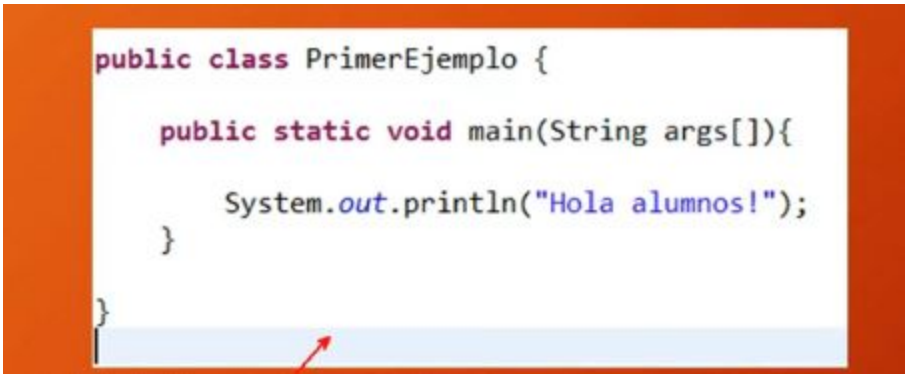
- **Estructura y bloques fundamentales.**



Public: modificador de acceso

Class: Todo programa en java debe estar al menos dentro de una clase. Ese del ejemplo de arriba es el programa mas sencillo que se puede hacer en java.

{} **Llaves**: Delimitan una parte del código. Deben cerrarse en el mismo orden que se abren.

Todas las sentencias terminan en **;**

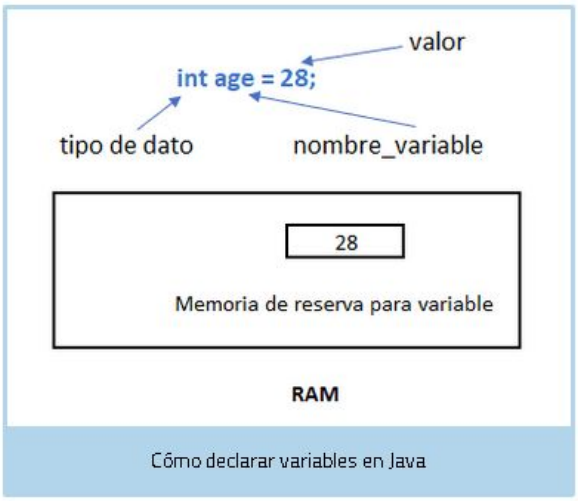
El programa en java estará en la carpeta src.

En la carpeta bin se genera el archivo .class, Este es el archivo multiplataforma, es decir, el que va a poder ejecutarse en cualquier plataforma (Mac,etc)

1. ¿Qué es una Variable?

Una variable es el nombre dado a una ubicación de memoria. Es la unidad básica de almacenamiento en un programa. El valor almacenado en una variable se puede cambiar durante la ejecución del programa.

2. ¿Cómo declarar variables?



- **tipo de dato**: tipo de datos que se pueden almacenar en esta variable.
- **nombre_variable**: nombre dado a la variable.
- **valor**: es el valor inicial almacenado en la variable.

Ejemplos

1. **float numero;** //Declarando variable float
2. **int time = 10, speed = 20;** //Declarando e Inicializando la variable integer
3. **char var = 'h';** // Declarando e Inicializando la variable character

- **Tipos de datos.**

1. Tipos de Datos

Java está tipado estáticamente y es fuertemente tipado porque en Java, cada tipo de datos (como entero, carácter, hexadecimal, decimal empaquetado, etc.) está predefinido como parte del lenguaje de programación y todas las constantes o variables definidas para un programa dado debe describirse con uno de los tipos de datos.

Java tiene dos categorías de datos:

- Datos Primitivos (p. Ej., int, char)
- Datos Objeto (tipos creados por el programador)

Los tipos de datos utilizados con más frecuencia en las declaraciones de variables Java se enumeran en la siguiente tabla, junto con una breve descripción:

2. Datos Primitivos

Los datos primitivos son solo valores únicos; ellos no tienen capacidades especiales. Java soporta **8 tipos de datos primitivos**:

Tabla Datos Primitivos en Java.

TIPO	DESCRIPCIÓN	DEFAULT	TAMAÑO	EJEMPLOS
boolean	true o false	false	1 bit	true, false
byte	entero complemento de dos	0	8 bits	100, -50
char	carácter unicode	\u0000	16 bits	'a', '\u0041', '\101', '\\'
short	entero complemento de dos	0	16 bits	10000,-20000
int	entero complemento de dos	0	32 bits	100000,-2,-1,0,1,2,-200000
long	entero complemento de dos	0	64 bits	-2L,-1L,0L,1L,2L
float	coma flotante IEEE 754	0.0	32 bits	1.23e100f, -1.23e-100f, .3ef, 3.14f
double	coma flotante IEEE 754	0.0	64 bits	1.2345e300d, -1.2345e-300f, 1e1d

Los tipos de datos primitivos se pueden organizar en 4 grupos:

- **Numéricos enteros**: Son los tipos *byte*, *short*, *int* y *long*. Los 4 representan números enteros con signo.
- **Carácter:** El tipo *char* representa un carácter codificado en el sistema unicode.
- **Numérico decimal**: Los tipos *float* y *double* representan números decimales en coma flotante.
- **Lógicos**: El tipo *boolean* es el tipo de dato lógico; los dos únicos posibles valores que puede representar un dato lógico son *true* y *false*. *true* y *false* son palabras reservadas de Java.

Nota: Recuerde que los valores de datos de **char** siempre deben estar rodeados por comillas simples, y los valores de datos de **String** siempre deben estar rodeados por comillas dobles.

2.1. Tipo de dato boolean

El tipo de datos booleano representa solo un bit de información: true (verdadero) o false (falso). Los valores de tipo booleano no se convierten implícita o explícitamente (con casts) en ningún otro tipo. Pero, el programador puede escribir fácilmente el código de conversión.

Por ejemplo:

```
// Un programa Java para demostrar el tipo de datos booleanos
class JavadesdeCero
{
    public static void main(String args[])
    {
        boolean b = true;
```

```
        if (b == true)
            System.out.println("Hola Javeros!");
    }
}
```

Salida:

```
Hola Javeros!
```

- **Literales.**

Una literal es la representación de un valor en el código fuente del programa.

1 Literales Enteros

Cualquier valor numérico entero es un literal entero. Los literales enteros se consideran de tipo int. Para especificar que un literal es de tipo long, debe añadirse, como sufijo, la letra L ó l.

Por ejemplo:

Los siguientes literales son de tipo int
12, 34, 0, -50, etc.

Los siguientes literales son de tipo long
92233720368547758O7L, 25L, -1L, etc.

2 Literales Reales

Cualquier valor numérico decimal con parte fraccionaria es un literal real. Los literales reales se consideran de tipo double. Para especificar que un literal es de tipo float, debe añadirse, como sufijo, la letra F ó f.

Por ejemplo:

Los siguientes literales son de tipo double
1.23, 3.456, -2.0, 3.25E+12, 2.7e-5, etc.

Los siguientes literales son de tipo float
2.75f, -4.567f, 2.0F, 6.73e+2f, etc.

3 Literales Booleanos

Los únicos literales booleanos son los siguientes: true, false

4 Literales de Carácter

Un literal de carácter consiste de un único carácter encerrado dentro de un par de comillas simples.

Por ejemplo:

'a', '1', '2', '$', etc.

Una secuencia de escape es un conjunto de caracteres

5 Literales de Cadena

Un literal de cadena consiste de un conjunto de carácteres encerrados entre comillas dobles.

Por ejemplo:

"Hola mundo"
"Bienvenido a Java"
"Algoritmos Computacionales"
"abcde123xy"
"Edad inválida"

▪ Constantes.

La palabra clave "**final**" es un modificador que se puede usar al declarar variables para **evitar cualquier cambio posterior** en los valores que inicialmente se les asignaron.
Esto es útil cuando se almacena un valor fijo en un programa para evitar que se altere accidentalmente.

Las variables creadas para almacenar valores fijos de esta manera se conocen como "**constantes**", y es convencional nombrar constantes con **todos los caracteres en mayúsculas**, para distinguirlas de las variables regulares. Los programas que intentan cambiar un valor constante no se compilarán, y el compilador **javac** generará un mensaje de error.

Ejemplo:

```
class Constants
{
public static void main ( String[] args ) {
   //inicializamos tres constantes enteras
   final int CONSTANTE1 = 6 ;
   final int CONSTANTE2 = 1 ;
```

```
   final int CONSTANTE3 = 3 ;

   //declaramos variables regulares del tipo int
   int td,pat,fg,total;

   //Inicializamos las variables regulares
   td = 4*CONSTANTE1;
   pat = 3*CONSTANTE2;
   fg = 2*CONSTANTE3;

  total = (td+pat+fg) ;

  System.out.println("Resultado: " + total);
  }
}
```

Salida:

```
Resultado: 33
```

- **Operadores y expresiones.**

Operador

Permiten realizar operaciones sobre uno, dos o tres operandos. Devuelven un resultado tras completar la operación.

Algunos de los operadores que podemos encontrar en Java son:

- **Operador de asignación**

Operador	Ejemplo
+=	a += b;
-=	a -= b;
*=	a *= b;
/=	a /= b;
%=	a %= b;

- **Operadores aritméticos**

Operador	Descripción	Sintaxis
+	Suma	a + b
–	Resta	a – b
*	Producto	a * b
/	División	a / b
%	Módulo	a % b

- **Operadores relacionales**

Operador	Descripción	Sintaxis
>	Mayor que	a > b
>=	Mayor o igual que	a >= b
<	Menor que	a < b
<=	Menos o igual que	a <= b
==	Igual que	a == b
!=	Distinto de	a != b

- **Operadores lógicos**

Operador	Descripción	Sintaxis
&&	And lógico (y)	a && b
||	Or lógico (o)	a || b
!	Negación lógica (no)	!a

Operando

Pueden ser variables, constantes y/o expresiones. Se sitúan en el lado derecho o izquierdo del operador.

Ejemplo:
int a = 2;
int b = 5;
a + b

Teniendo en cuenta que + es un operador (suma), *a* y *b* serán operandos. En este ejemplo son dos variables enteras.

Expresión

Conjunto de operandos y operadores que devuelve un resultado.

Ejemplo:
*a * ((b + c) / d)*

En el ejemplo todo el conjunto de operadores y operandos conforman una expresión, pero si la desglosamos nos daremos cuenta que está compuesta a su vez por otras expresiones que actuan como operandos.

Por ejemplo, b + c es una expresión, b y c son los operandos y + es el operador. A su vez es un operando de la expresión (b + c) / d. El primer operando es la expresión b + c, el segundo operando es d y el operador es / (división).

▪ Conversiones de tipo.

Java es un lenguaje fuertemente tipado, por lo que será necesario indicar el tipo de dato de un variable en su declaración. La información que se almacenará en dicha variable deberá ser

compatible con el tipo indicado durante toda la ejecución. Esto hace que el código sea más confiable, pero habrá ocasiones en las que nos puede resultar útil convertir un dato almacenado con un tipo a otro tipo diferente. A esta conversión de tipos se la conoce como **casting**.

Existen dos tipos de conversiones de tipo: **implícita** y **explícita**.

Conversión de tipos implícita

Se realiza de forma automática al asignar un valor de un tipo a otro compatible.

Ejemplo:
byte x = 5;
int y = x;

El valor almacenado en *x* (byte) se convertirá a entero al asignarse a *y* (int).

Las posibles conversiones implícitas son:

Nuevo tipo	Tipos origen
short	byte
int	byte, short, char
long	byte, short, char, int
float	byte, short, char, int, long
double	byte, short, char, int, long, float

Conversión de tipos explícita

Es tarea del programador especificar el nuevo tipo al que se va a transformar el dato. Se escribe de forma explicita entre paréntesis delante del dato.

Ejemplo:
byte a = 20;
int x = (int) a;

Al escribir int entre paréntesis se fuerza a cambiar el dato de tipo byte a int.

Hay que tener cuidado al realizar esta conversión ya que se puede aplicar a tipos no compatibles, lo que puede derivar en pérdidas de información e incluso errores en ejecución.

Ejemplo:
float x = 5.7;
int y = (int) x;

Se realizará la conversión, pero se perderá la parte decimal del número con punto flotante al guardarlo en *y*. Solo se guardará 5 en *y*.

Aunque parezca ilógico, habrá ocasiones en las que este tipo de conversiones nos pueden resultar útiles.

- **Comentarios.**

Comentarios en Java de una sola línea:

Pueden ser colocados en cualquier parte de nuestro código en Java y comienzan por un doble slash "//", al colocar el doble slash en cualquier línea de código, todo lo que haya de ahí en adelante en dicha línea será tomado como comentario, ten en cuenta que el doble slash solo convierte en comentario al texto que haya justo después de éstos y que pertenezca a su misma línea, las líneas de abajo de este, no se verán afectadas, tal como es de esperarse, el doble slash "//", solo afecta una línea desde el lugar donde se colocan.

Comentarios en Java de múltiples líneas:

Los comentarios multi-línea en Java tal como el nombre lo indica nos permiten comentar varias líneas de nuestro código Java de manera mucho más sencilla en vez de esta añadiendo doble slash "//" a cada línea. Estos comentarios van cerrados entre "/*" y "*/", es decir comienzan donde se ponga "/*" y terminan donde esté el "*/". Estos comentarios funcionan de manera similar a los comentarios de una sola línea, pero deben tener un comienzo y un final. A diferencia de los comentarios de una sola línea, al poner el símbolo "/*" todo el código que haya tanto en la misma línea, como en las línea posteriores de este se convertirán en comentarios hasta que pongamos el "*/", de manera que si iniciamos un comentario de múltiples líneas, debemos cerrarlo, tal como sucede con las llaves o los corchetes en Java.

- **Clase Scanner.**

El uso de la **clase Scanner** es una de las mejores maneras de ingresar datos por teclado en Java.

Scanner es una clase en el paquete **java.util** utilizada para obtener la entrada de los tipos primitivos como int, double etc. y también String. Es **la forma más fácil de leer datos** en un programa Java, aunque

no es muy eficiente si se quiere un método de entrada para escenarios donde el tiempo es una restricción, como en la programación competitiva.

En resumen:

Para crear un objeto de clase Scanner, normalmente pasamos el objeto predefinido **System.in**, que representa el flujo de entrada estándar. Podemos pasar un objeto de clase **File** si queremos leer la entrada de un archivo.

Para leer valores numéricos de un determinado tipo de datos XYZ, la función que se utilizará es nextXYZ(). Por ejemplo, para leer un valor de tipo *short*, podemos usar **nextShort()**.

Para leer cadenas (strings), usamos **nextLine()**.

Para leer un solo carácter, se usa **next().charAt(0)**. La función next()devuelve el siguiente token/palabra en la entrada como cadena y la función charAt (0) devuelve el primer carácter de esa cadena.

```
// Programa Java para leer datos de varios tipos usando la clase Scanner

import java.util.Scanner;


public class ScannerDemo

{

public static void main(String[] args)

{

// Declarar el objeto e inicializar con

// el objeto de entrada estándar predefinido


Scanner sc = new Scanner(System.in);


// entrada de una cadena

String name = sc.nextLine();


// entrada de un carácter

char gender = sc.next().charAt(0);
```

```
// Entrada de datos numéricos
// byte, short y float
int age = sc.nextInt();
long mobileNo = sc.nextLong();
double average = sc.nextDouble();

// Imprima los valores para verificar si la entrada
// fue obtenida correctamente.
System.out.println("Nombre: "+name);
System.out.println("Género: "+gender);
System.out.println("Edad: "+age);
System.out.println("Teléfono: "+mobileNo);
System.out.println("Promedio: "+average);
}
}
```