

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Programación			CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)			

Tema 1 – Introducción a Java.

Definiciones iniciales

Este “es un curso de programación en Java”; pero lo más importante es que es un “**curso de programación**” y por tanto el lenguaje y las herramientas usadas en principio no importan. El objetivo es obtener una mentalidad de programador, de pensar con las mismas estructuras en que se programa un ordenador y en resumen plantear soluciones a problemas mediante estas estructuras que se utilizan en el mundo de la programación. Por supuesto que se necesita un lenguaje y herramientas para programar, pero en el fondo todos los lenguajes (dentro de una categoría) son muy similares y simplemente cambian elementos sintácticos pero no de fondo, ya que lo que buscan los lenguajes es procesar datos de la forma más eficiente y cómoda posible.

Por tanto vamos a empezar a programar y como nuestra herramienta es el Java aprenderemos este lenguaje. Pero ¿qué se puede hacer con Java? Básicamente los siguientes tipos de aplicaciones:

- Aplicaciones de consola y servicios.
- Aplicaciones en entorno gráfico (PC de escritorio o dispositivo móvil).
- Aplicaciones para el mundo Web (en cliente y servidor).

En esta materia nos quedaremos en los dos primero tipos de aplicaciones. Empezaremos por el primer caso en el que fabricaremos pequeñas aplicaciones que funcionarán en una consola del sistema operativo para luego evolucionar y terminar haciendo aplicaciones en un entorno gráfico de ventanas como puede ser GTK+, MS Windows, MacOS y otros.

En este primer tema veremos las bases de la programación, aquellos elementos o ladrillos que son necesarios manejar con absoluta fiabilidad antes de meterse a aprender estructuras más complejas como las clases, colecciones, streams, etc.

En cualquier caso Java, y en general cualquier lenguaje orientado a objetos es un lenguaje cuyo aprendizaje en principio puede parecer un poco extraño pues es necesario hacer un pequeño ejercicio de fe y creerse ciertas cosas que se irán entendiendo a medida que pasen los temas.

Durante los primeros temas empezaremos los programas de forma muy similar, con las líneas:

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

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

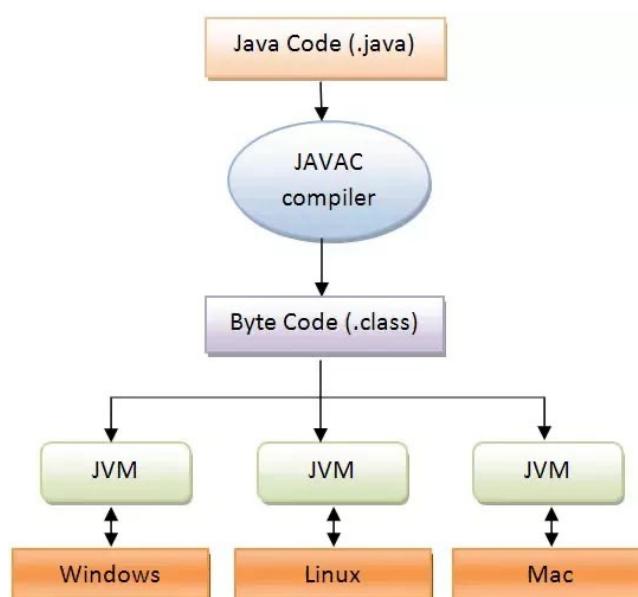
y no entenderemos qué es eso de *public* o de *static* hasta que lleguemos al tema de Orientación a Objetos. ¿Y por qué no empezamos por ese tema? Porque se haría muy tedioso y complicado entrar directamente en esa filosofía que se va a usar luego con total naturalidad.

Podríamos pararnos a continuación a contar un montón de aspectos sobre Java pero entiendo que en otro módulo de este ciclo profundizaremos en los distintos lenguajes de programación y será ahí donde el alumno obtendrá una visión más amplia del Java dentro del mundo de la programación.

Máquina Virtual de Java (JVM) y otros conceptos

Sin entrar demasiado en profundidad hay que entender que Java es un lenguaje que no se traduce directamente al código máquina del ordenador en el que estamos, si no que se traduce al código máquina de una cpu virtual denominada jvm.

La ventaja de esto es que se ejecuta sobre una "máquina software", las principales desventajas es que es necesario que exista dicha máquina (jvm) en el ordenador que queremos ejecutar java y por otro que es algo menos eficiente pues la jvm tiene que traducir cada comando a instrucciones del sistema en el que se está ejecutando.



<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR	Francisco Bellas Aláez (Curro)						

Profundizaremos en estos detalles a lo largo del curso.

Pero antes de entrar en materia conviene conocer **algunos acrónimos y nombres** del mundo de Java para tener en cuenta de qué hablamos en cada momento:

Java SE: Standar Edition. Versión estándar para programar en Java.

Java EE: Enterprise Edition. Enfocado al mundo empresarial. Para trabajo con arquitecturas cliente-servidor entre otros.

JRE: Java Runtime Environment. Es el conjunto de elementos básicos para poder ejecutar una aplicación Java. Al menos se compone de la JVM (Máquina virtual Java) y una serie de paquetes con clases (API). Tanto la JVM como la API deben estar en la misma versión para que no haya inconsistencias. Por supuesto con el JRE sólo se pueden ejecutar aplicaciones, para construirlas es necesario el JDK.

JDK: Java Development Kit. Conjunto de paquetes que permiten crear aplicaciones en Java. Además de incluir el JRE dispone también del compilador (javac), del depurador (jdb) y del creador de documentos(javadoc).

Resumiendo: si sólo quiero ejecutar programas Java me llega el JRE. Si además quiero programar, crear programas en Java necesito el JDK.

Se pueden ver más definiciones de distintos elementos de la arquitectura de Java en el [Apéndice I de este tema](#).

Finalmente indicar que no debe confundirse el lenguaje Java con Javascript, ni siquiera el segundo nació del primero.



COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Instalación del JDK

Se debe tener en cuenta que no llega con tener la JRE pues esta solo permite ejecutar programas hechos en Java, es necesario disponer del JDK que es el conjunto de herramientas que permite programar en Java.

A continuación se presentan los pasos que habría que hacer para preparar un equipo:

Windows

Aunque Java pertenece a Oracle, existe una versión libre de este lenguaje que es la que usaremos (además es la que usa también Android).

En la web:

<https://adoptium.net/es/>

vamos a poder bajar la versión del JDK deseada en el botón **Otras plataformas y versiones**.

Ahora seleccionamos:

Sistema Operativo: **Windows**.

Arquitectura: **x64** (Ojo, si en casa tienes un Windows de 32 bits deberías seleccionar x86).

Versión: **11-LTS**. Esto significa Long Time Service y, por definirlo rápido, viene a significar que de dicha versión se dará soporte a largo plazo. Es, de hecho, la que está usando actualmente en todas las versiones de Android salvo la última (Android 14) que ya usa la versión 17-LTS de Java.

En cualquier caso para empezar a programar apenas veremos las diferencias entre una y otra por lo que ambas son muy válidas y por ahora trabajaremos con java 11 (en algún momento haremos referencia a 17 cuando tengamos más conocimientos).

De las posibilidades de descarga están los JRE y los JDK. Estos últimos son los que debemos descargar para poder programar en Java. Además se recomienda descargar el **msi** que es el archivo instalable.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

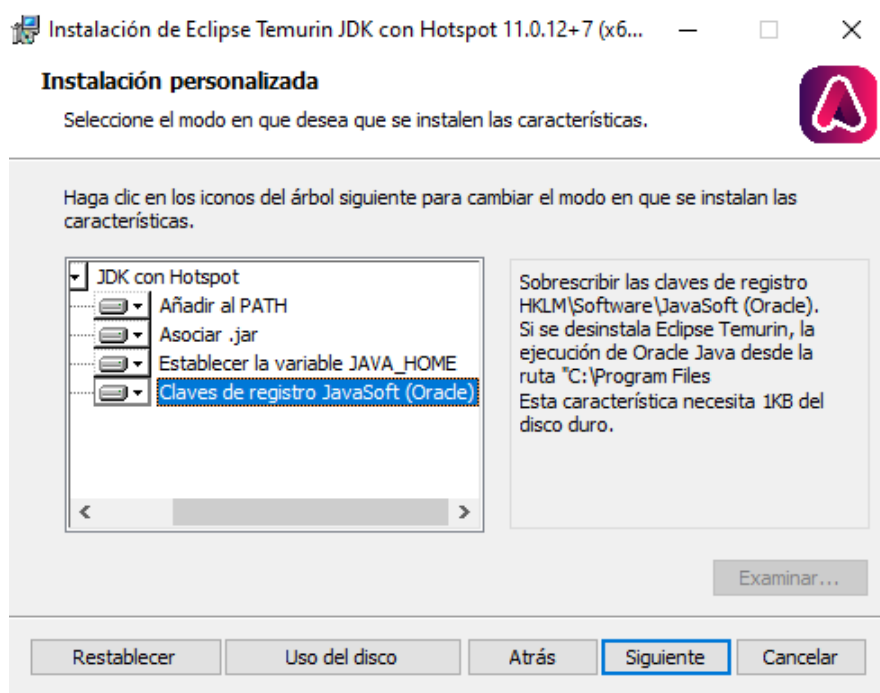
Sistema Operativo	Arquitectura	Tipo de paquete	Version
Windows	x64	JDK	11

jdk-11.0.16.1+1 Temurin 19 de agosto de 2022	Windows	x64	JDK - 175 MB Checksum	.msi
			JDK - 197 MB Checksum	.zip

Nota: si tienes un equipo de 32 bits debes descargar el equivalente pero x86 en vez de x64.

Ejecuta el archivo descargado para que salga el asistente de instalación.

En la siguiente pantalla del asistente de instalación marca todos los campos, pues son necesarios para que Windows y OpenJDK funcionen en conjunto. Entenderás más adelante cómo funcionan las variables de entorno.



COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Para comprobar que todo está bien vamos a probar los dos programa principales de java. Para ello abre una consola y escribe lo siguiente **para ver la versión del JRE** (el sistema para ejecutar programas Java):

```
c:\>java -version
```

Salida:

```
openjdk version "11.0.12" 2021-07-20
OpenJDK Runtime Environment Temurin-11.0.12+7 (build 11.0.12+7)
OpenJDK 64-Bit Server VM Temurin-11.0.12+7 (build 11.0.12+7, mixed mode)
```

Por supuesto los números de versiones no tienen que coincidir exactamente pues los archivos descargados para estos apuntes probablemente sean más antiguos que los que has descargado tú. Sin embargo sí debe ser la versión 11 de java (el 1^{er} número debe ser un 11).

Y luego el que nos permite **compilar programas Java a código máquina de la JVM**:

```
c:\>javac -version
```

Salida:

```
javac 11.0.12
```

Si quieres ver el proceso más desarrollado, en esta Web indican como instalar la versión 14, pero el proceso es el mismo:

<https://joanpaon.wordpress.com/2020/07/17/como-instalar-openjdk-14-0-2-para-windows/>

Linux (Ubuntu, Mint o similar)

Si ya tenemos el JRE instalado debemos comprobar la versión del mismo para instalar el mismo JDK. Para ello:

```
$ java -version
```

la salida será algo similar a esto:

```
openjdk version "1.8.0_131"
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

```
OpenJDK Runtime Environment (build 1.8.0_131-8u131-b11-2ubuntu1.16.04.3-
b11)
OpenJDK 64-Bit Server VM (build 25.131-b11, mixed mode)
```

La primera línea es la versión genérica de Java.

La segunda la versión y "modelo" del JRE. En este caso se puede ver que no es el JRE de Oracle ("Propietario" de Java) si no que es el OpenJDK que es un "Java alternativo" que cumple todas las especificaciones de Java pero es Open Source.

Finalmente la tercera línea es la JVM usada. También existen varias tanto de código abierto como privativo.

Los más probable es que no tengas nada instalado, en ese caso directamente al instalar el JDK ya instalas el JRE.

Por tanto:

```
$ sudo apt-get update
$ sudo apt-get install openjdk-11-jdk
```

Si fuera otra versión habría que sustituir el número correspondiente.

Si la versión que tienes de Linux no tiene estos repositorios (da error lo anterior), antes debes añadirlos. Es decir algo así:

```
$ sudo add-apt-repository ppa:openjdk-r/ppa
$ sudo apt-get update
$ sudo apt-get install openjdk-11-jdk
```

Si por algún motivo no estuviera instalado el JRE, al instalar el JDK ya instalaría el JRE automáticamente. Si se quisiera instalar solo el JRE llegaría con cambiar el paquete a instalar por openjdk-11-jre

Nota: Si en un ordenador hubiera varias versiones de java se puede seleccionar la que se usa por defecto mediante el comando:

```
$ sudo update-alternatives --config java
```

Si lo que quieres cambiar es la versión de javac, simplemente ejecuta lo anterior acabado en **javac** en lugar de **java**

```
$ sudo update-alternatives --config javac
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

MacOS

La instalación en macOS es similar a la de Windows, simplemente se debe seleccionar los archivos ajustados a la versión del sistema operativo que se tenga.

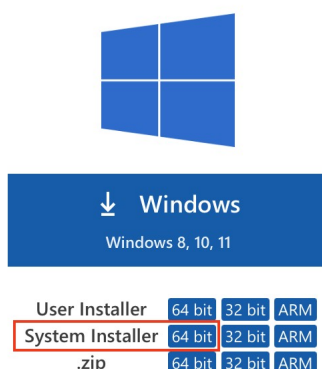
COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Creando, compilando y ejecutando un programa en Java

Una vez que tenemos el equipo preparado vamos a realizar nuestro primer programa en Java. Este simplemente mostrará una frase en la consola.

Para ello se requiere un editor de texto cualquiera (**iNo procesador de texto!**), de hecho valdría el bloq de notas de Windows, pero es excesivamente simple y no trae ninguna ayuda añadida. Por eso vamos a usar en su lugar el Visual Studio Code. Descárgalo e instálalo (con opciones por defecto) de la web:

<https://code.visualstudio.com/download>



Selecciona la versión System Installer que lo instala para todos los usuarios del ordenador. También coge versión de 64 bits (o 32 si tienes un ordenador con dicha arquitectura).

Si en un momento dado estuvieras en un ordenador sin permisos y te hiciera falta el vscode siempre puedes descargar e instalar el User Installer.

Abre ahora el vscode, crea un nuevo archivo (File → New Text File) y escribe el siguiente código.

```
public class Hola {
    public static void main (String[] args) {
        System.out.println("Welcome to the Java World");
    }
}
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Guarda el programa con el nombre (¡Exactamente igual!)

Hola.java

ya que es imprescindible que el nombre del archivo coincida con el nombre de la clase (*class* indicará por ahora el nombre del programa aunque ya se verá en un futuro tema la utilidad de dicho concepto), si no nos dará un error al compilar. **¡Incluso el uso de mayúsculas es necesario cumplirlo!** Este es un requisito de Java. La clase pública (en la mayoría de los casos la única clase en el archivo) tendrá el mismo nombre que el archivo.

Por defecto vamos a guardar este archivo y otros de teoría en una carpeta denominada ***programas*** dentro de Documentos.

Abre ahora un terminal y vete al directorio donde hayas grabado el código. Suponiendo que está en el directorio *programas* usa el comando ***cd*** (change directory) de la siguiente manera.

```
C:\Users\curro>cd Documents\programas
```

Por supuesto la carpeta en la que estás llevará tú nombre de usuario, no el mío (curro).

A partir de ahora en vez de todo el prompt (C:\Users...) , escribiré solo como inicio de consola el símbolo mayor que (>) para simplificar salvo en casos que sea aclaratorio.

Si ahora escribes el siguiente comando:

```
>dir
```

verás los archivos que hay en la carpeta *programas* con información de tamaño fecha y hora.

Tiene que aparecer, por supuesto, Hola.java.

Nota: En el caso de **Linux** el comando para cambiar de directorio también es ***cd*** y el de listar los archivos del directorio es ***ls***.

A continuación vamos a compilar el código Java a un lenguaje (bytecodes) que comprenda la máquina virtual (JVM). Para ello ejecutamos:

```
> javac Hola.java
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Si hay algún error saldrá ahora algún tipo de mensaje. Los errores más típicos son el escribir una letra minúscula en mayúscula o viceversa, olvidarse de un punto y coma o de una llave o escribir mal un comando. Si no ha habido problema ejecutemos el comando `ls` y se puede ver que ha aparecido un nuevo archivo denominado *Hola.class*. Ese es el programa que puede ejecutar la JVM. Para ello:

```
> java Hola
```

Se puede observar que no es necesario escribir el `.class` (de hecho no funciona si se hace). Como resultado aparece la frase que hemos puesto entre comillas en el comando **`System.out.println()`** de Java. Efectivamente, mediante dicho comando, sacaremos mensajes de distinto tipo a la consola.

Truco: Cuando tengas nombres de archivos más largos, para evitar escribirlos enteros en la consola puedes usar la tecla Tab. Por ejemplo, empiezas a escribir un comando y el principio del nombre del archivo (`javac H`) y en ese momento al darle al Tab te autocompleta con el primer nombre que empieza por H (si le das más veces buscaría otros si los hubiera).

En general compilar implica traducir el código fuente (Java, Pascal, C, ...) al código máquina (binario) que entiende la máquina en la que se ejecutará. En el caso de Java esa máquina es virtual, es decir, es un emulador de una máquina pero también tiene su propio juego de instrucciones en binario. Se profundizará en esto en otro módulo.

Desde Java 11, mediante el comando `java` se puede hacer la compilación y ejecución en un único paso. Borra el archivo *Hola.class* y prueba a escribir en la consola:

```
> java Hola.java
```

Si quieres borrar desde consola usa el comando:

```
> del Hola.class
```

Verás que hace en un solo comando los dos anteriores. También, si ejecutas **`dir`** puedes comprobar que no ha creado ningún archivo **`.class`**. Esto es porque este método hace la compilación en memoria y luego ejecuta. Es válido solo cuando tenemos un archivo simple por lo que no va a poder usarse en trabajos más complejos.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Truco: Para no tener que repetir comandos previos en la consola puedes darle al cursor para arriba, esto recupera el comando anterior. Si le das más veces recupera los anteriores por orden inverso al que se escribieron. Dale al cursos abajo para volver a los más recientes.

Más adelante iremos automatizando estas tareas con plugins (extensiones) más potentes que instalaremos en el Visual Studio Code y lo iremos convirtiendo en un entornos de desarrollo completo (como podría ser Netbeans, Eclipse, o IntelliJ, que también nos valdrían)

Nota sobre codificación: Por cierto, si te surgen problemas con la consola y las tildes, debes tener el mismo sistema de codificación en el vscode y en la consola CMD. Para ello en la consola ejecutas el comando

```
>chcp 65001
```

para cambiar la consola a UTF-8 (el actual es 850). También se puede cambiar el vscode en barra de estado inferior donde pone UTF8.

Para cambiarlo de forma permanente mira esta entrada:

<https://stackoverflow.com/a/56091362>

Los sistemas de codificación son tablas que indican como representan las letras (caracteres alfanuméricos en general) los ordenadores. Iremos hablando de esto a lo largo del curso.

Veamos ahora un ejemplo de lo que sería una aplicación gráfica (muy simple) en Java. En un nuevo archivo escribe el siguiente código:

```
import javax.swing.*;

public class HolaWin{
    public static void main(String[] args){
        JOptionPane.showMessageDialog(null,
            "Welcome to the Java World",
            "Graphic Mode",
            JOptionPane.INFORMATION_MESSAGE);
    }
}
```

En modo gráfico se complican las cosas. En futuros temas entenderemos el significado de todo lo aquí expuesto.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR	Francisco Bellas Aláez (Curro)						

Conceptos básicos de programación y estructura de un programa en Java

Programa (o Algoritmo): Serie de operaciones detalladas y no ambiguas, a ejecutar paso a paso, y que conducen a la resolución de un problema.

Hay diversas formas de realizar un algoritmo. Simplemente hay que definirlo de forma no ambigua.

Dato: Información que procesa un algoritmo (es decir, el programa).

Tipo de dato: Tipo de información que se maneja. Es un concepto muy importante y que hay que ir teniendo muy claro al desarrollar programas.

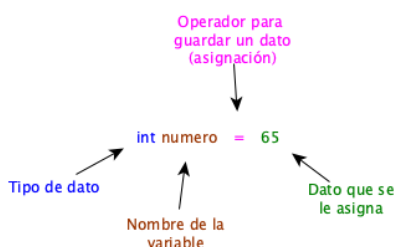
Cuando se trabaja con un dato, hay que saber si dicho dato es un número entero, un número real, un carácter, una frase (una lista de caracteres), un valor lógico, etc... ya que el ordenador todo lo que guarda son 0s y 1s y por lo tanto es necesario indicar el significado de esos valores binarios.

Ejemplo:

01000001 si lo tratamos como tipo de dato **número entero** es el **65** en decimal.

Pero si lo tratamos como tipo de dato **carácter**, según tabla ASCII o Unicode, simboliza la letra **'A'**.

Variable: Es un elemento que contiene un dato y su valor puede variar a lo largo del programa. Internamente en el ordenador es una zona de memoria reservada donde se guarda un dato de determinado tipo y cuyo valor cambia a lo largo de la ejecución.



Atributos:

- **Nombre o identificador:** Lo que la designa
- **Tipo de dato:** El tipo de información que puede manejar esa variable. Es muy importante porque en memoria todo son

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

números y dependiendo del tipo de dato esos números se interpretan de distinta forma.

Se utilizarán para recoger datos, para manejar datos internos variables y para devolver datos.

En principio las variables se almacenan en una zona de memoria reservada para el programa denominada *heap* o *montículo*.

Expresiones: Combinaciones de variables, valores, símbolos de operación, paréntesis y nombres de funciones especiales.

Ejemplo:

```
a = m*(n+5)-p
```

Estructura

Veamos la **estructura de un programa** en Java con un nuevo ejemplo ligeramente más amplio que el primero.

```
import java.util.Scanner;

public class Hola2 {
    public static void main(String args[]) {
        // Declaración de variables
        String nombre;
        int edad;
        Scanner sc = new Scanner(System.in);

        // Código
        System.out.print("Dime tu nombre: ");
        nombre = sc.nextLine();
        System.out.println("Ahora tu edad");
        edad = sc.nextInt();
        System.out.println(nombre + ", Welcome to the Java World");
        System.out.println("Nos vemos, ser humano de " + edad + " años");
    }
}
```

Antes de explicar las distintas líneas veamos unos aspectos generales:

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Programación			CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)			

- Cada sentencia acaba en ; ya que es el indicador de fin de sentencia. Puedo escribir una sentencia en varias líneas como hicimos en el ejemplo de GUI ya que lo que marca el final es el ;
- Las llaves {} marcan inicio y fin de bloque. Cuando hay una serie de comandos o sentencias que queremos meter dentro de un bloque los meteremos entre estos símbolos.
- La indentación, tabulación o sangrado es esencial en el mundo de la programación. Sirve para entender que comandos están dentro de que bloques de forma clara. En algunos lenguajes si no se indenta el programa no funciona.

Los elementos que nos encontramos a lo largo de cualquier programa son:

- **Palabras reservadas:** Son palabras que pertenecen al lenguaje Java y por tanto no pueden ser usadas por nosotros para nombrar variables. La lista completa es:

```

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

```

- **Identificadores:** Son palabras que crea el programador para dar diferentes nombres a elementos como las variables, constantes, las clases, funciones, etc...

Hay algunas limitaciones para poner nombres a las variables:

- Se usan sólo letras, números, y los símbolos _ y \$
- No puede, por tanto, contener espacios.
- No puede empezar por número.
- Se distingue entre mayúsculas y minúsculas .
- No se pueden usar palabras reservadas (class, import, etc...)

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Actividad: Indica cuales de estos nombres son válidos para identificadores y cuales no:

nombre, \$numero, dato1, 1dato, media total, kilometro/hora, kiloBitsPorSegundo, x8c, mi_dato.

De cara a la claridad de programación, vamos además a usar algunas reglas en cuanto a la creación de nombres de identificadores: Las clases siempre empezarán por mayúsculas y las variables por minúsculas. Si una clase o variable es una palabra compuesta, las siguientes palabras se escribirán comenzándolas en mayúscula: Clase **SistemaOperativo**, variable **datoTransferido**.

Esta forma de escribir es un estándar denominado **CamelCase** (en el caso particular de Java es lowerCamelCase).

- **Comentarios:** Son líneas de código que no afecta al funcionamiento del programa. No se compilan. Por tanto son usadas para aclarar ciertas partes del código que tienen un funcionamiento especial o para documentar de forma técnica el programa. Existen tres tipos de comentarios en java:
 - Comentario de línea: Comienza donde se escriba `//` hasta el fin de la línea.
 - Comentario multilínea: Comienza con `/*` y acaba con `*/` en la misma línea o en otra.
 - Comentario de documentación: el SDK de Java incorpora un programa denominado **javadoc** que facilita la creación de documentación técnica y cuyo funcionamiento veremos en otra asignatura. Los comentarios detectados por javadoc comienzan por `/**` y acaban por `*/`.
- **Literales:** Datos explícitos como `3`, `"hola"`, `'$'` ó `-78.23`. Son de un tipo de dato determinado (real, entero, cadena de caracteres,...).

Teniendo en cuenta estos puntos anteriores vamos a explicar de forma aproximada el funcionamiento del programa anterior:

Importación

Lo primero que encontramos en un programa en Java son las líneas de importación. En estas líneas se indican los paquetes (**packages**) donde existen clases de Java

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

que podemos necesitar y ya están programados. Una clase podemos verla como un contenedor. Por tanto todos los comandos de Java (salvo las instrucciones más básicas) aparecerán dentro de clases que a su vez se guardan en paquetes.

En este caso se importa la clase *Scanner* que vamos a usar para pedir datos al usuario. Esa clase se encuentra en el paquete *util* que a su vez está en el paquete genérico *java*. Ese es el significado de `java.util.Scanner`.

Puede haber varias líneas de importación para usar varias clases, pero en el caso de usar diversas clases que están en un paquete puede usarse el comodín `*`. Es decir, si en un programa tenemos que importar clases de esta manera:

```
import java.io.File;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.IOException;
```

podría ser más cómodo escribirlo en una única línea así:

```
import java.io.*;
```

Ese `*` indica "*todas las clases del paquete io que está en java*". Los puristas dicen que es mejor escribir sólo las clases que se van a usar por claridad, pero realmente no se pierde eficiencia.

Actividad: Prueba a quitar la línea `import` y comprueba el error que se produce.

La clase principal.

```
public class Hola2 {
}
```

Tras la zona de importación en el archivo viene la clase principal. Como ya se comentó una clase la veremos por el momento como un contenedor de elementos y ya veremos definiciones más precisas y correctas. Nos quedamos con la idea de que todo el código que vamos a meter hay que hacerlo dentro de la clase.

Una clase comienza por la palabra *class* y a continuación se le da el nombre deseado. Ya comentamos también que por claridad y estandarización vamos a nombrar las clases comenzando por letra mayúscula.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Programación			CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)			

Todo lo que va dentro de la clase se mete entre llaves.

En un archivo se pueden meter varias clases pero por ahora meteremos sólo una que sirva para contener nuestros programas. Además el archivo ha de tener el mismo nombre que dicha clase (conservando mayúsculas y minúsculas) y precediendo a la palabra *class* irá el modificador *public* que indica que "cualquiera" puede acceder a la clase. Entenderemos mejor esto último en el tema de orientación a objetos.

Actividad: Si no lo has probado con algún ejemplo anterior prueba a nombrar la clase de forma distinta al archivo para ver el error que da. Prueba también a cambiar simplemente mayúsculas o minúsculas y haz la misma comprobación. Finalmente prueba a quitar alguna llave.

Programa principal (función principal).

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

Un apartado denominado por la palabra reservada *main* (principal) y precedido por las palabras reservadas *public*, *static* y *void* es el bloque donde meteremos las sentencias ejecutables que conformarán nuestro programa al menos al principio.

A este apartado se le denominará también función o método principal o función o método main.

En general una función o método es una zona del programa que contiene comandos ejecutables y que puede ser invocada desde otras partes del programa o sistema. Crearemos más adelante otros métodos distintos del principal.

El elemento especificado entre paréntesis (*String[] args* o *String args[]*, ambas formas permitidas) no lo tendremos en cuenta al menos por el momento. Simplemente indicar que es útil para recoger información externa al programa en el momento de la ejecución.

Cuando se construye un programa Java hay que indicar las variables que se van a usar y el tipo de dato que son. Realmente se puede hacer en cualquier parte de la función pero por claridad conviene separar la zona de declaración de variables (se puede poner al principio) del resto del código.

Vamos a ver dos grandes bloques de variables, por un lado las básicas que simplemente hay que declarar su nombre y el tipo de dato que contendrán:

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

```
String nombre; //Variable que guarda una cadena de caracteres
int edad; // Variable que guarda un valor entero
```

y los objetos o variables referenciadas que, sin entrar en detalle, son “variables” para las cuales hay que reservar memoria y por tanto se declaran e inicializan en la misma línea.

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

aunque podría hacerse en líneas distintas (una de declaración y otra de reserva de memoria):

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

En este caso se inicializa la variable *sc* que es el objeto que representa al teclado (*System.in*) y va a servir para pedirle datos al usuario.

La diferencia entre las simples y las referenciadas es que las simples básicamente manejan un valor único y las referenciadas son complejas en cuanto a que pueden guardar varios valores e incluso contener código ejecutable asociado a la variable (son los denominados objetos).

Finalmente describimos brevemente cada línea de código:

```
System.out.print("Dime tu nombre: ");
```

Muestra una frase en pantalla y el cursor se queda justo a continuación.

```
nombre = sc.nextLine();
```

Se queda esperando a que el usuario introduzca una cadena de caracteres y una vez que el usuario pulsa enter, se guarda el dato en la variable *nombre*.

```
System.out.println("Ahora tu edad");
```

Muestra una frase en pantalla y el cursor pasa a la línea siguiente (diferencia *print/println*).

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

Se queda esperando a que el usuario introduzca un número entero y pulse Enter. En ese momento guarda el dato numérico en *edad* (Si se mete un dato no entero daría un error).

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

```
System.out.println(nombre + ", Welcome to the Java World");
```

Muestra en pantalla el contenido de la variable nombre seguido del texto

, Welcome to de Java World

sin las comillas. La **operación +** con cadenas de caracteres se denomina **concatenación** (coloca una cadena detrás de otra). Además una vez que termina de escribir pasa a la línea siguiente.

```
System.out.println("Nos vemos, ser humano de " + edad + " años");
```

Construye una frase con concatenación y la muestra en pantalla.

Nota: si leéis algún documento en la que se explique una versión de Java anterior a la 1.5 la introducción de datos se realizaba con otro objeto de la siguiente manera:

```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

```
nombre=br.readLine();
```

Además había que hacer el denominado control de excepciones que veremos más adelante pero que aumenta la complejidad de la entrada de datos.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Tipos de Datos primitivos en Java

En este apartado veremos qué tipos de datos podemos manejar y, sobre todo, guardar en las variables que usamos en nuestros programas.

Como se comentó en un apartado anterior, existen dos grandes grupos de tipos de datos, los sencillos o primitivos que serán la mayoría de los que veamos en este apartado, y los referenciados, compuestos u objetos, que necesitan más memoria y se reserva de una forma especial, no llega sólo la declaración.

Enteros

byte: Entero de 1 byte en el rango de -128 a +127.

short: Enteros cortos de 2 bytes en el rango de -32768 a 32767.

int: Enteros de 4 bytes en el rango de -2147483648 a 2147483647

long: Enteros largos de 8 bytes en el rango de -9223372036854775808 a +9223372036854775807.

En general los valores numéricos se pueden expresar en decimal, hexadecimal, octal o binario. Para ello se usan lo siguientes códigos:

Decimal: 30

Octal: 036 (se antepone un 0)

Hexadecimal: 0x1E ó 0x1e (se antepone 0x)

Binario: 0b1110 (se antepone 0b **sólo válido desde Java 7**)

Si se desea forzar un número como entero largo se le puede poner una L al final.

Reales

float: Reales de 4 bytes según el estándar IEEE 754.

Cubre el rango de 1.40129846432481707e-45 a 3.40282346638528860e+38 tanto positivos como negativos.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Los literales con decimales en java por defecto son double, esto significa que al realizar esta operación:

```
float b=2.3;
```

obtenemos error de compilación. Para especificar que un número es float se le puede poner una F ó f al final.

```
float b=2.3F; // esto ya no da error
```

También debes tener en cuenta que los reales tienen una precisión finita, por lo que no pueden guardar las cifras decimales que uno quiera. Esto puede llevar a situaciones curiosas como esta:

```
System.out.println(1.1f+1.2f);
```

¿Cuál es el resultado? ¿Por qué sucede esto?

double: Reales de 8 bytes según el estándar IEEE 754.

Cubre el rango de $4.94065645841246544e-324$ a $1.79769313486231570e+308$ tanto positivos como negativos.

Si se deseara usar decimales con una gran precisión, no debe usarse ni *double* ni *float* si no la clase *java.math.BigDecimal*.

*Nota: desde **Java 7** se permite la separación de cifras mediante `_`. Por ejemplo se podría poner `666_123_456` como número de móvil por claridad y el programa entenderá `666123456`.*

Lógicos

boolean: Valor lógico que implica verdadero o falso, por lo que toma los valores *false* o *true*. Teóricamente ocupa 1-bit pero no está especificado y puede ser dependiente de la máquina virtual. Se debe tener cuidado no usar como valores *True* y *False* o *TRUE* y *FALSE* que también existen pero no son tipo *boolean*.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM	
	MÓDULO	Programación			CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:	DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)			

Alfanuméricos

char: Caracter unicode de 2 bytes que cubre de 0 al 65535 o expresados en modo caracteres de '\u0000' a '\uffff' el código unicode está en hexadecimal. Ten en cuenta que solo se verán los códigos unicode que estén instalados.

En general un char suele contener algún símbolo que aparece en nuestro teclado y la representación más simple es dicho símbolo (letra, número, ...) entre comilla simple: 'a' , 'A', '7', '\$', '@', ...

Se debe tener en cuenta que las minúsculas y mayúsculas son caracteres distintos ya que les corresponde distinto código unicode. Por ejemplo 'a' tiene el código 97 y 'A' el 65.

También hay que tener cuidado con los dígitos ya que el '7' no tiene código 7 si no que tiene código 55. Y además en un *char* sólo cabe un dígito. Por ejemplo '71' sería una incorrección.+

Además existen "secuencias de escape" que permite introducir ciertos caracteres que tienen un significado particular. Son los siguientes:

```
\t    tabulador
\n    linea siguiente
\"    comilla doble, "
\'    comilla simple, '
\\    backslash, \
```

String: A pesar de ponerlo aquí no es un tipo primitivo (por eso su primera letra va con mayúscula, porque es una clase) y permite representar cadenas de caracteres (textos) de cualquier longitud. En el caso que deseemos concatenar varias frases se puede usar el operador de concatenación +.

Actividad: Veamos como se puede mezclar los caracteres de escape en un String. ¿Qué sale en pantalla si se escribe el siguiente String?

```
"Letra: \"\u0041\" \nNúmero:\t\u0031"
```

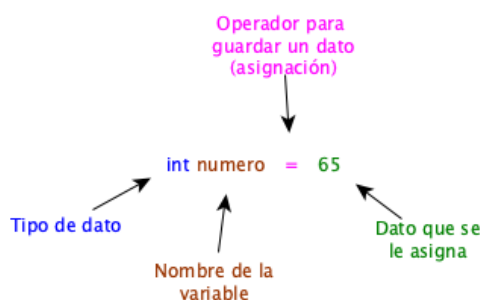
Existe en Java los equivalentes a los tipos primitivos (int, float, ...) pero en formato clase, es decir, su nombre empieza por mayúscula (Byte, Float, Double,...) y tienen otras características. Son las llamadas clases wrapper y por el momento no las utilizaremos por lo que no debes confundirte al escribir los nombres.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Variables y constantes definidas por el programador.

Vimos ya que una variable es un identificador que da nombre a una zona de memoria donde se guarda un dato de determinado tipo. Precisamente como el nombre de las variables las da el programador antes de usar una variable hay que declararla. Esto implica que se le da un nombre, que se dice el tipo que se va a guardar en la misma y que el compilador va a reservar memoria para un dato de ese tipo. La declaración tiene este esquema:

```
Tipo_de_dato nombre_variable1;
```



Se pueden declarar varias variables del mismo tipo en la misma línea. Así estos ejemplos son válidos:

```
int a, num;
String nombre, direccion;
float temperatura;
```

Una variable sólo se declara una vez, a partir de ese momento se puede usar en el programa. La operación más básica con una variable es la asignación que consiste en guardar un dato en la variable. Se usa el símbolo = para realizar dicha operación.

```
a=0b111101;
nombre="Curro";
direccion="c/Príncipe 17";
temperatura=12.5F;
System.out.println(a);
System.out.println("Hola "+nombre+", en "+direccion+
    " hay una temperatura de "+temperatura+"°");
```

Actividad: Ejecuta el programa anterior y comprueba que lo entiendes todo.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM				
	MÓDULO	Programación					CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:			
	AUTOR		Francisco Bellas Aláez (Curro)					

Cuando se declara una variable es posible **darle un valor al mismo tiempo**. Es decir, estas líneas son válidas:

```
int a = 23, num = 0;
String nombre = "Curro";
char espacio = ' ';
```

Es posible obtener las denominadas **constantes** definidas por el programador. Son similares a las variables pero su valor no puede cambiar a lo largo del programa. Se declaran y se inicializan al mismo tiempo anteponiendo la palabra reservada **final**.

```
final char espacio='\u0020';
final double pi=3.14159265;
```

Está claro que no se le puede dar otros valores en otras partes del programa.

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Operaciones aritméticas y de bit

Una operación es la realización de algún cálculo a partir de variables y constantes. El resultado de dicha operación tiene que tener algún destino: almacenarlo en una variable o mostrarlo por pantalla son dos ejemplos de lo que se puede hacer con el resultado de una expresión.

Para realizar operaciones con números disponemos de los siguientes operadores:

```
+ Suma
- Resta
* Multiplicación
/ División (cociente si los operandos son enteros)
% Resto de la división
```

De esta forma se puede hacer una operación como esta:

```
(12*n+dato)/3;
```

Pero en Java no puede existir esta línea porque sí, ya que su resultado se perdería. No es una sentencia válida. Veamos como ejemplo las dos posibilidades comentadas antes.

```
res=(12*n+dato)/3; //Almacenar resultado en la variable res
```

ó

```
System.out.println((12*n+dato)/3); //Mostrar el resultado por pantalla
```

El primer caso interesa cuando el resultado va a usarse para más cosas.

Lo que es muy importante resaltar es que una expresión **no es una ecuación**. No hay nada que despejar. Cuando aparece una asignación esta no es el igual de matemáticas si no que implica un movimiento de datos. Se opera con lo que hay a la derecha de la asignación (variables y constantes **de valores ya conocidos**) y cuando el cálculo está hecho se vuelca en la variable de la izquierda.

Así nunca vamos a tener expresiones como estas pues no tienen sentido:

```
res+5=n-23;
10=2*n+1;
```

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Actividad: Comprobar los errores que da el compilador cuando se usan incorrectamente las expresiones.


También tenemos los siguientes operadores que actúan **a nivel de bit** tal cual se conoce la lógica binaria (Algebra de Boole):

```
& and
| or
^ xor
~ not
a >> b desplaza a b bits a la derecha
a << b desplaza a b bits a la izquierda
```

Una operación válida sería:

```
resultado = 0x1F5 & 0x00F;
```

Actividad: ¿Qué valor se guardaría en resultado?

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Programación interactiva (entrada/salida de datos)

Resumamos brevemente los comandos vistos para mostrar datos y pedir datos al usuario.

Salida

Para mostrar datos usaremos habitualmente

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

La diferencia es que el primero introduce al final un retorno de carro, cosa que no hace el segundo.

Para mostrar varios datos simplemente se concatenan mediante el operador +.

```
System.out.println("Valores de Pi");
System.out.println("-----\n");
System.out.print("El numero Pi es " + Math.PI);
System.out.print(" y su doble es " + 2 * Math.PI);
```


Nota: Se debe tener en cuenta que el retorno de carro se hace de forma distinta en distintos sistemas operativos. En el caso de Windows se usan dos caracteres: CR (Carriage Return: 0x0D) y LF (Line Feed: 0x0A), sin embargo en sistemas Unix se usa solo LF.

Esto puede llevar a que aparezcan distintas formas los textos y hay que tener cuidado. También puede depender de los comandos. Por ejemplo print y println detectan el sistema operativo y aunque estemos usando \n que sería CR, en un Windows automáticamente mete el LF, pero otros comandos no y es necesario formar un \n\r para que en Windows salga bien.

Puedes probar esta línea como curiosidad a ver si sabes lo que pasa al ejecutarla:

```
System.out.println("Uno \r dos \n Tres \n\r cuatro");
```

Veremos formas de detectar en que sistema operativo estamos.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Entrada

Para pedir datos al usuario usaremos la clase `Scanner` como ya hemos visto. Para ello hay que realizar la importación:

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

Luego declarar la variable tipo *Scanner* que representa al teclado (denominado en Java *System.in*).

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

A partir de esta variable (realmente es un objeto) tenemos varios comandos dependiendo del dato que queramos leer. Algunos de ellos son:

```
nextInt()
nextLong()
nextDouble()
nextLine()
```

Cada uno se queda esperando a que el usuario meta un dato e intenta convertirlo al tipo especificado. En el caso de *nextLine()* se obtiene un `String`.


Esto puede provocar un problema. Prueba el código siguiente que es una variante de un ejemplo previo:

```
import java.util.Scanner;

public class Hola2 {

    public static void main(String args[]) {
        // Declaración de variables
        String nombre;
        int edad;
        Scanner sc = new Scanner(System.in);

        // Código
        System.out.println("Dime tu edad");
        edad = sc.nextInt();
        System.out.print("Ahora tu nombre: ");
        nombre = sc.nextLine();
        System.out.println(nombre + ", Welcome to the Java World");
        System.out.println("Nos vemos, ser humano de " + edad + " años");
    }
}
```

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Comprobarás que no deja meter el nombre. Esto se debe a que el `nextInt()` "ha recogido" el numero pero no el retorno de carro que pulsó el usuario. Esto provoca que el `nextLine()` recoja dicho retorno de carro previo y ya no espere a que el usuario meta algo.

Solución: Leer el retorno de carro después del número.

```
System.out.println("Dime tu edad");
edad = sc.nextInt();
sc.nextLine();
System.out.print("Ahora tu nombre: ");
nombre = sc.nextLine();
System.out.println(nombre + ", Welcome to the Java World");
System.out.println("Nos vemos, ser humano de " + edad + " años");
```

Otra posibilidad es disponer de dos Scanner (`sc1` y `sc2`). Uno lo usaríamos solo para `nextLine()` y el otro para `nextInt` y `nextDouble`.

Puedes ver esta y otras formas de realizar entrada de datos en el siguiente enlace:


<http://www.codejava.net/java-se/file-io/3-ways-for-reading-input-from-the-user-in-the-console>

Nota: Si ya tienes instaladas las extensiones para trabajo en Java con `vscode`, observarás que subraya `sc` como si hubiera un "error". En realidad es un aviso de que podría haber un problema, pero está más enfocado al uso de archivos (El Scanner también se usa para leer archivos del disco duro). Para evitar que aparezca, simplemente añade antes de la clase la siguiente línea:

```
@SuppressWarnings({ "resource" })
```

Es lo que se denomina una anotación en Java. Las veremos más adelante.

Ejercicios : Realiza los ejercicios del Tema 1 del Boletín 1.

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Apéndice I: Otras definiciones

Java ME: Micro edition (dispositivos móviles). Reducido y muy optimizado para equipos con bajos recursos.

Hay otros: Java Card (tarjetas tipo SIM y otros dispositivos pequeños), JavaTV (Video digital), etc...

Se puede ver toda la familia en:

<http://www.oracle.com/technetwork/java/javase/overview/index.html>

Servlets: Componentes de un servidor para servir peticiones (en Java EE)

JSP: Java Server Pages: Tecnología similar a los servlets pero más cómoda a la hora de generar páginas web dinámicamente del lado del servidor. Viene incluido también en el Java EE y requiere, lógicamente, de un servidor capaz de gestionar las JSP. Es una alternativa al ASP o al PHP.

MIDlet: Aplicación similar al applet pero realizada con el Java ME.

JavaBean: Componente Java.

EJB: Enterprise JavaBean. API de componentes para la construcción de aplicaciones empresariales del lado del servidor incluido en Java EE.

Java Web Start: Tecnología que permite a una aplicación en cuanto se ejecuta a comprobar su versión y si no es la más reciente, se conecta con un servidor, se baja la más reciente y se ejecuta en local de forma automática.

SWT: Standart Widgets Toolkit. Sistema gráfico que busca la eficacia y mejora de Swing pero a su vez intenta que en cada plataforma se vean las aplicaciones como nativas. Esto provoca que las SWT son distintas para cada plataforma. Además SWT no pertenece a Oracle, está desarrollado por el grupo de Eclipse y lleva un ritmo distintos y además funciona de forma más o menos eficiente dependiendo de la plataforma.

JIT: Just In Time compiler. Para compilar a código máquina nativo y que la aplicación sea más eficiente.

JVM: Máquina virtual de Java. Programa que emula un procesador virtual que hace que el lenguaje Java sea multiplataforma. Se profundizará sobre esto en otro módulo del ciclo.

<div>COLEXIO</div> <div>VIVAS</div> <div>S.L.</div>	RAMA:	Informática	CICLO:	DAM			
	MÓDULO	Programación				CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:		
	AUTOR	Francisco Bellas Aláez (Curro)					


Applets: Aplicaciones incrustadas en clientes (navegadores) Web. Podrían incrustarse en otras aplicaciones.

AWT: Abstract Window Toolkit. El primer paquete de componentes gráficos para realizar aplicaciones en Java. Hoy se usa para tareas de bajo nivel de otras clases gráficas. Algunos programadores la siguen usando porque produce que las aplicaciones tengan el aspecto gráfico del sistema nativo. Pero sigue siendo muy limitado pues dispone de aquellos elementos comunes en todos los sistemas operativos. Además no funciona bien del todo.

Swing: El sistema actual que mediante Java 2D realiza sus propios dibujos (en lugar de hacerlo el SO con el awt). Al ser independiente del SO, las aplicaciones tienen el mismo aspecto independientemente del equipo en el que se ejecuten. Se supone que se irá sustituyendo por JavaFX que es el nuevo sistema gráfico.

Ficheros JAR: Es un fichero ZIP que contiene una serie de elementos que conforman una aplicación: imágenes, clases java, archivos de audio, etc... De esta forma se pueden tener toda una aplicación empaquetada aunque conste de muchos archivos independientes.

jshell: consola de snippets (comandos) de java, para pruebas sencillas.
<https://www.adictosaltrabajo.com/2016/03/23/jshell-una-consola-repl-como-novedad-en-java-9/>

	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Apéndice II: Variables de entornos

Java usa diversas variables de entorno para entenderse correctamente con el sistema operativo. Las 3 principales para que esté bien configurado son:

PATH: Esta variable tiene una lista de los directorios de donde se ha de buscar un comando cuando se ejecuta si no está en el propio directorio de ejecución. Para ver la configuración actual simplemente en una consola escribe el comando **path**.

Para configurarla sigue estos pasos: <https://www.abrirllave.com/java/configurar-la-variable-de-entorno-path.php>

CLASSPATH: Variable opcional que permite, si está configurada, que Java busque en los directorios de dicha variable posibles archivos .class para ejecución o uso.

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

JAVA_HOME: Variable que indica al sistema donde se encuentra el JDK de java.


Más información y configuración: <https://www.abrirllave.com/java/configurar-la-variable-de-entorno-java-home.php>

COLEXIO VIVAS S.L.	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Apéndice III: Cambios en Java 11

Los principales cambios (aunque no los únicos) de la versión LTS Java 11 frente a Java 8 han sido:

- Soporte LTS de 8 años, hasta 2026.
- Un nuevo sistema organizativo: los módulos (además de los packages de siempre)
- Existen tipos inferidos de forma automática mediante la cláusula var
- Se dispone de *jshell*, un intérprete java en consola.
- Con el comando java se puede compilar y ejecutar directamente pasando como argumento un archivo .java
- Nuevos métodos en clase String (repeat, strip, stripLeading, stripTrailing, isBlank)
- Unicode más completo (emojis,)


	RAMA:	Informática	CICLO:	DAM		
	MÓDULO	Programación				CURSO: 1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:	
	AUTOR	Francisco Bellas Aláez (Curro)				

Apéndice IV: Cambios en Java 17

Aunque aún esté muy extendido el uso de Java 11, el soporte del mismo acabará en 2026 por lo que conviene ir adaptándose a la nueva versión LTS (Long Term Support) que es Java 17.

Los principales cambios (aunque no los únicos) de la versión LTS Java 17 frente a Java 11 han sido:

- Soporte LTS hasta 2029
- Triple comilla (""") para strings formateados. Nuevos caracteres de escape \s y \ para eliminar retorno de carro.
- Uso de lambdas (->) en switch para devolver valores.
- Tipo immutable record (Automatiza creación de constructor, equals, toString,...)
- Clases selladas (sealed) para permitir herencia de forma selectiva.
- Uso extendido de instance of (Permite casting en el propio comando)
- NullPointerException con un mensaje más explicativo del problema.
- Escritura compacta de números (1K en vez de 1000)

	RAMA:	Informática	CICLO:	DAM			
	MÓDULO	Programación				CURSO:	1º
	PROTOCOLO:	Apuntes clases	AVAL:		DATA:		
	AUTOR	Francisco Bellas Aláez (Curro)					

Bibliografía (además de los enlaces indicados durante el texto):

Wikipedia

Java for the Beginning Programmer. Jeff Heaton. Heaton Research Inc.

Java for programmers. Paul&Harvey Deitel. Prentice Hall

Oracle: <http://www.oracle.com/technetwork/java/index.html>

<http://www.cafeaulait.org/course/>

<http://www.java2s.com/Code/JavaAPI/CatalogJavaAPI.htm>

Aprende Java2 como si estuvieras en primero.

Java 2. Apuntes de Abraham Otero (Javahispano)

<https://profile.es/blog/variables-tipos-datos-java/>

<https://medium.com/javarevisited/java-17-vs-java-11-exploring-the-latest-features-and-improvements-6d13290e4e1a>