

Introducción á Linguaxe Java

Introducción

Java é unha linguaxe compilada a p-code, que é executado por unha máquina virtual Java (JVM). Existe varias implementacións de compiladores e JVM java, pero a de referencia son o **Java de Oracle** (antes Sun, o creador da linguaxe) e **OpenJDK**. A distribución de Java ten dous paquetes distintos: o **JDK** e o **JRE**. O JDK (Java Development Kit) inclúe o compilador a p-code (javac) e a JVM necesaria para a execución de programas Java, mentras que o JRE so proporciona a JVM para a execución de aplicacións Java.

A linguaxe Java é unha linguaxe puramente orientada a obxectos, polo que todo o código que se escribe ten que estar definido en algunha clase, non existe a posibilidade de definir código fora das clases.

Java é unha linguaxe de herencia simple de raíz común. Unha clase so pode heredar de unha única clase base, e todas as clases proveñen da clase inicial **Object**.

Java é unha linguaxe de memoria xestionada. Non é necesario que nos preocupemos da liberación da memoria que xa non necesitamos, a JVM (Máquina Virtual Java) se encarga de facelo cando estime oportuno mediante a execución dun software denominado *recolector de lixo*.

Java é unha linguaxe fortemente tipada. É necesario definir as variables antes de utilizalas indicando o tipo de información que van a almacenar.

Java é unha linguaxe de instanciación dinámica. Todos os obxectos se manexan mediante referencias a onde están almacenados. Si comparamos dous obxectos `objetoA == objetoB`, o que estamos comparando realmente é si as dúas variables apuntan ao mesmo obxecto, non si son dous obxectos iguais...

Java é una linguaxe case-sensitive. As maiúsculas e minúsculas son letras distintas. Non é o mesmo *nome* que *Nome*, son identificadores distintos.

Identificadores

Os identificadores son os nomes que o programador asigna a variables, constantes, clases, métodos, paquetes... etc dun programa. Na linguaxe Java deben cumprir coas seguintes características:

- Deben estar formadas por letras e díxitos.
- Non poden comezar por un díxito.
- Non poden conter caracteres especiais reservados (+ - * / = % & # ! ? ^ " ' ~ \ | < > () [] { } ; : , .)
- Non poden ser unha palabra reservada Java. As palabras reservadas Java son as seguintes:

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Tipos de Datos Primitivos

Java é unha linguaxe puramente orientada a obxecto, polo que todo deberían ser obxectos que pertencen a Clases. Sin embargo, por motivos de velocidade Java dispón de varios tipos de datos que non son obxectos, se non que son **datos** que pertencen a un dos seguintes **tipos primitivos**:

Nome do tipo	Tamaño en bytes	Valor Mínimo	Valor Máximo
byte	8 bits	-128	127
short	16 bits	-32768	32767
int	32 bits	-2147483648	2147483647
long	64 bits	-9223372036854775808	9223372036854775807
float	32 bits: Formato en coma flotante IEE 754	-3.402823e38	3.402823e38
double	64 bits: Formato en coma flotante IEE 754	-1.79769313486232e308	1.79769313486232e308
boolean	Depende da VM. So admite como valores true (verdadeiro) ou false (falso)		
char	16 bits (<i>Unicode</i> , que codifica as letras con números de 8 bits)	'\u0000'	'\uffff'

Definición de Variables

Java é unha linguaxe fortemente tipada, polo que antes de utilizar unha variable é necesario indicar qué tipo de valor vai a almacenar, definindo polo tanto o espazo de memoria que utiliza e o formato en que se almacena a información. Para definir unha variable en Java se coloca o tipo seguido do identificador:

```
// Dentro da dirección de memoria identificada por idade podemos gardar un número no rango definido por int
int idade;
```

```
// Dentro da dirección de memoria identificada por letra podemos gardar un número que representa unha letra en formato UNICODE
char letra;
```

```
// Dentro da dirección de memoria identificada por l podemos gardar unha dirección dun obxecto da clase Lampada
Lampada l;
```

Como podemos observar, en Java o ; delimita o final de todas as instrucións.

É posible definir máis dunha variable do mesmo tipo cunha sola sentencia:

```
int a,b,c;
```

A clase String

A clase String é unha clase Java que serve para representar cadeas textuais. A clase String é de uso intensivo nas aplicacións Java, e son inmutables (constantes). Un obxecto String non pode cambiar de valor.

```
// Na dirección de memoria identificada por s almacenamos unha referencia ao obxecto String "HOLA"
String s="HOLA";
// Operación + sobre cadeas indica concatenación. Se xenera un novo obxecto String "HOLA Que Tal",
// e se almacena a súa referencia na posición de memoria identificada por s
s=s+" Que Tal";
```

A clase String contén numerosos métodos que permiten facer as operacións máis típicas coas cadeas de texto que se poden consultar no API de Java dispoñible en Internet.

Expresións aritméticas e lóxicas

Unha expresión é unha combinación de variables, operadores e chamadas a métodos construída de acordo á sintaxe da linguaxe e que devolve un valor.

Operadores Aritméticos	Operadores de Asignación	Operadores relacionais	Operadores Especiais
+ Suma a+b	= Asignación a=5	== Igual	++ Incremento
- Resta a-b	+= Suma e asignación (a+=3)	!= Distinto	-- Decremento
* Multiplicación a*b	-= Resta e asignación (a-=b)	> Maior que	+ (En Strings, concatena)
/ División a / b	*= Multiplicación e asignación	< Menor que	. Acceso a atributos e métodos
% Resto a % b	/= División e asignación	>= Maior ou igual que	() Agrupación de expresións
^ XOR bit a bit a ^ b	%= Módulo e asignación	<= Menor ou igual que	new Instancia unha clase
& AND bit a bit a & b		! NOT lóxico	
OR bit a bit a b		&& AND lóxico	
~ NOT de bits ~a		OR lóxico	
>> Desprazamento			
>>> Desprazamento			
<< Desprazamento			

As operacións aritméticas de desprazamento consisten en desprazar o número de bits indicados hacia a dereita ou a esquerda, insertando 0 polo outro extremo da variable, salvo >> que respeta o valor do bit de signo.

Por exemplo:

```
byte a=7; // En base 2 00000111
byte c=a >>> 1; // Se despraza 1 bit á dereita: 00000011, queda un 3
byte c=a << 1; // Se despraza 1 bit á esquerda: 00001110, queda un 14
```

As operacións de incremento/decremento poden ser preincremento (++a) ou postincremento (a++). No primeiro caso, se incrementa primeiro e logo se avalía a expresión e no segundo caso se avalía a expresión e se incrementa. Todos os operadores están suxeitos a unhas regras de precedencia que se poden alterar mediante o uso de parénteses. A orde de precedencia, de primeiro a último, é:

Operador	Notas
. [] ()	Os corchetes son para acceso a táboas (arrays)
++ -- ! ~	! es el NOT lóxico e ~ é complemento de bits
new (tipo)expr	new se utiliza para crear instancias de clases
* / %	
+ -	
<< >> >>>	Desprazamento de bits
< > <= >=	Relacionais
== !=	Igualdade
&	AND (aritmética AND entre bits)
^	OR exclusivo (aritmética XOR entre bits)
	OR inclusivo (aritmética OR entre bits)
&&	AND lóxico
	OR lóxico
? :	Abreviatura da expresión condicional "if"
= += -= *= /= %= &= ^= = <= >= >>=	Operación e Asignación

Sentencias de Control

As sentencias de control en Java son as seguintes:

Execución Condicional

```
if ( expresión condicional, avaliada a true ou false ) {  
    instrucións a executar no caso de que a condición se cumpra  
}
```

ou

```
if ( expresión condicional, avaliada a true ou false ) {  
    Instrucións a executar en caso de que a condición se cumpra  
} else {  
    Instrucións a executar en caso de que a condición se cumpra  
}
```

Esta expresión pode abreviarse do seguinte xeito:

(expresión condicional)? Instrucción cando se cumpre : instrucción cando non se cumpre;

Execucións Repetitivas

Forma 1: Se avalía a condición ANTES de comezar a repetición

```
while (condición avaliada a true ou false) {  
    instrucións a repetir mentres a condición se cumpra  
}
```

Forma 2: Se comeza a repetición ANTES de avaliar a condición

```
do {  
    instrucións a repetir mentres a condición se cumpra  
} while (condición avaliada a true ou false);
```

Forma 3: Modo abreviado da Forma 1. Se debe utilizar sobre todo para contar de xeito secuencial de modo ascendente ou descendente.

```
for (pre-instrucións; condición avaliada a true ou false; post-instrucións) {  
    instrucións a repetir mentres a condición se cumpra  
}
```

Na forma 3, se levan a cabo as pre-instrucións unha única vez antes de comezar a repetición ou avaliar a condición. Estas pre-instrucións poden ser unha única instrución ou varias separadas por comas. Logo se avalía a condición e si se cumpre se executarán as instrucións a repetir seguido das post-instrucións (que poden ser unha única instrución ou varias separadas por comas). E dicir, a estrutura:

```
int x;  
x=0;  
while(x < 20) {  
    instrucións a repetir;  
    x=x+1;  
}
```

é equivalente a esta outra

```
int x;  
for(x=0;x<20;x=x+1) {  
    instrucións a repetir:  
}
```