Gobierno de Canarias
Servicio Canario de Empleo

Centro GASTRONÓMICO del Atlántico

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL
MINISTERIO DE TRABAJO Y ECONOMÍA SOCIAL
SERVICIO PÚBLICO DE EMPLEO ESTATAL
SEPE

Módulo 1: Programación en Java 2023

Tema 3 Tipos primitivos de datos

Tema 3

Contenido

1. Tipos Primitivos de Datos

1.1. Objetivos

- Describir los tipos de datos primitivos (numéricos, booleano y de tipo carácter) en el lenguaje de programación Java y su formato de representación.
- Escribir la declaración de constantes y variables de cualquiera de los tipos de datos primitivos.
- Presentar el concepto de conversión y los distintos tipos de conversiones de datos de tipo primitivo en Java.

Todo lenguaje de programación consta de elementos específicos que permiten realizar las operaciones básicas de la programación: tipos de datos, operadores e instrucciones o sentencias. En este apartado se introducen los distintos tipos de dato que pueden emplearse en la programación con Java. En concreto, se presentan los tipos primitivos en Java, así como las constantes y las variables.

En los capítulos sucesivos se mostrarán el resto de elementos básicos de programación, incluyendo otras estructuras de datos más complejas.

1.2. Categorías de Tipos de Datos

Los tipos de datos utilizados en Java se pueden clasificar según diferentes categorías:

- De acuerdo con el tipo de información que representan. Esta correspondencia determina los valores que un dato puede tomar y las operaciones que se pueden realizar con él. Según este punto de vista, pueden clasificarse en:

    - **Datos de tipo primitivo**: Representan un único dato simple que puede ser de tipo *char*, *byte*, *short*, *int*, *long*, *float*, *double* o *boolean*. Por ejemplo: 'a', 12345, 750.68, False,... Cada tipo de dato presenta un conjunto de valores o constantes literales.
    - **Variables referencia (Arrays, Clases, Interfaces,...)**: Se implementan mediante un nombre o referencia (puntero) que contiene la dirección en memoria de un valor o conjunto de valores (por ejemplo, un objeto creado con new).

- Según cambie su valor o no durante la ejecución del programa. En este caso, se tienen:

    - **Variables**: sirven para almacenar datos durante la ejecución del programa; el valor asociado puede cambiar varias veces durante la ejecución del programa.
    - **Constantes o variables finales**: también sirven para almacenar datos pero una vez asignado el valor, éste no puede modificarse posteriormente.

- Según su papel en el programa. Pueden ser:

    - **Variables miembros de una clase** o **atributos**: Se definen dentro de una clase, fuera de los métodos. Pueden ser de tipos primitivos o referencias y también variables o constantes.
    - **Variables locales**: Se definen dentro de un método o, en general, dentro de cualquier bloque de sentencias entre llaves {}. La variable desaparece una vez finalizada la ejecución del método o del bloque de sentencias. También pueden ser de tipos primitivos o referencias.

A continuación se describen cada uno de estos tipos de dato.

1.3. Tipos de Dato Primitivos en Java

A todo dato (constante, variable o expresión) le corresponde un tipo específico en Java. Como se ha indicado anteriormente un tipo de dato determina los valores que pueden asignarse a un dato, el formato de representación correspondiente y las operaciones que pueden realizarse con dicho dato.

En *Java casi todo es un objeto*. Existen algunas excepciones como, por ejemplo, los **tipos primitivos**, tales como *int*, *char*, ..., que no se consideran objetos y se tratan de forma especial. Java tiene un conjunto de tipos primitivos para representar datos enteros (cuatro tipos diferentes), para datos numéricos reales en coma flotante (dos tipos diferentes), para caracteres y para datos lógicos o booleanos. Cada uno de ellos tiene idéntico tamaño y comportamiento en todas las versiones de Java y para cualquier tipo de ordenador. Esto implica que no hay directivas de compilación condicionales y asegura la portabilidad de los programas a diferencia de lo que ocurre, por ejemplo, con el lenguaje de programación C.

Nota: En otros lenguajes de programación el formato y tamaño de los tipos de dato primitivos puede depender de la plataforma o sistema operativo en la que se ejecute el programa. Sin embargo Java especifica el tamaño y formato de todos los tipos de dato primitivos para que el programador no tenga que preocuparse sobre las dependencias del sistema.

Por otro lado, a partir de estos tipos primitivos de dato pueden construirse otros tipos de datos compuestos como *arrays*, *clases* e *interfaces*.

En la siguiente tabla se muestran los tipos de datos primitivos de Java con el intervalo de representación de valores que puede tomar y el tamaño en memoria correspondiente:

Tipo	Descripción	Tamaño (en bits)	Valor mínimo	Valor máximo	Ejemplos
boolean	Valor booleano	1	N.A.	N.A.	false, true
char	Carácter Unicode	16	\u0000	\uFFFF	a, A, n, N, ñ, Ñ
byte	Entero con signo	8	-128	128	-20, 127
short	Entero con signo	16	-332768	32767	-250, 2017
int	Entero con signo	32	-2147483648	2147483647	5, 512, 12345
long	Entero con signo	64	-9223372036854775808	9223372036854775807	36854775808
float	Coma flotante de precisión simple	32	±3.40282347E+38	±1.40239846E-45	0.0F
double	Coma flotante de precisión doble	64	±1.79769313486231570E+308	±4.94065645841246544E-324	0.0, 0.0D

Nota: Los tipos de datos **float** y **double** vienen definidos por la norma **IEEE 754**[1], que es el *estándar del IEEE para aritmética en coma flotante*.

Los tipos de datos numéricos en Java no pueden representar cualquier número entero o real. Por ejemplo, el tipo de dato entero *int* tiene un intervalo de representación entre -2147483648 y 2147483647. Si se desea representar el valor correspondiente a la población mundial del planeta (más de 6 mil millones de habitantes) no puede hacerse con un dato de tipo *int*.

1.3.1. Literales o constantes literales de tipos de dato primitivos

Un **literal**, **valor literal** o **constante literal** es una constante cuyo nombre o identificador es la representación escrita de su valor y tiene ya ese significado en el código fuente de un programa Java. Seguidamente se muestran algunos ejemplos de valores o constantes literales pertenecientes a los tipos primitivos que pueden utilizarse directamente en un programa fuente de Java.

Las **constantes literales booleanas** son *false* y *true*.

Las **constantes literales de tipo carácter** aparecen entre comillas simples. Como ya se ha comentado anteriormente, los caracteres, cadenas e identificadores en Java se componen de caracteres pertenecientes al conjunto de caracteres **Unicode**[2]. Un *dato de tipo carácter* representa un único carácter. El formato de Unicode utiliza 16 bits (2 bytes) para poder codificar un total de 65536 caracteres diferentes que incluyen caracteres y

[1] https://es.wikipedia.org/wiki/IEEE_754
[2] https://home.unicode.org/

símbolos procedentes de distintas lenguas del mundo. En este conjunto de caracteres encontramos:

- **Letras mayúsculas**: 'A', 'B', 'C', ...
- **Letras minúsculas**: 'a', 'b', 'c', ...
- **Signos de puntuación**: ',', ';', ':', ...
- **Dígitos**: '0', '1', '2', ...
- **Símbolos especiales**: '#', '&', '%', ...
- **Caracteres de control**: tabulador, retorno de carro,...

Hay algunos caracteres que pueden causar algún problema en el código fuente de un programa Java debido a que se utilizan para tareas específicas dentro del lenguaje. Por ejemplo, como se verá más adelante, el carácter correspondiente a las *comillas dobles* (") se emplea para delimitar una cadena de caracteres. Para solucionar este inconveniente es necesario escaparlo con el carácter "\":

```
System.out.println("En un \"lugar\" de la Mancha");
```

El comando anterior generaría la siguiente salida:

En un "lugar" de la Mancha

Una **secuencia de escape** es una serie de caracteres que comienza por el carácter "\" y que indica que el texto que viene a continuación debe ser interpretado de otra manera. La **barra invertida** "\" se denomina *carácter de escape*, el cual indica que la secuencia de caracteres que le sigue justo a continuación debe interpretarse de otra manera. Por ejemplo, la secuencia de caracteres o secuencia de escape *\uxxxx* puede emplearse en cualquier lugar en un programa de Java para representar un carácter Unicode, siendo *xxxx* una secuencia de cuatro dígitos hexadecimales.

A continuación se muestra una tabla de caracteres representados por una secuencia de escape:

Secuencia de escape	Valor	Código Unicode equivalente
\b	Retroceso o backspace	\u0008
\t	Tabulador	\u0009
\n	Nueva línea	\u000A
\f	Salto de página	\u000C
\r	Retorno de carro	\u000D
"	Doble comilla	\u0022
'	Comilla simple	\u0027
\	Barra diagonal	\u005C

Por ejemplo, la letra 'E' corresponde con el código Unicode 0045:

```
System.out.println("\u0045n un \"lugar\" de la Mancha");
```

El comando anterior generaría la siguiente salida:

En un "lugar" de la Mancha

En la URL https://unicode-table.com/es se pueden consultar los códigos Unicode relacionados con los caracteres más comunes.

En Java una cadena no es un tipo primitivo de dato, aunque se pueden construir constantes literales de cadenas de caracteres. **Las constantes literales de cadenas de texto** se indican entre comillas dobles. Por ejemplo:

"Hola, mundo"

Al construir una cadena de caracteres se puede incluir cualquier carácter Unicode excepto un carácter de nueva línea. Si se desea incluir un salto de línea en una cadena de caracteres debe utilizarse la secuencia de escape '\n'.

Por ejemplo:

```
System.out.println("Línea 1\nLínea 2");
```

El comando anterior generaría la siguiente salida:

Línea 1

Línea 2

Las **constantes enteras** son secuencias de dígitos octales, decimales o hexadecimales en las que no se emplea el punto o coma decimal. Si comienza por un 0 indica un formato octal. Si comienza por un 0x ó 0X indica un formato hexadecimal. El resto se considera en formato decimal. Las constantes de tipo *long* se indican con una 'l' o una 'L' al final. Normalmente se emplea la 'L' para no confundir con el '1' (uno). Si no se indica ninguna de estas terminaciones Java supone que la constante es de tipo *int*.

A continuación se muestran algunos ejemplos de constantes enteras:

Valor	Descripción
34	De tipo int, solamente dígitos
-78	De tipo int, dígitos con signo negativo sin punto decimal
034	En octal (equivale al 28 decimal)
0x1C	En hexadecimal (equivale al 28 decimal)

875L	De tipo long

Las **constantes reales** o en **coma flotante** se expresan con coma decimal y opcionalmente seguidos de un exponente. El valor puede finalizarse con una 'f' o una 'F' para indicar el formato *float*. Si no se indica ninguna terminación Java supone que es un *double*. Por ejemplo:

Valor	Descripción
15.2	Valor con punto decimal, de tipo double
15.2D	El mismo valor de tipo double
1.52e1	El mismo valor de tipo double
15.8f	Valor de tipo float
15.8F	El mismo valor de tipo float

Como se verá más adelante cada tipo de dato primitivo tiene una clase correspondiente (*Boolean*, *Character*, *Byte*, *Short*, *Integer*, *Long*, *Float* y *Double*), llamadas **wrappers**, que definen también constantes y métodos útiles para trabajar con los datos.

1.3.2. Declaraciones de variables

Una variable es un espacio de la memoria correspondiente a un dato cuyo valor puede modificarse durante la ejecución de un programa y que está asociado a un identificador. **Toda variable ha de declararse antes de ser usada en el código de un programa en Java**. En la declaración de una variable debe indicarse explícitamente el identificador de la variable y el tipo de dato asociado. El tipo de dato determina el espacio reservado en memoria, los diferentes valores que puede tomar la variable y las operaciones que pueden realizarse con ella. La declaración de una variable en el código fuente de un programa de Java puede hacerse de la siguiente forma:

tipo_de_dato identificador_de_la_variable;

También se puede declarar múltiples variables en una misma línea del mismo tipo, separados por comas:

tipo_de_dato ident_1, ident_2, . . . , ident_n;

Por ejemplo:

```
int n;
double x, y;
```

En el primer ejemplo se declara 'n' como una variable de tipo *int*. En el segundo ejemplo se declaran dos variables 'x' e 'y' de tipo *double*. En Java una variable queda definida únicamente dentro del bloque de sentencias (entre llaves { }) en el que ha sido declarada. De esta forma queda determinado su ámbito o alcance (scope) en el que puede emplearse.

El identificador elegido para designar una variable debe respetar las normas de construcción de identificadores de Java. Además, por convención:

- Los identificadores de las variables comienzan con una letra minúscula. Por ejemplo: *n*, *x2*, *mes*, *clave*, *suma*, *nombre*, ...
- Si el identificador es una palabra compuesta, las palabras restantes comienzan por una letra mayúscula. Por ejemplo: *esDivisible*, *mayorEdad*, ...
- El carácter del subrayado puede emplearse en cualquier lugar del identificador de una variable, pero suele emplearse para separar nombres en identificadores de constantes.

La **declaración** e **inicialización** de una variable de tipo primitivo puede realizarse de forma simultánea en la misma línea empleando el operador asignación '='. Por ejemplo:

```
int n = 15;
```

Independientemente de haber inicializado o no, el valor asignado a la variable puede modificarse las veces que se quiera durante la ejecución del programa. También puede realizarse la declaración e inicialización de varias variables del mismo tipo primitivo en la misma línea separándolas por comas. Por ejemplo:

```
double x = 12.5, y = 25.0;
```

Como se verá más adelante, la declaración de un objeto o instancia equivalente utilizando una clase *wrapper* se realiza empleando la palabra reservada **new**. Por ejemplo:

```
Integer n = new Integer(15);
```

Esta declaración se verá más adelante con detenimiento.

1.3.3. Declaración de variables final o constantes

Las **variables finales** en Java son similares a las constantes empleadas en otros lenguajes de programación. Una vez inicializada una variable final su valor no puede ser modificado. La declaración de variables finales o constantes se realiza empleando la palabra reservada final antes del identificador del tipo de dato. Por ejemplo:

```
final int MAXIMO = 15;
```

La asignación de valor se puede posponer en el código, aunque en ningún caso su valor puede modificarse una vez ha sido inicializada ya que se generaría un error. Ejemplo de inicialización posterior a la declaración de la constante:

```
final int MAXIMO;
...
MAXIMO = 15;
```

Al igual que ocurre con las variables, el identificador elegido para designar una constante debe respetar las normas de construcción de identificadores de Java. Por convención:

- Los identificadores de las constantes se componen de letras mayúsculas. Por ejemplo: MAXIMO.
- El carácter de subrayado (_) es aceptable en cualquier lugar dentro de un identificador, pero se suele emplear sólo para separar palabras dentro de los identificadores de las constantes. Por ejemplo: MAXIMO_VALOR.

1.4. Conversiones entre tipos de dato

El proceso consiste en almacenar el valor de una variable de un determinado tipo primitivo en otra variable de distinto tipo. Suele ser una operación más o menos habitual en un programa y la mayoría de los lenguajes de programación facilitan algún mecanismo para llevarla a cabo. En cualquier caso, no todas las conversiones entre los distintos tipos de dato son posibles. Por ejemplo, en Java no es posible convertir valores booleanos a ningún otro tipo de dato y viceversa. Además, en caso de que la conversión sea posible es importante evitar la pérdida de información en el proceso.

En general, existen dos categorías de conversiones:

- **De ensanchamiento o promoción**: pasar de un valor de tipo *int* a *double* o pasar un valor de *int* a *long*.
- **De estrechamiento o contracción**: pasar de un valor *double* a *int* o pasar de un valor *long* a *int*.

Las **conversiones de promoción** transforman un dato de un tipo a otro con el mismo o mayor espacio en memoria para almacenar información. En estos casos puede haber una cierta pérdida de precisión al convertir un valor entero a real al desechar algunos dígitos significativos.

Las conversiones de promoción en Java se resumen en la siguiente tabla:

Tipo de origen	Tipo de destino
byte	short, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	double
float	float, double
double	-

Las **conversiones de contracción** son más comprometidas ya que transforman un dato de un tipo a otro con menor espacio en memoria para almacenar información. En estos casos se corre el riesgo de perder o alterar sensiblemente la información.

Las conversiones de contracción en Java se resumen en la siguiente tabla:

Tipo de origen	Tipo de destino
byte	char
short	byte, char
char	byte, short
int	byte, short, char
long	byte, short, char, int
float	byte, short, char, int, long
double	byte, short, char, int, long, float

Tanto las conversiones de promoción como las de contracción se realizan por **asignación**, **promoción aritmética** o **casting**.

1.4.1. Por asignación

Cuando una variable de un determinado tipo se asigna a una variable de otro tipo. Sólo admite conversiones de promoción. Por ejemplo: si 'n' es una variable de tipo *int* que vale 25 y 'x' es una variable de tipo *double*, entonces se produce una **conversión por asignación** al ejecutarse la sentencia:

```
int n = 25;
double x;
x = n;
```

La variable 'x' toma el valor 25.0 (valor en formato real). El valor de 'n' no se modifica.

1.4.2. Por promoción aritmética

Como resultado de una operación aritmética. Como en el caso anterior, sólo admite conversiones de promoción. Por ejemplo, si *producto* y *factor1* son variables de tipo *double* y *factor2* es de tipo *int* entonces al ejecutarse la sentencia:

```
double factor1 = 10.0;
int factor2 = 5;
double producto;
producto = factor1 * factor2;
```

el valor de *factor2* se convierte internamente en un valor en formato real para realizar la operación aritmética que genera un resultado de tipo *doublé*, en este caso la variable *producto* tendría el valor *50.0*. El valor almacenado en formato entero en la variable *factor2* no se modifica.

1.4.3. Con casting o "moldes"

Con operadores que producen la conversión entre tipos. Admite las conversiones de promoción y de contracción indicadas anteriormente. Por ejemplo: si se desea convertir un valor de tipo double a un valor de tipo int se utilizará el siguiente código:

```
int n;
double x = 82.4;
n = (int) x;
```

la variable 'n' toma el valor 82 (valor en formato entero). El valor de 'x' no se modifica. El código fuente del siguiente programa ilustra algunas de las conversiones que pueden realizarse entre datos de tipo numérico.

```
package com.cga.prog;
/**
 * Descripción: Ejemplos de conversiones entre distintos tipos de datos numéricos
 * @author ROGERGAMES
 *
 */
public class Conversiones {
	public static void main(String[ ] args) {
		int a = 2;
		double b = 3.0;
		float c = (float) (200 * a / b + 5);
		System.out.println("Valor en formato float: " + c);
		System.out.println("Valor en formato double: " + (double) c);
		System.out.println("Valor en formato byte: " + (byte) c);
		System.out.println("Valor en formato short: " + (short) c);
		System.out.println("Valor en formato int: " + (int) c);
		System.out.println("Valor en formato long: " + (long) c);
	}
}
```

Salida:

```
Valor en formato float: 138.33333
Valor en formato double: 138.3333282470703
Valor en formato byte: -118
Valor en formato short: 138
Valor en formato int: 138
Valor en formato long: 138
```

1.4.4. La Clase Scanner

La clase **Scanner** de Java provee métodos para leer valores de entrada de varios tipos y forma parte del paquete *java.util*. Los valores de entrada pueden venir de varias fuentes, ya sea desde teclado o desde fichero. Para utilizar esta clase deberemos importar la clase java.util.Scanner en nuestro código:

```
import java.util.Scanner;
```

Posteriormente tenemos que crear primero un objeto de ella para poder invocar sus métodos. La siguiente declaración crea un objeto llamado *teclado* de la clase Scanner que lee valores de entrada del teclado.

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

El propósito de pasar a **System.in** como argumento es conectar o establecer una relación entre el objeto tipo *Scanner*, con nombre teclado en la declaración anterior, y el objeto *System.in*, que representa el sistema estándar de entrada de información en Java. Si no se indica lo contrario, el teclado es, por omisión, el sistema estándar de entrada de información en Java.

Luego que se tenga un objeto de la clase Scanner asociado al sistema estándar de entrada System.in, llamamos, por ejemplo, su método **nextInt()** para entrar un valor del tipo *int*. Para entrar otros valores de otros tipos de datos primitivos, se usan los métodos correspondientes como ***nextByte()*** o **nextDouble()**.

Método	Ejemplo
nextByte()	byte b = teclado.nextByte();
nextDouble()	double d = teclado.nextDouble();
nextFloat()	float f = teclado.nextFloat();
nextInt()	int i = teclado.nextInt();
nextLong()	long l = teclado.nextLong();
nextShort()	short s = teclado.nextShort();
next()	String p = teclado.next();
nextLine()	String o = teclado.nextLine();

El método **next()** sirve para leer una palabra sola, por ejemplo "Fernando", mientras que el método **nextLine()** sirve para leer varias palabras, por ejemplo "Fernando González Perdomo".

A continuación, se muestra un ejemplo de uso de la clase *Scanner*:

```
import java.util.Scanner;
public class EntradaDatos {
    public static void main(String[ ] args){
        //Se crea el lector
        Scanner sc = new Scanner(System.in);
        //Se pide un dato al usuario, en este caso su nombre
        System.out.print("Por favor introduzca su nombre: ");
        //Se lee el nombre con nextLine() que retorna un String con el dato
        String nombre = sc.nextLine();
        //Se pide otro dato al usuario, en este caso su edad
        System.out.print("Bienvenido " + nombre + ". Por favor, introduzca su edad: ");
        //Se guarda la edad directamente con nextInt()
        int edad = sc.nextInt();
        System.out.println("Te llamas " + nombre + " y tienes " + edad + " años.");
        // Cerramos el lector
        sc.close();
    }
}
```

Por ejemplo si cuando pedimos el nombre de la persona introducimos Fernando González, y cuando pedimos la edad de la persona introducimos 17, obtenemos el siguiente resultado:

Te llamas Fernando González y tienes 17 años.

El método **close( )** se llama cuando ya no vamos a pedir más datos por pantalla.

2. Bibliografía

- Material suministrado por CGA.
- https://desarrolloweb.com/home/java
- http://www.manualweb.net/java/tipos-datos-primitivos-java/
- https://www.abrirllave.com/java/tipos-de-datos-primitivos.php
- https://javadesdecero.es/io/clase-scanner-ejemplos/