es decir que el lenguaje sea multiplataforma. De esta manera el código compilado JAVA se puede ejecutar en cualquier plataforma (arquitectura + sistema operativo) que tenga instalada el JRE.

"Write once, run anywhere" es la política desde el primer dia de JAVA, es decir construir la aplicación una vez y ejecutarla en “cualquier lado”.

**La variable de entorno CLASSPATH**

La variable de entorno CLASSPATH se utiliza para referenciar el directorio donde estarán ubicadas todas las clases construidas en JAVA, para que el JRE al ejecutar una clase sepa dónde ubicar el resto de las clases o archivos empaquetados que contienen clases.

**SINTAXIS Y SEMÁNTICA DE JAVA**

**Definición de variables**

**Identificador**

A la hora de nombrar las variables es importante tener en cuenta que no pueden comenzar con un número ni utilizar caracteres "%" o "\*" o "@" porque están reservados para otras operaciones.

Pueden comenzar con los caracteres "\_" o "$", aunque la especificación no recomienda el uso del caracter “$” en variables.

**Tipos de variables**

Las variables pueden ser declaradas con tipo de datos primitivos (int, long, float, double, etc) o también como tipos, pertenecientes a alguna clase, como por ejemplo la clase String para trabajar con cadenas de caracteres.

**Declaración y definición**

Un variable del tipo entero puede ser declarada de la siguiente forma:

int var;

Si se desea declarar y asignar un valor, en este caso resulta necesario definirla. La definición se realiza de la siguiente manera:

int var = 200;

**Vectores**

Para manejar un conjunto de valores se utilizan los vectores, que agrupan una serie de valores en una única variable. Para trabajar con arreglos (vectores) es necesario determinar el tipo de información que contendrá y la longitud – o cantidad de elementos – que tendrá el propio vector.

Si, por ejemplo, resulta necesario construir un vector para almacenar números enteros, se utilizará un vector del tipo entero de, por ejemplo, 10 posiciones, que luego será necesario llenarlo con valores. Lo dicho anteriormente, se realiza de la siguiente manera:

int[ ] vector =  new int[10];

vec[0] = 150;

vec[1] = 300;

vec[2] = 500;

Si se conocen de antemano los valores que contendrá el vector, podemos armar el vector de una forma más resumida, presentada a continuación:

int[ ] vector = {150,300,500,4,5,6};

Los vectores cuentan con la posibilidad de solicitarles que longitud poseen, esto se realiza a través del atributo length.

Es importante recordar que en JAVA – a diferencia de otros lenguajes de programación - la primera posición de un vector es la posición 0 (cero).

**Tipos de dato primitivos**

**boolean**

El tipo de dato boolean se utiliza para almacenar las palabras claves true o false, es decir verdadero o falso. Ocupan 1 byte en memoria.

**char**

El tipo de dato char se utiliza para almacenar un solo caracter, del tipo Unicode. Ocupan 2 bytes en memoria.

**byte**

El tipo de dato byte es un tipo de dato numérico y entero, se utiliza para almacenar números comprendidos entre -128 y 127. Ocupa 1 byte de memoria.

**short**

El tipo de dato short es un tipo de dato numérico y entero, se utiliza para almacenar números comprendidos entre -32768 y 32767. Ocupa 2 bytes de memoria.

**int**

El tipo de dato int es un tipo de dato numérico y entero, se utiliza para almacenar números comprendidos entre -2.147.483.648 y 2.147.483.647. Ocupa 4 bytes de memoria.

**long**

El tipo de dato long es un tipo de dato numérico y entero, se utiliza para almacenar números comprendidos entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807. Ocupa 8 bytes de memoria.

**float**

El tipo de dato float es un tipo de dato numérico y de punto flotante, se utiliza para almacenar números comprendidos entre -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38. Ocupa 4 bytes de memoria, y maneja entre 6 y 7 cifras decimales.

**double**

El tipo de dato float es un tipo de dato numérico y de punto flotante, se utiliza para almacenar números comprendidos de 1.79769313486232E308 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308. Ocupa 8 bytes de memoria, y maneja unas 15 cifras decimales.

**Operadores**

**Operadores Aritméticos**

Los operadores aritméticos son utilizados para realizar operaciones aritméticas, estos son la suma (+), resta (-), multiplicación (\*), división (/)

int suma = 100 + 500;

int resta = 100 - 35;

int multiplicacion = 10 \* 5;

int division = 10 / 2;

También se encuentra disponible el resto de la división (%)

int resto = 10 % 3; // el resto de la división es en este caso 1

**Operadores de Asignación**

Los operadores de asignación se utilizan para asignar valores a las variables. En la tabla siguiente se presentan los operadores, junto con un ejemplo de utilización y su expresión equivalente en una forma extendida:

|  |  |  |
| --- | --- | --- |
| Operador | Utilización | Expresión equivalente |
| = | op1 = op2 | op1 = op2 |
| += | op1 += op2 | op1 = op1 + op2 |
| -= | op1 -= op2 | op1 = op1 - op2 |
| \*= | op1 \*= op2 | op1 = op1 \* op2 |
| /= | op1 /= op2 | op1 = op1 / op2 |

**Operador instanceof**

El operador instanceof permite saber si un objeto pertenece o no a una determinada clase. Su utilización es la siguiente:

nombreObjeto instanceof nombreClase

Retorna true o false según el objeto pertenezca o no a la clase

**Operador condicional ?:**

El operador condicional “?:” es conocido también como inline-if. Se utiliza de la siguiente manera:

String respuesta = (numero1 > numero2) ? “SI” : “NO”;

El operador evalúa expresión booleana “numero1 > numero2” y retorna “SI” en caso afirmativo y “NO” en otro caso.

**Operadores incrementales y decrementales**

Los operadores de incremento (++) y decremento (--) se utilizan para sumar o restar una unidad de una variable determinada.

Su forma de uso es la siguiente:

int numero = 100;

numero++; // número toma el valor 101

numero++; // número toma el valor 102

numero--; // número toma el valor 101

**Operadores relacionales**

Los operadores relacionales se utilizan para realizar comparaciones de igualdad, desigualdad y relación de menor o mayor. El resultado de estos operadores es siempre un valor booleano (true o false).

A continuación, se presentan los posibles operadores junto con su utilización:

|  |  |  |
| --- | --- | --- |
| Operador | Utilización | El resultado es true |
| > | op1 > op2 | Si op1 es mayor que op2 |
| >= | op1 >= op2 | Si op1 es mayor igual que op2 |
| < | op1 < op2 | Si op1 es menor que op2 |
| <= | op1 <= op2 | Si op1 es menor igual que op2 |
| == | op1 == op2 | Si op1 y op2 son iguales |
| != | op1 != op2 | Si op1 y op2 son diferentes |

**Operadores lógicos**

Los operadores lógicos se utilizan para construir expresiones lógicas, combinando valores lógicos (true y/o false). En ciertos casos el segundo operando no se evalúa porque no resulta necesario.

A continuación, se presentan los posibles operadores junto con su utilización:

|  |  |  |
| --- | --- | --- |
| Operador | Utilización | Resultado |
| && | op1 && op2 | True si op1 y op2 son true. Si op1 es false ya no se evalúa op2 |
| || | op1 || op2 | True si op1 u op2 son true. Si op1 es true ya no se evalúa op2 |
| ! | !op | True si op es false y false si op es true |
| & | op1 & op2 | True si op1 y op2 son true. Siempre se evalúa op2 |
| | | op1 | op2 | True si op1 u op2 son true. Siempre se evalúa op2 |

**Operador concatenación de caracteres**

El operador “+” se utilizan para realizar sumas, pero también está definido para poder concatenar cadenas de caracteres.

Su utilización es la siguiente:

String respuesta = "Se han comprado " + variableCantidad + " unidades";

**Operadores aplicables a bits**

Los operadores aplicables a bits se utilizan para manipular a nivel bits un valor, donde se pueden realizar operaciones lógicas como también desplazamientos de bits a izquierda / derecha.

A continuación, se presentan los posibles operadores junto con su utilización:

|  |  |  |
| --- | --- | --- |
| Operador | Utilización | Resultado |
| >> | op1 >> op2 | Desplaza los bits de op1 a la derecha una distancia op2 |
| << | op1 << op2 | Desplaza los bits de op1 a la izquierda una distancia op2 |
| & | op1 & op2 | Operador AND a nivel de bits |
| | | op1 | op2 | Operador OR a nivel de bits |
| ^ | op1 ^ op2 | Operador XOR a nivel de bits (1 si sólo uno de los operandos es 1) |
| ~ | ~op2 | Operador complemento (invierte el valor de cada bit) |

**Clasificación**

Los operadores pueden ser clasificados según la cantidad de operandos que utilicen. Las categorías son las siguientes:

**Operadores Unarios**

Son aquellos operadores que necesitan un único operando para realizar la operación, por ejemplo, el operador incremento (++) o el operador negación (!)

**Operadores Binarios**

Son aquellos operadores que necesitan dos operandos para realizar la operación, por ejemplo, el operador suma (+) o el operador AND (&&)

**Operadores Ternarios**

Son aquellos operadores que necesitan tres operandos para realizar la operación, el único operador ternario que posee Java es “?”:

**Estructuras de control de flujo**

**Introducción**

Java igual que C y C++ utiliza las estructuras de control if-else, while, do-while, y como selector de opciones la estructura switch.

Las estructuras de control se rigen en su ejecución a partir de una condición del tipo boolean, es decir, el resultado de operadores lógicos o el resultado de un método que retorne un valor de verdad, verdadero o falso.

**Bifurcacion if-else**

La bifurcación if-else es la más simple de las estructuras:

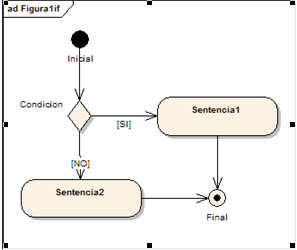
    if(expresionBooleana){

            sentencias1;

        }else{

            sentencias2;

        }



**Bifurcacion if-else-if-else**

En su forma genérica puede escribirse como:

if(expresionBooleana){

            sentencias1;

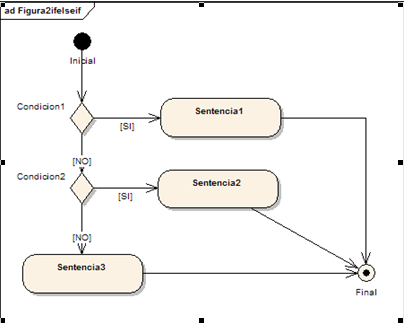
        }else if(expresionBooleana2){

            sentencias2;

        }else{

            sentencias3;

        }



**Bifurcacion switch**

El switch es clasificado como selector de sentencias de ejecución evaluando una expresión integral.

switch(expresionIntegral){

        case valor1: sentencia1; break;

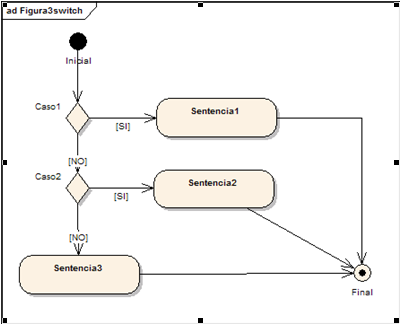
        case valor2: sentencia2; break;

        case valor3: sentencia3; break;

        default: sentencia4; break;

        }

La ejecución se resuelve evaluando la expresión integral y se busca desde la primera expresión hacia la expresión por defecto, cual etiqueta de evaluación de caso resuelve ser igual en valor, al valor de la expresión integral. En caso de no ser ninguna de las etiquetas igual al valor de la expresión se ejecutan las sentencias de la opción default. En caso de no agregar la sentencia break, se ejecutará de continuo una etiqueta tras otra hasta encontrar un break o salir del alcance definido por el switch.



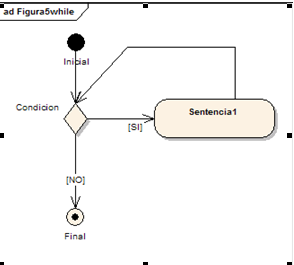
**Bucle while**

Las sentencias de bucles, son clasificadas como sentencias de iteración. Las sentencias serán ejecutadas en la medida en que sea verdadera la expresión booleana, solo dejarán de ejecutarse cuando esta expresión sea falsa. La expresión puede nunca ser verdadera, en ese caso nunca se ejecutarán las sentencias que están dentro del ciclo.

while(expresionBooleana){

sentencias;

 }



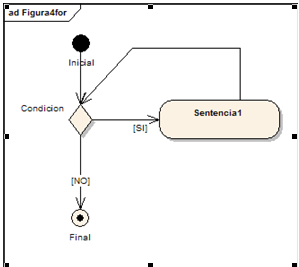
**Bucle for**

El bucle for permite en un principio una inicialización, luego itera comenzando con una evaluación de la expresión booleana y por último realiza algún tipo de paso a próximo, o reducción de la complejidad del algoritmo, en camino hacia convertir la expresión booleana en falso. Se ejecutan las sentencias, mientras la expresión booleana se mantenga en un estado de verdadero por cada paso.

for(inicializacion; expresionBooleana; postAccion){

        sentencias;

}



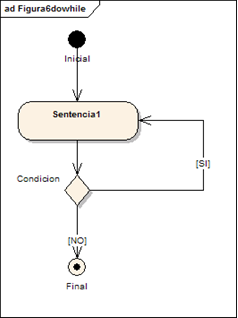
**Bucle do-while**

La única diferencia con el bucle while es que el bucle do-while ejecuta al menos una vez las sentencias, aunque desde el principio la expresión booleana sea falsa.

do{

        sentencias;

}while(expresionBooleana);



**Sentencia break**

La sentencia break puede encontrarse dentro del alcance de un ciclo. Generalmente se encuentra dentro de una bifurcación, tal que si una condición booleana se cumple se ejecutará el break y saldrá del alcance del ciclo.

Si bien se escribe de la misma forma que el break del switch, su funcionamiento es distinto dado que no salta al siguiente paso, sino que corta la ejecución del ciclo, sin importar el estado de verdad o falsedad de la expresión booleana, y lo deja sin efecto.

**Sentencia continue**

La sentencia continue puede utilizarse en ciclos, y permite la omisión de la ejecución de las sentencias definidas justo luego de la sentencia, dado que salta la ejecución hasta la próxima iteración. Generalmente se encuentra dentro de una bifurcación, de forma tal que una expresión booleana permita saltar las sentencias siguientes, hasta la próxima iteración del ciclo.

Debe tomarse muy en cuenta que antes de la ejecución de un continue debe haberse logrado reducir la complejidad del algoritmo, ya que al saltar a la próxima iteración, si la expresión booleana sigue en la misma condición y no se ha reducido la complejidad, puede generarse un ciclo infinito de ejecución.

**Comentarios**

**Tipos de comentarios**

Existen tres tipos de comentarios. Los comentarios de línea, de bloque y los javadoc.

**Comentarios de Línea**

Los comentarios de línea se utilizan para comentar en una sola línea, una sentencia, explicar la definición de una variable, o un paso a paso del código que se está observando. Un ejemplo de comentario de línea es el siguiente:

int num = 5; // este comentario me permite decir que esta es una cota del sistema.

**Comentarios de Bloque**

Los comentarios de bloque permiten comentar un bloque de código o simplemente agregar varias líneas de texto para explicar algo que solo debe ser visto en el momento de la programación. El modo de utilizarlo es el siguiente:

/\*

        En este comentario se puede explicar varias cosas.

        Algunas condiciones que se deben cumplir para continuar con la ejecución del código.

        Algún código alternativo en caso de estar testeando alguna versión, o modificando el programa.

        O simplemente dejar alguna advertencia sobre el código que se está observando.

\*/

**Comentarios Javadoc**

Los comentarios del tipo javadoc permiten documentar de una forma más útil, más publica nuestro programa. Pueden ir en cualquier sector de las clases, los packages, los atributos, los métodos. Una vez que esta documentación se ha escrito explicando o definiendo una clase un paquete o los atributos o métodos, con la ejecución de la herramienta javadoc se obtiene un sitio con la api del programa, donde se encontrarán las explicaciones que se fueron agregando en las clases. Todos estos comentarios son vinculados como en la api de java standard.

**Caracteres especiales**

La representación de caracteres especiales como el salto de línea o la tabulación, se logran a partir de la barra (\). Los caracteres especiales más utilizados son:

* \n Nueva línea
* \t Tabulador
* \’ Comilla simple
* \” Comilla doble