



2021/22

Desarrollo de aplicaciones WEB

# PROGRAMACIÓN

Preparado por:  
Guadalupe Cano

# TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

---

INTRODUCCIÓN A LA PROGRAMACIÓN .....	1
1. PROGRAMAS: BUSCANDO UNA SOLUCIÓN.....	2
1.1 ALGORITMOS Y PROGRAMAS.....	3
2. PARADIGMAS DE LA PROGRAMACIÓN.....	5
3. FASES DE LA PROGRAMACIÓN .....	7
3.1. RESOLUCIÓN DEL PROBLEMA.....	8
3.2. IMPLEMENTACIÓN .....	9
3.3. EXPLOTACIÓN.....	11
4. CICLOS DE VIDA DEL SOFTWARE .....	11
5. LENGUAJES DE PROGRAMACIÓN .....	13
5.1. LENGUAJE MÁQUINA. ....	14
5.2. LENGUAJE ENSAMBLADOR.....	15
5.3. LENGUAJES COMPILADOS. ....	16
5.4. LENGUAJES INTERPRETADOS. ....	17
6. EL LENGUAJE DE PROGRAMACIÓN JAVA.....	19
6.1. BREVE HISTORIA .....	20
6.2. LA POO Y JAVA.....	21
6.3. INDEPENDENCIA DE LA PLATAFORMA Y TRABAJO EN RED .....	23
6.4. SEGURIDAD Y SIMPLICIDAD.....	24
6.5. JAVA Y LOS BYTECODES.....	25
7. PROGRAMAS EN JAVA .....	26
7.1. ESTRUCTURA DE UN PROGRAMA.....	26
7.2. EL ENTORNO BÁSICO DE DESARROLLO JAVA.....	28
7.3. LA API DE JAVA. ....	29
7.4. AFINANDO LA CONFIGURACIÓN. ....	30
7.5. CODIFICACIÓN, COMPILACIÓN Y EJECUCIÓN DE APLICACIONES.....	31
7.5.1. Estandarización del código. ....	33
7.5.2. Solución a posibles problemas con la codificación de caracteres acentuados.....	36
7.6. TIPOS DE APLICACIONES EN JAVA.....	38
8. ENTORNOS INTEGRADOS DE DESARROLLO (IDE).....	39
8.1. ¿QUÉ SON? .....	40
8.2. IDE ACTUALES.....	41

TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN	
8.3.	EL ENTORNO NETBEANS.....
8.4.	INSTALACIÓN Y CONFIGURACIÓN.....
8.5.	ASPECTO DEL ENTORNO Y GESTIÓN DE PROYECTOS.....
9.	VARIABLES E IDENTIFICADORES.....
9.1.	IDENTIFICADORES.....
9.2.	CONVENIOS Y REGLAS PARA NOMBRAR VARIABLES.....
9.3.	PALABRAS RESERVADAS.....
9.4.	TIPOS DE VARIABLES I.....
9.4.1.	TIPOS DE VARIABLES II.....
10.	LOS TIPOS DE DATOS.....
10.1.	TIPOS DE DATOS PRIMITIVOS I.....
10.1.1.	Tipos de datos primitivos II.....
10.2.	DECLARACIÓN E INICIALIZACIÓN.....
10.3.	TIPOS REFERENCIADOS.....
10.4.	TIPOS ENUMERADOS.....
11.	LITERALES DE LOS TIPOS PRIMITIVOS.....
12.	OPERADORES Y EXPRESIONES.....
12.1.	OPERADORES ARITMÉTICOS.....
12.2.	OPERADORES DE ASIGNACIÓN.....
12.3.	OPERADORES DE RELACIÓN.....
12.4.	OPERADORES LÓGICOS.....
12.5.	OPERADOR CONDICIONAL.....
12.6.	OPERADORES DE BITS.....
12.7.	TRABAJOS CON CADENAS.....
12.8.	PRECEDENCIA DE OPERADORES.....
13.	CONVERSIÓN DE TIPO.....
14.	COMENTARIOS.....
	ANEXO I. LISTADO DE IDES.....
	ANEXO II. CONVERSIÓN DE TIPOS DE DATOS EN JAVA.....

## INTRODUCCIÓN A LA PROGRAMACIÓN.

### Caso práctico

La evolución de Internet y de las nuevas tecnologías, así como las diferentes posibilidades para establecer nuevas líneas de negocio para la empresa BK Programación, han hecho que Ada haya decidido abrir una vía de innovación. Para ello, su empresa deberá realizar el desarrollo de sus aplicaciones a través de lenguajes y técnicas de programación modernos, aunque con una eficiencia y flexibilidad contrastadas.

María y Juan, ayudados y orientados por Ada, recordarán y ampliarán sus conocimientos relacionados con la programación, permitiéndoles crear software que pueda adaptarse a nuevas situaciones, como el funcionamiento en diferentes plataformas (PDA, Móviles, Web, etc.) o la interacción con bases de datos. Todo ello sin perder de vista de donde parten y hacia dónde quieren redirigir sus esfuerzos.

Estas innovaciones, junto a la predisposición para adaptarse y evolucionar que BK Programación está potenciando en todas sus áreas, repercutirán en una mayor capacidad de respuesta ante las necesidades de sus posibles clientes. En definitiva, conseguir mayor competitividad.

¿Cuántas acciones de las que has realizado hoy, crees que están relacionadas con la programación? Hagamos un repaso de los primeros instantes del día: te ha despertado la alarma de tu teléfono móvil o radio-despertador, has preparado el desayuno utilizando el microondas, mientras desayunabas has visto u oído las últimas noticias a través de tu receptor de televisión digital terrestre, te has vestido y puede que hayas utilizado el ascensor para bajar al portal y salir a la calle, etc. Quizá no es necesario que continuemos más para darnos cuenta de que casi todo lo que nos rodea, en alguna medida, está relacionado con la programación, los programas y el tratamiento de algún tipo de información.

El volumen de datos que actualmente manejamos y sus innumerables posibilidades de tratamiento constituyen un vasto territorio en el que los programadores tienen mucho que decir.

En esta primera unidad realizaremos un recorrido por los conceptos fundamentales de la programación de aplicaciones. Iniciaremos nuestro camino conociendo con qué vamos a trabajar, qué técnicas podemos emplear y qué es lo que pretendemos conseguir. Continuando con el análisis de las diferentes formas de programación existentes, identificaremos qué fases conforman el desarrollo de un programa, avanzaremos detallando las características relevantes de cada uno de los lenguajes de programación disponibles, para posteriormente, realizar una visión general del lenguaje de programación Java. Finalmente, tendremos la oportunidad de conocer con qué herramientas podríamos desarrollar nuestros programas, escogiendo entre una de ellas para ponernos manos a la obra utilizando el lenguaje Java.

# 1. PROGRAMAS: BUSCANDO UNA SOLUCIÓN.

## Caso práctico

**Ada** conoce bien lo que significa tener que llevar a cabo el proceso completo de creación de software y sabe que, en ocasiones, no se le da la importancia que debería a las fases iniciales de este proceso. Quiere que **Juan**, que desarrolla programas casi sin darse cuenta, recuerde las ventajas que aporta un buen análisis inicial de los problemas a solucionar y que no aborde el desarrollo de sus programas sentándose directamente ante el ordenador a teclear código.

**Juan** le comenta a **Ada** y **María**: —La verdad es que cuando conoces bien un lenguaje de programación crees que puedes hacer cualquier programa directamente sobre el ordenador, pero al final te das cuenta de que deberías haberte parado a planificar tu trabajo. Muchas veces tienes que volver atrás, recodificar y en ocasiones, rehacer gran parte del programa porque lo que tienes no está bien planteado.

**María**, que permanece atenta a lo que dicen **Ada** y **Juan**, quiere aprender bien desde el principio y tendrá la ventaja de tener a su lado a dos expertos.

Generalmente, la primera razón que mueve a una persona hacia el aprendizaje de la programación es utilizar el ordenador como herramienta para resolver problemas concretos. Como en la vida real, la búsqueda y obtención de una solución a un problema determinado, utilizando medios informáticos, se lleva a cabo siguiendo unos pasos fundamentales. En la siguiente tabla podemos ver estas analogías.

Resolución de problemas	
En la vida real.	En Programación.
Observación de la situación o problema.	<b>Análisis del problema:</b> requiere que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle.
Pensamos en una o varias posibles soluciones.	<b>Diseño o desarrollo de algoritmos:</b> procedimiento paso a paso para solucionar el problema dado.
Aplicamos la solución que estimamos más adecuada.	<b>Resolución del algoritmo elegido en la computadora:</b> consiste en convertir el algoritmo en programa, ejecutarlo y comprobar que soluciona verdaderamente el problema.

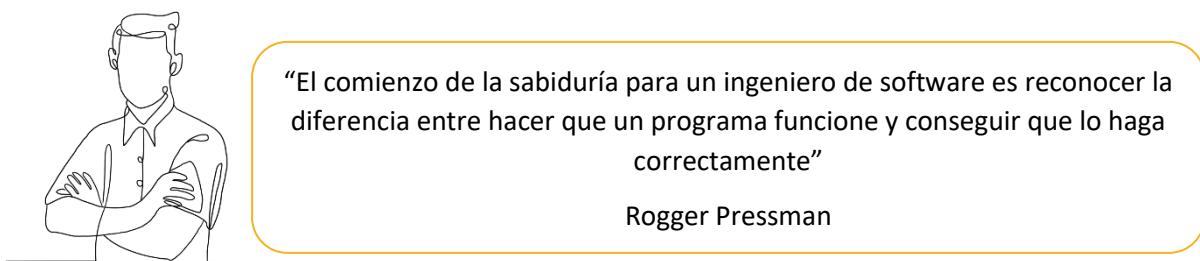
¿Qué virtudes debería tener nuestra solución?

- ✓ **Corrección y eficacia:** si resuelve el problema adecuadamente.
- ✓ **Eficiencia:** si lo hace en un tiempo mínimo y con un uso óptimo de los recursos del sistema.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Para conseguirlo, cuando afrontemos la construcción de la solución tendremos que tener en cuenta los siguientes conceptos:

1. **Abstracción:** se trata de realizar un análisis del problema para descomponerlo en problemas más pequeños y de menor complejidad, describiendo cada uno de ellos de manera precisa. **Divide y vencerás**, esta suele ser considerada una filosofía general para resolver problemas y de aquí que su nombre no sólo forme parte del vocabulario informático, sino que también se utiliza en muchos otros ámbitos.
2. **Encapsulación:** consiste en ocultar la información para poder implementarla de diferentes maneras sin que esto influya en el resto de elementos.
3. **Modularidad:** estructuraremos cada parte en módulos independientes, cada uno de ellos tendrá su función correspondiente.



### 1.1 ALGORITMOS Y PROGRAMAS

Después de analizar en detalle el problema a solucionar, hemos de diseñar y desarrollar el algoritmo adecuado. Pero, ¿Qué es un algoritmo?

**Algoritmo:** secuencia ordenada de pasos, descrita sin ambigüedades, que conducen a la solución de un problema dado.

Los algoritmos son independientes de los lenguajes de programación y de las computadoras donde se ejecutan. Un mismo algoritmo puede ser expresado en diferentes lenguajes de programación y podría ser ejecutado en diferentes dispositivos. Piensa en una receta de cocina, ésta puede ser expresada en castellano, inglés o francés, podría ser cocinada en fogón o vitrocerámica, por un cocinero o más, etc. Pero independientemente de todas estas circunstancias, el plato se preparará siguiendo los mismos pasos.

La diferencia fundamental entre algoritmo y **programa** es que, en el segundo, los pasos que permiten resolver el problema, deben escribirse en un determinado **lenguaje de programación** para que puedan ser ejecutados en el ordenador y así obtener la solución.

**Los lenguajes de programación** son sólo un medio para expresar el algoritmo y el ordenador un procesador para ejecutarlo. El diseño de los algoritmos será una tarea que necesitará de la creatividad y conocimientos de las técnicas de programación. Estilos distintos, de distintos programadores a la hora de obtener la solución del problema, darán lugar a algoritmos diferentes, igualmente válidos.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

En esencia, todo problema se puede describir por medio de un algoritmo y las características fundamentales que éstos deben cumplir son:

- ✓ Debe ser **preciso** e indicar el orden de realización paso a paso.
- ✓ Debe estar **definido**, si se ejecuta dos o más veces, debe obtener el mismo resultado cada vez.
- ✓ Debe ser **finito**, debe tener un número finito de pasos.

Pero cuando los problemas son complejos, es necesario descomponer éstos en subproblemas más simples y, a su vez, en otros más pequeños. Estas estrategias reciben el nombre de **diseño descendente** (*Metodología de diseño de programas, consistente en la descomposición del problema en problemas más sencillos de resolver.*) o **diseño modular (top-down design)** (*Metodología de diseño de programas, que consiste en dividir la solución a un problema en módulos más pequeños o subprogramas. Las soluciones de los módulos se unirán para obtener la solución general del problema.*). Este sistema se basa en el lema **divide y vencerás**.

Para representar gráficamente los algoritmos que vamos a diseñar, tenemos a nuestra disposición diferentes herramientas que ayudarán a describir su comportamiento de una forma precisa y genérica, para luego poder codificarlos con el lenguaje que nos interese. Entre otras tenemos:

- ✓ **Diagramas de flujo:** Esta técnica utiliza símbolos gráficos para la representación del algoritmo. Suele utilizarse en las fases de análisis.
- ✓ **Pseudocódigo:** Esta técnica se basa en el uso de palabras clave en lenguaje natural, constantes (Estructura de datos que se utiliza en los lenguajes de programación que no puede cambiar su contenido en el transcurso del programa.), variables (Estructura de datos que, como su nombre indica, puede cambiar de contenido a lo largo de la ejecución de un programa.), otros objetos, instrucciones y estructuras de programación que expresan de forma escrita la solución del problema. Es la técnica más utilizada actualmente.
- ✓ **Tablas de decisión:** En una tabla son representadas las posibles condiciones del problema con sus respectivas acciones. Suele ser una técnica de apoyo al pseudocódigo cuando existen situaciones condicionales complejas.

**Para saber más:** Ver vídeo sobre elementos gráficos fundamentales que se utilizan para la generación de un diagrama de flujo y ejemplo de uso de PSeInt.

[https://www.youtube.com/watch?v=tMEscFCEP0g&ab\\_channel=calichilo](https://www.youtube.com/watch?v=tMEscFCEP0g&ab_channel=calichilo)

[https://www.youtube.com/watch?v=nUnULP-suvA&ab\\_channel=ErickaZavala](https://www.youtube.com/watch?v=nUnULP-suvA&ab_channel=ErickaZavala)

El **ANEXO III** se trata el pseudocódigo, échale un vistazo.

## Autoevaluación

Rellena el hueco con el concepto adecuado:

A los pasos que permiten resolver el problema, escritos en un lenguaje de programación, para que puedan ser ejecutados en el ordenador y así obtener la solución, se les denomina: \_\_\_\_\_.

Envia

## 2. PARADIGMAS DE LA PROGRAMACIÓN.

### Caso práctico

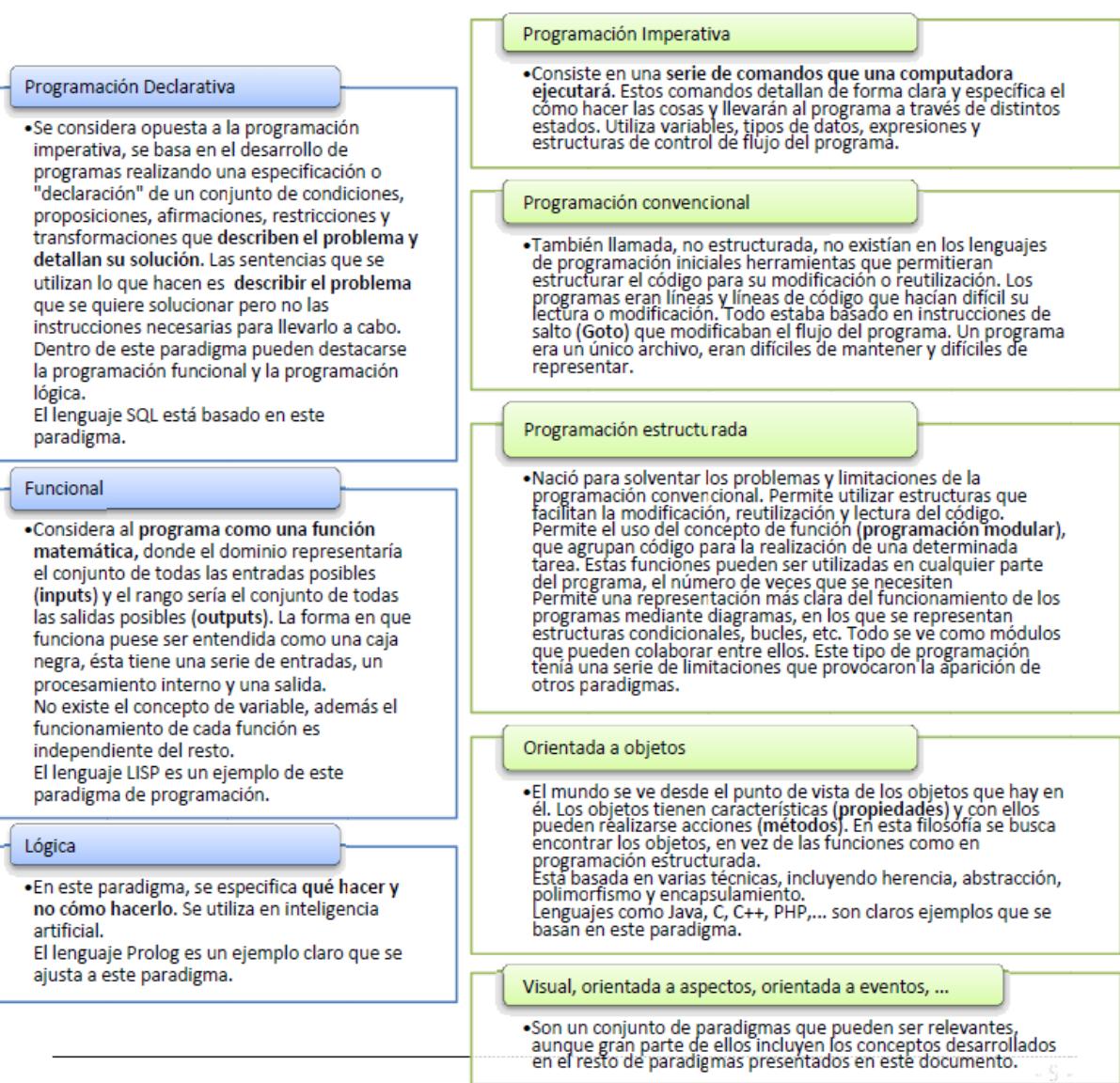
Ada comenta con Juan y María los distintos enfoques para el desarrollo de programas que han existido a lo largo de la historia de la programación, destacando que todos van a tener que “renovar” su forma de pensar, si quien en comenzar a utilizar un lenguaje moderno que les permita construir programas adaptados a las nuevas necesidades de sus clientes.

¿Cuántas formas existen de hacer las cosas? Supongo que estarás pensando: varias o incluso, muchas. Pero cuando se establece un patrón para la creación de aplicaciones nos estamos acercando al significado de la palabra paradigma.

**Paradigma de programación:** *Es un modelo básico para el diseño y la implementación de programas. Este modelo determinará cómo será el proceso de diseño y la estructura final del programa.*

# TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

## PARADIGMAS DE PROGRAMACIÓN



El paradigma representa un enfoque particular o filosofía para la construcción de software. Cada uno tendrá sus ventajas e inconvenientes, será más o menos apropiado, pero no es correcto decir que exista uno mejor que los demás.

Existen múltiples paradigmas, incluso puede haber lenguajes de programación que no se clasifiquen únicamente dentro de uno de ellos. Un lenguaje como [Smalltalk](#) es un lenguaje basado en el paradigma orientado a objetos. El lenguaje de programación [Scheme](#), en cambio, soporta sólo programación funcional. [Python](#), soporta múltiples paradigmas.

<https://javierleal.wordpress.com/2009/08/27/paradigmas-de-programacion/>

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

¿Cuál es el objetivo que se busca con la aplicación de los diferentes enfoques?

- ✓ Reducir la dificultad para el mantenimiento de las aplicaciones.
- ✓ Mejorar el rendimiento del programador.
- ✓ Mejorar la productividad y calidad de los programas.

### Autoevaluación

**¿En qué paradigma de programación podríamos enmarcar el lenguaje de programación Java?**

- Programación Estructurada.
- Programación Declarativa.
- Programación Orientada a Objetos.

## 3. FASES DE LA PROGRAMACIÓN

### Caso práctico

**Juan** pregunta a **Ada** cómo van a realizar todo el proceso de producción, y duda si el utilizar un nuevo lenguaje supondrá cambiar drásticamente los métodos aprendidos en el pasado.

**Ada** tranquiliza a **Juan** y a **María**: —Está claro que las fases principales que hemos estado llevando a cabo a lo largo de nuestros anteriores proyectos se seguirán aplicando, aunque con algunas diferencias. Lo más importante Juan, es que sigamos adecuadamente el método de trabajo para conseguir buenos resultados.

—¿Me costará mucho trabajo adaptarme? —pregunta **María**.

**Ada** le contesta sentándose a su lado: —No te preocupes **María**, se trata de adaptar conocimientos que ya tienes y aprender algunos otros.

Sea cual sea el estilo que escojamos a la hora de automatizar una determinada tarea, debemos realizar el proceso aplicando un método a nuestro trabajo. Es decir, sabemos que vamos a dar solución a un problema, aplicando una filosofía de desarrollo y lo haremos dando una serie de pasos que deben estar bien definidos.

El proceso de creación de software puede dividirse en diferentes fases:

- ✓ **Fase de resolución del problema.**
- ✓ **Fase de implementación.**
- ✓ **Fase de explotación y mantenimiento.**

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

A continuación, analizaremos cada una de ellas.

### 3.1. RESOLUCIÓN DEL PROBLEMA.

Para el comienzo de esta fase, es necesario que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle. A su vez, la fase de resolución del problema puede dividirse en dos etapas:

- a) **Análisis:** Por lo general, el análisis indicará la especificación de requisitos que se deben cubrir. Los contactos entre el analista/programador y el cliente/usuario serán numerosos, de esta forma podrán ser conocidas todas las necesidades que precisa la aplicación. Se especificarán los procesos y estructuras de datos que se van a emplear. La creación de prototipos será muy útil para saber con mayor exactitud los puntos a tratar.

El análisis inicial ofrecerá una idea general de lo que se solicita, realizando posteriormente sucesivos refinamientos que servirán para dar respuesta a las siguientes cuestiones:

- ✓ **¿Cuál es la información que ofrecerá la resolución del problema?** La respuesta a la primera pregunta se identifica con los resultados deseados o las salidas del problema.
- ✓ **¿Qué datos son necesarios para resolver el problema?** La respuesta a esta segunda pregunta indicará qué datos se proporcionan o las entradas del problema.

En esta fase debemos aprender a analizar la documentación de la empresa, investigar, observar todo lo que rodea el problema y recopilar cualquier información útil.

### Ejercicio resuelto

Vamos a ilustrar esta fase realizando el análisis del siguiente problema:

“Leer el radio de un círculo y calcular e imprimir su superficie y circunferencia.”

Está claro que las entradas de datos en este problema se reducen al radio del círculo, pero piensa ¿qué salidas de datos ofrecerá la solución?

**retroalim**

- b) **Diseño:** En esta etapa se convierte la especificación realizada en la fase de análisis en un diseño más detallado, indicando el comportamiento o la secuencia lógica de instrucciones capaz de resolver el problema planteado. Estos pasos sucesivos, que indican las instrucciones a ejecutar por la máquina, constituyen lo que conocemos como algoritmo.

Consiste en plantear la aplicación como una única operación global, e ir descomponiéndola en operaciones más sencillas, detalladas y específicas. En cada nivel de refinamiento, las operaciones identificadas se asignan a módulos separados.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Hay que tener en cuenta que antes de pasar a la implementación del algoritmo, hemos de asegurarnos que tenemos una solución adecuada. Para ello, todo diseño requerirá de la realización de la **prueba o traza** del programa. Este proceso consistirá en un seguimiento paso a paso de las instrucciones del algoritmo utilizando datos concretos. Si la solución aportada tiene errores, tendremos que volver a la fase de análisis para realizar las modificaciones necesarias o tomar un nuevo camino para la solución. Sólo cuando el algoritmo cumpla los requisitos y objetivos especificados en la fase de análisis se pasará a la fase de implementación.

### 3.2. IMPLEMENTACIÓN

Si la fase de resolución del problema requiere un especial cuidado en la realización del análisis y el posterior diseño de la solución, la fase de implementación cobra también una especial relevancia. Llevar a la realidad nuestro algoritmo implicará cubrir algunas etapas más que se detallan a continuación.

#### a) Codificación o construcción:

Esta etapa consiste en transformar o traducir los resultados obtenidos a un determinado lenguaje de programación. Para comprobar la calidad y estabilidad de la aplicación se han de realizar una serie de pruebas que comprueben las funciones de cada módulo (**pruebas unitarias**), que los módulos funcionen bien entre ellos (**pruebas de interconexión**) y que todos funcionan en conjunto correctamente (**pruebas de integración**).

Cuando realizamos la traducción del algoritmo al lenguaje de programación debemos tener en cuenta las reglas gramaticales y la sintaxis de dicho lenguaje. Obtendremos entonces el código fuente, lo que normalmente conocemos por programa.

Pero para que nuestro programa comience a funcionar, antes debe ser **traducido** a un lenguaje que la máquina entienda. Este proceso de traducción puede hacerse de dos formas, **compilando** o **interpretando** el código del programa.

Una vez traducido, sea a través de un proceso de compilación o de interpretación, el programa podrá ser ejecutado.

**Compilación:** es el proceso por el cual se traducen las instrucciones escritas en un determinado lenguaje de programación a lenguaje que la máquina es capaz de interpretar.

**Compilador:** programa informático que realiza la traducción. Recibe el código fuente, realiza un análisis lexicográfico, semántico y sintáctico, genera un código intermedio no optimizado, optimiza dicho código y finalmente, genera un código objeto para una plataforma específica.

**Intérprete:** programa informático capaz de analizar y ejecutar otros programas, escritos en un lenguaje de alto nivel. Los intérpretes se diferencian de los compiladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.

b) Prueba de ejecución y validación.

Para esta etapa es necesario implantar la aplicación en el sistema donde va a funcionar, debe ponerse en marcha y comprobar si su funcionamiento es correcto. Utilizando diferentes datos de prueba se verá si el programa responde a los requerimientos especificados, si se detectan nuevos errores, si éstos son bien gestionados y si la interfaz es amigable. Se trata de poner a prueba nuestro programa para ver su respuesta en situaciones difíciles.

Mientras se detecten errores y éstos no se subsanen no podremos avanzar a la siguiente fase. Una vez corregido el programa y testeado se documentará mediante:

Documentación interna: encabezados, descripciones, declaraciones del problema y comentarios que se incluyen dentro del código fuente.

Documentación externa: son los manuales que se crean para una mejor ejecución y utilización del programa, así como algunos tipos de diagramas que ayudan a comprender mejor la "arquitectura" de nuestra solución, como pueden ser los diagramas de flujo, diagrama de clases, etc.

## Autoevaluación

**Rellena los huecos con los conceptos adecuados:**

En la fase de codificación, hemos de tener en cuenta la [ ] del lenguaje para obtener el código fuente o programa. Posteriormente, éste deberá ser [ ] o [ ] para que pueda ser ejecutado posteriormente.

Envía

### 3.3. EXPLORACIÓN.

Cuando el programa ya está instalado en el sistema y está siendo de utilidad para los usuarios, decimos que se encuentra en **fase de explotación**.

Periódicamente será necesario realizar evaluaciones y, si es necesario, llevar a cabo modificaciones para que el programa se adapte o actualice a nuevas necesidades, pudiendo también corregirse errores no detectados anteriormente. Este proceso recibe el nombre de **mantenimiento** del software.

**Mantenimiento del software:** *es el proceso de mejora y optimización del software después de su entrega al usuario final. Involucra cambios al software en orden de corregir defectos y dependencias encontradas durante su uso, así como la adición de nuevas funcionalidades para mejorar su usabilidad y su aplicabilidad.*

Será imprescindible añadir una documentación adecuada que facilite al programador o a la programadora la comprensión, uso y modificación de dichos programas.

## 4. CICLOS DE VIDA DEL SOFTWARE

### Caso práctico

**María** le pregunta a **Juan**: —¿Juan, ¿qué ocurre cuando terminas un programa? ¿Se entrega al cliente y ya está? La verdad es que los programas que he hecho han sido para uso propio y no sé cómo termina el proceso con los clientes.

Contesta **Juan**: —Pues verás, cuando terminas un programa, o crees que lo has terminado, hay que llevar a cabo toda clase de pruebas para ver dónde puede fallar. Despues mejoras los posibles fallos y posteriormente se entrega al cliente, ahí es donde ves si tu software ha sido bien construido. El cliente lo utilizará y durante un tiempo puede ser que haya que arreglar alguna cosilla. Y cuando ya está todo correcto, en ocasiones, se establece un contrato de mantenimiento con el cliente. Comoves, desarrollar software no consiste sólo en programar y ya está.

Sean cuales sean las fases en las que realicemos el proceso de desarrollo de software, y casi independientemente de él, siempre se debe **aplicar un modelo de ciclo de vida**.

**Ciclo de vida del software:** *es una sucesión de estados o fases por las cuales pasa un software a lo largo de su "vida".*

El proceso de desarrollo puede involucrar siempre las siguientes etapas mínimas:



## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Existen varios tipos de ciclos de vida del software, a continuación, te mostramos un resumen de los más importantes.

<b>Modelo de cascada</b>	<p>Es también conocido por modelo clásico, modelo tradicional o modelo lineal secuencial. Para comenzar una fase ha de finalizarse la anterior. Es rígido y en la práctica presenta algunos problemas de aplicación. Es el más utilizado por su escasa complejidad.</p>	<div style="border: 1px solid black; padding: 5px;">         Definición de requerimientos          Análisis y diseño de software          Implementación y prueba de unidades          Integración y prueba del sistema          Operación y mantenimiento       </div> 
<b>Modelo por prototipos</b>	<p>Este ciclo de vida se basa en la creación de prototipos que irán mejorando el conocimiento del problema, tanto para el usuario como para los desarrolladores. La fase de especificación de requerimientos está compuesta por las susfases que se mencionan a la derecha</p>	<ul style="list-style-type: none"> <li>* Pequeño análisis y especificación.</li> <li>* Diseño y realización.</li> <li>* Evaluación.</li> <li>* Modificación.</li> <li>* Finalización de los requerimientos.</li> </ul>
<b>Modelo evolutivo</b>	<p>Se emplea para facilitar la creación de aplicaciones flexibles y escalables, que permitan incorporar modificaciones muy rápidamente una vez que se finalice su desarrollo</p> <p>Este modelo permite adaptarse a requisitos que varíen en el tiempo. Es un modelo iterativo, permiten desarrollar versiones cada vez más completas y complejas, hasta llegar al objeto final deseado; incluso evolucionar más allá, durante la fase de explotación</p>	<p>Los modelos iterativo incremental y espiral son del tipo evolutivo</p>
<b>Modelo incremental</b>	<p>El funcionamiento del modelo iterativo incremental permite la entrega de versiones parciales a medida que se va construyendo el producto final</p> <p>Por ejemplo, un procesador de texto podría incluir inicialmente funciones básicas. En un incremento posterior podría incorporar funciones para previsualización y paginación. En los siguientes incrementos podría disponer de funciones de corrección ortográfica, etc. Tras sucesivos incrementos lograríamos obtener el procesador de texto final</p>	
<b>Modelo espiral</b>	<p>Este modelo emplea lo mejor de los modelos convencional y por prototipos. Está dividido en cuatro fases. El paso por cada una de estas fases se repetirá tantas veces como sea necesario hasta que se cumplan todos los requerimientos del usuario</p>	<ol style="list-style-type: none"> <li>1. Planificación</li> <li>2. Análisis de riesgo</li> <li>3. Desarrollo</li> <li>4. Evaluación del cliente</li> </ol>

## 5. LENGUAJES DE PROGRAMACIÓN

### Caso práctico

**Ada** y **Juan** están recordando lo complejos que eran algunos lenguajes de programación, **Ada** comenta:  
—Cuando yo empecé en esto, había relativamente pocos lenguajes de programación y no permitían hacer programas como los que ahora desarrollamos.

**Juan** indica que él conoce las características generales de algunos lenguajes, pero que le gustaría saber algo más sobre los que hubo, hay y habrá.

**María** que asiente con la cabeza, piensa que aprender más sobre los lenguajes disponibles en la actualidad puede ayudar a la hora de elegir entre unos u otros.

Como hemos visto, en todo el proceso de resolución de un problema mediante la creación de software, después del análisis del problema y del diseño del algoritmo que pueda resolverlo, es necesario traducir éste a un lenguaje que exprese claramente cada uno de los pasos a seguir para su correcta ejecución. Este lenguaje recibe el nombre de **lenguaje de programación**.

**Lenguaje de programación:** conjunto de reglas sintácticas y semánticas, símbolos y palabras especiales establecidas para la construcción de programas. Es un lenguaje artificial, una construcción mental del ser humano para expresar programas.

**Gramática del lenguaje:** reglas aplicables al conjunto de símbolos y palabras especiales del lenguaje de programación para la construcción de sentencias correctas.

**Léxico:** es el conjunto finito de símbolos y palabras especiales, es el vocabulario del lenguaje.

**Sintaxis:** son las posibles combinaciones de los símbolos y palabras especiales. Está relacionada con la forma de los programas.

**Semántica:** es el significado de cada construcción del lenguaje, la acción que se llevará a cabo.

Hay que tener en cuenta que pueden existir sentencias sintácticamente correctas, pero semánticamente incorrectas. Por ejemplo, “Un avestruz dio un zarpazo a su cuidador” está bien construida sintácticamente, pero es evidente que semánticamente no.

Una característica relevante de los lenguajes de programación es, precisamente, que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos. A través de este conjunto se puede lograr la construcción de un programa de forma colaborativa.

Los lenguajes de programación pueden ser clasificados en función de lo cerca que estén del lenguaje humano o del lenguaje de los computadores. El lenguaje de los computadores son códigos binarios, es decir, secuencias de unos y ceros. Detallaremos seguidamente las características principales de los lenguajes de programación.

## 5.1. LENGUAJE MÁQUINA.

Este es el lenguaje utilizado directamente por el procesador, consta de un conjunto de instrucciones codificadas en binario. Es el sistema de códigos directamente interpretable por un circuito microprogramable (Dispositivo o conjunto de dispositivos de propósito general, que, según sea necesario, se programan para resolver distintos problemas).

Este fue el primer lenguaje utilizado para la programación de computadores. De hecho, cada máquina tenía su propio conjunto de instrucciones codificadas en ceros y unos. Cuando un algoritmo está escrito en este tipo de lenguaje, decimos que está en código máquina.

Programar en este tipo de lenguaje presentaba los siguientes inconvenientes:

- ✓ Cada programa era válido sólo para un tipo de procesador u ordenador.
- ✓ La lectura o interpretación de los programas era extremadamente difícil y, por tanto, insertar modificaciones resultaba muy costoso.
- ✓ Los programadores de la época debían memorizar largas combinaciones de ceros y unos, que equivalían a las instrucciones disponibles para los diferentes tipos de procesadores.
- ✓ Los programadores se encargaban de introducir los códigos binarios en el computador, lo que provocaba largos tiempos de preparación y posibles errores.

A continuación, se muestran algunos códigos binarios equivalentes a las operaciones de suma, resta y movimiento de datos en lenguaje máquina.

**Algunas operaciones en  
lenguaje máquina.**

Operación	Lenguaje máquina
SUMAR	00101101
RESTAR	00010011
MOVER	00111010

Dada la complejidad y dificultades que ofrecía este lenguaje, fue sustituido por otros más sencillos y fáciles utilizar. No obstante, hay que tener en cuenta que todos los programas para poder ser ejecutados, han de traducirse siempre al lenguaje máquina que es el único que entiende la computadora.

**Para saber más:** Como recordatorio, te proponemos ver un vídeo sobre cómo funciona el sistema binario.

[https://www.youtube.com/watch?v=cJhy9JutK\\_4&t=1s&ab\\_channel=algodemates](https://www.youtube.com/watch?v=cJhy9JutK_4&t=1s&ab_channel=algodemates)

## Autoevaluación

**Rellena el hueco con el concepto adecuado:**

En el lenguaje máquina de algunos procesadores, la combinación 00101101 equivale a la operación de  .

Envia

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

### 5.2. LENGUAJE ENSAMBLADOR.

La evolución del lenguaje máquina fue el lenguaje ensamblador.

Las instrucciones ya no son secuencias binarias, se sustituyen por códigos de operación que describen una operación elemental del procesador. Es un lenguaje de bajo nivel, al igual que el lenguaje máquina, ya que dependen directamente del hardware donde son ejecutados.

**Nemotécnico:** son palabras especiales, que sustituyen largas secuencias de ceros y unos, utilizadas para referirse a diferentes operaciones disponibles en el juego de instrucciones que soporta cada máquina en particular.

En ensamblador, cada instrucción (mnemotécnico) se corresponde a una instrucción del procesador. En la siguiente tabla se muestran algunos ejemplos.

**Algunas operaciones y su mnemotécnico en lenguaje Ensamblador.**

Operación	Lenguaje Ensamblador
MULTIPLICAR	MUL
DIVIDIR	DIV
MOVER	MOV

En el siguiente gráfico puedes ver parte de un programa escrito en lenguaje ensamblador. En color rojo se ha resaltado el código máquina en hexadecimal (Sistema numéricico en base 16, esto significa que contiene 16 símbolos únicos para representar datos: los números del 0 al 9 y las letras de la A a la F.), en magenta el código escrito en ensamblador y en azul, las direcciones de memoria donde se encuentra el código.

The image shows a hex dump of assembly code. The left column lists memory addresses from 0CFD:0100 to 0CFD:0140. The right column shows the assembly code and its corresponding machine code bytes. The assembly code includes instructions like MOV, INT, and CD21. The machine code is shown in hexadecimal. A red box highlights the first few lines of assembly code, and a blue box highlights the first few lines of machine code. To the right of the dump, there is a small note in Spanish: "Hola, este es un programa hecho en assembler para la Wikipedia\$".

-u 100 1a	0CFD:0100 BAOB01	MOV DX,010B
	0CFD:0103 B409	MOV AH,09
	0CFD:0105 CD21	INT 21
	0CFD:0107 B400	MOV AH,00
	0CFD:0109 CD21	INT 21
-d 10b 13f	0CFD:0100 20 65 73 74 65 20 65 73-20 75 6E 20 70 72 6F 67	48 6F 6C 61 2C
	0CFD:0110 72 61 6D 61 20 68 65 63-68 6F 20 65 6E 20 61 73	0A 65 6C 61 20 68 65 6E 20 61 73
	0CFD:0120 73 65 6D 62 6C 65 72 20-70 61 72 61 20 6C 61 20	73 65 6D 62 6C 65 72 20-70 61 72 61 20 6C 61 20
	0CFD:0130 57 69 6B 69 70 65 64 69-61 24	57 69 6B 69 70 65 64 69-61 24
	0CFD:0140	

Pero aunque ensamblador fue un intento por aproximar el lenguaje de los procesadores al lenguaje humano, presentaba **múltiples dificultades**:

- ✓ Los programas **seguían dependiendo directamente del hardware** que los soportaba.
- ✓ Los programadores **tenían que conocer detalladamente la máquina** sobre la que programaban, ya que debían hacer un uso adecuado de los recursos de dichos sistemas.
- ✓ La **lectura, interpretación o modificación** de los programas seguía presentando **dificultades**.

Todo programa escrito en lenguaje ensamblador necesita de un intermediario, que realice la traducción de cada una de las instrucciones que componen su código al lenguaje máquina correspondiente. Este intermediario es el programa ensamblador. El programa original escrito en lenguaje ensamblador constituye el código fuente y el programa traducido al lenguaje.

### 5.3. LENGUAJES COMPIADOS.

Para paliar los problemas derivados del uso del lenguaje ensamblador y con el objetivo de acercar la programación hacia el uso de un lenguaje más cercano al humano que al del computador, nacieron los **lenguajes compilados**.

Algunos ejemplos de este tipo de lenguajes son: **Pascal, Fortran, Algol, C, C++** (Es el lenguaje de programación C ampliado para poder utilizar los mecanismos que permitan la manipulación de objetos. Es un lenguaje multiparadigma.), etc.

Al ser lenguajes más cercanos al humano, también se les denomina **lenguajes de alto nivel**. Son más fáciles de utilizar y comprender, las instrucciones que forman parte de estos lenguajes utilizan palabras y signos reconocibles por el programador.

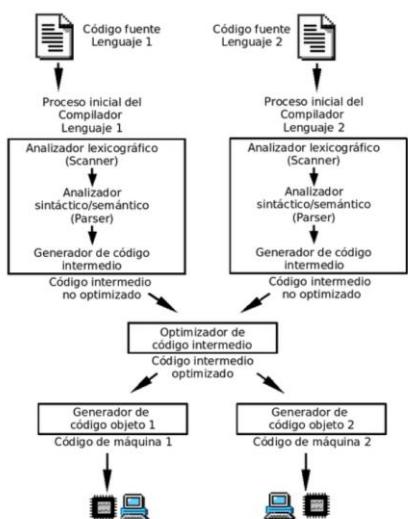
¿Cuáles son sus **ventajas**?

- ✓ Son **mucho más fáciles de aprender y de utilizar** que sus predecesores.
- ✓ Se **reduce el tiempo para desarrollar** programas, así como los **costes**.
- ✓ Son **independientes del hardware**, los programas pueden ejecutarse en diferentes tipos de máquina.
- ✓ La **lectura, interpretación y modificación** de los programas es **mucho más sencilla**.

Pero un programa que está escrito en un lenguaje de alto nivel también **tiene que traducirse** a un código que pueda utilizar la máquina. Los programas traductores que pueden realizar esta operación se llaman **compiladores**.

**Compiladores:** es un programa cuya función consiste en traducir el código fuente de un programa escrito en un lenguaje de alto nivel a lenguaje máquina. Al proceso de traducción se le conoce con el nombre de compilación.

Para ilustrar el proceso de compilación de programas te proponemos el siguiente esquema:



El compilador realizará la traducción y además informará de los posibles errores. Una vez subsanados, se generará el programa traducido a código máquina, conocido como código objeto. Este programa aún no podrá ser ejecutado hasta que no se le añadan los módulos de enlace o bibliotecas, durante el proceso de enlazado. Una vez finalizado el enlazado, se obtiene el código ejecutable.



## Para saber más

En el siguiente documento puedes consultar información sobre el proceso de enlazado:

[Documento sobre el proceso de enlazado](#) (92 KB)

[https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod\\_scorm/content/49/Compilacion.pdf](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod_scorm/content/49/Compilacion.pdf)



## Autoevaluación

Durante la fase de enlazado, se incluyen en el código fuente determinados módulos (bibliotecas) que son necesarios para que el programa pueda realizar ciertas tareas, posteriormente se obtendrá el código ejecutable.

- Verdadero  Falso

## 5.4. LENGUAJES INTERPRETADOS.

¿Recuerdas que en un apartado anterior ya hablamos de que existen dos formas de traducir los programas escritos en un lenguaje de alto nivel a código máquina y que una de esas formas es mediante un intérprete? Pues bien, ahora vamos a ver cuáles son las características de los lenguajes interpretados.

Se caracterizan por estar diseñados para que su ejecución se realice a través de un **intérprete**. Cada instrucción escrita en un lenguaje interpretado se analiza, traduce y ejecuta tras haber sido verificada. Una vez realizado el proceso por el intérprete, la instrucción se ejecuta, pero no se guarda en memoria.

**Intérprete:** es un programa traductor de un lenguaje de alto nivel en el que el proceso de traducción y de ejecución se llevan a cabo simultáneamente, es decir, la instrucción se pasa a lenguaje máquina y se ejecuta directamente. No se genera programa objeto, ni programa ejecutable.

Los programas en lenguajes interpretados presentan el inconveniente de ser algo más lentos que los compilados, ya que han de ser traducidos durante su ejecución. Por otra parte, necesitan disponer en la máquina del programa intérprete ejecutándose, algo que no es necesario en el caso de un programa compilado, para los que sólo es necesario tener el programa ejecutable para poder utilizarlo.

Ejemplos de lenguajes interpretados son: **Perl, PHP, Python, JavaScript**, etc.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

A medio camino entre los lenguajes compilados y los interpretados, existen los lenguajes que podemos denominar **pseudo-compilados** o **pseudo-interpretados**, es el caso del **Lenguaje Java**. Java puede verse como compilado e interpretado a la vez, ya que su código fuente se compila para obtener el código binario en forma de bytecodes, que son estructuras parecidas a las instrucciones máquina, con la importante propiedad de no ser dependientes de ningún tipo de máquina (se detallarán más adelante). La Máquina Virtual Java se encargará de interpretar este código y, para su ejecución, lo traducirá a código máquina del procesador en particular sobre el que se esté trabajando.

La máquina virtual java no es más que una aplicación que permite terminar de traducir los bytecodes al lenguaje máquina de la plataforma concreta en que se quiera ejecutar una aplicación java. Cualquier plataforma "se presenta al compilador de java" con el aspecto de una única máquina capaz de entender y ejecutar los bytecodes que se producen al compilar. Por tanto, la compilación se hace una sola vez, con independencia de la plataforma en la que se vayan a ejecutar los bytecodes. Por eso hablamos de máquina virtual, porque todas las plataformas "ofrecen el aspecto de una misma máquina" que en realidad no existe, ya que cada plataforma es diferente. Ese aspecto común lo proporciona un programa específico y distinto para cada plataforma, que es al que llamamos máquina virtual, que se encarga de interpretar y ejecutar esos bytecodes genéricos al lenguaje máquina concreto de la plataforma.



### Debes conocer

Puedes entender por qué Java es un lenguaje compilado e interpretado a través del siguiente enlace:

[El lenguaje Java es compilado e interpretado.](#)

<http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/introduccion/virtual.htm>

## 6. EL LENGUAJE DE PROGRAMACIÓN JAVA.

### Caso práctico

**Ada** indica a **Juan y María** que el lenguaje elegido para sus desarrollos va a ser Java.

La flexibilidad, facilidad de aprendizaje, similitud con algunos lenguajes que ya conocen y su capacidad para adaptarse a cualquier plataforma, hacen que sea ideal para producir las nuevas aplicaciones de **BK Programación**.

Java es un lenguaje sencillo de aprender, con una sintaxis parecida a la de C++, pero en la que se han eliminado elementos complicados y que pueden originar errores. Java es orientado a objetos, con lo que elimina muchas preocupaciones al programador y permite la utilización de gran cantidad de bibliotecas ya definidas, evitando reescribir código que ya existe. Es un lenguaje de programación creado para satisfacer nuevas necesidades que los lenguajes existentes hasta el momento no eran capaces de solventar.

Una de las principales virtudes de Java es su **independencia del hardware**, ya que el código que se genera es válido para cualquier plataforma. Este código será ejecutado sobre una máquina virtual **denominada Máquina Virtual Java** (MVJ o JVM – Java Virtual Machine), que interpretará el código convirtiéndolo a código específico de la plataforma que lo soporta. De este modo el programa se escribe una única vez y puede hacerse funcionar en cualquier lugar. Por eso, ese es el lema del lenguaje: “**Write once, run everywhere**”.

Antes de que apareciera Java, el lenguaje C era uno de los más extendidos por su versatilidad. Pero cuando los programas escritos en C aumentaban de volumen, su manejo comenzaba a complicarse. Mediante las técnicas de programación estructurada y programación modular se conseguían reducir estas complicaciones, pero no era suficiente.

Fue entonces cuando la Programación Orientada a Objetos (POO) entra en escena, aproximando notablemente la construcción de programas al pensamiento humano y haciendo más sencillo todo el proceso. Los problemas se dividen en objetos que tienen propiedades e interactúan con otros objetos, de este modo, el programador puede centrarse en cada objeto para programar internamente los elementos y funciones que lo componen.

Las características principales de lenguaje Java se resumen a continuación:

- ✓ El código generado por el compilador Java es independiente de la arquitectura.
- ✓ Está totalmente orientado a objetos.
- ✓ Su sintaxis es similar a C y C++.
- ✓ Es distribuido, preparado para aplicaciones TCP/IP. (Familia de protocolos de Internet. Es un conjunto de protocolos de red en los que se basa Internet y que permiten la transmisión de datos entre redes de ordenadores.)
- ✓ Dispone de un amplio conjunto de bibliotecas.
- ✓ Es robusto, realizando comprobaciones del código en tiempo de compilación y de ejecución.
- ✓ La seguridad está garantizada, ya que las aplicaciones Java no acceden a zonas delicadas de memoria o de sistema.



### Debes conocer

Obtén una descripción detallada de las características reseñadas anteriormente a través del siguiente artículo:

[Características detalladas del lenguaje Java](#)

## 6.1. BREVE HISTORIA

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a programar pequeños dispositivos electrónicos. La dificultad de estos dispositivos es que cambian continuamente y para que un programa funcione en el siguiente dispositivo aparecido, hay que rescribir el código. Por eso la empresa Sun quería crear un lenguaje **independiente del dispositivo**.

Pero no fue hasta 1995 cuando pasó a llamarse Java, dándose a conocer al público como lenguaje de programación para computadores. Java pasa a ser un lenguaje totalmente independiente de la plataforma y a la vez potente y orientado a objetos. Esta filosofía y su facilidad para crear aplicaciones para redes TCP/IP ha hecho que sea uno de los lenguajes más utilizados en la actualidad.

El factor determinante para su expansión fue la incorporación de un intérprete Java en la versión 2.0 del navegador Web Netscape Navigator, lo que supuso una gran revuelta en Internet. A principios de 1997 apareció **Java 1.1**, que proporcionó sustanciales mejoras al lenguaje.

**Java 1.2**, más tarde rebautizado como **Java 2**, nació a finales de 1998.

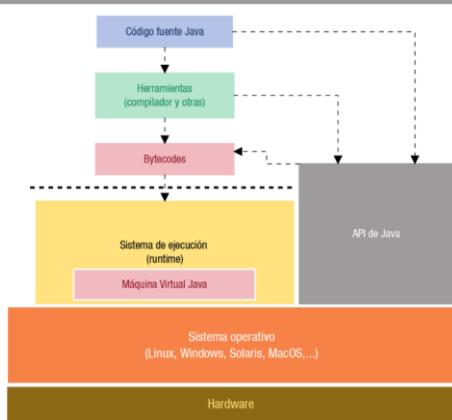
El principal objetivo del lenguaje Java es llegar a ser el **nexo universal** que conecte a los usuarios con la información, esté ésta situada en el ordenador local, en un servidor Web, en una base de datos o en cualquier otro lugar.

Para el desarrollo de programas en lenguaje Java es necesario utilizar un entorno de desarrollo denominado **JDK** (Java Development Kit), que provee de un compilador y un entorno de ejecución (**JRE** – Java Runtime Environment) para los bytecodes generados a partir del código fuente. Al igual que las diferentes versiones del lenguaje han incorporado mejoras, el entorno de desarrollo y ejecución también ha sido mejorado sucesivamente.

**Java 2** es la tercera versión del lenguaje, pero es algo más que un lenguaje de programación, incluye los siguientes elementos:

- ✓ Un lenguaje de programación: Java.
- ✓ Un conjunto de bibliotecas estándar que vienen incluidas en la plataforma y que son necesarias en todo entorno Java. Es el Java Core.
- ✓ Un conjunto de herramientas para el desarrollo de programas, como es el compilador de bytecodes, el generador de documentación, un depurador, etc.
- ✓ Un entorno de ejecución que en definitiva es una máquina virtual que ejecuta los programas traducidos a bytecodes.
- ✓ El siguiente esquema muestra los elementos fundamentales de la plataforma de desarrollo Java 2.

### ELEMENTOS DE LA PLATAFORMA DE DESARROLLO JAVA 2



El siguiente esquema muestra los elementos fundamentales de la plataforma de desarrollo Java 2.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Actualmente hay varias ediciones de la plataforma Java, que puedes ver en el siguiente enlace, y que de forma muy resumida podríamos clasificar en:

- ✓ **Java SE:** es la plataforma base para desarrollo de aplicaciones en Java. Es usado para desarrollar aplicaciones de escritorio, applets y otros tipos de aplicaciones. Es fundamental, dado que es la base sobre la que se cimientan el resto de plataformas Java.
- ✓ **Java EE:** es una plataforma de desarrollo para aplicaciones empresariales y del lado del servidor.
- ✓ **Java ME:** es una plataforma de desarrollo de aplicaciones Java para dispositivos móviles.



### Debes conocer

Lee atentamente el siguiente artículo, pues resume las ediciones de la plataforma Java de manera muy sencilla.

 [Java: lenguaje, plataforma, ediciones,...](#)

<https://www.campusmpv.es/recursos/post/Descifrando-Java-lenguaje-plataforma-ediciones-implementaciones.aspx>



### Para saber más

Si deseas conocer más sobre los orígenes del lenguaje Java, aquí te ofrecemos más información:

 [Las versiones de Java y su historia](#)

 [Historia de Java](#)

 [Línea de tiempo de la historia de Java desde julio de 2011](#)

<https://www.arquitecturajava.com/las-versiones-de-java/>

[https://es.wikipedia.org/wiki/Java\\_%28lenguaje\\_de\\_programaci%C3%B3n%29#Historia](https://es.wikipedia.org/wiki/Java_%28lenguaje_de_programaci%C3%B3n%29#Historia)

<https://www.java.com/releases/>

## 6.2. LA POO Y JAVA

En Java, los datos y el código (funciones o métodos) se combinan en entidades llamadas **objetos**. El objeto tendrá un **comportamiento** (su código interno) y un **estado** (los datos). Los objetos permiten la reutilización del código y pueden considerarse, en sí mismos, como piezas reutilizables en múltiples proyectos distintos. Esta característica permite reducir el tiempo de desarrollo de software.

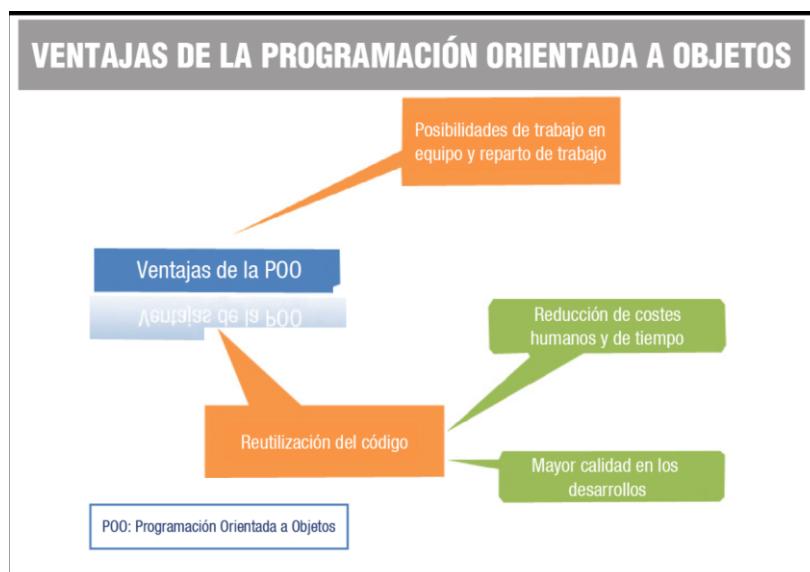
Por simplificar un poco las cosas, un programa en Java será como una representación teatral en la que debemos preparar primero cada personaje, definir sus características y qué va a saber hacer. Cuando esta fase esté terminada, la obra se desarrollará sacando personajes a escena y haciéndoles interactuar.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Al emplear los conceptos de la Programación Orientada a Objetos (POO), Java incorpora las tres características propias de este paradigma:

- ✓ **Encapsulación:** En programación modular y más específicamente en programación orientada a objetos, se denomina así al ocultamiento de los datos y elementos internos de un objeto. Sólo se puede modificar un objeto a través de las operaciones definidas para éste
- ✓ **Herencia:** Mecanismo que permite衍生 una clase de otra, de manera que extienda su funcionalidad.
- ✓ **Polimorfismo:** Capacidad para que varias clases derivadas de una antecesora utilicen un mismo método de forma diferente. Podemos crear dos clases distintas: Pez y Ave que heredan de la superclase Animal. La clase Animal tiene el método abstracto mover que se implementa de forma distinta en cada una de las subclases (peces y aves se mueven de forma distinta).

Los patrones o tipos de objetos se denominan clases (Es una construcción que se utiliza como un modelo (o plantilla) para crear objetos de ese tipo. El modelo describe el estado y el comportamiento que todos los objetos de la clase comparten) y los objetos que utilizan estos patrones o pertenecen a dichos tipos, se identifican con el nombre de instancias (Una instancia se produce con la creación de un objeto perteneciente a una clase (se dice que se instancia la clase). El objeto que se crea tiene los atributos, propiedades y métodos de la clase a la que pertenece. Los objetos y sus características se usan en la construcción de programas, ya sea como contenedores de datos o como partes funcionales del programa) Pero, no hay que alarmarse, estos conceptos se verán más adelante en sucesivas unidades.



Otro ejemplo para seguir aclarando ideas, piensa en los bloques de juegos de construcción. Suponemos que conoces los cubos de plástico en varios colores y tamaños. Por una de sus caras disponen de pequeños conectores circulares y en otra de sus caras pequeños orificios en los que pueden conectarse otros bloques, con el objetivo principal de permitir construir formas más grandes. Si usas diferentes piezas del lego puedes construir aviones, coches, edificios, etc. Si te fijas bien, cada pieza es un objeto pequeño que puede unirse con otros objetos para crear objetos más grandes.

Pues bien, aproximadamente así es como funciona la programación dirigida a objetos: unimos elementos pequeños para construir otros más grandes. Nuestros programas estarán formados por muchos componentes (objetos) independientes y diferentes; cada uno con una función determinada en nuestro software y que podrá comunicarse con los demás de una manera predefinida.

### 6.3. INDEPENDENCIA DE LA PLATAFORMA Y TRABAJO EN RED

Existen dos características que distinguen a **Java** de otros lenguajes, como son la **independencia de la plataforma** y la posibilidad de trabajar en red o, mejor, la posibilidad de **crear aplicaciones que trabajan en red**.

Estas características las vamos a explicar a continuación:

- c) **Independencia:** los programas escritos en Java pueden ser ejecutados en cualquier tipo de hardware. El código fuente se compila, generándose el código conocido como Java Bytecode (instrucciones máquina simplificadas que son específicas de la plataforma Java), el bytecode será interpretado y ejecutado en la **Máquina Virtual Java (MVJ o JVM – Java Virtual Machine)** que es un programa escrito en código nativo de la plataforma destino, entendible por el hardware. Con esto se evita tener que realizar un programa diferente para cada CPU (Unidad Central de Proceso) o plataforma. Por tanto, la parte que realmente es dependiente del sistema es la Máquina Virtual Java, así como las librerías o bibliotecas básicas que permiten acceder directamente al hardware de la máquina.
  
- d) **Trabajo en red:** esta capacidad del lenguaje ofrece múltiples posibilidades para la comunicación vía TCP/IP. Para poder hacerlo existen librerías que permiten el acceso y la interacción con protocolos como http, (Es el protocolo utilizado para la transacción de la World Wide Web. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.) ftp, (Es un protocolo de red para la transferencia de archivos entre sistemas conectados a una red TCP (Transmission Control Protocol), basado en arquitectura cliente-servidor.) etc., facilitando al programador las tareas del tratamiento de la información a través de redes.



#### Autoevaluación

¿Qué elemento es imprescindible para que una aplicación escrita en Java pueda ejecutarse en un ordenador?

- Que disponga de conexión a Internet y del hardware adecuado.
- Que tenga instalado un navegador web y conexión a Internet.
- Que tenga la Máquina Virtual Java adecuada instalada.

## 6.4. SEGURIDAD Y SIMPLICIDAD.

Junto a las características diferenciadoras del lenguaje Java relacionadas con la independencia y el trabajo en red, han de destacarse dos virtudes que hacen a este lenguaje uno de los más extendidos entre la comunidad de programadores: su **seguridad y su simplicidad**.

- ✓ **Seguridad:** en primer lugar, los posibles accesos a zonas de memoria “sensibles” que en otros lenguajes como C y C++ podían suponer peligros importantes, se han eliminado en Java.

Y en segundo lugar, el código Java se comprueba y verifica para evitar que determinadas secciones del código produzcan efectos no deseados. Los test que se aplican garantizan que las operaciones, operandos, conversiones, uso de clases y demás acciones son seguras.

En definitiva, podemos afirmar que Java es un lenguaje seguro.

- ✓ **Simplicidad:** aunque Java es tan potente como C o C++, es bastante más sencillo. Posee una curva de aprendizaje muy rápida y, **para alguien que comienza a programar en este lenguaje**, como será el caso de la mayoría de quienes comienzan a estudiar este módulo, le resulta relativamente fácil comenzar a escribir aplicaciones interesantes.

Si has programado alguna vez en C o C++ encontrarás que Java te pone las cosas más fáciles, ya que se han eliminado: la aritmética de punteros, (Un puntero o apuntador es una variable que referencia una región de memoria; en otras palabras, es una variable cuyo valor es una dirección de memoria., los registros, la definición de tipos, la gestión de memoria, etc) los registros, la definición de tipos, la gestión de memoria, etc. Con esta simplificación se reduce bastante la posibilidad de cometer errores comunes en los programas. Un programador experimentado en C o C++ puede cambiar a este lenguaje rápidamente y obtener resultados en muy poco espacio de tiempo.

Muy relacionado con la simplicidad que aporta Java está la incorporación de un elemento muy útil como es el **Recolector de Basura (Garbage collector)**. Permite al programador liberarse de la gestión de la memoria y hace que ciertos bloques de memoria puedan reaprovecharse, disminuyendo el número de huecos libres (fragmentación de memoria). (La fragmentación es la memoria que queda desperdiciada al usar los métodos de gestión de memoria. Puede ser interna o externa)

Cuando realicemos programas, crearemos objetos, haremos que éstos interaccionen, etc. Todas estas operaciones requieren de uso de memoria del sistema, pero la gestión de ésta será realizada de manera transparente al programador. Todo lo contrario que ocurría en otros lenguajes. Podremos crear tantos objetos como solicitemos, pero nunca tendremos que destruirlos. El entorno de Java borrará los objetos cuando determine que no se van a utilizar más. Este proceso es conocido como **recolección de basura**, y simplifica tu trabajo al programar una barbaridad.



### Autoevaluación

Rellena los huecos con los conceptos adecuados:

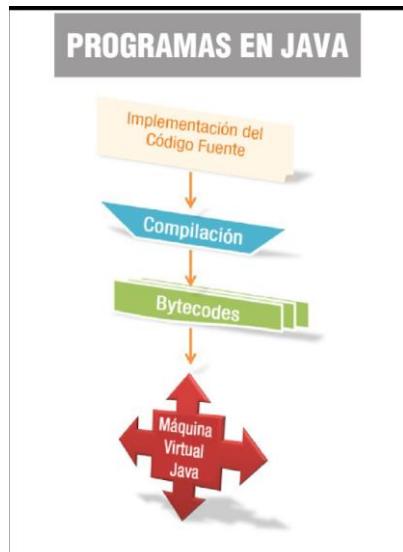
En Java se ha simplificado la gestión de memoria a través de la eliminación de la Aritmética de punteros, por lo que la incorporación del Garbage Collector evita que se produzca un crecimiento de los huecos libres en memoria, que recibe el nombre de fragmentación de memoria.

Enviar

## 6.5. JAVA Y LOS BYTECODES.

Un programa escrito en Java no es directamente ejecutable, es necesario que el código fuente sea interpretado por la Máquina Virtual Java. ¿Cuáles son los pasos que se siguen desde que se genera el código fuente hasta que se ejecuta? A continuación, se detallan cada uno de ellos.

Una vez escrito el código fuente (archivos con extensión .java), éste es precompilado generándose los códigos de bytes, Bytecodes o Java Bytecodes (archivos con extensión .class) que serán interpretados directamente por la Máquina Virtual Java y traducidos a código nativo de la plataforma sobre la que se esté ejecutando el programa.

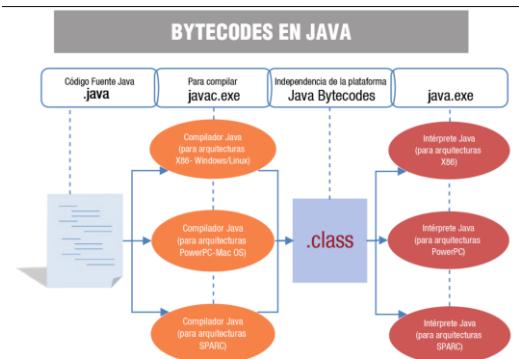


**Bytecode:** son un conjunto de instrucciones en lenguaje máquina que no son específicas para ningún procesador o sistema de cómputo. Un intérprete de código de bytes (bytecodes) para una plataforma específica será quien los ejecute. A estos intérpretes también se les conoce como Máquinas Virtuales Java o intérpretes Java de tiempo de ejecución.

En el proceso de precompilación, existe un verificador de códigos de bytes que se asegurará de que se cumplen las siguientes condiciones:

- ✓ El código satisface las especificaciones de la Máquina Virtual Java.
- ✓ No existe amenaza contra la integridad del sistema.
- ✓ No se producen desbordamientos de memoria.
- ✓ Los parámetros y sus tipos son adecuados.
- ✓ No existen conversiones de datos no permitidas.

Para que un bytecode pueda ser ejecutado en cualquier plataforma, es imprescindible que dicha plataforma cuente con el intérprete adecuado, es decir, la **máquina virtual específica para esa plataforma**. En general, la Máquina Virtual Java es un programa de reducido tamaño y gratuito para todos los sistemas operativos.



### Autoevaluación

En Java el código fuente es compilado, obteniéndose el código binario en forma de bytecodes. Pero, ¿cuál es la extensión del archivo resultante?

- Extensión .obj
- Extensión .class
- Extensión .java

## 7. PROGRAMAS EN JAVA

### Caso práctico

**Juan** celebra que **BK Programación** vaya a desarrollar sus programas en un lenguaje como Java. En algunas ocasiones ha asistido a congresos y ferias de exposiciones de software en las que ha podido intercambiar impresiones con compañeros de profesión sobre los diferentes lenguajes que utilizan en sus proyectos. Una gran mayoría destacaba lo fácil y potente que es programar en Java.

**Juan** está entusiasmado y pregunta:

—¿**Ada**, ¿cuándo empezamos? ¿Tienes código fuente para empezar a ver la sintaxis? ¿Podremos utilizar algún entorno de desarrollo profesional?

**Ada** responde sonriendo:

—¡Manos a la obra! **Maria**, ¿preparada? Vamos a echarle un vistazo a este fragmento de código...

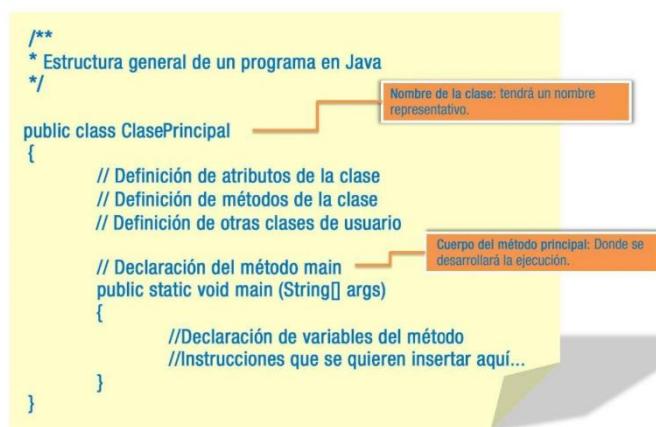
Hasta ahora, hemos descrito el lenguaje de programación Java, hemos hecho un recorrido por su historia y nos hemos instruido sobre su filosofía de trabajo, pero te preguntarás:

- ✓ ¿Cuándo empezamos a desarrollar programas?
- ✓ ¿Qué elementos forman parte de un programa en Java?
- ✓ ¿Qué se necesita para programar en este lenguaje?
- ✓ ¿Podemos crear programas de diferente tipo?

No te impacientes, cada vez estamos más cerca de comenzar la experiencia con el lenguaje de programación Java. Iniciaremos nuestro camino conociendo cuáles son los elementos básicos de un programa Java, la forma en que debemos escribir el código y los tipos de aplicaciones que pueden crearse en este lenguaje.

### 7.1. ESTRUCTURA DE UN PROGRAMA

En el gráfico al que puedes acceder a continuación, se presenta la estructura general de un programa realizado en un lenguaje orientado a objetos como es Java.



## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Vamos a analizar los elementos que aparecen en esa estructura:

- ✓ **public class ClasePrincipal:** todos los programas han de incluir elemento como éste. Podrá llamarse ClasePrincipal, ProgramaPrincipal, ClaseDePruebas, ProgramaDePruebas, Programa01 o como queramos. Pero debe tener un nombre. Se trata de una clase general en la que se incluyen todos los demás elementos del programa. En unidades posteriores veremos qué es una clase y cuáles son sus componentes principales. Por ahora es suficiente con que sepamos que nuestro programa va a comenzar con las líneas `public class NombrePrograma`, donde NombrePrograma será el nombre de nuestro programa.
- ✓ Dentro de este elemento principal observamos el método o función `main()` que contiene las líneas de código de nuestro programa. También veremos más adelante qué es un método. Baste por ahora saber que, al igual que en el caso anterior, nuestro programa debe contener también esas líneas `public static void main (String args[])`. Aquí dentro se podrán incluir las instrucciones que estimemos oportunas para la ejecución del programa.

Ten en cuenta que **Java distingue entre mayúsculas y minúsculas**. Si le das a la clase principal el nombre PrimerPrograma, el archivo .java tendrá como identificador **PrimerPrograma.java**, que es totalmente diferente a primerprograma.java. Además, para Java los elementos PrimerPrograma y primerprograma serían considerados dos clases diferentes dentro del código fuente.

Más adelante hablaremos de las convenciones que suelen seguirse para formar los nombres de las clases en Java, así como de cualquier otro identificador usado por el lenguaje.

- ✓ **Comentarios:** los comentarios se suelen incluir en el código fuente para realizar aclaraciones, anotaciones o cualquier otra indicación que el programador estime oportuna. Estos comentarios pueden introducirse de dos formas:
  - Con // estaríamos estableciendo una única línea completa de comentario, es decir, todo lo que hay detrás, hasta que haya un salto de línea, es comentario.
  - Con /\* \*/. De esta forma con /\* comenzaríamos el comentario y éste no terminaría hasta que no insertáramos \*/.
- ✓ **Bloques de código:** son conjuntos de instrucciones que se marcan mediante la apertura y cierre de llaves { }. El código así marcado es considerado interno al bloque.
- ✓ **Punto y coma (;):** aunque en el ejemplo de la imagen no hemos terminado ninguna línea de código con punto y coma, para no distraer de momento con los detalles, hay que hacer hincapié en que cada línea de código ha de terminar con punto y coma (;). En caso de no hacerlo, tendremos errores sintácticos.



### Autoevaluación

`public static void main (String[] args)` contiene las líneas de código de nuestro programa principal.

- Verdadero
- Falso

## 7.2. EL ENTORNO BÁSICO DE DESARROLLO JAVA.

Ya conoces cómo es la estructura de un programa en Java, pero, ¿qué necesitamos para llevarlo a la práctica?

La herramienta básica para empezar a desarrollar aplicaciones en Java es el **JDK (Java Development Kit o Kit de Desarrollo Java)**, que incluye un compilador y un intérprete para línea de comandos. Estos dos programas son los empleados en la precompilación e interpretación del código.

Como veremos, existen diferentes entornos para la creación de programas en Java que incluyen multitud de herramientas, pero por ahora nos centraremos en el entorno más básico, extendido y gratuito, el Java Development Kit (JDK). JDK es un entorno de desarrollo para construir aplicaciones, applets y componentes utilizando el lenguaje de programación Java. Incluye herramientas útiles para el desarrollo y prueba de programas escritos en Java y ejecutados en la Plataforma Java.

Así mismo, junto a JDK se incluye una implementación del entorno de ejecución Java, el **JRE (Java Runtime Environment)** para ser utilizado por el JDK. El JRE incluye la Máquina Virtual de Java (MVJ ó JVM – Java Virtual Machine), bibliotecas de clases y otros ficheros que soportan la ejecución de programas escritos en el lenguaje de programación Java.

Java fue creado por Sun Microsystems, posteriormente absorbida por Oracle, que ha ido lanzando las sucesivas versiones del JDK. Con el lanzamiento de Java 11 Oracle hizo un cambio de licencia de modo que se convirtió en una tecnología de pago en caso de usarlo en ciertas circunstancias. Podemos usar el JDK de Oracle o bien otras implementaciones abiertas, en nuestro caso trabajaremos con la implementación que vemos a continuación.

Para poder desarrollar nuestros primeros programas en Java sólo necesitaremos un editor de texto plano y los elementos que acabamos de instalar a través de Java SE.



### Para saber más

En el artículo que tienes a continuación puedes leer más sobre el cambio de licencia de Oracle con el JDK 11.

[Cambio de licencia](#)

<https://www.muylinux.com/2019/03/19/java-11-open-source-gratuito/>



### Autoevaluación

Podemos desarrollar programas escritos en Java mediante un editor de textos y a través del JDK podremos ejecutarlos.

- Verdadero  Falso

Instalar AdoptOpenJDK en Linux: <https://youtu.be/H-tWycZkYfc>



**Debes conocer**

En el siguiente artículo se comenta cómo instalar OpenJDK en las principales distribuciones Linux: <https://www.muylinux.com/2019/05/24/installar-adoptopenjdk-principales-distribuciones/>

### 7.3. LA API DE JAVA.

Junto con el kit de desarrollo que hemos descargado e instalado anteriormente, vienen incluidas gratuitamente todas las bibliotecas de la API (Application Programming Interface – Interfaz de programación de aplicaciones) de Java, es lo que se conoce como **Bibliotecas de Clases Java**. Este conjunto de bibliotecas proporciona al programador paquetes de clases útiles para la realización de múltiples tareas dentro de un programa. Está organizada en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas semánticamente. (Referencia a los aspectos del significado, sentido o interpretación del significado de un determinado elemento, símbolo, palabra, expresión o representación formal. En principio cualquier medio de expresión (lenguaje formal o natural) admite una correspondencia entre expresiones de símbolos o palabras, y situaciones o conjuntos de cosas que se encuentran en el mundo físico o abstracto, que puede ser descrito por dicho medio de expresión.)

En décadas pasadas una biblioteca era un conjunto de programas que contenían cientos de rutinas (una rutina es un procedimiento o función bien verificados, en determinado lenguaje de programación). Las rutinas de biblioteca manejaban las tareas que todos o casi todos los programas necesitaban. El programador podía recurrir a esta biblioteca para desarrollar programas con rapidez.

Una biblioteca de clases es un conjunto de clases de programación orientada a objetos. Esas clases contienen métodos que son útiles para los programadores. En el caso de Java cuando descargamos el JDK obtenemos la biblioteca de clases API. Utilizar las clases y métodos de las API de Java reduce el tiempo de desarrollo de los programas. También, existen diversas bibliotecas de clases desarrolladas por terceros que contienen componentes reutilizables de software, y están disponibles a través de la Web.



#### Para saber más

Si quieras acceder a la información oficial sobre la API de Java, te proponemos el siguiente enlace (está en Inglés).

[Información oficial sobre la API de Java](https://docs.oracle.com/en/java/javase/index.html)

<https://docs.oracle.com/en/java/javase/index.html>



#### Autoevaluación

Indica qué no es la API de Java:

- Un entorno integrado de desarrollo.
- Un conjunto de bibliotecas de clases.
- Una parte del JDK, incluido en el Java SE.

## 7.4. AFINANDO LA CONFIGURACIÓN.

¿Y es necesario configurar la instalación del JDK que acabamos de hacer?

Para que podamos compilar y ejecutar ficheros Java es necesario que realicemos unos pequeños ajustes en la configuración del sistema. Vamos a indicarle dónde encontrar los ficheros necesarios para realizar las labores de compilación y ejecución, en este caso **javac.exe** y **java.exe**, así como las librerías contenidas en la API de Java y las clases del usuario.

**La variable PATH:** como aún no disponemos de un IDE (Integrated Development Environment - Entorno Integrado de Desarrollo) la única forma de ejecutar programas es a través de línea de comandos. Pero sólo podremos ejecutar programas directamente si la ruta hacia ellos está indicada en la variable PATH del ordenador. Es necesario que incluyamos la ruta hacia estos programas en nuestra variable PATH. Esta ruta será el lugar donde se instaló el JDK hasta su directorio **bin**.

Si has instalado AdoptJDK como se indicaba en el vídeo de instalación, no tienes que hacer nada. Si no fuera así, tendrías que indicar dónde está instalado Java. Por ejemplo, así se haría si se tuviera instalado jdk 8 de Oracle.



### Debes conocer

En el siguiente documento aprenderás cómo habría que configurar la variable PATH en Windows, aunque AdoptOpenJdk ya lo hace.

[Cómo configurar la variable PATH de Windows](#) (1.1 MB)

[https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod\\_scorm/content/49/PROG01\\_CONT\\_R21\\_Como\\_configurar\\_PATH.pdf](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod_scorm/content/49/PROG01_CONT_R21_Como_configurar_PATH.pdf)

La **variable CLASSPATH**: esta variable de entorno establece dónde buscar las clases o bibliotecas de la API de Java, así como las clases creadas por el usuario. Es decir, los ficheros .class que se obtienen una vez compilado el código fuente de un programa escrito en Java. Es posible que en dicha ruta existan directorios y ficheros comprimidos en los formatos zip o jar que pueden ser utilizados directamente por el JDK, conteniendo en su interior archivos con extensión **class**.

(Por ejemplo: C:\Program Files\AdoptOpenJDK\jdk-11.0.7.10-hotspot\bin)

Si no existe la variable **CLASSPATH** debes crearla, para modificar su contenido sigue el mismo método que hemos empleado para la modificación del valor de la variable PATH, anteriormente descrito. Ten en cuenta que la ruta que debes incluir será el lugar donde se instaló el JDK hasta su directorio lib.

(Por ejemplo: C:\Program Files\AdoptOpenJDK\jdk-11.0.7.10-hotspot\lib)



## Debes conocer

En el siguiente artículo aprenderás cómo se configura el CLASSPATH en Windows.

[Cómo configurar el CLASPATH de Windows](#)

<https://www.abrirllave.com/java/configurar-la-variable-de-entorno-classpath.php>



## Autoevaluación

¿Qué variable de sistema o de entorno debemos configurar correctamente para que podamos compilar directamente desde la línea de comandos nuestros programas escritos en lenguaje Java?

- CLASSPATH.
- PATH.
- Javac.exe.

## 7.5. CODIFICACIÓN, COMPILACIÓN Y EJECUCIÓN DE APLICACIONES.

Una vez que la configuración del entorno Java está completada y tenemos el código fuente de nuestro programa escrito en un archivo con extensión .java, la compilación de aplicaciones se realiza mediante el programa **javac** incluido en el software de desarrollo de Java.

Para llevar a cabo la compilación desde la línea de comandos, escribiremos:

`javac archivo.java`

Donde **javac** es el compilador de Java y **archivo.java** es nuestro código fuente.

El resultado de la compilación será un archivo con el mismo nombre que el archivo Java pero con la **extensión .class**. Esto ya es el archivo con el código en forma de bytecode, es decir, con el código precompilado. Si en el código fuente de nuestro programa figuraran más de una clase, veremos cómo al realizar la compilación se generarán tantos archivos con extensión .class como clases tengamos. Además, si estas clases tenían método **main()** podremos ejecutar dichos archivos por separado para ver el funcionamiento de dichas clases.

Para que el programa pueda ser ejecutado, siempre y cuando esté incluido en su interior el método **main()**, podremos utilizar el intérprete incluido en el kit de desarrollo.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

La ejecución de nuestro programa desde la línea de comandos podremos hacerla escribiendo:

```
java archivo
```

Donde `java` es el intérprete y `archivo` es el archivo con el código precompilado. Es **IMPORTANTE** aclarar que donde ponemos `archivo`, nos referimos al archivo ya precompilado en bytecodes, que tiene la extensión `.class`, pero **NO hay que poner dicha extensión** para ejecutar el archivo, sólo el nombre. Esto se ve más claro en las explicaciones del ejercicio resuelto.



### Ejercicio resuelto

Vamos a llevar a la práctica todo lo que hemos estado detallando a través de la creación, compilación y ejecución de un programa sencillo escrito en Java.

[Mostrar retroalimentación](#)

Observa el código que se muestra más abajo, seguro que podrás entender parte de él. Cópialo en un editor de texto, respetando las mayúsculas y las minúsculas. Puedes guardar el archivo con extensión `.java` en la ubicación que prefieras. Recuerda que el nombre de la clase principal (en el código de ejemplo `MiModulo`) por ser `public`, debe ser exactamente igual al del archivo con extensión `.java`. Si tienes esto en cuenta la aplicación podrá ser compilada correctamente y ejecutada.

`/** La clase MiModulo implementa una aplicación que simplemente imprime "Módulo profesional - Programación" en pantalla. */`

```
class MiModulo {  
  
    public static void main(String[] args) {  
        // Muestra la cadena de caracteres  
        System.out.println("Módulo profesional - Programación");  
    }  
}
```

Accede a la línea de comandos y teclea, teniendo como directorio activo la carpeta donde has guardado el archivo `.java` (y suponiendo que se ha establecido correctamente el valor para la variable de entorno `PATH`), el comando **para compilarlo**:

```
javac MiModulo.java
```

El compilador genera entonces un fichero de código de bytes: `MiModulo.class`. Si visualizas ahora el contenido de la carpeta verás que en ella está el archivo `.java` y uno o varios (depende de las clases que contenga el archivo con el código fuente) archivos `.class`.

Finalmente, **para realizar la ejecución** del programa debes utilizar la siguiente sentencia:

```
java MiModulo
```

Si todo ha ido bien, verás escrito en pantalla: "Módulo profesional – Programación".

### 7.5.1. Estandarización del código.

Cada vez que escribamos el código de un programa es importante que sigamos unas normas o estándares que nos ayuden a realizar ese proceso siempre de la misma manera. De este modo todos los programas tendrán una estructura similar y nos será mucho más sencillo localizar cada uno de sus componentes.

En el caso del lenguaje Java ya hemos visto la estructura general un programa. Basándonos en esa estructura general vamos a definir una plantilla a partir de la cual podríamos desarrollar nuestros propios programas sin tener que escribir de nuevo las partes comunes una y otra vez.

Nuestra plantilla podría tener la siguiente estructura:

1. **Declaración de la clase principal:** Aquí podríamos poner el **nombre de nuestro programa**. Es decir, en lugar de llamar a la clase *ClasePrincipal*, podríamos llamarla *Concurso*, *Contabilidad*, *Juego*, *Ejercicio1*, *EjercicioNadador*, *CalculoDeAreas*, etc. Esto es, algún nombre que nos dé una idea sobre la tarea que va a llevar a cabo el programa.
2. Método *main*, dentro del componente anterior (encerrado entre llaves). Este nombre no se puede cambiar (siempre será main). En su interior estarán las líneas de código de nuestro programa:
  1. **Declaración de constantes y variables.**
    1. Declaración de **constants**.
    2. Declaración de **variables de entrada**.
    3. Declaración de **variables de salida**.
    4. Declaración de **variables auxiliares**.
  2. **Cuerpo del programa.**
    1. **Entrada de datos.**
    2. **Procesamiento.**
    3. **Salida de resultados.**

Si seguimos siempre esa estructura, será muy fácil poder analizar nuestros programas, pues estarán divididos en distintos elementos con significado propio cada uno de ellos. Ahora bien, tampoco hay que ser excesivamente rígidos. Dependiendo de la naturaleza y la complejidad del programa es posible que en ocasiones que algunas de esas tres partes se fundan en una (la entrada y el procesamiento, o el procesamiento y la salida, o incluso las tres), especialmente cuando el programa sea interactivo y se sigan introduciendo datos a la vez que se va procesando e incluso devolviendo resultados.

Esa estructura escrita en lenguaje Java podría quedar más o menos así:

```
/*
 * Plantilla para programas de prueba
 */
import java.util.Scanner;

public class NombreProgramaJava {

    public static void main(String[] args) {
        //-----
        //      Declaración de variables
        //-----
    }
}
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
// Constantes

// Variables de entrada

// Variables de salida

// Variables auxiliares

// Clase Scanner para petición de datos de entrada
Scanner teclado= new Scanner (System.in);

//-----
//          Entrada de datos
//-----
System.out.println("PLANTILLA DE PROGRAMA ");
System.out.println("-----");
System.out.println(" ");

// Aquí vendrá nuestro código de entrada de datos

//-----
//          Procesamiento
//-----

// Aquí vendrá nuestro código de procesamiento

//-----
//          Salida de resultados
//-----
System.out.println ();
System.out.println ("RESULTADO");
System.out.println ("-----");
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
// Aquí vendrá nuestro código de salida de resultados

System.out.println ();
System.out.println ("Fin del programa.");
}

}
```

Es probable que aún no entiendas muchas de las sentencias que hay escritas en el programa. No te preocupes, ya las irás entendiendo. Mientras tanto simplemente tendrás que ir "rellenando" las partes que necesites para construir tu programa.

Aquí tienes el archivo java que puedes utilizar como plantilla para escribir tus programas a partir de ahora: archivo plantilla de programa Java (java - 1.52 KB).  
[https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod\\_scorm/content/49/PlantillaProgramaJava.java](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod_scorm/content/49/PlantillaProgramaJava.java)

```
/*
 * Plantilla para programas de prueba
 */
import java.util.Scanner;

public class PlantillaProgramaJava {
    public static void main(String[] args) {
        //-----
        // Declaración de variables
        //-----

        // Constantes

        // Variables de entrada

        // Variables de salida

        // Variables auxiliares

        // Clase Scanner para petición de datos de entrada
        Scanner teclado= new Scanner (System.in);

        //-----
        // Entrada de datos
        //-----
        System.out.println("PLANTILLA DE PROGRAMA ");
        System.out.println("-----");
        System.out.println(" ");

        //-----
        // Procesamiento
        //-----
    }
}
```

Cada vez que vayas a escribir un nuevo programa, puedes usar este código cambiando el nombre a la clase (en lugar de **PlantillaProgramaJava** le asignas el nombre del archivo (clase) que le hayas puesto con el editor de texto que estés utilizando) y completas con el código de las acciones que tengas que realizar en ese programa (declarar variables, pedir datos por teclado, realizar cálculos, mostrar información por pantalla, etc.). Ya iremos viendo poco a poco y con detalle cómo llevar a cabo cada una de esas acciones.

## 7.5.2. Solución a posibles problemas con la codificación de caracteres acentuados.

¿Te ha pasado que al probar a ejecutar un programa que debe escribir algún carácter acentuado, en su lugar aparecen caracteres extraños? ¿Alguna vez has visitado una web en la que todos los caracteres acentuados aparecen como un cuadradito negro con una interrogación dentro (Mar**?**a) o la ñ como Ñ± (EspaÑ±a)? ¿Cómo evitar el problema de los caracteres extraños al mostrar caracteres acentuados en Java?

Esto suele pasar cuando se intenta ejecutar un programa en Java que contiene caracteres acentuados, y trabajamos con la consola de comandos. En cualquier caso, no debéis preocuparos demasiado de estos fallos en este módulo, ya que la forma normal de programar en Java NO es usando el JDK "a secas" (aunque esté bien que se sepa que es posible), sino que se suele programar haciendo uso de un entorno de desarrollo como es el caso de NetBeans, que este tipo de problemas los "solventa" por defecto, usando la codificación adecuada, de forma que los acentos se ven correctamente, etc.



### Para saber más

A continuación, recomendar un enlace en el que explica muy bien a qué se debe el problema, por qué se produce y cómo solucionarlo en distintos entornos.

[Sobre las reglas de codificación o... ¿de dónde salen esos caracteres "raros"?](#)

<https://www.adictosaltrabajo.com/2009/09/08/characterencoding-native-2ascii/>

Una solución muy simple, y que suele funcionar en la mayoría de los casos consiste en guardar el archivo **.java**, compilarlo, e indicarle en el momento de ejecutar cuál es la codificación correcta que tiene que usar, forzando a que muestre la salida con esa codificación:

```
Java -Dfile.encoding=cp850 PROG_programa1
```

En este ejemplo, la página de códigos que usamos en nuestro sistema es la 850, y por eso ponemos **encoding=cp850**, pero puedes forzar a usar cualquier otra codificación que use tu sistema. Para averiguar cuál es la codificación de tu sistema, puedes usar desde la consola el comando chcp (son las iniciales de CHange CodePage, ya que sirve para mostrar y también para cambiar la codificación que usa tu equipo). Mira un ejemplo en la siguiente captura de pantalla:

```
Administrator: C:\Windows\system32\cmd.exe
D:\ejercicios_java>chcp
Página de códigos activa: 850
D:\ejercicios_java>
```

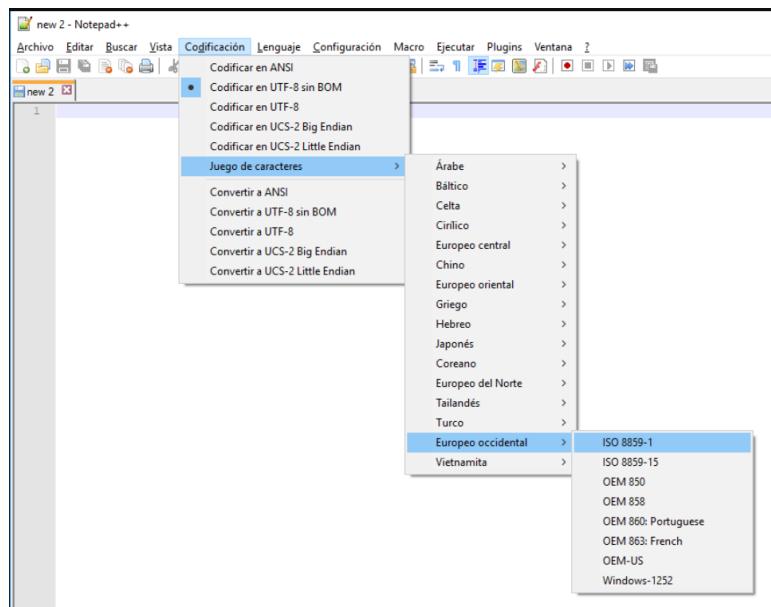
## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Como regla general, podemos seguir una serie de pasos para no tener problemas con las tildes (en entornos Windows). Es una manera alternativa para no tener que usar las opciones de forzado de uso de una cierta tabla de caracteres al ejecutar los comandos **javac** o **java**. Aunque esta solución implica que debemos saber de antemano qué juego de caracteres se usará allá donde vayamos a ejecutar nuestro programa. Para los primeros ejercicios donde sabemos que los vamos a hacer en la consola de comandos, nos puede valer esta "artimaña", aunque como ya se ha dicho, cuando utilicemos entornos de desarrollo (en nuestro caso NetBeans) podremos "olvidarnos" de estas cuestiones.

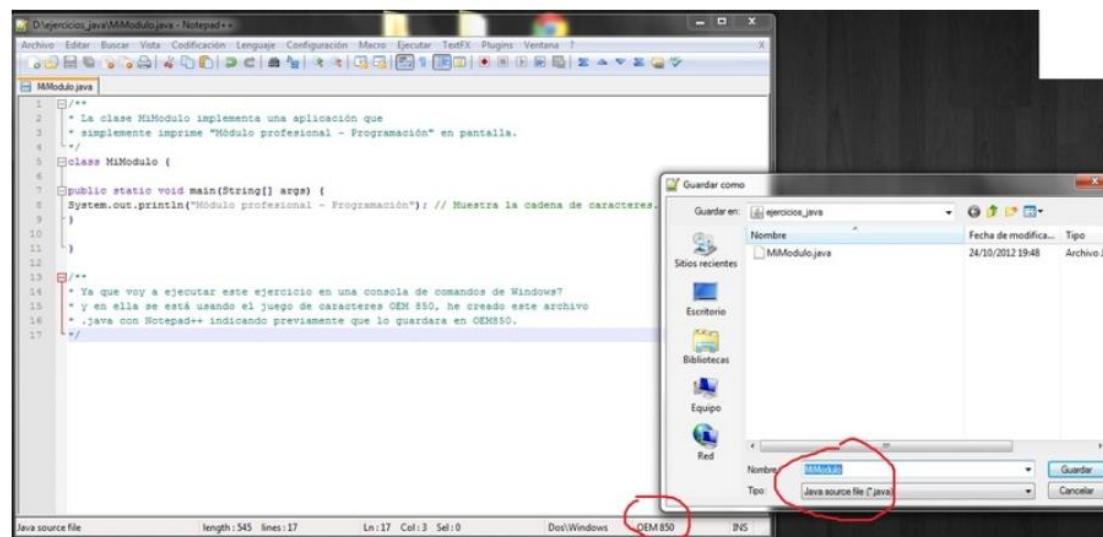
La consola de comandos es una pantalla del sistema operativo que proporciona un método diferente de "dar órdenes" al sistema operativo. Permite dar las instrucciones al Sistema operativo por medio de una línea de texto simple, que se introduce detrás del prompt (una señal que indica que el sistema está listo esperando a recibir la siguiente instrucción).

Los pasos a seguir son los siguientes:

- ✓ En la consola de Windows ejecutamos el comando **chcp** para que nos diga qué juego de caracteres está usando nuestro sistema. (Ver la captura de pantalla anterior).
- ✓ Con Notepad++ creo un archivo en blanco y antes de nada le indico que quiero usar el juego de caracteres de Europa occidental/ISO 8859-1.



Pego el código de mi programa y lo guardo como archivo Java source (**.java**)



## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

- ✓ Finalmente pasamos a compilar el **.java** y a ejecutar el **.class** y vemos en esta captura que se hace correctamente:

The screenshot shows a Windows Command Prompt window titled "Administrador: C:\Windows\system32\cmd.exe". The command line shows the following sequence of operations:

```
D:\ejercicios_java>dir
El volumen de la unidad D es DATOS
El n mero de serie del volumen es: AC13-CB98

Directorio de D:\ejercicios_java

24/10/2012 19:48    <DIR>      .
24/10/2012 19:48    <DIR>      ..
24/10/2012 19:48           539 MiModulo.java
               1 archivo      539 bytes
               2 dirs   110,161,199,104 bytes libres

D:\ejercicios_java>javac MiModulo.java

D:\ejercicios_java>dir
El volumen de la unidad D es DATOS
El n mero de serie del volumen es: AC13-CB98

Directorio de D:\ejercicios_java

24/10/2012 19:52    <DIR>      .
24/10/2012 19:52    <DIR>      ..
24/10/2012 19:52           445 MiModulo.class
24/10/2012 19:48           539 MiModulo.java
               2 archivos     984 bytes
               2 dirs   110,161,199,104 bytes libres

D:\ejercicios_java>java MiModulo
M dulo profesional - Programaci n

D:\ejercicios_java>
```

## 7.6. TIPOS DE APLICACIONES EN JAVA.

La versatilidad del lenguaje de programaci n Java permite al programador crear distintos tipos de aplicaciones. A continuaci n, describiremos las caracter sticas m s relevantes de cada uno de ellos:

- ✓ **Aplicaciones de consola:**
- Son programas independientes al igual que los creados con los lenguajes tradicionales.
  - Se componen como m nimo de un archivo **.class** que debe contar necesariamente con el m todo **main()**.
  - No necesitan un navegador web y se ejecutan cuando invocamos el comando **java** para iniciar la M quina Virtual de Java (JVM). De no encontrarse el m todo **main()** la aplicaci n no podr  ejecutarse.
  - Las aplicaciones de consola leen y escriben hacia y desde la entrada y salida est ndar, sin ninguna interfaz gr fica de usuario.
- ✓ **Aplicaciones gr ficas:**
- Aquellas que utilizan las clases con capacidades gr ficas, como **Swing**, que es la biblioteca para la interfaz gr fica de usuario avanzada de la plataforma Java SE.
  - Incluyen las instrucciones **import**, que indican al compilador de Java que las clases del paquete **javax.swing** se incluyan en la compilaci n.
- ✓ **Applets:**
- Son programas incrustados en otras aplicaciones, normalmente una p gina web que se muestra en un navegador. Cuando el navegador carga una web que contiene un applet, este se descarga en el navegador web y comienza a ejecutarse. Esto nos permite crear programas que cualquier usuario puede ejecutar con tan solo cargar la p gina web en su navegador.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

- Se pueden descargar de Internet y se observan en un navegador. Los applets se descargan junto con una página HTML desde un servidor web y se ejecutan en la máquina cliente.
- No tienen acceso a partes sensibles (por ejemplo: no pueden escribir archivos), a menos que uno mismo le dé los permisos necesarios en el sistema.
- No tienen un método principal.
- Son multiplataforma y pueden ejecutarse en cualquier navegador que soporte Java.

### ✓ Servlets:

- Son componentes de la parte del servidor de Java EE, encargados de generar respuestas a las peticiones recibidas de los clientes.
- Los servlets, al contrario de los applets, son programas que están pensados para trabajar en el lado del servidor y desarrollar aplicaciones Web que interactúen con los clientes.

### ✓ Midlets:

- Son aplicaciones creadas en Java para su ejecución en sistemas de propósito simple o dispositivos móviles. Algunos juegos Java creados para teléfonos móviles son midlets.
- Son programas creados para dispositivos embebidos (se dedican a una sola actividad), más específicamente para la máquina virtual Java Micro Edition (Java ME).
- Generalmente son juegos y aplicaciones que se ejecutan en teléfonos móviles.



### Autoevaluación

Un Applet es totalmente seguro, ya que no puede acceder, en ningún caso, a zonas sensibles del sistema. Es decir, no podría borrar o modificar nuestros archivos.

- Verdadero  Falso

## 8. ENTORNOS INTEGRADOS DE DESARROLLO (IDE)

### Caso práctico

**Ada, Juan y María** están navegando por Internet buscando información sobre herramientas que les faciliten trabajar en Java. **Ada** aconseja utilizar alguno de los entornos de desarrollo integrado existentes, ya que las posibilidades y rapidez que ofrecen aumentarían la calidad y reducirían el tiempo requerido para desarrollar sus proyectos.

**Juan**, que está chateando con un miembro de un foro de programadores al que pertenece, corrobora lo que **Ada** recomienda.

En los comienzos de Java la utilización de la línea de comandos era algo habitual. El programador escribía el código fuente empleando un editor de texto básico, seguidamente, pasaba a utilizar un compilador y con él obtenía el código compilado. En un paso posterior, necesitaba emplear una tercera herramienta para el ensamblado del programa. Por último, podía probar a través de la línea de comandos el archivo ejecutable. El problema surgía cuando se producía algún error, lo que provocaba tener que volver a iniciar el proceso completo.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Estas circunstancias hacían que el desarrollo de software no estuviera optimizado. Con el paso del tiempo, se fueron desarrollando aplicaciones que incluían las herramientas necesarias para realizar todo el proceso de programación de forma más sencilla, fiable y rápida. Para cada lenguaje de programación existen múltiples entornos de desarrollo, cada uno con sus ventajas e inconvenientes. Dependiendo de las necesidades de la persona que va a programar, la facilidad de uso o lo agradable que le resulte trabajar con él, se elegirá entre unos u otros entornos.

Para el lenguaje de programación Java existen múltiples alternativas, siendo los principales entornos de desarrollo **NetBeans** (que cuenta con el apoyo de la empresa Sun), **Eclipse** y **JCreator**. Los dos primeros son gratuitos, con soporte de idiomas y multiplataforma (Windows, Linux, MacOS).

¿Y cuál será con el que vamos a trabajar? El entorno que hemos seleccionado llevar a cabo nuestros desarrollos de software en este módulo profesional será **NetBeans**, al haber sido construido por la misma compañía que creó Java, ser de código abierto y ofrecer capacidades profesionales. Aunque, no te preocupes, también haremos un recorrido por otros entornos destacables.

### 8.1. ¿QUÉ SON?

Son aplicaciones que ofrecen la posibilidad de llevar a cabo el proceso completo de desarrollo de software a través de un único programa. Podremos realizar las labores de edición, compilación, depuración, detección de errores, corrección y ejecución de programas escritos en Java o en otros lenguajes de programación, bajo un entorno gráfico (no mediante línea de comandos). Junto a las capacidades descritas, cada entorno añade otras que ayudan a realizar el proceso de programación, como por ejemplo: código fuente coloreado, plantillas para diferentes tipos de aplicaciones, creación de proyectos, etc.

Hay que tener en cuenta que un entorno de desarrollo no es más que una fachada para el proceso de compilación y ejecución de un programa. ¿Qué quiere decir eso? Pues que si tenemos instalado un IDE (Entorno Integrado de Desarrollo) y no tenemos instalado el compilador, no tenemos nada.



#### Para saber más

Si deseas conocer algo más sobre lo que son los Entornos Integrados de Desarrollo (IDE) accede a las definiciones que te proponemos a continuación:

[Definición de Entorno Integrado de Desarrollo](#)

[Definición de Entorno Integrado de Desarrollo en Wikipedia](#)

[https://www.ecured.cu/IDE\\_de\\_Programaci%C3%B3n](https://www.ecured.cu/IDE_de_Programaci%C3%B3n)

[https://es.wikipedia.org/wiki/Entorno\\_de\\_desarrollo\\_integrado](https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado)

## 8.2. IDE ACTUALES.

Existen en el mercado multitud de entornos de desarrollo para el lenguaje Java, los hay de libre distribución, de pago, para principiantes, para profesionales, que consumen más recursos, que son más ligeros, más amigables, más complejos que otros, etc.

Entre los que son gratuitos o de libre distribución tenemos: **NetBeans, Eclipse, BlueJ, jGRASP, JCreator LE.**

Entre los que son propietarios o de pago tenemos: **IntelliJ IDEA, JCreator, JDeveloper.**



### Debes conocer

Cada uno de los entornos nombrados más arriba posee características que los hacen diferentes unos de otros, pero para tener una idea general de la versatilidad y potencia de cada uno de ellos, accede a la siguiente tabla comparativa:

[Comparativa entornos para Java](#)

[https://en.wikipedia.org/wiki/Comparison\\_of\\_integrated\\_development\\_environments#Java](https://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments#Java)

Pero, ¿cuál o cuáles son los más utilizados por la comunidad de programadores Java?

El puesto de honor se lo disputan entre **Eclipse, IntelliJ IDEA y NetBeans**.

En los siguientes epígrafes haremos una descripción de NetBeans y Eclipse, para posteriormente desarrollar los puntos claves del entorno NetBeans.



### Para saber más

Si quieras conocer la situación actual de uso y comparar diferentes entornos integrados de desarrollo para el lenguaje de programación Java, puedes ampliar datos en el siguiente artículo:

[Artículo sobre utilización de entornos Java.](#)

Para acceder a los lugares de Internet donde obtener los diferentes entornos integrados de desarrollo, puedes utilizar la lista que te ofrecemos en el anexo que hay al final de la unidad.

<https://www.bbvaapimarket.com/es/mundo-api/herramientas-basicas-para-los-desarrolladores-en-java/>



### Autoevaluación

¿Cuál de los siguientes entornos sólo está soportado en la plataforma Windows?

- Eclipse.
- IntelliJ IDEA.
- JCreator.

### 8.3. EL ENTORNO NETBEANS.

Como se ha indicado anteriormente, el entorno de desarrollo que vamos a utilizar a lo largo de los contenidos del módulo profesional será **NetBeans**. Por lo que vamos primero a analizar sus características y destacar las ventajas que puede aportar su utilización.

Se trata de un entorno de desarrollo orientado principalmente al lenguaje Java, aunque puede servir para otros lenguajes de programación. Es un producto libre y gratuito sin restricciones de uso. Es un proyecto de código abierto de gran éxito, con una comunidad de usuarios numerosa, en continuo crecimiento y apoyado por varias empresas.

El origen de este entorno hay que buscarlo en un proyecto realizado por estudiantes de la República Checa. Fue el primer IDE creado en lenguaje Java. Un tiempo más tarde, se formó una compañía que sería comprada en 1999 por Sun Microsystems (quien había creado el lenguaje Java). Poco después, Sun decidió que el producto sería libre y de código abierto y nació Netbeans como IDE de código abierto para crear aplicaciones Java.

NetBeans lleva tiempo pugnando con Eclipse por convertirse en la plataforma más importante para crear aplicaciones en Java y hasta el nombre (Eclipse) es toda una declaración de intenciones de hacerle la competencia a Oracle, (la empresa propietaria de NetBeans). Oracle Corporation adquirió Sun Microsystems en 2009. NetBeans sigue siendo software libre y ofrece las siguientes posibilidades:

- ✓ Escribir código en C, C++, Ruby, Groovy, Javascript, CSS y PHP además de Java.
- ✓ Permitir crear aplicaciones J2EE gracias a que incorpora servidores de aplicaciones Java (actualmente Glassfish y Tomcat).
- ✓ Crear aplicaciones Swing de forma sencilla, al estilo del Visual Studio de Microsoft.
- ✓ Crear aplicaciones **JME** para dispositivos móviles.

**Ruby:** Lenguaje de programación interpretado, reflexivo y orientado a objetos. Su implementación oficial es distribuida bajo una licencia de software libre.,

**Groovy:** Lenguaje de programación orientado a objetos implementado sobre la plataforma Java. Desde Groovy se puede acceder directamente a todas las API existentes en Java. El bytecode generado en el proceso de compilación es totalmente compatible con el generado por el lenguaje Java para la Java Virtual Machine (JVM), por tanto, puede usarse directamente en cualquier aplicación Java. Todo lo anterior unido a que la mayor parte de código escrito en Java es totalmente válido en Groovy hacen que este lenguaje sea de muy fácil adopción para programadores Java., Javascript, CSS y PHP además de Java.

**Javascript:** Lenguaje de programación interpretado. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web.

**CSS:** Hojas de estilo en cascada (en inglés Cascading Style Sheets), CSS es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML.

**PHP:** Hojas de estilo en cascada (en inglés Cascading Style Sheets), CSS es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML o XML.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

La versión que vamos a usar en este curso es la que tienes en el anexo de la programación docente.

Puedes consultar la hoja de ruta de desarrollos NetBeans mientras que lo desarrolló Oracle en el siguiente enlace: <https://netbeans.apache.org/download/archive/index.html>

Después de Oracle, el desarrollo de NetBeans pasó a manos de la fundación Oracle:

<https://netbeans.apache.org/>

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de Java escritas para interactuar con las API de NetBeans y un archivo especial (manifest file) que lo identifica como módulo.

Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en esta plataforma pueden ser extendidas fácilmente por cualquiera que desarrolle también software.

Cada módulo provee una función bien definida, tales como el soporte de Java, edición, o soporte para el sistema de control de versiones. NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiendo a la persona que va a realizar el programa comenzar a trabajar inmediatamente.



### Para saber más

Encuentra más información sobre esta plataforma en el enlace que te proponemos a continuación:

[Información oficial sobre NetBeans](https://netbeans.apache.org/)

## 8.4. INSTALACIÓN Y CONFIGURACIÓN.

Para realizar la instalación del entorno NetBeans, seguiremos los siguientes pasos básicos:

1. Descargar la versión deseada desde la web oficial.
2. Ejecutar el fichero de instalación. Más abajo hay un vídeo describiendo la instalación

Para llevar a cabo la descarga puedes dirigirte al siguiente enlace:

<https://netbeans.apache.org/download/index.html>

A continuación vemos cómo instalar el entorno de desarrollo NetBeans:

[https://www.youtube.com/watch?v=VWGL5k3\\_0Y4&ab\\_channel=CF\\_DAMAndaluc%C3%ADa](https://www.youtube.com/watch?v=VWGL5k3_0Y4&ab_channel=CF_DAMAndaluc%C3%ADa)

Y ya podemos empezar a crear programas. Por ejemplo, en el siguiente vídeo ejecutamos NetBeans y creamos un proyecto muy sencillo para crear un programa que escriba una frase por consola:

[https://www.youtube.com/watch?v=uYAZO64wleM&ab\\_channel=CF\\_DAMAndaluc%C3%ADa](https://www.youtube.com/watch?v=uYAZO64wleM&ab_channel=CF_DAMAndaluc%C3%ADa)

## 8.5. ASPECTO DEL ENTORNO Y GESTIÓN DE PROYECTOS.

La pantalla inicial de nuestro entorno de desarrollo ofrece accesos directos a las operaciones más usuales: aprendizaje inicial, tutoriales, ejemplos, demos, los últimos programas realizados y las novedades de la versión.

Para comenzar a describir el aspecto del entorno, es necesario crear un nuevo proyecto accediendo al menú **File - New Project**, indicaremos el tipo de aplicación que vamos a crear.



### Debes conocer

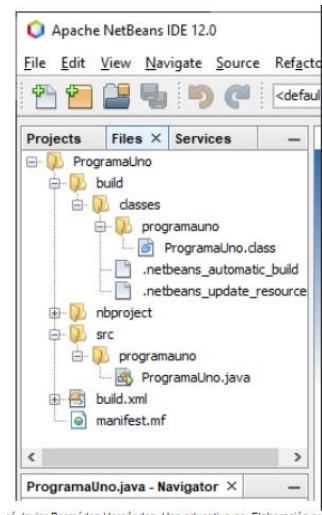
Para continuar con la creación de nuestro proyecto y la descripción del aspecto del entorno, accede al siguiente documento:

[Descripción del entorno NetBeans](#) (1.05 MB)

Cuando trabajemos con NetBeans, nuestros proyectos harán uso de clases para poder desarrollar las operaciones de nuestros programas. Estas clases se agruparán en paquetes y en el siguiente documento puedes aprender cómo se gestionan a través del entorno:

[Documento sobre paquetes](#) (696 KB)

Una de las ventajas que ofrece este entorno es poder examinar nuestros proyectos a través de la vista Archivos. Esta vista nos enseña la realidad de los archivos del proyecto, la carpeta build contiene los archivos compilados (.class), la carpeta src el código fuente y el resto, son archivos creados por Netbeans para comprobar la configuración del proyecto o los archivos necesarios para la correcta interpretación del código en otros sistemas (en cualquier caso no hay que borrarlos). Para activar esta vista, selecciona en el menú principal **Windows - Files**.



### Autoevaluación

Rellena los huecos con los conceptos adecuados:

En NetBeans, los archivos .class de un proyecto están alojados en la carpeta  y los .Java en la carpeta .

## 9. VARIABLES E IDENTIFICADORES.

### Caso práctico

**María** y **Juan** han comprobado que una aplicación informática es un trabajo de equipo que debe estar perfectamente coordinado. El primer paso es la definición de los datos y las variables que se van a utilizar.

Las decisiones que se tomen pueden afectar a todo el proyecto, en lo referente al rendimiento de la aplicación y ahorro de espacio en los sistemas de almacenamiento.

Después de la última reunión del equipo de proyecto ha quedado claro cuáles son las especificaciones de la aplicación a desarrollar. **Juan** no quiere perder el tiempo y ha comenzado a preparar los datos que va a usar el programa. Le ha pedido a **María** que vean juntos qué variables y tipos de datos se van a utilizar, Juan piensa que le vendrá bien como primera tarea para familiarizarse con el entorno de programación y con el lenguaje en sí.

Un programa maneja datos para hacer cálculos, presentarlos en informes por pantalla o impresora, solicitarlos al usuario, guardarlos en disco, etc. Para poder manejar esos datos, el programa los guarda en variables.

**Variable:** Una variable es un espacio en memoria principal del ordenador en donde el programa puede almacenar valores para utilizarlos posteriormente durante su ejecución. Es una zona en la memoria del computador que puede almacenar un valor de un determinado tipo para ser usado más tarde en el programa.

Una variable es una zona en la memoria del computador que puede almacenar un valor de un determinado tipo para ser usado más tarde en el programa. Las variables vienen determinadas por:

- ✓ Un **nombre**, que permite al programa acceder al valor que contiene en memoria. Debe ser un identificador válido. Por ejemplo, podría llamarse **precioVentaPublico**.
- ✓ Un **tipo de dato**, (Un tipo especifica la naturaleza del dato, esto es, el rango de valores que puede adquirir, así como las operaciones que se pueden llevar a cabo sobre esos valores) que especifica qué clase de información puede ser guardada por la variable en esa zona de memoria. Por ejemplo, puede ser de tipo entero, o de tipo cadena de caracteres, o de tipo real, etc. La variable anterior, si pretendemos que almacene precios, podría definirse como de tipo real, para que admita decimales para los céntimos. Cualquier lenguaje permitirá definir distintos tipos de reales. En Java, por ejemplo, float y double serían dos tipos de dato real.
- ✓ Un **rango de valores** que puede admitir dicha variable. Establece el valor máximo y mínimo que puede almacenarse en esa variable, así como la precisión. Normalmente va asociado al tipo de dato que puede almacenar. Siguiendo con el ejemplo, en Java, la diferencia entre definir una variable float o double supone que se podrán almacenar números reales más pequeños o más grandes y en la precisión (número de decimales máximo) que se pueden usar. Así, para un precio, con float tendremos suficiente rango, y suficiente precisión, ya que sólo necesitamos dos decimales, y ese tipo nos permite usar incluso más.

Al nombre que le damos a la variable se le llama **identificador**. Los identificadores permiten nombrar los elementos que se están manejando en un programa.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Vamos a ver con más detalle ciertos aspectos sobre los identificadores que debemos tener en cuenta.



### Para saber más

Bruce Eckel es el autor de los libros sobre Java y C++ (C orientado a objetos) dirigidos a programadores que desean aprender sobre estos lenguajes y sobre la programación orientada a objetos. Este escritor ha tenido la buena costumbre de editar sus libros para que puedan descargarse gratis. Así, podemos acceder de forma totalmente gratuita a la tercera edición de su libro "Thinking in Java" en el siguiente enlace (en inglés):

[Libro "Thinking in Java"](#) (1,5 MB)

A partir de ahora es conveniente que utilices algún manual de apoyo para iniciarte a la programación en Java. Te proponemos el de la serie de libros "Aprenda Informática como si estuviera en primero", de la Escuela Superior de Ingenieros Industriales de San Sebastián (Universidad de Navarra):

[Manual de apoyo sobre Java](#) (938 KB)

[https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod\\_scorm/content/49/PROG02\\_CONT\\_R03\\_X01\\_Java2.pdf](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod_scorm/content/49/PROG02_CONT_R03_X01_Java2.pdf)

## 9.1. IDENTIFICADORES

Un **identificador** en Java es una secuencia ilimitada sin espacios de letras y dígitos Unicode, (El estándar Unicode son unas normas de codificación de caracteres que utilizan de 8 a 32 bits para asignar un nombre y un código único a carácter o símbolo de cada posible lenguaje.) de forma que **el primer símbolo de la secuencia debe ser una letra, un símbolo de subrayado (\_) o el símbolo dólar (\$)**. Por ejemplo, son válidos los siguientes identificadores:



En la definición anterior decimos que un identificador es una secuencia ilimitada de caracteres Unicode. Pero... ¿qué es Unicode? Unicode es un código de caracteres o sistema de codificación, un alfabeto que recoge los caracteres de prácticamente todos los idiomas importantes del mundo. Las líneas de código en los programas se escriben usando ese conjunto de caracteres Unicode.

Esto quiere decir que en Java se pueden utilizar diversos sistemas de escritura como los alfabetos griego o árabe, los caracteres chinos o los kanjis japoneses. De esta forma, los programas están más adaptados a los lenguajes e idiomas locales, por lo que son más significativos y fáciles de entender tanto para los programadores que escriben el código, como para los que posteriormente lo tienen que interpretar, para introducir alguna nueva funcionalidad o modificación en la aplicación.

El estándar Unicode originalmente utilizaba 16 bits, pudiendo representar hasta 65.536 caracteres distintos, que es el resultado de elevar dos a la potencia dieciséis. Actualmente Unicode puede utilizar más o menos bits, dependiendo del formato que se utilice: **UTF-8 (siglas en inglés para Formato de Transformación Unicode 8), UTF-16 ó UTF-32. A cada carácter le corresponde únicamente un número entero perteneciente al intervalo de 0 a 2 elevado a n, siendo n el número de bits utilizados para representar los caracteres.** Por ejemplo, la letra ñ es el entero 164. Además, el código Unicode es “compatible” con el código ASCII, ya que para los caracteres del código ASCII, Unicode asigna como código los mismos 8 bits, a los que les añade a la izquierda otros 8 bits todos a cero. La conversión de un carácter ASCII a Unicode es inmediata.



## Recomendación

Una buena práctica de programación es seleccionar nombres adecuados para las variables, eso ayuda a que el programa se autodocumente, y evita un número excesivo de comentarios para aclarar el código.



## Para saber más

Enlace para acceder a la documentación sobre las distintas versiones de Unicode en la página web oficial del estándar:

[Documentación sobre Unicode](http://www.unicode.org/versions/)

<http://www.unicode.org/versions/>

## 9.2. CONVENIOS Y REGLAS PARA NOMBRAR VARIABLES.

A la hora de nombrar un identificador existen una serie de **normas de estilo** de uso generalizado, que se aceptan por convenio, aunque no son obligatorias, y que de hecho se usan en la mayor parte del código Java, lo que ayuda a entender más rápidamente la **semántica** de cada identificador. Estas reglas para la nomenclatura de variables son las siguientes:

- ✓ **Java distingue las mayúsculas de las minúsculas.** Por ejemplo, *Alumno* y *alumno* son variables diferentes.
- ✓ **No se suelen utilizar identificadores que comiencen con «\$» o «\_»,** además el símbolo del dólar, por convenio, no se utiliza nunca para identificadores que define el usuario (aunque lo usa internamente el lenguaje para crear ciertas clases que genera automáticamente, pero de eso no nos tenemos que ocupar ni preocupar al programar).
- ✓ **No se puede utilizar el valor booleano** (true o false) **ni el valor nulo (null).**
- ✓ **Los identificadores deben ser lo más descriptivos posibles.** Es mejor usar palabras completas en vez de abreviaturas crípticas. Así nuestro código será más fácil de leer y comprender. En muchos casos también hará que nuestro código se autodocumente. Por ejemplo, si tenemos que darle el nombre a una variable que almacena los datos de un cliente sería recomendable que la misma se llamara algo así como *FicheroClientes* o *ManejadorCliente*, y no algo poco descriptivo como Cl33., por más que Java lo considere correcto.

Además de estas restricciones, en la siguiente tabla puedes ver otras convenciones, que, no siendo obligatorias, sí son recomendables a la hora de crear identificadores en Java.

**Semántica:** En este contexto, el significado asociado a un identificador concreto, que nos permite saber si se trata de una constante, de una variable, si pertenece a una clase o a un objeto de esa clase, etc. Hay toda una serie de significados asociados a un identificador que se hacen explícitos si usamos correctamente los convenios de formación de identificadores.

# TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

## Convenciones sobre identificadores en Java

Identificador	Convención	Ejemplo
Nombre de variable.	Comienza por letra minúscula, y si está formado por varias palabras, se colocan juntas y todas las siguientes a partir de la segunda comenzarán por mayúsculas para ayudar a identificar visualmente dónde comienza cada nueva palabra a pesar de que no haya espacios.	numAlumnosMatriculados, suma
Nombre de constante.	Con todas sus letras en mayúsculas, separando las palabras con el guión bajo, y además por convenio el guión bajo no se utiliza en ningún otro sitio.	TAM_MAX, PI
Nombre de una clase.	Comienza por letra mayúscula.	String, MiTipo
Nombre de función o método.	Un nombre de una función o de un <b>método</b> comienza con letra minúscula. Sigue en realidad la misma nomenclatura que cualquier variable y se distingue que se trata de un <b>método</b> o función porque obligatoriamente debe ir seguido de paréntesis, que encierran la lista de <b>parámetros</b> que se le dan al método para trabajar. El paréntesis es obligado aunque no se le pase ningún parámetro.	modificaValor(), obtieneValor()

**Método:** Es un conjunto de instrucciones en un programa a las que se les da un nombre, de forma que en cualquier otra parte del programa donde aparezca ese nombre del método, el compilador se encarga de desplegar todas las instrucciones que lleva asociadas y las ejecuta en el orden que esté especificado en su interior.

**Parámetro:** Un valor que se le pasa a una función o un método para que haga algo con él. Siguiendo una analogía matemática, una función  $f(x)=2x$  establece que con el parámetro formal  $x$  que se le pasa debe hacer una operación matemática, multiplicándolo por dos, y devolviendo el resultado. Así, si ejecuto la función pasándole como parámetro actual un 3,  $f(3)$  multiplicará 2 por 3 y me dará como valor de vuelta un 6.



## Autoevaluación

Elige la opción correcta. Un identificador es una secuencia de uno o más símbolos Unicode que cumple que.... :

- Todos los identificadores han de comenzar con una letra, el carácter subrayado (\_) o el carácter dólar (\$).
- No puede incluir el carácter espacio en blanco.
- Puede tener cualquier longitud, no hay tamaño máximo.
- Todas las respuestas anteriores son correctas.



## Para saber más

Te recomendamos consultar los tutoriales que nos ofrece el propio lenguaje Java para resolver cualquier duda que te vaya surgiendo. En concreto, resulta interesante ir a las fuentes para ver las reglas de formación de identificadores para variables, etc. Por eso te recomendamos que veas el apartado Variables - Naming, del siguiente enlace al tutorial de Java sobre los conceptos básicos del lenguaje. Está en inglés, pero reiteramos que es muy importante que te vayas acostumbrando a consultar documentación técnica en inglés.

 [Nombrando variables.](#)

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html>

### 9.3. PALABRAS RESERVADAS.

Las palabras reservadas, a veces también llamadas palabras clave o keywords, son secuencias de caracteres formadas con letras ASCII cuyo uso se reserva al lenguaje y, por tanto, **no pueden utilizarse para crear identificadores**.

Las palabras reservadas en Java son:

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

Hay palabras reservadas que no se utilizan en la actualidad, como es el caso de const y goto, que apenas se utilizan en la actual implementación del lenguaje Java. Por otro lado, puede haber otro tipo de palabras o texto en el lenguaje que, aunque no sean palabras reservadas tampoco se pueden utilizar para crear identificadores. Es el caso de true y false que, aunque puedan parecer palabras reservadas, porque no se pueden utilizar para ningún otro uso en un programa, técnicamente son **literales booleanos**. Igualmente, null es considerado un literal, no una palabra reservada.

**Literal booleano:** es un valor posible para un tipo dado. Un literal booleano es un valor posible para variables de tipo booleano, por tanto. Y finalmente, un tipo booleano es el que puede tomar sólo dos valores distintos, verdadero o falso. En Java serían true o false

Cuando tras haber consultado la documentación de Java aún no tengas seguridad de cómo funciona alguna de sus características, pruébala en tu ordenador, y analiza cada mensaje de error que te dé el compilador para corregirlo. Busca en foros de Internet errores similares para ayudarte de la experiencia de otros usuarios y usuarias.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Normalmente, los editores y entornos integrados de desarrollo utilizan colores para diferenciar las palabras reservadas del resto del código, los comentarios, las constantes y literales, etc. De esta forma se facilita muchísimo la lectura del programa y la detección de errores de sintaxis. Dependiendo de la configuración del entorno se utilizarán unos colores u otros, es posible que los que utilice tu versión de NetBeans se parezcan a éstos:

- ✓ Las palabras reservadas en azul.
- ✓ Los comentarios aparecen en gris.
- ✓ Las variables dentro de una clase aparecen en verde.
- ✓ Los errores en rojo.
- ✓ Y el resto del código aparece en negro.

Puede que te interese cambiar los colores que utiliza NetBeans para la sintaxis de tus programas, por ejemplo, si quieras que los comentarios aparezcan en verde en lugar de en gris o que los destaque con algún tipo de sombreado, o aumentar el tamaño por defecto de la fuente si tienes problemas de visión, o indicar que tienes la autoría de los mismos, en lugar de que te aparezca el nombre de usuario del sistema operativo. En el siguiente documento puedes ver cómo se cambian los colores y las propiedades de usuario en un proyecto NetBeans:

[https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod\\_scorm/content/49/CambiarTexto.pdf](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod_scorm/content/49/CambiarTexto.pdf)

### 9.4. TIPOS DE VARIABLES I.

Dentro de un programa podemos encontrar distintos tipos de variables. Las diferencias entre unas variables y otras dependerán de diversos factores como, por ejemplo, el tipo de datos que representan, si su valor puede cambiar o no, o cuál es el papel que tienen en el programa.

Además, dependiendo del lenguaje de programación que se utilice, es posible que encontremos ciertas variaciones en los tipos de variables que se ofrecen. En el caso concreto del **lenguaje de programación Java** podemos distinguir los distintos tipos de variables según los siguientes criterios:

- a) Según el **tipo de información que contengan**, podemos hablar de **variables de tipos primitivos y variables referencia**. En función de a qué grupo pertenezca la variable, tipos primitivos o tipos referenciados, podrá tomar unos valores u otros, y se podrán definir sobre ella unas operaciones u otras.
- b) Dependiendo de su **mutabilidad**, podemos hablar de variables **mutables** (o **variables**) e **inmutables** (o **constantes**), es decir, según su valor pueda ser modificado o no durante la ejecución del programa.

En conclusión, podemos decir sobre una variable que:

1. Es un **almacén temporal de información** que usan los programas para registrar datos y operar con ellos;
2. Puede contener un valor de un **tipo primitivo** (entero, real, carácter, etc.) o bien una **referencia** (una dirección de memoria o "puntero") a una zona de memoria que contendrá información mucho más compleja que un simple valor en un tipo primitivo;
3. Se crea y usa **dentro de un bloque de código** (en el caso de Java, dentro de un método: elementos de una clase u objeto compuestos por una serie de sentencias que sirven para describir las acciones a realizar con esa clase u objeto);
4. Deja de existir (se "**destruye**") cuando la ejecución de ese bloque de código finaliza.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

**Tipos primitivos:** Son tipos de datos tan básicos que sin ellos apenas podría hacerse ningún programa, y que trae incorporados el lenguaje, incluso como palabras reservadas. Suelen ser los distintos tipos de datos numéricos (enteros cortos, enteros largos, reales,...) algún tipo carácter y el tipo lógico o booleano.

**Tipo referenciado:** Tipo de dato donde el valor de una variable de este tipo es una referencia (un puntero o dirección de memoria) hacia el valor real. En Java tenemos los arrays, las clases y los interfaces como tipos de datos referenciados.

El siguiente ejemplo puedes observar cómo se declaran algunas variables. En algunos casos se ha indicado un valor inicial y en otros no.

```
int cantidadLargos = 10;
double longitudPiscina;
char letraDni;
String nombre;
final int MAXIMO_NUMERO_PLANTAS= 12;
final double PI= 3.1415926536;
```

Analicemos cada una de las variables o elementos de memoria que se han declarado:

Descripción de variables

Nombre	Tipo	Mutabilidad	Primitivo/Referencia	Contenido	Descripción
cantidadLargos	entero (int)	variable	primitivo (un número entero)	valor entero 10	Se trata de una variable que puede cambiar de valor, que almacenará la cantidad de largos realizados por un nadador y cuyo valor inicial es 10.
longitudPiscina	real (double)	variable	primitivo (un número real)	todavía nada	Se trata de una variable que puede cambiar de valor, que almacenará la longitud de una piscina y que aún no tiene ningún valor asignado.
letraDni	carácter (char)	variable	primitivo (un carácter)	aún nada	Se trata de una variable que puede cambiar de valor, que almacenará la letra de un DNI.
nombre	cadena (String)	variable	referencia (puntero a una cadena)	aún (referencia o nula) nada vacía	Se trata de una variable que puede cambiar de valor, que almacena la dirección de memoria (referencia o "puntero") donde se encontrará una cadena de caracteres que representa el nombre de una persona.
MAXIMO_NUMERO_PLANTAS	entero (int)	constante	primitivo (un número entero)	valor entero 12	Se trata de una variable que una vez que se le asigne un primer valor ya no podrá cambiar y que almacena el máximo número de plantas que puede tener un edificio. Se le ha asignado el valor 12. Ese valor ya no se podrá cambiar.
PI	real (double)	constante	primitivo (un número real)	valor real 3.1415926536	Se trata de una variable que una vez que se le asigne un primer valor, ya no podrá cambiarse y que almacena el valor la constante geométrica $\pi$ (4,1415926536).

Como puedes observar se trata de pequeños casilleros donde se almacena cierta información básica (tipos **primitivos**) o bien una referencia (dirección de memoria) a información más compleja (tipos **referenciados**). En unos casos puede tratarse de información cambiante (**mutable**) y en otros constante (**inmutable**).

En Java para indicar que una variable es inmutable o constante hay que utilizar el modificador **final** en su declaración.

### 9.4.1. TIPOS DE VARIABLES II.

El siguiente ejemplo muestra el código para la creación de un par de variables:

- ✓ **Variable inmutable (*final*) llamada MAXIMO**, con valor 4.245. Al haber sido declarada como constante (*final*) no podrá cambiar su valor a lo largo de la vida del programa;
- ✓ **Variable mutable llamada y**. Sólo se podrá acceder a ella dentro del bloque de código donde ha sido declarada (método *main*), ya que fuera de él no existe. Podrá modificarse el valor que contiene tantas veces como se considere oportuno a lo largo de la vida del programa.

En apartados posteriores veremos cómo darle más funcionalidad a nuestros programas, mostrar algún resultado por pantalla, hacer operaciones, etc. Por el momento, si ejecutas el siguiente ejemplo simplemente NetBeans mostrará tres líneas indicando que este es el primer programa y el valor de la variable y de la constante declarada:

```
public class EjemploVariables {

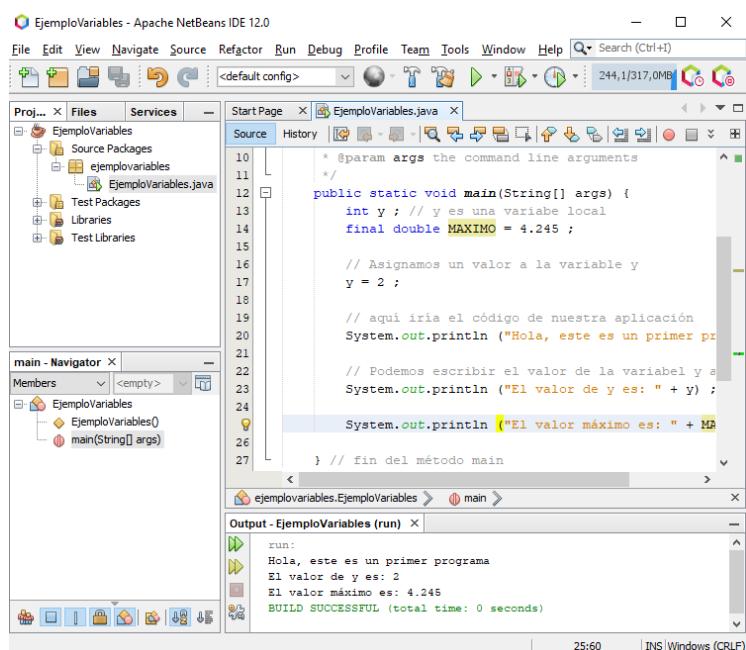
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int y ; // y es una variable local
        final double MAXIMO = 4.245 ;

        // Asignamos un valor a la variable y
        y = 2 ;

        // aquí iría el código de nuestra aplicación
        System.out.println ("Hola, este es un primer programa") ;

        // Podemos escribir el valor de la variable y así:
        System.out.println ("El valor de y es: " + y) ;

        System.out.println ("El valor máximo es: " + MAXIMO) ;
    }
}
```



La salida por pantalla del resultado de la ejecución de este programa sería algo similar a lo siguiente:

```
Hola, este es un primer programa
El valor de y es: 2
El valor máximo es: 4.245
```

En el siguiente documento presentación puedes ver cómo hemos creado este primer programa:  
[https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod\\_scorm/content/49/PROG01\\_CONT\\_CreacionPrimerPrograma.pdf](https://educacionadistancia.juntadeandalucia.es/formacionprofesional/pluginfile.php/19829/mod_scorm/content/49/PROG01_CONT_CreacionPrimerPrograma.pdf)

## 10. LOS TIPOS DE DATOS.

### Caso práctico

**María** ya ha hecho sus pinitos como programadora. Ahora mismo está ayudando a **Juan** con las variables y le ha surgido un problema. —El lenguaje se está comportando de una forma extraña, quiero llamar a una variable **final** y no me deja —comenta **María** a **Juan**. —Claro, eso es porque final es una palabra reservada y ya hemos visto que no la puedes utilizar para nombrar variables —le responde **Juan**. —¡Vaya, es verdad! ¿Y qué otros requisitos debo tener en cuenta a la hora de declarar las variables? —Pues lo importante es saber qué tipo de información hay que guardar, para poder asignarles el tipo de dato adecuado. ¿Tienes un momento y te lo cuento? — le dice **Juan**.

En los lenguajes fuertemente tipados, a todo dato (constante, variable o expresión) le corresponde un tipo que es conocido antes de que se ejecute el programa.

**Lenguajes fuertemente tipados:** Un lenguaje de programación es fuertemente tipado cuando una variable de un tipo de dato concreto, no se puede usar como si fuera una variable de otro tipo a menos que se haga una conversión de tipo. Si la variable es de tipo entero, por ejemplo, no podrá asignarle un valor real

El tipo limita el valor de la variable o expresión, las operaciones que se pueden hacer sobre esos valores, y el significado de esas operaciones. Esto es así porque **un tipo de dato no es más que una especificación de los valores que son válidos para esa variable, y de las operaciones que se pueden realizar con ellos**.

Debido a que el tipo de dato de una variable se comprueba durante la revisión que hace el compilador para detectar errores, o sea, en tiempo de compilación, esta labor es mucho más fácil, ya que no hay que esperar a que se ejecute el programa para saber qué valores va a contener esa variable. Esto se consigue con un control muy exhaustivo de los tipos de datos que se utilizan, lo cual tiene sus ventajas e inconvenientes. Por ejemplo, cuando se intenta asignar un valor de un tipo a una variable de otro tipo el compilador "se quejará" mostrando un mensaje de error. Sin embargo, en Java, puede haber **conversión entre ciertos tipos de datos**, como veremos posteriormente.

Ahora no es el momento de entrar en detalle sobre la conversión de tipos, pero sí debemos conocer con exactitud de qué tipos de datos dispone el lenguaje Java. Ya hemos visto que las variables, según la información que contienen, se pueden dividir en variables de tipos primitivos y variables referencia. Pero ¿qué es un tipo de dato primitivo? ¿Y un tipo referencia? Aunque lo hemos indicado brevemente antes, la primera vez que aparecían estos conceptos para que no te sonara totalmente a chino, esto es lo que vamos a ver a continuación con más detalle. Los tipos de datos en Java se dividen principalmente en dos categorías:

- ✓ **Tipos de datos sencillos o primitivos.** Representan valores simples que vienen predefinidos en el lenguaje; contienen valores únicos, como por ejemplo un carácter o un número.
- ✓ **Tipos de datos referencia.** Se definen con un nombre o referencia (**puntero**) que contiene la dirección en memoria de un valor o grupo de valores. Dentro de este tipo tenemos por ejemplo los vectores (también conocidos en informática como arrays), que son una serie de elementos del mismo tipo, o las clases, que son los modelos o plantillas a partir de los cuales se crean los objetos.

**Puntero:** En general un puntero no es más que una variable cuyo contenido no es el dato que nos interesa, sino que es una dirección de memoria donde está el dato que nos interesa. En Java las referencias son más o menos esto, de forma que cuando uso el nombre de una referencia para decirle al programa que haga algo, lo que hace el programa en realidad es ir a la zona de memoria donde está la variable referencia, buscar a qué dirección de memoria apunta, e ir a esa posición para leer el dato (o estructura de datos) que contenga y trabajar con esa información. Poco a poco irás entendiendo que esto tiene bastantes ventajas a la hora de gestionar la memoria que usa el programa.

En el siguiente apartado vamos a ver con detalle los diferentes tipos de datos que se engloban dentro de estas dos categorías.



### Autoevaluación

El tipado fuerte de datos supone que:

- A todo dato le corresponde un tipo que es conocido antes de que se ejecute el programa.
- El lenguaje hace un control muy exhaustivo de los tipos de datos.
- El compilador puede optimizar mejor el tratamiento de los tipos de datos.
- Todas las anteriores son correctas.

## 10.1. TIPOS DE DATOS PRIMITIVOS I.

Los tipos primitivos son aquellos datos sencillos que constituyen los tipos de información más habituales: números, caracteres y valores lógicos o booleanos. Al contrario que en otros lenguajes de programación orientados a objetos, **en Java no son objetos**.

Una de las mayores ventajas de tratar con tipos primitivos en lugar de con objetos, es que el compilador de Java puede optimizar mejor su uso. Otra importante característica, es que cada uno de los tipos primitivos tiene **idéntico** tamaño y comportamiento en todas las versiones de Java y para cualquier tipo de ordenador. Esto quiere decir que no debemos preocuparnos de cómo se representarán los datos en distintas plataformas, y asegura la **portabilidad** de los programas, a diferencia de lo que ocurre con otros lenguajes. Por ejemplo, el tipo **int** siempre se representará con 32 bits, con signo, y en el formato de representación complemento a 2 (Formato de representación de números enteros basado en el sistema de numeración binario.) en cualquier plataforma que soporte Java.



### Reflexiona

Java especifica el tamaño y formato de todos los tipos de datos primitivos con independencia de la plataforma o sistema operativo donde se ejecute el programa, de forma que el programador no tiene que preocuparse sobre las dependencias del sistema, y no es necesario escribir versiones distintas del programa para cada plataforma.



### Debes conocer

En el siguiente enlace se muestran los tipos de datos primitivos en Java con el rango de valores que pueden tomar, el tamaño que ocupan en memoria y sus valores por defecto.

[Tipos de datos primitivos en Java](#)

<http://blog.oscarscode.com/es/java-es/tipos-de-datos-primitivos-en-java/>

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Hay una peculiaridad en los tipos de datos primitivos, y es que el tipo de dato **char** es considerado por el compilador como un tipo numérico, ya que los valores que guarda son el código Unicode correspondiente al carácter que representa, no el carácter en sí, por lo que **puede operarse con caracteres** como si se tratara de números enteros.

Una cuestión importante: a la hora de elegir el tipo de dato que vamos a utilizar ¿qué criterio seguiremos para elegir un tipo de dato u otro?

Pues deberemos tener en cuenta **cómo es la información** que hay que guardar, si es de tipo texto, numérico, etc., y además **qué rango de valores** puede alcanzar. En este sentido, hay veces que, aunque queramos representar un número sin decimales, tendremos que utilizar datos de tipo real.

Por ejemplo, el tipo de dato **int** no podría representar la población mundial del planeta, ya que el valor máximo que alcanza es de 2.147.483.647, siendo éste el número máximo de combinaciones posibles con 32 bits, teniendo en cuenta que la representación de los números enteros en Java utiliza complemento a 2. Si queremos representar el valor correspondiente a la población mundial del planeta, cerca de 7.000.000.000 habitantes, tendríamos que utilizar al menos un tipo de dato **long**, o si tenemos problemas de espacio un tipo **float**. Sin embargo, los tipos reales tienen otro problema: la **precisión**. Vamos a ver más sobre ellos en el siguiente apartado.



### Para saber más

Si quieres obtener información sobre cómo se lleva a cabo la representación interna de números enteros y sobre la aritmética binaria, puedes consultar el siguiente enlace:

[Aritmética binaria](http://platea.pntic.mec.es/~lgonzale/tic/binarios/aritmetica.html)

<http://platea.pntic.mec.es/~lgonzale/tic/binarios/aritmetica.html>

### 10.1.1. Tipos de datos primitivos II.

¿Cómo representaremos los datos de tipo real en Java?

El tipo de dato real permite representar **números con decimales**. Al igual que ocurre con los enteros, la mayoría de los lenguajes definen más de un tipo de dato real, en función del número de bits usado para representarlos. Cuanto mayor sea ese número:

- ✓ **Más grande podrá ser** el número real representado en valor absoluto.
- ✓ **Mayor será la precisión** de la parte decimal.

Entre cada dos números reales cualesquiera, siempre tendremos matemáticamente hablando infinitos números reales, pero un ordenador no puede representar infinitos números, porque no dispone de capacidad ilimitada, por lo que la **mayoría de ellos los representaremos de forma aproximada**.

¡¡No fastidies!! ¿Tanta informática, tanto ordenador cada vez más potente, y ni siquiera puedo representar de forma exacta la mayoría de los números reales?

Justamente, sentimos decirte que así es, pero no se hunde el mundo por ello, y de una forma u otra, tenemos precisión suficiente para casi todo lo que nos propongamos, ya que como humanos, la "precisión infinita" tampoco la manejamos muy bien.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Por ejemplo, en la aritmética convencional, cuando dividimos 10 entre 3, el resultado es 3.333333..., con la secuencia de 3 repitiéndose infinitamente. En el ordenador sólo podemos almacenar un número finito de bits, por lo que el almacenamiento de un número real será siempre una aproximación.

Los números reales se representan en **coma flotante** o notación científica, que consiste en trasladar la coma decimal a la primera cifra significativa del valor, con objeto de poder representar el máximo de números posible.

Un número en el interior de un computador se expresa como:  $Valor = \text{mantisa} \cdot 2^{\text{exponente}}$

Donde la mantisa son las cifras significativas del número (pues hemos trasladado la coma decimal hasta el principio). De este modo, para almacenar el número, sólo se guardan la mantisa y el exponente al que va elevada la base. Los bits empleados por la mantisa representan la **precisión** del número real, es decir, el número de cifras decimales significativas que puede tener el número real, mientras que los bits del exponente expresan la diferencia entre el mayor y el menor número representable, lo que viene a ser el **intervalo de representación**.

En Java las variables de tipo **float** se emplean para representar los números en coma flotante de simple precisión de 32 bits, de los cuales 24 son para la mantisa y 8 para el exponente. La mantisa es un valor entre -1.0 y 1.0 y el exponente representa la potencia de 2 necesaria para obtener el valor que se quiere representar. Por su parte, las variables tipo **double** representan los números en coma flotante de doble precisión de 64 bits, de los cuales 53 son para la mantisa y 11 para el exponente.

La mayoría de los programadores en Java emplean el tipo **double** cuando trabajan con datos de tipo real. Es una forma de asegurarse de que los errores cometidos por las sucesivas aproximaciones sean menores. De hecho, Java considera los valores en coma flotante como de tipo **double** por defecto. Así, el literal **3.65** será considerado por defecto como un literal de tipo **double**. Si quiero que sea considerado como **float**, (que es de menor precisión, y ocupa menos espacio) tendré que indicarlo explícitamente añadiéndole una **f** detrás: **3.65f** o **3.65F**.

Con el objetivo de conseguir la máxima portabilidad de los programas, Java utiliza el estándar internacional **IEEE 754** (Formato de representación o normativa cuyo objetivo es estandarizar el uso de la representación de números en coma flotante entre distintos fabricantes. Está desarrollado por el IEEE (Institute of Electrical and Electronics Engineers, en español Instituto de Ingenieros Eléctricos y Electrónicos.) para la representación interna de los números en coma flotante, que es una forma de asegurarse de que el resultado de los cálculos sea el mismo para diferentes plataformas.



### Para saber más

La siguiente página en inglés es la web oficial sobre el estándar internacional IEEE 754-2008 para representación de números en coma flotante (Standard for Binary Floating-Point Arithmetic, edición de 2008 desarrollado por el IEEE: Institute of Electrical and Electronics Engineers, conocido como "IE cubo"):

[Notación IEEE 754](#)

[https://grouper.ieee.org/groups/msc/ANSI\\_IEEE-Std-754-2019/](https://grouper.ieee.org/groups/msc/ANSI_IEEE-Std-754-2019/)



## Autoevaluación

Relaciona los tipos primitivos con los bits y rango de valores correspondientes, escribiendo el número asociado en el hueco correspondiente.

### Ejercicio de relacionar

Tipo	Relación	Característica
short	<input type="checkbox"/>	Coma flotante de 64 bits, usando la representación IEEE 754-2008
byte	<input type="checkbox"/>	Entero de 32 bits, rango de valores de -2.147.483.648 (-2 <sup>31</sup> ) a 2.147.483.647 (+2 <sup>31</sup> -1)
double	<input type="checkbox"/>	Entero de 16 bits, rango de valores de -32.768 (-2 <sup>15</sup> ) a +32.767 (+2 <sup>15</sup> -1)
long	<input type="checkbox"/>	Coma flotante de 32 bits, usando la representación IEEE 745-2008
int	<input type="checkbox"/>	Entero de 8 bits, rango de valores de -128 (-2 <sup>7</sup> ) a +127 (+2 <sup>7</sup> -1)
float	<input type="checkbox"/>	Entero de 64 bits, rango de valores de -9.223.372.036.854.775.808 (-263) a 9.223.372.036.854.775.807 (+263-1)

## 10.2. DECLARACIÓN E INICIALIZACIÓN.

Llegados a este punto cabe preguntarnos ¿cómo se crean las variables en un programa? ¿Qué debo hacer antes de usar una variable en mi programa?

Pues bien, como podrás imaginar, debemos crear las variables antes de poder utilizarlas en nuestros programas, indicando qué nombre va a tener y qué tipo de información va a almacenar, en definitiva, debemos **declarar la variable**.

Las variables se pueden declarar en cualquier bloque de código, dentro de llaves. Y lo hacemos indicando su **identificador y el tipo de dato**, separadas por comas si vamos a declarar varias a la vez, por ejemplo:

```
int numeroAlumnos = 15;
double radio = 3.14, importe = 102.95;
```

De esta forma, estamos declarando **numeroAlumnos** como una variable de tipo **int**, (y al mismo tiempo asignándole el valor 15, pero de momento nos fijamos solo en la declaración: **int numeroAlumnos;**) y otras dos variables **radio** e **importe** de tipo **double**. No es obligatorio asignar valor a la vez que se declara, pero puede hacerse y en el ejemplo anterior hemos aprovechado la declaración de las variables para inicializarlas, asignándoles como primer valor **15, 3.14** y **102.95** respectivamente.

Si la variable va a permanecer inalterable a lo largo del programa, la declararemos como **constante**, utilizando la palabra reservada **final** de la siguiente forma:

```
final double PI = 3.1415926536;
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

En ocasiones puede que al declarar una variable no le demos valor. ¿Qué crees que ocurre en estos casos?

Pues que el compilador le asigna un valor por omisión, aunque depende del tipo de variable que se trate:

- ✓ Las **variables miembro** sí se inicializan automáticamente, si no les damos un valor.
  - Cuando son de tipo numérico, se inicializan por defecto a **0**.
  - Si son de tipo carácter, se inicializan al carácter **null (\0)**.
  - Si son de tipo **boolean** se les asigna el valor por defecto **false**.
  - Si son tipo referencia se inicializan a **null**.
- ✓ Las **variables locales** no se inicializan automáticamente. Debemos asignarles nosotros un valor antes de ser usadas, ya que si el compilador detecta que la variable se usa antes de que se le asigne un valor, produce un error. Por ejemplo en este caso:

```
int p;  
  
int q = p; // error
```

Estamos intentando inicializar la variable **q** con el valor que tenga **p**, pero como **p** no ha sido inicializada no tiene ningún valor, así que el compilador no sabe qué valor asignar, y da un error.



### Autoevaluación

De las siguientes, señala cuál es la afirmación correcta:

- La declaración de una variable consiste básicamente en indicar el tipo que va a tener seguido del nombre y su valor.
- Java no tiene restricción de tipos.
- Todos los tipos tienen las mismas operaciones a realizar con ellos: suma, resta, multiplicación, etc.
- Todas las anteriores son incorrectas.

## 10.3. TIPOS REFERENCIADOS.

¿No te parece que los tipos vistos hasta ahora son un poco limitados? Por ejemplo, no parece razonable que, para almacenar las notas de los 800 alumnos y alumnas de un centro escolar para la primera, segunda y tercera evaluación, tuviéramos que declarar en nuestro programa  $3 \times 800 = 2.400$  variables, ¿no te parece? ¿Y qué pasa si el curso próximo el centro tiene 805 alumnos? Tendríamos que ir cambiando el programa cada vez que cambiara el número de alumnos y alumnas. O si quisieramos meter una cuarta nota para la evaluación final, etc. ¡¡Se haría inmanejable!!

A partir de los ocho tipos de datos primitivos, se pueden construir otros tipos de datos. Estos tipos de datos se llaman **tipos referenciados** o **referencias**, porque se utilizan para almacenar la dirección de los datos en la memoria del ordenador.

```
int[] arrayDeEnteros;  
Cuenta cuentaCliente;
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

En la primera instrucción declaramos una lista de números del mismo tipo, en este caso, enteros (le llamamos array de enteros, vector de enteros). En la segunda instrucción estamos declarando la variable u objeto **cuentaCliente** como una referencia de tipo **Cuenta**. (Cuenta es una clase que habrá definido el usuario, y que especificará qué elementos forman parte de una cuenta, y qué operaciones se pueden hacer con ella).

Como comentábamos al principio del apartado de Tipos de datos, cualquier aplicación de hoy en día necesita no perder de vista una cierta cantidad de datos. Cuando el conjunto de datos utilizado tiene características similares se suelen agrupar en estructuras para facilitar el acceso a los mismos, son los llamados **datos estructurados**. Son datos estructurados los **arrays**, **listas**, **árboles**, etc. Pueden estar en la memoria del programa en ejecución, guardados en el disco como ficheros, o almacenados en una base de datos.

Además de los ocho tipos de datos primitivos que ya hemos descrito, Java proporciona un tratamiento especial a los textos o cadenas de caracteres mediante el tipo de dato **String**. Java crea automáticamente un nuevo objeto de tipo **String** cuando se encuentra una cadena de caracteres encerrada entre comillas dobles. En realidad, se trata de objetos, y por tanto son tipos referenciados, pero el lenguaje nos permite utilizarlos **también de forma sencilla como si fueran variables de tipos primitivos**:

```
String primerMensaje;
primerMensaje="El primer mensaje";
String segundoMensaje="Otro mensaje más";
```

Hemos visto qué son las variables, cómo se declaran y los tipos de datos que pueden adoptar. Anteriormente hemos visto también un ejemplo de creación de variables. Ahora vamos a crear más variables, pero de distintos tipos primitivos y las vamos a mostrar por pantalla. Los tipos referenciados los veremos en unidades posteriores.

Aquí tienes otro ejemplo donde se declaran más variables, una de ellas de tipo referenciado:

```
public class EjemploTipos {

    public static void main(String[] args) {
        int i = 10;
        double d = 3.14;
        char c1 = 'a';
        char c2 = 65;
        boolean encontrado = true;
        String mensaje = "Bienvenido a Java";

        System.out.println("La variable i es de tipo entero y su valor es: " + i);
        System.out.println("La variable d es de tipo double y su valor es: " + d);
        System.out.println("La variable c1 es de tipo carácter y su valor es: " + c1);
        System.out.println("La variable c2 es de tipo carácter y su valor es: " + c2);
        System.out.println("La variable encontrado es de tipo booleano y su valor es: " + encontrado);
        System.out.println("La variable mensaje es de tipo String y su valor es: " + mensaje);
    }
}
```

El resultado que se obtendrá por pantalla al ejecutar el programa tendrá el siguiente aspecto:

```
La variable i es de tipo entero y su valor es: 10
La variable d es de tipo double y su valor es: 3.14
La variable c1 es de tipo carácter y su valor es: a
La variable c2 es de tipo carácter y su valor es: A
La variable encontrado es de tipo booleano y su valor es: true
La variable mensaje es de tipo String y su valor es: Bienvenido a Java
```



## Reflexiona

¿Cuál de las variables anteriores es de tipo referenciado?

[Mostrar retroalimentación](#)

La variable de tipo referenciado es `mensaje`, que es de tipo `String`. Pero fíjate que la hemos usado exactamente igual que una variable de tipo primitivo. Más adelante veremos las consecuencias de que las cadenas en Java sean objetos y por tanto de tipo referenciado. Por ahora nos quedamos con que podemos declararlas y asignarles valor como a los tipos primitivos.

Por cierto, fíjate en que los tipos primitivos van siempre en minúscula. Sin embargo, `String` comienza por mayúscula.

## 10.4. TIPOS ENUMERADOS.

Los **tipos de datos enumerados** permiten una forma de declarar una variable con un conjunto restringido de valores. Por ejemplo: los días de la semana, las estaciones del año, los meses, etc. Es como si definiéramos nuestro propio tipo de datos.

Para declararlos se usa la palabra reservada `enum`, seguida del nombre de la variable y la lista de valores que puede tomar entre llaves. A los valores que se colocan dentro de las llaves se les considera como constantes, van separados por comas y deben ser valores únicos.

La lista de valores se coloca entre llaves, porque un tipo de datos `enum` no es otra cosa que una especie de clase en Java, y todas las clases llevan su contenido entre llaves.

Al considerar Java este tipo de datos como si de una clase se tratara, no solamente podemos definir los valores de un tipo enumerado, sino que también podemos definir operaciones a realizar con él y otro tipo de elementos, lo que hace que este tipo de dato sea más versátil y potente que en otros lenguajes de programación. Pero eso se verá en unidades más avanzadas. Por el momento debemos quedarnos con que **los enum nos permiten definir un nuevo tipo cuyos valores posibles son los que nosotros definamos**.

A continuación, tienes un ejemplo de creación y uso de un tipo de dato enumerado (`enum` en Java). Declaramos un `enum Dias` que contiene valores que representan todos los posibles días de la semana. **Para acceder a cada posible valor de un tipo enumerado se utiliza el nombre del enum seguido de un punto y el valor en la lista:**

```
public class TiposEnumerados {
    public enum Dias {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO};

    public static void main(String[] args) {
        Dias diaActual = Dias.MARTES ;
        Dias diaSiguiente = Dias.MIERCOLES ;

        System.out.println("Hoy es: " + diaActual);
        System.out.println("Mañana es " + diaSiguiente);
    }
}
```

Este programa, además de declarar el `enum Dias` con sus posibles valores, declara dos variables de ese nuevo tipo (`diaActual` y `diaSiguiente`) que acabamos de definir y les asigna valores de entre el conjunto de valores posibles. El resultado de su ejecución debería mostrar por pantalla lo siguiente:

Hoy es: MARTES  
Mañana es MIERCOLES



### Ejercicio Resuelto

Crea un enumerado que contenga las unidades de medida de volumen que van desde mililitro hasta hectolitro, después muestra por pantalla cada valor del enumerado.

[Mostrar retroalimentación](#)

```
1 | public class EjercicioUnidadesVolumen {  
2 |  
3 |     public enum UnidadVolumen {MILILITRO, CENTILITRO, DECILITRO, LITRO, DECALITRO, HECTOLITRO };  
4 |  
5 |     public static void main(String[] args) {  
6 |  
7 |         System.out.println(UnidadVolumen.MILILITRO);  
8 |         System.out.println(UnidadVolumen.MILILITRO);  
9 |         System.out.println(UnidadVolumen.CENTILITRO);  
10 |        System.out.println(UnidadVolumen.DECILITRO);  
11 |        System.out.println(UnidadVolumen.LITRO);  
12 |        System.out.println(UnidadVolumen.DECALITRO);  
13 |        System.out.println(UnidadVolumen.HECTOLITRO);  
14 |    }  
15 |}  
16 |}
```

## 11. LITERALES DE LOS TIPOS PRIMITIVOS.

### Caso práctico

**Ada** se encuentra con **María y Juan**.

—¿Cómo van esos avances con Java?

**Juan** sabe lo que significa eso, **Ada** se interesa por el trabajo que están llevando a cabo. Ya tienen claro qué tipos de datos utilizar, pero necesitan cerciorarse de los valores que pueden almacenar esos tipos de datos, es decir, qué literales pueden contener, para estar seguros que han hecho la elección adecuada.

—Muy bien —contesta **Juan**—. Si quieras te hacemos una demostración para que veas la estructura del programa.

A **Ada** le satisface la eficacia con que trabajan **María y Juan**, apenas han comenzado con el proyecto y pronto podrá ver resultados inmediatos.

Un **literal**, **valor literal** o **constante literal** es un valor concreto para los tipos de datos primitivos del lenguaje, el tipo **String** o el valor **null**.

Los **literales booleanos** son dos únicos valores, los que puede aceptar el tipo: **true** y **false**. Por ejemplo, con la instrucción **boolean encontrado = true;** estamos declarando una variable de tipo booleana a la cual le asignamos el valor literal **true**.

Los **literales enteros** se pueden representar en tres notaciones:

- ✓ **Decimal:** por ejemplo **20**. Es la forma más común.
- ✓ **Octal:** por ejemplo **024**. Un número en octal siempre empieza por cero, seguido de dígitos octales (del 0 al 7).
- ✓ **Hexadecimal:** por ejemplo **0x14**. Un número en hexadecimal siempre empieza por **0x** seguido de dígitos hexadecimales (del 0 al 9, de la ‘a’ a la ‘f’ o de la ‘A’ a la ‘F’).

A los literales de tipo **long** se les debe añadir detrás una ele mayúscula o minúscula (**l** ó **L**), por ejemplo **873L**, de lo contrario se considera por defecto un literal de tipo **int**. Se suele utilizar **L** para evitar la confusión de la ele minúscula con 1.

Los **literales reales** o en coma flotante se expresan con coma decimal o en notación científica, o sea, seguidos de un exponente **e** ó **E**. El valor puede finalizarse con una **f** o una **F** para indicar que se trata de un literal de tipo **float** o con una **d** o una **D** para indicar el formato **double** (por defecto, si no se pone nada, es **double**). Por ejemplo, podemos representar un mismo literal real de las siguientes formas: **13.2**, **13.2D**, **1.32e1**, **0.132E2**. Otras constantes literales reales son por ejemplo: **.54**, **31.21E-5**, **2.f**, **6.022137e+23f**, **3.141e-9d**.

Desde Java SE 7 y posteriores, pueden aparecer caracteres **\_** entre dígitos en un literal numérico. La idea subyacente es mejorar la legibilidad del código. Así, por ejemplo, sería válido escribir:

```
long numeroDeTarjeta = 1234_5678_9012_3456L;
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Un **literal carácter** puede escribirse como un carácter entre comillas simples como 'a', 'ñ', 'Z', 'p', etc., o por su código de la tabla Unicode, anteponiendo la secuencia de escape '\' si el valor lo ponemos en octal o '\u' si ponemos el valor en hexadecimal. Por ejemplo, si sabemos que tanto en ASCII como en Unicode, la letra A (mayúscula) es el símbolo número 65, y que 65 en octal es 101 y 41 en hexadecimal, podemos representar esta letra como '\101' en octal y '\u0041' en hexadecimal. Existen unos caracteres especiales que se representan utilizando secuencias de escape:

### Secuencias de escape en Java

Secuencia de escape	Significado	Secuencia de escape	Significado
\b	Retroceso	\r	Retorno de carro
\t	Tabulador	\"	Carácter comillas dobles
\n	Salto de línea	\'	Carácter comillas simples
\f	Salto de página	\\\	Barra diagonal

Normalmente, los objetos en Java deben ser creados con el operador `new`. Sin embargo, los literales `String` no lo necesitan ya que son objetos que se crean implicitamente por Java, pudiendo prescindir del uso del operador `new`.

Los **literales de cadenas de caracteres** se indican entre comillas dobles. En el ejemplo anterior “*El primer mensaje*” es un literal de tipo cadena de caracteres. Al construir una cadena de caracteres se puede incluir cualquier carácter Unicode excepto un carácter de retorno de carro. Por ejemplo, en la siguiente instrucción utilizamos la secuencia de escape \" para escribir dobles comillas dentro del mensaje:

```
String texto = "Juan dijo: \"Hoy hace un día fantástico...\"";
```

En realidad, dentro de la variable texto se está incluyendo la cadena literal *Juan dijo: "Hoy hace un día fantástico..."*, que incluye una parte entre comillas dobles.

En el ejemplo del apartado anterior para tipos enumerados ya estábamos utilizando secuencias de escape, para introducir un salto de línea en una cadena de caracteres, utilizando el carácter especial \n.



### Para saber más

Tienes más ejemplos sobre el uso de literales numéricos con el carácter underscore en el siguiente enlace en inglés:

[Literales numéricos con el carácter underscore](#)

<https://docs.oracle.com/javase/8/docs/technotes/guides/language/underscores-literals.html>



## Ejercicio Resuelto

Declara e inicializa las variables o constantes necesarias en Java, de forma que encajen con cada uno de los siguientes enunciados:

- ✓ La edad de un niño es 12.
- ✓ La velocidad de la luz es de 300000 kilómetros por segundo.
- ✓ La edad mínima para apuntarse a actividades de tiro con arco es de 10 años.
- ✓ El correo electrónico de una persona es vaya@mail.com.
- ✓ El peso de un atleta es de 40,44 kilogramos.
- ✓ El número de meses del año es 12.
- ✓ La letra del DNI de una persona es c.
- ✓ El número de teléfono es 887-44-42-12.
- ✓ La distancia de la tierra al sol es de 147.100.000.000 metros
- ✓ La distancia que recorre la luz en un año es de 9.460.740.478.580,8 kilómetros.

**Mostrar retroalimentación**

Una solución posible podría ser la siguiente:

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

        //VARIABLE: La edad de un niño es 12.
        //Variable tipo int declarada e inicializada en el mismo sitio.
        int edadNiño = 12;

        //CONSTANTE: La velocidad de la luz es de 300000 (trescientos mil) kilómetros por segundo.
        //Constante tipo double inicializada en la misma línea.
        // El literal de entero (300000) promociona a double automáticamente.
        final double VELOCIDAD_LUZ = 300000;

        //CONSTANTE: La edad mínima para apuntarse a actividades de tiro con arco es de 10 años.
        //Constante tipo int declarada e inicializada en distintas líneas.
        final int EDAD_MINIMA_TIRO_ARCO;
        EDAD_MINIMA_TIRO_ARCO = 10;

        //VARIABLE: el correo electrónico de una persona es vaya@mail.com
        //Variable tipo String declarada e inicializada en la misma línea.
        String mail="vaya@mail.com";

        //VARIABLE: el peso de un atleta es de 40,44 kilogramos.
        //Variable tipo double declada e inicializada en líneas diferentes.
        double pesoAtleta;
        pesoAtleta=40.44d; //d indica literal de double

        //CONSTANTE: el número de meses del año es 12
        //Constante entera declarada e inicializadas en líneas diferentes.
        final int MESES_AÑO;
        MESES_AÑO=12;
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
//VARIABLE: la letra del dni de una persona es C
//Variable tipo char declarada e inicializada en la misma línea.
char letraDNI='C';

//VARIABLE: El número de teléfono es 887-44-42-12.
//Variable tipo String declarada e inicializada en líneas diferentes.
String telefono;
telefono="887-44-42-12";

//CONSTANTE: la distancia de la tierra al sol es de 147.100.000.000 metros
//Constante tipo long inicializada y declarada en la misma línea
final long DISTANCIA_TIERRA_SOL=147_100_000_000L;

//CONSTANTE: la distancia que recorre la luz en un año es de 9.460.740.478.580,8
kilómetros.
//Constante tipo double inicializada y declarada en la misma línea.
final double KM_RECORRIDOS_LUZ_AÑO=9_460_740_478_580.8d;
}

}
```

## 12. OPERADORES Y EXPRESIONES.

### Caso práctico

**María y Juan** tienen definidas las variables y tipos de datos a utilizar en la aplicación. Es el momento de ponerse a realizar los cálculos que permitan manipular esos datos, sumar, restar, multiplicar, dividir y mucho más. En definitiva, se trata de llevar los conocimientos matemáticos al terreno de la programación, ver cómo se pueden hacer operaciones aritméticas, lógicas o de comparación en el lenguaje Java. También necesitarán algún operador que permita evaluar una condición y decidir las acciones a realizar en cada caso. Es importante conocer bien cómo el lenguaje evalúa esas expresiones, en definitiva, cuál es la precedencia que tiene cada operador.

Los **operadores** llevan a cabo operaciones sobre un conjunto de datos u operandos, representados por literales y/o identificadores. Los operadores pueden ser **unarios, binarios o terciarios**, según el número de operandos que utilicen sean uno, dos o tres, respectivamente. Los operadores actúan sobre los tipos de datos primitivos y devuelven también un tipo de dato primitivo.

Los operadores se combinan con los literales y/o identificadores para formar **expresiones**.

Una **expresión** es una combinación de operadores y operandos que se evalúa produciendo un único resultado de un tipo determinado.

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

El resultado de una expresión puede ser usado como parte de otra expresión o en una sentencia o instrucción. Las expresiones, combinadas con algunas palabras reservadas o por sí mismas, forman las llamadas **sentencias o instrucciones**.

Por ejemplo, pensemos en la siguiente expresión Java:

```
i + 1
```

Con esta expresión estamos utilizando un operador aritmético para sumarle una cantidad a la variable **i**, pero es necesario indicar al programa qué hacer con el resultado de dicha expresión:

```
suma = i + 1;
```

Que lo almacene en una variable llamada **suma**, por ejemplo. En este caso ya tendríamos una acción completa, es decir, una sentencia o instrucción.

Más ejemplos de sentencias o instrucciones los tenemos en las declaraciones de variables, vistas en apartados anteriores, o en las estructuras básicas del lenguaje como sentencias condicionales o bucles, que veremos en unidades posteriores.

Como curiosidad comentar que las **expresiones de asignación**, al poder ser usadas como parte de otras asignaciones u operaciones, son consideradas tanto expresiones en sí mismas como sentencias.

### 12.1. OPERADORES ARITMÉTICOS.

Los operadores aritméticos son aquellos operadores que combinados con los operandos forman expresiones matemáticas o aritméticas

#### Operadores aritméticos básicos

Operador	Operación Java	Expresión Java	Resultado
-	Operador unario de cambio de signo	-10	-10
+	Adición	1.2 + 9.3	10.5
-	Sustracción	312.5 - 12.3	300.2
*	Multiplicación	1.7 * 1.2	2.04
/	División (entera o real)	0.5 / 0.2	2.5
%	Resto de la división entera	25 % 3	1

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

El resultado de este tipo de expresiones depende de los operandos que utilicen:

### Resultados de las operaciones aritméticas en Java

Tipo de los operandos	Resultado
Un operando de tipo <code>long</code> y ninguno real ( <code>ni float ni double</code> )	<code>long</code>
Ningún operando de tipo <code>long</code> ni real ( <code>float o double</code> )	<code>int</code>
Al menos un operando de tipo <code>double</code>	<code>double</code>
Al menos un operando de tipo <code>float</code> y ninguno <code>double</code>	<code>float</code>

Otro tipo de operadores aritméticos son los **operadores unarios de incremento y decremento**. Producen un resultado del mismo tipo que el operando, y podemos utilizarlos con **notación prefija**, si el operador aparece antes que el operando, o **notación postfija**, si el operador aparece después del operando. En la tabla puedes ver un ejemplo de utilización de cada uno de estos operadores.

### Operadores incrementales en Java

Tipo operador	Expresión Java	
<code>++ (incremental)</code>	Prefija:  <code>x=3; y=++x; // x vale 4 e y vale 4</code>	Postfija:  <code>x=3; y=x++; // x vale 4 e y vale 3</code>
<code>--(decremental)</code>	<code>5-- // el resultado es 4</code>	

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

En el siguiente ejemplo vemos un programa básico que utiliza algunos operadores aritméticos.

### Ejemplo de uso de operadores aritméticos en Java

Ejemplo de código	Salida por pantalla
<pre>public class OperadoresAritmeticos {     public static void main(String[] args) {         // Declaración de variables         short x = 7;         int y = 5;         float f1 = 13.5f;         float f2 = 8f;          // Ejemplos de operaciones         System.out.println("EJEMPLOS DE USO DE OPERADORES ARITMÉTICOS");         System.out.println("-----");         System.out.println("El valor de x es " + x + ", y es " + y);         System.out.println("El resultado de x + y es " + (x + y));         System.out.println("El resultado de x - y es " + (x - y));         System.out.println("División entera: x / y = " + (x/y));         System.out.println("Resto de la división entera: x % y = " + (x % y));         System.out.println("El valor de f1 es " + f1 + ", f2 es " + f2);         System.out.println("El resultado de f1 / f2 es " + (f1 / f2));     } }</pre>	<p>EJEMPLOS DE USO DE OPERADORES ARITMÉTICOS</p> <hr/> <p>-----</p> <p>El valor de x es 7, y es 5 El resultado de x + y es 12 El resultado de x - y es 2 División entera: x / y = 1 Resto de la división entera: x % y = 2 El valor de f1 es 13.5, f2 es 8.0 El resultado de f1 / f2 es 1.6875</p>



### Ejercicio Resuelto

Partiendo de una variable entera llamada `x` cuyo valor inicial es `6`, haz que se muestre por pantalla la secuencia de números siguiente, aplicando una operación matemática sobre `x`:

6, 10, -4, 12, 3, 2, 9, -6, 0

Para hacer este ejercicio deberás realizar 9 operaciones matemáticas sobre la variable `x`, y deberás utilizar cada uno de los operadores vistos en este apartado al menos una vez.

[Mostrar retroalimentación](#)

Una solución posible podría ser la siguiente:

```
public class EjercicioListaNumeros {

    public static void main(String[] args) {

        int x=6;

        System.out.println(x - 0);
        System.out.println(x + 4);
        System.out.println(x - 10);
        System.out.println(x * 2);
        System.out.println(x / 2);
        System.out.println(x % 4);
        System.out.println(x + 3);
        System.out.println(-x);
        System.out.println(x - x);

    }
}
```



## Ejercicio Resuelto

Con el siguiente código se lee desde teclado el precio de un producto:

```
import java.util.Scanner;

public class CalcularIVA {
    public static void main(String args[]) {
        Scanner scanner=new Scanner(System.in);
        System.out.print("Introduce el precio del producto: ");
        double precioProducto=scanner.nextDouble();
    }
}
```

¿Podrías modificar el código anterior para que mostrara el precio con IVA de la siguiente forma? (ten en cuenta que el IVA es del 21%)

```
Introduce el precio del producto: 8,4
Importe del IVA: 1.764
Precio con IVA: 10.164
```

[Mostrar retroalimentación](#)

Una solución al problema anterior podría ser la siguiente:

```
import java.util.Scanner;

public class CalcularIVA {
    public static void main(String args[]) {
        Scanner scanner=new Scanner(System.in);
        System.out.print("Introduce el precio del producto: ");
        double precioProducto=scanner.nextDouble();
        double iva=0.21*precioProducto;
        double precioConIVA=precioProducto+iva;
        System.out.println("Importe del IVA: " +iva);
        System.out.println("Precio con IVA: " +precioConIVA);
    }
}
```



## Ejercicio Resuelto

El siguiente código pregunta al usuario por la velocidad (en kilómetros por hora) y el tiempo (en segundos) de un vehículo.

```
import java.util.Scanner;

public class Velocidad {
    public static void main(String args[]) {
        Scanner scanner=new Scanner(System.in);
        System.out.print("Velocidad en Kilometros por hora:");
        double velocidad=scanner.nextDouble();
        System.out.print("Tiempo en segundos:");
        int segundos=scanner.nextInt();
    }
}
```

¿Serías capaz de modificarlo para que calculase la distancia recorrida en kilómetros y en metros? Un ejemplo de salida sería:

```
Velocidad en kilometros por hora:30
Tiempo en segundos:90
Distancia recorrida en kilometros:0.75
Distancia recorrida en metros:750.0
```

[Mostrar retroalimentación](#)

Una solución posible sería la siguiente:

```
import java.util.Scanner;

public class Velocidad {
    public static void main(String args[]) {
        Scanner scanner=new Scanner(System.in);
        System.out.print("Velocidad en kilometros por hora:");
        double velocidad=scanner.nextDouble();
        System.out.print("Tiempo en segundos:");
        int segundos=scanner.nextInt();

        double distanciaKM=velocidad*(segundos/3600.0);
        double distanciaM=distanciaKM*1000;

        System.out.println("Distancia recorrida en kilometros:"+distanciaKM);
        System.out.println("Distancia recorrida en metros:"+distanciaM);

    }
}
```

Fíjate que el tiempo en segundos se divide por 3600.0 (literal `double`) y no por 3600 (literal `int`). Esto se realiza así para forzar a que la división `segundos/3600.0` se realice con precisión `double` y no con precisión `int`.

# TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

## **12.2. OPERADORES DE ASIGNACIÓN.**

El principal operador de esta categoría es el operador asignación "`=`" que permite al programa darle un valor a una variable, que ya hemos utilizado en varias ocasiones en esta unidad. Además de este operador, Java proporciona otros operadores de asignación combinados con los operadores aritméticos, que permiten abreviar o reducir ciertas expresiones.

Por ejemplo, el operador `"+="` suma el valor de la expresión a la derecha del operador con el valor de la variable que hay a la izquierda del operador, y almacena el resultado en dicha variable. En la siguiente tabla se muestran todos los operadores de asignación compuestos que podemos utilizar en Java.

## Operadores de asignación combinados en Java

Operador	Ejemplo en Java	Expresión equivalente
<code>+=</code>	<code>op1 += op2</code>	<code>op1 = op1 + op2</code>
<code>-=</code>	<code>op1 -= op2</code>	<code>op1 = op1 - op2</code>
<code>*=</code>	<code>op1 *= op2</code>	<code>op1 = op1 * op2</code>
<code>/=</code>	<code>op1 /= op2</code>	<code>op1 = op1 / op2</code>
<code>%=</code>	<code>op1 %= op2</code>	<code>op1 = op1 % op2</code>

A continuación, tienes un ejemplo de uso de algunos de estos operadores de asignación.

Ejemplo de código	Salida por pantalla
<pre>public class OperadoresAsignacion {      public static void main(String[] args) {         int x;         int y;          x = 5; // operador asignación         y = 3; // operador asignación          System.out.println ("EJEMPLOS DE USO DE OPERADORES DE ASIGNACIÓN");         System.out.println ("-----");         System.out.println (" El valor de x es: " + x);         System.out.println (" El valor de y es: " + y);         System.out.println ();          // Ejemplos de uso de operadores de asignación combinados         x += y;         System.out.println(" Suma combinada:      x += y " + " ..... x vale " + x);          x = 5;         x -= y;         System.out.println(" Resta combinada:      x -= y " + " ..... x vale " + x);          x = 5;         x *= y;         System.out.println(" Producto combinado: x *= y " + " ..... x vale " + x);          x = 5;         x /= y;         System.out.println(" División combinada: x /= y " + " ..... x vale " + x);          x = 5;         x %= y;         System.out.println(" Resto combinado:      x %= y " + " ..... x vale " + x);     } }</pre>	<p>EJEMPLOS DE USO DE OPERADORES DE ASIGNACIÓN</p> <p>-----</p> <p>El valor de x es: 5 El valor de y es: 3</p> <p>Suma combinada: x += y ..... x vale 8 Resta combinada: x -= y ..... x vale 2 Producto combinado: x *= y ..... x vale 15 División combinada: x /= y ..... x vale 1 Resto combinado: x %= y ..... x vale 2</p>



### Ejercicio Resuelto

Dada la variable entera `x`, cuyo valor inicial es `10`, haz un programa en el que se vaya modificando el valor de la variable `x` de manera que dicha variable tome la siguiente secuencia de valores:

5, 6, 12, 6, -4, 1

La variable `x` deberá modificarse haciendo una operación aritmética sobre el valor que ya posee, por ejemplo:

```
int x=10;
x=x+100; //La variable x toma el valor 110 (10 + 100)
```

Para hacer el ejercicio debes usar al menos una vez los operadores aritméticos de suma, resta, división y multiplicación. No olvides mostrar el valor de la variable `x` después de cada modificación.

[Mostrar retroalimentación](#)

Un solución posible sería la siguiente:

```
public class EjercicioModificacionValorVariable {
    public static void main(String[] args) {
        int x=10;
        x=x-5;
        System.out.println(x);
        x=x+1;
        System.out.println(x);
        x=x*2;
        System.out.println(x);
        x=x/2;
        System.out.println(x);
        x=x-10;
        System.out.println(x);
        x=x+5;
        System.out.println(x);
    }
}
```



### Ejercicio Resuelto

Una factoría papelera confecciona **cuadernos** en los que se van alternando hojas de color **rojo** y **verde**. Siempre se comienza por el color rojo, siguiéndole el verde y comenzando nuevamente con el rojo.

Implementar un programa en Java que calcule, para un cuaderno de **20 hojas**, cuántas hojas contiene de cada color.

[Mostrar retroalimentación](#)

Implementar un programa en Java que calcule, para un cuaderno de **61 hojas**, cuántas hojas contiene de cada color.

[Mostrar retroalimentación](#)

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Está claro que cada dos hojas se repiten consecutivamente los colores rojo y verde. Por tanto, cada paquete de dos hojas implica una hoja de cada uno de estos colores. Eso significa que si la cantidad de hojas fuera múltiplo de dos (par), podríamos obtener la cantidad de hojas de cada color mediante el cociente de la división entera entre dos. Ahora bien, si la cantidad de hojas no es múltiplo de dos (es impar), habrá un desfase de una hoja roja (+1) adicional.

¿Cómo calculamos ese posible "desfase" de hoja roja adicional? Podemos hacerlo usando el operador módulo (resto de la división entera).

Para el caso de las hojas verdes, sabemos que nunca habrá desfase respecto a la división entre dos:

Cantidad de hojas verdes = hojas totales / 2

Para el caso de las hojas rojas, podemos calcular el desfase de hojas mediante el operador módulo (cero o una hojas adicionales).

Cantidad de hojas rojas = hojas totales / 2 + (hojas totales % 2)

Una solución posible para un cuaderno de 20 hojas podría ser la siguiente:

```
public class EjercicioCuadernoColores1 {  
    public static void main(String[] args) {  
        int hojasTotales= 20;  
        int hojasRojas = hojasTotales / 2;  
        int hojasVerdes = hojasTotales / 2 + (hojasTotales % 2);  
  
        System.out.println ("Total de hojas en el cuaderno: " + hojasTotales);  
        System.out.println ("Cantidad de hojas rojas: " + hojasRojas);  
        System.out.println ("Cantidad de hojas verdes: " + hojasVerdes);  
    }  
}
```

El programa sería prácticamente igual. Lo único que cambiaría sería el contenido de la variable con el total de hojas (61):

```
public class EjercicioCuadernoColores2 {  
    public static void main(String[] args) {  
        int hojasTotales= 61;  
        int hojasRojas = hojasTotales / 2;  
        int hojasVerdes = hojasTotales / 2 + (hojasTotales % 2);  
  
        System.out.println ("Total de hojas en el cuaderno: " + hojasTotales);  
        System.out.println ("Cantidad de hojas rojas: " + hojasRojas);  
        System.out.println ("Cantidad de hojas verdes: " + hojasVerdes);  
    }  
}
```

## 12.3. OPERADORES DE RELACIÓN.

Los operadores relacionales se utilizan para comparar datos de tipo primitivo (numérico, carácter y booleano). El resultado de la comparación se utilizará en otras expresiones o sentencias, que permitirán comprobar en una condición dicho resultado, y ejecutar una acción u otra en función de si se cumple o no la condición comprobada al comparar.

Estas expresiones en Java dan siempre como resultado un valor booleano **true** o **false**. En la tabla siguiente aparecen los operadores relacionales en Java.

Hasta ahora hemos visto ejemplos que creaban variables y se inicializaban con algún valor. Pero ¿y si lo que queremos es que el usuario introduzca un valor al programa?

**Operadores relacionales en Java**

Operador	Ejemplo en Java	Significado
<code>=</code>	<code>op1 == op2</code>	<code>op1</code> igual a <code>op2</code>
<code>!=</code>	<code>op1 != op2</code>	<code>op1</code> distinto de <code>op2</code>
<code>&gt;</code>	<code>op1 &gt; op2</code>	<code>op1</code> mayor que <code>op2</code>
<code>&lt;</code>	<code>op1 &lt; op2</code>	<code>op1</code> menor que <code>op2</code>
<code>&gt;=</code>	<code>op1 &gt;= op2</code>	<code>op1</code> mayor o igual que <code>op2</code>
<code>&lt;=</code>	<code>op1 &lt;= op2</code>	<code>op1</code> menor o igual que <code>op2</code>

Hasta ahora hemos visto ejemplos que creaban variables y se inicializaban con algún valor. Pero ¿y si lo que queremos es que el usuario introduzca un valor al programa?

Entonces debemos agregarle interactividad a nuestro programa, por ejemplo utilizando la clase **Scanner**. Aunque no hemos visto todavía qué son las clases y los objetos, de momento vamos a pensar que la clase **Scanner** nos va a permitir leer los datos que se escriben por teclado, y que para usarla es necesario importar el paquete de clases que la contiene. El código del ejemplo lo tienes a continuación. El programa se quedará esperando a que el usuario escriba algo en el teclado y pulse la tecla intro. En ese momento se convierte lo leído a un valor del tipo **int** y lo guarda en la variable indicada. Además de los operadores relacionales, en este ejemplo utilizamos también el operador condicional, que compara si los números son iguales. Si lo son, devuelve la cadena iguales y si no, la cadena distintos.

```
import java.util.Scanner; //importamos el paquete necesario para poder usar la clase Scanner
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
public class EjemploRelacionales {  
  
    public static void main( String args[] ){  
  
        Scanner teclado = new Scanner( System.in );  
        int x, y;  
        String cadena;  
        boolean resultado;  
  
        System.out.print( "Introducir primer número: " );  
        x = teclado.nextInt(); // pedimos el primer número al usuario  
        System.out.print( "Introducir segundo número: " );  
        y = teclado.nextInt(); // pedimos el segundo número al usuario  
  
        // realizamos las comparaciones y las mostramos por pantalla  
        cadena=(x==y)?"iguales":"distintos";  
        System.out.printf("Los números %d y %d son %s\n",x,y,cadena);  
        resultado=(x!=y);  
        System.out.println("x != y // es " + resultado);  
        resultado=(x < y );  
        System.out.println("x < y // es " + resultado);  
        resultado=(x > y );  
        System.out.println("x > y // es " + resultado);  
        resultado=(x <= y );  
        System.out.println("x <= y // es " + resultado);  
        resultado=(x >= y );  
        System.out.println("x >= y // es " + resultado);  
    }  
}
```



### Autoevaluación

Señala cuáles son los operadores relacionales en Java.

- ==, !=, >, <, >=, <=.
- =, !=, >, <, >=, <=.
- ==, !=, >, <, =>, =<.
- ==, !=, >, <, >=, <=.



## Ejercicio Resuelto

Dada una variable entera `x` cuyo valor inicial es `5`, y otra variable entera `z` cuyo valor inicial es `6`, haz que se muestre la secuencia de resultados: `true, false, true, false, true, y false` (6 en total); usando un operador de relación diferente en cada caso.

[Mostrar retroalimentación](#)

Una solución posible podría ser la siguiente:

```
public class EjercicioOperadoresRelacionales {
    public static void main(String[] args) {
        int x=5;
        int z=6;
        boolean b=x<z;
        System.out.println(b);
        b=x>z;
        System.out.println(b);
        b=x<=z;
        System.out.println(b);
        b=x>=z;
        System.out.println(b);
        b=x!=z;
        System.out.println(b);
        b=x==z;
        System.out.println(b);
    }
}
```

## 12.4. OPERADORES LÓGICOS

Los operadores lógicos realizan operaciones sobre valores booleanos, o resultados de expresiones relacionales, dando como resultado un valor booleano.

Los operadores lógicos los podemos ver en la tabla que se muestra a continuación.

**Existen ciertos casos en los que el segundo operando de una expresión lógica no se evalúa para ahorrar tiempo de ejecución, porque con el primero ya es suficiente para saber cuál va a ser el resultado de la expresión.**

Por ejemplo, en la expresión `a && b` si `a` es falso, no se sigue comprobando la expresión, puesto que ya se sabe que la condición de que ambos sean verdadero no se va a cumplir. En estos casos es más conveniente colocar el operando más propenso a ser falso en el lado de la izquierda. Igual ocurre con el operador `||`, en cuyo caso es más favorable colocar el operando más propenso a ser verdadero en el lado izquierdo

## Operadores lógicos en Java

Operador	Ejemplo en Java	Significado
!	!op	Devuelve true si el operando es false y viceversa.
&	op1 & op2	Devuelve true si op1 y op2 SON true
	op1   op2	Devuelve true Si op1 U op2 SON true
^	op1 ^ op2	Devuelve true si sólo uno de los operandos es true
&&	op1 && op2	Igual que &, pero si op1 ES false ya no se evalúa op2
	op1    op2	Igual que  , pero si op1 es true ya no se evalúa op2

En el siguiente código puedes ver un ejemplo de utilización de operadores lógicos. Observa la salida que se debería obtener:

EJEMPLO DE CÓDIGO	SALIDA POR PANTALLA
<pre>public class OperadoresLogicos {      public static void main(String[] args) {          System.out.println("EJEMPLOS DE USO DE OPERADORES LÓGICOS");         System.out.println("-----");          System.out.println("Negacion:\n !false es : " + (!false));         System.out.println(" !true es : " + (! true));          System.out.println("Operador AND (&amp;):");         System.out.println(" false &amp; false es : " + (false &amp; false));         System.out.println(" false &amp; true  es : " + (false &amp; true));         System.out.println(" true  &amp; false es : " + (true &amp; false));         System.out.println(" true  &amp; true  es : " + (true &amp; true));          System.out.println("Operador OR ( ):");         System.out.println(" false   false es : " + (false   false));         System.out.println(" false   true  es : " + (false   true));         System.out.println(" true    false es : " + (true   false));         System.out.println(" true    true  es : " + (true   true));     } }</pre>	<p>EJEMPLOS DE USO DE OPERADORES LÓGICOS</p> <hr/> <p>Negacion:</p> <p>!false es : true !true es : false</p> <p>Operador AND (&amp;):</p> <p>false &amp; false es : false false &amp; true es : false true &amp; false es : false true &amp; true es : true</p> <p>Operador OR ( ):</p> <p>false   false es : false false   true es : true true   false es : true true   true es : true</p> <p>Operador OR Exclusivo o XOR (^):</p> <p>false ^ false es : false false ^ true es : true true ^ false es : true true ^ true es : false</p> <p>Operador AND (&amp;&amp;):</p> <p>false &amp;&amp; false es : false false &amp;&amp; true es : false true &amp;&amp; false es : false true &amp;&amp; true es : true</p> <p>Operador OR (  ):</p> <p>false    false es : false false    true es : true true    false es : true true    true es : true</p>

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
System.out.println(" true | false es : " + (true  
| false));  
  
System.out.println(" true | true es : " + (true  
| true));  
  
System.out.println("Operador OR Exclusivo o XOR  
(^):");  
System.out.println(" false ^ false es : " +  
(false ^ false));  
System.out.println(" false ^ true es : " + (false  
^ true));  
System.out.println(" true ^ false es : " + (true  
^ false));  
System.out.println(" true ^ true es : " + (true  
^ true));  
  
System.out.println("Operador AND (&&):");  
System.out.println(" false && false es : " +  
(false && false));  
System.out.println(" false && true es : " +  
(false && true));  
System.out.println(" true && false es : " + (true  
&& false));  
System.out.println(" true && true es : " + (true  
&& true));  
  
System.out.println("Operador OR (||):");  
System.out.println(" false || false es : " +  
(false || false));  
System.out.println(" false || true es : " +  
(false || true));  
System.out.println(" true || false es : " + (true  
|| false));  
System.out.println(" true || true es : " + (true  
|| true));  
}  
}
```



## Ejercicio Resuelto

Dadas las variables siguientes:

```
1 | int x1=10, x2=5, x3=0;
2 | char c1='F', c2='S';
```

Crea una expresión lógica, que utilice operadores lógicos y relacionales, para cada uno de los siguientes casos (intenta evaluar mentalmente el resultado de la expresión antes de mostrarlo por pantalla):

- ✓  $x_1$  es igual a  $x_2$
- ✓  $c_1$  es distinto a  $c_2$
- ✓  $x_1$  está entre 10 y 100
- ✓  $x_2$  no está entre 10 y 100
- ✓  $x_3$  no está entre 10 y 100
- ✓  $x_1$  es mayor que  $x_2$  y  $c_1$  es igual a  $c_2$
- ✓ O  $x_1$  es mayor que  $x_2$ , o  $c_1$  es distinto a  $c_2$ , cualquiera de los dos casos.
- ✓  $x_1$  es menor o igual que 7 y  $c_2$  es igual a  $c_1$
- ✓  $c_2$  es distinto de 'A' y  $x_2$  es mayor que 0
- ✓ 'F' es distinto de  $c_1$  o  $x_1$  es mayor que 20
- ✓ 'F' es distinto de  $c_1$  o  $x_1$  es mayor que 20 o  $x_2$  es mayor que 2
- ✓ 'F' es igual a  $c_1$  y  $x_3$  es menor que  $x_1$
- ✓ 'F' es igual a  $c_1$  y  $x_3$  es menor que  $x_1$  y  $x_2$  es menor o igual que  $x_3$
- ✓  $x_2$  está entre  $x_3$  y  $x_1$ , y  $c_2$  es 'S'
- ✓  $x_3$  no está entre  $x_2$  y  $x_1$
- ✓  $x_2$  no está entre  $x_3$  y  $x_1$ , o  $c_2$  es igual a  $c_1$
- ✓ no se cumple que  $x_3$  es menor que  $x_1$
- ✓ ni  $x_3$  es mayor que  $x_1$ , ni  $c_2$  es distinto a  $c_1$
- ✓ no se cumple que  $x_1$  es menor que 100 y  $x_2$  es mayor que 10
- ✓  $c_2$  es anterior alfabéticamente a  $c_1$

[Mostrar retroalimentación](#)

Una solución posible sería la siguiente:

```
public class EjercicioOperadoresLogicos {
    public static void main(String[] args) {
        int x1=10, x2=5, x3=0;
        char c1='F', c2='S';
        boolean b;

        //x1 es igual a x2
        b=x1==x2; //false
        System.out.println("C1: "+b);

        //c1 es distinto a c2
        b=c1!=c2; //true
        System.out.println("C2: "+b);

        //x1 está entre 10 y 100
        b=x1>=10 && x1<=100; //true (x1 está en ese rango)
        System.out.println("C3: "+b);

        //x2 no está entre 10 y 100
        b=! (x2>=10 && x2<=100); //true (x2 no está en ese rango)
        System.out.println("C4: "+b);
    }
}
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
//x3 no está entre 10 y 100  
b=x3<10 || x3>100; //true (x3 no está en ese rango)  
System.out.println("C5: "+b);  
  
//x1 es mayor que x2 y c1 es igual a c2  
b=x1>x2 && c1==c2; //false (c1 no es igual a c2)  
System.out.println("C6: "+b);  
  
//0 x1 es mayor que x2, o c1 es distinto a c2, cualquiera de los dos casos.  
b=x1>x2 || c1!=c2; //true (se cumplen ambas condiciones)  
System.out.println("C7: "+b);  
  
//x1 es menor o igual que 7 y c2 es igual a c1  
b=x1<=7 && c2==c1; //false (ambas condiciones son falsas)  
System.out.println("C8: "+b);  
  
//c2 es distinto de 'A' y x2 es mayor que 0  
b=c2!='A' && x2>0; //true (ambas condiciones son true)  
System.out.println("C9: "+b);  
  
//'F' es distinto de c1 o x1 es mayor que 20  
b='F'!=c1 || x1>20; //false (ambas condiciones son false)  
System.out.println("C10: "+b);  
  
//'F' es distinto de c1 o x1 es mayor que 20 o x2 es mayor que 2  
b='F'!=c1 || x1>20 || x2>2; //true (la última condición es cierta)  
System.out.println("C11: "+b);  
  
//'F' es igual a c1 y x3 es menor que x1  
b='F'==c1 && x3<x1; //true (ambas condiciones son ciertas)  
System.out.println("C12: "+b);  
  
//'F' es igual a c1 y x3 es menor que x1 y x2 es menor o igual que x3  
b='F'==c1 && x3<x1 && x2<=x3; //false (la última condición es falsa)  
System.out.println("C13: "+b);  
  
//x2 está entre x3 y x1, y c2 es 'S'  
b=x2>=x3 && x2<=x1 && c2=='S'; //true (todas las condiciones se cumplen)  
System.out.println("C14: "+b);  
  
//x3 no está entre x2 y x1  
b=x3<x2 || x3>x1; //true (todas las condiciones se cumplen)  
System.out.println("C15: "+b);
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
//x2 no está entre x3 y x1, o c2 es igual a c1  
b=x2<x3 || x2>x1 || c2==c1; // false (solo la última condición no se cumple)  
System.out.println("C16: "+b);  
  
//no se cumple que x3 es menor que x1  
b=! (x3<x1); //false (x3 si es menor que x1)  
System.out.println("C17: "+b);  
  
//ni x3 es mayor que x1, ni c2 es distinto a c1  
b=!(x3>x1) && !(c2!=c1); //false (c2 si es distinto a c1 )  
System.out.println("C18: "+b);  
  
//no se cumple que x1 es menor que 100 y x2 es mayor que 10  
b=!(x1<100 && x2>10); //true  
System.out.println("C19: "+b);  
  
//c2 es anterior alfabéticamente a c1  
b=c2<c1; //false (la 'S' es posterior a la 'F')  
System.out.println("C20: "+b);  
  
}  
}
```

**Nota importante:** el tipo de datos **char** es internamente un número, por eso se pueden comparar usando operadores relacionales, pero esto no es posible con cadenas de caracteres (**String**).

## 12.5. OPERADOR CONDICIONAL.

El **operador condicional** “**? :**” sirve para **evaluar una condición y devolver un resultado u otro** en función de si es verdadera o falsa dicha condición. Es el único **operador ternario** de Java, y como tal, **necesita tres operandos** para formar una expresión.

- ✓ El **primer operando** se sitúa **a la izquierda del símbolo de interrogación (?)**, y siempre será una **expresión booleana**, también llamada condición.
- ✓ El **siguiente operando** se sitúa **a la derecha del símbolo de interrogación (?) y antes de los dos puntos (:)**, y es el **valor que devolverá el operador condicional si la condición es verdadera**.
- ✓ El **tercer y último operando**, que aparece **después de los dos puntos (:)**, es la **expresión cuyo resultado se devolverá si la condición evaluada es falsa**.

**Operador condicional en Java**

Operador	Expresión en Java
?:	condición ? expl : exp2

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Por ejemplo, en la expresión:

```
(x>y) ? x : y ;
```

Se evalúa la condición de si **x** es mayor que **y**. En caso afirmativo se devuelve el valor de la variable **x**, y en caso contrario se devuelve el valor de **y**. Se trata de una manera muy elegante de calcular el **máximo** de esos dos valores.



### Para saber más

El operador condicional se puede sustituir por la sentencia `if-then-else` que veremos en la siguiente unidad sobre las **estructuras de control**, de la que viene a ser una versión abreviada muy útil para algunos casos, permitiendo hacer operaciones potentes con una única sentencia bastante simple. Por ejemplo, la línea anterior de ejemplo, devuelve el mayor de los dos números que se comparan, de una manera sintética y elegante, aunque lo mismo se podría haber hecho usando una sentencia `if-then-else`.



### Ejercicio Resuelto

El siguiente código generará un número aleatorio entre 0 y 100 (aunque el 100 no estará incluido):

```
//d contendrá un número aleatorio entre 0 y 100.  
double d=Math.random()*100;
```

Usando exclusivamente el operador condicional, junto con operadores de relación, lógicos y de asignación, escribe un código en Java que muestre por pantalla si el número aleatorio generado está entre los rangos siguientes:

- ✓ **d** está entre 0 y 20, 20 no incluido.
- ✓ **d** está entre 20 y 50, ambos incluidos.
- ✓ **d** está entre 50 y 75, ninguno incluido.
- ✓ **d** está entre 75 y 100, ambos incluidos.

[Mostrar retroalimentación](#)

Una solución posible podría ser:

```
public class EjercicioOperadorCondicional {  
    public static void main(String[] args) {  
        //d contendrá un número aleatorio entre 0 y 100.  
        double d= Math.random() * 100;  
  
        String cadena;  
        // Comprobamos si d está entre 0 y 20, 20 no incluido.  
        cadena= (d<20) ? "Sí está entre 0 y 20." : "No está entre 0 y 20."  
        System.out.println (cadena);  
        // Comprobamos si d está entre 20 y 50, ambos incluidos.  
        cadena= (d>=20 && d<=50) ? "Sí está entre 20 y 50." : "No está entre 20 y 50."  
        System.out.println (cadena);  
        // Comprobamos si d está entre 50 y 75, ninguno incluido.  
        cadena= (d>50 && d<75) ? "Sí está entre 50 y 75." : "No está entre 50 y 75."  
        System.out.println (cadena);  
        // Comprobamos si d está entre 75 y 100, ambos incluidos.  
    }  
}
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
cadena= (d>=75 && d<=100) ? "Sí está entre 75 y 100" : "No está entre 75 y 100.";  
System.out.println (cadena);  
// Mostramos el número aleatorio  
System.out.println ("El número aleatorio es:" + d);  
}  
}
```



### Ejercicio Resuelto

Sabemos si un número es **par** o **impar** (divisible entre dos) si el resto de la división entera de ese número entre dos es **cero** o **uno**.

Escribe un programa en Java que pida un número entero al usuario e indique si ese número es **par** o **impar**.

[Mostrar retroalimentación](#)

Dado que dependiendo de que el número sea par o impar habrá que mostrar un texto diferente por pantalla ("par" o "impar"), podríamos utilizar el operador condicional (?) para decidir qué mensaje se va a mostrar:

```
String mensajeResultado = (numero % 2 == 0) ? "par" : "impar";
```

Por tanto, el programa que realice todo el proceso podría quedar así:

```
import java.util.Scanner;  
  
public class CalcularParImpar {  
    public static void main(String args[]) {  
        Scanner scanner = new Scanner(System.in);  
        int numero;  
        String mensajeResultado;  
  
        System.out.print("Introduce un número entero: ");  
        numero = scanner.nextInt();  
        mensajeResultado = (numero % 2 == 0) ? "par" : "impar" ;  
        System.out.println ("El número es " + mensajeResultado);  
    }  
}
```

## 12.6. OPERADORES DE BITS.

Los operadores a nivel de bits se caracterizan porque realizan operaciones sobre números enteros (o *char*) en su representación binaria, es decir, sobre cada dígito binario.

Aunque estos operadores no son de uso frecuente, sino más bien para aplicaciones muy específicas, no está de más que al menos sepas cuáles son. En la tabla tienes los operadores a nivel de bits que utiliza Java.

### Operadores a nivel de bits en Java

Operador	Ejemplo en Java	Significado
<code>~</code>	<code>~op</code>	Realiza el complemento binario de <code>op</code> (invierte el valor de cada bit)
<code>&amp;</code>	<code>op1 &amp; op2</code>	Realiza la operación AND binaria sobre <code>op1</code> y <code>op2</code>
<code> </code>	<code>op1   op2</code>	Realiza la operación OR binaria sobre <code>op1</code> y <code>op2</code>
<code>^</code>	<code>op1 ^ op2</code>	Realiza la operación OR-exclusivo (XOR) binaria sobre <code>op1</code> y <code>op2</code>
<code>&lt;&lt;</code>	<code>op1 &lt;&lt; op2</code>	Desplaza <code>op2</code> veces hacia la izquierda los bits de <code>op1</code>
<code>&gt;&gt;</code>	<code>op1 &gt;&gt; op2</code>	Desplaza <code>op2</code> veces hacia la derecha los bits de <code>op1</code>
<code>&gt;&gt;&gt;</code>	<code>op1 &gt;&gt;&gt; op2</code>	Desplaza <code>op2</code> (en positivo) veces hacia la derecha los bits de <code>op1</code>



### Para saber más

Los operadores de bits raramente los vas a utilizar en tus aplicaciones de gestión. No obstante, si sientes curiosidad sobre su funcionamiento, puedes ver el siguiente enlace dedicado a este tipo de operadores:

[Operadores de bits](#)

[https://es.wikibooks.org/wiki/Programaci%C3%B3n\\_en\\_Java/Operadores\\_de\\_bits](https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/Operadores_de_bits)



### Ejercicio Resuelto

En el siguiente código, la variable `i` almacena el número `129`, pero en vez de darle valor con un literal decimal, se le ha dado con un literal binario:

```
int i = 0b1000001;
System.out.println(Integer.toBinaryString(i));
System.out.println(i);
```

En la segunda línea, se muestra el número `i` en binario por la salida estándar. Dado el código anterior, el objetivo es ampliar el código para conseguir lo siguiente:

- 1.- Modificando el valor de la variable `i` con la operación OR binaria, haz que muestre `10000011` (`131` en decimal).
- 2.- Partiendo del valor del paso 1, modifica el valor de la variable `i` con la operación AND binaria para que muestre `10000010` (`130` en decimal).
- 3.- Partiendo del valor del paso 2, modifica el valor de la variable `i` con el operador de desplazamiento de bits a la derecha, para que muestre `1000001` (`65` en decimal).
- 4.- Partiendo del valor del paso 3, modifica el valor de la variable `i` con el operador de desplazamiento de bits a la izquierda, para que muestre `1000001000` (`520` en decimal).

[Mostrar retroalimentación](#)

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

Una solución posible sería la siguiente:

```
public class EjemploOperadoresDeBits {  
    public static void main(String[] args) {  
        //i contendrá el número 129.  
        int i = 0b10000001;  
        System.out.println(Integer.toBinaryString(i));  
  
        i=i | 0b00000010;  
        System.out.println(Integer.toBinaryString(i));  
        System.out.println(i); //El valor de i ahora es 131  
  
        i=i & 0b10000010;  
        System.out.println(Integer.toBinaryString(i));  
        System.out.println(i); //El valor de i ahora es 130  
  
        i=i >> 1;  
        System.out.println(Integer.toBinaryString(i));  
        System.out.println(i); //El valor de i ahora es 65  
  
        i=i << 3;  
        System.out.println(Integer.toBinaryString(i));  
        System.out.println(i); //El valor de i ahora es 520  
    }  
}
```

Si se ejecuta el programa debería obtenerse por pantalla algo similar a lo siguiente:

```
10000001  
10000011  
131  
10000010  
130  
10000001  
65  
1000001000  
520
```

## 12.7. TRABAJOS CON CADENAS.

Ya hemos visto en el apartado de literales que el objeto ***String*** se corresponde con una secuencia de caracteres entrecomillados, como por ejemplo “*hola*”. Este literal se puede utilizar en Java como si de un tipo de datos primitivo se tratase, y como caso especial, no necesita del operador ***new*** para ser creado.

No se trata aquí de que nos adentremos en lo que es una clase u objeto, puesto que lo veremos en unidades posteriores, y trabajaremos mucho sobre ello. Aquí sólo vamos a utilizar determinadas operaciones que podemos realizar con el objeto ***String***, y lo verás mucho más claro con ejemplos descriptivos.

Para aplicar una operación a una variable de tipo ***String***, escribiremos su nombre seguido de la operación, separados por un punto. Entre las principales operaciones que podemos utilizar para trabajar con cadenas de caracteres están las siguientes:

- **Creación.** Como hemos visto en el apartado de literales, podemos crear una variable de tipo ***String*** simplemente asignándole una cadena de caracteres encerrada entre comillas dobles.
- **Obtención del carácter** que se encuentra en una posición determinada de la cadena. Para ello se utiliza el método ***charAt***.
- **Obtención de longitud.** Si necesitamos saber la longitud de un ***String***, utilizaremos el método ***length***.
- **Concatenación.** Se utiliza el operador + o el método ***concat()*** para concatenar cadenas de caracteres.
- **Comparación.** El método ***equals*** nos devuelve un valor booleano que indica si las cadenas comparadas son o no iguales. El método ***equalsIgnoreCase*** hace lo propio, ignorando las mayúsculas de las cadenas a considerar. Las comparaciones entre objetos ***String*** nunca deben hacerse con el operador ==.
- **Obtención de subcadenas.** Podemos obtener cadenas derivadas de una cadena original con el método ***substring()***, al cual le debemos indicar el inicio y el fin de la subcadena a obtener.
- **Cambio a mayúsculas/minúsculas.** Los métodos ***toUpperCase*** y ***toLowerCase*** devuelven una nueva variable que transforma en mayúsculas o minúsculas, respectivamente, la variable inicial.
- **Conversiones.** Utilizaremos el método ***valueOf*** para convertir un tipo de dato primitivo (***int***, ***long***, ***float***, etc.) a una variable de tipo ***String***.

A continuación, tienes varios ejemplos de las distintas operaciones que podemos realizar con cadenas de caracteres o ***String*** en Java:

```
public class EjemploCadenas {

    public static void main(String[] args) {

        // Cadenas de ejemplo con las que trabajar
        String cadena1 = "CICLO DAM-DAW";
        String cadena2 = "ciclo dam-daw";

        System.out.println ("EJEMPLOS DE OPERACIONES CON CADENAS");
        System.out.println ("-----");

        // Mostramos las cadenas originales
```

# TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
System.out.println ("La cadena cadena1 es " + cadena1);
System.out.println ("La cadena cadena2 es " + cadena2);
System.out.println ();

System.out.println ("Longitud de cadena1: " + cadena1.length());
System.out.println ("Longitud de cadena2: " + cadena2.length());

// Concatenación de cadenas (concat o bien operador +)
System.out.println ("Concatenación cadena1 y cadena2: " + cadena1.concat(cadena2));
System.out.println ("Concatenación cadena2 y cadena1: " + cadena2 + cadena1);

// Comparación de cadenas
System.out.println ("cadena1.equals(cadena2) es: " + cadena1.equals(cadena2));
System.out.println ("cadena1.equalsIgnoreCase(cadena2) es: " +
cadena1.equalsIgnoreCase(cadena2));

// Obtención de subcadenas
System.out.println ("cadena1.substring(0,5) es: " + cadena1.substring(0, 5));

// Pasar a minúsculas
System.out.println ("cadena1.toLowerCase() es: " + cadena1.toLowerCase());
System.out.println();
```

El resultado que se debería obtener es el siguiente:

#### EJEMPLOS DE OPERACIONES CON CADENAS

La cadena cadena1 es CICLO DAM-DAW

La cadena cadena2 es ciclo daw-daw

Longitud de cadena1: 13

Longitud de cadena2: 13

Concatenación cadena1 y cadena2: CICLO DAM-DAWciclo dam-daw

Concatenación cadena2 y cadena1: ciclo ddm-dawCICLO DDM-DAW

`cadena1.equals(cadena2)` es: false

`cadena1.equalsIgnoreCase(cadena2)` es: true

cadena1.substring(0,5) es: CTCI0

cadena1.toLowerCase() es: ciclo daw-daw



## Ejercicio Resuelto

Dada la variable `cadena` tipo `String`, haz que vaya mostrando por pantalla la secuencia siguiente:

```
La casa de
La casa de Juan es
La casa de Juan es el número
La casa de Juan es el número 25.
```

Para ello tienes que ir modificando el valor de la variable `cadena`, partiendo del valor que tiene con anterioridad. Para ello puedes usar una de las dos formas siguientes:

```
1 | cadena=cadena+"texto añadido";
2 | cadena+="texto añadido"
```

Fíjate por último que `25` es un número, y no un texto.

[Mostrar retroalimentación](#)

Una solución posible sería:

```
public class EjemploConcatenacion {
    public static void main(String[] args) {
        String cadena="La";
        cadena+=" casa de";
        System.out.println(cadena);

        cadena+=" Juan es";
        System.out.println(cadena);

        cadena+=" el número ";
        System.out.println(cadena);

        cadena+=25 + ".";
        System.out.println(cadena);
    }
}
```



## Autoevaluación

¿Cuál de las siguientes formas es la forma aconsejada para comparar si dos cadenas son iguales entre sí?

```
cad1.equals(cad2)
```

```
cad1==cad2
```

```
cad1====cad2
```

```
cad1.compareTo(cad2)
```

## Ejercicio Resuelto

El siguiente código Java pregunta al usuario por una palabra y la muestra por pantalla:

```
import java.util.Scanner;

public class Palabra {
    public static void main(String args[]) {
        Scanner teclado=new Scanner(System.in);
        System.out.print ("Introduzca una palabra: ");
        String palabra= teclado.nextLine();
        System.out.print ("La palabra introducida es " + palabra);
    }
}
```

Para obtener el carácter ubicado en una determinada posición de una cadena en Java disponemos del método **charAt** teniendo en cuenta que la primera posición es cero y que la última es la del tamaño de la cadena menos uno.

Por ejemplo, dada la siguiente cadena:

```
String palabra= "ejemplo";
```

podríamos obtener el carácter ubicado en la primera posición aplicando a la variable **palabra** el método **charAt** con el valor **0**:

```
char primeraLetra= palabra.charAt (0);
```

en tal caso obtendríamos el carácter 'e' ¿Cómo obtendríamos el último? De manera similar, pero accediendo a la posición **palabra.length() - 1**:

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
char ultimaLetra= palabra.charAt (palabra.length()-1);
```

en este caso obtendríamos el carácter 'o'.

Amplía el fragmento de código anterior para que además de solicitar una palabra se muestren por pantalla la primera y la última letra de esa palabra.

Por ejemplo, si la palabra introducida es "prueba" el programa debería mostrar por pantalla los caracteres 'p' y 'a'.

Para mostrar el primer y último carácter habrá que usar los métodos **charAt** y **length** tal y como acabamos de ver:

```
import java.util.Scanner;

public class Palabra {
    public static void main(String args[]) {
        Scanner teclado=new Scanner(System.in);
        System.out.print ("Introduzca una palabra: ");
        String palabra= teclado.nextLine();

        char primeraLetra= palabra.charAt (0);
        char ultimaLetra= palabra.charAt ( palabra.length()-1 );
        System.out.println ("La palabra introducida es: " + palabra);
        System.out.println ("La primera letra de la palabra es: " + primeraLetra);
        System.out.println ("La última letra de la palabra es: " + ultimaLetra);
    }
}
```

## 12.8. PRECEDENCIA DE OPERADORES.

El orden de precedencia de los operadores determina la secuencia en que deben realizarse las operaciones cuando en una expresión intervienen operadores de distinto tipo.

Las reglas de precedencia de operadores que utiliza Java coinciden con las reglas de las expresiones del álgebra convencional. Por ejemplo:

- ✓ La multiplicación, división y resto de una operación se evalúan primero. Si dentro de la misma expresión tengo varias operaciones de este tipo, empezaré evaluándolas de izquierda a derecha.
- ✓ La suma y la resta se aplican después que las anteriores. De la misma forma, si dentro de la misma expresión tengo varias sumas y restas empezaré evaluándolas de izquierda a derecha.

A la hora de evaluar una expresión es necesario tener en cuenta la **asociatividad** de los operadores. La asociatividad indica qué operador se evalúa antes, en condiciones de igualdad de precedencia. Los operadores de asignación, el operador condicional (?:), los operadores incrementales (++, --) y el casting son asociativos por la derecha. El resto de operadores son asociativos por la izquierda, es decir, que se empiezan a calcular en el mismo orden en el que están escritos: de izquierda a derecha. Por ejemplo, en la expresión siguiente:

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

10 / 2 \* 5

Realmente la operación que se realiza es **(10 / 2) \* 5**, porque ambos operadores, división y multiplicación, tienen igual precedencia y por tanto se evalúa primero el que antes nos encontramos por la izquierda, que es la división. El resultado de la expresión es **25**. Si fueran asociativos por la derecha, puedes comprobar que el resultado sería diferente, primero multiplicaríamos **2 \* 5** y luego dividiríamos entre **10**, por lo que el resultado sería **1**. En esta otra expresión:

x = y = z = 1

Realmente la operación que se realiza es **x = (y = (z = 1))**. Primero asignamos el valor de **1** a la variable **z**, luego a la variable **y**, para terminar asignando el resultado de esta última asignación a **x**. Si el operador asignación fuera asociativo por la izquierda esta operación no se podría realizar, ya que intentaríamos asignar a **x** el valor de **y**, pero **y** aún no habría sido inicializada.

En la tabla se detalla el orden de precedencia y la asociatividad de los operadores que hemos comentado en este apartado. Los operadores se muestran de mayor a menor precedencia.

**Orden de precedencia de operadores en Java**

Operador	Tipo	Asociatividad
<code>++--</code>	Unario, notación postfija	Derecha
<code>++--+-</code> <code>(cast) ! ~</code>	Unario, notación prefija	Derecha
<code>* / %</code>	Aritméticos	Izquierda
<code>+ -</code>	Aritméticos	Izquierda
<code>&lt;&lt;&gt;&gt;&gt;</code>	Bits	Izquierda
<code>&lt;&lt;= &gt;&gt;=</code>	Relacionales	Izquierda
<code>== !=</code>	Relacionales	Izquierda
<code>&amp;</code>	Lógico, Bits	Izquierda
<code>^</code>	Lógico, Bits	Izquierda
<code> </code>	Lógico, Bits	Izquierda
<code>&amp;&amp;</code>	Lógico	Izquierda
<code>  </code>	Lógico	Izquierda
<code>? :</code>	Operador condicional	Derecha
<code>= += -= *=</code> <code>/= %=</code>	Asignación	Derecha



## Reflexiona

¿Crees que es una buena práctica de programación utilizar paréntesis en expresiones aritméticas complejas, aún cuando no sean necesarios?

[Mostrar retroalimentación](#)

Probablemente acertaste. El uso de paréntesis, incluso cuando no son necesarios, puede hacer más fácil de leer las expresiones aritméticas complejas.

No obstante, usar de forma exhaustiva los paréntesis como si no existieran las reglas de precedencia de operadores puede hacer que las expresiones se alarguen y se hagan más incómodas de manejar de forma innecesaria.

En el término medio está la virtud.



## Ejercicio Resuelto

Dada la variable `a1` tipo `double` inicializada a `10` y la variable `a2` también tipo `double` inicializada a `20`, realiza las siguientes operaciones aritméticas y muestra el resultado por pantalla:

- 1.- Restamos `4` al doble de `a1`.
- 2.- Restamos `4` a `a1` y calculamos el doble.
- 3.- Sumamos `2` a `a1` y dividimos por `12`, a todo ello sumamos `a2`.
- 4.- Dividimos `a2` entre `a1`, y todo ello dividido entre `2`.
- 5.- Dividimos `a2` entre la mitad de `a1`.
- 6.- Restamos a `a2` un cuarto de `a1`.
- 7.- Restamos `a1` a `a2`, y todo ello lo dividimos entre `4`.
- 8.- Dividimos `a2` entre `a1`, y todo ello lo multiplicamos por `2`.
- 9.- Dividimos `a2` entre el doble de `a1`.
- 10.- Restamos al doble de `a2` un cuarto de `a1`.
- 11.- Multiplicamos `a2` por `100` menos `a1`.
- 12.- Multiplicamos `a2` por `50` mas `a1`, y todo ello lo dividimos por `10`.

En este ejercicio es importante que intentes minimizar el uso de paréntesis, usándolos solo cuando sean necesarios.

El resultado que debería aparecer por pantalla debería ser similar al siguiente:

```
Ej1: 16.0  
Ej2: 12.0  
Ej3: 21.0  
Ej4: 1.0  
Ej5: 4.0  
Ej6: 17.5  
Ej7: 2.5  
Ej8: 4.0  
Ej9: 1.0  
Ej10: 37.5  
Ej11: 1800.0  
Ej12: 120.0
```

[Mostrar retroalimentación](#)

Una solución posible sería:

```
public class EjercicioPrecedenciaOperadores {  
    public static void main(String[] args) {  
        double d1=10, d2=20;  
        double calc;  
        //Restamos 4 al doble de d1.  
        calc=d1*2 - 4;  
        System.out.println("Ej1: "+calc);  
  
        //Restamos 4 a d1 y calculamos el doble.  
        calc=(d1-4)*2;  
        System.out.println("Ej2: "+calc);  
    }  
}
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
//Sumamos 2 a d1 y dividimos por 12, a todo ello sumamos d2.  
calc=(d1+2)/12 + d2;  
System.out.println("Ej3: "+calc);  
  
//Dividimos d2 entre d1, y todo ello dividido entre 2.  
calc=d2/d1/2;  
System.out.println("Ej4: "+calc);  
  
//Dividimos d2 entre la mitad de d1.  
calc=d2/(d1/2);  
System.out.println("Ej5: "+calc);  
  
//Restamos a d2 un cuarto de d1.  
calc=d2-d1/4;  
System.out.println("Ej6: "+calc);  
  
//Restamos d1 a d2, y todo ello lo dividimos entre 4.  
calc=(d2-d1)/4;  
System.out.println("Ej7: "+calc);  
  
//Dividimos d2 entre d1, y todo ello lo multiplicamos por 2.  
calc=d2/d1*2;  
System.out.println("Ej8: "+calc);  
  
//Dividimos d2 entre el doble de d1.  
calc=d2/(d1*2);  
System.out.println("Ej9: "+calc);  
  
//Restamos al doble de d2 un cuarto de d1.  
calc=d2*2 - d1/4;  
System.out.println("Ej10: "+calc);  
  
//Multiplicamos d2 por 100 menos d1.  
calc=d2 * (100 - d1);  
System.out.println("Ej11: "+calc);  
  
//Multiplicamos d2 por 50 mas d1, y todo ello lo dividimos por 10.  
calc=d2 * (50 + d1) / 10;  
System.out.println("Ej12: "+calc);  
  
}  
}
```

## 13. CONVERSIÓN DE TIPO.

### Caso práctico

**María** ha avanzado mucho en sus conocimientos sobre Java y ha contado con mucha ayuda por parte de **Juan**. Ahora mismo tiene un problema con el código, y le comenta:

—Estoy atrancada en el código. Tengo una variable de tipo **byte** y quiero asignarle un valor de tipo **int**, pero el compilador me da un error de posible pérdida de precisión. ¿Tú sabes qué significa eso?

—Claro —le contesta **Juan**—, es un problema de conversión de tipos, para asignarle el valor a la variable de tipo **byte** debes hacer un casting.

—¡Ah! , ¿y cómo se hace eso?

Imagina que queremos dividir un número entre otro: ¿tendrá decimales el resultado de esa división?

Podemos pensar que siempre que el denominador no sea divisible entre el divisor, tendremos un resultado con decimales, pero no es así.

**Si el denominador y el divisor son variables de tipo entero, el resultado será entero y no tendrá decimales.**

Para que el resultado tenga decimales necesitaremos hacer una **conversión de tipo**.

Las conversiones de tipo se realizan para hacer que el resultado de una expresión sea del tipo que nosotros deseamos. En el ejemplo anterior, para hacer que el resultado de la división sea de tipo real, con decimales, debemos hacer una conversión de tipo. Existen dos tipos de conversiones:

- **Conversiones automáticas.** Cuando a una variable de un tipo se le asigna un valor de otro tipo numérico con menos bits para su representación, se realiza una conversión automática. En ese caso, el valor se dice que es **promocionado** al tipo más grande (el de la variable), para poder hacer la asignación. También se realizan conversiones automáticas en las operaciones aritméticas, cuando estamos utilizando valores de distinto tipo, el valor más pequeño se promociona al valor más grande, ya que el tipo mayor siempre podrá representar cualquier valor del tipo menor (por ejemplo, de **int** a **long** o de **float** a **double**).
- **Conversiones explícitas.** Cuando hacemos una conversión de un tipo con más bits a un tipo con menos bits. En estos casos debemos indicar que queremos hacer la conversión de manera expresa, ya que se puede producir una pérdida de datos y hemos de ser conscientes de ello. Este tipo de conversiones se realiza con el **operador cast**. El operador **cast** es un operador unario que se forma colocando delante del valor a convertir el tipo de dato entre paréntesis. Tiene la misma precedencia que el resto de operadores unarios y se asocia de derecha a izquierda. Las conversiones explícitas también suelen ser conocidas como **casting**.

Debemos tener en cuenta que **un valor numérico nunca puede ser asignado a una variable de un tipo menor en rango, si no es con una conversión explícita**. Por ejemplo:

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

```
int a;  
byte b;  
a = 12;           // no se realiza conversión alguna  
b = 12;           // se permite porque 12 está dentro  
                  // del rango permitido de valores para b  
b = a;            // error, no permitido (incluso aunque  
                  // 12 podría almacenarse en un byte)  
byte b = (byte) a; // Correcto, forzamos conversión explícita
```

En el ejemplo anterior vemos un caso típico de error de tipos, ya que estamos intentando asignarle a **b** el valor de **a**, siendo **b** de un tipo más pequeño. El compilador detecta que **a** es "un recipiente más grande", y que por tanto si intentamos volcar su contenido en **b** que es un recipiente más pequeño, puede no caber, así que mejor avisar del error, y no dejar ni intentarlo, no sea que se pierda el contenido "por desbordamiento del recipiente".

Lo correcto es forzar la conversión de **a** al tipo de datos **byte**, (puesto que su contenido, **12**, realmente puede ser visto como un literal **byte** también) y entonces asignarle su valor a la variable **b**.



### Debes conocer

En el Anexo II de esta misma unidad hay información importante sobre cómo se producen las conversiones de tipos en Java, tanto automáticas como explícitas.

## 14. COMENTARIOS.

### Caso práctico

**Juan** ha podido comprobar los avances que ha hecho **María** con la programación. Ya domina todos los aspectos básicos sobre sintaxis, estructura de un programa, variables y tipos de datos. **Ada** le acaba de comunicar que van a sumarse al proyecto dos personas más, **Ana** y **Carlos** que están haciendo las prácticas en la empresa, vienen del ciclo de Desarrollo de Aplicaciones Multiplataforma y Desarrollo de Aplicaciones Web, respectivamente.

—Al principio de cada programa indicaremos una breve descripción y el autor. En operaciones complicadas podríamos añadir un comentario, les ayudará a entender mejor qué es lo que hace — indica **Juan**.

—De acuerdo —comenta **María**—, y podemos ir metiendo los comentarios de la herramienta esa que me comentaste, **Javadoc**, para que se cree una documentación aún más completa.

— ¡Ajá! , pues ¡manos a la obra!

Los comentarios son muy importantes a la hora de describir qué hace un determinado programa. A lo largo de la unidad los hemos utilizado para documentar los ejemplos y mejorar la comprensión del código. Para lograr ese objetivo, es normal que cada programa comience con unas líneas de comentario que indiquen, al menos, una breve descripción del programa, el autor o autora del mismo y la última fecha en que se ha modificado.

Todos los lenguajes de programación disponen de alguna forma de introducir comentarios en el código. En el caso de Java, nos podemos encontrar los siguientes tipos de comentarios:

- ✓ **Comentarios de una sola línea.** Utilizaremos el delimitador `//` para introducir comentarios de sólo una línea.

```
// comentario de una sola línea
```

- ✓ **Comentarios de múltiples líneas.** Para introducir este tipo de comentarios, utilizaremos una barra inclinada y un asterisco (`/*`), al principio del párrafo y un asterisco seguido de una barra inclinada (`*/`) al final del mismo. Aunque no es necesario poner asteriscos al comienzo de cada nueva línea dentro del comentario, suele hacerse para que de esa forma quede visualmente delimitado qué líneas abarca el comentario. De esta forma, en el ejemplo siguiente se ve cómo suele ponerse el comentario primero... y luego se repite con lo que sería estrictamente necesario

```
/* Esto es un comentario
 * de varias líneas */
```

```
/* Esto es otro comentario
 de varias líneas */
```

## TEMA 1: INTRODUCCIÓN A LA PROGRAMACIÓN

- ✓ **Comentarios Javadoc.** Utilizaremos los delimitadores `/**` y `*/`. Al igual que con los comentarios tradicionales, el texto entre estos delimitadores será ignorado por el compilador. Este tipo de comentarios se emplean para generar documentación automática del programa. A través del programa **javadoc**, incluido en Java SE, se recogen todos estos comentarios y se llevan a un documento en formato **.html**. Al igual que en los comentarios de múltiples líneas, no serían necesarios los asteriscos iniciales para la segunda, tercera y cuarta líneas del comentario siguiente.

```
/** Comentario de documentación.  
 * Javadoc extrae los comentarios del código y  
 * genera un archivo html a partir de este tipo de comentarios  
 */
```



### Reflexiona

Una buena práctica de programación es añadir en la última llave que delimita cada bloque de código, un comentario indicando a qué clase o método pertenece esa llave.



### Para saber más

Si quieras ir familiarizándote con la información que hay en la web de Oracle, en el siguiente enlace puedes encontrar más información sobre la herramienta **Javadoc** incluida en el Kit de Desarrollo de Java SE (en inglés):

[Página oficial de Oracle sobre la herramienta Javadoc](https://docs.oracle.com/javase/6/docs/technotes/guides/javadoc/index.html)

<https://docs.oracle.com/javase/6/docs/technotes/guides/javadoc/index.html>

## ANEXO I. LISTADO DE IDEs

### Listado de diferentes IDEs de trabajo para Java.

Herramienta
<a href="#">BlueJ Java IDE</a>
<a href="#">DrJava Java IDE</a>
<a href="#">Eclipse (multiplataforma)</a>
<a href="#">Javelin from Step Ahead SW (Windows)</a>
<a href="#">JCreator Java IDE LE (Light Edition) (Windows)</a>
<a href="#">JEdit</a>
<a href="#">jGRASP</a>
<a href="#">JIPE Java IDE</a>
<a href="#">IBM Rational Application Developer (Windows y Gnu/Linux)</a>
<a href="#">IDEA</a>
<a href="#">NetBeans</a>
<a href="#">Oracle JDeveloper</a>
<a href="#">SlickEdit</a>
<a href="#">Stylus Studio</a>
<a href="#">UltraStudio</a>
<a href="#">Visual Paradigm Integrated Development Environment (VP-JIDE)</a>
<a href="#">XEmacs</a>

## ANEXO II. CONVERSIÓN DE TIPOS DE DATOS EN JAVA.

---

**Tabla de Conversión de Tipos de Datos Primitivos**

		Tipo destino							
		boolean	char	byte	short	int	long	float	double
Tipo origen	boolean	-	N	N	N	N	N	N	N
	char	N	-	C	C	CI	CI	CI	CI
	byte	N	C	-	CI	CI	CI	CI	CI
	short	N	C	C	-	CI	CI	CI	CI
	int	N	C	C	C	-	CI	CI*	CI
	long	N	C	C	C	C	-	CI*	CI*
	float	N	C	C	C	C	C	-	CI
	double	N	C	C	C	C	C	C	-

Explicación de los símbolos utilizados:

- **N:** Conversión no permitida (un **boolean** no se puede convertir a ningún otro tipo y viceversa).
- **CI:** Conversión implícita o automática. Un asterisco indica que puede haber posible pérdida de datos.
- **C:** Casting de tipos o conversión explícita.
- **-:** Mismo tipo. No hay que convertir nada.

El asterisco indica que puede haber **una posible pérdida de datos**, por ejemplo al convertir un número de tipo **int** que usa los 32 bits posibles de la representación, a un tipo **float**, que también usa 32 bits para la representación, pero 8 de los cuales son para el exponente.

En cualquier caso, las conversiones de números en coma flotante a números enteros siempre necesitarán un **casting**, y deberemos tener mucho cuidado debido a la pérdida de precisión que ello supone.

### Reglas de Promoción de Tipos de Datos.

Cuando en una expresión hay datos o variables de distinto tipo, el compilador realiza la promoción de unos tipos en otros, para obtener como resultado el tipo final de la expresión. Esta promoción de tipos se hace siguiendo unas reglas básicas en base a las cuales se realiza esta promoción de tipos, y resumidamente son las siguientes:

- Si uno de los operandos es de tipo **double**, el otro es convertido a **double**.
- En cualquier otro caso:
  - Si uno de los operandos es **float**, el otro se convierte a **float**.
  - Si uno de los operandos es **long**, el otro se convierte a **long**.
  - Si no se cumple ninguna de las condiciones anteriores, (ningún operando es **double**, ni **float**, ni **long**) entonces ambos operandos son convertidos al tipo **int**.

## Tabla sobre otras consideraciones con los Tipos de Datos

Conversiones de números en coma flotante (float, double) a enteros (int)	Conversiones entre caracteres (char) y enteros (int)	Conversiones de tipo con cadenas de caracteres (String)
<p>Cuando convertimos números en coma flotante a números enteros, la parte decimal se trunca (redondeo a cero). Si queremos hacer otro tipo de redondeo, podemos utilizar, entre otras, las siguientes funciones:</p> <ul style="list-style-type: none"> <li>✓ <b>Math.round(numero):</b> Redondeo al siguiente número entero.</li> <li>✓ <b>Math.ceil(numero):</b> El menor de los enteros que sigue siendo mayor o igual a numero.</li> <li>✓ <b>Math.floor(numero):</b> El mayor de los enteros que sigue siendo inferior o igual a numero.</li> </ul> <pre>double numero=3.5; char c; numero = (int) 'A'; // numero = 65 c = (char) 65; // c = 'A' c = (char) ((int) 'A' + 1); // c = 'B'</pre>	<p>Como un tipo char lo que guarda en realidad es el código Unicode de un carácter, los caracteres pueden ser considerados como números enteros sin signo.</p> <p><b>Ejemplo:</b></p> <pre>int numero; char c; numero = (int) 'A'; // numero = 65 c = (char) 65; // c = 'A' c = (char) ((int) 'A' + 1); // c = 'B'</pre>	<p>Para convertir cadenas de texto a otros tipos de datos se utilizan las siguientes funciones:</p> <pre>numero=Byte.parseByte(cadena); numero=Short.parseShort(cadena); numero=Integer.parseInt(cadena); numero=Long.parseLong(cadena); numero=Float.parseFloat(cadena); numero=Double.parseDouble(cadena);</pre>
<p>Por ejemplo, si hemos leído de teclado un número que está almacenado en una variable de tipo String llamada cadena, y lo queremos convertir al tipo de datos byte, haríamos lo siguiente:</p> <pre>byte n=Byte.parseByte(cadena);</pre>		