UNIDAD 1

Conceptos Básicos

Contenido

Contenido

Operadores
Descripción de los operadores
Prioridad entre operadores
Desbordamiento de memoria
Conversión de tipos
Tipos de conversiones
Enlace de conversión de tipos

Algoritmo

Un algoritmo informático es un **conjunto de instrucciones definidas, ordenadas y acotadas para resolver un problema o realizar una tarea**.

Las tres **partes** de un algoritmo son:

- **Input** (entrada). Información que damos al algoritmo con la que va a trabajar para ofrecer la solución esperada.
- **Proceso**. Conjunto de pasos para que, a partir de los datos de entrada, llegue a la solución de la situación.
- **Output** (salida). Resultados, a partir de la transformación de los valores de entrada durante el proceso.

Características comunes:

- **Precisos**. Objetivos, sin ambigüedad.
- **Ordenados**. Presentan una secuencia clara y precisa para poder llegar a la solución.
- **Finitos**. Contienen un número determinado de pasos.
- **Concretos**. Ofrecen una solución determinada para la situación o problema planteados.
- **Definidos**. El mismo algoritmo debe dar el mismo resultado al recibir la misma entrada.

Programa

Un **programa informático o programa de computadora** es una secuencia de instrucciones, escritas para realizar una tarea específica en un computador.

Este dispositivo requiere programas para funcionar, por lo general, ejecutando las instrucciones del programa en un procesador central.

Lenguaje de Programación

Es un lenguaje formal (con reglas gramaticales bien definidas) que proporciona al programador la capacidad de escribir (o programar) una serie de instrucciones o secuencias de órdenes en forma de algoritmos con el fin de controlar el comportamiento físico o lógico de un sistema informático.

Lenguaje de Programación

Existe una gran variedad de lenguajes de programación, adaptados a diferentes necesidades o formas de programar.

- Según el nivel de detalle.
- Según el paradigma utilizado. (declarativo, estructurado, orientado a objetos, etc.).
- Según si se dispone o no de intérprete.
- Etc.

Esto hace a unos lenguajes más apropiados que otros para resolver ciertos problemas.

Tipos de Lenguaje

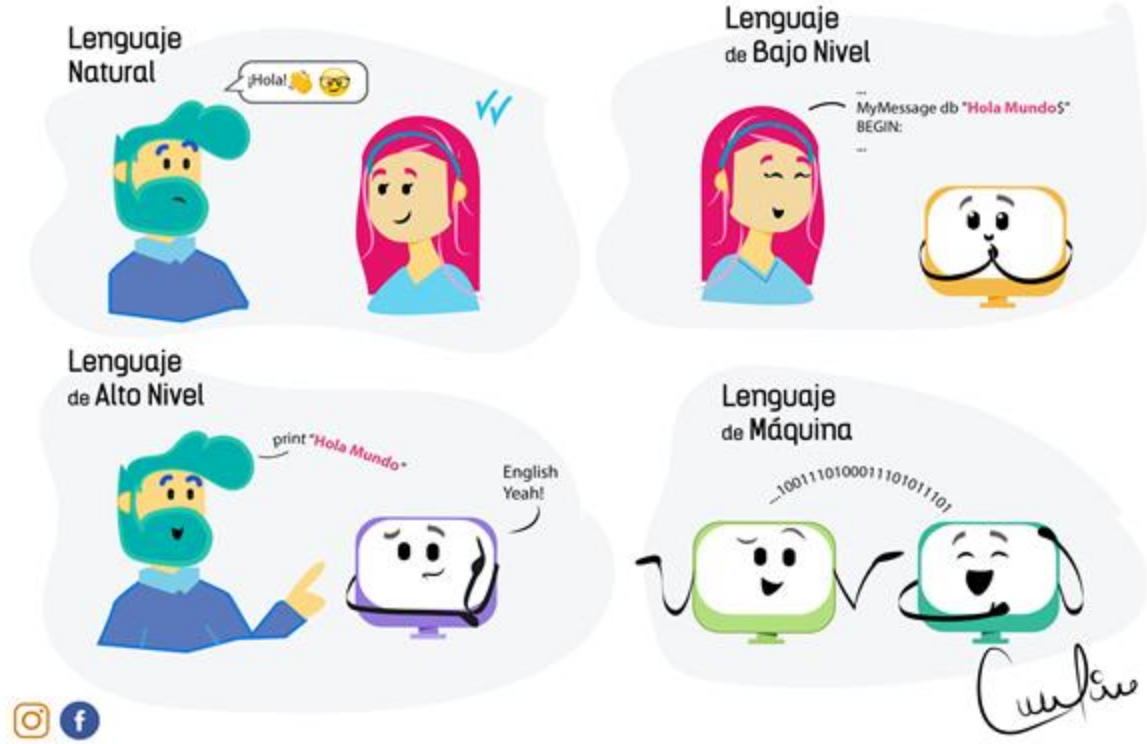
- Lenguaje de **bajo nivel**
  - Instrucciones muy cercanas a la máquina.
  - Ensamblador.
  - Rápidos y control total de recursos.
  - Difíciles de dominar.

- Lenguaje de **alto nivel**
  - se asemeja al lenguaje del ser humano.
  - Complejos y optimizados.
  - Sencillos de aprender.
  - No se tiene control de recursos.

- Lenguaje **interpretado**
  - Necesita un intérprete para poder ejecutar el programa.
  - Sencillos y multiplataforma.

- Lenguaje **compilado**
  - El código se compila previamente a ejecutarse.
  - Complejos y optimizados.
  - Dependen de la plataforma.
  - Necesitan una compilación para cada tipo de máquina y Sistema Operativo.

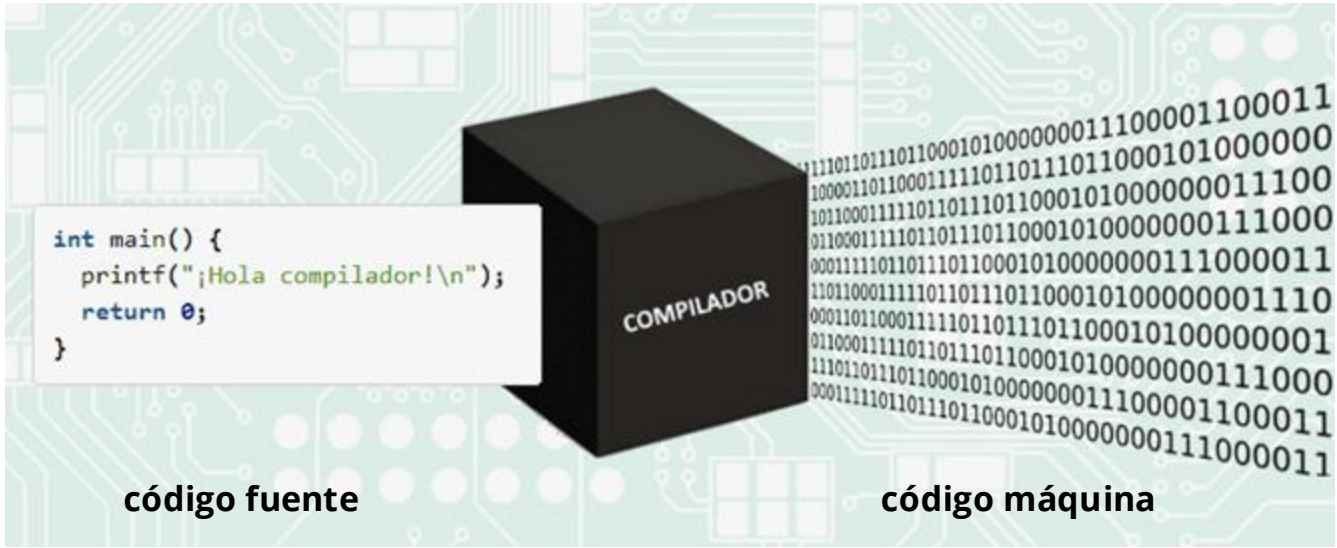
Tipos de Lenguaje



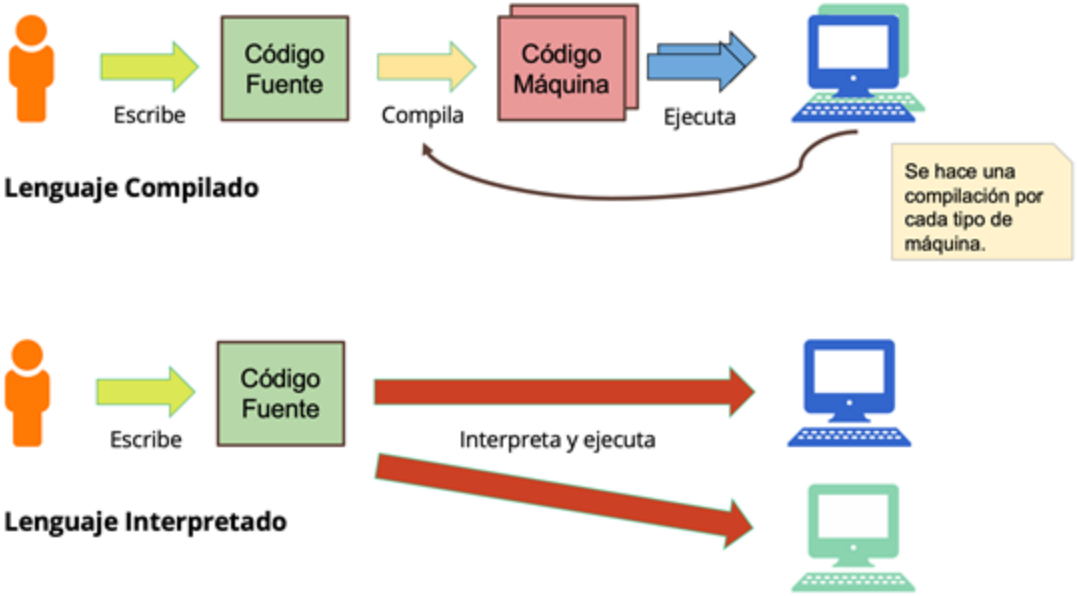
Tipos de Lenguaje

El compilador

Un compilador es un programa que traduce **código fuente** escrito en un lenguaje de alto nivel como Java, a un lenguaje legible por la máquina llamado código máquina (o código intermedio).

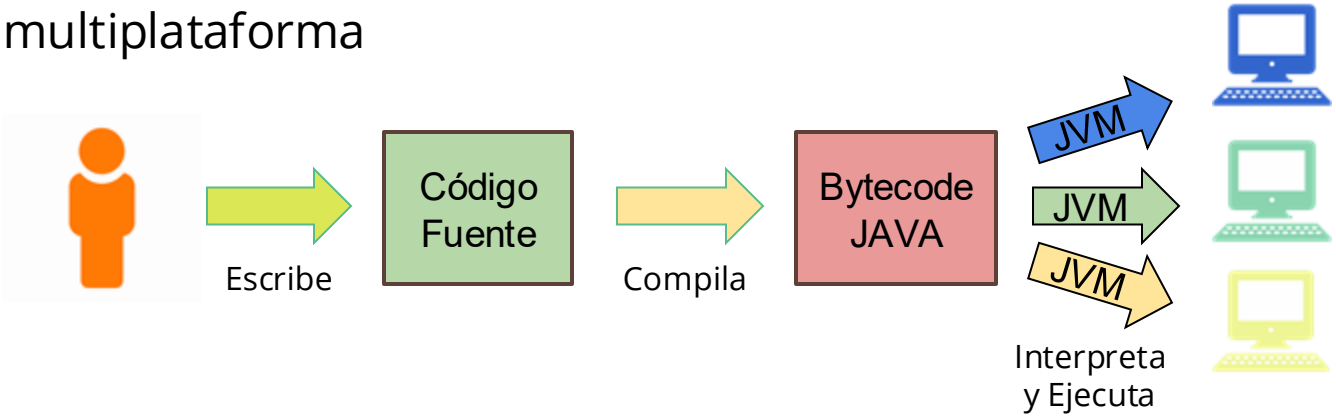


Tipos de Lenguaje



El lenguaje JAVA

- Es uno de los lenguajes con mayor utilización en el mundo.
- Posee un gran número de librerías y frameworks
- Lenguaje compilado, concurrente y orientado a objetos
- A pesar de ser compilado, posee un intérprete (JVM) que lo hace multiplataforma



Características de JAVA

Sigue el paradigma de Programación Orientada a Objetos, aunque puede usarse a través de programación estructurada.

Es multiplataforma, el mismo código se puede ejecutar en diferentes sistemas.

Pensado para el trabajo en red y en remoto.

Optimizado para el uso de la concurrencia.

Fácil de usar. Sintaxis sencilla y muy parecida a otros lenguajes de uso común como C.

Abstrae al programador del uso de la memoria del sistema. No usa punteros Posee un recolector de basura.

Herramientas para el desarrollo con JAVA

Java es un lenguaje interpretado, por lo que es necesario descargarse el intérprete Java.

Java incluye un gran número de librerías que facilitan la labor al programador, se encuentran en el JDK (Java Development Kit)

Para poder programar en Java, ayuda el uso de un Integrated Development Environment (IDE) como NetBeans

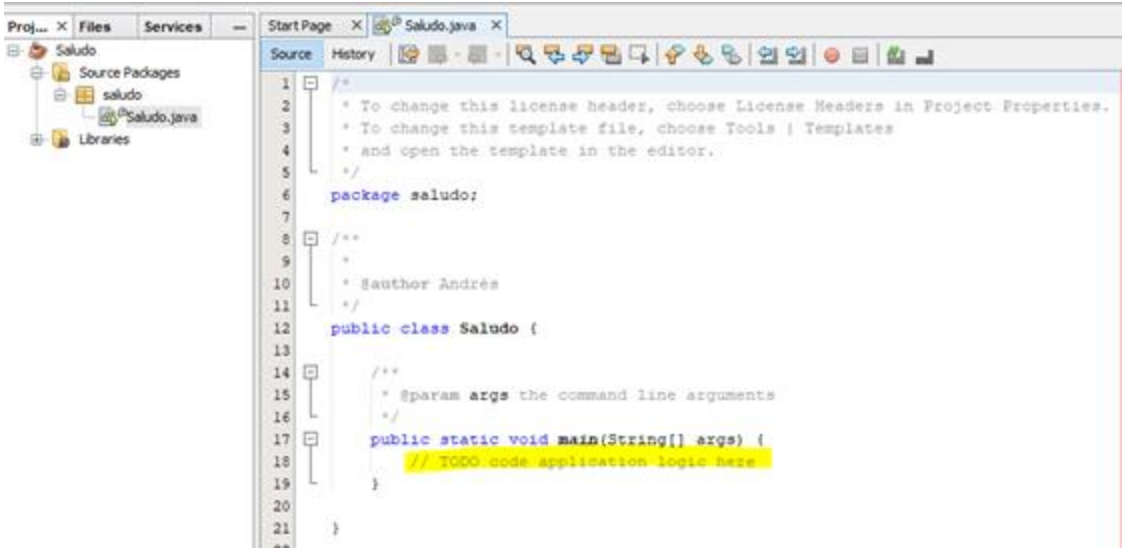
De cara a la instalación de ambas cosas, debe hacerse en el siguiente orden:

1. JDK: https://www.oracle.com/java/technologies/downloads/?er=221886#javasejdk

2. NetBeans: https://netbeans.apache.org/front/main/download/

Mi primer programa en JAVA: Hola Mundo

1. Crear el proyecto: File -> **New Project**
2. Elegir **Java with Ant** en el panel de la izquierda y **Java Application** en el de la izquierda.
3. Poner en "**Project Name**" el nombre del programa.
   Por ejemplo: "*Saludo*".

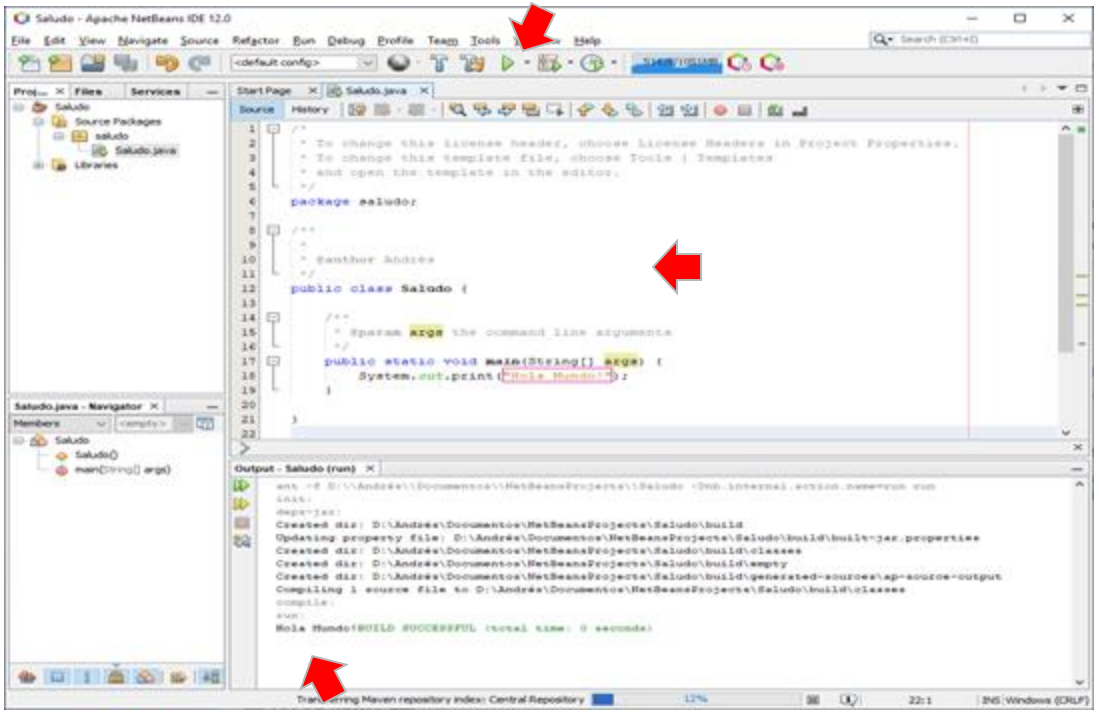


Mi primer programa en JAVA: Hola Mundo

Borra el comentario TODO y sustituye por la siguiente línea:

System.out.print(“Hola Mundo!”);

Pulsa el botón “Run”.



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, expresiones, declaraciones, o llamadas a funciones.

Si no existen sentencias específicas de selección o salto, el programa se ejecuta de forma secuencial en el mismo orden en que se ha escrito el código fuente (es el que podríamos considerar orden "natural" de ejecución).

Delimitadores

Símbolos utilizados como separadores de las distintas construcciones de un lenguaje de programación.

- **( )** **Paréntesis**: Listas de parámetros, precedencia en expresiones, expresiones para control de flujo y conversiones de tipo.
- **{ }** **Llaves**: Inicialización de arrays, bloques de código, clases, métodos y ámbitos locales.
- **[ ]** **Corchetes**: Arrays.
- **;** **Punto y coma**: Separador de sentencias.
- **,** **Coma**: Identificadores consecutivos en una declaración de variables y sentencias encadenadas dentro de una sentencia for.
- **.** **Punto**: Separador de nombres de paquetes, subpaquetes y clases; separador entre variables y métodos/miembros.

Comentarios

Los comentarios no son sentencias. Son ignorados por el compilador. No se van a ejecutar.

Lo incluye el programador para mejorar la inteligibilidad del programa.

En Java hay tres tipos de comentarios:

```
//      Comentario de una sola línea

/*      Comentario de una
                    o más líneas */

/**  Comentario de documentación, normalmente generada
     automáticamente */
```

Palabras Reservadas

Palabras que tienen un significado concreto en el lenguaje de programación, sin necesidad de que se lo asignemos nosotros.

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

No se permite la utilización de una palabra reservada para nombrar los elementos de Java

Identificadores

Un identificadores es un nombre con el que se hace referencia a:

- Una función
- Una clase
- El contenido de una variable o una constante

A la hora de definir un identificador, se debe tener en cuenta lo siguiente:

- Se forma por una secuencia de letras y dígitos (incluido _)
- No puede contener ni espacios en blanco ni caracteres especiales como +, -, *, /, ;, etc.
- El primer carácter debe ser una letra o _
- Distingue entre mayúsculas y minúsculas
- Conviene incluir nombres que indiquen lo que representan o su utilidad

Identificadores

En la regla de estilo que sigue Java, el identificador de las variables comienzan por minúscula, mientras que los nombres de las clases comienza por mayúscula. Esto permite, de un vistazo, distinguir en el código qué es cada cada uno de los identificadores usados.

Variable

Elemento del lenguaje que **almacena un valor** cualquiera de los comprendidos dentro de un conjunto y que puede cambiar a lo largo del tiempo.

A las variables se les asigna valores concretos mediante el operador de asignación (=).

temperatura = 25

donde, Identificador (o nombre) es temperatura y valor es 25

Esto quiere decir que el dato 25 se va a almacenar en memoria y se va a identificar dicho dato con el identificador temperatura. Mediante dicho identificador se podrá leer o sobreescribir dicho dato.

El rango o conjunto de valores que puede tomar una variable viene definida por su **tipo de datos**.

Variables en JAVA

Se definen de la forma:

tipo nombre

En Java es obligatorio indicar el tipo de la variable, así como inicializarla.

Ejemplos:

```
int suma = 0;
double numDecimal = 3.8;
boolean esCierto = true;
char caracterSimple = 'c';
String cadena = "Ejemplo";
```

Constante

Es un caso especial de variable, donde, no varían. Es decir, se le asigna un valor y este se mantiene constante durante toda la ejecución.

La declaración de constantes es similar a las variables pero añadiendo la palabra reservada ***final***.

final tipo nombreConstante;

Ejemplo:

```
final double PI = 3.141592;

final byte NUMERO_CARTAS = 40;
```

Tipos de datos

Todo dato manejado en Java debe tener un tipo.

El tipo puede ser:

- Explícito. Indicado por el programador
- Implícito. El compilador interpreta el tipo automáticamente

El tipo de datos sirve al compilador para:

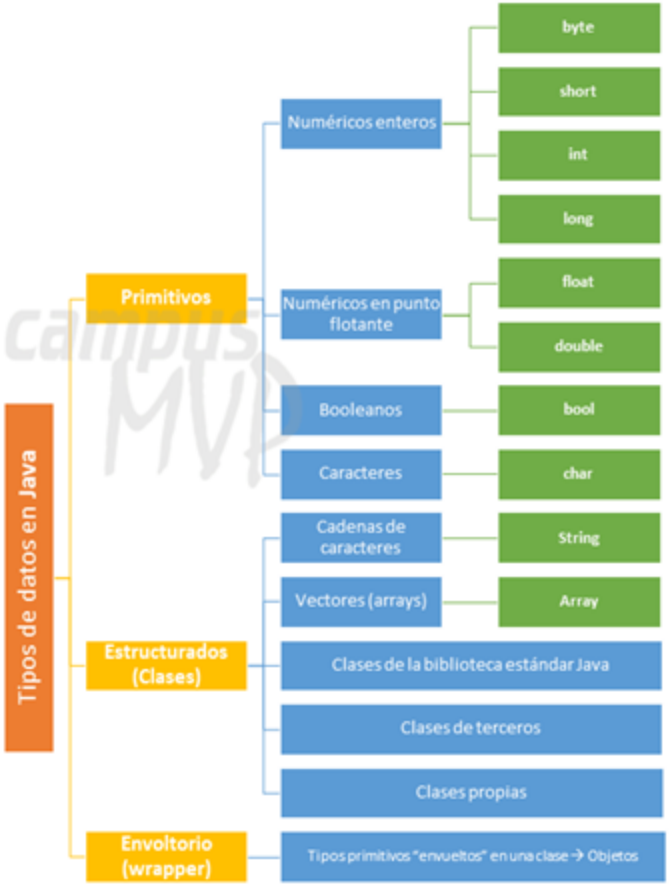
- Conocer el espacio de memoria a reservar para ese dato
- Conocer una serie de directrices a la hora de manejarlo

Tipos de datos

Primitivos: son los fundamentales o básicos. El resto de las estructuras se basan en ellos. No son objetos, no tienen métodos.

Estructurados o No primitivos (u objetos): destinados a contener múltiples valores de tipos más simples, primitivos.

Envoltorio (wrapper): son tipos de datos estructurados equivalentes a cada uno de los tipos primitivos. Añade ciertos métodos y propiedades útiles al tipo primitivo.



Rango de los tipos primitivos

VARIABLES DE TIPOS PRIMITIVOS.					
Nombre	**Tipo**	**Tamaño**	**Valor por defecto**	**Forma de inicializar**	**Rango**
Boolean	Lógico	1 bit	False	Boolean a=true	True-false
Char	Carácter	16 bits	Null	Char a='Z'	Unicode
Byte	Numero entero	8 bits	0	Byte a =0	-128 a 127
Short	Numero entero	16 bits	0	Short a =12	-32.768 a 32.767
Int	Numero entero	32 bit	0	Int a= 1250	-2.147.483.648 a 2.147.483.649
Long	Numero entero	64 bits	0	Long a= 125000	-9*10^18 a 9*10^18
Float	Numero real	32 bits	0	Float a =3.1	-3,4*10^38 a 3,4*10^38
Double	Numero real	64 bits	0	Double a = 125.2333	-1,79*10^308 a 1,79*10^308

Tipos de datos no primitivos o estructurados

Es posible declarar variables cuyo tipo no sea un tipo primitivo, sino una clase. Por ahora, pensaremos en las clases como tipos de datos complejos, hasta que las estudiemos en profundidad

A estas variables se les denomina variables de objeto y las utilizaremos para poder aprovechar las herramientas que Java proporciona.

De igual manera que las variables de tipos primitivos se inicializan por defecto, si en su declaración no se les asigna un valor, las variables de objeto se inicializan por defecto con el valor especial null, que representa que la variable se encuentra vacía.

Salida de datos por consola

- System.out.**print**(“Mensaje”); // Sin salto de línea.
- System.out.**println**(Mensaje”); // Con salto de línea

Recuerda que puedes concatenar textos usando + y combinarlo con variables.

Ejemplo:

```
String nombre = “Ana”;
int edad = 24;
System.out.println(nombre + “ tiene “ + edad + “años.”);
```

También puedes usar el carácter especial \n que equivale al retorno de carro.

```
System.out.println(“Primera línea.\nSegunda línea.”);
```

Entrada de datos por consola

Para ello se ha de usar la clase ***Scanner***.

Veamos cómo usarlo de forma simple:

- Se debe importar la clase java.util.Scanner.
- Se debe crear un objeto Scanner indicando que se va a escanear la entrada al sistema (System.in).
- Se indica que se va a leer una línea sc.nextLine().

```
package saludo;

import java.util.Scanner;

public class Saludo {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Escribe tu nombre: ");

        String nombre = sc.nextLine();

        System.out.println("Hola "+nombre);

    }

}
```

La clase Scanner

Funciona parando el programa y esperando que el usuario teclee los datos hasta que pulse la tecla ***Intro***.

Todo el contenido tecleado es interpretado y asignado a una variable. El tipo depende del método que se use.

La clase Scanner dispone de los siguientes métodos.

- sc.nextInt(); // lee un número entero (int)
- sc.nextDouble(); // lee un número real (double)
- sc.nextLine(); // lee una cadena de caracteres (texto)
- sc.next(); // lee una cadena de caracteres hasta tab, space o intro
- sc.nextBoolean(); // lee un booleano: true o false.

API DE JAVA

La **API Java** es una interfaz de programación de aplicaciones (API, por sus siglas del inglés: Application Programming Interface) provista por los creadores del lenguaje de programación Java, que da a los programadores los medios para desarrollar aplicaciones Java.

Son herramientas avanzadas que ya están integradas en JAVA y que facilitan el trabajo del desarrollador.

Ejemplos:

- Lectura/escritura de datos
- Cálculos complejos
- Manejo de errores

Paquetes

Los paquetes son el mecanismo que usa Java **para facilitar la modularidad del código**. Un paquete puede contener una o más definiciones de interfaces y clases, distribuyéndose habitualmente como un archivo.

Para utilizar los elementos de un paquete es necesario importar este en el módulo de código en curso, usando para ello la sentencia **import**.

Puedes poner paquetes dentro de otros paquete.

Ejemplo:

La clase Math se encuentra en el paquete lang que a su vez está en el paquete java.

java.lang.Math

Operadores

Carácter o grupo de caracteres especiales que actúa sobre una, dos o más variables y/o constantes para obtener un resultado.

Se pueden combinar dando lugar a expresiones, pudiendo usar paréntesis para modificar su precedencia.

Para dar un valor a una variable, se utiliza el operador de asignación =

Ejemplo:

```
int  a =  3;      //  “a” vale 3
```

Descripción de los operadores

Enlace con descripción de todos los operadores:

https://www.manualweb.net/java/operadores-asignacion-aritmeticos-java/

Pruebas

Operadores aritméticos:

Dentro de un System.out.println( ) escribe las siguientes operaciones:

- 23-3 System.out.println(23-3);
- 120+20
- 100*50
- 10/3
- 10/3.0
- 10.0/3
- 10%3

Observa las salidas. ¿ves la diferencia entre números entero y decimales?

Pruebas

Trata de realizar las operaciones anteriores pero creando una variable del tipo adecuado más pequeño posible.

Ejemplo:

- 23-3

```
byte valor1 = 23-3;

System.out.println(valor1);
```

Curiosidad

Prueba el siguiente código:

```
byte a = 2+3;
System.out.println(a);
```

No hay problema. Debería devolver 5.

Ahora vamos a poner el 2 y el 3 en dos variables, x e y respectivamente.

```
byte x = 2;
byte y = 3;
byte a = x+y;
System.out.println(a);
```

Da un error al asignar x+y a la variable a.

Esto es porque para las **operaciones aritméticas JAVA convierte el resultado en int**, y si luego lo tratamos de meter en un byte, que ocupa menos, nos da el error de que quizás se pierda el dato.

Pruebas

Operadores unitarios (incremental y decremental):

```
int x = 3;
        System.out.println(x++);
        int y = 3;
        System.out.println(++y);

// ¿Por qué uno devuelve 3 y el otro 4? ¿Cuánto valen x e y al final?

        int a = 3;
        int b = a++;
        int c = ++a;
        // ¿Cuánto valen a, b y c en este momento?
```

Pruebas

```
byte a = 4;
a += 5;

byte b = 3;
b -= 2:

byte c = 2;
c *= 5;

byte d=15;
d /= 4;

byte e=15;
e %= 4;
```

¿Cuánto vale a,b,c,d y e?

Prioridad entre operadores

Al igual que en matemáticas, existe una precedencia a la hora de realizar las operaciones. Que algunas se pueden realizar antes que otras y no en el orden en el que se leen.

Puedes consultar las tablas de prioridad que se encuentran junto a los tipos de operadores, pero resumiendo podemos decir que de mayor a menor precedencia son:

1. Unitarios: Incrementos (++) y decrementos (--)
2. Aritméticos:
    a. * / %
    b. + -
3. Relacionales: < <= > >=
4. Comparación: == !=
5. Lógicos:
    a. &&
    b. ||
6. Asignación: = += -= ....

Pruebas

Intenta calcular cuál sería el resultado. Luego pruébalo:

- 4+6/2
- 4*(6+2)
- (9-4)*(2+3)/3

- int x = 3;
- int y = 4 * x++;
- System.out.println(y);

- int x=2;
- System.out.println(3x+2);

- int a=4;
- int b=5;
- int c=a+b;
- a++;
- System.out. println(c);

- int e=34;
- e/=2;
- int f = --e;
- f*=2;
- System.out.println(f++);

Desbordamiento de memoria

Se produce cuando un dato ocupa más que el asignado según su tipo de datos.

En JAVA esto no ocurre debido a:

- Si se asigna un valor fuera de rango, el compilador avisa con un error.
- Si en tiempo de ejecución un cálculo sobrepasa los límites del rango permitido, se continúa desde 0. Como dar la vuelta al cuentakilómetros de un coche. No da error, pero no es un comportamiento que deseemos.

Para evitar esto se deben asignar los tipos apropiados o incluso hacer conversión de tipos.

Probando el desbordamiento

Vamos a probar el desbordamiento circular.

- Crea una variable tipo byte.
- Inicializa la variable al valor máximo para ese tipo.
  - rango del tipo byte: -128 a 127
- Incrementa el valor en 1 (usa el operador ++).
- Muestra por pantalla el valor.

Código: probando el desbordamiento

```
byte valor = 127;

valor++;

System.out.println(valor);
```

Conversión de tipos

Cuando declaras una variable, se le asigna un tipo. Por tanto, todo valor que vayas a poner a esa variable debe ser de ese tipo.

Si no es del mismo tipo, dará un **error**.

Cuando tenemos un valor que no es del tipo de la variable, siempre podemos intentar convertirlo al tipo que corresponde con la variable. Pero esto puede llevar a la pérdida parcial de datos.

Tipos de conversiones

- De **ensanchamiento** o **promoción**. Por ejemplo: pasar de un valor entero a un real.
  - No hay problema porque el nuevo tipo es más grande y cubre al anterior.
- De **estrechamiento** o **contracción**. Por ejemplo: pasar de un valor real a un entero.
  - No se puede hacer directamente porque el compilador da un error por pérdida de información.
  - Se debe indicar explícitamente.

Enlace de conversión de tipos

Enlace con descripción de las conversiones de tipos:

https://es.stackoverflow.com/questions/1487/guia-definitiva-de-conversi%C3%B3n-de-tipos-en-java

Pruebas

int entero = 6;

double decimal = entero;

double decimal = 6;

int entero = decimal;

Al convertir el decimal (double) en entero (int) se supone una pérdida de información (todos los decimales se perderán). Por tanto, el compilador da un error aunque como vemos, la variable “decimal” tiene un número sin decimales.

Para evitar el error de compilación, si queremos convertir un número decimal (double o float) a uno entero (int) hay que indicarlo explícitamente.

double decimal = 6.5;

int entero = (int) decimal;

En este caso no hay error y se convierte el 6.5 en 6. Por tanto, se pierde información.