El gran libro de Java a fondo
Curso de programación

Pablo Augusto Sznajdleder

4.ª edición

Alfaomega

Marcombo 1945-2020 75 años

El gran libro de Java a fondo

Curso de programación

4.ª edición

Pablo Sznajdleder

El gran libro de Java a fondo

Curso de programación

4.ª edición

Pablo Sznajdleder

Alfaomega

Marcombo 1945-2020 75 años

El gran libro de Java a fondo. Curso de programación
Pablo Augusto Sznajdleder

Cuarta edición: 2020
ISBN: 978-987-3832-50-5

Cuarta edición: MARCOMBO, S.L. 2020

ISBN: 978-84-267-2841-8
D.L.: B-3747-2020

Impreso en Servicepoint
Printed in Spain

A mi esposa Analía, a mi hijo Octaviano y a mi mamá Nélida. A todos ellos: GRACIAS.

Pablo Sznajdleder

Mensaje del editor

Los conocimientos son esenciales en el desempeño profesional, sin ellos es imposible lograr las habilidades para competir laboralmente. La universidad o las instituciones de formación para el trabajo ofrecen la oportunidad de adquirir conocimientos que serán aprovechados más adelante en beneficio propio y de la sociedad; el avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos. Cuando se toma la decisión de embarcarse en una vida profesional, se adquiere un compromiso de por vida: mantenerse al día en los conocimientos del área u oficio que se ha decidido desempeñar.

Alfaomega tiene por misión ofrecerles a estudiantes y profesionales conocimientos actualizados dentro de lineamientos pedagógicos que faciliten su utilización y permitan desarrollar las competencias requeridas por una profesión determinada. Alfaomega espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Alfaomega hace uso de los medios impresos tradicionales en combinación con las tecnologías de la información y las comunicaciones (TIC) para facilitar el aprendizaje.

Libros como éste tienen su complemento en una página Web, en donde el alumno y su profesor encontrarán materiales adicionales.

Esta obra contiene numerosos gráficos, cuadros y otros recursos para despertar el interés del estudiante, y facilitarle la comprensión y apropiación del conocimiento. Cada capítulo se desarrolla con argumentos presentados en forma sencilla y estructurada claramente hacia los objetivos y metas propuestas.

Los libros de Alfaomega están diseñados para ser utilizados dentro de los procesos de enseñanza-aprendizaje, y pueden ser usados como textos para diversos cursos o como apoyo para reforzar el desarrollo profesional.

Alfaomega espera contribuir así a la formación y el desarrollo de profesionales exitosos para beneficio de la sociedad.

Acerca del autor

Pablo Augusto Sznajdleder es Ingeniero en Sistemas de Información y Magister en la misma materia. Su tesis de maestría propone una metodología para implementar transferencias de conocimiento mediadas por tecnologías informáticas.

Es profesor universitario en las asignaturas de “Algoritmos y estructura de datos” y “Patrones algorítmicos para estructuras avanzadas”, ambas en la Universidad Tecnológica Nacional, Facultad Regional Buenos Aires (UTN.BA).

Además de la presente obra, es autor de “Algoritmos a fondo”, “JEE a fondo”, “Programación estructurada a fondo” y “Programación orientada a objetos a fondo” entre otras.

Desde 1996 se desempeña como instructor y consultor en tecnologías Java proveyendo servicios de capacitación, desarrollo y *coaching* a las principales empresas del país.

THE JAVA LISTENER

Learning music, enjoying Java

thejavalistener.com

/thejavalistener

Contenido

Prólogo

Para los que recién se inician en del desarrollo de aplicaciones empresariales, sepan que en todo el mundo existe una enorme demanda de programadores; dicha demanda es cada vez mayor.

En la empresa donde trabajo desde hace más de 20 años, todos los desarrollos de aplicaciones se realizan en Java.

En Oracle tenemos decenas de miles de programadores. Y por cada extensión y/o adaptación del código fuente de cualquier aplicación necesitamos acudir a los desarrolladores Java. Y esto también es válido para la totalidad de las empresas que disponen de soluciones empresariales de clase mundial y tantas otras con desarrollos de software nicho o local que han confiado en el modelo abierto de Java.

Lograr una ejecución óptima en cualquier sistema operativo, tener extensiones y definiciones para armar aplicaciones en múltiples capas, disponer de *frameworks open source* y *frameworks* propietarios de empresas que después se transforman en estándares a través del trabajo de las comunidades tecnológicas y poder ejecutar las aplicaciones en servidores provistos por múltiples oferentes son algunos de los beneficios de este lenguaje de programación.

Tengo la firme convicción de que invertir tiempo en aprender Java para emprender luego una carrera de desarrollador es una decisión bien encaminada. Y quienes oyeron mis sugerencias han podido conseguir trabajos muy bien pagos.

Conozco a Pablo desde que aprendí a escribir Java. Éramos muy jóvenes en esos tiempos. Yo estaba encargado de difundir la tecnología Oracle y él dictaba cursos; incluso algo más: formaba jóvenes para que, después de un entrenamiento intensivo, pudieran adquirir la experiencia de haber desarrollado aplicaciones.

El primer artículo que leí de Pablo fue el famoso “Hola Mundo” y me ayudo a entender, de manera fácil y divertida, de qué se trataba este lenguaje de programación. A fines de los años 90 ya discutíamos sobre *design patterns* y mejores usos de las herramientas.

Para los que no conocen a Pablo, deben saber que su fanatismo por Java lo llevó a conseguir un número de móvil que finalizara con “Java": 9999-JAVA (5282). Esto en un país donde no se permite elegir qué número queremos tener en nuestro teléfono.

Volviendo al libro, celebro que acerquemos las tecnologías empresariales de un modo sencillo. Java a fondo describe los patrones de diseño más útiles para emprender desarrollos empresariales y los explica con muchísima sencillez.

Néstor Camilo
Director de Arquitectura Empresarial
Oracle Latin America

Agradecimientos

A mi familia por el aguante y a mi editor y amigo Damián Fernández por la paciencia.

Antes de comenzar a leer

En este libro, se utiliza la tipografía `Courier` en los casos en los que se hace referencia a código o acciones por realizar en el ordenador, ya sea en un ejemplo o cuando se refiere a alguna función mencionada en el texto. También se usa para indicar menús de programas, teclas, URL, grupos de noticias o direcciones de correos electrónicos.

Los términos o definiciones cuyos significados están muy asociados al inglés se expresan en dicho idioma en *cursiva*.

El código fuente de los ejemplos, así como todos los recursos didácticos y de programación que se utilizan en este libro pueden descargarse desde www.marcombo.info con el código `JAVA4`.

CAPÍTULO 1
INTRODUCCIÓN AL LENGUAJE DE PROGRAMACIÓN JAVA

1.1. INTRODUCCIÓN

1.1.1. JAVA EN SUS INICIOS

El lenguaje de programación Java apareció en 1995 y, desde entonces, se caracterizó por marcar tendencia gracias a sus aportes e innovaciones.

Durante los primeros años, la principal característica del lenguaje fue la posibilidad de programar Applets; pequeños programas java que podían ser incrustados en las páginas Web para proveerles dinamismo e interacción.

Pero, con el tiempo, Java se fue reconvirtiendo y posicionando fuertemente del lado del *server*. Gracias a esta reconversión dejó de ser un simple lenguaje de programación para convertirse en una verdadera plataforma de desarrollo y de ejecución de aplicaciones empresariales; las cuales, por supuesto, se programan en Java.

1.1.2. JSE Y JEE – JAVA ESTÁNDAR Y JAVA ENTERPRISE EDITION

Para diferenciar entre el lenguaje de programación Java y la plataforma de desarrollo de aplicaciones empresariales, en 1998 se lanzó al mercado el recordado Java*2*; que materializó esta separación mediante dos distribuciones independientes entre sí:

- J*2*SE (*Java2 Standard Edition*) y
- J*2*EE (Java*2 Enterprise Edition*).

Hoy en día, a más de 20 años, JSE y JEE se mantienen permanentemente actualizadas y adaptadas a las nuevas tendencias de mercado y realidades tecnológicas.

La edición estándar (JSE) incluye el lenguaje de programación propiamente dicho, el compilador, las bibliotecas básicas y una serie de herramientas adicionales.

Por su parte, la *enterprise edition* (JEE) se compone de un conjunto de bibliotecas de uso e interés netamente empresarial. También forman parte de esta edición una serie de lineamientos denominados Patrones de Diseño y un conjunto de especificaciones técnicas que estandarizan la construcción de servidores.

1.1.3. DESARROLLO DE APLICACIONES

Aunque Java es un lenguaje de programación de propósitos generales, los desarrolladores solemos apegarnos fuertemente a los lineamientos y patrones de diseño recomendados en la edición empresarial.

Así, el objetivo de este libro no es solo enseñar un lenguaje de programación; es también explicar las buenas prácticas que hacen al diseño de las aplicaciones Java, los principales patrones de diseño, el desarrollo en capas y la separación entre el *frontend* y el *backend* de la aplicación.

1.1.4. HOLA MUNDO

El siguiente programa escribe “Hola Mundo!” en la consola.

```
package demo;

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

```
        // Escribimos en la consola
        System.out.println("Hola Mundo!");
    }
}
```

Analicemos el programa línea por línea.

El código comienza con la sentencia `package` que indica en qué paquete (carpeta) quedará ubicado el programa.

```
package demo;
```

Los *packages* establecen un espacio de nombres (*namespace*) que hace posible que dos o más clases tengan el mismo nombre; siempre y cuando estén ubicadas en diferentes paquetes. Este tema lo estudiaremos más adelante.

En la siguiente línea de código se indica el nombre de la clase.

```
public class HolaMundo
{
```

Como Java es un lenguaje de programación orientada a objetos, los programas se escriben dentro de clases. Clases y Objetos lo estudiaremos más adelante.

Luego, el código continúa con la función `main`. Cuando una clase contiene a la función (o método) `main` se convierte en un programa que puede ser ejecutado.

```
    public static void main(String args[])
    {
        // Escribimos en la consola
        System.out.println("Hola Mundo!");
    }
}
```

Los comentarios comienzan con `//` (doble barra). Finalmente, la sentencia `System.out.println` escribe una cadena en la consola.

Los bloques de código comienzan con `{` (llave que abre) y finalizan con `}` (llave que cierra). De este modo, el primer grupo de llaves delimita dónde comienza y finaliza la clase `HolaMundo`. Y el segundo grupo de llaves delimita dónde comienza y finaliza la función `main`.

Todas las sentencias que no precedan a un bloque de código deben finalizar con `;` (punto y coma).

1.1.5. ENTORNO INTEGRADO DE DESARROLLO (IDE)

Cómo Instalar Eclipse, compilar y depurar

Generalmente, para programar utilizamos una herramienta de programación que se denomina IDE; iniciales de *Integrated Development Enviroment*.

La IDE nos asiste durante todo el proceso de programación, y nos permite editar el código fuente del programa, compilarlo, ejecutarlo, depurarlo y documentarlo; entre muchas otras cosas.

Existen diversas IDE para trabajar con Java; destacándose, principalmente, las siguientes: Eclipse, NetBeans e IntelliJ IDEA. Las dos primeras son *open source* (gratuitas) mientras que la segunda es una herramienta de pago.

El lenguaje de programación Java es totalmente independiente de todas estas herramientas. Gracias a esto podemos utilizar la IDE que más nos agrade.

1.1.6. VERSIONES JAVA

Desde el inicio, las versiones del lenguaje Java se denominaron JDK (iniciales de *Java Development Kit*). La primera versión fue JDK1.0.2; la siguiente fue JDK1.1.*x* y luego, a partir de JDK1.2, se comenzó a hablar de Java*2*. La siguiente versión fue Java*5*, Java*7*, Java*8*, Java*10*, y así sucesivamente.

Cada versión del lenguaje Java agrega más características de programación; corrige *bug*s y mejora la *performance* de la versión anterior. Sin embargo, no es de esperar

que todas las versiones agreguen cosas significativas. Por esto, en el párrafo anterior, solo incluí aquellas versiones que sí introdujeron cambios relevantes.

A partir de Java*10*, se tomó la decisión de liberar una nueva versión cada seis meses. De este modo, en las sucesivas versiones, los aportes que podremos observar al pasar de una versión a la siguiente serán mínimos. En la mayoría de los casos innecesarios si los observamos desde el punto de vista del programador; más aún, desde el punto de vista de un programador sin experiencia en este lenguaje.

El siguiente cuadro resume cómo, cada nueva versión del lenguaje de programación fue incorporando más herramientas y características.

Versión	JDK	Aportes
JDK1.0.2	JDK1.0.2	Clases básicas, IO, AWT, *networking*, Applet
JDK1.1.*x*	JDK1.1.*x*	JDBC, tecnología de *beans*, RMI
Java*2*	JDK1.2.*x*	Swing
Java*5*	JDK1.5.*x*	*Annotations*, *generics*
Java*7*	JDK1.7.*x*	*try* con recurso, inferencia de tipos genéricos
Java*8*	JDK1.8.*x*	Expresiones lambda, *streams*
Java*10*	JDK10.*x*	Inferencia de tipo de dato

Tabla 1.1. Evolución del lenguaje de programación Java.

Las nuevas versiones del lenguaje Java siempre se diseñan de modo inclusivo respecto de las anteriores. Así, quien conoce Java*2* puede, perfectamente, programar utilizando cualquiera de las versiones más recientes.

1.2. ELEMENTOS DEL LENGUAJE DE PROGRAMACIÓN

En este apartado estudiaremos los principales elementos del lenguaje de programación Java: estructuras de control, tipos de dato y sintaxis.

1.2.1. ENTRADA Y SALIDA ESTÁNDAR

Por defecto, la entrada estándar es el teclado y la salida es la consola. Para escribir un texto en la consola utilizamos `System.out.println`.

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

En esta línea de código estamos invocando al método `println` sobre el objeto `out`, declarado como público y estático en la clase `System`. Todos estos son conceptos que estudiaremos más adelante.

El ingreso de datos a través del teclado lo hacemos con la clase `Scanner`, como se muestra a continuación.

Para simplificar la lectura de los ejemplos optaremos por omitir la línea que indica el *package* donde la clase está ubicada.

```
import java.util.Scanner;

public class HolaMundoPersonalizado
{
   public static void main(String[] args)
   {
      // Preparamos para leer por teclado
      Scanner scanner = new Scanner(System.in);

      // El usuario ingresa su nombre
      System.out.print("Ingrese su nombre: ");
      String nombre = scanner.nextLine();

      // HolaMundo personalizado
      System.out.println("Hola Mundo, "+nombre);

      // Cerramos el scanner
      scanner.close();
   }
}
```

En el ejemplo anterior utilizamos la sentencia `import` para "importar" la clase `Scanner`. Aunque este tema lo abordaremos más adelante, podemos anticipar que con `import` le indicamos al compilador dónde debe ir a buscar las clases que utilizamos en el programa.

Tengamos en cuenta que ciertas clases como `String` y `System` no requieren ser importadas porque, al pertenecer al *core* del lenguaje, se importan automáticamente.

Es importante subrayar la diferencia que existe entre `System.out.print` y `System.out.println`. Ambos imprimen una cadena. Pero el segundo, además, agrega un salto de línea al final.

La clase `Scanner` permite ingresar datos de diferentes tipos. Algunos ejemplos son:

```
// Ingresamos un int por teclado
int n = scanner.nextInt();
System.out.println(n);

// Ingresamos un double por teclado
double d = scanner.nextDouble();
System.out.println(d);

// Ingresamos un String por teclado
String s = scanner.next();
System.out.println(s);
```

1.2.2. IDENTIFICADORES Y DECLARACIÓN DE VARIABLES

Podemos declarar variables en cualquier parte del código del programa, indicando el tipo de dato y el nombre de la variable o identificador. Por ejemplo:

```
// Tipo: int. Variable o identificador: a
int a;
```

Nombres de variable **válidos** son: `fecha`, `iFecha`, `fecha3`, `fechaNacimiento`, `_fecha` y `fecha_nacimiento` entre otros.

Nombres de variable **NO válidos** son: `fecha-nacimiento`, `fecha+nacimiento`, `3fecha`, `-fecha`, etcétera.

1.2.3. COMENTARIOS EN EL CÓDIGO

Como en todos los lenguajes de programación, podemos incluir comentarios que nos ayuden a entender el código del programa.

Comentarios en una sola línea	Comentarios de varias líneas

`// Esto es un comentario`
`// Esto es otro comentario`
	`/*`
	`Todo este parrafo esta`
	`comentado y puede contener`
	`tantas lineas como quiera`
	`*/`

1.2.4. TIPOS DE DATO

Aunque Java es un lenguaje de programación orientada a objetos, existen varios tipos de dato primitivos que conviven con las clases.

1.2.4.1. Tipos de dato primitivos

Existen los siguientes tipos de dato, cada uno de los cuales tiene la longitud (en bytes) que se indica a la derecha:

Tipos enteros	bytes
`byte`	1
`char`	2
`short`	2
`int`	4
`long`	8

Tipos flotantes	bytes
`float`	4
`double`	8

Tipo lógico	bytes
`boolean`	1

Tabla 1.2. Tipos de dato primitivos y sus longitudes.

Además, existe la clase `String` que no es un tipo de dato primitivo, pero se utiliza como si lo fuera.

A diferencia del lenguaje C, que permite indicar si queremos que una variable de tipo entero sea signada o no, en Java todos los tipos enteros son signados. Salvo el tipo `char` que es *unsigned*.

1.2.4.2. Wrappers: clases que representan a los tipos primitivos

Para cada uno de los tipos primitivos existe una clase que lo representa y permite realizar diferentes operaciones relacionadas con el tipo de dato en cuestión. A estas clases las llamaremos *wrappers*.

Tipo primitivo	Wrapper
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Tabla 1.3. Tipos de dato primitivos con sus correspondientes wrappers.

Algunos ejemplos sobre cómo se utilizan los *wrappers* son los siguientes:

```
int i=1234;

String a = Integer.toString(i);
System.out.println(a); // SALIDA: 1234

String b = Integer.toBinaryString(i);
System.out.println(b); // SALIDA: 10011010010

String c = Integer.toHexString(i);
System.out.println(c); // SALIDA: 4D2

String d = "1234";

int j = Integer.parseInt(d);
System.out.println(j); // SALIDA: 1234
```

1.2.4.3. Autoboxing: conversión automática entre tipos primitivos y wrappers

La conversión entre un tipo de dato y su correspondiente *wrapper* es automática. Esta característica se denomina *autoboxing*.

Veamos algunos ejemplos:

```
Integer i = new Integer(1234);
int j = i;
System.out.println(j); // SALIDA: 1234

int n = new Integer(567);
System.out.println(n); // SALIDA: 567
```

En reglas generales, para declarar variables utilizamos los tipos de dato primitivos; y usamos los *wrappers* para realizar las operaciones vinculadas a cada uno de estos. Sin embargo, los *wrappers*, al ser objetos, pueden contener el valor `null`; lo cual, como veremos más adelante, resulta ser de gran utilidad.

```
Integer I = null;

// ...

if( i!=null )
{
   // ...
}
```

1.2.5. CONSTANTES

Para declarar un valor constante utilizamos el modificador `final`. Generalmente, lo acompañamos con `public` y `static`. De este modo, es habitual ver que las constantes están declaradas como en el siguiente ejemplo:

```
public static final int DIA_LUNES = 1;
```

```
public static final int DIA_MARTES = 2;
public static final int DIA_MIERCOLES = 3;

// :
// :

public static final int DIA_DOMINGO   = 7;
```

1.2.6. VALORES LITERALES

Podemos expresar valores literales para todos los tipos de dato primitivos. Por ejemplo: en el siguiente código vemos cómo expresar valores de tipo `char`, `int`, `boolean`, `double` y `String` (aunque este último no sea un tipo primitivo).

```
// Literal de tipo char
char c='A';

// Literal de tipo int
int i = 123;

// Literal de tipo boolean (true o false)
boolean b = true;

// Literal de tipo double
double d = 3.14;

// Literal de tipo String
String s = "Hola";
```

Dado que los valores literales numéricos con decimales son, por defecto, `double`, para expresar un valor literal de tipo `float` tenemos que agregarle el sufijo `f`.

```
float f = 3.14f;
```

En la práctica resulta poco frecuente usar el tipo `float`. Incluso, como el tipo `double` suele presentar problemas de aproximación, los programadores con experiencia utilizan, en su lugar, la clase `BigDecimal`.

1.2.7. LITERALES EXPRESADOS EN OTROS SISTEMAS DE NUMERACIÓN

1.2.7.1. Números enteros expresados en binario

Anteponiendo el prefijo `0b` (cero b) podemos expresar un valor literal numérico entero mediante su representación en el sistema binario. Opcionalmente, el carácter _ (guion bajo) puede ayudarnos a separar los dígitos del número.

```
// Valor entero expresado en sistema binario
int b = 0b00000101_00111001_01111111_10110001;
System.out.println(b); // SALIDA 87654321
```

Por cuestiones de claridad agregamos, al número binario del ejemplo anterior, ceros a la derecha; para completar los 8 bit del primer byte. Estos ceros a la derecha podrían no estar y el número seguiría siendo el mismo.

```
int b = 0b101_00111001_01111111_10110001;
```

1.2.7.2. Números enteros expresados en hexadecimal

Para expresar un valor numérico entero mediante su representación hexadecimal anteponemos el prefijo `0x` (cero equis).

```
// Valor entero expresado en sistema hexadecimal
int c = 0b0101_0011_1001_0111_1111_1011_0001;
int d = 0x5____3____9____7____F____B____1;
int e = 0x05397FB1;

System.out.println(c); // SALIDA: 87654321
System.out.println(d); // SALIDA: 87654321
System.out.println(e); // SALIDA: 87654321
```

1.2.7.3. Valores enteros expresados en octal

También podemos representar valores enteros como números octales, utilizando el prefijo `0` (cero).

```
int f = 0b101_001_110_010_111_111_110_110_001;
int g =  05___1___6___2___7___7___6___6___1;
int h = 000516277661;

System.out.println(f); // SALIDA: 87654321
System.out.println(g); // SALIDA: 87654321
System.out.println(h); // SALIDA: 87654321
```

1.2.8. CARACTERES ESPECIALES

Llamamos carácter especial a aquellos caracteres que, en el código fuente de un programa, tienen otro uso que no es la representación del carácter en sí mismo.

Por ejemplo: el carácter " (comilla doble) se utiliza, en los programas, para delimitar cadenas de caracteres; y el carácter ' (comilla simple) para encerrar valores literales de tipo `char`.

Los caracteres "salto de línea" y "tabulador", por ejemplo, al presionar sus teclas en el editor de texto producen sus respectivas acciones; pero no tienen una representación directa en forma de carácter. Es decir: vemos que el texto del programa salta de una línea a otra, pero no vemos representado el carácter en sí mismo.

Todos los caracteres especiales tienen una representación literal y, por ende, pueden ser utilizados como `char` o parte de una cadena de caracteres.

Por ejemplo, el carácter "salto de línea" se representa así: `\n` (léase: barra ene). En el siguiente código incluimos un carácter "salto de línea" dentro de una cadena literal:

```
String s = "Esto es una cadena \nque sigue en la otra linea";
System.out.println(s);
```

La salida será:

```
Esto es una cadena
que sigue en la otra linea
```

Análogamente, el carácter \" (léase: barra comilla) representa a la comilla doble.

```
String s = "El lenguaje \"Java\" es muy bueno";
System.out.println(s);
```

La salida será:

```
El lenguaje "Java" es muy bueno
```

El carácter \ (carácter barra) se llama "carácter de escape" y, para usarlo como valor literal, también debe anteponérsele un carácter barra.

```
String s = "El caracter \\ es el caracter de escape";
System.out.println(s);
```

La salida será:

```
El caracter \ es el caracter de escape
```

En resumen, los caracteres especiales que podemos utilizar anteponiéndoles el carácter de escape son los siguientes:

Carácter	Descripción
\n	Salto de línea
\b	Retorno de carro
\t	Tabulador
\"	Comilla doble
\'	Comilla simple
\\	Barra
\u*dddd*	Cualquier carácter expresado mediante su código Unicode

Tabla 1.3. Caracteres especiales que deben ser precedidos por el carácter de escape.

1.2.9. ESTRUCTURAS DE CONTROL

A continuación, veremos las diferentes estructuras de control.

if

```
if( condicion )
{
   // ...
}
else
{
   // ...
}
```

while

```
while( condicion )
{
   // ...
}
```

for

```
for(int i=0; condicion; i++)
{
   // ...
}
```

do-while

```
do
{
   // ...
}while( condicion );
```

Adicionalmente, existe el `if` *inline* que puede expresarse en una sola línea.

Expresión lógica	?	valor por `true`	:	valor por `false`
Expresión lógica cuyo valor de verdad se debe determinar		Resultado en caso de que la expresión resulte verdadera		Resultado en caso de que la expresión resulte ser falsa.

En el siguiente código le pedimos al usuario que ingrese su edad. Posteriormente emitiremos un mensaje que será diferente según cuál sea la edad que ingresó.

```
// El usuario ingresa su edad
System.out.print("Ingrese su edad: ");
int edad = scanner.nextInt();

// Mostramos un mensaje segun sea mayor de 18 o no
String mssg = (edad>=18)?"Bienvenido":"Debe ser mayor de 18";
System.out.print(mssg);
```

También podemos utilizar la estructura de decisión múltiple: `switch`.

En el siguiente ejemplo el usuario ingresa el número de día de la semana y el programa le muestra qué día es.

```
int iDia = scanner.nextInt();

String sDia = "";
switch(iDia)
{
   case 1:
      sDia = "Lunes";
      break;
   case 2:
      sDia = "Martes";
      break;
//     :
//     :
   case 7:
      sDia = "Domingo";
      break;
   default:
      sDia = "valor incorrecto";
}

System.out.println(iDia+" es: "+sDia);
```

El `switch` puede utilizarse, incluso, con cadenas de caracteres.

```
int iDia=0;
String sDia=scanner.nextLine();

switch(sDia)
{
   case "Lunes":
      iDia=1;
      break;
   case "Martes":
      iDia=2;
      break;
```

```
//      :
//      :
   case "Domingo":
      iDia=7;
      break;
   default:
      System.out.println("El dia ingresado es incorrecto");
}

if( iDia!=0 )
{
   String mssg = sDia+" es el dia "+iDia+" de la semana";
   System.out.println(mssg);
}
```

1.2.10. OPERADORES ARITMÉTICOS, RELACIONALES Y LÓGICOS

A continuación, veremos el resumen de los principales operadores aritméticos, relacionales y lógicos que existen en el lenguaje de programación Java.

Aritméticos	Relacionales
Sean las variables `a`, `b` y `c`, todas de tipo `int`, entonces:	Sean las variables `a`, `b` de tipo `int` y `c` de tipo `boolean`, entonces:

```
c = a+b; // suma
c = a-b; // resta
c = a*b; // producto
c = a/b; // division
c = a%b; // residuo (o modulo)
a++;  // equivale: a = a+1
a--;  // equivale: a = a-1
a+=b; // equivale: a = a+b
a-=b; // equivale: a = a-b
a*=b; // equivale: a = a*b
```

```
c = a<b;  // true si a<b
c = a>b;  // true si a>b
c = a==b; // true si a=b
c = a<=b; // true si a<=b
c = a>=b; // true si a>=b
c = a!=b; // a distinto b
```

Operadores lógicos

Sean las variables `b`, `p` y `q` de tipo `boolean`, entonces:

```
b = p && q; // AND, Producto logico
b = p || q; // OR,  Suma logica
b = !p;     // NOT, Negacion(en este caso: NOT p)
```

1.2.11. FUNCIONES O MÉTODOS ESTÁTICOS

Aunque Java es un lenguaje de programación orientada a objetos, podemos declarar métodos (funciones) e invocarlos desde `main` u otros métodos; siempre y cuando todos sean métodos estáticos.

En el siguiente ejemplo vemos el código de la clase `Funciones` que declara y resuelve las funciones `factorial` y `esPrimo`; que muestran, además, cómo utilizar las estructuras de control que estudiamos más arriba: `while`, `for` e `if` y algunos de los operadores aritméticos y lógicos.

```
public class Funciones
{
   // Calcula el factorial de n
   public static double factorial(int n)
   {
      double r=1;
      for(int i=2; i<=n; i++)
      {
         r=r*i;
      }

      return r;
   }

   // Determina si n es un numero primo o no
   public static boolean esPrimo(int n)
   {
      int i=2;
      while( n%i!=0 && i<n )
      {
         i++;
      }

      // si i es igual a n entonces: i==n es true
      return i==n;
```

```
    }

    // Sigue mas abajo
    // :
```

Un método (o función) estático (`static`) es un método de la clase. Este tema lo estudiaremos más adelante.

Ahora, continuando con el ejemplo, veamos el código del método `main` que invoca a las funciones (o métodos) `factorial` y `esPrimo`.

```
    // :
    // Viene de mas arriba

    public static void main(String args[])
    {
        Scanner scanner = new Scanner(System.in);

        // El usuario ingresa un valor
        System.out.print("Ingrese un valor: ");
        int n = scanner.nextInt();

        // Calculamos su factorial
        double f = factorial(n);
        System.out.println("Factorial de "+n+": "+f);

        // Determinamos si es numero primo o no
        boolean p = esPrimo(n);

        // if en linea
        String mssg = p?" es ": " no es ";
        System.out.println(n+mssg+"primo");

        scanner.close();
    }
}
```

1.2.12. INFERENCIA DE TIPO DE DATO

Desde Java*10* podemos prescindir de declarar, explícitamente, el tipo de dato de las variables; el compilador es capaz de inferirlo a partir del valor que le hayamos asignado por primera vez.

```
var a = 5;   // a es int
var b = 3;   // b es int
var c = a+b; // c es int
System.out.println(c); // SALIDA: 8

var s = a+"+"+b+"="+c; // s es String
System.out.println(s); // SALIDA: 5+3=8
```

Que el tipo de dato pueda ser inferido no significa que la variable no sea tipada. El tipo de la variable se establece cuando le asignamos un valor inicial. Luego, no podremos asignarle valores de otros tipos.

Esto lo comentamos solo a título ilustrativo, pues en esta obra trabajaremos declarando explícitamente los tipos de dato de las variables.

1.2.13. CLASES Y OBJETOS

Aunque será abordado con mayor profundidad en otro capítulo, considero importante explicar, aunque sea superficialmente, algunos conceptos que nos ayudarán a comprender lo que sigue.

Las clases son tipos de dato definidos por el programador. Algo similar a las estructuras (`struct`) de C. Los objetos son variables cuyo tipo de dato es una clase.

Dado que `String` es una clase, las cadenas de caracteres son objetos.

Por ejemplo, en el siguiente código declaramos la variable `a` (de tipo `int`) y el objeto `s` (de tipo `String`).

```
int a;
String s;
```