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 o simples: representan un único dato simple que puede ser de tipo `char`, `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`.

Variables referencia o datos compuestos (variables `arrays`, de una clase/ instancias, interfaces, etc.). Se implementan mediante un nombre o referencia (puntero) que contiene la dirección en memoria de un valor o conjunto de valores (objeto creado con new).

Según cambie su valor o no durante la ejecución del programa. Podemos hablar de:

- **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:** 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 datos.

Tipos de Datos Simples o Primitivos.

Es uno de los conceptos fundamentales de cualquier lenguaje de programación. Estos definen los métodos de almacenamiento disponibles para representar información, junto con la manera en que dicha información ha de ser interpretada.

Para crear una variable (de un tipo primitivo) en memoria debe declararse indicando su **tipo de variable y su identificador** que la identificará de forma única. La sintaxis de declaración de variables es la siguiente:

```
TipoSimple, Identificador1, Identificador2;
```

Ejemplo:

```
int variable1, suma;
```

Esta sentencia indica al compilador que reserve memoria para dos variables del tipo simple `int` con nombres `variable1` y `suma`.

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

Tipo	Representación / Valor	Tamaño (en bits)	Valor mínimo	Valor máximo	Valor por defecto
`boolean`	`true` o `false`	1	N.A.	N.A.	`false`
`char`	Carácter Unicode	16	\u0000	\uFFFF	\u0000
`byte`	Entero con signo	8	-128	128	0
`short`	Entero con signo	16	-32768	32767	0
`int`	Entero con signo	32	-2147483648	2147483647	0
`long`	Entero con signo	64	-9223372036854775808	9223372036854775807	0
`float`	Coma flotante de precisión simple Norma IEEE 754	32	±3.40282347E+38	±1.40239846E-45	0.0
`double`	Coma flotante de precisión doble Norma IEEE 754	64	±1.79769313486231570E+308	±4.94065645841246544E-324	0.0

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 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, ó nombre.
- si el identificador es una palabra compuesta, las palabras restantes comienzan por una letra mayúscula. Por ejemplo: `esDivisible`.
- 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;
```

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;
```

En el caso de que no se inicialice explícitamente la variable, ésta toma el valor 0 si es numérica, false si es booleana y '\0' si es de tipo carácter.

Declaración de Variables Finales o Constantes.

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.

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.`

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. Por ejemplo: pasar de un valor entero a un real.

- De estrechamiento o contracción. Por ejemplo: pasar de un valor real a un entero.

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 medio de los siguientes mecanismos:

- **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:

  ```
  x = n;
  ```

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

  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 la ejecutarse la sentencia `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 `double`. El valor almacenado en formato entero en la variable `factor2` no se modifica.

  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.

Ejemplo de compilación y ejecución del programa anterior y salida por

```
                    {
  public static void main (String [] args) {
      int a =
      double b = 3.0;
      float c = (float) (20000*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);
```

pantalla correspondiente:

```
$ > javac Conversiones.java
$ > java Conversiones
Valor en formato float: 13338.333
Valor en formato double: 13338.3330078125
Valor en formato byte: 26
Valor en formato short: 13338
Valor en formato int: 13338
Valor en formato long: 13338
```

Otro ejemplo:

```java
class TipoDatos
{
    public static void main(String args[])
    {
        // declarando el carácter
         char a = 'G';

        // El tipo de datos enteros es generalmente
        // utilizado para valores numéricos
        int i=89;

        // use byte y short si la memoria es una prioridad
        byte b = 4;

        // esto dará error ya que el número es
        // mayor que el rango de bytes
        // byte b1 = 7888888955;

        short s = 56;

        // esto dará error ya que el número es
        // más grande que el rango de short
        // short s1 = 87878787878;


        // por defecto, el valor de la fracción es double en Java
        double d = 4.355453532;

        // para float use 'f' como sufijo
        float f = 4.7333434f;

        System.out.println("char: " + a);
        System.out.println("integer: " + i);
        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("float: " + f);
        System.out.println("double: " + d);
    }
}
```