**JAVA**

Que es JAVA?

Java es un lenguaje de programación de propósitos generales. Podemos usar Java para desarrollar el mismo tipo de aplicaciones que programamos con otros lenguajes como C o Pascal.

Habitualmente, tendemos a asociar el término “Java” al desarrollo de páginas de Internet porque permite “incrustar” programas dentro de las páginas Web para que sean ejecutados en el navegador del usuario. Estos son los famosos Applets, que fueron muy promocionados durante los años noventa pero que hoy en día son obsoletos y, prácticamente, quedaron en desuso.

Tampoco debemos confundir Java con JavaScript. El primero es el lenguaje de programación. El segundo es un lenguaje de scripting que permite agregar cierta funcionalidad dinámica en las páginas Web.

No obstante, podemos utilizar Java para desarrollar páginas Web. La tecnología Java que permite construir este tipo de aplicaciones está basada en el desarrollo de Servlets, que se conoce como JEE (Java Enterprise Edition).

JEE es un conjunto de bibliotecas que permiten desarrollar “aplicaciones empresariales” con Java. Es decir que para programar con JEE primero debemos conocer el lenguaje de programación Java.

Java, como lenguaje de programación, se caracteriza por dos puntos bien definidos:

• Es totalmente orientado a objetos.

• La sintaxis del lenguaje es casi idéntica a la del lenguaje C++.

Más allá de esto, debemos mencionar que incluye una biblioteca muy extensa (árbol de clases) que provee funcionalidad para casi todo lo que el programador pueda necesitar. Esto abarca desde manejo de cadenas de caracteres (strings) hasta Sockets (redes, comunicaciones), interfaz gráfica, etcétera.

¿Que es la JVM?

Los programas Java son ejecutados dentro de una máquina virtual comúnmente llamada JVM (Java Virtual Machine) o también JRE (Java Runtime Enviroment). Esto significa que para poder correr cualquier programa Java previamente debemos tener instalado el JRE. Muchos sistemas operativos traen el JRE preinstalado. Otros (como Windows) no lo traen y es necesario instalarlo manualmente.

El JRE define un entorno único y homogéneo sobre el cual se ejecutarán los programas Java. Como existen versiones disponibles JRE para prácticamente todos los sistemas operativos y todas las arquitecturas de hardware, decimos que Java es un lenguaje multiplataforma.

Dicho de otro modo: un mismo programa Java puede correr en una PC con Windows, en una PC con Linux, en una Mac, en un equipo Sun con Solaris, etc. Solo debemos tener instalado el JRE correspondiente.

Dado que Java establece un estándar las diferentes empresas de tecnología están habilitadas para proveer sus propias implementaciones de JRE, obviamente, todos compatibles entre sí. Esto significa que existe un JRE desarrollado por Oracle, otro desarrollado por IBM, etcétera.

Si además de ejecutar aplicaciones Java queremos poderlas programar debemos instalar el JDK (el compilador).

¿Cómo maneja la memoria?

Cada vez que creamos un objeto estamos tomando memoria dinámicamente. El objeto funciona como un puntero que apunta a esta memoria dinámica. En lenguajes como C o Pascal, luego de utilizar memoria dinámica, tenemos que liberarla para que otros procesos la puedan usar.

En Java esto no es responsabilidad del programador. Dentro de la máquina virtual, existe un proceso que se ocupa de buscar y liberar la memoria que queda desreferenciada.

Definición de variables

Podemos definir variables en cualquier parte del código simplemente indicando el tipo de

datos y el nombre de la variable (identificador).

Identificadores válidos son:

fecha

iFecha

fechaNacimiento

fecha\_nacimiento

fecha3

\_fecha

Identificadores NO válidos son:

3fecha

fecha-nacimiento

fecha+nacimiento

-fecha

Comentarios en el código

Java soporta comentarios in-line (de una sola línea) y comentarios de varias líneas.

Comentarios de una sola línea:

// esto es una linea de codigo comentada

Java admite los mismos tipos de co-

mentarios que C: comentarios "en

línea" que comienzan con // y co-

mentarios en bloque delimitados por

/\* y \*/ .

Comentarios de más de una línea:

/\*

Estas son varias

lineas de codigo

comentadas

\*/

Estructuras de decisión

En Java disponemos de tres estructuras de decisión o estructuras condicionales:

Decisión simple: if

Decisión múltiple: switch

Decisión in-line: a>b ? "a es Mayor" : "a es Menor"

Comenzaremos analizando la sentencia if cuya estructura es la siguiente:

if( condicion ) {

accion1;

} else {

accion2;

}

La decisión múltiple switch tiene la siguiente estructura:

switch(variableEntera) {

case valor1:

accionA;

accionB;

break;

case valor2:

accionX;

accionY;

break;

default:

masAcciones;

}

Dependiendo del valor de variableEntera , el programa ingresará por el case , cuyo valor coincide con el de la variable. Se ejecutarán todas las acciones desde ese punto hasta el final, salvo que se encuentre una sentencia break que llevará al control del programa hasta la llave que cierra el switch. Por este motivo, es muy importante recordar poner siempre el break al finalizar cada case.

En caso de que el usuario haya ingresado un valor para el cual no hemos definido ningún case entonces el programa ingresará por default.

Notemos también que utilizamos la sentencia break para finalizar cada case. Esto es muy importante ya que si no la utilizamos el programa, luego de entrar al case correspondiente, seguirá secuencialmente ejecutando todas las sentencias posteriores.

Estructuras iterativas

Disponemos de tres estructuras iterativas: el while , el do-while y el for .

Nuevamente, para aquellos que conocen lenguaje C estas instrucciones son idénticas.

Comencemos por analizar el uso del while cuya estructura es la siguiente:

while(condicion) {

accion1;

accion2;

}

El ciclo itera mientras condición resulte verdadera.

Vemos que el ciclo while itera mientras que el valor de i sea menor o igual que el valor de n (que fue ingresado por teclado). Por cada iteración mostramos el valor de la variable i y luego la incrementamos.

Analicemos el ciclo do-while cuya estructura es la siguiente:

do

{

accion1;

accion2;

}

while( condicion );

Este ciclo también itera mientras se verifique la condición, pero a diferencia del ciclo anterior en este caso la entrada al ciclo no está condicionada; por lo tanto, las acciones encerradas entre el do y el while se ejecutarán al menos una vez.

Por último, veremos el ciclo for cuya estructura es la siguiente:

for( inicializacion; condicion; incremento )

{

accion1;

accion2;

}

Este ciclo tiene tres secciones separadas por ; (punto y coma). En la primera sección, se define e inicializa una variable entera que llamaremos variable de control. En la segunda sección, se especifica una condición lógica que (frecuentemente) estará en función de esta variable. En la tercera sección, se define el incremento de la variable de control.

Definición de constantes

Las constantes se definen fuera de los métodos utilizando el modificador final.

Habitualmente, se las define como públicas y estáticas ( public, static ). Más adelante, explicaremos el significado de public, static y “método”.

Arrays

En Java los arrays comienzan siempre desde cero.

Un array es un conjunto de variables del mismo tipo cuyas direcciones de memoria son contiguas. Esto permite definir un nombre para el array (conjunto de variables) y acceder a cada elemento del conjunto (a cada variable) a través del nombre común (nombre del array) más un subíndice que especifica la posición relativa del elemento al que queremos acceder.

En Java los arrays comienzan siempre desde cero y se defi nen de la siguiente manera:

// define un array de 10 elementos enteros numerados de 0 a 9

int arr[] = new int[10];

También podemos construir un array de n elementos, siendo n una variable.

int n = 10;

int arr[] = new int[n];

Debe quedar claro que el array es estático. Una vez definido su tamaño este será fijo. No se pueden agregar ni eliminar elementos en un array.

Para acceder a un elemento del array, lo hacemos a través de un subíndice.

// asigno el numero 123 en la posicion 5 del array arr

arr[5] = 123;

Si conocemos de antemano los valores que vamos a almacenar en el array entonces podemos definirlo “por extensión”. Esto crea el array con la dimensión necesaria para contener el conjunto de valores y asigna cada elemento del conjunto en la posición relativa del array.

// creo un array de Strings con los nombres de los Beatles

String beatles[] = { "John", "Paul", "George", "Ringo" };

En Java los arrays son objetos y tienen un atributo length que indica su dimensión.

// imprime en consola cuantos son los Beatles

System.out.println("Los Beatles son: "+ beatles.length);

Notemos también que no es lo mismo “definir” un array que “crear (o instanciar)” el array.

// definimos un array de Strings (aun sin dimensionar)

String arr[];

// creamos (instanciamos) el array

arr = new String[10];

O bien

// definimos e instanciamos el array de 10 Strings

String arr[] = new String[10]

Matrices

Una matriz es un array de dos dimensiones. Se definen de la siguiente manera:

// define una matriz de enteros de 3 filas por 4 columnas

int mat[][]=new int[3][4];

Literales de cadenas de caracteres

Una cadena de caracteres literal se representa encerrada entre comillas dobles, por ejemplo: "Esto es una cadena" . En cambio, un carácter literal se representa encerrado entre comillas simples, por ejemplo: 'A' .

En Java las cadenas son tratadas como objetos, por lo tanto, "Esto es una cadena" es un objeto y podemos invocar sus métodos como veremos a continuación:

// imprime ESTO ES UNA CADENA (en mayusculas)

System.out.println( "Esto es una cadena".toUpperCase() );

En cambio, los caracteres son valores numéricos enteros. Por ejemplo, 'A' es, en realidad, el valor 65 ya que este es el código ASCII de dicho carácter.

Notemos además que no es lo mismo "A" que 'A' . El primero es una cadena de caracteres que contiene un único carácter; es un objeto. El segundo es un char ; un valor numérico.

Tratamiento de Strings

Como vimos anteriormente, las cadenas de caracteres son tratadas como objetos porque String no es un tipo de datos simple. String es una clase.

Aún no hemos explicado nada sobre clases y objetos y tampoco explicaremos nada ahora ya que este tema se tratará en detalle en el siguiente capítulo.

Sin embargo, el lector ya habrá llegado a la conclusión de que un objeto es una variable que además de contener información contiene los métodos (o funciones) necesarios para manipular esta información. Agreguemos también que las clases definen los tipos de datos de los objetos.

Con esta breve e informal definición, podremos estudiar algunos casos de manejo de cadenas de caracteres sin necesidad de tratar en detalle el tema de objetos que es ajeno a este capítulo.

Acceso a los caracteres de un String

Una cadena representa una secuencia finita de cero o más caracteres numerados a partir

de cero. Es decir que la cadena "Hola" tiene 4 caracteres numerados entre 0 y 3.

El método charAt retorna al carácter (tipo char ) ubicado en una posición determinada. El método length retorna la cantidad de caracteres que tiene la cadena.

No debemos confundir el atributo length de los arrays con el método length de los strings. En el caso de los arrays, por tratarse de un atributo se lo utiliza sin paréntesis.

En cambio, en el caso de los strings está implementado como un método, por lo tanto, siempre debe invocarse con paréntesis. Veamos el siguiente ejemplo:

char c[] = { 'H', 'o', 'l', 'a' };

System.out.println( c.length );

String s = "Hola";

System.out.println( s.length() );

Mayúsculas y minúsculas

Recordemos que s es un objeto. Contiene información (la cadena en sí misma) y los métodos necesarios para manipularla. Entre otros, los métodos toUpperCase y toLowerCase que utilizamos en este ejemplo para pasar la cadena original a mayúsculas y a minúsculas respectivamente.

Ocurrencias de caracteres

El método indexOf retorna la posición de la primera ocurrencia de un carácter dentro del string. Si la cadena no contiene ese carácter entonces retorna un valor negativo.

Análogamente, el método lastIndexOf retorna la posición de la última ocurrencia del carácter dentro del string o un valor negativo en caso de que el carácter no esté contenido dentro de la cadena.

Subcadenas

El método substring puede invocarse con dos argumentos o con un único argumento.

Si lo invocamos con dos argumentos, estaremos indicando las posiciones desde (inclusive) y hasta (no inclusive) que delimitarán la subcadena que queremos extraer. En cambio, si lo invocamos con un solo argumento estaremos indicando que la subcadena a extraer comienza en la posición especificada (inclusive) y se extenderá hasta el final del string.

Decimos que un método está “sobrecargado” cuando podemos invocarlo con diferentes cantidades y/o diferentes tipos de argumentos. Este es el caso del método substring.

“Sobrecarga de métodos” es uno de los temas que estudiaremos en el capítulo de programación orientada a objetos.

Prefijos y sufijos

Con los métodos startWith y endsWith, podemos verificar muy fácilmente si una

cadena comienza con un determinado prefijo o termina con algún sufijo.

Posición de un substring dentro de la cadena

Los métodos indexOf y lastIndexOf están sobrecargados de forma tal que permiten detectar la primera y la última ocurrencia (respectivamente) de un substring dentro de la cadena en cuestión.

Concatenar cadenas

Para concatenar cadenas podemos utilizar el operador + como se muestra a continuación:

String x = "";

x = x + "Hola ";

x = x + "Que tal?";

System.out.println(x); // imprime "Hola Que tal?"

Si bien lo anterior funciona bien no es la opción más eficiente ya que cada concatenación implica instanciar una nueva cadena y descartar la anterior.

Mucho más eficiente será utilizar la clase StringBuffer .

La clase StringBuffer

Esta clase representa a un string cuyo contenido puede variar (mutable). Provee métodos a través de los cuales podemos insertar nuevos caracteres, eliminar algunos o todos y cambiar los caracteres contenidos en las diferentes posiciones del string.

El compilador utiliza un string buffer para resolver la implementación del operador de concatenación + . Es decir que, en el ejemplo anterior, se utilizará una instancia de StringBuffer de la siguiente manera:

String x = new StringBuffer().append("Hola ")

.append("Que Tal?")

.toString();

La diferencia de rendimiento entre utilizar el operador + y la clase StringBuffer para concatenar cadenas es abismal y a continuación lo demostraremos.

Las operaciones sobre StringBuffer son sincronizadas (ver Capítulo 6). La clase StringBuilder provee la misma funcionalidad y los mismos métodos, pero sin sincronización.

Conversión entre números y cadenas

Java provee clases para brindar la funcionalidad que los tipos de datos primitivos ( int , double , char , etc) no pueden proveer (justamente) por ser solo tipos de datos primitivos. A estas clases se las suele denominar wrappers (envoltorios) y permiten, entre otras cosas, realizar conversiones entre cadenas y números, obtener expresiones numéricas en diferentes bases, etcétera.

Representación numérica en diferentes bases

Java, igual que C, permite expresar valores enteros en base 8 y en base 16. Para representar en entero en base 16, debemos anteponerle el prefijo 0x (léase “cero equis”).

int i = 0x24ACF;

// en decimal es 150223

System.out.println(i); // imprime 150223

Para expresar enteros en base 8, debemos anteponerles el prefijo 0 (léase cero).

int j = 0537;

// en decimal es 351

System.out.println(j); // imprime 351

Utilizando la clase Integer podemos obtener la representación binaria, octal, hexadecimal y en cualquier base numérica de un entero dado. Esto lo veremos en el siguiente programa.

La clase StringTokenizer

La funcionalidad de esta clase la explicaremos sobre el siguiente ejemplo.

Sea la cadena s definida en la siguiente línea de código:

String s = "Juan|Marcos|Carlos|Matias";

Si consideramos como separador al carácter | (léase “carácter pipe”) entonces llamaremos token a las subcadenas encerradas entre las ocurrencias de dicho carácter y a las subcadenas encerradas entre este y el inicio o el fi n de la cadena s .

Para hacerlo más simple, el conjunto de tokens que surgen de la cadena s considerando como separador al carácter | es el siguiente:

tokens = { Juan, Marcos, Carlos, Matias }

Pero si en lugar de tomar como separador al carácter | consideramos como separador al carácter a sobre la misma cadena s el conjunto de tokens será:

tokens = { Ju, n|M, rcos|C, rlos|M, ti, s };

Usar expresiones regulares para particionar una cadena

La clase String provee el método split que permite particionar una cadena a partir de una expresión regular. Si el lector tiene conocimientos sobre expresiones regulares, entonces este método le resultará más útil que la clase StringTokenizer .

Comparación de cadenas

En el lenguaje Java (al igual que en C), no existe un tipo de datos primitivo para representar cadenas de caracteres. Las cadenas se representan como objetos de la clase String .

Ahora bien, la clase String tiene tanta funcionalidad y (yo diría) tantos “privilegios” que muchas veces se la puede tratar casi como si fuera un tipo de datos primitivo.

Informalmente, diremos que “un objeto es una variable cuyo tipo de datos es una clase”.

Así, en el siguiente código:

String s = "Hola a todos !";

s resulta ser un objeto (una variable) cuyo tipo de datos es String . Es decir: un objeto de la clase String o (también podríamos decir) una instancia de esta clase.

Los objetos son, en realidad, referencias (punteros) que indican la dirección física de memoria en donde reside la información que contienen. Por este motivo, no podemos utilizar el operador de comparación == (igual igual) para comparar objetos, porque lo que estaremos comparando serán direcciones de memoria, no contenidos. Como las cadenas son objetos, comparar cadenas con este operador de comparación sería un error.

Lo correcto será comparar las cadenas utilizando el método equals . Este método compara los contenidos y retorna true o false según estos sean iguales o no.

Operadores

Veremos aquí los diferentes tipos de operadores soportados en Java. No nos preocuparemos por buscar ejemplos para mostrar su uso ya que los mismos se irán aplicando y explicando a medida que avancemos en los capítulos de este libro.

Operadores aritméticos

La siguiente tabla resume los principales operadores aritméticos del lenguaje.

|  |  |
| --- | --- |
| Operador | Descripción |
| + | suma |
| - | resta |
| \* | multiplicación |
| / | división |
| % | módulo |
| += | acumulador |
| -= | restador |
| \*= | multiplicador |
| /= | divisor |

Operadores lógicos

La siguiente tabla resume los operadores lógicos disponibles en Java.

|  |  |
| --- | --- |
| Operador | Descripción |
| && | and |
| || | or |
| ! | not |

Operadores relacionales

La siguiente tabla resume los operadores relacionales provistos en Java.

|  |  |
| --- | --- |
| Operador | Descripción |
| == | igual |
| != | distinto (no igual) |
| > | mayor que |
| < | menor que |
| >= | mayor o igual que |
| <= | menor o igual que |

Operadores lógicos de bit

|  |  |
| --- | --- |
| Operador | Descripción |
| & | and binario |
| | | or binario |

Operadores de desplazamiento de bit

|  |  |
| --- | --- |
| Operador | Descripción |
| << | desplazamiento a izquierda |
| >> | desplazamiento a derecha |
| >>> | desplazamiento a derecha incluyendo bit de signo |

Estructuras dinámicas

Las estructuras dinámicas permiten almacenar una cantidad variable de datos. Pueden crecer o decrecer según sea necesario incrementando o decrementando la cantidad de memoria que ocupan.

En general la estructuras dinámicas se forman encadenando unidades de información llamadas “nodo”. Una estructura dinámica se compone de un conjunto de nodos enlazados entre sí más un conjunto de operaciones asociadas a través de las cuales podemos manipular y acceder al conjunto de nodos y a la información que estos contienen.

El nodo

Llamamos así a la unión de un dato más una referencia (o dirección de memoria) a otro nodo.

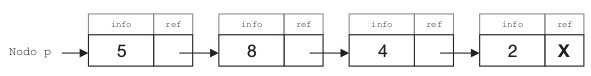
En Java podemos implementar un nodo como una clase con dos atributos: el dato (lo llamaremos info ) y la referencia a otro nodo (la llamaremos ref ). Dado que, en general, se espera que la estructura contenga datos homogéneos podemos hacer que la clase Nodo sea genérica en T de forma tal que el tipo de dato del atributo info será T mientras que el tipo de dato de la referencia al otro nodo será Nodo<T> .

En una instancia de Nodo<T> , podemos almacenar una unidad de información y la referencia a otro nodo. Si pensamos en varias instancias de Nodo<T> donde cada una tiene la referencia a la próxima podemos visualizar una lista de nodos. Una “lista enlazada”.

Lista enlazada (linked list)

La lista enlazada es la estructura dinámica básica y funciona como base para otras estructuras. Consiste en un conjunto de nodos donde cada nodo mantiene una referencia al siguiente nodo de la lista. El acceso a la lista siempre será a partir del primer nodo, por lo tanto tenemos que mantenerlo referenciado.

Recordemos que las estructuras se definen como un conjunto de nodos enlazados más un conjunto de operaciones. Las principales operaciones que podemos aplicar sobre una lista enlazada son: agregarAlFinal , agregarAlPrincipio , buscar y eliminar .



Pila (stack)

Llamamos “pila” a una estructura de datos de acceso restrictivo en la que el último elemento que se agregue siempre será el primero que se saque. A este tipo de estructuras se las llama LIFO (Last In First Out).

Las operaciones asociadas a una pila son: apilar y desapilar (o en inglés “push” y “pop”). Implementaremos una pila sobre una lista enlazada en la cual para apilar un elemento siempre lo agregaremos al principio de la lista y para desapilarlo simplemente eliminaremos el primero y lo retornaremos.

Cola (queue)

Recordemos que una cola es una estructura de acceso restrictivo en la cual el primer elemento que se agrega siempre será el primero que se saque. A este tipo de estructuras lo llamamos FIFO (First In First Out).

La cola tiene dos operaciones asociadas: encolar y desencolar .

Existen dos maneras de implementar una cola. La primera consiste en mantener dos referencias: una apuntando al primer nodo de la lista ( p ) y otra apuntando al último ( q ).

Así, cuando encolamos siempre agregamos un nodo al final de la lista y cuando desencolamos eliminamos el nodo del principio.

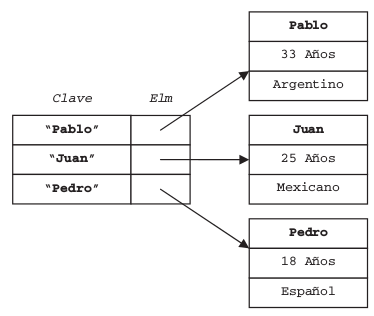
Clases LinkedList , Stack y Queue

Java provee las clases LinkedList (lista enlazada), Stack (pila) y la interface Queue (cola). Todas ubicadas en el paquete java.util .

Dado que la clase LinkedList implementa la interface Queue podemos utilizar esta interface para acotar el conjunto de métodos de la clase y limitarnos a ver sólo aquellos que tienen que ver con las restricciones que impone una cola.

Tablas de dispersión ( Hashtable )

Una tabla de dispersión ( Hashtable ) es una estructura de datos que permite mantener elementos asociados a una determinada clave (usualmente un string). Luego, el acceso a cada elemento se hará especificando el valor de esta clave (key) de acceso.

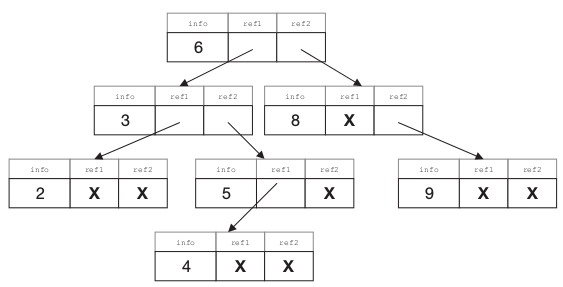


Árboles

Los árboles son estructuras de datos recursivas que se forman cuando enlazamos nodos

que tienen más de una referencia a otros nodos del mismo tipo.

En el siguiente gráfico, vemos un árbol cuyos nodos tienen dos referencias ( ref1 y ref2 ).



Dependencias y Maven

Es común en java necesitar de librerías y tener dependencias para un proyecto. Pero no solo necesitamos ciertas librerías sino específicamente una versión de cada una de ellas. Y a su vez ellas pueden necesitar otras.

Maven solventa esta problema a través del concepto de Artefacto. Un Artefacto puede verse como una librería con esteroides (aunque agrupa más conceptos). Contiene las clases propias de la librería pero además incluye toda la información necesaria para su correcta gestión (grupo, versión, dependencias etc).

Para definir un Artefacto necesitamos crear un fichero POM.xml (Proyect Object Model) que es el encargado de almacenar toda la información que hemos comentado anteriormente

La estructura del fichero puede llegar a ser muy compleja y puede llegar a depender de otros POM.

Una vez definidos correctamente todos los Artefactos que necesitamos, Maven nos provee de un Repositorio donde alojar, mantener y distribuir estos. Permitiéndonos una gestión correcta de nuestra librerías, proyectos y dependencias.

Maven nos permite crear un proyecto de Java con una sola línea de comando, respetando un formato específico. El más comun y sencillo es el siguiente:

mvn archetype:generate -DgroupId={project-packaging}  
 -DartifactId={project-name}  
 -DarchetypeArtifactId=maven-archetype-quickstart  
 -DinteractiveMode=false