

## Unidad 2. Introducción a java

1	Introducción .....	4
1.1	Un bocado de historia.....	5
2	Compiladores, intérpretes y la Máquina Virtual Java (JVM) .....	6
2.1	Como trabaja la JVM .....	11
3	Herramientas para desarrolladores.....	11
3.1	Instalación de JDK .....	12
3.1.1	Instalando la JDK y la JVM .....	12
3.2	Estructura del programa Java .....	18
3.2.1	Creando nuestro primer programa java en el bloc de notas. ....	20
3.3	IDE's .....	24
	Componentes habituales de los IDE.....	24
3.4	Eclipse IDE .....	25
3.4.1	Instalación de Eclipse.....	26
3.4.2	Nuevo espacio de trabajo o workspace en eclipse IDE .....	27
4	Fundamentos de Java .....	36
4.1	Terminología .....	36
4.2	Identificadores .....	37
4.3	La sintaxis básica de Java .....	40
4.4	Comentarios de Java .....	42
4.5	Tipos de dato java.....	43
4.5.1	Conversión de tipos Java para tipos primitivos .....	46
	Ampliación o conversión automática de tipos .....	46
	Casteo o conversión explícita .....	48
	Promoción de tipos en expresiones .....	50
4.6	Palabras reservadas en Java .....	51
4.7	Identificadores .....	51
4.8	Variables .....	51
	Cómo las variables se almacenan en la memoria RAM.....	52
4.9	Consola input/output en Java .....	60
4.9.1	Notas Cornell .....	62
4.10	Operadores Java .....	63
4.10.1	Operadores de asignación.....	64
4.10.2	Operadores aritméticos .....	66
4.10.3	Operadores binarios / operadores bit a bit .....	67
	Practica .....	68
4.11	Booleanos. Operadores Java Booleanos .....	69
4.11.1	Operadores que devuelven como resultado booleanos .....	69
	Operadores de comparación de Java .....	69
	Operadores lógicos Java.....	70
4.11.2	Caracteres y cadenas .....	73
5	Estructuras de control en Java.....	77
5.1	Estructuras de control en Java .....	77
5.1.1	Introducción a las sentencias condicionales en Java. ....	79
	Actividad. Notas Cornell: .....	84
	Practica. Pruébalo tú mismo .....	84

5.1.2	If anidados .....	85
	Actividad de la guía .....	88
	Actividad independiente .....	88
5.2	Instrucción Switch en Java .....	88
	Menús e instrucciones de switch .....	92
5.2.1	Tipos Enum y Java Switch.....	95
5.3	Instrucciones iterativas en Java. Bucles .....	97
5.3.1	Variables en instrucciones de bucle. ....	98
	Contadores. ....	98
	Acumuladores. ....	99
	Máximos y Mínimos. ....	99
5.3.2	El bucle While básico .....	99
	Bucles infinitos .....	102
5.3.3	Bucles usados para hacer menús. ....	102
5.3.4	Ejercicio bucle.....	105
5.3.5	Bucle anidado while .....	106
	Cornell notes. Actividad.....	108
	Actividad.....	108
5.3.6	La instrucción do .. while .....	109
5.3.7	Bucle For. ....	113
	Practica guiada .....	116
	Práctica. ....	116
5.3.8	Comparativa entre bucles java.....	117

### Código de colores

**Amarillo:** definiciones importantes. / declaraciones de código de alta relevancia

**Azul:** conceptos y temas / declaraciones de variables

**Rojo:** temas/cuestiones clave

# 1 Introducción

En esta unidad, comenzará a explorar todas las características fundamentales del lenguaje Java. El capítulo tratará de deconstruir primero Java, repasando sus elementos o constituyentes, que aprendemos en la unidad anterior. Después de eso, nos sumergiremos en el propio lenguaje Java. Sintaxis, operadores, tipos y entidades de objetos orientados. Desde la perspectiva del autor, no hay mejor manera de aprender un idioma que usarlo, así que pongámonos a trabajar en Java.

Java se ha convertido en el lenguaje de programación orientado a objetos de elección. Algunos de los conceptos importantes de Java incluyen:

1. Una máquina virtual Java (JVM), que proporciona la base fundamental para la plataforma
2. independencia
3. Técnicas automatizadas de administración del almacenamiento de información, como la recolección de elementos no utilizados
4. Sintaxis del lenguaje que es similar a la del lenguaje C

El resultado es un lenguaje orientado a objetos y eficaz para la programación de aplicaciones. El capítulo tratará de presentar todas las herramientas que Java ofrece para escribir programas. En esta sección se tratan los siguientes temas:

- Componentes clave del lenguaje Java
- Estructura java orientada a objetos
- Características de java de programación orientada a objetos

## 1.1 Un poco de historia

Java es un **lenguaje de programación orientado a objetos** basado en clases que está diseñado para tener el menor número posible de dependencias de implementación. Es un **lenguaje de programación de propósito general** destinado a permitir a los desarrolladores de aplicaciones escribir una vez, **ejecutar en cualquier lugar** (write once, run anywhere WORA), lo que significa que el código Java compilado puede **ejecutarse en todas las plataformas que admiten Java sin la necesidad de volver a compilar**. Las aplicaciones Java se compilan normalmente en bytecode código de bytes que se puede ejecutar en cualquier máquina virtual Java (JVM), independientemente de la arquitectura de computadora subyacente. La **sintaxis de Java es similar a C y C++** pero tiene menos recursos de bajo nivel que cualquiera de ellos. El tiempo de ejecución java proporciona capacidades dinámicas (como la reflexión y la modificación del código de tiempo de ejecución) que normalmente no están disponibles en los lenguajes compilados tradicionales. A partir de 2019, **Java era uno de los lenguajes de programación más populares en uso según GitHub**, particularmente para aplicaciones web cliente-servidor, con un reportado de 9 millones de desarrolladores.

Java fue desarrollado originalmente por **James Gosling en Sun Microsystems** (que desde entonces ha sido adquirido por Oracle) y lanzado en 1995 como un componente central de la plataforma **Java de Sun Microsystems**. Los compiladores Java originales y de implementación de referencia, máquinas virtuales y bibliotecas de clases fueron lanzados originalmente por Sun bajo licencias propietarias. A partir de mayo de 2007, en cumplimiento con las especificaciones del Proceso de la Comunidad Java, Sun había definido la licencia de la mayoría de sus **tecnologías Java bajo la Licencia Pública General de GNU**. **Oracle** ofrece su propia máquina virtual Java HotSpot, sin embargo, la implementación de referencia oficial es la **JVM OpenJDK** que es software de código abierto gratuito y utilizado por la mayoría de los desarrolladores y es la JVM predeterminada para casi todas las distribuciones de Linux.

A partir de octubre de 2021, la última versión es Java 17, con Java 11, una **versión de soporte a largo plazo (LTS)** actualmente compatible, lanzada el 25 de septiembre de 2018. **Oracle lanzó la última actualización pública de costo cero para la versión heredada Java 8 LTS en enero de 2019 para uso comercial**, aunque de lo contrario **seguirá soportando Java 8 con actualizaciones públicas** para uso personal indefinidamente. Otros proveedores han comenzado a ofrecer **compilaciones de costo cero de OpenJDK 8 y 11** que todavía están recibiendo actualizaciones de seguridad y otras.

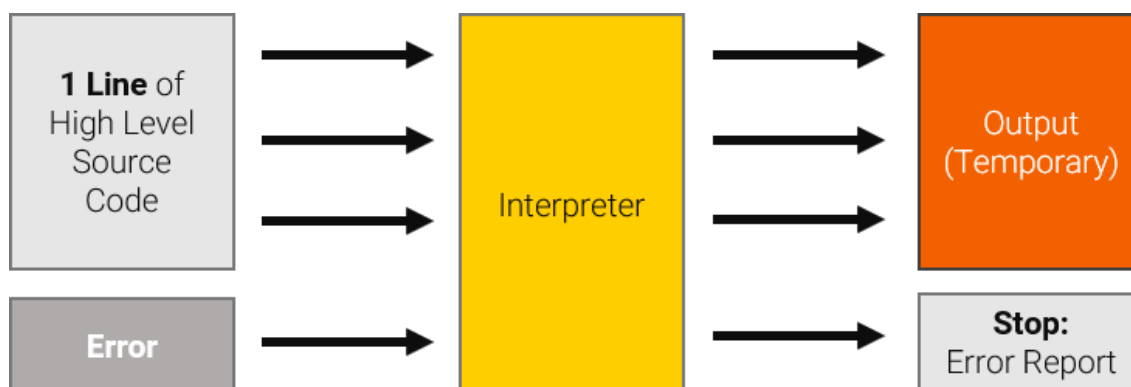
Oracle (y otros) recomiendan encarecidamente desinstalar versiones obsoletas de Java **debido a graves riesgos debido a problemas de seguridad no resueltos**. Dado que Java 9, 10, 12, 13, 14 y 15 ya no son compatibles, Oracle aconseja a sus usuarios que pasen inmediatamente a la última versión (actualmente Java 17) o a una versión LTS.

## 2 Compiladores, intérpretes y la Máquina Virtual Java (JVM)

El lenguaje **Java bytecode** de la Java Virtual Machine consiste en **instrucciones muy simples** que pueden ser **ejecutadas directamente por la CPU de un ordenador**. Casi todos los programas, sin embargo, están escritos en lenguajes de programación de alto nivel, como Java, Pascal o C++. **Un programa escrito en un lenguaje de alto nivel no se puede ejecutar directamente en ningún equipo**. En primer lugar, **debe traducirse al lenguaje de máquina**. Esta traducción puede ser realizada por un **programa llamado compilador**. **Un compilador toma un programa de lenguaje de alto nivel y lo traduce en un programa ejecutable de lenguaje máquina**. Una vez realizada la traducción, el programa de lenguaje de máquina se puede ejecutar cualquier número de veces, pero por supuesto solo se puede ejecutar en un tipo de computadora (ya que cada tipo de computadora tiene su propio lenguaje de máquina individual).



Si **el programa se va a ejecutar en otro tipo de ordenador, otra plataforma**, debe volver a traducirse, utilizando **un compilador diferente, al lenguaje de máquina apropiado**. Hay una alternativa a la compilación de un programa de lenguaje de alto nivel. **En lugar de usar un compilador**, que traduce el programa a la vez, **puede usar un intérprete**, que lo traduce instrucción por instrucción, según sea necesario. Un **intérprete** es un programa que actúa como una CPU, con una especie de ciclo de fetch-and-execute. Para ejecutar un programa, el intérprete se ejecuta en un bucle en el que lee repetidamente una instrucción del programa, decide lo que es necesario para llevar a cabo esa instrucción y, a continuación, genera los comandos de lenguaje de máquina apropiados para hacerlo.



Un posible uso de los intérpretes es ejecutar lenguajes de programas de alto nivel. Por ejemplo, el lenguaje de programación Lisp suele ser ejecutado por un intérprete en lugar de un compilador. Sin embargo, **los intérpretes tienen otro propósito: pueden permitirle usar un programa de lenguaje de máquina destinado a un tipo de computadora en un tipo**

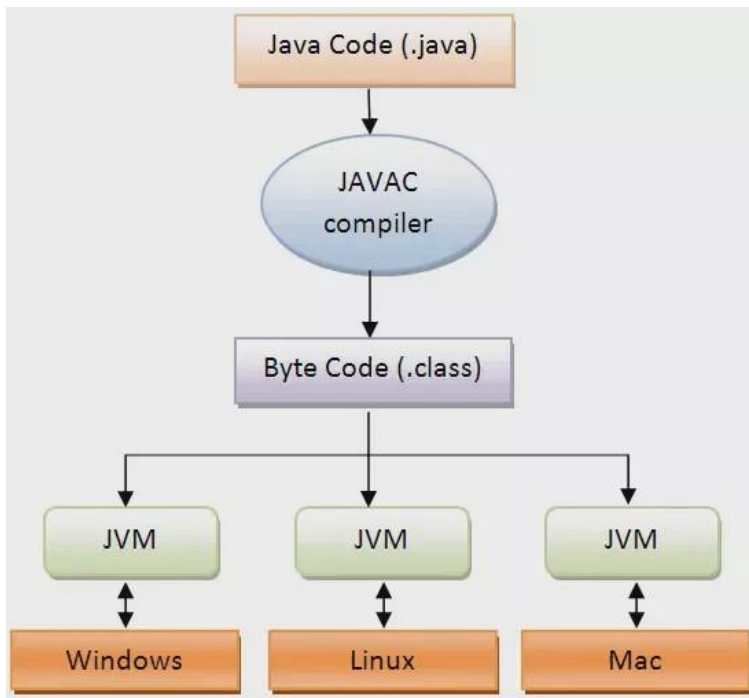
de computadora completamente diferente. Por ejemplo, hay un programa denominado "Virtual PC" que se ejecuta en equipos Macintosh. Virtual PC es un intérprete que ejecuta programas de lenguaje de máquina escritos para equipos clonados de IBM-PC.

Si se ejecuta Virtual PC en Macintosh, puede ejecutar cualquier programa de PC, incluidos los programas escritos para Windows. (Desafortunadamente, un programa de PC se ejecutará mucho más lentamente de lo que lo haría en un clon real de IBM. El problema es que Virtual PC ejecuta varias instrucciones de lenguaje de máquina Macintosh para cada instrucción de lenguaje de máquina de PC en el programa que está interpretando. Los programas compilados son inherentemente más rápidos que los programas interpretados.) En la siguiente imagen, Virtual PC ejecuta Windows a través de Mac OS.



Los diseñadores de Java optaron por utilizar una combinación de compilación e interpretación. Los programas escritos en Java se compilan en lenguaje de máquina, pero es un lenguaje de máquina para una computadora que realmente no existe. Este llamado ordenador "virtual" se conoce como la máquina virtual Java. El lenguaje de máquina para la máquina virtual Java se denomina código de bytes Java.

No hay ninguna razón por la que el bytecode de Java no se pueda usar como el lenguaje de máquina de una computadora real, en lugar de una computadora virtual. Sin embargo, uno de las principales ventajas de Java es que en realidad se puede utilizar en cualquier ordenador. Todo lo que el ordenador necesita es un intérprete de código de bytes Java. Este tipo de intérprete simula la máquina virtual Java de la misma manera que virtual PC, Virtual BOX o vmware, máquina virtual que simula un ordenador PC. Por supuesto, se necesita un intérprete de código de bytes Java diferente para cada plataforma, pero una vez que un ordenador tiene un intérprete de Bytecode Java, puede ejecutar cualquier programa de Java. Y el mismo programa de Java se puede ejecutar en cualquier ordenador que tenga un intérprete de este tipo. Esta es una de las características esenciales de Java: el mismo programa compilado se puede ejecutar en muchos tipos diferentes de computadoras.



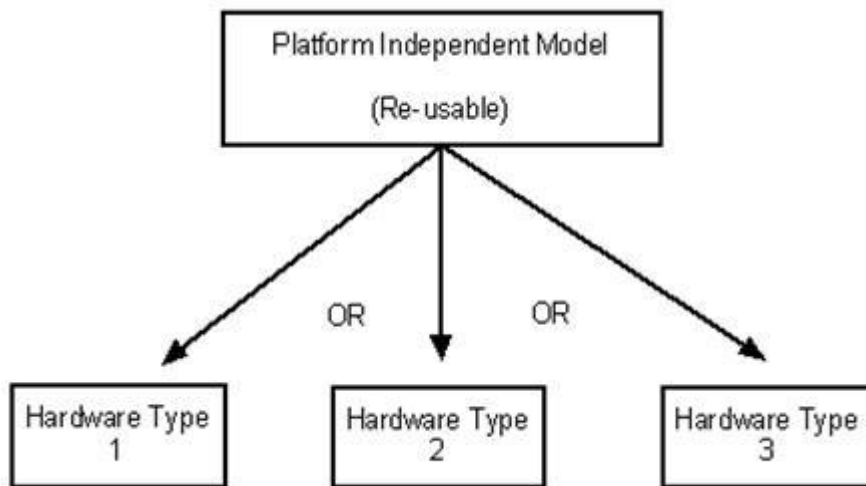
¿Por qué usar el bytecode de java intermedio en absoluto? ¿Por qué no simplemente distribuir el programa Java original y dejar que cada persona lo compile en el lenguaje de máquina de cualquier computadora en la que quiera ejecutarlo? Hay muchas razones. En primer lugar, un compilador tiene que entender Java, un lenguaje complejo de alto nivel. El compilador es en sí mismo un programa complejo. Un intérprete de bytecode Java, por otro lado, es un programa bastante pequeño y simple. Esto facilita la escritura de un intérprete para una nueva plataforma; una vez hecho esto, ese equipo puede ejecutar cualquier programa Java compilado. Sería mucho más difícil escribir un compilador Java para el mismo equipo.

Se tiene en cuenta que no hay ninguna conexión necesaria entre Java y el bytecode Java. Un programa escrito en Java ciertamente podría ser compilado en el lenguaje de máquina de una computadora real. Y los programas escritos en otros lenguajes podrían compilarse en código de bytes Java. Sin embargo, es la combinación de Java y bytecode Java que es independiente de la plataforma, seguro y compatible con la red, al tiempo que le permite programar en un lenguaje moderno orientado a objetos de alto nivel.

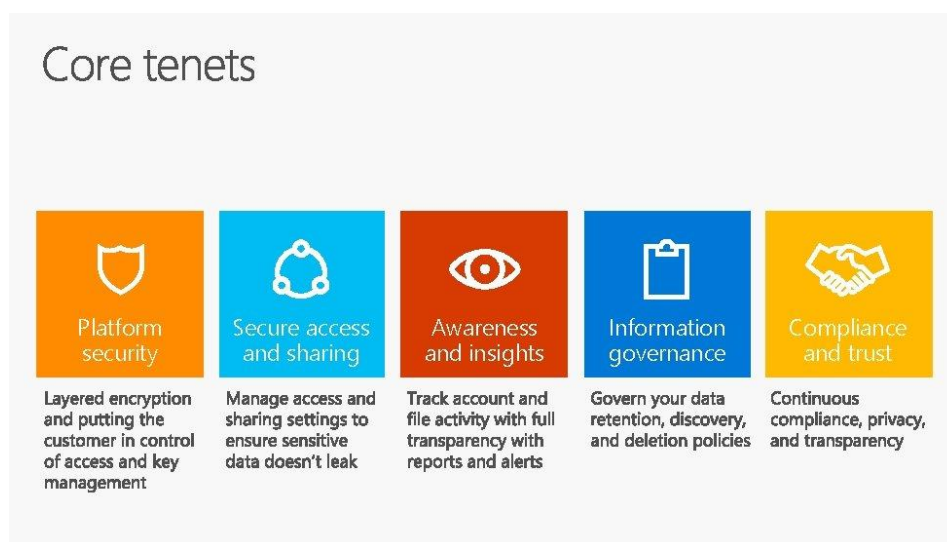
## Conceptos

**Independiente de la plataforma:** Software que puede ejecutarse en una variedad de plataformas de hardware o arquitecturas de software. El software independiente de la plataforma se puede utilizar en muchos entornos diferentes, lo que requiere menos planificación y traducción en toda una empresa. Por ejemplo, el lenguaje de programación Java fue diseñado para ejecutarse en múltiples tipos de hardware y múltiples sistemas operativos. Si la independencia de la plataforma Java se convierte en una realidad, las organizaciones con múltiples tipos de computadoras podrán escribir una aplicación especializada una vez y tendrán que ser utilizadas por prácticamente todo el mundo, en lugar de tener que escribir, distribuir y mantener múltiples versiones del mismo programa.



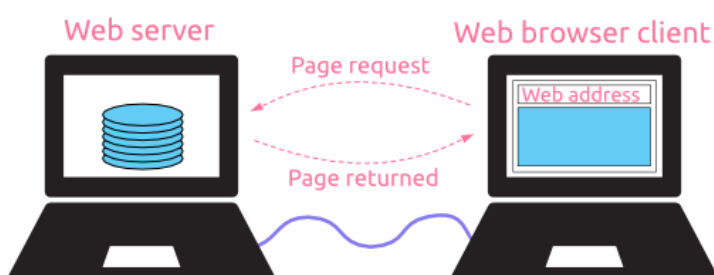


La seguridad de la plataforma o plataforma segura se refiere a la arquitectura de seguridad, herramientas y procesos que garantizan la seguridad de toda una plataforma informática. Utiliza software, sistemas y procesos de seguridad agrupados/unificados para permitir la seguridad del hardware, software, red, almacenamiento y otros componentes de una plataforma informática. Por ejemplo, Java como lenguaje no permite que su programa acceda a una ubicación de memoria que no pertenezca a su programa; Por lo tanto, el programa no puede modificar otros. Esta técnica se denomina Sandbox.



**Plataforma compatible con la red:** la plataforma proporciona todos los medios necesarios, incluidas bibliotecas, software de comunicación, implementaciones de protocolos de red (es decir, TCP/IP, NETBIOS) para construir aplicaciones de red y distribuidas.

Ejemplo de aplicación de red. El modelo cliente-servidor de sitios web que visita a diario.



Ejemplo de aplicación distribuida, blockchain, donde múltiples computadoras comparten la carga de mantener las transacciones de datos blockchain. Blockchain utiliza múltiples computadoras conectadas a una red para hacer trabajo colaborativo.



Java Virtual Machine (JVM) es un motor que proporciona un entorno de ejecución para controlar el código Java o las aplicaciones. Convierte el bytecode a lenguaje máquina. JVM forma parte de Java Run Environment (JRE). En otros lenguajes de programación, el compilador produce código máquina para un sistema determinado. Sin embargo, el compilador de Java produce código para una máquina virtual conocida como máquina virtual Java, que se llama bytecode. Podemos deconstruirlo en tres etapas en la definición, realización e implementación. Por lo tanto, las máquinas virtuales Java son:

1. Una especificación donde la máquina virtual Java se define teóricamente. Es el deber del proveedor elegir cómo implementar esta especificación. Su implementación ha sido proporcionada por Oracle y otras empresas.
2. Una implementación: Su implementación se conoce como JRE (Java Runtime Environment).
3. Instancia de tiempo de ejecución: Siempre que se escribe el comando java en consola para ejecutar una clase java, se crea una instancia de JVM.

## 2.1 Como trabaja la JVM

Resumen: El código Java se compila en código, bytecode. Este código se interpreta en diferentes equipos.

Entre el sistema host y el código fuente Java, Bytecode es un lenguaje intermediario. JVM en Java es responsable de asignar el espacio de memoria.



La JVM realiza la siguiente operación:

1. Carga el código.
2. Comprueba el código.
3. Ejecuta código.
4. Proporciona un entorno de tiempo de ejecución.

JVM proporciona definiciones para:

- **Área de memoria** : donde asignar nuestros programas en memoria para ser ejecutados.
- **Formato de archivo de clase**. Este es el formato de los archivos de código fuente java. La JVM señala cómo gestionarlos.
- **Recolector de basura (garbage collector)**: para liberar la memoria que ya no es utilizada por nuestras aplicaciones.
- **Pila (Heap)**: memoria especial que los programas utilizan para gestionar llamadas a funciones y métodos.
- **Informes de errores irrecurables**: administración de errores de aplicación.

## Java Runtime/ Ejecución

**Runtime** es el período de tiempo en que se está ejecutando un programa. **Comienza cuando se abre (o ejecuta) un programa y termina con el programa se cancela o se cierra.** **Tiempo de ejecución** es un término técnico, utilizado con mayor frecuencia en el desarrollo de software.

Java **Runtime Environment** es una capa de software que se ejecuta sobre el sistema operativo de un ordenador, proporcionando servicios adicionales específicos de Java. El JRE **suaviza la diversidad de sistemas operativos**, asegurando que los programas Java puedan **ejecutarse en prácticamente cualquier sistema operativo** sin modificaciones.

## 3 Herramientas para desarrolladores

Para los usuarios normales, Oracle y la organización **OpenJDK** proporcionan implementaciones para **JRE y JDK**. **Java Runtime Environment** (JRE) es un **conjunto de herramientas de software para el desarrollo de aplicaciones Java**. Combina la máquina virtual Java (JVM), las clases principales de la plataforma y las bibliotecas de soporte.

**JRE es parte del Java Development Kit** (JDK) pero se puede descargar por separado. JRE fue desarrollado originalmente por Sun Microsystems Inc., una subsidiaria de propiedad total de Oracle Corporation.

El **Java Development Kit** es una implementación de cualquiera de las plataformas **Java Platform, Standard Edition, Java Platform, Enterprise Edition o Java Platform, Micro Edition** lanzadas por Oracle Corporation en forma de un producto binario dirigido a desarrolladores Java en Solaris, Linux, macOS o Windows. Proporciona un framework de librerías y clases para permitir a los desarrolladores crear sus propias aplicaciones java.

Oracle Corporation es una corporación multinacional estadounidense de tecnología informática con sede en Austin, Texas. La compañía anteriormente tenía su sede en Redwood Shores, California, hasta diciembre de 2020, cuando trasladó su sede a Texas.

**OpenJDK es una implementación gratuita y de código abierto de Java Platform, Standard Edition**. Es el resultado de un esfuerzo que Sun Microsystems comenzó en 2006. La implementación está licenciada bajo la Licencia Pública General de GNU versión 2 con una excepción de enlace

### **3.1 Instalación de JDK**

El objetivo de este curso es construir desarrolladores profesionales en Java. El primer paso es instalar el Java Development Kit (JDK). En este apartado se explicará cómo instalar las herramientas JDK para Java y configurar las variables de entorno necesarias para trabajar con java en modo consola.

Primero debéis instalar la herramienta JRE, y luego puedes agregar diferentes implementaciones del JDK. Permitirá a los estudiantes darse cuenta de cómo los desarrolladores podemos construir aplicaciones con diferentes versiones del JDK. Por lo tanto, lo primero es lo primero:

#### **3.1.1 Instalando la JDK y la JVM**

Abren el navegador al enlace que se proporciona a continuación al sitio web oficial de Oracle:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html> y descargar el JRE. Sin embargo, en este curso vamos a proporcionar otra versión que mejor se adapte mejor a los objetivos de esta clase. Por lo tanto, puede ejecutar [https://download.oracle.com/java/17/latest/jdk-17\\_windows-x64\\_bin.exe](https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe)

**Java SE Development Kit 17.0.1 downloads**

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications and co programming language.

The JDK includes tools for developing and testing programs written in the Java programming language and running on the Java platform.

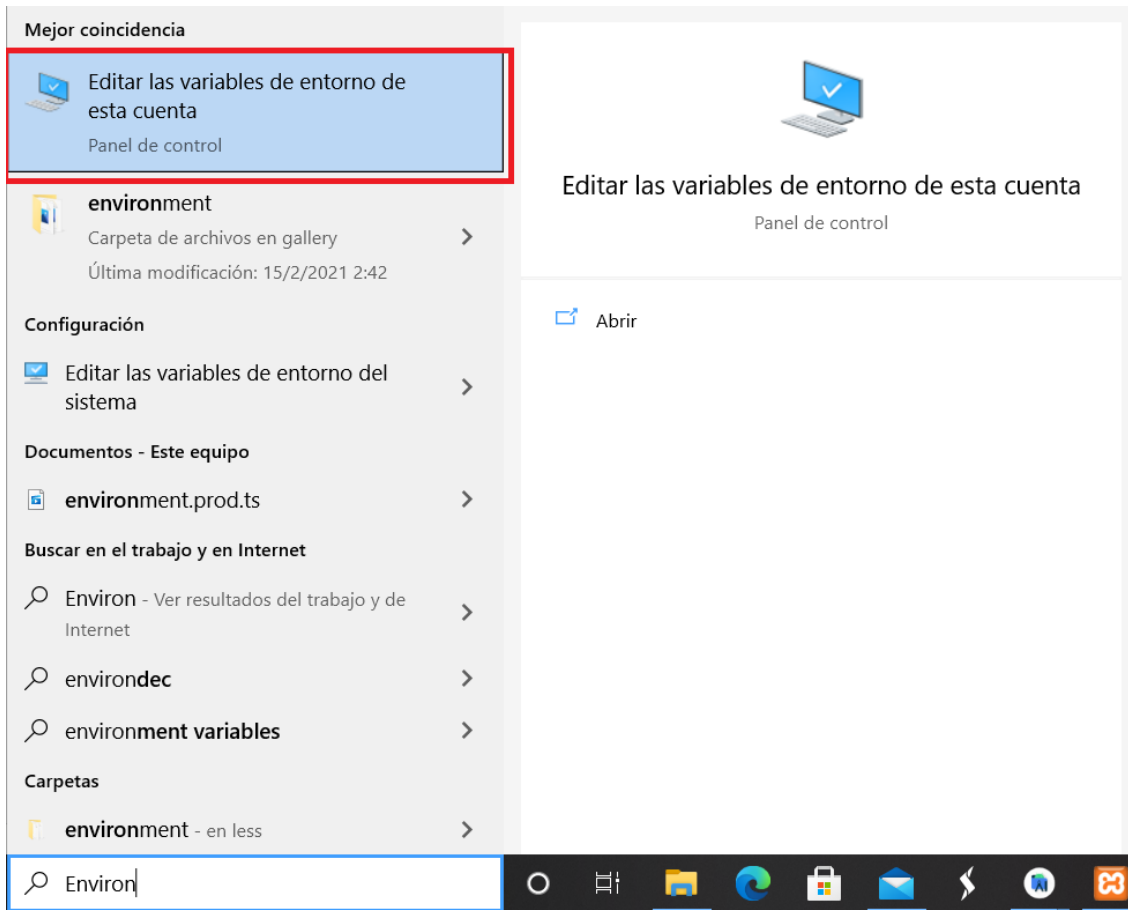
Linux macOS **Windows**

Product/file description	File size	Download
x64 Compressed Archive	170.66 MB	<a href="https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip">https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.zip</a> (sha256 <a href="#">↗</a> )
<b>x64 Installer</b>	152 MB	<a href="https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe">https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.exe</a> (sha256 <a href="#">↗</a> )
x64 MSI Installer	150.89 MB	<a href="https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi">https://download.oracle.com/java/17/latest/jdk-17_windows-x64_bin.msi</a> (sha256 <a href="#">↗</a> )

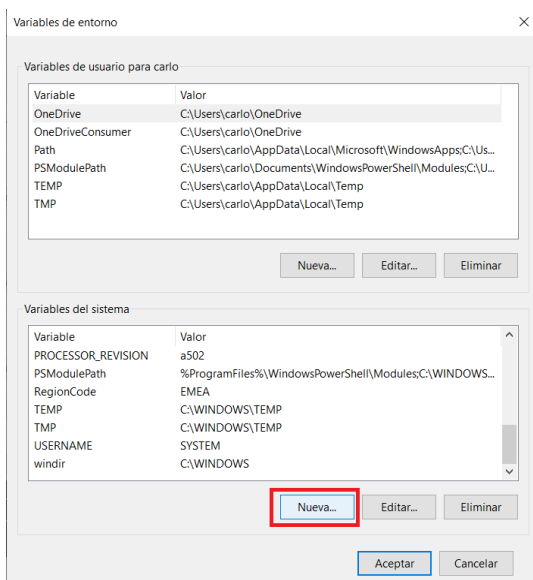
1. Descargar el JRE o JDK indicado en los apuntes para tu sistema operativo.
2. Seguid las instrucciones de instalación que se proporcionan con el JRE.
3. Guardar la vía de acceso al archivo ejecutable de JRE. Lo necesitará más adelante.
4. Continuar con las instrucciones de verificación del entorno JRE.

## Configuración de variables de entorno

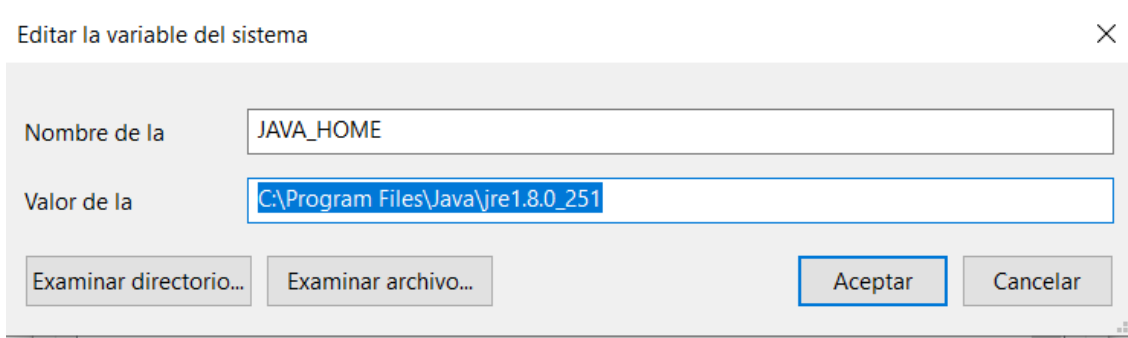
1. Para configurar variables de entorno Java en Windows, debe escribir en el área de búsqueda en Variables de entorno de Windows



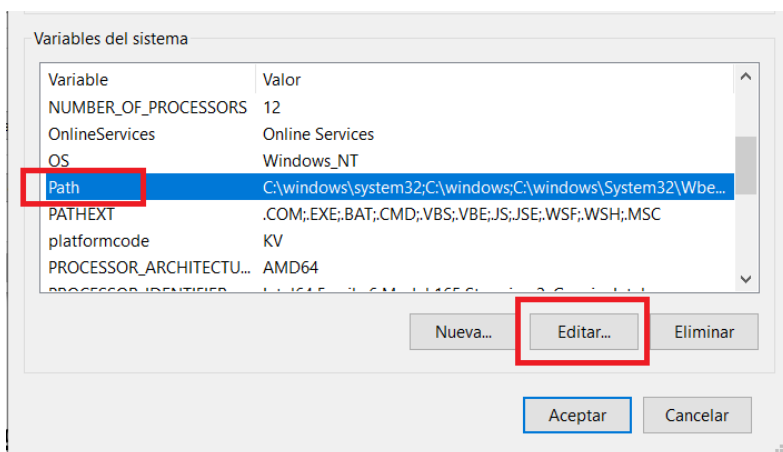
1. Después de eso, abrirá el Editor de variables y presionará el botón Nuevo. Agregamos la variable JAVA\_HOME. Significa dirección o carpeta de contenido de su instalación java.



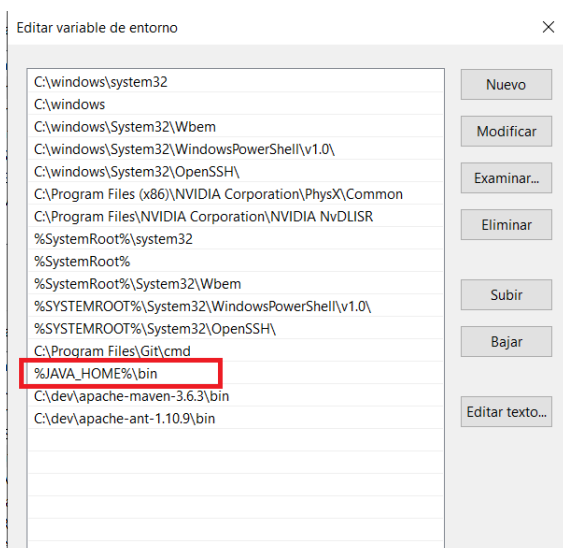
2. **Agrega la variable.**



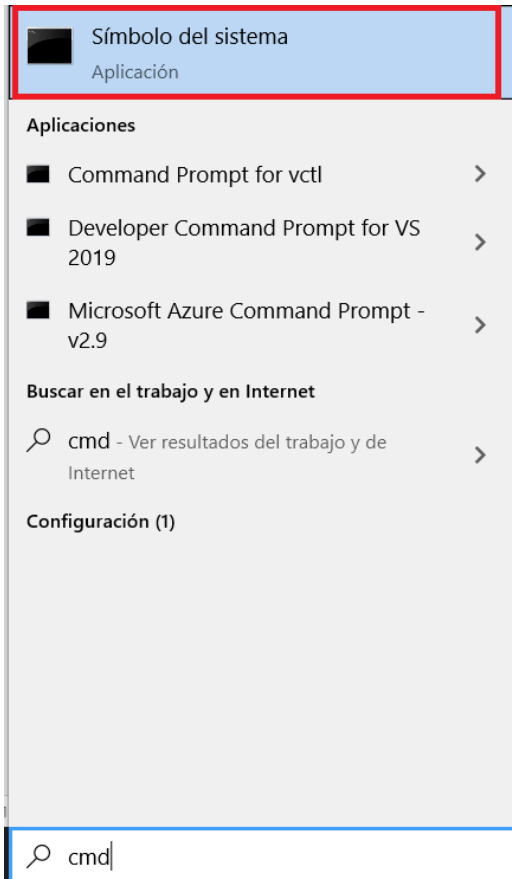
3. El siguiente paso es añadir una entrada Java a la variable Path. En la variable path de windows almacenar todas las rutas de acceso que contiene programas, archivos exec, que deben estar disponibles para ser ejecutados desde cualquier parte o carpeta de sus ventanas. Si añades tu carpeta JRE bin a la vía de acceso podrás ejecutar java desde cualquier ubicación de tu ordenador. Seleccione la variable Ruta y pulse Editar.



4. Debes agregar esta entrada a la ruta de acceso. Es obligatorio para la carpeta bin de instalación de Java aquí porque es donde se encuentra el archivo java.exe. Agregue la entrada marcada a la ruta haciendo clic en nuevo.



5. Ya has configurado JAVA\_HOME en C:\Archivos de programa\Java\jre1.17.0\_XXX. Por lo tanto, a la ruta de acceso ha añadido C:\Archivos de programa\Java\ jre1.17.0\_XXX \bin.
6. El último paso es verificar que java es accesible desde cualquier lugar, el entorno JRE. Por lo tanto, abra la consola de Windows escribiendo cmd en la herramienta de búsqueda de Windows



7. Escribe el siguiente comando en la consola: `java -version`. Puede comparar el resultado con la salida de la siguiente imagen. Si ha configurado todo siguiendo las instrucciones, funcionaría.

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19042.928]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\carlo>java -version
java version "1.8.0_251"
Java(TM) SE Runtime Environment (build 1.8.0_251-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.251-b08, mixed mode)

C:\Users\carlo>
```



<https://developer.ibm.com/tutorials/j-introtojava1/>

Este es el momento de introducir Java desde un punto de vista práctico. En un nivel básico (el nivel de lenguaje de máquina), una computadora puede realizar solo operaciones muy simples. Un equipo realiza tareas complejas encadenando un gran número de estas operaciones. Tales tareas deben ser un "script" completo y correcto para el programa.

### 3.1.2 Programas en Java

Crear **programas complejos nunca será realmente fácil**, pero la dificultad se puede manejar hasta cierto punto dándole al programa una estructura general clara. **El diseño de la estructura general** de un programa es lo que yo llamo **"programación en grande"**.

**La programación en el pequeño**, que a veces se llama codificación, se referiría entonces a rellenar los detalles de ese **diseño**. Los **detalles son las instrucciones explícitas, paso a paso, para realizar tareas a pequeña escala**. Cuando haces codificación, estás trabajando bastante "cerca de la máquina", con algunos de los mismos conceptos que podrías usar en lenguaje de máquina: **ubicaciones de memoria, operaciones aritméticas, bucles y ramas**. En un lenguaje de alto nivel, como Java, se puede trabajar con estos conceptos en un nivel varios pasos por encima del lenguaje de máquina. Sin embargo, te tienes que preocupar por configurar todos los detalles bien.

Los lenguajes de programación **se diferencian de los lenguajes humanos ordinarios en ser completamente inequívocos** y **muy estrictos** sobre lo que está y no está permitido en un programa. Las reglas que determinan lo que se permite se denominan la sintaxis del lenguaje.

**Syntaxis**, las reglas de **sintaxis** especifican el **vocabulario** básico del lenguaje y cómo se pueden construir programas utilizando cosas **como bucles, estructuras de selección y métodos**. Un programa **sintácticamente correcto** es aquel que se puede compilar o interpretar con **éxito**. Los programas que tienen errores de sintaxis serán rechazados (esperemos que con un mensaje de error útil que nos ayude a solucionar el problema).

Por lo tanto, para ser un **programador correcto**, debemos desarrollar un conocimiento detallado de la **sintaxis** del lenguaje de programación que está utilizando. Sin embargo, la sintaxis es sólo una parte de la historia. No basta con escribir un programa que se ejecute, ¡quieres **un programa que se ejecute y produzca el resultado correcto**! Es decir, **el significado del programa tiene que ser correcto**. El significado de un programa se conoce como su **semántica**. Un programa semánticamente correcto es aquel que hace lo que usted quiere que **haga**.

Además, un **programa puede ser sintáctica y semánticamente correcto**, pero aún así ser un programa **bastante malo**. Usar el lenguaje correctamente no es lo mismo que usarlo bien. Por ejemplo, un **buen programa tiene "estilo"**. Está escrito **de una manera que hará que sea fácil para la gente leer y entender**. Sigue convenciones que serán familiares para otros programadores. Y tiene un diseño general que tendrá sentido para los lectores humanos. **La computadora no es consciente de la calidad del software**, pero para un lector humano,

es primordial. Estos aspectos de la programación a veces se conocen como pragmática.

Comenzamos nuestra exploración de Java con el problema que se ha vuelto tradicional para tales inicios: ¡escribir un programa que muestre el mensaje “Hello World!”. Esto puede parecer un problema trivial, pero conseguir que una computadora haga esto es realmente un gran primer paso en el aprendizaje de un nuevo lenguaje de programación (especialmente si es tu primer lenguaje de programación).

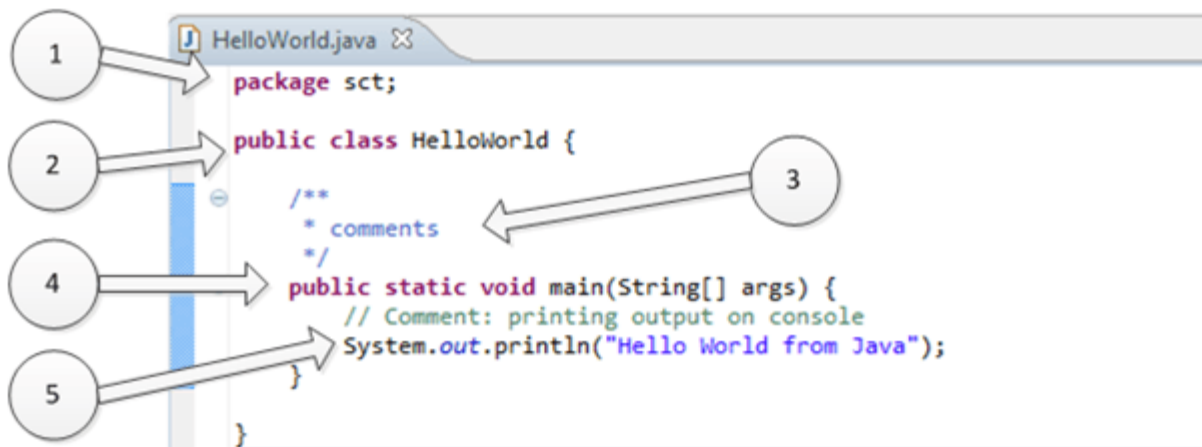
Significa que usted entiende el proceso básico de:

1. obtener el texto del programa en el ordenador,
2. compilar el programa, y
3. ejecutar el programa compilado.

Hasta ahora, la teoría se ha ido introduciendo. Es hora de abrir tu Eclipse y crear tu primer proyecto java. **Primeros pasos**

### 3.2 Estructura del programa Java

Vamos a usar el ejemplo del programa HelloWorld Java para entender la estructura y las características de la clase y del programa. Este programa está escrito en pocas líneas, y su única tarea es imprimir “Hello World from Java” en la pantalla. Consulte la siguiente imagen.



Estructura del programa Java

#### 1. “package sct”:

Es una declaración de declaración de paquete. La instrucción `package` define un espacio de nombres en el que se almacenan las clases. El paquete se utiliza para organizar las clases en función de la funcionalidad. Si omite la instrucción `package`, los nombres de clase se colocan en el paquete predeterminado, que no tiene nombre. La instrucción `package` no puede aparecer en ninguna parte del programa. Debe ser la primera línea de su programa o puede omitirla.

#### 2. “public class HelloWorld”:

Esta línea tiene varios aspectos de la programación java.

1. **public:** Esta es la palabra clave modificadora de acceso que indica al compilador acceso a la clase. Varios valores de modificadores de acceso pueden ser public, protected, private o default (sin valor).
2. **class:** esta palabra clave utilizada para declarar una clase. Nombre de la clase (HelloWorld) seguido de esta palabra clave.

### 3. Sección de comentarios:

Podemos escribir comentarios en java de dos maneras.

- a. **Comentarios de línea:** comienza con dos barras diagonales (//) y continúa hasta el final de la línea actual. Los comentarios de línea no requieren un símbolo final.
- b. **Los comentarios de bloque** comienzan con una barra diagonal y un asterisco (/\*) y terminan con un asterisco y una barra diagonal (/). Los comentarios de bloque también pueden extenderse a través de tantas líneas como sea necesario.

### 4. “public static void main (String [ ] args)”:

Su método (Function) denominado main con array de cadenas como argumento.

- a. **public:** Modificador de acceso
- b. **static:** static es una palabra clave reservada que significa que un método es accesible y utilizable aunque no existan objetos de la clase.
- c. **void:** esta palabra clave declara que el método no devolvería nada. El método puede devolver cualquier primitivo u objeto.
- d. Contenido del método dentro de llaves. { }

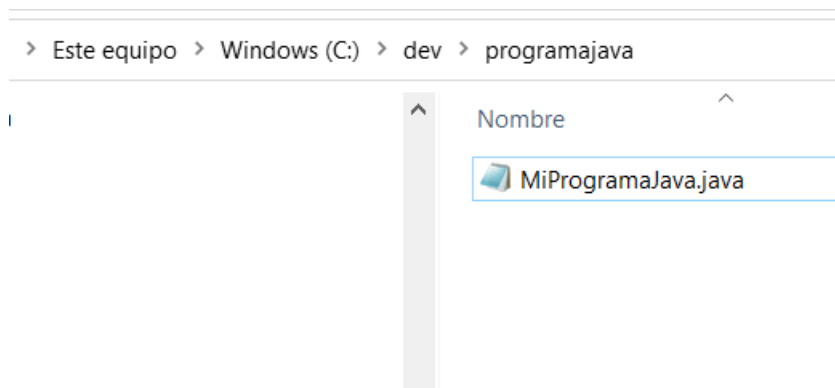
### 5. System.out.println("Hello World from Java") :

1. **Sistema:** Es el nombre de la clase de utilidad Java.
2. **out:** Es un objeto que pertenece a la clase System.
3. **println:** Es el nombre del método de utilidad que se utiliza para enviar cualquier cadena a la consola.
4. **"Hello World from Java":** Es un literal String establecido como argumento para el método println.

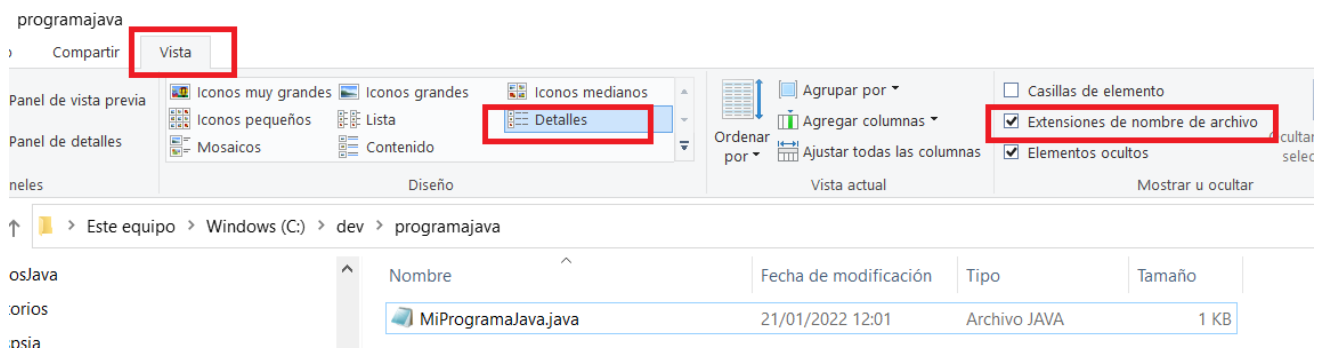
### 3.2.1 Creando nuestro primer programa java en el bloc de notas.

Es hora de empezar a programar en Java. Esta sección proporcionará instrucciones para crear nuestro primer programa java en el bloc de notas, compilarlo y ejecutarlo desde el la línea de comandos.

1. Debes crear una carpeta en c:\dev , denominada c:\dev\programajava y, a continuación, crearía el archivo MiProgramaJava.java como el ejemplo de la ilustración siguiente. (Boton derecho sobre el explorador de Windows, nuevo-> fichero de texto). Cambiad la extensión de .txt a .java. Importante que el fichero se llame igual que la clase Java que crearemos dentro más la extensión .java.




**Nota:** Si no podéis cambiar la extensión a .java, en vuestro explorador de Windows ir al menú Vista y marcar la casilla extensiones de nombre de archivo.



2. Pega este código en el archivo y guárdelo. Tenga en cuenta que el nombre de clase es el mismo que el archivo. Es obligatorio en Java que el nombre del archivo sea el mismo que la clase Java principal en el código.

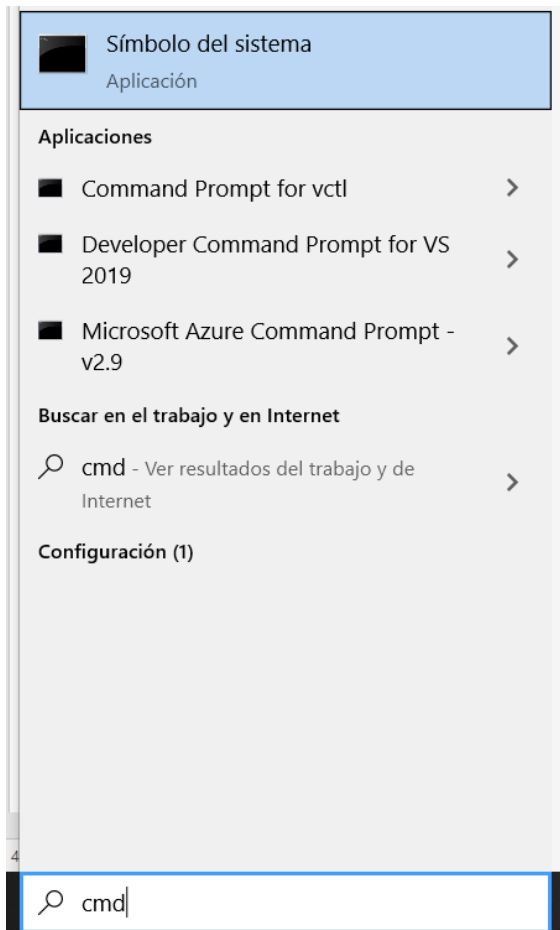
```
public class MiProgramaJava{  
  
    public static void main(String[] args) {  
  
        System.out.println("Hello World, Howdy");  
    }  
}
```

```
}
```

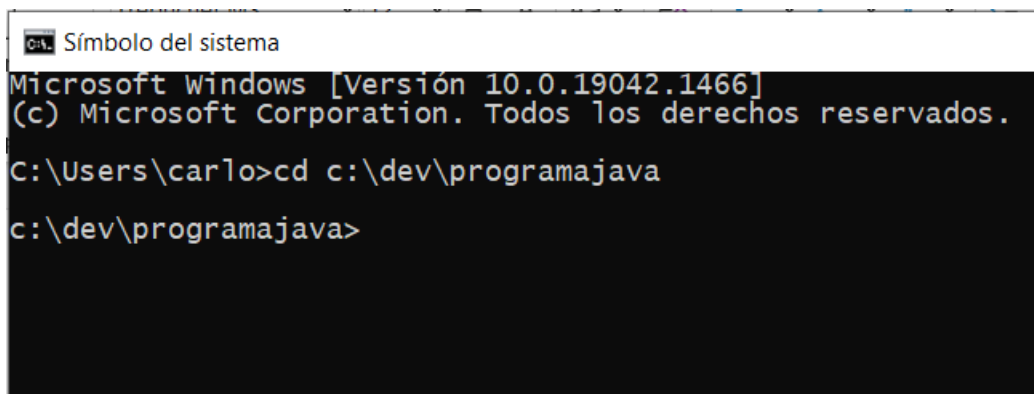
 MiProgramaJava.java: Bloc de notas  
Archivo Edición Formato Ver Ayuda

```
public class MiProgramaJava{  
  
    public static void main(String[] args) {  
  
        System.out.println("Hello World, Howdy");  
  
    }  
  
}
```

3. Después, debe abrir la consola de Windows. Para ello, escriba cmd en el área de búsqueda de Windows:



4. Abra la consola y escriba `cd c:\dev\programajava` y pulsa enter



## Compilando

1. El JDK que instalaste previamente, ofrece a los desarrolladores herramientas para compilar el programa Java y ejecutarlos. La primera herramienta que se utiliza para compilar es **javac**, **el compilador Java**. Por lo tanto, escriba `javac MiProgramaJava.java`. Como resultado, se ha generado un archivo java de compilación del programa, `MiProgramaJava.class`. Los expertos se refieren a estos archivos como archivos binarios.

```
c:\dev\programajava>javac MiProgramaJava.java

c:\dev\programajava>dir
El volumen de la unidad C es windows
El número de serie del volumen es: 5071-71CF

Directorio de c:\dev\programajava

21/01/2022  13:24    <DIR>          .
21/01/2022  13:24    <DIR>          ..
21/01/2022  13:24                440 MiProgramaJava.class
21/01/2022  12:01                136 MiProgramaJava.java
                2 archivos                576 bytes
                2 dirs 61.358.755.840 bytes libres

c:\dev\programajava>_
```

## Ejecutar

1. El JDK lo utiliza como herramienta para ejecutar programas java en windows, java.exe. Esta herramienta añade el código de bytes de su programa a la JVM. La JVM interpreta el código de bytes, lo transforma en un lenguaje de máquina compatible con el sistema y crea un nuevo proceso con este código. Intenta escribir en la consola `java -cp c:\dev\programajava MiProgramaJava`

```

C:\> Símbolo del sistema

Microsoft Windows [Versión 10.0.19042.1466]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\carlo>cd c:\dev\programajava

c:\dev\programajava>java -cp c:\dev\programajava MiProgramaJava
Hello World, Howdy

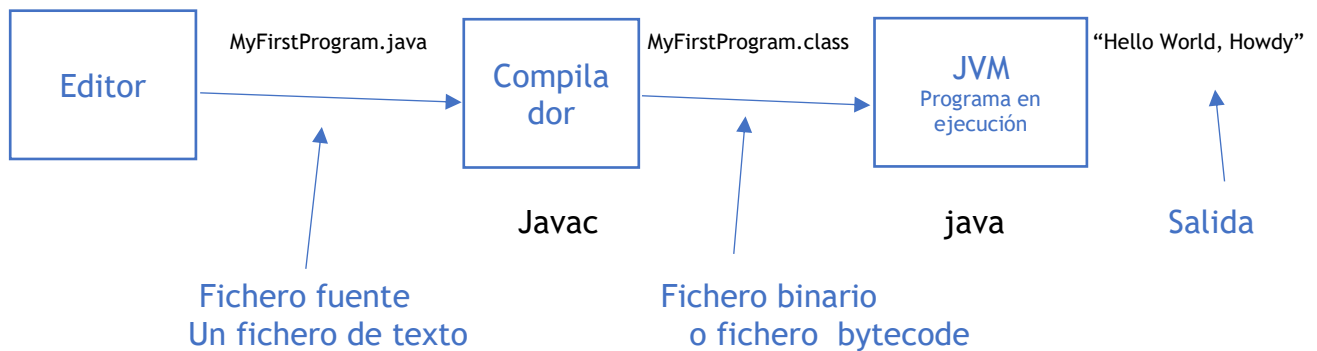
c:\dev\programajava>_
```

Incluyendo la opción `-cp C:\dev\programajava`, se proporciona a la JVM con la ruta de clases, el Classpath. La vía de acceso a las clases `classpath` una variable especial de los programas Java que señala la vía de acceso de las clases de programa. Java necesita saber dónde están sus clases para ejecutarlas. Después de la versión java 8 si no se configura, java.exe puede no funcionará.

Hasta ahora, has aprendido cómo crear y compilar tus propios programas Java. Para completar esta tarea, debes utilizar un Editor, un compilador y la consola. Este procedimiento es lo que los primeros desarrolladores solían hacer en los años 60 y 70 para

desarrollar. Hoy en día, los IDE, entornos de desarrollo integrados, ayudan a los programadores a tener éxito. Los IDE ofrecen editores de código, compiladores como los que usas antes para crear tu primer programa java. Además, proporcionan innumerables herramientas para gestionar, automatizar, probar, controlar versiones, documentar, modelar, etc. el proceso de desarrollo de software.

Para concluir el proceso que ha tomado para ejecutar el programa desde su código fuente donde:



### 3.3 IDE's

Un **entorno de desarrollo integrado (IDE)** es una aplicación de software que **proporciona instalaciones integrales a los programadores de computadoras para el desarrollo de software**. Un IDE normalmente consta de al menos un editor de código fuente, herramientas de automatización de compilación y un depurador. Algunos IDE, como NetBeans y Eclipse, contienen el compilador, intérprete o ambos necesarios; otros, como SharpDevelop y Lazarus, no lo hacen.

El límite entre un IDE y otras partes del entorno de desarrollo de software más amplio no está bien definido; a veces se integra un **sistema de control de versiones o varias herramientas para simplificar la construcción de una interfaz gráfica de usuario (GUI)**. Muchos IDE modernos también tienen un explorador de clases, un examinador de objetos y un diagrama de jerarquía de clases para su uso en el desarrollo de software orientado a objetos.

### Componentes habituales de los IDE

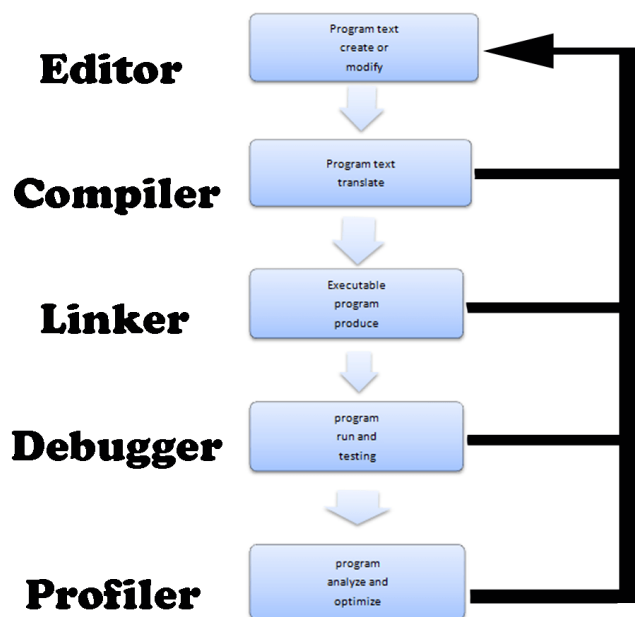
- **Editor de texto**: Esta es la parte donde puede escribir su código y ofrecer sus propias funciones de edición. Cambia los colores dependiendo de si son instrucciones, variables, palabras reservadas, etc.
- **Compilador o Compiler**: Se encarga de traducir el código fuente escrito por el



programador de alto nivel, a un programa escrito a un bajo nivel llamado lenguaje máquina.

- **Interprete o interpret**: Realice la traducción a medida que se ejecuta el programa de instrucción a instrucción. Son más lentos que los compiladores porque deben realizar la traducción sobre la ejecución.
- **Linker**: un programa informático que toma uno o más archivos objeto generados por un compilador y los combina en un programa ejecutable.
- **Debugger**: Es responsable de depurar y limpiar los errores de código fuente de un programa informático. Permite recorrer , instrucción a instrucción e inspeccionar las diferentes situaciones y cambios en las variables.
- **Constructor de Interfaz gráfica (Graphical interface builder GUI)**: es una herramienta que simplifica la creación de interfaces gráficas de usuario permitiendo al diseñador colocar controles con arrastrar y soltar.
- **Control de versiones**: Esta es una herramienta que le permite controlar los cambios que se realizan en las aplicaciones. Los parches y versiones se generan en cada momento del desarrollo.
- **Profiler**: una herramienta para optimizar, formatear y mejorar tu código.

Estos componentes trabajan juntos para producir el resultado final, un buen software.



### 3.4 Eclipse IDE

Eclipse es un entorno de desarrollo integrado (IDE) utilizado en programación informática. Contiene un espacio de trabajo base y un sistema de plug-in extensible para personalizar el entorno. Eclipse está escrito principalmente en Java y su uso principal es para el desarrollo de aplicaciones Java, pero también se puede usar para desarrollar aplicaciones en otros lenguajes de programación a través de plug-ins, incluyendo Ada, ABAP, C, C++, C#, Clojure, COBOL, D, Erlang, Fortran, Groovy, Haskell, JavaScript, Julia,[6] Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby (incluyendo ruby on rails framework), Rust, Scala y Scheme.

También se puede utilizar para desarrollar documentos con LaTeX (a través de un plug-in TeXlipse) y paquetes para el software Mathematica. Los entornos de desarrollo incluyen las herramientas de desarrollo Java de Eclipse (JDT) para Java y Scala, Eclipse CDT para C/C++ y Eclipse PDT para PHP, entre otros.

### 3.4.1 Instalación de Eclipse

El primer requisito para instalar Eclipse Ide es la instalación de JDK. En este punto, debes haber completado esta tarea. Independientemente de su sistema operativo, deberá instalar alguna máquina virtual Java (JVM). Puedes instalar el Java Runtime Environment (JRE) o un Java Development Kit (JDK), dependiendo de lo que desee hacer con Eclipse. En este curso, finalmente instalará el JDK.

Nuestro objetivo es instalar la versión estable de Eclipse debido a dos razones:

1. Puede mantener diferentes versiones de Eclipse ejecutándose en su computadora.
2. No es necesario ejecutar una versión instalable que establezca el entorno para usted. Vale la pena configurar el entorno Java tu mismo como lo hizo cuando instaló el JDK.

Se puede instalar eclipse estable versión 18, Eclipse 20-12, proporcionado en el sitio web de Eclipse y en la clase Moddle. Las versiones estables (también llamadas lanzamientos finales) se producen cada nueve meses y están destinadas al uso normal y cotidiano. Las versiones estables han recibido extensas pruebas de la comunidad antes del lanzamiento. Por lo general, salen en un archivo zip. Aquí está el enlace:

[https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2020-12/R/eclipse-jee-2020-12-R-win32-x86\\_64.zip](https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2020-12/R/eclipse-jee-2020-12-R-win32-x86_64.zip)

Contenido descargado y descomprimido en c:\dev\

mpartir Vista

> Este equipo > Windows (C:) > dev >

	Nombre	Fecha de modificación	Tipo	Tamaño
	apache-ant-1.10.9	08/03/2021 15:44	Carpeta de archivos	
	apache-maven-3.6.0	11/02/2021 19:09	Carpeta de archivos	
	apache-maven-3.6.2	08/02/2021 16:16	Carpeta de archivos	
	apache-maven-3.6.3	15/02/2021 1:04	Carpeta de archivos	
Inicio	eclipse-jee-2020-09-R-win32-x86_64	15/01/2021 14:00	Carpeta de archivos	
	eclipse-jee-2020-12-R-win32-x86_64 (1)	15/01/2021 13:01	Carpeta de archivos	

Para ejecutar la aplicación, vaya al directorio bin, que contiene archivos binarios de Eclipse y haga clic en eclipse.exe. **Espera a leer primero el siguiente capítulo:**

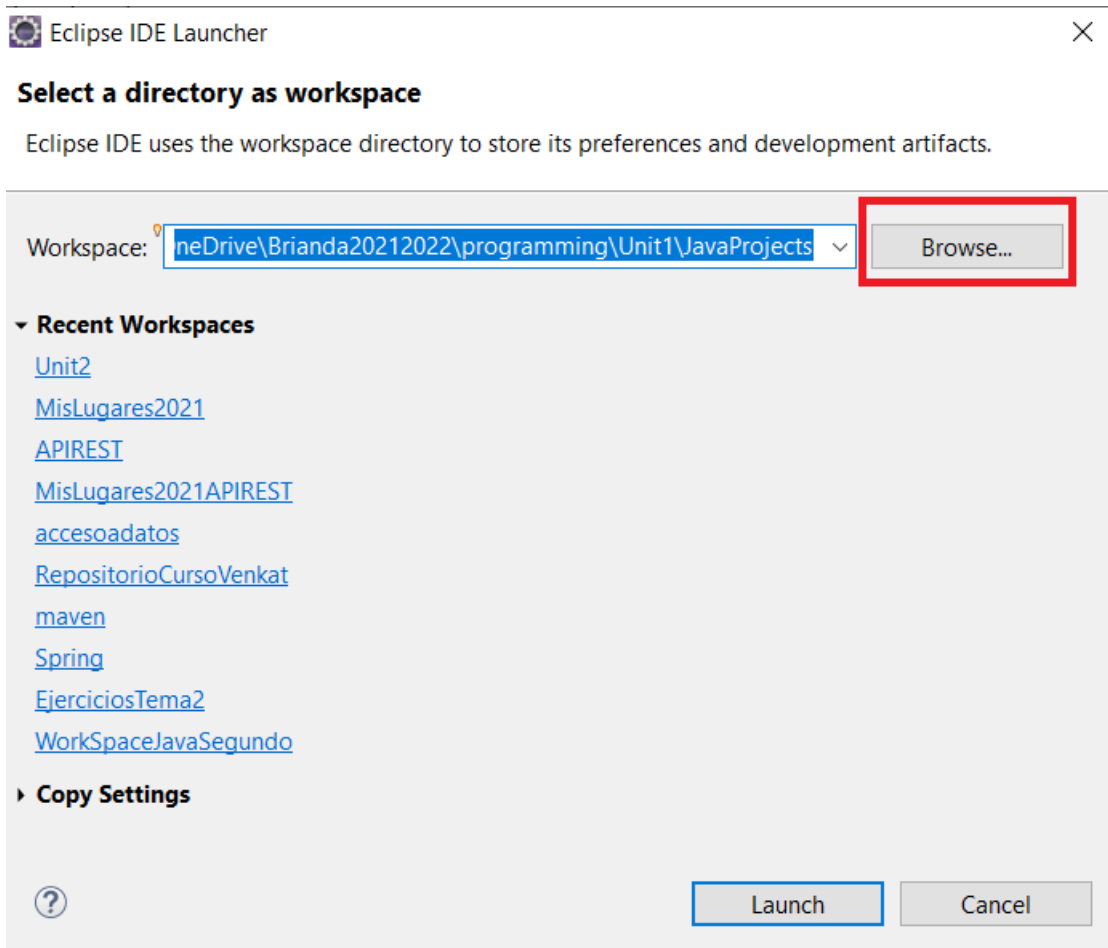
### 3.4.2 Nuevo espacio de trabajo o workspace en eclipse IDE

Lo primero que debes hacer es organizar tus proyectos. **Para seguir el curso, se recomienda crear una carpeta para la unidad de programación.** Dentro de esta carpeta debe incluir una carpeta denominada proyectos. Debe crear en la lista un escritorio de WorkSpaces para cada unidad a lo largo del año. Para esta unidad, podemos añadir también otra subcarpeta en proyectos llamada \WorkSpaceUnidad3. El resultado debe ser similar a:

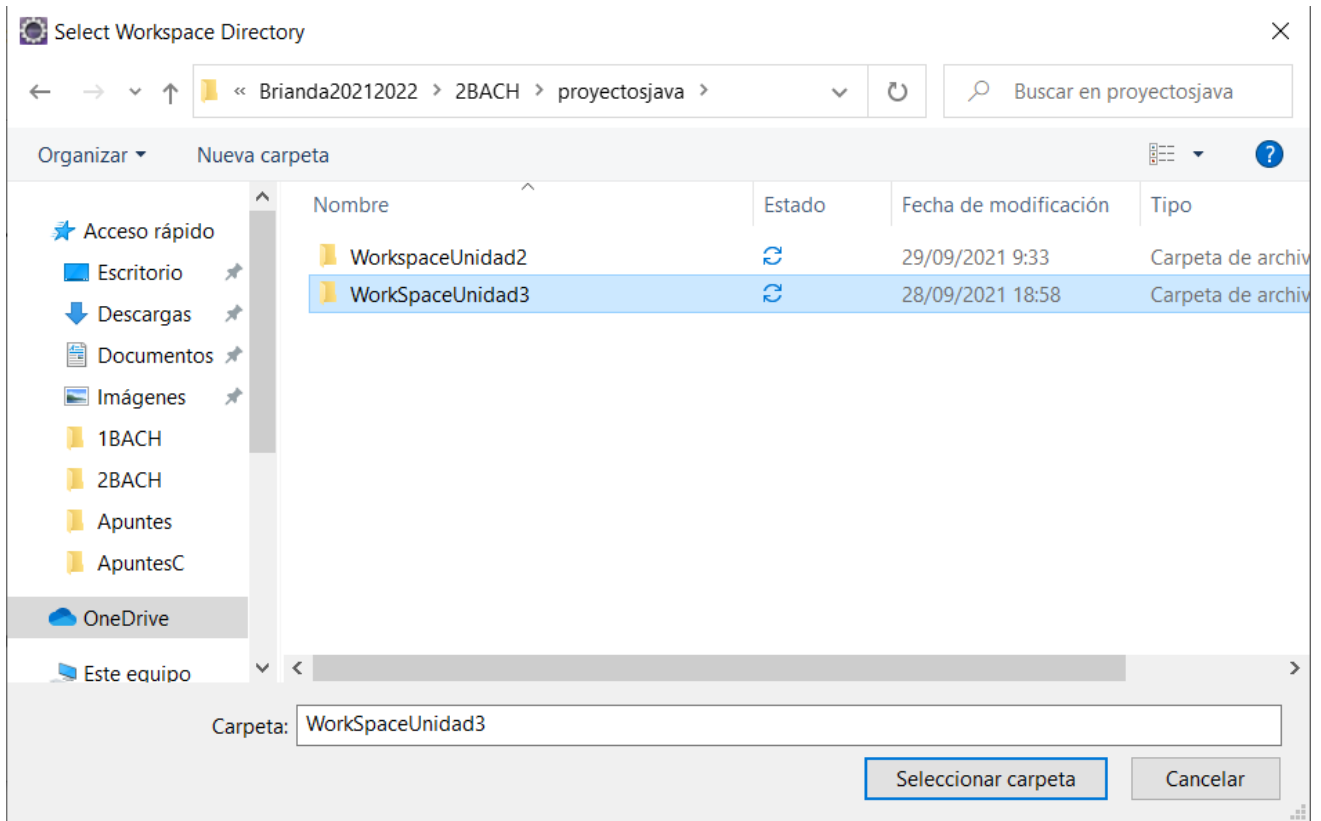
> OneDrive > Brianda20212022 > 2BACH > proyectosjava > WorkSpaceUnidad3 >

	Nombre	Estado	Fecha de modificación
	.metadata	🔄	28/09/2021 18:57
	ProyectoUnidad3	✅	28/09/2021 18:57
	ProyectoUnidad3Espanol	✅	28/09/2021 18:58

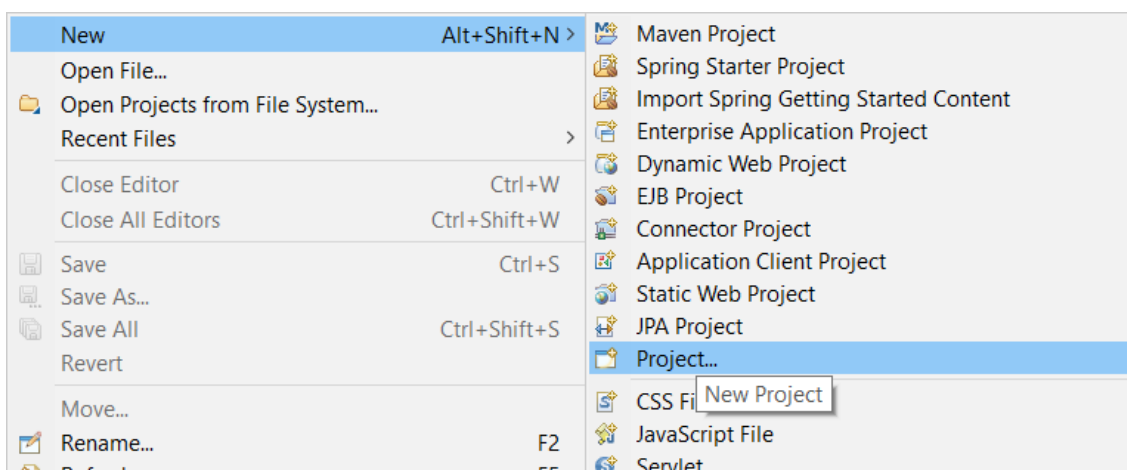
1. Después, abra su IDE de Eclipse y haga clic en examinar.



2. Seleccione la carpeta WorkspaceUnidad3 y pulse el botón Iniciar



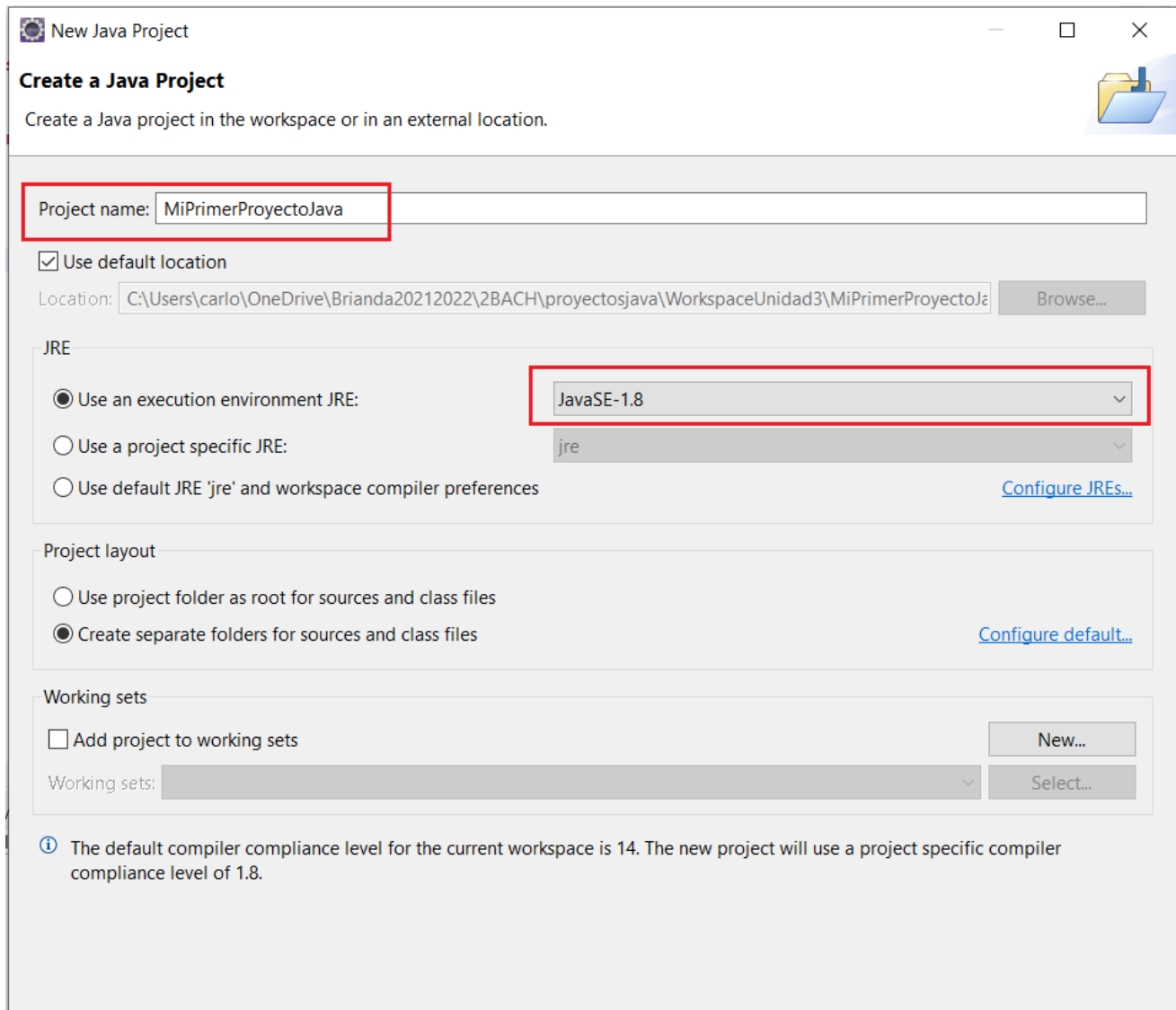
3. A continuación, cierre la ventana de bienvenida. Como resultado, has configurado un espacio de trabajo, un área de trabajo donde los desarrolladores pueden agregar proyectos Java. A continuación, debe crear un nuevo proyecto Java. Para ello, haga clic en el menú Archivo >nuevo proyecto -> proyecto java



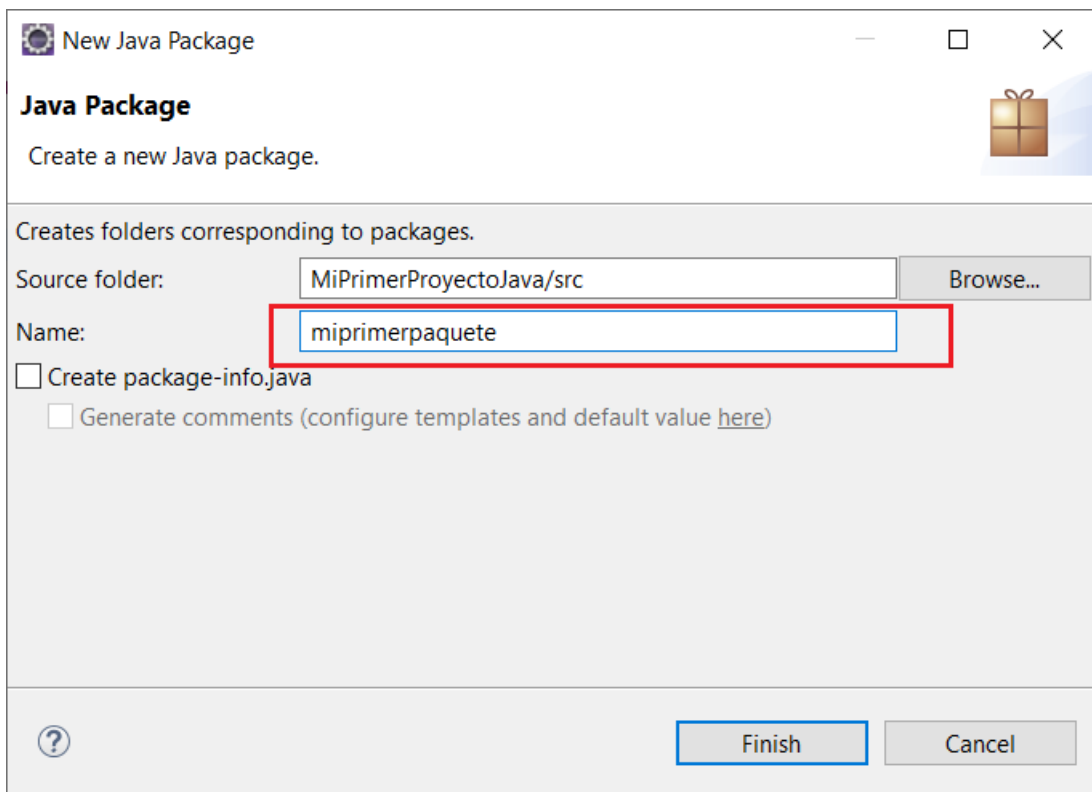
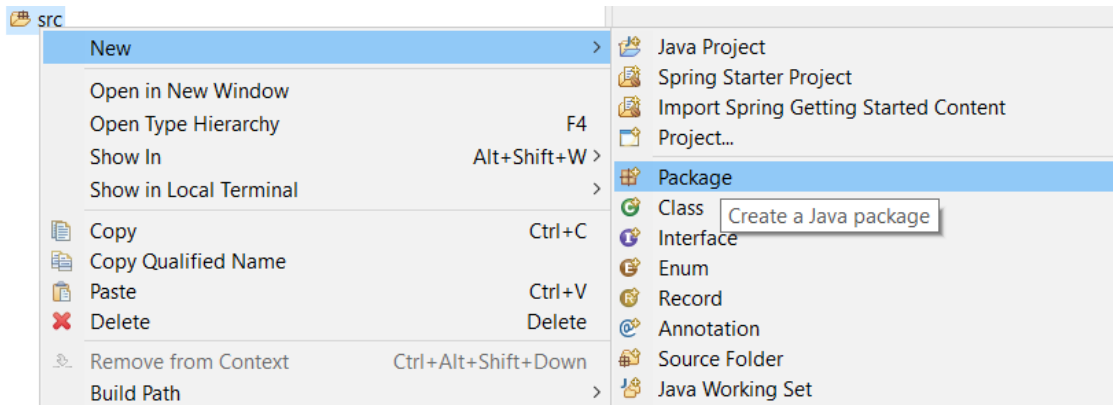
4. Escriba en el cuadro de texto del asistente Proyecto Java

5. Seleccione la opción Proyecto Java. Haga clic en siguiente. En la siguiente ventana debe rellenar tres Opciones. En primer lugar, el nombre del proyecto,

MiPrimerProyectoJava. En segundo lugar, debe elegir entre las diferentes versiones de JDK. Sólo para empezar, se sugiere elegir JAVA-SE-1.8. Y finalmente, es más apropiado separar la carpeta de archivos de origen java y compilar la carpeta de archivos de clase. Haga clic en finalizar “Finish” y el trabajo está hecho. Tu primer proyecto java. No olvides abrir la perspectiva del proyecto.



6. A la salida de todos estos pasos es un proyecto Java vacío. La estructura del proyecto y las carpetas está construida, pero es necesario agregar el ingrediente principal, que es el código fuente Java, una clase java. Pero primero, agregaremos un paquete. Los paquetes son casi obligatorios desde Java 8. Seleccione la carpeta src, haga clic en el botón derecho y aparecerá el menú. Seleccionar nuevo paquete (new> package).



7. Lo nombramos como miprimerpaquete. En las siguientes lecciones se introducirá la convención de nombres. A partir de ahora, utilizará letras minúsculas para los paquetes. A continuación, seleccione el paquete haga clic en el botón derecho del ratón de nuevo, new->Class y va a crear una nueva clase MiPrimeraClase . En Java, los desarrolladores capitalizan cada primera letra de nombres compuestos para las clases. También añade el método principal. Este es el punto de entrada a tu programa.

**New Java Class**

**Java Class**  
Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☒ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

8. El producto final es un nuevo programa Java que puede modificar, compilar, compilar, compilar y ejecutar en su eclipse o desde la consola.

## Haz que tu Código fluya.

1. Ahora puede escribir su primera línea de código. Agregue la siguiente línea en amarillo a la función principal

```
package miprimerpaquete;  
  
public class MiPrimeraClase {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        System.out.println("Mi primer programa. Mi primera clase Java");  
    }  
}
```

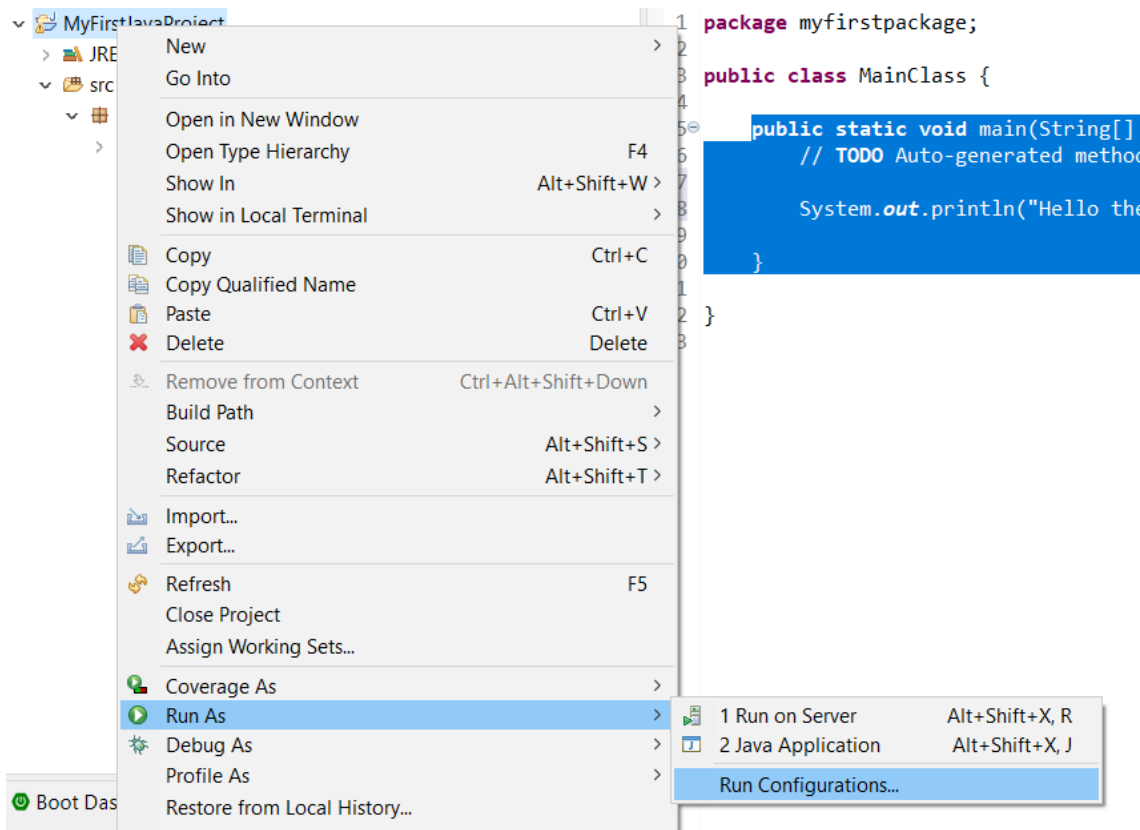


```

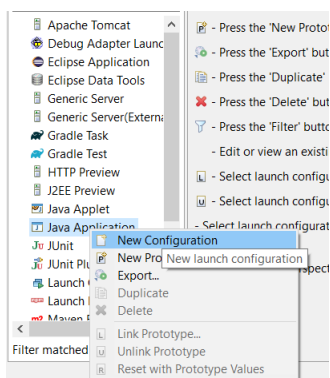
    }
}

```

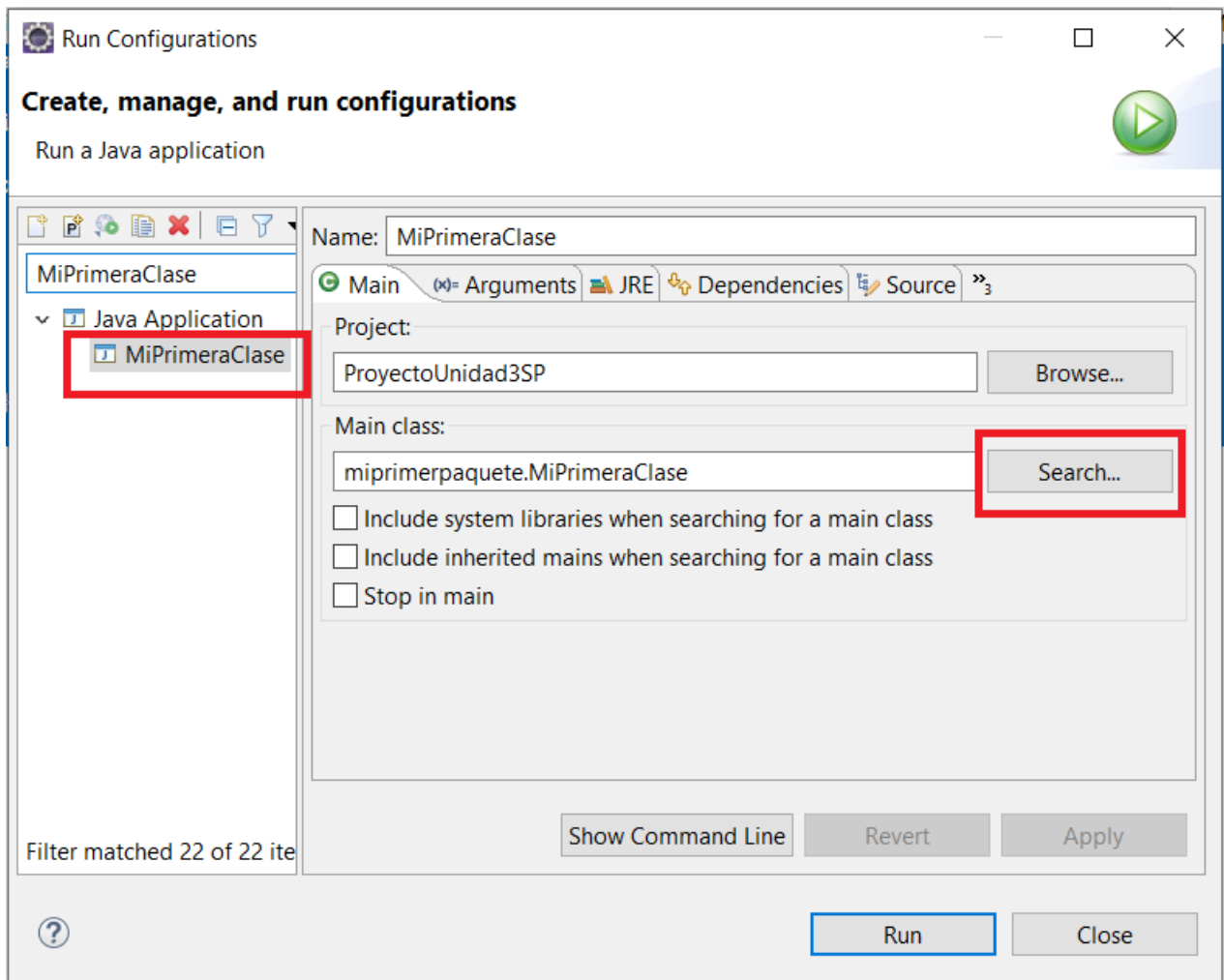
2. Println es una función o método de la librería java.lang que nos permite imprimir texto en la consola. Debes configurar el proyecto para que se ejecute. Seleccione su proyecto, haga clic con el botón derecho del mouse y ejecute las configuraciones de > ejecutar



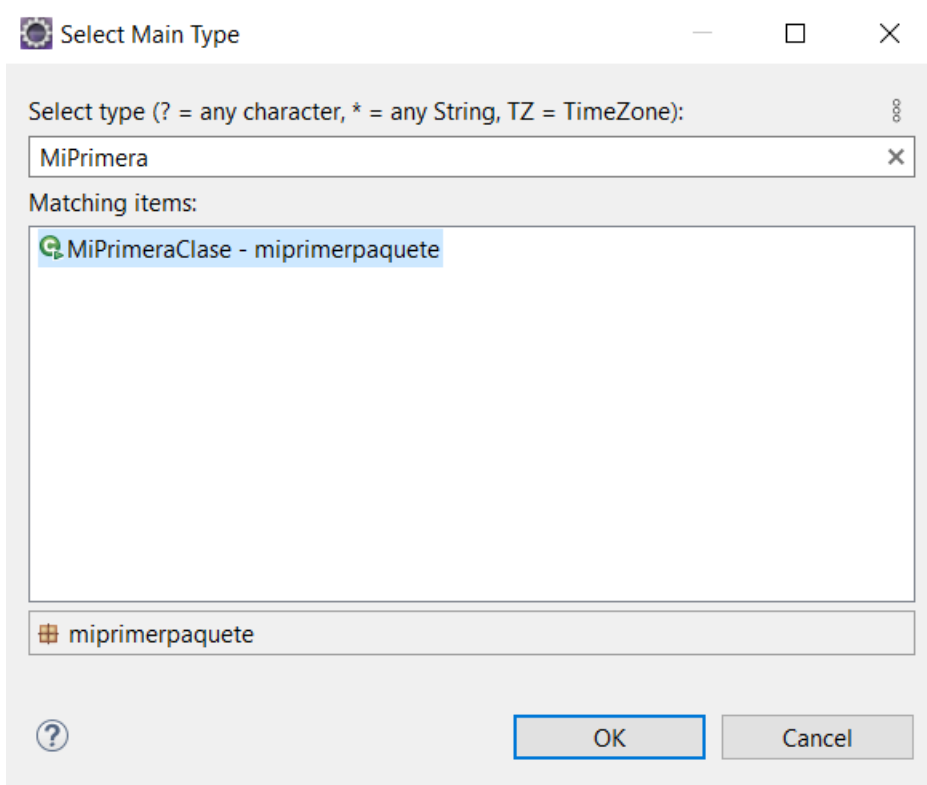
3. Botón derecho en Java Application-> nueva configuración



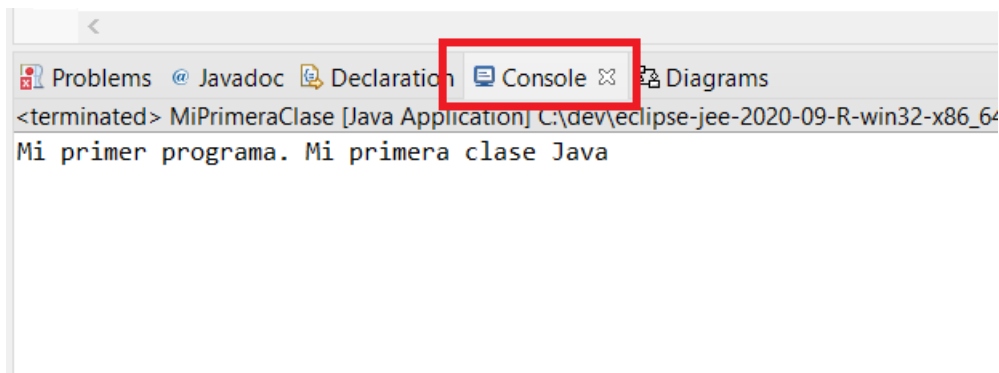
4. Haga clic en el cuadro De texto de búsqueda a la derecha de la etiqueta de la clase principal,



5. Seleccione la clase MainClass de tu propio proyecto y presione Ok.



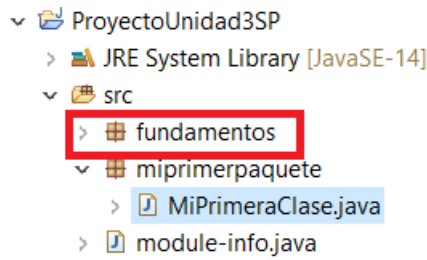
6. Pulse Aplicar y Cerrar. Y de nuevo seleccione su proyecto y haga clic en el botón derecho del ratón, y ejecutar -> como aplicación Java. El resultado en el área de la consola.



7. Además, se puede ejecutar clase por clase en el proyecto. La única condición que la clase debe cumplir es tener un método principal main. Lo único que debe hacer es seleccionar tu clase, el botón derecho del ratón y Run as > Java Application. El resultado debe ser igual al del anterior ejemplo.

## 4 Fundamentos de Java

Vamos a crear el paquete fundamentos en el mismo proyecto.



### 4.1 Terminología

Para hablar de los fundamentos de Java, necesitamos introducir algo de terminología. Para ello, comenzamos con el siguiente fragmento de código:

```
int a, b, c;  
a = 1234;  
b = 99;  
c = a + b;
```

La primera línea es una instrucción de declaración que declara los nombres de tres variables utilizando los identificadores a, b y c y su tipo para que sea int.

Las tres líneas siguientes son instrucciones de asignación que cambian los valores de las variables, utilizando los literales 1234 y 99, y

#### Literales

Un literal es una representación de código Java de un valor de tipo de datos. Usamos secuencias de dígitos como 1234 o 99 para representar valores de tipo int; sumamos un separador decimal, como en 3.14159 o 2.71828, para representar valores de tipo double; usamos las palabras clave true o false para representar los dos valores de tipo boolean; y usamos secuencias de caracteres entre comillas, como "Hello, World", para representar valores de tipo String.

#### Operadores

Un operador es una representación de código Java de una operación de tipo de datos. Java utiliza + y \* para representar la suma y la multiplicación de enteros y números de punto flotante; Java utiliza &&, |, y ! para representar operaciones booleanas; y así sucesivamente. Describiremos los operadores más utilizados en los tipos integrados de java, conocidos como tipos primitivos más adelante en esta sección.

## 4.2 Identificadores

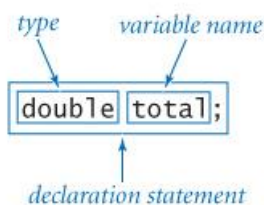
Un **identificador** es una representación de código Java de un nombre (por ejemplo, para una variable). Cada **identificador** es una secuencia de letras, **dígitos**, **caracteres de subrayado** y **símbolos de dolar**, el primero de los cuales no es un dígito. Por ejemplo, las secuencias de caracteres `abc`, `Ab $`, `abc123` y `a_b` son todos **identificadores java validos**, pero `Ab*`, `1abc` y `a + b` no lo son. Los **identificadores distinguen entre mayúsculas y minúsculas**, por lo que `Ab`, `ab` y `AB` son nombres diferentes. Algunas palabras reservadas, como `public`, `static`, `int`, `double`, `String`, `true`, `false` y `null`, son especiales y **no se pueden utilizar como identificadores**.

## Variables

Una **variable** es una entidad que contiene un valor de tipo de datos, al que podemos hacer referencia por su nombre. En Java, cada variable tiene un tipo específico y almacena uno de los valores posibles de ese tipo. Por ejemplo, **una variable int puede almacenar el valor 99 o 1234 pero no 3.14159 o "Hello, World"**. Diferentes variables del mismo tipo pueden almacenar el mismo valor. Además, como su nombre indica, **el valor de una variable puede cambiar a medida que se desarrolla un cálculo**. Por ejemplo, usamos una variable llamada `sum` en varios programas de este libro para mantener la suma corriente de una secuencia de números. Creamos variables mediante instrucciones de declaración y calculamos con ellas en expresiones, como se describe a continuación.

## Instrucciones de declaracion

Para crear una variable en Java, utilice una sentencia **de declaración**, o simplemente **una declaración** para short. Una declaración incluye un tipo seguido de un nombre de variable. Java reserva suficiente memoria para almacenar un valor de tipo de datos del tipo especificado y **asocia el nombre de la variable a esa área de memoria**, de modo que pueda acceder al valor cuando utilice la variable en código posterior. Para ser más concisos, **se pueden declarar varias variables del mismo tipo en una sola instrucción** de declaración.



*Anatomy of a declaration*

El texto `"double total;"` se etiqueta como **"de declaración"**. El texto `"double"` está etiquetado como **"tipo"** y el texto `"total"`.

## Variables constantes

Utilizamos el término oximorónico **variable constante** para **describir una variable cuyo valor no cambia durante la ejecución de un programa** (o de una ejecución del programa a la siguiente). En este tema, **nuestra convención es dar a cada variable constante un nombre que consta de una letra mayúscula seguida de letras mayúsculas**, dígitos y guiones bajos. Por ejemplo, podríamos usar los nombres de variables constantes **SPEED\_OF\_LIGHT** y **DARK\_RED**.

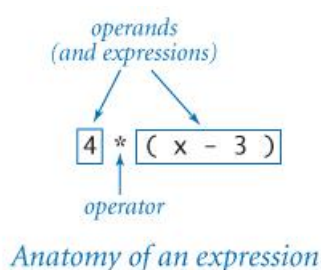
En java usamos **el modificador final para construir variables de tipo constante**. Para la convención de nombres, los programadores están acostumbrados a usar letras mayúsculas y subrayado "\_" para separar palabras en constantes de varias palabras dado que proporcionaron una manera de distinguir las constantes de variables de las variables regulares.

```
final int MAX_TEMP = 120;
```

## Expresiones

Una **expresión Java** es una **combinación de literales, variables y operaciones que Java evalúa para producir un valor**. Para los tipos primitivos, las expresiones a menudo se parecen a las **fórmulas matemáticas**, utilizando **operadores para especificar las operaciones de tipo de datos** que se realizarán en un operando más. La mayoría de los operadores que usamos son operadores binarios que toman exactamente dos operandos, como  $x - 3$  o  $5 * x$ .

Cada **operando** puede ser cualquier expresión, quizás entre paréntesis. Por ejemplo, podemos escribir  $4 * (x - 3)$  o  $5 * x - 6$  y Java entenderá lo que queremos decir. Una expresión es una instrucción para realizar una secuencia de operaciones; la expresión es una representación del valor resultante.



Podemos distinguir entre expresiones simples, como  $x-3$ , y expresiones complejas. Las expresiones complejas se componen de expresiones simples, operandos y expresiones entre paréntesis. Usamos paréntesis para alterar el **orden de precedencia**. Por ejemplo, en Java, el producto se evalúa antes de la adición y la resta. Siempre que  $x = 2$ ,  $4*x-3$  se evalúa siguiendo este orden:

1.  $4*2-3 = 8-3$
2.  $8-3 = 5$

A veces es necesario cambiar este orden de precedencia. Para ello, usamos la expresión entre paréntesis como  $4*(x-3)$ . Con respecto a los paréntesis, el **compilador Java examinará**

primero cuál está dentro del paréntesis. Como resultado, esta expresión se evaluaría en este orden.

1.  $4*(2-3) = 4*-1$
2.  $4*-1 = -4$

### Expresión entre paréntesis anidada

¿Qué pasa si tenemos expresión entre paréntesis dentro de expresiones entre paréntesis? Si has estudiado matemáticas, ya sabes la respuesta. Porque el compilador java, como casi cualquier compilador de lenguaje de programación sigue el orden matemático de precedencia, teniendo en cuenta que las expresiones entre paréntesis internas se resuelven primero.

Vamos a repasar esta compleja expresión  $x * (x / (x + 4))$ :

Suponiendo que  $x=4$ . Esta debería ser la evaluación natural en java:

En primer lugar, la expresión entre paréntesis más interna  $(x+4)$   
 $4*(4/4+4) = (4*(4/8))$

Segundo el otro parentesis interno  $(x/(x+4))$

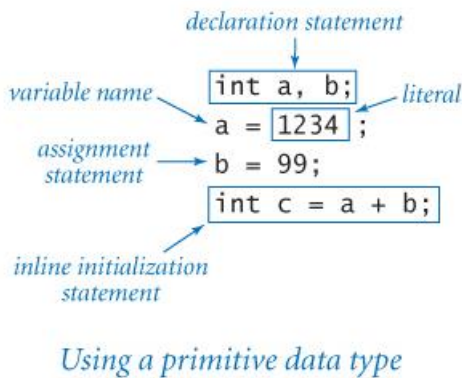
$4*(4/8) = 4*(0,5)$

Finalmente el product exterior  $x * (x / (x + 4))$ :

$4*(0,5) = 2$

### Instrucciones de asignación

Una instrucción de asignación asocia un valor de tipo de datos a una variable. Cuando escribimos  $c = a + b$  en Java, no estamos expresando igualdad matemática, sino que estamos expresando una acción en su lugar: establecemos el valor de la variable  $c$  para que sea el valor de  $a$  más el valor de  $b$ . Es cierto que el valor de  $c$  es matemáticamente igual al valor de  $a + b$  inmediatamente después de que se haya ejecutado la instrucción de asignación, pero el objetivo de la instrucción es cambiar (o inicializar) el valor de  $c$ . El lado izquierdo de una instrucción de asignación debe ser una sola variable; el lado derecho puede ser cualquier expresión que genere un valor de un tipo compatible. Así, por ejemplo, tanto  $1234 = a$ ; y  $a + b = b + a$ ; son sentencias no válidas en Java. En resumen, el significado de  $=$  decididamente no es el mismo que en las ecuaciones matemáticas.



Un subconjunto de las instrucciones del programa era "int a, b; a = 1234 ; b = 99; int c = a + b;" La instrucción "int a, b;" tiene la etiqueta "Declaración de declaración". La letra 'a' en la instrucción "a = 1234 ;" se etiqueta como "Nombre de variable" y el número "1234" se conoce como "Literal". La instrucción "b = 99;" se denomina "Instrucción de asignación". La instrucción "int c = a + b;" se denomina "Instrucción de inicialización en línea".

## Inline initialization, declaración de variable más inicialización

Para poder utilizar una variable en una expresión, primero debe declarar la variable y asignarle un valor inicial. Si no se hace cualquiera de los dos, se produce un error en tiempo de compilación. Por razones de economía, se puede combinar una instrucción de declaración con una instrucción de asignación en una especie de oración conocida como instrucción de inicialización en línea. Por ejemplo, el código siguiente declara dos variables a y b y las inicializa en los valores 1234 y 99, respectivamente:

```
int a = 1234;
```

### 4.3 La sintaxis básica de Java

Vamos a revisar el ejemplo inicial para explicar brevemente la sintaxis Java. Predominantemente, en Java declaramos tres tipos de sentencias en un programa. Explicaremos más instrucciones de forma detallada en este curso, pero comenzaremos con este tres.

Aquí, deberías agregar el siguiente código a tu proyecto FundamentosDeJava.

```
package miprimerpaquete;

public class MiPrimeraClase {

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



```
// TODO Auto-generated method stub

System.out.println("Mi primer programa. Mi primera clase Java");

}

}
```

En java **declaramos clases**, plantillas para objetos:

```
public class MiPrimeraClase {
```

Usamos la palabra **class** como **palabra preservada**. Si la **clase es la clase principal del archivo**, primero se debe agregar el modificador **public**. Como recordatorio, **el nombre del archivo java debe ser el mismo que el nombre de la clase pública**. Solo puede haber una clase pública por archivo. Por lo tanto, el nombre de archivo para esta clase debe ser MainClass.java.

Por otra **parte, declaramos métodos**, similares a las subrutinas. Tienen firma de método y cuerpo.

```
public static void main(String[] args) → SIGNATURA

{
    // TODO Auto-generated method stub

System.out.println("Mi primer programa. Mi primera clase Java");
}
```

La firma o cabecera de la función tiene el siguiente formato:

Modificadores --- Parámetros ---- method\_Name ---- de valores devueltos.  
**static**                      **void**                      main                      (String[] args)

1. **Modificadores**: añadir significado adicional a las estructuras java, tales como métodos o clases.
2. **Valor devuelto**: void si el método no devuelve nada. Un tipo de datos como int o String, en todos los demás casos.
3. **Function\_name**: usaremos para llamar al método.
4. **Parámetros**: valores recibidos por la función. Están separados por comas y se colocan entre paréntesis.

El cuerpo es un bloque de instrucciones o código. Comienza y termina con llaves {}.

```
{
    // TODO Auto-generated method stub
```

```
System.out.println("Mi primer programa. Mi primera clase Java");
```

Los desarrolladores escriben instrucciones o sentencias en Java. Hay dos tipos de instrucciones.

- 1) **Instrucción de una sola línea** o **simple**. Contienen llamadas a funciones, operadores y literales (datos sin formato).

```
int i = i+5;
```

o

```
System.out.println("Mi primer programa. Mi primera clase Java");
```

- 2) **Instrucción de bloque**. Una instrucción que contiene un bloque de código. Las llaves describen el principio y el final de un bloque de código. Observa que las instrucciones dentro de la instrucción de bloque tienen sangría. El IDE de Eclipse utiliza la tabulación para aplicar sangría a nuestro código. Hace que el código parezca más organizado y es más fácil identificar qué instrucciones están dentro de una instrucción de bloque.

```
while (i<100) {  
    i = i+5;  
    cont = cont+1;  
}
```

## 4.4 Comentarios de Java

Los comentarios se pueden utilizar para explicar el código Java y para hacerlo más legible. También se puede utilizar para evitar la ejecución al probar código alternativo.

Los comentarios de una sola línea comienzan con dos barras diagonales (//).

Cualquier texto después de la barra doble "//" y el final de la línea es ignorado por Java (no se ejecutará).

En este ejemplo se utiliza un comentario de una sola línea antes de una línea de código:

```
// esto es un comentario  
System.out.println("Hello World");
```

## Comentarios multilínea de Java

Los comentarios multilínea comienzan con /\* y terminan con \*/.

Cualquier texto entre /\* y \*/ será ignorado por Java.

En este ejemplo se utiliza un comentario de varias líneas (un bloque de comentarios) para explicar el código:  
ejemplo

```
/* El siguiente Código escribe en la consola  
Hello World */  
System.out.println("Hello World");
```

## 4.5 Tipos de dato java

Como se explicó para PSeint, una variable en Java debe tener un tipo de datos específico.

Los tipos de datos se dividen en dos grupos:

1. Tipos de datos primitivos: incluye `byte`, `short`, `int`, `long`, `float`, `double`, `boolean` y `char`. Representar los datos básicos que podemos almacenar en variables. Números, 9, 7,6, caracteres como 'f' o '?'.
1. Tipos de datos no primitivos, como Cadenas, matrices y clases (obtendrá más información sobre ellos más adelante en el capítulo)

### Example

```
int myNum = 5;           // Integer (numero entero)  
float myFloatNum = 5.99f; // numero decimal  
char myLetter = 'D';     // Caracter  
boolean myBool = true;   // Booleano  
String myText = "Hello"; // String
```

1. **byte:** El tipo de datos `byte` es un entero complementario de dos con signo de 8 bits. Tiene un valor mínimo de -128 y un valor máximo de 127 (ambos inclusive). El tipo de datos `byte` puede ser útil para ahorrar memoria en matrices grandes, donde el ahorro de memoria realmente importa. También se pueden utilizar en lugar de `int`, donde sus límites ayudan a aclarar el código; el hecho de que el intervalo de una variable sea limitado puede servir como una forma de documentación.'

2. **short:** El tipo de datos `short` es un entero de complemento de dos con signo de 16

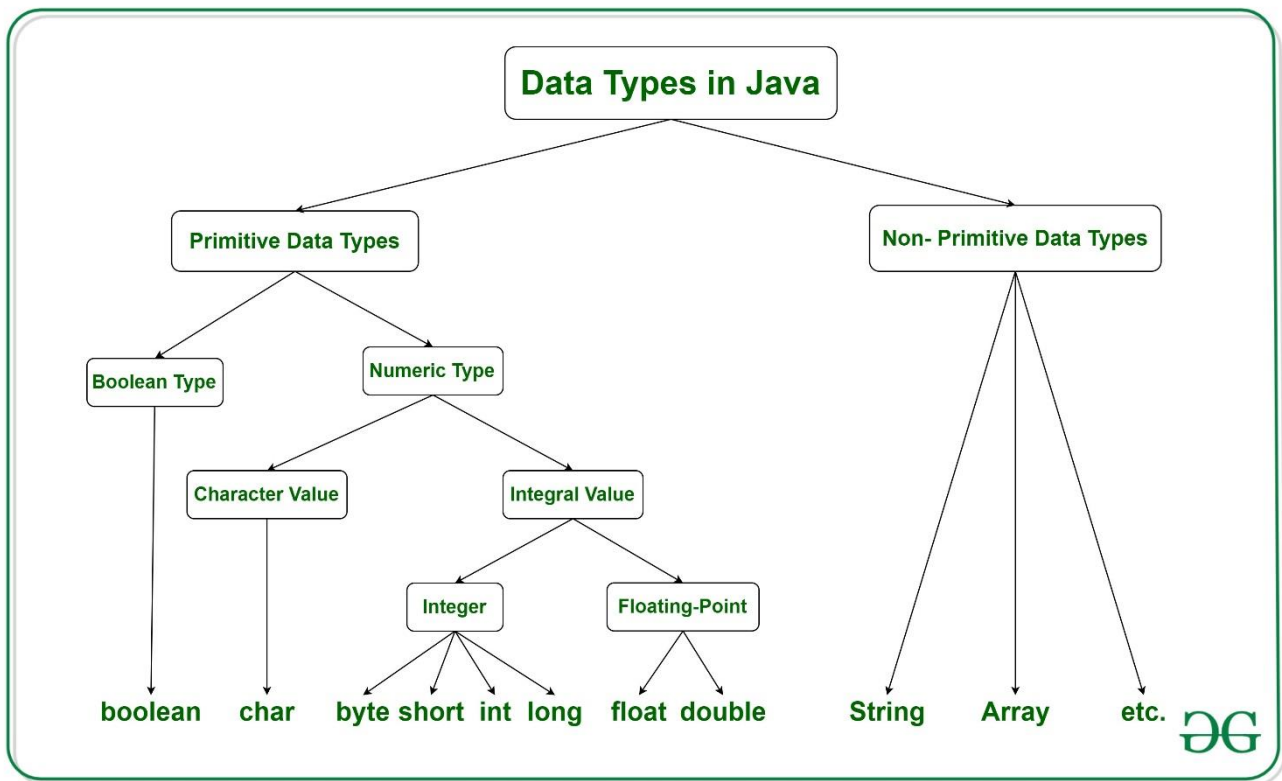
bits. Tiene un valor mínimo de -32.768 y un valor máximo de 32.767 (ambos inclusive). Al igual que con `byte`, se aplican las mismas directrices: puede usar un `cortocircuito` para ahorrar memoria en matrices grandes, en situaciones donde el ahorro de memoria realmente importa.

3. **int:** De forma predeterminada, el tipo de datos `int` es un entero complementario de dos con signo de 32 bits, que tiene un valor mínimo de  $-2^{31}$  y un valor máximo de  $2^{31}-1$ . En Java SE 8 y versiones posteriores, puede utilizar el tipo de datos `int` para representar un entero de 32 bits sin signo, que tiene un valor mínimo de 0 y un valor máximo de  $2^{32}-1$ . Utilice la clase `Integer` para utilizar el tipo de datos `int` como un entero sin signo. Consulte la sección Las clases numéricas para obtener más información. Métodos estáticos como `compareUnsigned`, `divideUnsigned`, etc. se han agregado a la clase `Integer` para admitir las operaciones aritméticas de enteros sin signo.
4. **long:** El tipo de datos `long` es un entero complementario de dos de 64 bits. El `long` con signo tiene un valor mínimo de  $-2^{63}$  y un valor máximo de  $2^{63}-1$ . En Java SE 8 y versiones posteriores, puede utilizar el tipo de datos `long` para representar una longitud de 64 bits sin signo, que tiene un valor mínimo de 0 y un valor máximo de  $2^{64}-1$ . Utilice este tipo de datos cuando necesite un intervalo de valores más amplio que los proporcionados por `int`. La clase `Long` también contiene métodos como `compareUnsigned`, `divideUnsigned`, etc. para admitir operaciones aritméticas para `long` sin signo.
5. **float:** El tipo de datos `float` es un punto flotante IEEE 754 de 32 bits de precisión simple. Su rango de valores está fuera del alcance de esta explicación, pero se especifica en la sección Tipos, formatos y valores de punto flotante de Java Language Specification. Al igual que con las recomendaciones para `byte` y `short`, utilice un `float` (en lugar de `double`) si necesita guardar memoria en matrices grandes de números de punto flotante. Este tipo de datos nunca se debe utilizar para valores precisos, como la moneda. Para eso, necesitará usar la clase `java.math.BigDecimal` en su lugar. `Numbers and Strings` cubre `BigDecimal` y otras clases útiles proporcionadas por la plataforma Java.
6. **double:** El tipo de datos `double` es un punto flotante IEEE 754 de 64 bits de precisión doble. Su rango de valores está fuera del alcance de esta explicación, pero se especifica en la sección [Tipos, formatos y valores](#) de punto flotante de Java Language Specification. Para los valores decimales, este tipo de datos suele ser la opción predeterminada. Como se mencionó anteriormente, este tipo de datos nunca se debe usar para valores precisos, como la moneda.
7. **boolean:** El tipo de datos `boolean` tiene sólo dos valores posibles: `true` y `false`. Utilice este tipo de datos para las marcas simples que realizan un seguimiento de las condiciones true/false. Este tipo de datos representa un bit de información, pero su "tamaño" no es algo que esté definido con precisión.
8. **char:** El tipo de datos `char` es un único carácter Unicode de 16 bits. Tiene un valor mínimo de `'\u0000'` (o 0) y un valor máximo de `'\uffff'` (o 65.535 inclusive).

**Ejercicio rellenar la tabla siguiendo los datos anteriores:**

Type	Data	Size	range
Byte	Numérico	8 bits	-128,127
short	Numérico	16 bits	
int	Numérico	32 bits	
long	Numérico	64 bits	
float	Numérico	32 bits	
double	Numérico	64 bits	
boolean	Lógico		
char	Alfanumérico	16 bits	

Java tiene **un tipo estático y un lenguaje fuertemente tipado** porque, en Java, cada tipo de datos (como entero, carácter, hexadecimal, decimal, etc.) está predefinido como parte del lenguaje de programación y todas las constantes o variables definidas para un programa determinado deben restringirse con uno de los tipos de datos.



#### 4.5.1 Conversión de tipos Java para tipos primitivos

Al asignar cualquier valor de tipo de datos a otra variable con tipo, es posible que los dos tipos no sean compatibles entre sí. Si los tipos de datos son compatibles, Java realizaría la conversión automáticamente conocida como **Conversión automática de tipos** y, si no, entonces deben ser casteados o convertidos explícitamente. Por ejemplo, asignar un valor `int` a una variable `long`.

#### Ampliación o conversión automática de tipos

La conversión de ampliación tiene lugar cuando dos tipos de datos se convierten automáticamente. Esto sucede cuando:

1. Los dos tipos de datos son compatibles.
2. Cuando asignamos el valor de un tipo de datos más pequeño a un tipo de datos más grande.

Por ejemplo, en java los tipos de datos numéricos son compatibles entre sí, pero no se admite ninguna conversión automática de tipo numérico a `char` o `boolean`. Además, `char` y `boolean` no son compatibles entre sí.

## Byte → Short → Int → Long → Float → Double

### Widening or Automatic Conversion

#### JavaTypeConversion.java

En el siguiente código, este programa declara variables de cualquier tipo numérico en java. Comienza inicialmente con el tipo más corto 8 bytes. A continuación, asignando a una variable de tipo más grande, convierte automáticamente el número "100" en tipos enteros o decimales. Puesto que byte es el tipo más pequeño, el código nunca pierde información en la conversión automática.

#### Ejemplo JavaConversionDeTipos.java

```
package fundamentos;

public class JavaConversionDeTipos {

    public static void main(String[] args)

    {

        byte b= 100;

        short s= b;

        int i = s;

        // conversión automática
        long l = i;

        // conversión automática
        float f = l;

        double d = b;

        System.out.println("byte "+b);
        System.out.println("short "+s);
        System.out.println("int "+i);
        System.out.println("long "+l);
        System.out.println("float "+f);
        System.out.println("double "+f);
    }

}
```

## Ejecución

```
byte 100
short 100
int 100
long 100
float 100.0
```

```
double 100.0
```

## Casteo o conversión explícita

Si queremos asignar un valor de tipo de datos más grande a un tipo de datos más pequeño, realizamos **la conversión o** casteo explícita de tipos.

Esto es útil para tipos de datos incompatibles donde no se puede realizar la conversión automática.

Aquí, target-type especifica el tipo deseado al que se va a convertir el valor especificado.

**Nota:** El carácter y el número no son compatibles entre sí. Veamos cuándo tratamos de convertir uno en otro

**Double → Float → Long → Int → Short → Byte**

### Narrowing or Explicit Conversion

Observe cómo tu IDE Eclipse muestra un error de compilación en el siguiente ejemplo

#### EjemploErrorConversionExplicita.java

```
package fundamentos;

public class EjemploErrorConversionExplicita {

    public static void main(String[] argv)
    {
        char ch = 'c';
        int num = 88;

        double numDec=44.5;
        ch = num;
        ch = numDec;

    }
}
```



```
}
```

Error:

7: error: incompatible types: possible lossy conversion from int to char

```
ch = num;
```

```
^
```

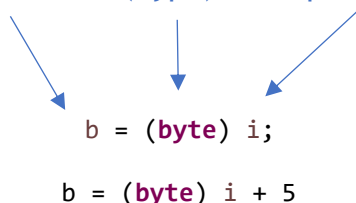
1 error

### ¿Cómo hacer la conversión explícita?

Para corregir este error, Java proporciona una herramienta para implementar la conversión explícita, que está etiquetada como "Casting". La sintaxis de conversión consiste en escribir el tipo entre paréntesis en el lado derecho de la instrucción de asignación, justo antes de la variable o expresión.

Casting

Variable = (Type) expression;



```
b = (byte) i;
```

```
b = (byte) i + 5
```

Mientras se asigna un valor a un tipo byte, la parte fraccionaria se pierde y se reduce a módulo 256 (intervalo de bytes).

ConversionTiposPerdida.java

```
package fundamentos;
```

```
public class ConversionTiposPerdida {
```

```
    public static void main(String args[])
    {
        byte b;
        int i = 257;
        double d = 323.142;
        System.out.println("Conversion de int a byte.");

        //i%256
        b = (byte) i;
        System.out.println("i = " + i + " b = " + b);
        System.out.println("\nConversion de double a byte.");

        //d%256
        b = (byte) d;
```

```

        System.out.println("\nDe double a byte d = " + d + " b= " + b);
    }

}

```

Conversion de int a byte.

i = 257 b = 1

Conversion de double a byte.

De double a byte d = 323.142 b= 67

## Promoción de tipos en expresiones

Al evaluar las expresiones, el valor intermedio puede superar el intervalo de operandos y, por lo tanto, se promoverá (convertirá) el valor de la expresión. Algunas condiciones para la conversión de tipos son:

1. Java convierte automáticamente cada operando byte, short o char a int al evaluar una expresión que contenga int.
2. Si un operando es long, float o double, la expresión completa se convierte a long, float o double respectivamente.

### ConversionExplicitaDeTipos.java

```

package fundamentos;

public class ConversionExplicitaDeTipos {

    public static void main(String[] args)
    {
        double d = 100.04;

        //casteo explícito
        long l = (long)d;

        //casteo explícito
        int i = (int)l;
        System.out.println("double " +d);

        //perdida parcial de datos
        System.out.println("long " +l);

        //perdida parcial de datos
        System.out.println("int " +i);
    }

}

```

### Execution

Double value 100.04

Long value 100  
Int value 100

## 4.6 Palabras reservadas en Java

Aquí teneis una lista de palabras clave en el lenguaje de programación Java. **No puede utilizar cualquiera de los siguientes como identificadores en los programas.** Las palabras clave `const` y `goto` están reservadas, aunque no se utilicen actualmente. `true`, `false` y `null` pueden parecer palabras clave, pero en realidad son literales; no se pueden utilizar como identificadores en los programas.

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

## 4.7 Identificadores

Un **identificador** es una representación de código Java de un nombre (por ejemplo, para una variable). **Cada identificador es una secuencia de letras, dígitos, guiones bajos (`_`) y símbolos de moneda (`$`), el primero de los cuales no es un dígito.** Por ejemplo, las secuencias de caracteres `abc`, `Ab`, `$`, `abc123` y `a_b` son todos identificadores java legales, pero `Ab*`, `1abc` y `a + b` no lo son. Los identificadores distinguen entre mayúsculas y minúsculas, por lo que `Ab`, `ab` y `AB` son nombres diferentes. **Algunas palabras reservadas, como `public`, `static`, `int`, `double`, `String`, `true`, `false` y `null`, son especiales y no se pueden utilizar como identificadores.**

por ejemplo:

`$1`, `ab$3`, `_AB` son identificadores válidos

`1fd` `2$7` produciría un error de compilación -> no puede iniciar un identificador con un dígito

## 4.8 Variables

**Las variables** son **contenedores para almacenar datos**. Para declarar una variable, indicamos el

tipo seguido por el nombre de la variable.

Tipo    nombre  
**int**    myNum

Tipo de datos	Nombre	Asignar un valor es opcional
<b>Int</b>	myNum	= 5;

```
package fundamentos;
```

```
public class VariableEjemplos {

    public static void main(String[] args) {

        int miNum = 5;
        int tuNum=miNum + 7;// Integer (número entero)
        float miFloatNum = 5.99f;
        double miDouble=0.0d;

        boolean respuesta=true;

        char letraMinuscula ='c';

        respuesta = false;

        miDouble= miFloatNum + miNum;

    }

}
```

## Convenciones de nomenclatura de variables

Los programadores suelen seguir las convenciones estilísticas al nombrar elementos de un programa. En este curso, nuestra convención es dar a cada variable un nombre significativo que consta de una letra minúscula seguida de letras minúsculas, letras mayúsculas y dígitos. Usamos letras mayúsculas para marcar las palabras de un nombre de variable de varias palabras. Por ejemplo, usamos los nombres de variable i, x, y, sum, letraMinuscula y miDouble, entre muchos otros. Los programadores se refieren a este estilo de nomenclatura como caso camel.

## Cómo las variables se almacenan en la memoria RAM

Nuestro código, las variables incluidas se almacenan en la memoria. Por lo tanto, cada variable está limitada a una dirección de memoria o posición.

Dirección de Memoria		Valor binario en memoria
..		
..		
..		
1000000	<u>miNum</u>	5
1000001	<u>tuNum</u>	12
1000002	<u>miFloatNum</u>	5.99
1000003	<u>miDouble</u>	0.0
1000004	<u>respuesta</u>	
1000005	..	
1000006	<u>respuesta</u> =true	
1000007	..	
1000008		
1000009	..	

programa Java

## Seguimiento de los cambios en los valores de las variables


Como resumen final del propósito de las instrucciones de asignación, estudia como el siguiente código intercambia los valores de a y b (supongamos que a y b son variables int):

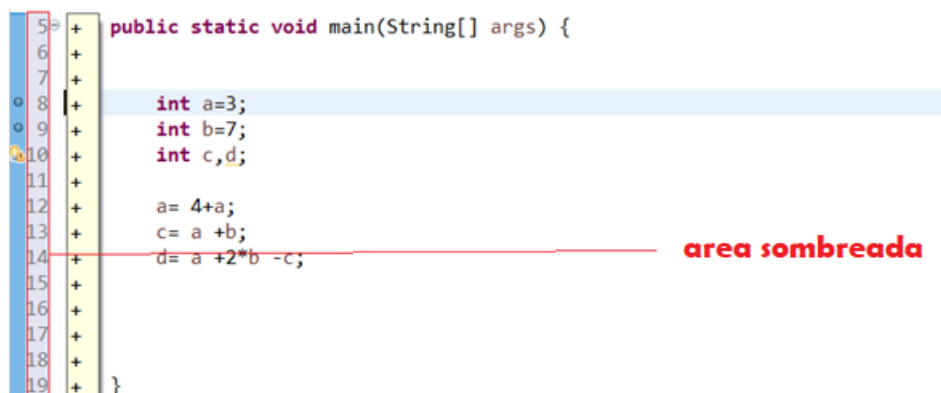
```
int t = a;
a = b;
b = t;
```

Para ello, introducimos puntos de interrupción en nuestro programa. Los puntos de interrupción son una herramienta ofrecida por los IDE para detener las ejecuciones de sus programas en el punto dado, lo cual es extremadamente útil, por lo que también podemos inspeccionar los valores de las variables.

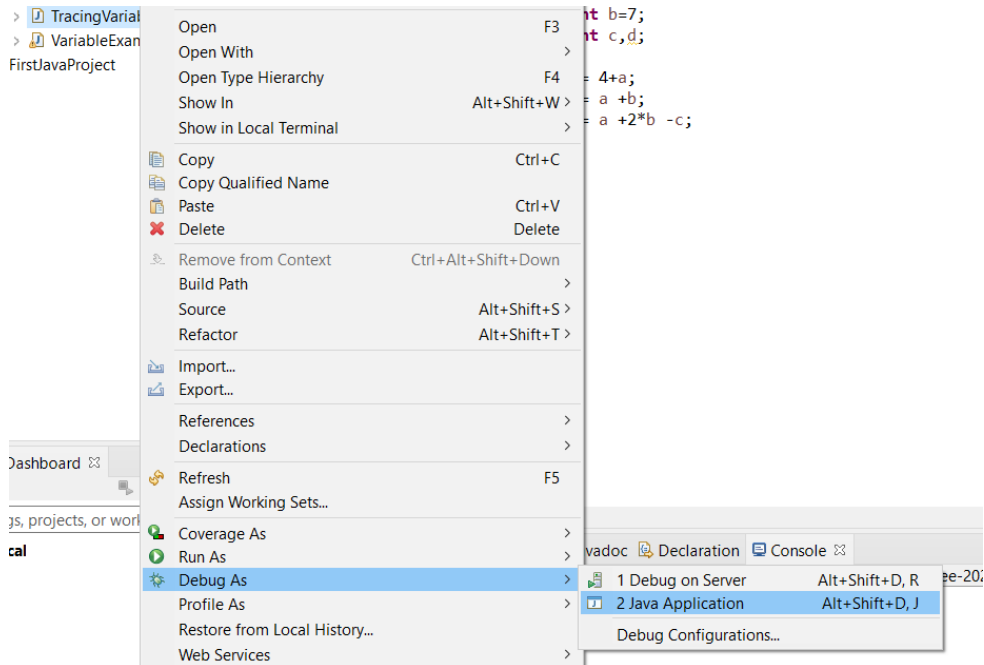
```
package fundamentos;
```

```
public class TracingVariables {  
  
    public static void main(String[] args) {  
  
        int a=3;  
        int b=7;  
        int c,d;  
  
        a= 4+a;  
        c= a +b;  
        d= a +2*b -c;  
  
    }  
}
```

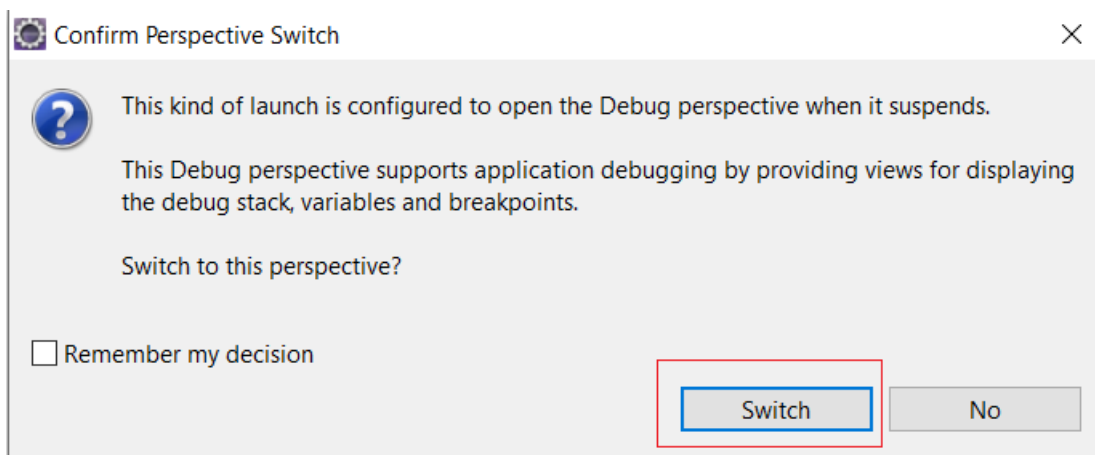
Para añadir un punto de ruptura se puede hacer doble clic en el área sombreada para iniciar el Editor IDE. Debe hacerlo en la línea base de la instrucción en la que desea pausar el programa. Tan pronto como hagas doble clic en un círculo azul debe aparecer a la izquierda en el número de línea de la instrucción  8.

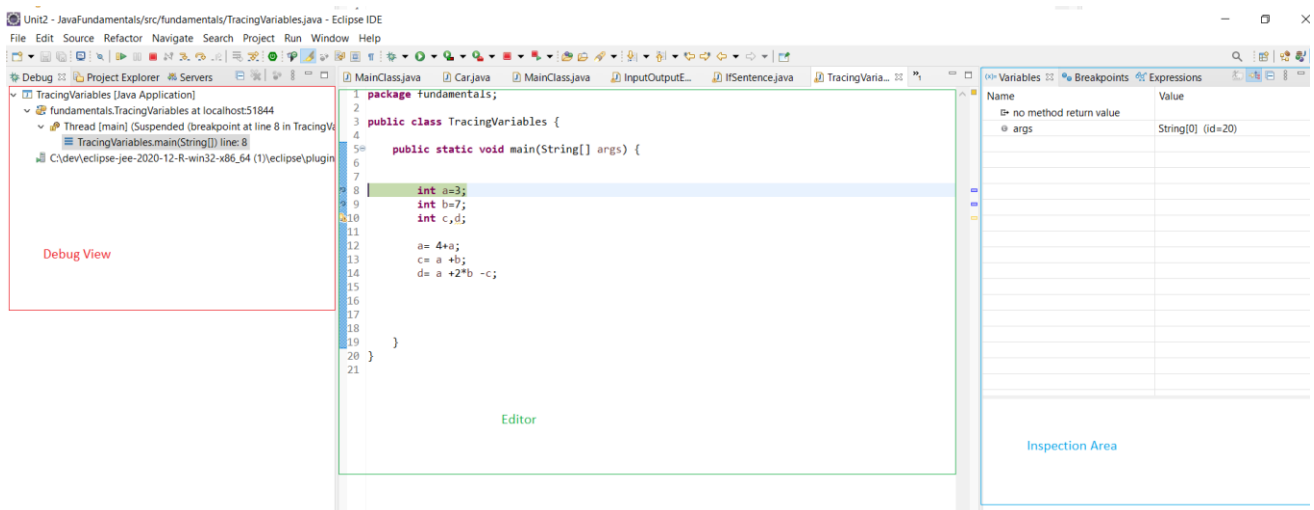


Después de corregir el punto de ruptura, para poder pausar el programa se debe depurar. Aunque es un proceso similar para ejecutar un programa, permite a los desarrolladores detener sus programas, inspeccionar los valores de almacén en variables y seguir el flujo del programa.

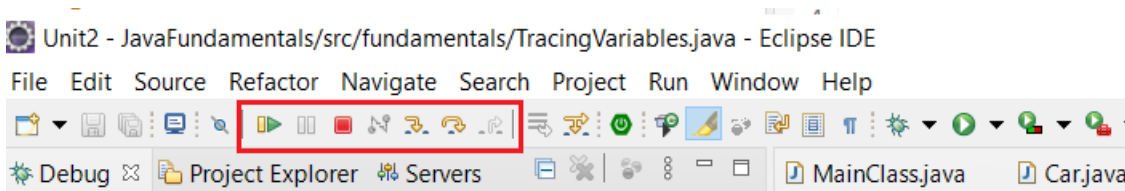


Siempre que inicies la depuración, el IDE solicita abrir la perspectiva de depuración. En esta perspectiva, se le permitirá inspeccionar variables y expresiones. Hagámoslo.





Puedes darte cuenta de que el programa se ha pausado justo en la primera línea de punto de interrupción. Hasta ahora, no hemos declarado todavía ninguna variable. Por lo tanto, no hay otra variable en el área de inspección que los parámetros principales. Para hacer que tu programa progrese, usaremos algunos iconos de depuración de la barra de herramientas relacionados con la perspectiva de depuración.



- El botón de play continua la ejecución del programa
- El botón de pause pausa la ejecución del programa en el instante dado
- El botón de stop, detiene totalmente la ejecución del programa
- El botón Step into, get into a function, providing that the statement is a function call.
- El botón Step Over button, ejecuta la instrucción y pasa a la siguiente.

Ahora, presiona el botón Step Over y comprueba el Área de inspección, en busca de cambios.

El flujo del programa pasa a la siguiente línea:



```

MainClass.java  Car.java  MainClass.java  InputOutput
1  package fundamentals;
2
3  public class TracingVariables {
4
5      public static void main(String[] args) {
6
7
8          int a=3;
9          int b=7;
10         int c,d;
11

```

Debido al hecho de que se declara la variable "a", ahora se nos permite comprobar su valor en el área de inspección. La declaración en línea `a=3;` produce el resultado deseado.

Name	Value
↳ no method return value	
🔍 args	String[0] (id=20)
🔍 a	3

En la memoria `a` está almacenando ahora un 3 como un valor. Todos los cambios que estamos inspeccionando a medida que ejecutamos el programa se almacenan en la memoria, recuérdalo.

Sigue ejecutando solo hasta la línea 12

```

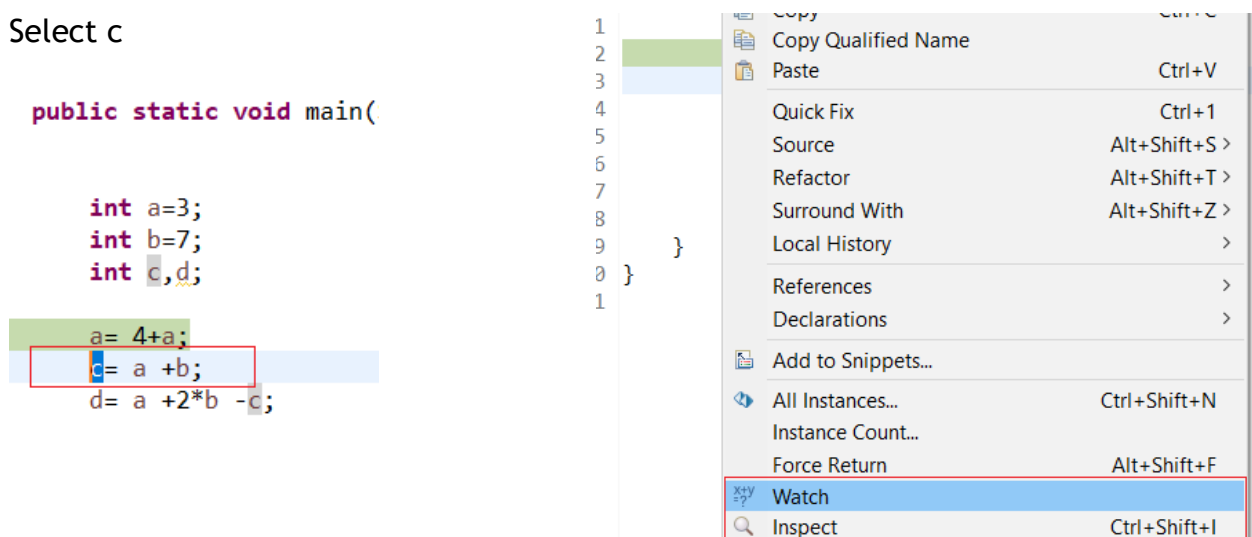
1  package fundamentals;
2
3  public class TracingVariables {
4
5      public static void main(String[] args) {
6
7
8          int a=3;
9          int b=7;
10         int c,d;
11
12         a= 4+a;
13         c= a +b;
14         d= a +2*b -c;
15
16
17
18

```

Name	Value
↳ no method return value	
🔍 args	String[0] (id=20)
🔍 a	3
🔍 b	7

Las variables a y b almacenan valores, pero c y d no. Incluso si se declaran variables, todavía no tienen valores asignados; Su valor es indefinido. Intentaremos observar o inspeccionar las variables c y d. Para realizar esto, puedes seleccionar la variable c, haga clic con el botón derecho del ratón y haz clic en la opción watch o inspect en el menú emergente.

Select c



En la zona de inspección están en las expresiones, se agrega c. Debido a que no almacena un valor, hay un error de evaluación. C no se puede resolver porque el valor de la variable no está definido.

## Watch

## Inspect



## 4.9 Consola input/output en Java

Para la entrada de consola, Java ofrece la clase Scanner. Más adelante introduciremos el concepto de clases. A partir de ahora debe utilizar la clase Scanner como se explica.

Para la salida usamos los métodos de la biblioteca System.out, System.out.println(), System.out.printf(), System.out.print(). Se explicará con ejemplos. Por lo tanto, echemos un vistazo al siguiente ejemplo InputOutputExample.java:

```
package fundamentos;

import java.util.Scanner;

public class InputOutputEjemplo {

    public static void main(String[] args) {

        double miDouble=0.0;

        Scanner sc = new Scanner(System.in);

        System.out.println("Introduce un numero con formato ddd.aaa, i.e. 55.666");
        miDouble = sc.nextDouble();

        System.out.println("El numero leído por la entrada es:"+ miDouble);

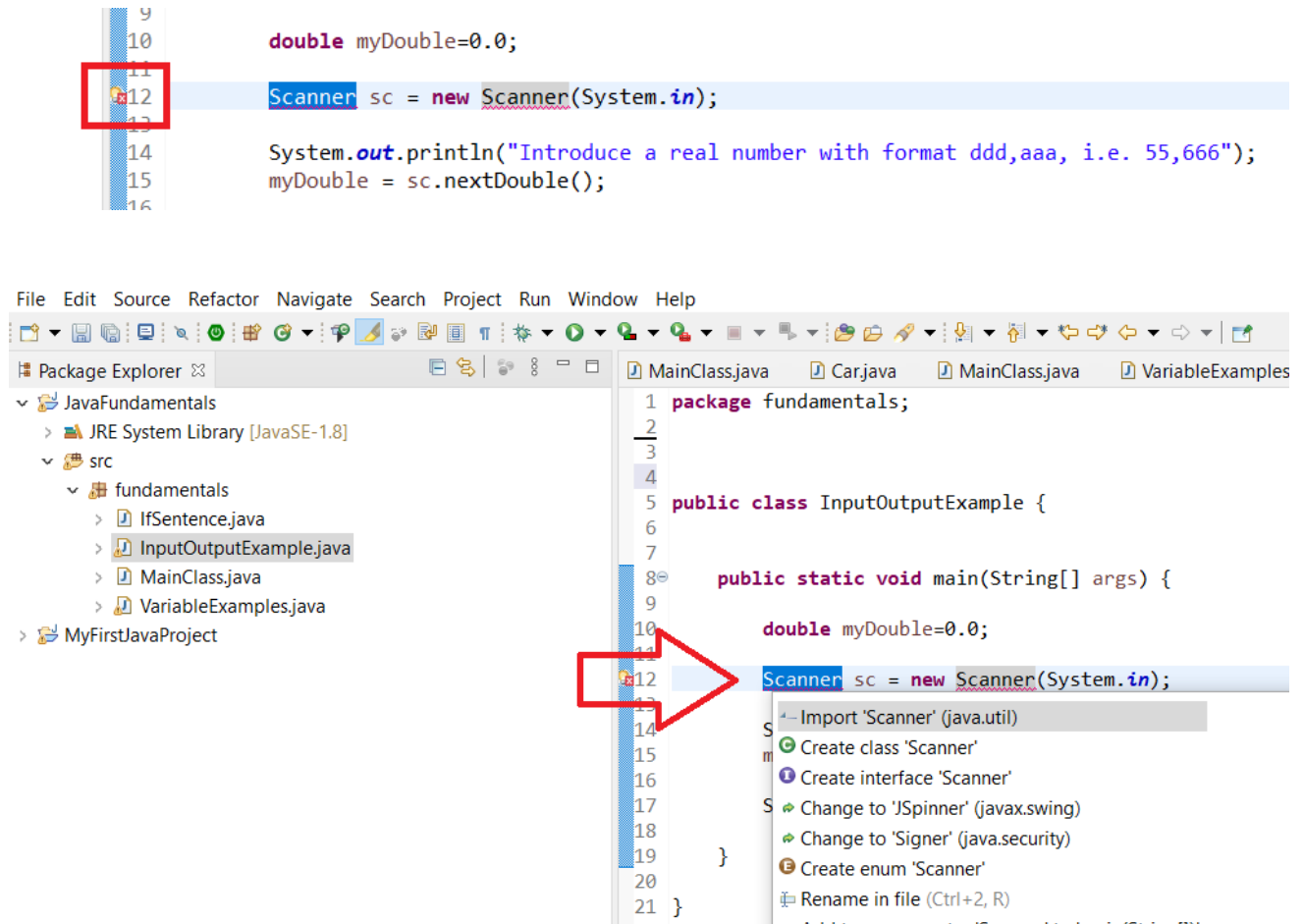
    }

}
```

Java permite a los programadores desarrollar programas Java que ofrecen la API de Java. Algunas de las clases se agregan implícitamente a los programas. Algunos otros deben importarse desde las bibliotecas de API de Java. Explicaremos el concepto de API en detalle en este capítulo. Sin embargo, solo para empezar, necesitaremos agregar algunas importaciones a nuestros programas.

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

El IDE Eclipse te dice cuándo hacerlo. En el caso de que la importación anterior no esté incluida en su programa, Eclipse mostrará un error de compilación. Como consecuencia, encontrarías el error de compilación ya que la oración o algunas de las palabras de su oración están subrayadas en rojo. El IDE haría sugerencias de cómo solucionar el problema. Para obtener los mensajes, haga clic en el icono x rojo a la izquierda de la oración.



Creamos un objeto Scanner, y lo asociamos con la entrada estándar en java `System.in`

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

```
myDouble = sc.nextDouble();
```

obtendrá el siguiente doble de la consola.

Para la salida `println` escribe una cadena en la salida estándar Java, la consola obtendrá la salida.

```
System.out.println("El numero leído por la entrada es:"+ miDouble);
```

Ahora, puedes ejecutar el programa y comprobar los resultados.

Introduce un numero con formato `ddd.aaa`, i.e. `55.666`

45,6

El numero leído por la entrada es:45.6

Los tipos de datos se describieron en el capítulo anterior. Scanner proporciona métodos para obtener datos de la consola para todos los tipos primitivos java. Algunos de los métodos se enumeran a continuación:

Method	Description
<code>nextInt()</code>	Lee un valor <code>int</code> introducido por el usuario
<code>nextFloat()</code>	Lee un <code>float</code> introducido por el usuario
<code>nextBoolean()</code>	Lee un <code>boolean</code> v
<code>nextLine()</code>	Lee una línea de texto introducido por el usuario (String)
<code>next()</code>	Lee una palabra introducida por el usuario
<code>nextByte()</code>	Lee un <code>byte</code> introducido por el usuario
<code>nextDouble()</code>	Lee un <code>double</code> introducido por el usuario
<code>nextShort()</code>	Lee un <code>short</code> introducido por el usuario
<code>nextLong()</code>	Lee un <code>long</code> introducido por el usuario

## Notas Cornell

### 4.9.1 Notas Cornell

Comentar cada línea del siguiente código después de montar el ejemplo.

¿Qué hace este programa?

```
package fundamentos;
```

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

```
public class InputOutputEjemplo2 {  
    final int MAX_TEMP = 120;  
  
    public static void main(String[] args) {
```

```
int edad=0;

Scanner sc = new Scanner(System.in);

System.out.println("Introduce la edad de la Persona");
edad = sc.nextInt();

if (edad >= 18)
    System.out.println("La persona es un adulto");
else if (edad <18)

    System.out.println("La persona is menor de edad");
else if (edad >=150)

    System.out.println("La persona no es humana con esa edad");

    }
}
```

## 4.10 Operadores Java

Esta sección proporcionará la lista completa de operadores java. En las instrucciones del programa, se combinarán operadores, literales y variables para realizar operaciones en cada una de nuestras oraciones de programa. Aquí un ejemplo.

```
int x= 0; // El valor de x es 0
int y; // El valor de Y aún no está definido
```

```
x=x+5; // El valor de x es 5
x++; // El valor de x es 6
```

```
y=3+x; // el valor de Y es 9
```

Para probar los operadores visita el sitio web de w3c.

[https://www.w3schools.com/java/java\\_operators.asp](https://www.w3schools.com/java/java_operators.asp)

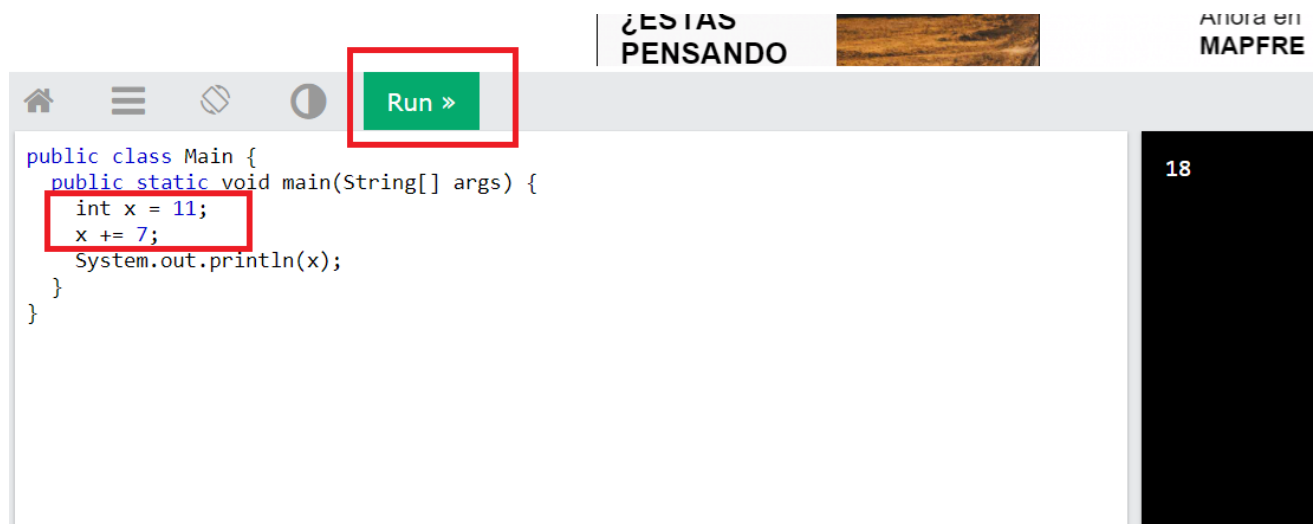
Haz click en try yourself button

### Example

```
int x = 10;  
x += 5;
```

[Try it Yourself »](#)

Cambia los valores de las variables y ejecuta el programa presionando el botón Run



#### 4.10.1 Operadores de asignación

Los operadores de asignación se utilizan para asignar valores a las variables. En el ejemplo siguiente, usamos el operador de asignación (=) para asignar el valor 10 a una variable llamada x:

```
int x = 10;
```

El operador de asignación de suma (+=) agrega un valor a una variable:

#### Example

```
int x = 10;  
x += 5; // es lo mismo que x = x+5; x contiene 15 después de la ejecución
```



Una lista de todos los operadores de asignación:

Operador	Ejemplo	Igual a	Significado
=	x = 5	x = 5	Asignación
+=	x += 3	x = x + 3	Asignación + suma
-=	x -= 3	x = x - 3	Asignación + resta
*=	x *= 3	x = x * 3	Asignación + producto
/=	x /= 3	x = x / 3	Asignación + division
%=	x %= 3	x = x % 3	Asignación + resto division
&=	x &= 3	x = x & 3	Asignación + and de bit
=	x  = 3	x = x   3	Asignación + or de bit

$\wedge =$	$x \wedge = 3$	$x = x \wedge 3$	Asignación + potencia $X = X^3$
$>> =$	$x >> = 3$	$x = x >> 3$	Asignación + desplazamiento de bit a la izquierda
$<< =$	$x << = 3$	$x = x << 3$	Asignación + desplazamiento de bit a la derecha

Para cada tipo de operadores, probaremos algunos de los operadores tutoriales de W3C. Hagámoslo.

#### 4.10.2 Operadores aritméticos

Los operadores aritméticos se utilizan para realizar operaciones matemáticas comunes.

Operator	Name	Description	Example
+	Suma	Suma dos valores	$x + y$
-	Resta	Resta un valor de otro	$x - y$
*	Multiplicación	Multiplica dos valores	$x * y$

/	División	Divide un valor por otro	x / y
%	Modulo	Devuelve el resto de la división.	x % y
++	Incremento	Aumenta el valor de una variable en 1	++x
--	Decremento	Disminuye el valor de una variable en 1	--x

### 4.10.3 Operadores binarios / operadores bit a bit

Java define varios operadores bit a bit, que se pueden aplicar a los tipos enteros, long, int, short, char y byte. El operador bit a bit funciona en bits y realiza la operación bit a bit. Supongamos que a = 60 y b = 13; ahora en formato binario serán los siguientes -

```

a = 0011 1100
b = 0000 1101
-----
a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

```

En la tabla siguiente se enumeran los operadores bit a bit y se ofrecen ejemplos de uso. En el ejemplo mencionado, una variable entera a contiene 60 y la variable b contiene 13

Operator	Description	Example
& (and de bit)	Operador Binario AND copia un bit al resultado si existe en ambos operandos.	(A & B) dará como resultado 0000 1100
(or de bit)	Operador OR binario copia el bit si existe en cualquier operando	(A   B) da como resultado 0011 1101, 61.
^ (XOR de bit)	El operador binario XOR copia el bit si se establece en un operando, pero no en ambos..	(A ^ B) da como resultado 0011 0001
~ (complemento)	Operador complemento a uno es unario y tiene el efecto de 'voltear' bits, ceros a unos y unos a ceros	(~A ) el resultado es 1100 0011 en la forma de complemento de 2
<< (desplazamiento a la izquierda)	Operador desplazamiento a la izquierda. El valor de los operando desplazamiento a la izquierda, se mueve a la izquierda por el número de bits especificado por el operando derecho.	A << 2 da como resultado 1111 0000
>> (desplazamiento a la derecha)	Operador desplazamiento a la derecha. El valor de los operandos izquierdos se mueve a la derecha por el número de bits especificado por el operando derecho.	A >> 2 da como resultado 1111
>>> (desplazamiento a la derecha con relleno de ceros)	Operador desplazamiento a la derecha relleno con ceros. El valor de operandos izquierdo se mueve a la derecha por el número de bits especificados por el operando derecho y los valores desplazados se llenan con ceros.	A >>>2 da como resultado 0000 1111

## Practica

1. Cree una clase en su llamada de proyecto **JavaSumaNumeros.java**. También agrega una función principal y programa este pseudocódigo en Java

```

PROGRAM JavaSumaNumeros
  Declare Integer CAN1, CAN2, SUM

  Assign CAN1 = 5
  Assign CAN2 = 3
  Assign SUM = CAN1 + CAN2
  Output "LA SUMA ES:"
  Output ToString(SUM)
End

```

2. Cree un programa Java en una llamada de clase java **AreaPentagono.java**. Recibe

como entrada la longitud lateral del pentágono regular, e imprime el Área regular del Pentágono en la consola. Debes buscar la fórmula en Internet.

### 4.11 Booleanos. Operadores Java Booleanos

Muy a menudo, en la programación, necesitará un tipo de datos que solo puede tener uno de dos valores, como:

- SÍ / NO
- ON / OFF
- VERDADERO / FALSO

Para ello, Java tiene un tipo de datos **boolean**, que puede tomar los valores **true** o **false**.

#### Valores Booleanos

Un tipo booleano se declara con la palabra clave **boolean** y solo puede tomar los valores **true** o **false**:

#### Ejemplo

```
boolean javaesdivertido= true;  
boolean aprogramar=true;  
boolean errores=false;  
boolean comentarios=false;
```

#### 4.11.1 Operadores que devuelven como resultado booleanos

#### Operadores de comparación de Java

Los operadores de comparación se utilizan para comparar dos valores:

Operador	Nombre	Ejemplo	Pruebalo
----------	--------	---------	----------

==	Igual a	x == y
!=	Distinto	x != y
>	Mayor	x > y
<	Menor	x < y
>=	Mayor o igual	x >= y
<=	Menor o igual	x <= y

Deberías probar algunos de los ejemplos del tutorial del W3C en el siguiente enlace:

[https://www.w3schools.com/java/java\\_operators.asp](https://www.w3schools.com/java/java_operators.asp)

## Operadores lógicos Java

Los operadores lógicos se utilizan para determinar la lógica entre variables o valores:

Operador	Nombre	Descripción	Ejemplo
&&	And lógico	Devuelve true si ambas expresiones son true	x < 5 && x < 10

	Or lógico	Devuelve true si una de las expresiones son true	$x < 5 \    \ x < 4$
!	Not lógico	Invierte el resultado, retorna false si el resultado es true	$!(x < 5 \ \&\& \ x < 10)$

Estudiaremos en detalle los operadores lógicos que repasan su verdadero gráfico.

### Operador And &&, tabla de verdad

Operación	Valor devuelto
true && true	True
true && false	False
false && true	False
false && false	False

El operador && devolverá true como un valor si y solo si ambas condiciones son verdaderas. Por ejemplo, supongamos que tiene una variable `int x = 5` en el programa

La operación lógica  $x > 0 \ \&\& \ x < 10$  debe devolver true, porque 5 es mayor que 0 menor que 10. Si una de las condiciones es false, el operador and && devolverá false. Para  $x > 6 \ \&\& \ x < 10$  sería false. Aunque 5 es menor que 10, 5 no es mayor que 6. Al final, ambas condiciones deben ser verdaderas, para permitir que una operación y sea verdadera.

### Operador Or || de operador , tabla de verdad

Operación	Valor devuelto
true    true	True
true    false	True
false    true	True
false    false	False

El operador Or || devolverá true un valor siempre que al menos una de las condiciones sea true. Por lo tanto, suponiendo que tiene una variable `y = 7`, la operación  $y > 8 \ || \ y < 10$  debe ser true porque 7 es menor que 10. Además, el  $y > 5 \ || \ y < 10$  debe ser cierto porque se

cumplen ambas condiciones.  $7 > 5$  y  $7 < 10$  son ambos verdaderos.

### Tabla de verdad del operador unario Not !

Operación	Valor devuelto
<b>!true</b>	<b>False</b>
<b>!false</b>	<b>True</b>

El operador NOT funciona invirtiendo (o negando) el valor de su operando. . Por ejemplo, considere que tiene una variable booleana `booleana bClose= true`. Si aplica el operador NOT a esta condición `!bClose`, el resultado debe ser false, ya que el operador not devuelve lo contrario de true, false.

Para terminar esta sección, prueba algunos de los ejemplos de W3C para operadores lógicos y el siguiente ejemplo.

### Ejemplo de valores booleanos en Java

Comprobar las tablas de verdad anteriores con este ejemplo:

#### EjemploBooleanos.java

```
package fundamentos;

public class EjemploBooleanos {

    public static void main(String[] argv)
    {
        boolean javaesdivertido= true;
        boolean programar=true;
        boolean errores=false;
        boolean comentarios=false;

        System.out.println(javaesdivertido&&programar);

        System.out.println(javaesdivertido&&errores);

        System.out.println(errores&&comentarios);

        System.out.println(javaesdivertido||programar);

        System.out.println(javaesdivertido||errores);

        System.out.println(errores||comentarios);

        System.out.println(!javaesdivertido);

        System.out.println(!comentarios);
```



```
}
```

```
}
```

### 4.11.2 Caracteres y cadenas

El tipo `char`, carácter representa caracteres alfanuméricos o símbolos individuales, como los que se escriben. Hay 216 valores de caracteres posibles diferentes, pero generalmente restringimos nuestro ámbito de uso a los que representan letras, números, símbolos y caracteres de espacio en blanco, como tabulación y nueva línea. Puedes especificar un literal de caracteres escribiendo un carácter entre comillas simples; por ejemplo, `'a'` representa la letra a.

Para tabulación, nueva línea o salto de línea, barra diagonal inversa, comillas simples y comillas dobles, usamos las secuencias de escape especiales `\t`, `\n`, `\\`, `\'`, y `\"`, respectivamente. Los caracteres se codifican como enteros de 16 bits mediante un esquema de codificación conocido como Unicode, y también hay secuencias de escape para especificar caracteres especiales que no se encuentran en el teclado. Por lo general, no realizamos ninguna operación directamente sobre caracteres que no sea la asignación de valores a las variables.

<i>values</i>	characters
<i>typical literals</i>	<code>'a'</code> <code>'\n'</code>

*Java's built-in char data type*

El tipo `String` representa secuencias de caracteres. Se puede especificar un literal `String` encerrando una secuencia de caracteres entre comillas dobles, como `"Hello, World"`. El tipo de datos `String` no es un tipo primitivo, pero Java a veces lo trata como tal. Por ejemplo, el operador de concatenación (+) toma dos operandos `String` y produce un tercer `String` que se forma añadiendo los caracteres del segundo operando a los caracteres del primer operando.

<i>values</i>	sequences of characters
<i>typical literals</i>	"Hello, World" " * "
<i>operation</i>	concatenate
<i>operator</i>	+

### *Java's built-in String data type*

La operación de concatenación (junto con la capacidad de declarar variables String y usarlas en expresiones y sentencias de asignación) es lo suficientemente potente como para permitirnos afrontar algunas tareas informáticas no triviales. Algunas expresiones de cadena típicas en Java pueden ser.

<i>expression</i>	<i>value</i>
"Hi, " + "Bob"	"Hi, Bob"
"1" + " 2 " + "1"	"1 2 1"
"1234" + " " + " " + "99"	"1234 99"
"1234" + "99"	"123499"

### *Typical String expressions*

## StringEjemplo.java

Mostramos ahora nuestro primer ejemplo del tipo String en Java. Lo que es especialmente útil en Strings es que puedes combinarlos con casi cualquier tipo primitivo usando el operador concat "+".

```
package fundamentos;
```

```
public class StringEjemplo {
```

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

```
    String miString= "Puedes concatenar cualquier tipo con Strings en Java y obtienes un String.";
```

```
    double media= 24.1;
```

```
    int miEdad= 35;
```

```
    final boolean STATEMENTTRUE=true;
```

```
    miString = miString + "\n" + "La edad media es " + media ;
```

```
    miString = miString + "\n" + "Sin Embargo, mi edad es " + media ;
```

```
miString = miString + "\n" + "Como resultado, estoy por encima de la edad media "
+ true;

    System.out.println(miString);

}

}
```

En el ejemplo anterior, declaramos cuatro tipos diferentes de variables: String, double, int, Boolean.

```
String miString= "Puedes concatenar cualquier tipo con Strings en Java y obtienes un String.";
double media= 24.1;
int miEdad= 35;
final boolean STATEMENTTRUE=true;
```

Java es tan flexible con Strings que permite a los programadores concatenar estos tipos primitivos con Strings. Automáticamente, a medida que opera la concatenación entre Strings y otros tipos, Java ofrece como resultado un nuevo String.

```
miString = miString + "\n" + "La edad media es " + media ;
```

El siguiente resultado resulta de las ejecuciones de esta sentencia. El programa se almacenará en MiString el resultado.

Puedes concatenar cualquier tipo con Strings en Java y obtienes un String.  
La edad media es 24.1

Ejecute el programa y compruebe que todos los tipos se han transformado en String después de utilizar el operador concatenación '+'. Aquí podéis ver un ejemplo de resultado:

Puedes concatenar cualquier tipo con Strings en Java y obtienes un String.  
La edad media es 24.1  
Sin Embargo, mi edad es 24.1  
Como resultado, estoy por encima de la edad media true

## Ejemplo Ruler.java

Como ejemplo más complejo, **Ruler.java** calcula una tabla de valores de la función ruler que describe las longitudes relativas de las marcas en una regla. Una característica notable de este cálculo es que ilustra lo fácil que es crear un programa corto que produce una gran cantidad de salida.

Si extiende este programa de la manera obvia para imprimir cinco líneas, seis líneas, siete líneas, etc., verás que cada vez que agregue dos instrucciones a este programa, duplica el tamaño de la salida. Específicamente, si el programa imprime  $n$  líneas, la  $n$ -ésima línea contiene  $2^n - 1$  números. Por ejemplo, si agregara instrucciones de esta manera para que el programa imprima 30 líneas, imprimiría más de 1.000 millones de números.

### Ruler.java

```
package fundamentos;

public class Ruler {
    public static void main(String[] args) {
        String ruler1 = "1";
        String ruler2 = ruler1 + " 2 " + ruler1; // 1 2 1
        String ruler3 = ruler2 + " 3 " + ruler2; // 1 2 1 3 1 2 1
        String ruler4 = ruler3 + " 4 " + ruler3; // 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
        System.out.println(ruler1);
        System.out.println(ruler2);
        System.out.println(ruler3);
        System.out.println(ruler4);
    }
}
```



### Ejecución

```
1
1 2 1
1 2 1 3 1 2 1
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```

## 5 Estructuras de control en Java

### 5.1 Estructuras de control en Java

Como hemos dicho anteriormente, en esta sección explicaremos en detalle las estructuras de flujo de control que ya introdujimos en la Unidad 2 para Diagramas de Flujo. En esta ocasión proporcionaremos formación en estas estructuras para Java. Como repaso, debemos señalar que existen cuatro estructuras básicas de flujo de control en la programación Java, que son la disposición de diferentes enunciados del lenguaje:

#### 1. Instrucción Declaración.

##### a. Declaración estandar

```
int i;
```

##### b. Declaracion en linea= declaration + asignacion

```
int i=0;
```

#### 2. Instruccion secuencial

##### a. Asignacion y expresi3n

##### i. Asignaci3n simple

```
i=0;
```

##### b. Asignaci3n compuesta: Puede incluir operadores y llamadas a m3todos

```
i= i + Coche.numeroDeCochesCreados();
```

##### c. Llamadas a m3todos

```
Coche.numeroDeCochesCreados ();
```

#### 3. Instrucciones de bloque : un grupo de instrucciones secuenciales rodeadas de llaves {

```
{
```

```
i= i + Coche.numeroDeCochesCreados();
```

```
System.out.println("Mi nuevo SUV:" + myTeslaSUVElectric.toString());
```

```
System.out.println("Total de coches creados: "+ Car.  
numeroDeCochesCreados());
```

```
miTeslaSUVElectric.repintar("blanco");
```

```
}
```

Se puede crear instrucciones a partir de métodos, o de las instrucciones condicionales e iterativas. Su principal característica es que cualquier instrucción en el método de bloque se ejecuta en orden secuencial una detrás de otra. Por lo tanto, en el bloque anterior la primera instrucción a ejecutar debe ser la asignación `i = i + Car.numberOfCarsCreated();`

A medida que esta asignación termina, la segunda a ejecutar sería

```
System.out.println("Mi nuevo SUV:" + myTeslaSUVElectric.toString());
```

Y así sucesivamente, hasta llegar al final del bloque de instrucciones.

4. Sentencias condicionales o de selección: controlan el flujo de nuestro programa basándose en condiciones lógicas.

```
if (edad >= 18)
    System.out.println("La persona es un adulto");
```

5. Instrucciones iterativas o bucles: ejecutan una instrucción o instrucción de bloque dependiendo de las condiciones establecidas por el programador. En el siguiente extracto de código, la instrucción iterativa `while` se repetiría 5 veces, cumpliendo la condición (`i < 5`). Puede inferir del siguiente ejemplo que las instrucciones iterativas pueden contener instrucciones de bloque. Además, las instrucciones condicionales pueden contener instrucciones de bloque.

```
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

6. Clases, Interface, declaración y definición de métodos. Los explicaremos en la siguiente unidad 4. Programación Orientada a Objetos. Dan forma a la estructura de nuestro programa, creando modelos complejos, que definen la modularidad y las dependencias de los módulos en nuestro programa. Vamos a mostrar un ejemplo. Aquí un ejemplo de una declaración de clase. En su interior puedes encontrar algunas declaraciones y definiciones de métodos:

```
package fundamentosorientacionaobjetos;
public class Coche {
    private String color;
    private String marca;
    private String modelo;
    private static int numCoches=0;

    public Coche (String color, String marca, String modelo) {

        this.color=color;
```

```

        this.marca=marca;
        this.modelo = modelo;
        Coche.numCoches++;
    }

    public void repintar(String color) {

        this.color=color;
    }

    @Override
    public String toString() {
        return "Car [color=" + color + ", marca=" + marca + ", modelo=" + modelo +
    "];";
    }

    public static void IncrementaNumeroDeCoches() {

        Coche.numCoches++;
    }
}

```

Ejemplo de declaración y definición de métodos

```

@Override
    public String toString() {
        return "Car [color=" + color + ", marca=" + marca + ", modelo=" + modelo +
    "];";
    }

```

Como puedes ver, tanto las clases como los métodos se componen de instrucciones de bloque.

### 5.1.1 Introducción a las sentencias condicionales en Java.

Java soporta las condiciones lógicas habituales de las matemáticas:

- Menos que:  $a < b$
- Menor o igual que:  $a \leq b$
- Mayor que:  $a > b$
- Mayor o igual que:  $a \geq b$
- Igual a:  $a == b$
- No es igual a:  $a != b$

Se puede utilizar estas condiciones para realizar diferentes acciones para diferentes decisiones en base a condiciones.

Java tiene las siguientes instrucciones condicionales:

1. Usar `if` para especificar un bloque de código que se va a ejecutar, si una condición

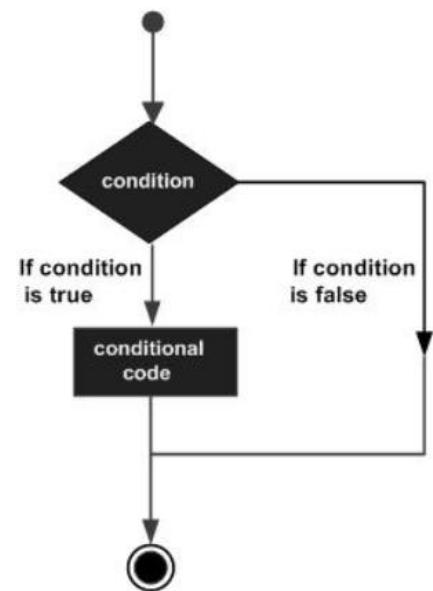
- especificada es true
2. Utiliza `else` para especificar un bloque de código que se va a ejecutar, si la misma condición es false
3. Usar `else if` para especificar una nueva condición para probar, si la primera condición es falsa
4. Utilizar `el modificador` para especificar muchos bloques de código alternativos que se ejecutarán

La instrucción `if` condicional tiene la siguiente estructura:

```
if (condition)
    statement;
```

o para un bloque de instrucciones:

```
if (condition) {
    statement1;
    statement2;
}
```



Una condición siempre devuelve un valor booleano, true o false. Colocamos la condición entre paréntesis. Puede ser una condición simple como `edad>19`, o una condición compleja (usando operadores condicionales), como `edad>19 && edad<69`.

Analizaremos la instrucción Java `if` en el siguiente ejemplo. Crearemos una nueva clase `InputOutputEjemplo2.java` y, a continuación, copiaremos este código:

```
package fundamentos;

import java.util.Scanner;

public class InputOutputEjemplo2 {
    final int MAX_TEMP = 120;

    public static void main(String[] args) {
        int edad=0;

        Scanner sc = new Scanner(System.in);

        System.out.println("Introduce la edad de la Persona");
        edad = sc.nextInt();

        if (edad >= 18)
```



```
        System.out.println("La persona es un adulto");  
    }  
}
```

El programa averigua si una persona es un adulto. Como entrada recibe la edad de la persona.

```
Scanner sc = new Scanner(System.in);  
        System.out.println("Introduce la edad de la Persona");  
        edad = sc.nextInt();
```

Suponiendo que la edad es mayor o igual a 18 años, el programa proporciona como resultado que la persona es un adulto.

```
if (edad >= 18)  
    System.out.println("La persona es un adulto");
```

Sin embargo, ¡cómo funciona esto! Es simple. Primero, se verifica la condición lógica. Por ejemplo, si la edad que recibe de la consola es edad=19, el programa valida (19>18) como true. Por lo tanto, la sentencia que sigue al si se ejecuta. Ejecute el programa y compruebe si hay resultados.

```
Introduce la edad de la persona  
19  
La persona es un adulto
```

Si la edad introducida es de 16 años, el programa no proporcionará ninguna respuesta, debido al hecho de que la instrucción `System.out.println("La persona es un adulto");` no se ejecuta.

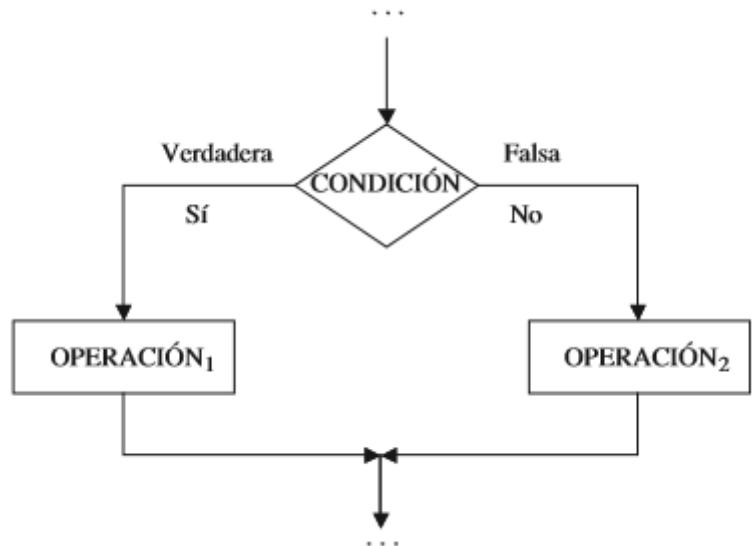
Para gestionar completamente ambos escenarios podemos añadir algo a la oración if. Ese es el if-else

La instrucción condicional if-else muestra la siguiente estructura:

```
if (condition)  
    statement1;  
else  
    statement2;
```

o para un bloque de instrucción:

```
if (condition) {
    statement1;
    statement2;
} else {
    Statement3;
    Statement4;
}
```



En el caso de las oraciones en bloque, **la tendencia o convención en java para escribir el else es:**

right brace else left brace

} else {

```
if (edad >= 18)
    System.out.println("La persona es un adulto");
else
    System.out.println("La persona is menor de edad");
```

Si (edad >= 18) es verdadero, el flujo del programa continuará en `System.out.println("La persona es un adulto");`. De otra modo se ejecutará `System.out.println("La persona is menor de edad");`

**If-else anidado , la instrucción else if**

Avancemos y hagamos que el ejemplo sea aún más complejo. Comprobaremos esta vez que la edad de la persona es superior a 150 no es una persona.

La instrucción condicional if-else tiene la siguiente estructura:

```
if (condition) statement1;
else if (condition) statement2;
else statement ;
```

, o para el bloque de instrucciones:

```
if (condition) {

statement1;
statement2;
}

else if (condition) {

statement3;
statement4;
...
...

}

else {
statement6;
statement7;
...
...

};
```

```
if (edad >= 18)
    System.out.println("La persona es un adulto");
else if (edad < 18)
    System.out.println("La persona es menor de edad");
else
    System.out.println("La persona no es humana con esa edad");
```

Fíjate, que esta vez estamos integrando tres escenarios. No hay límite para seguir agregando si es así; Sin embargo, se considera una mala práctica de codificación tener más de cinco o seis **si es que están** juntos. Por lo tanto, los desarrolladores evitan estas largas declaraciones condicionales que descomponen los programas.

Podemos ser más específicos usando los operadores condicionales. Vamos a explicarlo. ¿Qué pasa si queremos definir una condición más compleja, por ejemplo, la persona puede ser un adolescente? Por lo tanto, su edad debe estar entre los 15 y los 18 años. Para cumplir con este rango, podríamos escribir esta condición (`edad >= 15 && edad <= 18`)

```
package fundamentos;
```

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

```
public class InputOutputEjemplo2 {
    final int MAX_TEMP = 120;
```

```
public static void main(String[] args) {  
  
    int edad=0;  
  
    Scanner sc = new Scanner(System.in);  
  
    System.out.println("Introduce la edad de la Persona");  
    edad = sc.nextInt();  
  
    if (edad>= 18 && edad <150)  
        System.out.println("La persona es un adulto");  
    else if (edad1>=0 && edad<18)  
  
        System.out.println("La persona es menor de edad");  
  
    else if (edad<0)  
  
        System.out.println("La persona no ha nacido");  
    else if (edad>=150)  
  
        System.out.println("La persona no es humana con esa edad");  
  
    }  
}
```

### Actividad. Notas Cornell:

En esta actividad, debe comentar cada línea de su código. Por ejemplo:

```
// The program asks for the person age  
System.out.println("Introduce la edad de la Persona");  
  
//Asigna la entrada de consola a la variable Age  
edad = sc.nextInt();  
  
//La oración If verifica si la edad es mayor o igual a 15 años  
// y menos de 18  
    if (edad>= 18 && edad <150)
```

### Practica. Pruébalo tú mismo

En el ejemplo, necesitamos adolescentes. Modifica el ejemplo para controlar esa situación. Además, debe agregar jubilados al programa personas de 65 o más años de edad pero menores de 150. ¡Trate también de agregar al programa, personas adolescentes!

Siga estos pasos:

1. Escribir el código
2. Depurar el código
3. Comenta el código que has escrito

En esta unidad comenzaremos a trabajar en instrucciones de selección e instrucciones iterativas.

### 5.1.2 If anidados

Teniendo en cuenta que la instrucción if puede contener un bloque de oración, debe darse cuenta de que podemos incluir una instrucción if dentro de una instrucción if. En programación, etiquetamos este escenario como anidado si. Eso da como resultado que la instrucción if pueda ser parte de una instrucción de bloque. El anidado si debe seguir la siguiente estructura. Puede utilizar muchas variantes de esta estructura. Múltiples anidados si, si no- si.

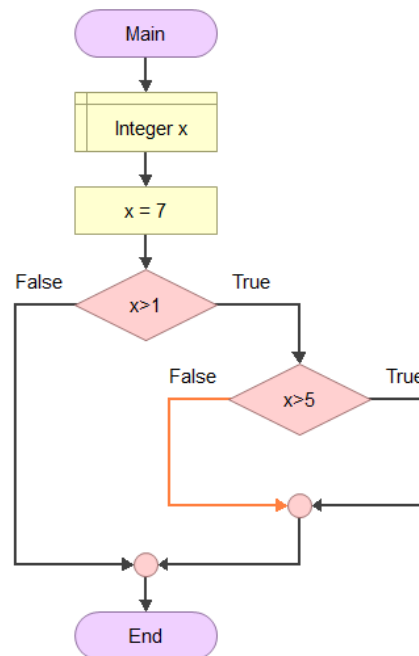
```
If (condicion1) {  
    Instruccion1;  
    If (condicion2) {  
        Instruccion 2;  
        Instruccion 3;  
    }  
    Instruccion 4;  
}
```

El diagrama de flujo es el siguiente y el código Java asociado es el siguiente:

```

int x = 7;
if (x > 1) {
    if (x > 5) {
    }
}

```



Para escribir código limpio, es muy recomendable no usar instrucciones if muy complejas. Además, para selecciones largas, la instrucción Java switch es la opción más adecuada. Revisaremos esta declaración en esta sección también. Introduzcamos un ejemplo anidado si:

```

package fundamentos;

import java.util.Scanner;

public class IfAnidado {

    public static void main(String[] args) {

        Integer numero=0;
        int numDivisores=0;
        Scanner sc = new Scanner(System.in);

        System.out.println("Introduce un entero");
        numero = sc.nextInt();

        if (numero%2==0) {

            System.out.println("el numero " + numero + " es divisible por
2");

            numDivisores++;
        } else if (numero%3==0) {

            System.out.println("el numero " + numero + " es
divisible por 3 y 6");

            numDivisores+=2;

```

```
    } else if (numero%5==0){  
  
        System.out.println("el numero " + numero + " es  
divisible por 5 y 10");  
        numDivisores+=2;  
    }  
  
}  
  
}
```

En este ejemplo, tenemos un if exterior

```
if (numero%2==0) {
```

y luego , la instrucción if- else interna, encerrada en la externa si

```
if (numero %3==0) {  
} else if (numero%3==0) {  
  
}
```

Cornell notes:

Haremos notas de cornell para este programa comentándolo línea por línea. Pero primero, puede depurar este programa dos veces.

Primera ejecución con entrada: 12

Segunda ejecución con entrada:20

Depuración: Intente seguir el flujo del programa paso a paso. Compruebe los valores de las variables en el área de inspección de depuración.

Comentarios: Después de eso, comente línea por línea lo que está haciendo el programa:

Ejemplo:

```
//Leemos el entero por pantalla
number = sc.nextInt();

//comprobamos si el número es divisible por 2
if (numero%2==0) {
```

### Actividad guiada

Escriba un programa Java que lea un número de coma flotante e imprima "cero" si el número es cero. De lo contrario, imprima "positivo" o "negativo". Agregue "pequeño" si el valor absoluto del número es menor que 1, o "grande" si excede 1,000,000.

### Actividad independiente

- 1) Escribe un programa Java que tome un número de la consola. Si el número está en el rango de 1 y 9, imprima una oración de saludo, también si el número es divisible por dos imprima "Somos par", mientras que si el número es divisible por 3, imprima "nos gustan los tríos de objetos".
- 2) **Más difícil.** Búsqueda web. Encuentra una manera de generar números aleatorios en Java. Modifique el programa anterior para obtener un número aleatorio entre 1 y 20 en lugar de leerlo desde la consola.

## 5.2 Instrucción Switch en Java

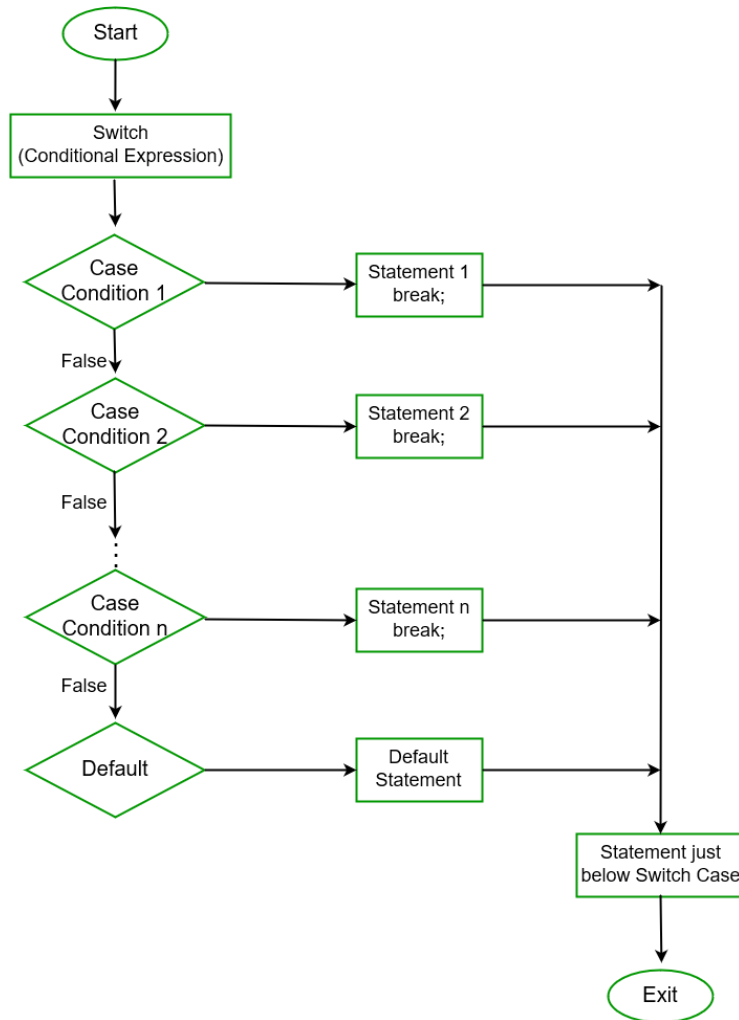
Las instrucciones Switch dan la posibilidad de elegir una opción entre muchas alternativas. Nos permite probar el valor de una expresión y, dependiendo de ese valor, saltar directamente a alguna ubicación dentro de la instrucción switch. Sólo se pueden utilizar expresiones de ciertos tipos. El valor de la expresión puede ser uno de los tipos enteros primitivos int, short o byte. Además, la expresión de valor puede ser un tipo char. Desde Java 7, puede utilizar el tipo String para la expresión de valor y las clases de ajuste para int (Integer), short (Short) y byte (Byte). La expresión no puede ser un número decimal, como float o double. En las versiones modernas de Java, podemos usar **tipos enumerados** o cambiar expresiones y casos



Las posiciones a las que puede saltar están marcadas con etiquetas de caso que toman la forma: "case hconstant-i:". Esto marca la posición a la que salta el equipo cuando la expresión se evalúa a la constante dada. Como caso final en una instrucción switch, puedes, opcionalmente, usar la etiqueta "default:", que proporciona un punto de salto predeterminado que se usa cuando el valor de la expresión no aparece en ninguna etiqueta de caso anterior.

Un switch statement, tiene la siguiente estructura:

```
switch (expression i) {  
    case hconstant-1:  
        instruccion-1;  
        break;  
    case hconstant-2 i:  
        instruccion-2;  
        instruccion-3;  
        instruccion-4;  
        break;  
    .  
    . // (more cases)  
    .  
    case constant-N i:  
        instruccion-N;  
        break;  
    default: // opción por defecto  
        instruccion -(N+1)  
} // fin del switch
```



Las instrucciones break son técnicamente opcionales. El efecto de un break es hacer que **EL PROGRAMA salte al final** de la instrucción switch. Si se omite la instrucción break, el programa seguirá adelante después de completar un caso y ejecutará las instrucciones asociadas con la siguiente etiqueta de caso.

en Java, **la instrucción break puede considerarse como una estructura de flujo de control per se**. Su función es sacar el flujo del programa de la instrucción de bloque donde se inserta. La instrucción break muestra el mismo comportamiento en bucles, aunque se considera una mala práctica utilizar las **instrucciones break en**. **Tratad de evitarlo**

En el siguiente ejemplo, todos los breaks están dentro del switch

```
switch (number) { // (Assume N is an integer variable.)
```

**case 1:**

```
System.out.println("El número es 1.");
break; // SI EL PROGRAMA EJECUTA ESTE BREAK

// EL BREAK LLEVA EL FLUJO DEL PROGRAMA HASTA EL FINAL
// DE LA FRASE DE CAMBIO. FUERA DEL CUERPO DEL INTERRUPTOR
```

```
...
...
...
```

```
} // cuerpo switch
```

Echemos un vistazo al siguiente ejemplo. Como el caso2 y el caso 4 no tienen break se ejecutará el caso 8 hasta llegar al break.

```
package sentenciaswitch;

import java.util.Scanner;

public class SwitchNumerico {

    public static void main(String[] args) {

        Integer numero = 0;

        Scanner sc = new Scanner(System.in);

        System.out.println("Introduce un entero entre 1 y 10");

        numero = sc.nextInt();

        switch (numero) { // (Asumimos que N es un entero.)
            case 1:
                System.out.println("El número es 1.");
                break;
            case 2:
            case 4:
            case 8:
                System.out.println("El número es 2, 4, or 8.");
                System.out.println("(Son potencia de 2!)");
                break;
            case 3:
            case 6:
            case 9:
                System.out.println("El número es 3, 6, or 9.");
```

```
        System.out.println("(Es múltiplo de 3!)");  
        break;  
    case 5:  
        System.out.println("El número es 5.");  
        break;  
    default:  
        System.out.println("El número es 7 o esta fuera del rango 1 to 9.");  
    }  
}  
}
```

## Actividad.

1. Añade el caso 7 para nuestro switch. Ejecute el programa, y explique los resultados.

```
case 7:  
    System.out.println("El número es 7.");
```

```
case 5:  
    System.out.println("El numero es 5.");  
    break;
```

- 1) Lo siguiente que harás es depurar el ejemplo dos veces. Uno para el número=2 y el otro para el número=8. ¿Son los resultados, los resultados esperados? Escribe los resultados y explícalos.
- 2) Parece que el programa no está bien escrito del todo. ¿Te importaría modificar el programa y considerar todas las opciones del 1 al 9?

## Menús e instrucciones de switch

Una aplicación de las instrucciones switch es en el procesamiento de menús. Un menú es una lista de opciones. El usuario selecciona una de las opciones. El ordenador tiene que responder a cada posible elección en un de manera diferente. Si las opciones están numeradas 1, 2, . . . , entonces el número de la opción elegida se puede usar en una instrucción switch para seleccionar la respuesta adecuada.

En un programa basado en consola, el menú se puede presentar como una lista numerada de opciones, y el usuario puede elegir una opción escribiendo su número.

El siguiente ejemplo codifica una aplicación de consola para convertir una cantidad de cualquier medida a pulgadas. Le da al usuario opciones. Después de seleccionar la opción,

requiere que una cantidad de unidades de la opción seleccionada se convierta en pulgadas. Podeis ver el resultado de la ejecución:

Que unidad de medida vas a usar?

1. pulgadas
2. pies
3. yardas
4. millas

Introduce el número seleccionado entre 1 y 10:

4

Introduce el número de millas:

63643634

La conversion a pulgadas es: 4.03246065024E12

Pasos para construir la aplicación:

1. Para escribir el menu usamos el método println

```
System.out.println("Que unidad de medida vas a usar?");
System.out.println();
System.out.println(" 1. pulgadas");
System.out.println(" 2. pies");
System.out.println(" 3. yardas");
System.out.println(" 4. millas");
System.out.println();
System.out.println("Introduce el número seleccionado entre 1 y 10: ");
opcionNumero = sc.nextInt();
```

2. Solicitamos la opción desde la consola y la almacenamos en la variable `opcionNumero`:

```
opcionNumero = sc.nextInt();
```

3. La opción del switch permite al programa distinguir entre opciones y realizar la conversión seleccionada.
4. Cada caso realizará la conversión específica. Podemos echar un vistazo al caso 2, ya que todos los casos funcionan de manera similar

**case 2:**

```
// 1. Lee el número de pies desde la consola
System.out.println("Introduce el número de pies: ");
medida = sc.nextDouble();
// 2. Se convierte de pies a pulgadas multiplicando por 12
pulgadas = medida * 12;

//3. Ofrece la solución en la salida de la consola
```

```
System.out.println("La conversión a pulgadas es: " + pulgadas);  
break;
```

La principal ventaja de la caja del switch es que permite al programa seleccionar entre varias posibilidades. Una vez seleccionada la opción, lo único que necesitará el programa es codificar el funcionamiento que requiere esta opción. En el caso 3, necesitábamos convertir de pies a pulgadas. Para hacerlo, necesitamos varios pies por 12.

### SwitchMenu.java

```
package sentenciaswitch;  
  
import java.util.Scanner;  
  
public class SwitchMenu {  
    public static void main(String[] args) {  
  
        int opcionNumero;  
        double medida;  
        double pulgadas;  
  
        Scanner sc = new Scanner(System.in);  
  
        System.out.println("Que unidad de medida vas a usar?");  
        System.out.println();  
        System.out.println(" 1. pulgadas");  
        System.out.println(" 2. pies");  
        System.out.println(" 3. yardas");  
        System.out.println(" 4. millas");  
        System.out.println();  
        System.out.println("Introduce el número seleccionado entre 1 y 4: ");  
        opcionNumero = sc.nextInt();  
  
        switch (opcionNumero) {  
            case 1:  
                System.out.println("Introduce el número de pulgadas ");  
                medida = sc.nextDouble();  
                pulgadas = medida;  
                System.out.println("La conversión a pulgadas es: " + pulgadas);  
                break;  
            case 2:  
                System.out.println("Introduce el número de pies: ");  
                medida = sc.nextDouble();  
                pulgadas = medida * 12;  
                System.out.println("La conversión a pulgadas es: " + pulgadas);  
                break;  
            case 3:  
                System.out.println("Introduce el número de yardas: ");  
                medida = sc.nextDouble();  
                pulgadas = medida * 36;  
                System.out.println("La conversión a pulgadas es: " + pulgadas);  
                break;  
            case 4:  
                System.out.println("Introduce el número de millas: ");  
                medida = sc.nextDouble();  
                pulgadas = medida * 63360;  
                System.out.println("La conversión a pulgadas es: " + pulgadas);  
                break;  
        }  
    }  
}
```

```
    }  
    }  
}
```

### 5.2.1 Tipos Enum y Java Switch

En esta sección, vamos a introducir el uso de Enumeration en instrucciones switch. Una de las ventajas de Java con respecto a la instrucción switch es la capacidad de usar enumeraciones como expresión de switch.

Otra de las novedades que ofrecía Java en sus versiones más recientes (desde Java 13) es la posibilidad de añadir listas o elementos a un caso:

```
package sentenciaswitch;
```

```
// A Java program to demonstrate working on enum  
// in switch case (Filename Test. Java)  
import java.util.Scanner;  
  
// An Enum class  
enum Mes  
{  
    ENERO, FEBRERO, MARZO, ABRIL, MAYO, JUNIO,  
    JULIO, AGOSTO, SEPTIEMBRE, OCTUBRE, NOVIEMBRE, DICIEMBRE;  
}  
  
// Driver class that contains an object of "day" and  
// main().
```

```
public class EjemploEnumMesesDelAno {  
    private Mes mes;  
  
    // Constructor  
    public EjemploEnumMesesDelAno(Mes mes)  
    {  
        this.mes = mes;  
    }  
  
    public Mes getMes() {  
        return this.mes;  
    }  
  
    // Prints a line about Day using switch  
    public void elDiaEs()  
    {  
        switch (mes)  
        {
```

```

        case DICIEMBRE, ENERO, FEBRERO:
            System.out.println("Frio, a veces lluvioso a veces nevado.");
            break;
        case MARZO, ABRIL, MAYO:
            System.out.println("La primavera ha llegado, mejor tiempo.");
            break;
        case JUNIO, JULIO, AGOSTO:
            System.out.println("Tiempo de verano, vacaciones para disfrutar de un
tiempo humedo y caluroso.");
            break;
        case SEPTIEMBRE, OCTUBRE, NOVIEMBRE:
            System.out.println("Buen tiempo para actividades físicas hasta
noviembre.");
            break;
        default:
            System.out.println("Te has olvidado algún mes?");
            break;
    }
}

// Driver method
public static void main(String[] args)
{
    String str = "JUNIO";
    EjemploEnumMesesDelAño t1 = new EjemploEnumMesesDelAño(Mes.valueOf(str));
    System.out.println("Como es el tiempo en " + t1.getMes() + "?");
    t1.elDiaEs();
}
}

```

La propiedad mes es de tipo de enumeración Month:

```
private Mes mes;
```

Podemos usarlo como parte de la expresión switch:

```
switch (mes)
```

También podemos definir una lista para cada de caso, `case DICIEMBRE, ENERO, FEBRERO:`

```

switch (mes)
{
    case DICIEMBRE, ENERO, FEBRERO:
        System.out.println("Frio, a veces lluvioso a veces nevado.");
        break;
    case MARZO, ABRIL, MAYO:
        System.out.println("La primavera ha llegado, mejor tiempo.");
        break;
    case JUNIO, JULIO, AGOSTO:

```

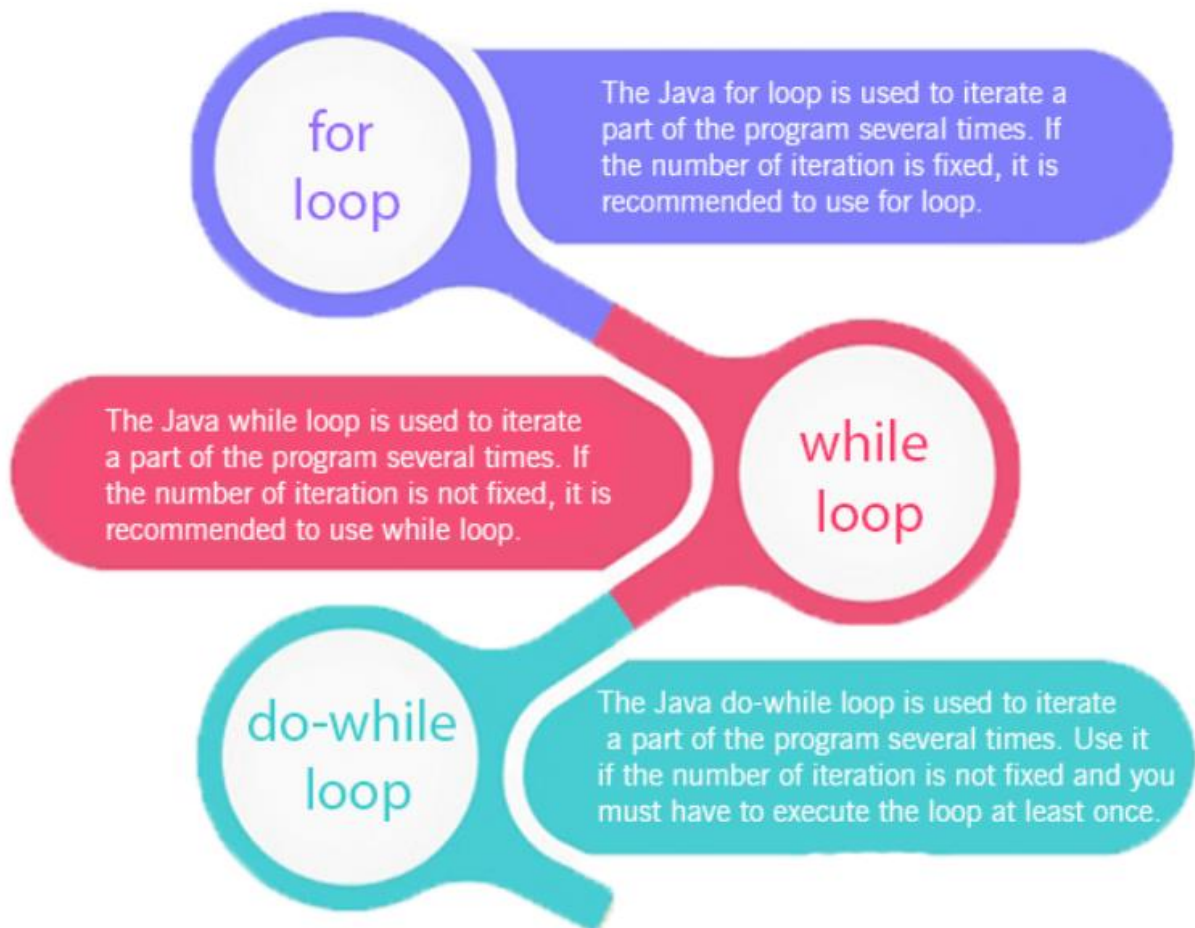


### **5.3 Instrucciones iterativas en Java. Bucles**

El lenguaje de programación java proporciona un conjunto de instrucciones iterativas que se utilizan para ejecutar una instrucción o un bloque de instrucciones repetidamente, siempre y cuando se satisfaga la expresión condicional dada, lo que significa que es cierto. Las instrucciones iterativas también se conocen como instrucciones en bucle o instrucciones repetitivas. Java proporciona las siguientes instrucciones iterativas.

1. Instrucción while
2. Instrucción do-while
3. Instrucción for
4. Instrucción for-each

Esta instrucción de control es una de las herramientas más poderosas de los lenguajes de programación. La capacidad de ejecutar un bloque de oraciones repetidas veces, permite a los programadores implementar algoritmos más complejos. Desde los programas de menú fáciles, hasta las operaciones matemáticas difíciles, como los números factoriales y primos, se resuelven fácilmente con las adiciones de bucles.



### 5.3.1 Variables en instrucciones de bucle.

Los acumuladores son objetos utilizados por un programa y por la función que realizan dentro del mismo toman un nombre **especial**, modelando su funcionamiento debido a su uso frecuente. Estas variables almacenan datos, como la suma de una serie de cantidades, o el recuento del número de alumnos en clase, o la nota máxima y mínima de un alumno, etc.

#### Contadores.

**Un contador** es un objeto que se utiliza para contar cualquier evento que pueda ocurrir dentro de un programa. Por lo general, cuentan naturalmente de 0 a 1 en 1, aunque se pueden realizar otros tipos de cuentas requeridas en algunos procesos.

Se utilizan realizando dos operaciones básicas sobre ellos:

- 1) **Inicialización**: Cualquier contador se inicializa en 0 si realiza una cuenta natural o en otro valor si desea realizar otro tipo de cuenta.

`C1 = 0;`

```
CP = 2;
```

- 2) **Decremento o incremento:** Cada vez que el evento a contar es aumentar o decrementar el contador en 1 si se hace una cuenta natural o a otro valor si se realiza otro tipo de cuenta.

```
C1 = C1 + 1;  
CP = CP + 2;  
CP = CP - 1;
```

## Acumuladores.

Acumuladores. Los acumuladores son un tipo especial de variable que básicamente usamos para actualizar algunos datos en ejecuciones repetidas. Una cosa que realmente debemos recordar es que la operación por la cual se produce la actualización del punto de datos tiene que **ser una operación asociada y conmutativa.**

Se trata de objetos que se utilizan en un programa para acumular elementos sucesivos con la misma operación. Generalmente se utilizan para calcular sumas y productos, sin descartar otros posibles tipos de acumulación.

Se realizarán dos operaciones básicas en ellos para su uso:

- **Inicialización:** Cualquier acumulador se inicializa a 0 si haces sumas y a 1 si haces productos.

```
SUM = 0;  
PRODUCT = 1;
```

1. **Acumulación:** Cuando el elemento a acumular mediante la realización de una lectura o cálculo está presente en la memoria, el elemento se acumula mediante la asignación:

```
SUM = SUM + CANTIDAD;  
PRODUC = PRODUC * NUMERO;
```

## Máximos y Mínimos.

**Un máximo** es una variable que guarda el valor máximo de un conjunto de valores. El máximo se inicializa a un valor muy pequeño para que seamos una seguridad de que el primer valor almacenado en esta variable es el mayor.

**Un mínimo** es similar al máximo solo tomando el valor más pequeño de una serie de valores. El mínimo se inicializa a un valor máximo.

### 5.3.2 El bucle While básico

La instrucción simple o de bloque (`{ ... }`) por sí sola realmente no afecta el flujo de control en un programa. Las cinco estructuras de control restantes lo hacen. Se pueden dividir en dos clases: instrucciones de bucle y instrucciones de ramificación, los `if` y `switch`'s. Realmente solo se necesita una estructura de control de cada categoría para tener un lenguaje de programación de propósito completamente general. Más instrucciones de bucle o selección o ramificación se utilizan para añadir más funcionalidad y opciones a un lenguaje de programación

En esta sección, introducimos el bucle `while` y la instrucción `if`. Vamos a explicar todos los detalles de esta instrucción y de las otras tres estructuras de control en secciones posteriores.

Un bucle `while` se utiliza para repetir una instrucción dada una y otra vez. Por supuesto, no es probable que queramos seguir repitiéndola para siempre. Eso sería un bucle infinito, que generalmente es algo malo, o una mala práctica de programación. (Hay una vieja historia sobre la pionera de la informática Grace Murray Hopper, quien leyó instrucciones en una botella de champú diciéndole que "enjabona, enjuague, repita". Según cuenta la historia, ella afirma que trató de seguir las instrucciones, pero se quedó sin champú.

Para ser más específicos, un bucle `while` repetirá una declaración una y otra vez, pero solo hasta el momento en que la condición especificada sigue siendo verdadera. Un bucle `while` tiene la forma:

While de instrucción simple:

```
while(condicion)
    Instrucción 1;
```

While de instrucción bloque:

```
while (condition) {
    Instrucción 1;
    Instrucción 2;
    Instrucción 3;
    ....
    Instrucción n;
}
```

Tanto la instrucción simple como la instrucción de bloque se conocen como el cuerpo del `while`.

Considera el siguiente código Java. Hemos declarado **la variable de contador i**. La primera vez que el flujo del programa alcanza la instrucción `while`, se verifica la condición `i<=10`. Dado que el valor de `i` es 1, la condición es `true`, el flujo del programa entra en el bloque de bucle. En la instrucción `block`, se escribe en la consola el número de repetición de bucle, y luego el valor de `i` se incrementa una unidad. Como resultado, "Número de repetición de bucle: 1" se escribe en la consola y el valor de la variable `i` es 2.

Este esquema se repite hasta que el tiempo no cumple con las condiciones. En otras palabras, para terminar el bucle el valor de `i` debe ser 11, de modo que 10 no sea mayor o igual que 10. En total, el cuerpo se ejecuta diez veces. Recuerde, si y mientras que las condiciones se **conocen como expresiones condicionales de test**

```
Número de repetición en el bucle: 1
Número de repetición en el bucle: 2
Número de repetición en el bucle: 3
Número de repetición en el bucle: 4
Número de repetición en el bucle: 5
Número de repetición en el bucle: 6
Número de repetición en el bucle: 7
Número de repetición en el bucle: 8
Número de repetición en el bucle: 9
Número de repetición en el bucle: 10
```

Se recomienda depurar este programa paso a paso. Hagámoslo.

```
package bucles;

public class WhileSimple {

    public static void main(String[] args) {
        int i;

        i = 1;
        while (i <= 10) {
            System.out.println("Número de repetición en el bucle: " + i);
            i = i + 1;
        }
    }
}
```

## Bucles infinitos

Como somos humanos, podemos cometer errores. Siguiendo esa línea de pensamiento, no es raro programar bucles infinitos. Por lo general, está relacionado con la configuración de condiciones incorrectas para el bucle. En otras palabras, los programas tienden a ser complejos y, a veces, cometemos errores. Por ejemplo, en el siguiente ejemplo, la condición está mal definida y el flujo del programa nunca sale del bucle.

```
package bucles;

public class BucleInfinito {

    public static void main(String[] args) {
        int i;

        i = 1;
        while (i >= 0) {

            System.out.println("Numero de Repetición en el bucle: " + i);
            i = i + 1;
        }
    }
}
```

Preste atención a la condición de test de bucle `while (i >= 0)`. El valor inicial de `i` es 1, por lo tanto, la primera vez que alcanza el `while`, `1 >= 0` es verdadero. Sin embargo, el bloque de bucle aumenta la variable `i` en una en cada repetición. Ciertamente, la condición `i >= 0` se cumple. El programador cometió un error al establecer la condición. Inevitablemente, el programa nunca abandonará la declaración `while`. Este código sería como el siguiente ejemplo, dado que la expresión de prueba (`i >= 0`) siempre se satisface.

```
while (true) {

    System.out.println("Numero de Repetición en el bucle: " + i);
    i = i + 1;
}
```

En programación, llamamos a este escenario **como bucle infinito. Tenemos que evitarlo a cualquier precio.**

### 5.3.3 Bucles usados para hacer menús.

En este apartado, combinaremos la instrucción `switch` para crear un menú, con un bucle para mantener el menú abierto hasta que pulsemos el teclado Exit, 5.

Necesitamos escribir pocas modificaciones en el `MenúSwitch`. Para crear este nuevo programa, incrustaremos el código del `SwitchMenu.java` ejemplo dentro del cuerpo de un tiempo. En la expresión de prueba `while`, controlaremos que la tecla pulsada no sea 5. Por el contrario, siempre que el número de entrada de la consola sea 5, el programa saldría del bucle. El amarillo resalta los cambios realizados con respecto al `SwitchMenu.java` ejemplo

Primero, comprobamos la opción elegida por el usuario guardada en `opcionNumero` en la expresión condicional del `while`. Si es 5 el flujo del programa salga del bucle. Por el contrario, la ejecución del programa continúa hasta el cuerpo. En el cuerpo `while` tenemos el **menú de salida**, la instrucción de **entrada** y el **insrucción switch** para realizar la operación que hemos leído desde la consola.

```
while (opcionNumero != 5) {
    System.out.println("Que unidad de medida vas a usar?");
    System.out.println();
    System.out.println(" 1. pulgadas");
    System.out.println(" 2. pies");
    System.out.println(" 3. yardas");
    System.out.println(" 4. millas");
    System.out.println();
    System.out.println("Introduce el número seleccionado entre 1 y 10:");

    opcionNumero = sc.nextInt();

    switch (opcionNumero) {
```

Menú

Entrada de Console

Switch

La principal ventaja de esta implementación es que podemos repetir el menú infinitas veces hasta pulsar 5. Este código añadido produce como resultado que el usuario puede realizar operaciones de conversión más de una vez. El menú estará disponible para hacer cálculos de medición hasta que el usuario pulse 5.

Podemos verificar esta nueva funcionalidad ejecutando el programa. Se describe lo que sucede durante la primera iteración. Debéis inferir lo que sucede en la segunda y tercera iteración. Explícalo escribiendo que sucede como en el siguiente ejemplo.

Que unidad de medida vas a usar?

1. pulgadas
2. pies
3. yardas
4. millas
5. Sal del programa

Introduce el número seleccionado entre 1 y 5:

:

Primera iteración del while

1 El usuario selecciona 1

La conversión a pulgadas es: 3.0 El programa convierte de pulgadas a pulgadas dado que seleccionamos el caso 2 del switch.

Que unidad de medida vas a usar?

1. pulgadas
2. pies
3. yardas
4. millas
5. Sal del programa

Introduce el número seleccionado entre 1 y 5: Segunda iteración del while

4

Introduce el numero de millas:

2

La conversión a pulgadas es: 126720.0

Que unidad de medida vas a usar?

1. pulgadas
2. pies
3. yardas
4. millas
5. Sal del programa

Introduce el número seleccionado entre 1 y 5: Ultima iteración del while

5

Final del programa. Adios

## WhileMenu.java

```
package bucles;
```

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

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

```
        int opcionNumero=1;
```

```
        double medida;
```

```
        double pulgadas;
```

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

```
        while (opcionNumero != 5) {
```

```
            System.out.println("Que unidad de medida vas a usar?");
```

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

```
            System.out.println(" 1. pulgadas");
```



```

System.out.println(" 2. pies");
System.out.println(" 3. yardas");
System.out.println(" 4. millas");
System.out.println();
System.out.println("Introduce el número seleccionado entre 1 y 5: ");
opcionNumero = sc.nextInt();

/* Read user's measurement and convert to inches. */
switch (opcionNumero) {
case 1:
    System.out.println("Introduce el número de pulgadas ");
    medida = sc.nextDouble();
    pulgadas = medida;
    System.out.println("La conversión a pulgadas es: " + pulgadas);
    break;
case 2:
    System.out.println("Introduce el número de pies: ");
    medida = sc.nextDouble();
    pulgadas = medida * 12;
    System.out.println("La conversión a pulgadas es: " + pulgadas);
    break;
case 3:
    System.out.println("Introduce el número de yardas: ");
    medida = sc.nextDouble();
    pulgadas = medida * 36;
    System.out.println("La conversión a pulgadas es: " + pulgadas);
    break;
case 4:
    System.out.println("Introduce el número de millas: ");
    medida = sc.nextDouble();
    pulgadas = medida * 63360;
    System.out.println("La conversión a pulgadas es: " + pulgadas);

case 5:
    System.out.println("Final del programa. Adios! ");
    break;

}

}

}

}

```

### 5.3.4 Ejercicio bucle.

Mejorar el programa anterior para no repetir cuatro veces las siguientes sentencias:

```

System.out.println("Introduce el número de yardas: ");
    medida = sc.nextDouble();

```

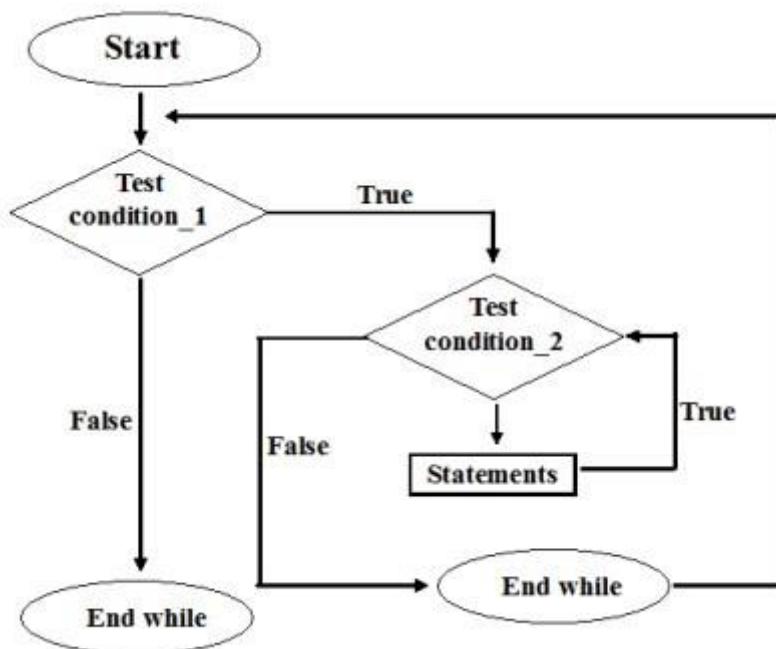
### 5.3.5 Bucle anidado while

Una vez más, como introdujimos anteriormente para las declaraciones if, podemos incluir declaraciones while dentro de un cuerpo de otro while. En otras palabras, un bucle dentro de un bucle. Cuando existe un bucle while dentro del cuerpo de otro bucle while, se conoce como bucle anidado while en Java. Inicialmente, el bucle externo se ejecuta una vez y el bucle interno posterior comienza a ejecutarse. La ejecución del bucle interno continúa hasta que se cumple la condición del bucle interno (hasta que la expresión de prueba es falsa).

Una vez que se satisface la condición del bucle interno, el flujo de control sale al bucle externo para la siguiente iteración. Para mejorar el rendimiento de nuestro programa, tenemos que controlar la duración del bucle anidado, para no crear whiles ineficientes. Debe tener en cuenta que cada vez que el bucle externo itera significa que el bucle interno ha completado todas sus iteraciones. En este caso, siempre que el bucle interno ejecute 100 iteraciones y el bucle externo realice 60 iteraciones, tenemos un total de  $60 * 100$  iteraciones, seis mil.

Para mejorar el rendimiento y la eficiencia de nuestros programas, tratamos de reducir el número de iteraciones tanto como podamos. Para lograrlo, intentamos definir las expresiones de prueba óptimas para nuestros bucles. Dicho esto, podemos continuar con un ejemplo de ratos anidados.

El diagrama de flujo para los while anidados es el siguiente. El diagrama vuelve al comienzo de cada bucle mientras se cumpla las condiciones. Cuando no se cumple sale del bucle.



En el siguiente ejemplo, se nos crea un programa que calcula la cuarta potencia para cada número del while externo. Para completar el cálculo, necesitamos un bucle interno que se repita cuatro veces. Cada vez multipliquemos la variable de bucle externo en el bucle interno.

A primera vista, se podrían identificar dos variables de contador, la variable *i* para el mientras que externo y la variable *j* para el mientras interno. Además, la variable de resultado, que es un acumulador, se inicializa con un 1 `int result = 1;`. Sirve para calcular y almacenar la cuarta potencia para cada valor de la variable *i*. En cualquier nueva iteración de bucle exterior, restablecemos el resultado a 1 con el objetivo de proceder a un nuevo cálculo de potencia.

.

### BucleAnidado.java

```
package bucles;

public class BucleAnidado {
    public static void main(String args[]) {
        int i = 1;
        int result = 1;

        while (i <= 3) {
            System.out.println("\n" + i + " " + "outer loop executed only once");
            System.out.println("input number to calculate the fourth power:" + i
+ " \n");

            int j = 1;
            result = 1;
            while (j <= 4) {

                result *= i;
                System.out.println(j + " " + "inner loop executed until to
completion");
                j++;
            }
            System.out.println("\nthe fourth power of the number " + i + " is "
+ result);

            i++;
        }
    }
}
```

Podemos ejecutar el programa y analizar los resultados.

En la primera iteración del while externo, solo completa una iteración, mientras que el bucle interno completa toda su iteración, cuatro. El número para calcular la potencia a cuatro es el valor almacenado en la variable *i*, 1. El resultado de la operación de energía es 1.

1 Por cada vez que se ejecuta el bucle exterior

Se ejecuta cuatro el interior **i=1**

1 iteración del bucle interno **j=1**

2 iteración del bucle interno **j=2**

3 iteración del bucle interno **j=3**

4 iteración del bucle interno **j=4**

La cuarta potencia del numero 1 es 1 **result=1 (1<sup>4</sup>=1)**

Posteriormente, en la segunda iteración, el valor de i se incrementa en 1; por lo tanto i=2.

1 Por cada vez que se ejecuta el bucle exterior

Se ejecuta cuatro el interior **i=2**

1 iteración del bucle interno **j=1**

2 iteración del bucle interno **j=2**

3 iteración del bucle interno **j=3**

4 iteración del bucle interno **j=4**

La cuarta potencia del numero 2 es 16 **result=1 (2<sup>4</sup>=16)**

### Cornell notes. Actividad

Después de las ejecuciones anteriores, ¿podrías señalar los valores almacenados de variables para la iteración externa 3? Explique con sus propias palabras el flujo del programa en estas dos iteraciones:

3 Por cada vez que se ejecuta el bucle exterior

Se ejecuta cuatro el interior

numero de entrada para calcular la potencia a la cuarta:3

1 iteración del bucle interno

2 iteración del bucle interno

3 iteración del bucle interno

4 iteración del bucle interno

La cuarta potencia del numero 3 es 81

### Actividad.

Modifica el programa para controlar el número de iteraciones del while externo. Debe tomar el número de iteraciones de la consola y almacenarlo en una variable int numeroiteracionesexternas. Si este número es 5, el bucle exterior realizaría cinco

iteraciones.

Modifique el programa para leer desde la consola la potencia que desea calcular. Se puede almacenar esta información en una variable con la etiqueta `potenciade`. Por ejemplo, si el número es seis, el programa debe calcular la sexta potencia.

### 5.3.6 La instrucción `do .. while`

A veces es más conveniente probar la condición del bucle al final de un bucle, en lugar de al principio, como se hace en el bucle `while`. El `do..` mientras que la declaración es muy similar a la declaración `while`, excepto que la palabra `"while"`, junto con la condición que se valida, se ha movido hasta el final. Se añade la palabra `"do"` para marcar el comienzo del bucle. A `do..` mientras que la instrucción tiene la forma de una declaración de cuerpo simple.

```
do
    instrucción i;

while ( Expresion booleana );
```

Do while de bloque

```
Do {
    instrucción i;
    instrucción i+1;
    instrucción i +2;

} while ( boolean-expression i );
```

Observa el punto y coma, `';`, al final. Este punto y coma es parte de la declaración, al igual que el punto y coma al final de una declaración o declaración de asignación forma parte de la instrucción. Omitirlo es un error de sintaxis. (De manera más general, cada instrucción en Java termina con un punto y coma o una llave derecha, `'}'`.)

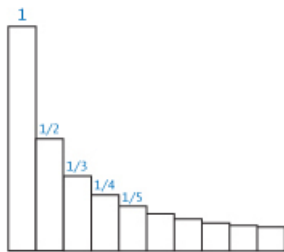
Para ejecutar un bucle `do`, el equipo primero ejecuta el cuerpo del bucle, es decir, las instrucciones dentro del bucle, y luego evalúa la expresión booleana. Si el valor de la expresión es `true`, el equipo vuelve al principio del bucle `do` y repite el proceso; si el valor es `false`, termina el bucle y continúa con la siguiente parte del programa. Dado que la condición no se prueba hasta el final del bucle, el cuerpo de un bucle `do` siempre se ejecuta al menos una vez.

Como se ha demostrado, la principal diferencia entre la instrucción `do-while` y la instrucción

while es la ubicación de la expresión de prueba. Siempre que la ejecución del programa alcance el bucle do while, ejecutará el cuerpo while al menos una vez debido a que la expresión de prueba se evalúa al final. Por el contrario, en la declaración while, la condición se evalúa al principio, por lo tanto, en este caso si la condición del while no se cumple el cuerpo no se ejecutará.

Vamos a dar ejemplo para la instrucción do-while. Intentaremos resolver el problema del algoritmo de los números armónicos.

**DoWhileHarmonicos()** utiliza el mismo paradigma para evaluar la suma finita  $H_n = 1 + 1/2 + 1/3 + \dots + 1/n$ . Estos números, que se conocen como números armónicos, surgen con frecuencia en matemáticas discretas. Los números armónicos son el análogo discreto del logaritmo. También se aproximan al área bajo la curva  $y = 1/x$ . Se puede utilizar este ejemplo como modelo para calcular los valores de otras sumas finitas, calcular una integral. Una serie armónica (también serie de tonos) es la secuencia de frecuencias, tonos musicales o tonos puros en la que cada frecuencia es un múltiplo entero de un fundamental.



n=1   n=2   n=3   n=4   n=5

La serie de harmónicos: 1,  $3/2$ ,  $11/6$ ,  $25/12$ ,  $137/60$ . ....

El objetivo de nuestro programa es mostrar cada miembro de la serie armónica de 1 a n (leeremos n de las consolas) y calcular la suma. Intentaremos en este ejemplo planificar nuestro programa con anticipación y tomar notas. **Hacemos notas de Cornell comentando el programa:**

¿Qué tipo de variable necesitamos?

Contador -> realizar el while para el elemento de la serie 1 al elemento n

Acumulador -> acumular la suma de cada miembro de la serie

elementoSerie -> Para mantener al miembro actual n, el tamaño de la serie. Necesitamos obtener esta información de la consola

Identificar todas las variables del ejemplo con comentarios

¿Qué tipo de expresión de prueba definiremos?

Necesitamos obtener los elementos de la serie de  $i$  a  $n$ . En consecuencia necesitamos hacer  $n$  iteraciones. Podríamos definir la condición como:

$i=1$

Do {

} While ( $i \leq n$ )

¿Cuáles son los objetivos de nuestro programa?

Mostrar cada elemento de la serie armónica.

Calcular la suma de todos ellos

Mostrar la suma al final del bucle.

Identificar todos los objetivos del programa

Definamos cada miembro de la serie

La serie armónica:  $1, \overset{n=1}{3/2}, \overset{n=2}{11/6}, \overset{n=3}{25/12}, \overset{n=4}{137/60}, \overset{n=5}{.....}$

Suma

Para  $N=1$  es 1

Para  $N=2$  es  $1+3/2= 5/2$

Para  $N=3$  es  $1+3/2 + 11/6 = 26/6=13/3$

Para  $N=4$  es  $1+3/2 + 11/6 + 25/12 = 77/12$

Y así sucesivamente:

**DoWhileHarmonics.java**

```

package bucles;

import java.util.Scanner;

public class DoWhileHarmonicos {

    public static void main(String[] args) {
        int i;

        i = 1;

        //variable n, el tamaño de la serie. Necesitamos obtener este valor
        // de la consola

        int n=1;
        double elementoSerie=1;
        double sumatorioSerie=elementoSerie;
        Scanner sc = new Scanner(System.in);

        System.out.println("Cuantos elementos de la serie harmónicos
quieres calcular? "
                           + "\nIntroduce un entero mayor que 1");

        //variable n, el tamaño de la serie. Necesitamos obtener
        //este
        //valor de la consola

        n = sc.nextInt();

        do {

            System.out.println("El termino " + i + " para la serie
harmónicos es "+ elementoSerie);
            i++;
            elementoSerie = elementoSerie + (1/(double) i);
            sumatorioSerie=sumatorioSerie+ elementoSerie;

        } while (i<=n);

        System.out.println("\nEl sumatorio de los " + n + "
elementos de la serie harmónicos es "+ sumatorioSerie);

    }
}

```

**Ejecución del programa**



Cuantos elementos de la serie harmónicos quieres calcular?

Introduce un entero mayor que 1

10

```
El termino 1 para la serie harmónicos es 1.0
El termino 2 para la serie harmónicos es 1.5
El termino 3 para la serie harmónicos es 1.8333333333333333
El termino 4 para la serie harmónicos es 2.0833333333333333
El termino 5 para la serie harmónicos es 2.2833333333333333
El termino 6 para la serie harmónicos es 2.4499999999999997
El termino 7 para la serie harmónicos es 2.5928571428571425
El termino 8 para la serie harmónicos es 2.7178571428571425
El termino 9 para la serie harmónicos es 2.8289682539682537
El termino 10 para la serie harmónicos es 2.9289682539682538
```

El sumatorio de los 10 elementos de la serie harmónicos es 25.23852813852813

### 5.3.7 Bucle For.

Pasamos en esta sección a otro tipo de bucle, la instrucción for. Cualquier bucle for es equivalente a algún bucle while, por lo que el lenguaje no obtiene ningún poder adicional al tener instrucciones para. Pero para un cierto tipo de problema, un bucle for puede ser más fácil de construir y más fácil de leer que el bucle while correspondiente. Es muy posible que en los programas reales, los bucles realmente superen en número a los bucles.

Structure of the form loop

```
for(initialización;condición;incr/decr)
    instrucción 1;
```

O

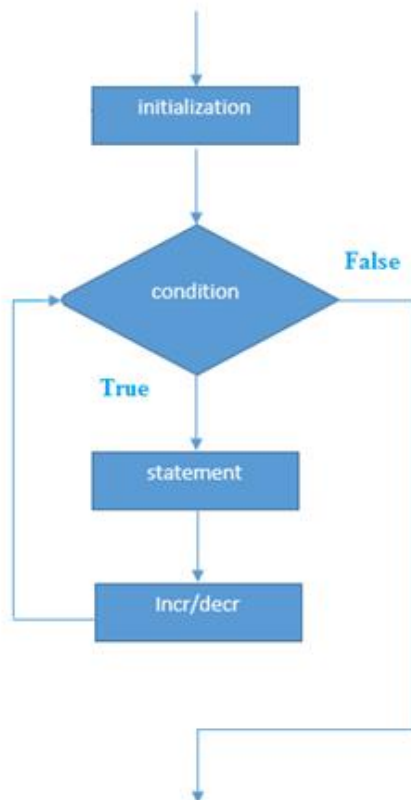
```
for(initialización;condición;incr/decr) {
    instrucción 1;
    instrucción 2;

    ....
    instrucción n;

}
```

Donde:

1. **Inicialización:** Es la instrucción inicial que se ejecuta una vez cuando se inicia el bucle. Aquí, podemos inicializar la variable, o podemos usar una variable ya inicializada. Es una condición opcional.
  2. **Condición:** Es la segunda instrucción que se ejecuta cada vez para probar la condición del bucle. Continúa la ejecución hasta que la condición es falsa. Debe devolver el valor booleano true o false. Es una condición opcional.
  3. **Instrucciones:** La instrucción del bucle se ejecuta cada vez hasta que la segunda instrucción es false.
1. **Incremento/Decremento:** Incrementa o disminuye el valor de la variable. Es una instrucción opcional.



ForExample.java

```
package bucles;
```

```
public class EjemploFor {  
    public static void main(String[] args) {  
        for (int i = 1; i <= 10; i++) {  
            System.out.println(i);  
        }  
    }  
}
```

Inicialización: `int i = 1;`

Condición; `i <= 10`

Incr/Decr: `i++`

El orden de ejecución para esto sería:

Primera Iteración:

1. La inicialización del contador de bucle. `int i = 1;` Solo se ejecuta una vez, ya que el programa intenta ejecutar el for.
2. La condición `i <= 10` se satisface. `1<=10`
3. El cuerpo del for se ejecuta `System.out.println(i);`
4. La expresión de incremento/decremento se ejecuta `i++` `i=2`

Segunda Iteración

1. La condición `i <= 10` se satisface. `2<=10`
2. El cuerpo del for se ejecuta `System.out.println(i);`
3. La expresión de incremento/decremento se ejecuta `i++` `i=3`

....

Decima Iteración

1. La condición `i <= 10` no se satisface. `11<=10`
2. El flujo del programa deja el bucle for

## Practica guiada

Vamos a hacer la tabla de multiplicar para el 5 con el bucle for:

```
for (int j = 1; j <= 10; j++) {  
  
}
```

## Práctica.

Crear una clase ForTablaProducto.

Uso de un anidado para intentar mostrar la tabla de productos para los primeros diez números

Puede utilizar esta estructura para el anidado para:

```
for (int i = 1; i <= 10; i++) {  
    for (int j = 1; j <= 10; j++) {  
  
    }  
}
```

Las ejecuciones del programa deben ser similares a las siguientes:

Tabla de multiplicar para el 1

1X1=1  
1X2=2  
1X3=3  
1X4=4  
1X5=5  
1X6=6  
1X7=7  
1X8=8  
1X9=9  
1X10=10

Tabla de multiplicar para el 2

2X1=2  
2X2=4  
2X3=6  
2X4=8  
2X5=10  
2X6=12  
2X7=14  
2X8=16  
2X9=18  
2X10=20

... .

... .

Tabla de multiplicar para el 10

10X1=10  
 10X2=20  
 10X3=30  
 10X4=40  
 10X5=50  
 10X6=60  
 10X7=70  
 10X8=80  
 10X9=90  
 10X10=100

### 5.3.8 Comparativa entre bucles java

Tabla comparativa resumen de los bucles

COMPARATIVA	BUCLE FOR	BUCLE WHILE	BUCLE DO WHILE
<b>INTRO</b>	El bucle Java for es una instrucción de flujo de control que itera una parte de los programas varias veces.	El bucle Java while es una instrucción de flujo de control que ejecuta una parte de los programas repetidamente sobre la base de una condición booleana dada.	El bucle Java do while es una instrucción de flujo de control que ejecuta una parte de los programas al menos una vez y la ejecución posterior depende de la condición booleana dada.
<b>CUÁNDO USAR</b>	Si el número de iteraciones es fijo, se recomienda usar para el bucle.	Si el número de iteraciones no es fijo, se recomienda usar el bucle while.	Si el número de iteraciones no es fijo y debe tener que ejecutar el bucle al

			menos una vez, se recomienda utilizar el bucle do-while.
<b>SINTAXIS</b>	<pre>for (init; condition     ;incr/decr) {     // código }</pre>	<pre>while (condition) {     // código }while (condition );</pre>	
<b>EJEMPLO</b>	<pre>//bucle for for (int     i=1; i&lt;=10; i++) {     System.out.println     (i); }</pre>	<pre>//bucle while int i=1; while (i&lt;=10) {     System.out.print     ln(i);     i++; }</pre>	<pre>//bucle do-while loop int i=1; do {     System.out.print     ln(i);     i++; }while (i&lt;=10);</pre>
<b>BUCLE INFINITO</b>	<pre>for (;;) {     // código }</pre>	<pre>while (true) {     // código }</pre>	<pre>do {     // código }while (true);</pre>