

```
"error:n?l:"No.conversion.from."+u+".to."-i}}u=i}
{}}function.$n(){return.setTimeout(function(){qn=t}
u=v.Deferred().always(function(){delete.a.elem}),a=
th(e,[f]),!1)},f=u.promise({elem:e,props:v.extend({
e,f.opts,t,n,f.opts.specialEasing[t]||f.opts.easing
s;Qn(l,f.opts.specialEasing);for(;i<o;i++){r=Xn[i].
ss(f.opts.progress).done(f.opts.done,f.opts.complet
&(i=s[1],s=e[n]=s[0]),n!==r&&(e[r]=s,delete.e[n]),o
g=e.nodeType&&Gt(e);n.queue||(l=v._queueHooks(e,"fx
.fire()}})),e.nodeType===1&&("height"in.t||"width"
ack":p.zoom=1)),n.overflow&&(p.overflow="hidden"
le";if(s==(g?"hide":"show"))continue;m.pu
"fxshow",!0);for(t.in.d)v.style(e,t
it(e,t,n,r,i)}function.Zn(e,t)
e.document,s=e.location
type.indexOf,h=0bi
/g,w=/^/?:[\w
-z]/o
```



Java™

2024

JAVA

MANUAL PARA PRINCIPIANTES

Autor: Jorge Moya Delgado |

Indice

1.	Java desde cero	2
1.1	¿Qué es Java?	2
	Lenguaje de objetos	2
	Java con vscode	2
2.	Primeras cosillas con java	6
2.1	Estructura de un programa en Java	6
2.2	Crear código en VSCode	8
2.3	Ejecutar código	10
3.	Variables	11
3.1	Datos primitivos	11
3.2	Tipos de datos primitivos	12
3.3	Valores por defecto	13
3.4	Tipos de variables	13
3.5	Constantes	13
3.6	Ejemplo completo	14
	Explicación	15
	Salida del código	15
4.	Operadores	16
4.1	Operadores aritméticos	16
	Ejemplo	16
4.2	Operadores relacionales	17
	Ejemplo	17
4.3	Operadores lógicos	18
	Ejemplo	18

4.4	Operadores de asignación.....	19
	Ejemplo	19
4.5	Operadores unitarios	20
	Ejemplo	20

1. Java desde cero

1.1 ¿Qué es Java?

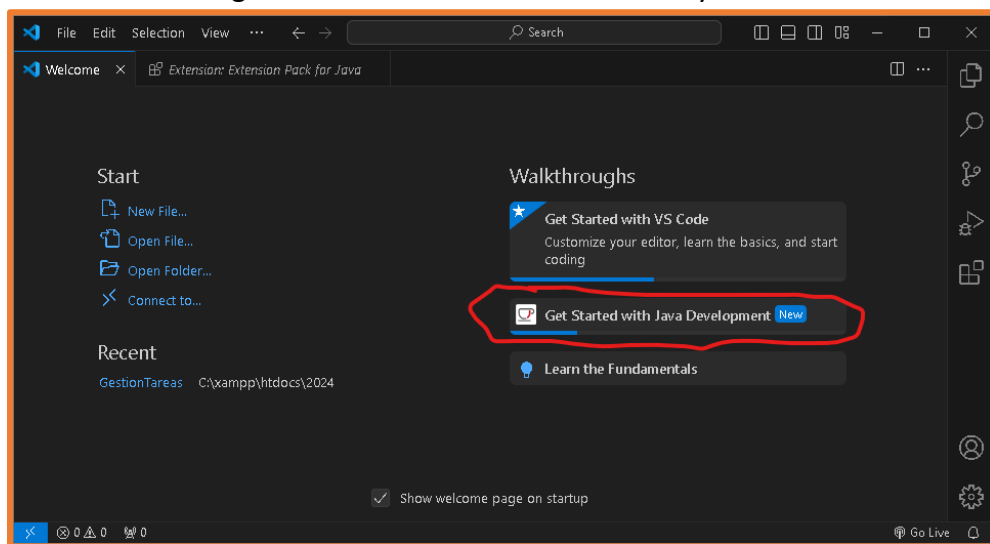
LENGUAJE DE OBJETOS JAVA CON VSCODE

Una vez tenemos instalado el VSCode en nuestro equipo debemos instalar la siguiente extensión:

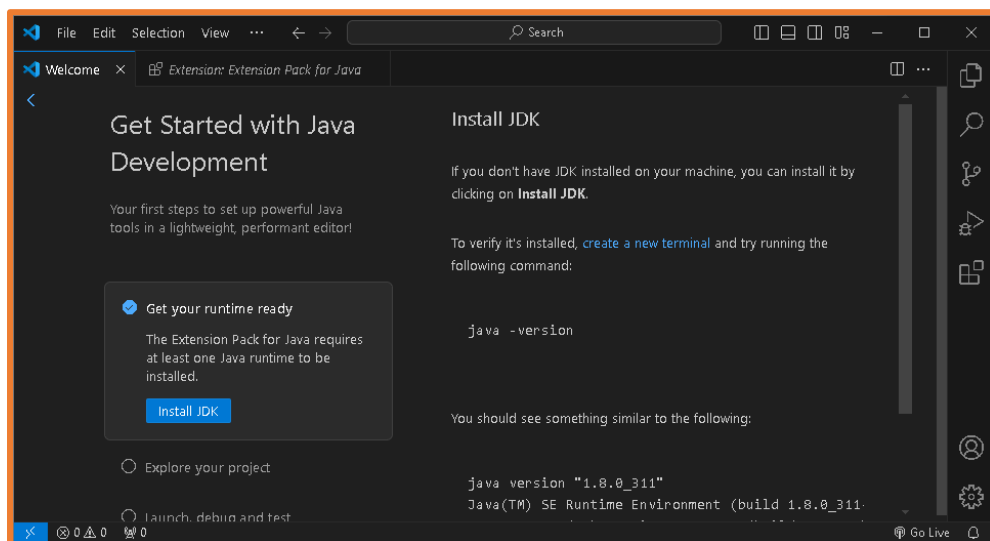
Extension Pack for Java:

<https://marketplace.visualstudio.com/items?itemName=vscjava.vscode-java-pack>

Una vez descargada e instalada. Cerramos VSCode y lo volvemos a abrir.

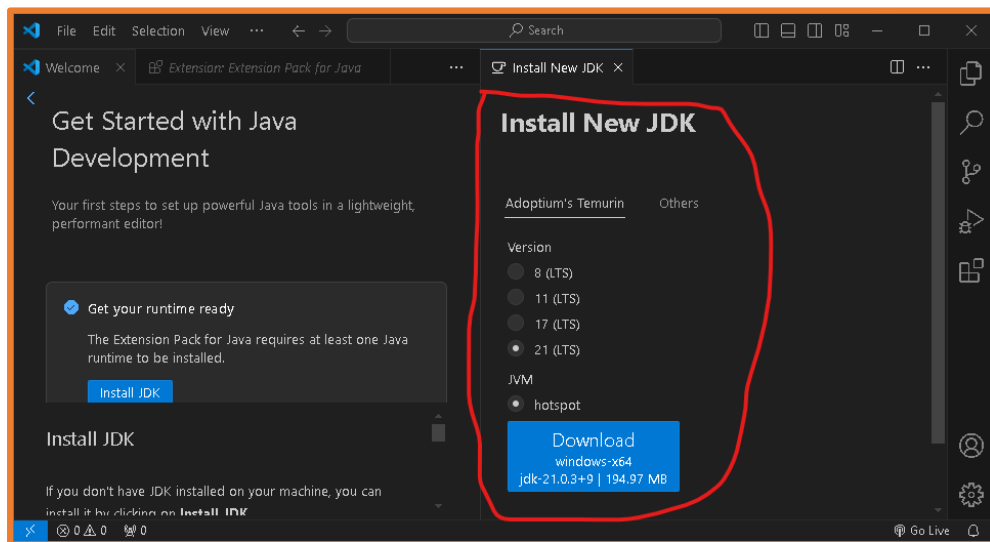


Seleccionamos la opción marcada. Y se nos abrirá la siguiente pestaña:

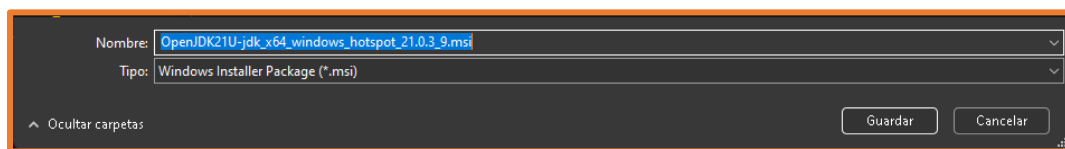


Seleccionamos **Install JDK**.

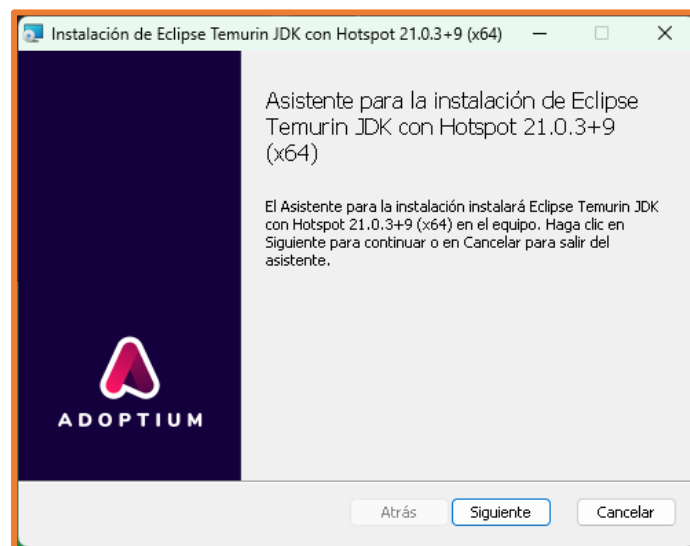
Se abrirá una 2ª ventana, donde podremos elegir la versión y otras configuraciones a tu gusto.

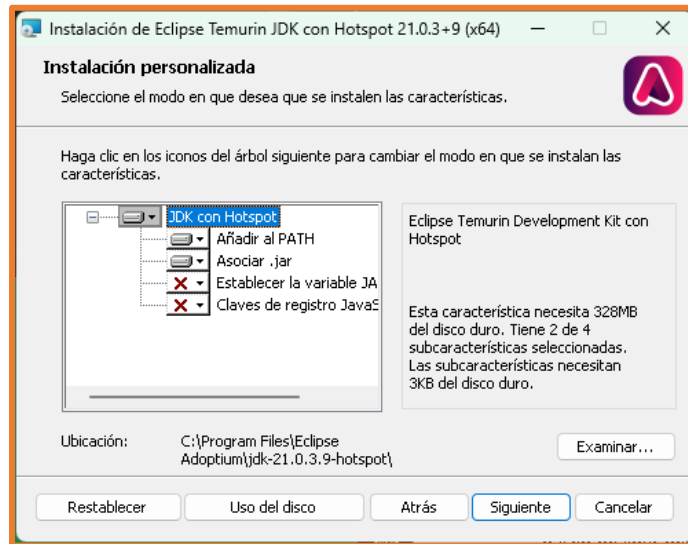


Descargamos el instalador

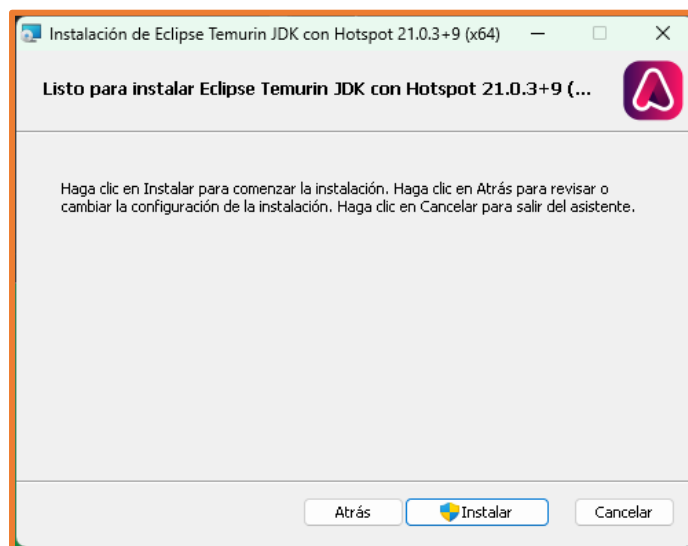


Una vez descargado lo abrimos para continuar con la instalación: (sucesión de capturas "irrelevantes") damos a siguiente siempre

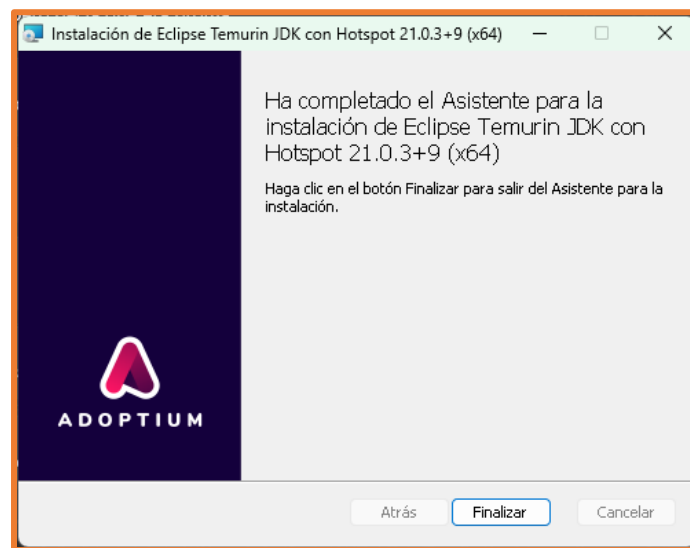
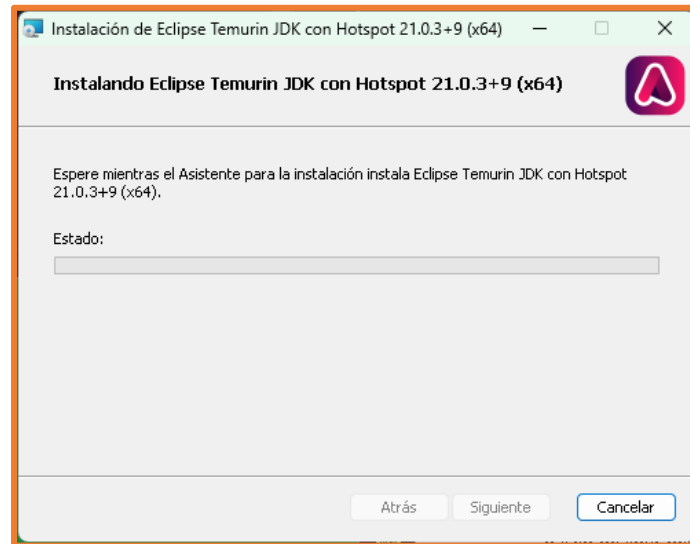




Ahora le damos a instalar:



Esperamos a que finalice la instalación.



2. Primeras cosillas con java

2.1 Estructura de un programa en Java



```
1 // 1. Una sentencia de paquete (package).
2 // package com.ejemplo.miapp;
3
4 // 2. Una o varias sentencias de importación (import).
5 // import java.util.Date;
```



```
1 // 3. Declaraciones de clases privadas deseadas.
2 class ClasePrivada {
3     private String atributo;
4
5     public ClasePrivada(String atributo) {
6         this.atributo = atributo;
7     }
8
9     public String getAtributo() {
10         return atributo;
11     }
12
13     public void setAtributo(String atributo) {
14         this.atributo = atributo;
15     }
16 }
```



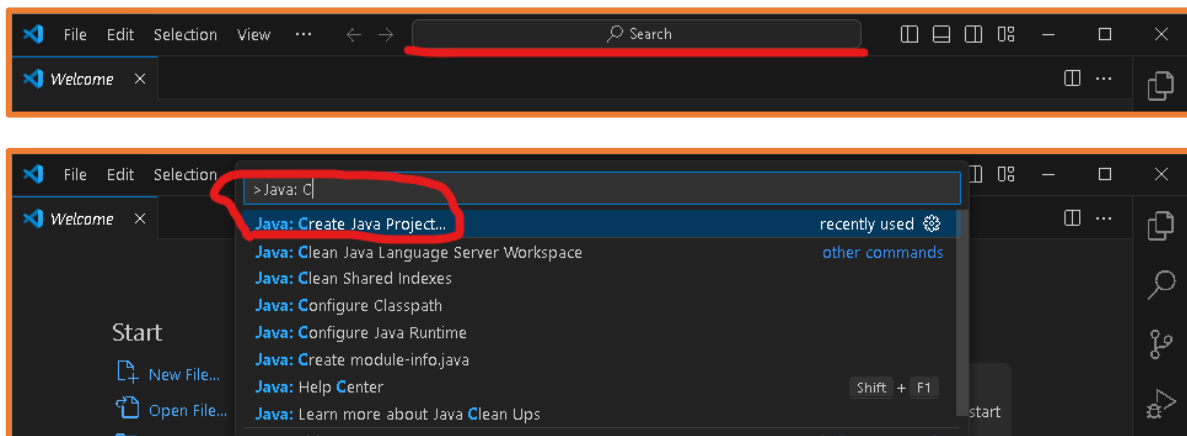
```
1 // 4. Una declaración de clase pública.
2 public class Ejemplo {
3     // Declaraciones de variables de la clase (estáticas).
4     public static final String CONSTANTE = "Constante de la clase";
5
6     // Declaraciones de variables de instancia.
7     private String nombre;
8     private int edad;
9
10    // Definiciones de constructores.
11    public Ejemplo(String nombre, int edad) {
12        this.nombre = nombre;
13        this.edad = edad;
14    }
15
16    // Definiciones de métodos.
17    public String getNombre() {
18        return nombre;
19    }
20
21    public void setNombre(String nombre) {
22        this.nombre = nombre;
23    }
24
25    public int getEdad() {
26        return edad;
27    }
28
29    public void setEdad(int edad) {
30        this.edad = edad;
31    }
32
33    // Comentarios en cualquier parte del programa.
34    public void mostrarInfo() {
35        // Imprime la información del objeto
36        System.out.println("Nombre: " + nombre);
37        System.out.println("Edad: " + edad);
38        System.out.println("Constante: " + CONSTANTE);
39    }
40
41    // Método main para ejecutar el programa
42    public static void main(String[] args) {
43        Ejemplo persona = new Ejemplo("Juan", 25);
44        persona.mostrarInfo();
45
46        ClasePrivada cp = new ClasePrivada("Valor privado");
47        System.out.println("Atributo de ClasePrivada: " + cp.getAtributo());
48    }
49 }
50
```

Resumiendo.

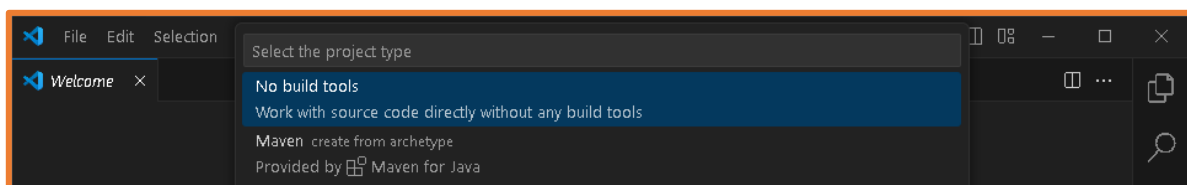
- 1- Sentencia de paquete (package)
- 2- Una o varias sentencias de importación (import)
- 3- Declaración de las clases privadas deseadas
- 4- Declaración de clase pública
 - a. Variables de clase
 - b. Variables de instancia
 - c. Definición de constructores
 - d. Definiciones de métodos
 - e. Comentarios por to' los laos'

2.2 Crear código en VSCode

Para crear un proyecto en Java debemos dirigirnos a la barra superior y poner lo siguiente:



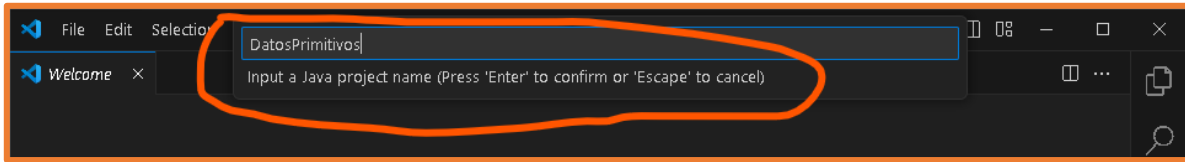
Seleccionamos No build tools:



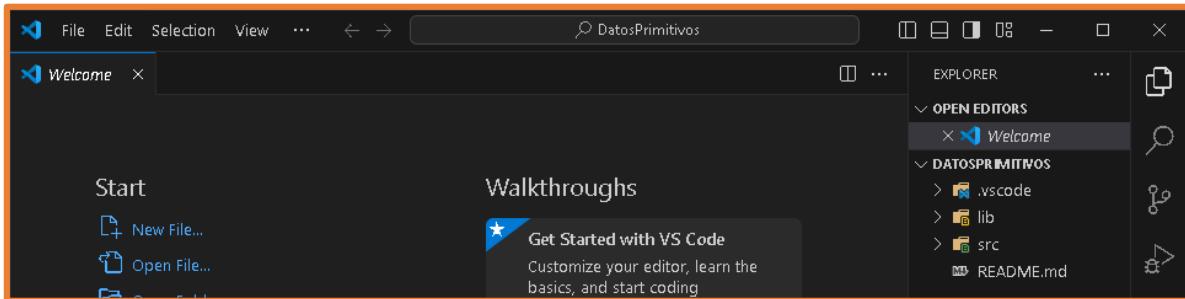
Elegimos nuestra carpeta de proyectos:



Le damos nombre a nuestro proyecto.

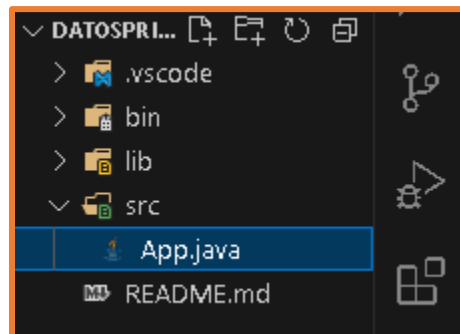


Se nos aparece el señor, bueno no, se aparece algo como esto:

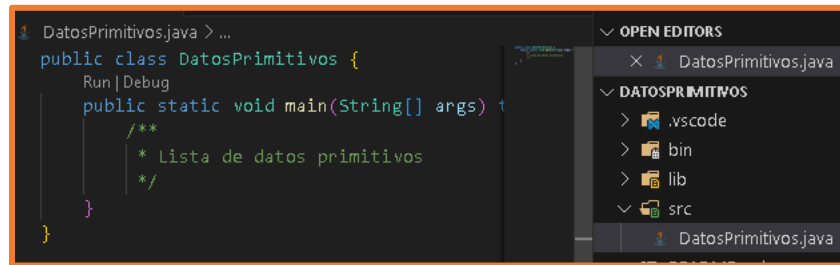


¿Dónde leñes se editan las movidas cojón? ¡Qué llevo leídas 10 páginas ya!

Empieza por aquí jefe crack titán.

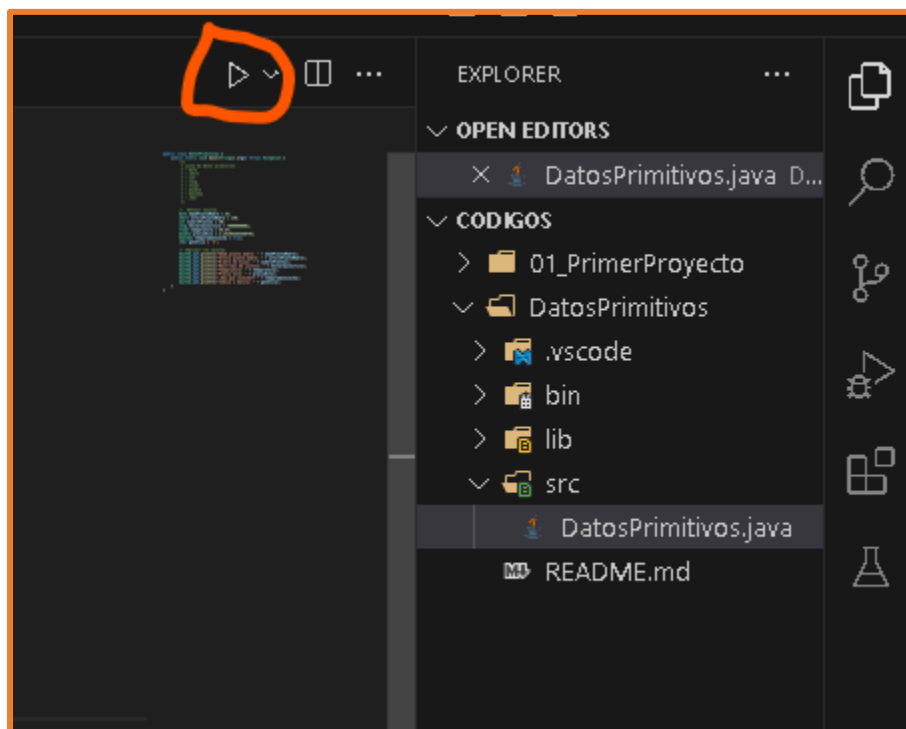


Si renombas el fichero App, por el nombre que te salga de los mismos, y lo cambias en el código también, podrás renombrar a tu gusto.



2.3 Ejecutar código

Para ejecutar nuestro proyecto debemos hacer clic en el icono del “play” de arriba a la izquierda:



3. Variables

3.1 Datos primitivos

Hay que saber que Java, al ser de tipado estático (moviolas que ya veremos) nos obliga a indicarle el tipo de dato de la variable antes de definir ésta (sí, con tilde porque soy un rebelde).

```
1 public static void main(String[] args) throws Exception {
2     /**
3      * Lista de datos primitivos
4      * - byte
5      * - short
6      * - int
7      * - long
8      * - float
9      * - double
10     * - boolean
11     * - char
12     */
13
14     // Definir valores
15     byte edadPlantaMadre = 25;
16     short alturaPlantaMadre = 150;
17     int numeroPlantas = 50;
18     long hectareasCultivo = 7800000000L;
19     float temperatura = 36.6f;
20     double precioKilo = 2.141592653589793;
21     boolean temporadaCosecha = true;
22     char genetica = 'S';
23
24     // Imprimir los valores
25     System.out.println("Edad planta madre: " + edadPlantaMadre);
26     System.out.println("Altura planta madre: " + alturaPlantaMadre);
27     System.out.println("Número de hijos: " + numeroPlantas);
28     System.out.println("Hectáreas de cultivo: " + hectareasCultivo);
29     System.out.println("Temperatura: " + temperatura);
30     System.out.println("Precio kilo: " + precioKilo);
31     System.out.println("¿Hay que cosechar? " + temporadaCosecha);
32     System.out.println("Indica o Sativa: " + genetica);
33 }
```

Una vez ejecutemos nos aparecerá lo siguiente por consola:

```
PS D:\Proyectos\Java\codigos> & 'C:\workspaceStorage\ad264888374f034a5ed839eca:
Edad planta madre: 25
Altura planta madre: 150
Número de hijos: 50
Hectáreas de cultivo: 7800000000
Temperatura: 36.6
Precio kilo: 2.141592653589793
¿Hay que cosechar? true
Indica o Sativa: S
PS D:\Jorge\Proyectos\Java\codigos>
```

3.2 Tipos de datos primitivos

Byte: Es un tipo de dato entero de 8 bits con signo, capaz de almacenar valores numéricos desde -128 hasta 127, incluidos ambos extremos.

Short: Es un tipo de dato entero de 16 bits con signo que puede almacenar valores numéricos que oscilan entre -32,768 y 32,767.

Int: Es un tipo de dato entero de 32 bits con signo, utilizado para almacenar números. Su rango va desde -2^{31} hasta $2^{31}-1$.

Long: Se trata de un tipo de dato entero de 64 bits con signo, que permite almacenar números entre -2^{63} y $2^{63}-1$.

Float: Es un tipo de dato de 32 bits para almacenar números en coma flotante con precisión simple.

Double: Este tipo de dato de 64 bits se utiliza para almacenar números en coma flotante con doble precisión.

Boolean: Sirve para definir datos booleanos, que solo pueden tener dos valores: true o false. Utiliza 1 bit de espacio.

Char: Es un tipo de dato que representa un carácter Unicode de 16 bits.

3.3 Valores por defecto

Byte	0
Short	0
Int	0
Long	0L
float	0.0f
Double	0.0d
Char	'�0000'
String (o cualquier objeto)	Null
boolean	False

3.4 Tipos de variables

- **Variables locales:** Se declaran dentro de un m todo y solo son accesibles dentro de este m todo
- **Variables de instancia:** Se declaran dentro de una clase, pero fuera de los m todos. Cada objeto de la clase tendr  su propia copia
- **Variables de clase (est ticas):** Se declaran con la palabra *static*. Pertenecen a la clase y no a instancias (objetos) de la clase

3.5 Constantes

Una constante es una manera de llamar a una variable (Si alg n profesor lee eso me cruje) que no cambia de valor NUNCA durante la ejecuci n del programa.

Se declaran con la palabra *final* y el nombre de la constante (Convencionalmente se usan letras may sculas separadas por barra baja)

3.6 Ejemplo completo

```
1 public class AtleticoDeMadrid {
2
3     // 1. Variable de clase (estática)
4     // Esta variable pertenece a la clase y es compartida por todos los objetos de la clase.
5     static String nombreDelEquipo = "Atlético de Madrid";
6
7     // 2. Variable de instancia
8     // Cada objeto tendrá un nombre de jugador único y la posición del jugador.
9     String nombreDelJugador;
10    String posicion;
11
12    // 3. Constante
13    // Esta constante indica el número máximo de jugadores permitidos en un equipo de fútbol.
14    final int MAX_JUGADORES = 25;
15
16    // Constructor para inicializar la variable de instancia
17    public AtleticoDeMadrid(String nombre, String posicion) {
18        this.nombreDelJugador = nombre; // Inicializando el nombre del jugador
19        this.posicion = posicion; // Inicializando la posición del jugador
20    }
21
22    // Método que demuestra el uso de variables locales
23    public void mostrarInformacionDelJugador() {
24        // 4. Variable local
25        // Esta variable solo es accesible dentro de este método.
26        int numeroDeGoles = 10; // Supongamos que el jugador ha marcado 10 goles
27
28        System.out.println("Nombre del jugador: " + nombreDelJugador); // Variable de instancia
29        System.out.println("Posición: " + posicion); // Variable de instancia
30        System.out.println("Goles marcados: " + numeroDeGoles); // Variable local
31        System.out.println("Equipo: " + nombreDelEquipo); // Variable de clase
32        System.out.println("Número máximo de jugadores permitidos en el equipo: " + MAX_JUGADORES); // Constante
33    }
34
35    public static void main(String[] args) {
36        // Creación de un objeto de la clase AtleticoDeMadrid representando a un jugador
37        AtleticoDeMadrid jugador1 = new AtleticoDeMadrid("Antoine Griezmann", "Delantero");
38
39        // Llamada al método para mostrar la información del jugador
40        jugador1.mostrarInformacionDelJugador();
41
42        // Creación de otro objeto representando a un segundo jugador
43        AtleticoDeMadrid jugador2 = new AtleticoDeMadrid("Jan Oblak", "Portero");
44        jugador2.mostrarInformacionDelJugador();
45    }
46 }
```


EXPLICACIÓN

Variables de clase (estáticas): *nombreDelEquipo* es una variable estática. Todos los objetos creados de la clase ***AtleticoDeMadrid*** compartirán este valor, ya que todos los jugadores forman parte del mismo equipo.

Variables de instancia: *nombreDelJugador* y *posicion* son variables de instancia. Cada jugador tendrá su propio nombre y posición, por lo que cada objeto (que representa a un jugador) tendrá diferentes valores para estas variables.

Variables locales: La variable *numeroDeGoles* es una variable local. Está definida dentro del método ***mostrarInformacionDelJugador*** y representa la cantidad de goles que un jugador ha marcado. Solo existe dentro del método y no es accesible fuera de él.

Constantes: La constante ***MAX_JUGADORES*** indica el número máximo de jugadores permitidos en el equipo de fútbol, que es 25. Este valor no puede cambiar y es igual para todos los objetos de la clase.

SALIDA DEL CÓDIGO

```
Nombre del jugador: Antoine Griezmann
Posición: Delantero
Goles marcados: 10
Equipo: Atlético de Madrid
Número máximo de jugadores permitidos en el equipo: 25
Nombre del jugador: Jan Oblak
Posición: Portero
Goles marcados: 10
Equipo: Atlético de Madrid
Número máximo de jugadores permitidos en el equipo: 25
```

4. Operadores

Los operadores son generalmente las personas que realizan una operación. Pero como esto es Java y no ciencias de la salud pues hay que dar otra explicación.

Los operadores en Java son símbolos que nos ayudan a realizar operaciones sobre 1,2 o las variables que sean, también pueden afectar a los valores.

4.1 Operadores aritméticos

Estos operadores (símbolos) nos permiten realizar operaciones matemáticas como la suma, resta, multiplicación, división y resto.

Operador	Operación	Ejemplo
+	Suma	A + b
-	Resta	A - b
*	Multiplicación	A * b
/	División	A / b
%	Resto de la división	A % b

EJEMPLO

```

1  // 1. Función para Operadores Aritméticos
2      public static void operadoresAritmeticos() {
3          int a = 10;
4          int b = 3;
5
6          System.out.println("\n=== Operadores Aritméticos ===");
7          System.out.println("Suma: " + (a + b)); // 13
8          System.out.println("Resta: " + (a - b)); // 7
9          System.out.println("Multiplicación: " + (a * b)); // 30
10         System.out.println("División: " + (a / b)); // 3
11         System.out.println("Resto: " + (a % b)); // 1
12     }

```

Importante recordar que sirven para variables (ejemplo de arriba) como para valores (por ejemplo: 3 + 2) te la hincó.

4.2 Operadores relacionales

Los operadores de igualdad en Java nos sirven para comparar valores y/o variables entre ellas. Imaginate que quieres saber quien tiene menos ganas de vivir en tu grupo de amigos, con estos operadores podrías saberlo.

Operador	Operación	Ejemplo
==	Igual a	A == b
!=	NO es igual a	A != b
>	Mayor que	A > b
<	Menor que	A < b
>=	Mayor o Igual que	A >= b
<=	Menor o igual que	A <= b

EJEMPLO

```

1 // 2. Función para Operadores Relacionales
2 public static void operadoresRelacionales() {
3     int a = 10;
4     int b = 20;
5
6     System.out.println("\n=== Operadores Relacionales ===");
7     System.out.println("a es igual a b: " + (a == b)); // false
8     System.out.println("a es diferente a b: " + (a != b)); // true
9     System.out.println("a es mayor que b: " + (a > b)); // false
10    System.out.println("a es menor o igual que b: " + (a <= b)); // true
11 }

```

4.3 Operadores lógicos

Los operadores lógicos en Java se usan para combinar expresiones booleanas.

Operador	Operación	Ejemplo
&&	AND	(3 < 5) && (1 > 6)
	OR	(2 > 1) (2 > 13)
!	NOT	!(3 > 2)
?	Ternario	(expresion)?valor_true:valor_false;

En el caso del operador lógico AND el resultado será **true** *siempre y cuando las dos expresiones evaluadas sean true*. Si una de las expresiones es false el resultado de la expresión condicional AND será false.

Para el operador lógico OR el resultado será **true** *siempre que alguna de las dos expresiones sea true*.

El operador lógico NOT es algo duro de entender al principio, digamos que es una negación. En este caso el resultado del paréntesis (3 > 2) devolvería **true** pero como tenemos un ! delante, la salida sería **false**. Se le coge cariño, lo juro.

Vale a estas alturas de la película no nos vamos a preocupar mucho por el tema del ternario, ya llegaremos con las estructuras de control.

EJEMPLO

```
1 // 3. Función para Operadores Lógicos
2 public static void operadoresLogicos() {
3     boolean x = true;
4     boolean y = false;
5
6     System.out.println("\n=== Operadores Lógicos ===");
7     System.out.println("x && y: " + (x && y)); // false
8     System.out.println("x || y: " + (x || y)); // true
9     System.out.println("!x: " + (!x)); // false
10    System.out.println("!(3 > 2): " + !(3 > 2)); // false
11 }
```

4.4 Operadores de asignación

Realmente estos operadores combinan los operadores aritméticos con el operador de asignación (=)

Operador	Operación	Ejemplo
=	Asignación	A = 5
+=	Asignación de suma	A += 5 (a = a + 5)
-=	Asignación de resta	A -= 5 (a = a - 5)
*=	Asignación de multiplicación	A *= 5
/=	Asignación de división	A /= 5
%=	Asignación de resto	A %= 5

EJEMPLO

```

1  // 4. Función para Operadores de Asignación
2      public static void operadoresAsignacion() {
3          int a = 10;
4          a += 5; // a = a + 5
5          int b = 10;
6          b -= 5;
7
8          System.out.println("\n=== Operadores de Asignación ===");
9          System.out.println("Nuevo valor de a: " + a); // 15
10         System.out.println("Nuevo valor de b: " + b); // 5
11     }

```

4.5 Operadores unitarios

Operador	Operación	Ejemplo
+	Valor positivo	+a
-	Valor negativo	-a
++	Incremento	A++ o ++a
--	Decremento	a-- o --a

EJEMPLO

```
1 // 5. Función para Operadores Unarios
2 public static void operadoresUnarios() {
3     int a = 5;
4
5     System.out.println("\n=== Operadores Unarios ===");
6     System.out.println("a: " + a); // 5
7     System.out.println("a: " + -a); // -5
8     System.out.println("a++: " + a++); // 5 (usa a y luego incrementa)
9     System.out.println("++a: " + ++a); // 7 (incrementa antes de usar)
10    System.out.println("a--: " + a--); // 7 (usa a y luego decrementa)
11    System.out.println("--a: " + --a); // 5 (decrementa antes de usar)
12 }
```