

# Curso rápido de Java

Este curso explica de manera rápida y sencilla la sintaxis básica del lenguaje de programación Java. Atención: en ningún momento pretende ser un curso de lógica de programación. Por el contrario, se supone que quien lo lea tiene un buen conocimiento previo sea en otro lenguaje imperativo (visual basic, Java, etc.) o de manera abstracta.

## Introducción

Java fue originalmente desarrollado por James Gosling de Sun Microsystems. Su sintaxis deriva en gran medida de C y Java, pero puede considerarse de más “alto nivel”. Es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo o sistema operativo. Esto quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

Cabe también mencionar que java es uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados (Fuente: Wikipedia).

## Sintaxis básica de un programa

Lo mínimo que debe tener todo programa en Java (al menos para nuestro curso) es:

```
public class Main {  
  
    public static void main(String[] args) {  
        //Este programa no hace nada  
    }  
  
}
```

Este código define una clase llamada *Main* que contiene un método principal llamado *public static void main(String[] args)*. Los comentarios (texto que va después del //) son opcionales y suelen usarse con fines descriptivos.

Es importante mencionar que la teoría general sobre clases y objetos no es el foco principal de este curso, para eso existe el curso Programación Orientada a Objetos. En otras palabras, dentro de este curso los usaremos más no ahondaremos en su teoría.

## Tipos de datos

Como cualquier lenguaje de programación, Java considera ciertos tipos de datos primitivos, siendo de nuestro interés los siguientes:

# Curso rápido de Java

Tipo	Notación	Tamaño	Rango
Entero muy corto	byte	1 byte	-127 a 127
Entero corto	short	2 bytes	-32768 a 32767
Entero normal	int	4 bytes	-2147483648 a 2147483648
Entero largo	long	8 bytes	-9,22 x 10 <sup>16</sup> a 9,22 x 10 <sup>16</sup>
Real simple	float	4 bytes	Pocas cifras decimales
Real doble	double	8 bytes	Números muy grandes o con muchas cifras decimales
Carácter simple	char	1 byte	Caracteres ASCII
Booleano	bool	1 bit	true o false

Es importante mencionar que dentro de nuestros programas en Java es necesario declarar TODAS las variables que utilicemos. Declarar significa que tenemos que indicar explícitamente de qué tipo de dato es cada variable que vamos a usar. Aunque generalmente esto se hace para todas las variables justo después de abrir la llave del `public static void main(String[] args)` en realidad puede ir en cualquier parte del código siempre y cuando se haga ANTES de usarlas.

## Operaciones de entrada y salida

Obviamente, la finalidad de nuestros programas será leer uno o más datos y mostrar uno o más resultados. Empecemos por lo segundo. Para mostrar algo por la consola de salida podemos usar la instrucción:

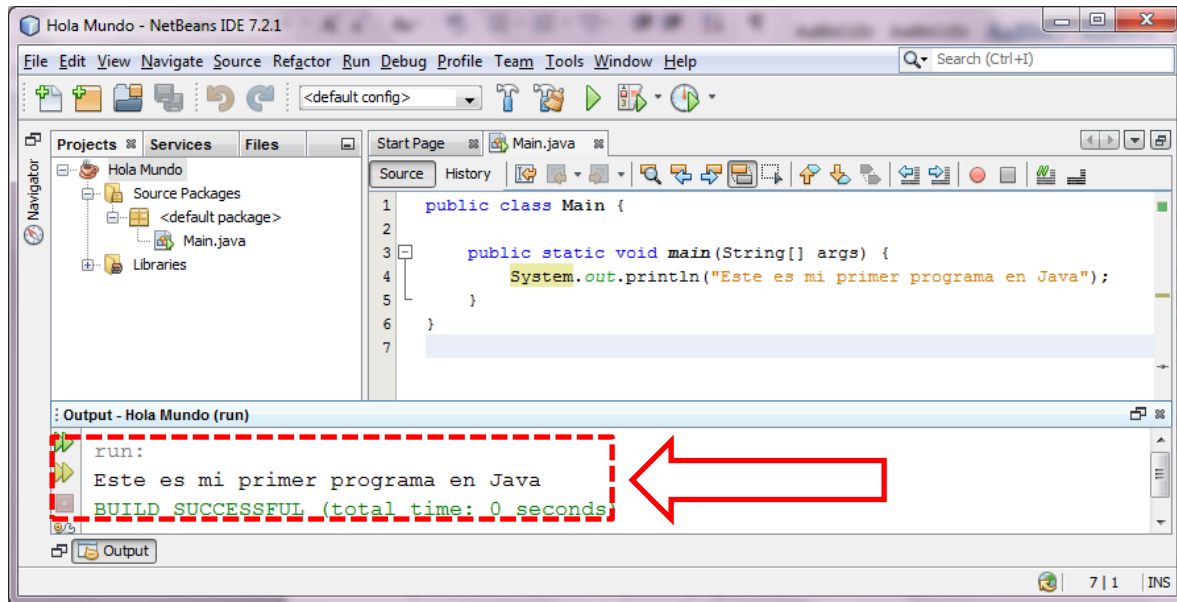
```
System.out.println("");
```

Colocando entre las comillas dobles el mensaje que queremos mostrar. Por ejemplo, un programa para mostrar “Este es mi primer programa en Java” sería algo como:

```
public class Main {  
  
    public static void main(String[] args) {  
        System.out.println("Este es mi primer programa en Java");  
    }  
  
}
```

El cual al ejecutarlo mostraría:

# Curso rápido de Java

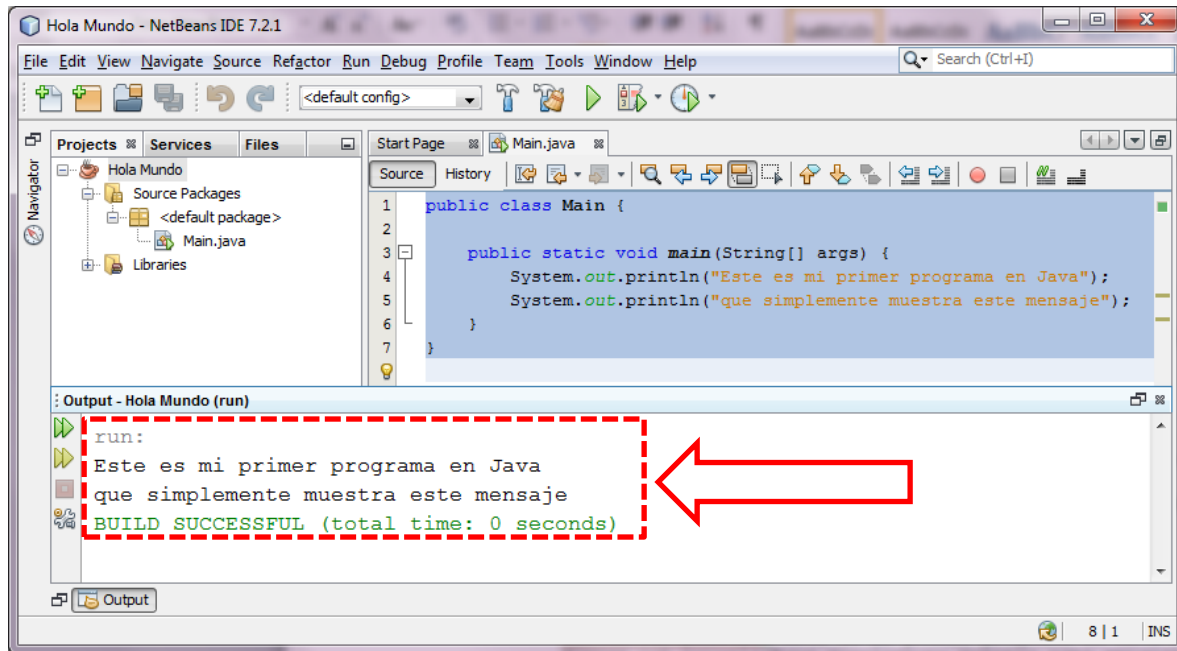


Notemos que la instrucción `System.out.println("");` imprime lo que está entre comillas dentro de una única línea (renglón). Así por ejemplo, el programa:

```
public class Main {  
  
    public static void main(String[] args) {  
        System.out.println("Este es mi primer programa en Java");  
        System.out.println("que simplemente muestra este mensaje");  
    }  
}
```

Mostraría al ejecutarlo:

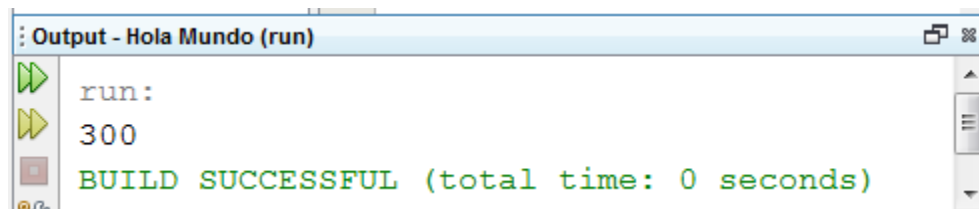
# Curso rápido de Java



Esta misma instrucción la podemos utilizar para mostrar, no mensajes literales (entre comillas) sino valores de variables o incluso resultados de operaciones. Por ejemplo el siguiente programa:

```
public class Main {  
  
    public static void main(String[] args) {  
        short a, b;  
        a = 100;  
        b = 200;  
        System.out.println(a+b);  
    }  
}
```

Mostraría al ejecutarlo:



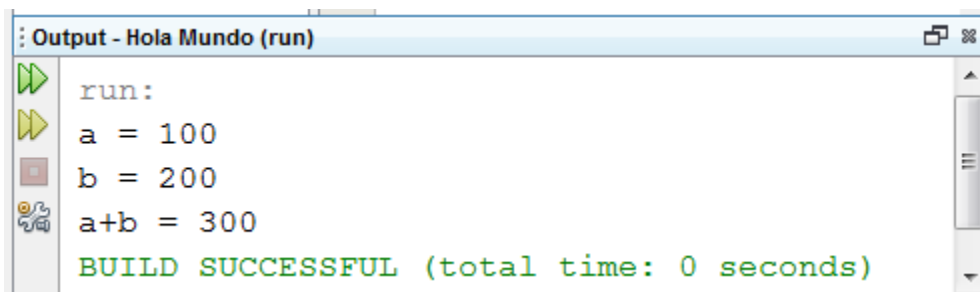
Incluso, podemos utilizarlo para combinar mensajes que tengan mensajes literales con valores. Así por ejemplo el siguiente programa:

```
public class Main {  
  
    public static void main(String[] args) {
```

# Curso rápido de Java

```
short a, b;  
a = 100;  
b = 200;  
System.out.println("a = " + a);  
System.out.println("b = " + b);  
System.out.println("a+b = " + (a+b));  
}  
}
```

Mostraría al ejecutarlo:



Notemos que el operador + sirve entonces tanto para sumar valores como para concatenar (unir) mensajes. Se pueden concatenar tantos mensajes como se desee. Sin embargo hay que tener en cuenta que al combinar textos literales con valores:

```
System.out.println("a+b = " + (a+b));
```

Daría un valor diferente de

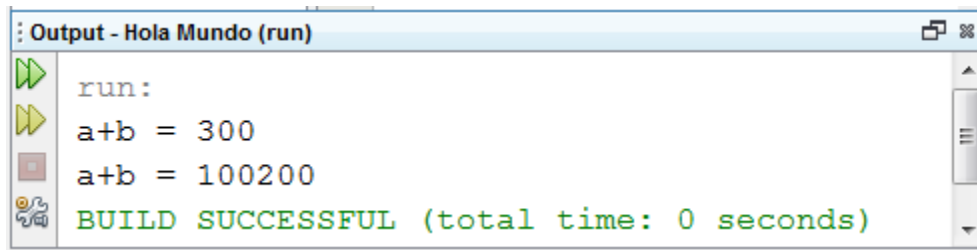
```
System.out.println("a+b = " + a + b);
```

El primero mostraría el valor de la suma de a más b, mientras que el segundo mostraría el valor de a concatenado al valor de b. En otras palabras, al ejecutar el programa:

```
public class Main {  
  
    public static void main(String[] args) {  
        short a, b;  
        a = 100;  
        b = 200;  
        System.out.println("a+b = " + (a+b));  
        System.out.println("a+b = " + a + b);  
    }  
}
```

Se mostraría:

# Curso rápido de Java



Es decir, cuando se combinan textos literales con sumas de valores es necesario darle una ‘manito’ a Java colocando las sumas entre paréntesis.

Ahora, no para mostrar sino para leer valores, podemos crear un objeto de la clase *Scanner*. Por ejemplo, con la instrucción:

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

Estamos definiendo un objeto llamado entrada de tipo Scanner, que luego podremos utilizar con alguno de los siguientes métodos que corresponden a los tipos de datos nativos descritos previamente:

Tipo	Método
Entero muy corto	<code>nextByte</code>
Entero corto	<code>nextShort</code>
Entero normal	<code>nextInt</code>
Entero largo	<code>nextLong</code>
Real simple	<code>nextFloat</code>
Real doble	<code>nextDouble</code>
Booleano	<code>nextBoolean</code>

Notemos que el único método faltante es para el tipo *char*. En cambio lo que hay es un *nextLine* que lee no un carácter sino una cadena de caracteres completa que luego podemos descomponer.

Así por ejemplo al ejecutar el siguiente código:

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        int x;
        Scanner entrada = new Scanner(System.in);
        x = entrada.nextInt();
    }
}
```

# Curso rápido de Java

```
}
```

Lo que haría el programa es esperar que el usuario digite un número entero normal por consola y, una vez lo hace, se lo asigna a la variable *x* (y no hace nada más :P).

Notemos que en el programa anterior fue necesario agregar al inicio del código la instrucción

```
import java.util.*;
```

Lo que sucede es que Java, al igual que muchos lenguajes, utiliza librerías. Una librería contiene un conjunto de clases y funciones que podemos utilizar. En el ejemplo anterior, la clase *Scanner* se encuentra dentro de la librería *java.util*

Si intentamos usar una clase o una función que requiere de cierta librería y la misma no está incluida (importada) dentro de nuestro código, el IDE nos mostrará un error.

Un ejemplo con entrada y salida es el siguiente:

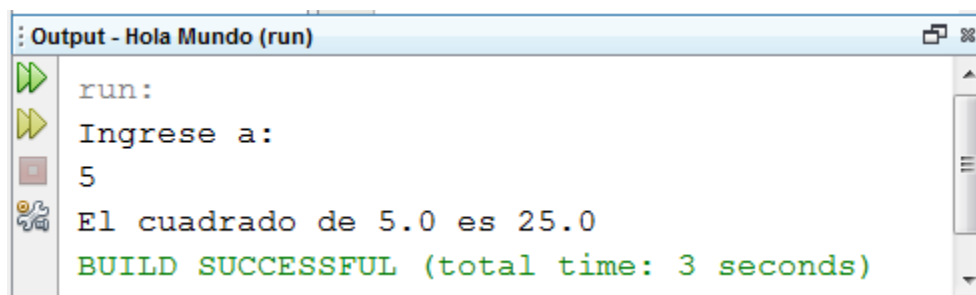
```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        float a;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Ingrese a:");
        a = entrada.nextFloat();
        System.out.println("El cuadrado de " + a + " es " + a*a);
    }

}
```

El cual al ejecutarlo, e ingresando el valor 5 (digitando 5 y luego dando ENTER), mostraría:



Para ver un ejemplo más complejo de entrada de datos miremos el siguiente programa:

```
import java.util.Scanner;

public class Main {
```

# Curso rápido de Java

```
public static void main(String[] args) {  
    float x;  
    int y;  
    Scanner entrada = new Scanner(System.in);  
    System.out.println("Ingrese x:");  
    x = entrada.nextFloat();  
    System.out.println("Ingrese y:");  
    y = entrada.nextInt();  
    System.out.println("x elevado a la y: " + Math.pow(x,y));  
}  
}
```

El cual al ejecutarlo, e ingresando los valores 2 y 3 (dando ENTER cada vez), mostraría:

Notemos que en el ejemplo anterior utilizamos la función *Math.pow(base, potencia)* que como se podrán imaginar sirve para realizar una potenciación. Así como esta, hay muchas funciones útiles que podemos usar en nuestros programas. Algunas de ellas son:

Funcion	Resultado
<code>Math.abs(x)</code>	Valor absoluto de x
<code>Math.acos(x)</code>	Coseno inverso de x
<code>Math.asin(x)</code>	Seno inverso de x
<code>Math.atan(x)</code>	Tangente inversa de x
<code>Math.cos(x)</code>	Coseno trigonométrico de x
<code>Math.cosh(x)</code>	Coseno hiperbólico de x
<code>Math.exp(x)</code>	$e^x$
<code>Math.log(x)</code>	Logaritmo base e de x
<code>Math.log10(x)</code>	Logaritmo base 10 de x
<code>Math.max(a,b)</code>	Valor mayor entre a y b
<code>Math.min(a,b)</code>	Valor menor entre a y b
<code>Math.round(x)</code>	Redondea x a entero
<code>Math.sin(x)</code>	Seno trigonométrico de x
<code>Math.sinh(x)</code>	Seno hiperbólico de x



# Curso rápido de Java

<code>Math.sqrt(x)</code>	Raíz cuadrada de x
<code>Math.tan(x)</code>	Tangente trigonométrica de x
<code>Math.tanh(x)</code>	Tangente hiperbólica de x

## Condicionales

En Java existen básicamente tres tipos de condicionales que pueden usarse según la necesidad: el condicional simple (una única condición), el condicional compuesto (una condición y su “contrario”), y el condicional múltiple (varias condiciones mutuamente excluyentes).

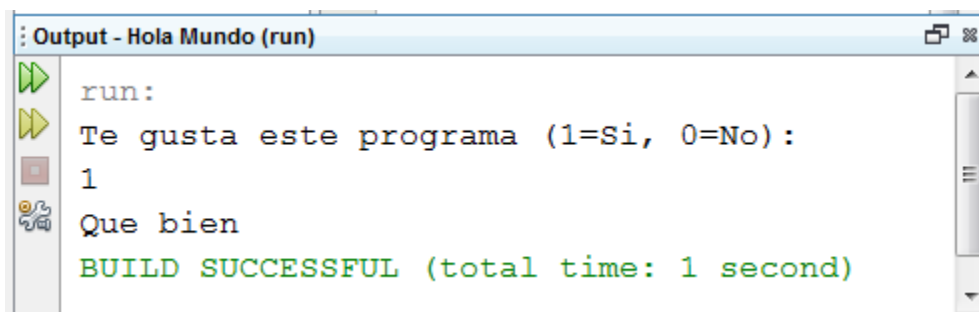
La sintaxis del condicional simple es la siguiente:

```
if ( /*Expresión lógica*/ ){  
    //Instrucciones que se ejecutan cuando la expresión lógica es verdadera  
}
```

Miremos el siguiente programa de ejemplo:

```
import java.util.Scanner;  
  
public class Main {  
  
    public static void main(String[] args) {  
        int x;  
        Scanner entrada = new Scanner(System.in);  
        System.out.println("Te gusta este programa (1=Si, 0=No):");  
        x = entrada.nextInt();  
        if (x == 1){  
            System.out.println("Que bien");  
        }  
    }  
}
```

El cual al ejecutarlo, e ingresar 1, mostraría lo siguiente:



```
Output - Hola Mundo (run)  
run:  
Te gusta este programa (1=Si, 0=No):  
1  
Que bien  
BUILD SUCCESSFUL (total time: 1 second)
```

Ahora, la sintaxis del condicional compuesto es la siguiente:

# Curso rápido de Java

```
if (/* expresión lógica */){
    //Instrucciones que se ejecutan cuando la expresión lógica es verdadera
}
else{
    //Instrucciones que se ejecutan cuando la expresión es falsa
}
```

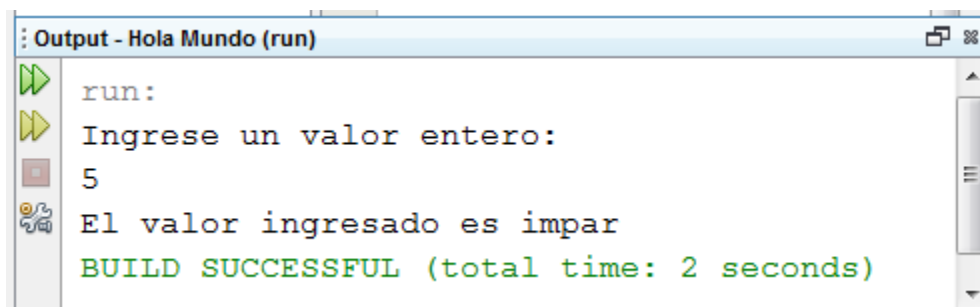
Miremos el siguiente programa de ejemplo:

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        int x;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Ingrese un valor entero:");
        x = entrada.nextInt();
        if (x % 2 == 0){
            System.out.println("El valor ingresado es par");
        }
        else{
            System.out.println("El valor ingresado es impar");
        }
    }
}
```

El cual al ejecutarlo, e ingresar 5, mostraría lo siguiente:



Finalmente, la sintaxis del condicional múltiple, con  $n$  alternativas, es la siguiente:

```
if (/* expresión lógica 1 */){
    //Instrucciones que se ejecutan si la expresión 1 es verdadera
}
else if (/* expresión lógica 2 */){
    //Instrucciones que se ejecutan si la expresión 1 es falsa
    //pero la 2 es verdadera
}
else if (/* expresión lógica 3 */){
    //Instrucciones que se ejecutan si la expresión 1 y 2 son falsas
    //pero la 3 es verdadera
}
```

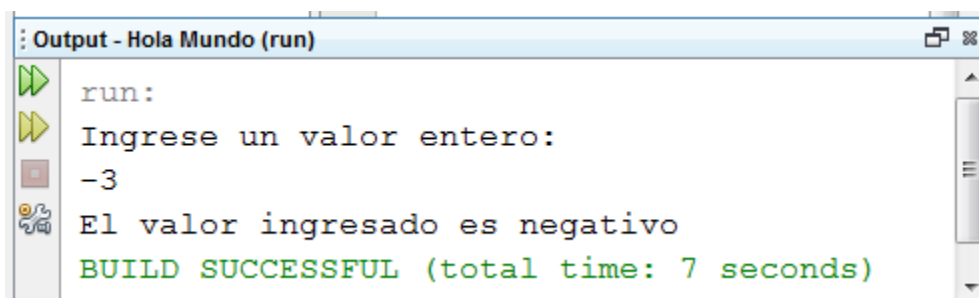
# Curso rápido de Java

```
}  
  
//...  
  
else if (/* expresión lógica n */){  
    //Instrucciones que se ejecutan si la expresión 1 a n-1 son falsas  
    //pero la n es verdadera  
}  
else{  
    //Instrucciones que se ejecutan si todas las expresiones son falsas  
}
```

Miremos el siguiente programa de ejemplo:

```
import java.util.Scanner;  
  
public class Main {  
  
    public static void main(String[] args) {  
        int x;  
        Scanner entrada = new Scanner(System.in);  
        System.out.println("Ingrese un valor entero:");  
        x = entrada.nextInt();  
        if (x > 0){  
            System.out.println("El valor ingresado es positivo");  
        }  
        else if (x < 0){  
            System.out.println("El valor ingresado es negativo");  
        }  
        else{  
            System.out.println("El valor ingresado es nulo");  
        }  
    }  
}
```

El cual al ejecutarlo, e ingresar -3, mostraría lo siguiente:



Ahora, respecto a los operadores de comparación que usa Java, y que pueden usarse en las expresiones lógicas, la sintaxis es la siguiente:

# Curso rápido de Java

Operador	Sintaxis
> (mayor que)	>
< (menor que)	<
≥ (mayor o igual que)	>=
≤ (menor o igual que)	<=
= (igual)	==
≠ (diferente)	!=

Notemos que el comparador de igualdad no es = sino == (doble igual) para que el Java distinga entre una asignación y una comparación.

Respecto a los conectores que también pueden usarse en las expresiones lógicas, Java maneja la siguiente sintaxis:

Conector	Sintaxis
Conjunción (Y)	&&
Disyunción (O)	
Negación (No)	!

Aprovechemos para recordemos las tablas de verdad de dichos conectores:

Conjunción	Disyunción	Negación
verdadero Y verdadero = verdadero verdadero Y falso = falso falso Y verdadero = falso falso Y falso = falso	verdadero Y verdadero = verdadero verdadero Y falso = verdadero falso Y verdadero = verdadero falso Y falso = falso	No verdadero = falso No falso = verdadero

Miremos el siguiente programa de ejemplo que hace uso de varios operadores de comparación y comparadores:

```
import java.util.Scanner;

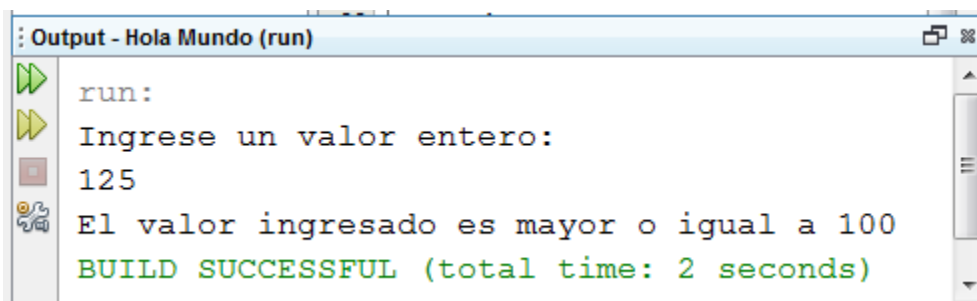
public class Main {

    public static void main(String[] args) {
        int x;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Ingrese un valor entero:");
        x = entrada.nextInt();
        if ((x % 2 == 0) && (x > 0) && (x < 100)){
            System.out.println("El valor ingresado es par positivo menor que 100");
        }
    }
}
```

# Curso rápido de Java

```
else if ((x % 2 == 1) && (x > 0) && (x < 100)){
    System.out.println("El valor ingresado es par positivo menor que 100");
}
else if (x >= 100) {
    System.out.println("El valor ingresado es mayor o igual a 100");
}
else{
    System.out.println("El valor ingresado es negativo o nulo");
}
}
}
```

El cual al ejecutarlo, e ingresar 125, mostraría lo siguiente:



Ejercicio: Realice una prueba de escritorio al siguiente código y determine cuál sería el resultado. Luego ejecute dicho código y compare.

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        int a = 3, b = 4, c = 5;
        if (a != b && c > b){
            System.out.println("Hola");
        }
        if ((3*b > 4*a) || (3*b < 2*c)) && b < c){
            System.out.println("Que");
        }
        if (!(a+b < c) && ((c == 2*a) || (b/2 < a))){
            System.out.println("Tal");
        }
    }
}
```

## Iteración definida

En Java la manera más común de utilizar la iteración definida es:

# Curso rápido de Java

```
for (X = valor_inicial; X /*comparación*/ valor_final; X = X ± variación){  
    //Instrucciones  
}
```

X puede ser cualquier variable de tipo entera o real y la comparación puede ser <, >, <=, >=, ó incluso !=. Notemos también que la variación de la variable no necesariamente tiene que ser positiva, incluso podría usarse otro operador como el de multiplicación.

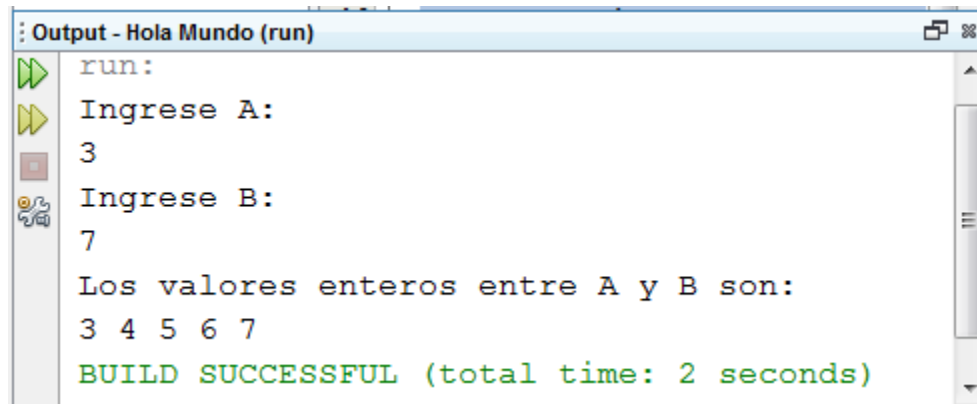
Algo a tener en cuenta es que Java tiene algunas particularidades respecto a su sintaxis con el fin de “abreviar código”. Así por ejemplo la instrucción  $X = X + \text{variación}$  puede abreviarse como  $X += \text{variación}$  (funciona también para los otros operadores algebraicos). Más aún, cuando la variación de la variable es incrementarla una unidad se puede reemplazar la expresión  $X = X + 1$  por  $X++$ . Igualmente, cuando la variación es disminuirla una unidad se puede reemplazar  $X = X - 1$  por  $X--$ .

Para aclarar todo esto, miremos este ejemplo para mostrar todos los números enteros entre A y B (incluyéndolos):

```
import java.util.Scanner;  
  
public class Main {  
  
    public static void main(String[] args) {  
        int A, B, i;  
        Scanner entrada = new Scanner(System.in);  
        System.out.println("Ingrese A:");  
        A = entrada.nextInt();  
        System.out.println("Ingrese B:");  
        B = entrada.nextInt();  
        if (A < B){  
            System.out.println("Los valores enteros entre A y B son:");  
            for (i = A; i <= B; i++){  
                System.out.print(i + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

El cual al ejecutarlo e ingresar los valores por ejemplo 3 y 7 nos mostraría:

# Curso rápido de Java



```
run:
Ingrese A:
3
Ingrese B:
7
Los valores enteros entre A y B son:
3 4 5 6 7
BUILD SUCCESSFUL (total time: 2 seconds)
```

Notemos en este ejemplo dos cosas. La primera es que dentro del *for* usamos `print` en vez de `println`. Esta instrucción, a diferencia de `println` no hace un “salto de línea (de renglón)” luego de imprimir. Es por eso que los valores del 3 al 7 se muestran en una sola línea.

Lo segundo es que al final del *for* añadimos un `println()` sin comillas y sin nada entre ellas. Esto nos sirve simplemente para “bajar de renglón”. Si no lo hubiéramos usado, el mensaje “BUILD SUCCESSFUL ...” que pone el IDE al final hubiera quedado pegado a nuestro mensaje.

**Ejercicio:** Realice una prueba de escritorio al siguiente código y determine cuál sería el resultado. Luego ejecute dicho código y compare.

```
public class Main {

    public static void main(String[] args) {
        int M, N, k;
        M = 15;
        N = 2;
        for (k = M; k > N; k -= 3){
            System.out.print(k + " ");
        }
        System.out.println();
    }

}
```

**Ejercicio:** Volvamos a hacer lo mismo pero ahora con un programa un poco más complejo:

```
public class Main {

    public static void main(String[] args) {
        int i, j;
        for (i = 5; i > 0; i --){
            for (j = 1; j <= i; j ++){
                System.out.print(j*10 + " ");
            }
            System.out.println();
        }
    }

}
```

# Curso rápido de Java

```
}
```

## Iteración indefinida

En Java hay dos maneras de utilizar la iteración indefinida:

```
while(/* expresión lógica */){  
    //Instrucciones  
}
```

y

```
do{  
    //Instrucciones  
} while(/* expresión lógica */);
```

La diferencia entre una y otra es que en la primera se evalúa primero la expresión lógica y luego se ejecutan las instrucciones, y en la segunda se hace al revés. Esto indirectamente significa que en la segunda forma las instrucciones se ejecutan al menos una vez, mientras que en la primera puede que nunca se ejecuten.

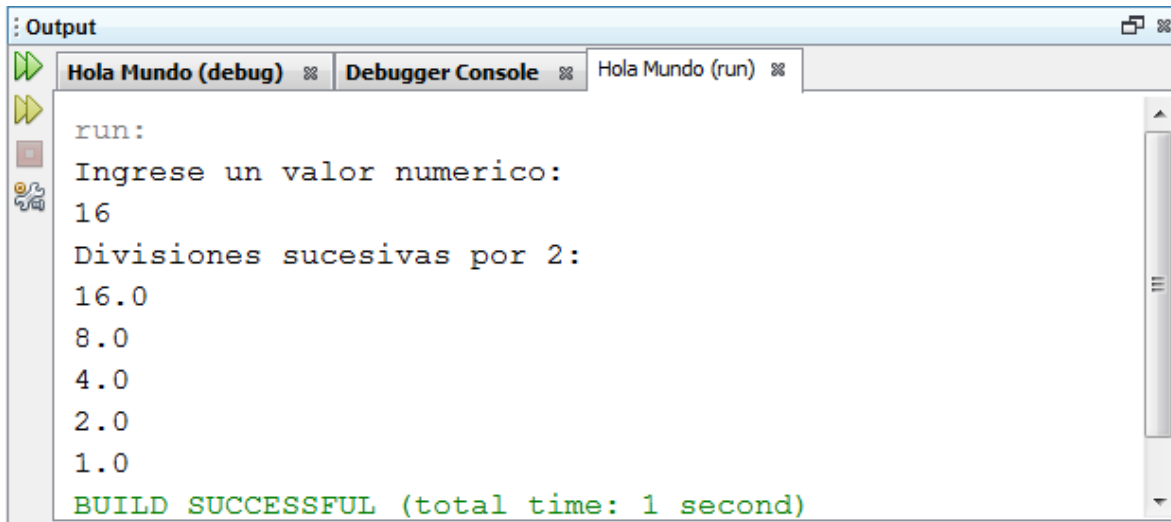
Para aclarar esto, miremos este ejemplo para mostrar todas las divisiones sucesivas por 2 de un número  $x$  hasta que el resultado sea menor o igual a 1:

```
import java.util.Scanner;  
  
public class Main {  
  
    static int x = 5; //variable global  
  
    public static void main(String[] args) {  
        float x;  
        Scanner entrada = new Scanner(System.in);  
        System.out.println("Ingrese un valor numerico:");  
        x = entrada.nextInt();  
        System.out.println("Divisiones sucesivas por 2:");  
        while (x >= 1){  
            System.out.println(x);  
            x /= 2;  
        }  
    }  
}
```

El cual al ejecutarlo, e ingresando un valor de 16, sería:



# Curso rápido de Java



```
run:
Ingrese un valor numerico:
16
Divisiones sucesivas por 2:
16.0
8.0
4.0
2.0
1.0
BUILD SUCCESSFUL (total time: 1 second)
```

## Arreglos

En Java para declarar un arreglo es necesario primero decir de qué tipo es y, por cada dimensión (una si es vector, dos si es matriz, etc.), poner una pareja de corchetes []. Luego se debe solicitar memoria y especificar cuál va a ser el tamaño de cada dimensión. Este tamaño no significa necesariamente que el arreglo contendrá ese número de elementos (algunos pueden quedar vacíos), si no que máximo puede tener ese número.

Así por ejemplo para declarar un arreglo unidimensional (un vector) A de 10 enteros normales se usaría la instrucción:

```
int A[] = new int[10];
```

Mientras que para un arreglo bidimensional (una matriz) de reales simples B de 4 filas por 5 columnas se usaría la instrucción:

```
float B[][] = new float[4][5];
```

Y así para los demás tipos de datos y con cuantas dimensiones se requiera.

Un aspecto SUMAMENTE IMPORTANTE es que la primera posición del arreglo (en cualquier dimensión) es la posición cero, no la uno. Así por ejemplo, en un arreglo unidimensional de 4 elementos, las posiciones de los mismos serán 0, 1, 2, y 3, es decir, para este ejemplo la posición 4 NO EXISTE.

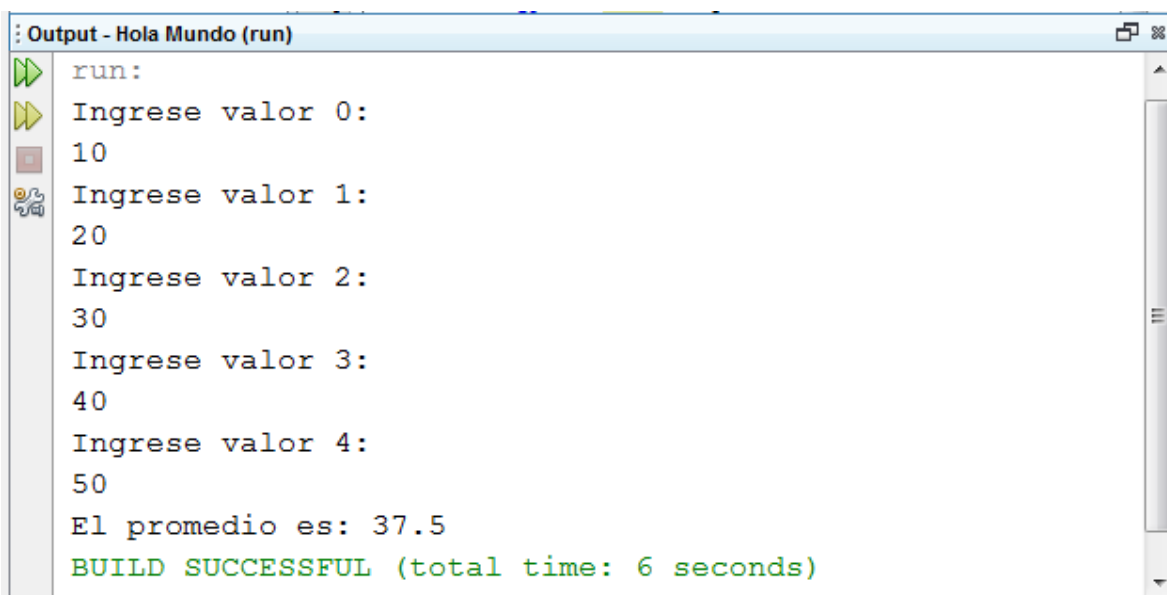
Para aclarar esto, miremos este programa de ejemplo para leer un vector de 5 números reales simples y mostrar su promedio:

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

# Curso rápido de Java

```
public class Main {  
  
    public static void main(String[] args) {  
        float A[] = new float[5];  
        int i;  
        float prom = 0;  
        Scanner entrada = new Scanner(System.in);  
        for (i = 0; i <= 4; i++){  
            System.out.println("Ingrese valor " + i + ":");  
            A[i] = entrada.nextInt();  
            prom += A[i];  
        }  
        prom /= 4;  
        System.out.println("El promedio es: " + prom);  
    }  
}
```

El cual al ejecutarlo, e ingresar los valores 10, 20, 30, 40 y 50, mostraría lo siguiente:



```
Output - Hola Mundo (run)  
run:  
Ingrese valor 0:  
10  
Ingrese valor 1:  
20  
Ingrese valor 2:  
30  
Ingrese valor 3:  
40  
Ingrese valor 4:  
50  
El promedio es: 37.5  
BUILD SUCCESSFUL (total time: 6 seconds)
```

## Funciones

En Java, a diferencia de otros lenguajes como VisualBasic, no hay diferenciación entre funciones y subprogramas, todas son funciones sólo que estas pueden o no devolver un valor. La sintaxis para declarar una función es:

```
static /* tipo */ /* nombre */ (/* tipo1 arg1 */, /* tipo2 arg2 */, /* tipoN argN */){  
    // Declaración de variables  
    // Instrucciones  
    return /* valor que se retorna si es que retorna algo*/;  
}
```

# Curso rápido de Java

Miremos en detalle cada uno de estos elementos:

*tipo*: Especifica el tipo de dato que devuelve la función. En caso que no devuelva nada el tipo debe ser *void*.

*nombre*: Es el nombre que se le desee poner a la función (preferiblemente de manera mnemotécnica) y debe seguir las mismas normas establecidas para dar nombre a las variables: comenzar con una letra, puede contener números o el símbolo guion bajo ‘\_’ y no puede tener espacios en blanco.

*argumentos*: Son los datos con los que se debe “alimentar” la función. Para cada uno de ellos se debe especificar el tipo.

*variables*: Son las variables que se requieren dentro de las instrucciones que son llevadas a cabo al interior de la función para dar con el resultado, pero que no son argumentos. El ámbito de estas variables es la función (más adelante se detallará el concepto de ámbito).

*instrucciones*: Todas las operaciones que lleva a cabo la función (lectura, escritura, asignación, operaciones matemáticas, lógicas, etc.)

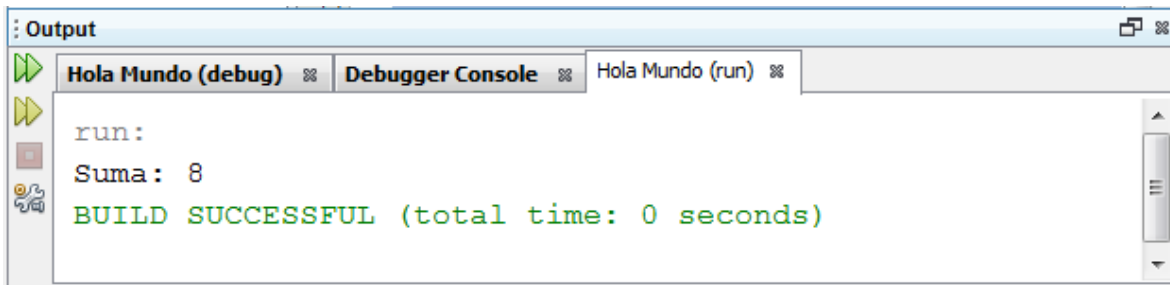
*return*: Determina cuál es el valor que devuelve la función, siempre y cuando no sea de tipo *void*.

Para aclarar miremos el siguiente programa de ejemplo:

```
public class Main {  
  
    public static void main(String[] args) {  
        int A, B, C;  
        A = 3;  
        B = 5;  
        C = suma(A, B);  
        System.out.println("Suma: " + C);  
    }  
  
    static int suma(int x, int y){  
        return x+y;  
    }  
  
}
```

El cual al ejecutarlo mostraría:

# Curso rápido de Java

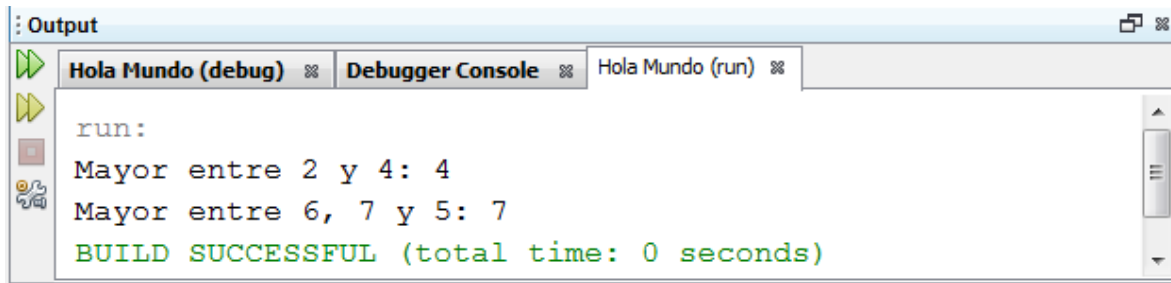


Una particularidad de las funciones en Java es que se pueden “sobrecargar”. Sobrecargar significa definir varias funciones con el mismo nombre, pero con diferentes conjuntos de parámetros. Esto es útil en problemas donde necesitamos de funciones que hagan lo mismo, pero de manera diferente según los argumentos que reciban (cantidad y tipo). Miremos el siguiente ejemplo para aclarar esto:

```
public class Main {  
  
    public static void main(String[] args) {  
        System.out.println("Mayor entre 2 y 4: " + mayor(2, 4));  
        System.out.println("Mayor entre 6, 7 y 5: " + mayor(6, 7, 5));  
    }  
  
    static int mayor(int x, int y){  
        if (x > y){  
            return x;  
        }  
        else{  
            return y;  
        }  
    }  
  
    static int mayor(int x, int y, int z){  
        if (x > y && x > z){  
            return x;  
        }  
        else if (y > x && y > z){  
            return y;  
        }  
        else{  
            return y;  
        }  
    }  
  
}
```

El cual al ejecutarlo mostraría:

# Curso rápido de Java



## Ámbito de las variables

Cuando declaramos una variable se reserva un espacio de memoria para guardarla y se le asigna un identificador (el nombre de la variable). Dependiendo de la parte del código donde se haga la declaración, la variable adquiere un ámbito, es decir, un lugar donde dicha variable es “visible”.

Generalmente, un ámbito se define mediante un bloque, el cual se delimita por medio de llaves { }, como en el caso del `public static void main` y en general de las funciones. Cuando el programa sale del ámbito de una variable el espacio asociado a esta se libera (la variable se destruye).

Una variable es global cuando se declara por fuera (es decir, antes) de todos los bloques (incluido el de la función `public static void main`). Al ser global su ámbito es todo el programa.

La visibilidad se refiere a las partes del programa en las que puede accederse al espacio de memoria asignado a una variable. Una variable es visible cuando es posible acceder a su contenido mediante el identificador que le fue asignado. En teoría toda variable es visible en su ámbito, sin embargo, en algunas ocasiones una declaración de variable local (perteneciente a la función donde fue declarada) puede ocultar una variable global con el mismo nombre.

Para aclarar esto, miremos el siguiente ejemplo:

```
import java.util.Scanner;

public class Main {

    static int x = 5; //variable global

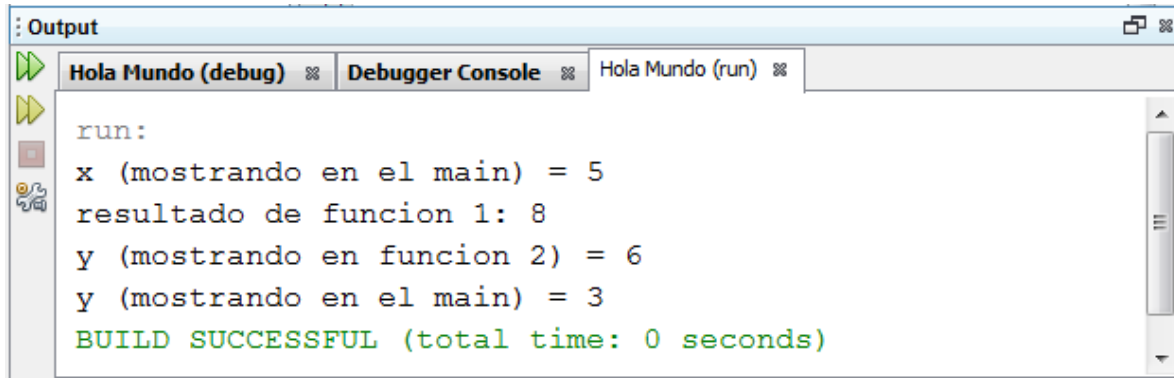
    public static void main(String[] args) {
        int y = 3;
        System.out.println("x (mostrando en el main) = " + x);
        System.out.println("resultado de funcion 1: " + funcion1());
        funcion2(y);
        System.out.println("y (mostrando en el main) = " + y);
    }

    static int funcion1(){
        int x = 4; //variable local que oculta la global dentro de esta funcion
        return 2*x;
    }
}
```

# Curso rápido de Java

```
static void funcion2(int z){  
    int y; //variable local diferente a la del main  
    y = 2*z;  
    System.out.println("y (mostrando en funcion 2) = " + y);  
}  
  
}
```

El cual al ejecutarlo mostraría:



## Tareas

Para poner en práctica lo aprendido en este curso rápido, realiza los siguientes ejercicios.

1. Implementar un programa en Java para resolver una ecuación cuadrática:  $AX^2 + BX + C = 0$  considerando todos los casos (no hay raíces, hay una sola raíz, hay raíces pero no son reales, hay dos raíces reales).
2. Implementar un programa en Java para leer tres valores correspondientes a la longitud de tres rectas, determinar si esas tres rectas pueden formar un triángulo y en caso afirmativo decir qué tipo de triángulo sería.
3. Implementar un programa en Java para sumar los números enteros del A al B.
4. Una forma de calcular el cuadrado de los números enteros es sumar los impares de la siguiente manera:

$$\begin{aligned}1 &= 1 \\4 &= 1 + 3 \\9 &= 1 + 3 + 5\end{aligned}$$

Utilizar este concepto para implementar un programa en Java que calcule el cuadrado de los N primeros números enteros.

5. Implementar un programa en Java para leer un valor entero y decir si es primo o no.

# Curso rápido de Java

6. Un número perfecto es aquel en que la suma de los divisores da el mismo número. Implementar un programa en Java para mostrar si un número entero  $Z$  es perfecto o no. Ejemplo: el número 6 es perfecto por que  $6 = 1 + 2 + 3$ , mientras que 8 no es perfecto porque  $8 \neq 1 + 2 + 4$ .
7. Implementar un programa en Java para leer cuatro valores enteros positivos  $A1, A2, A3, A4$ . El programa debe reemplazar estos valores por  $|A1 - A2|$ ,  $|A2 - A3|$ ,  $|A3 - A4|$  y  $|A4 - A1|$  respectivamente. Si al hacer los cuatro reemplazos todos los valores anteriores son igual a los nuevos el programa termina, en caso contrario debe continuar repitiendo el proceso. En cada iteración se deben mostrar los cuatro valores.
8. Implementar un programa en Java para leer un arreglo de 10 valores enteros normales y mostrar cuál es el menor, cuál el mayor, cuál es el promedio, y cuál es la desviación estándar.
9. Implementar un programa en Java para leer una matriz de  $3 \times 3$  y mostrar la sumatoria de los elementos de la diagonal principal.
10. Implementar un programa en Java que lea un número entero normal positivo y muestre su factorial. Dicho cálculo debe hacerse implementando una función.