



Capítulo 2

Programación estructurada

2.1. Introducción

Cuando ya se tienen claro los pasos a seguir para resolver un problema, hay que escribirlos utilizando un lenguaje de programación.

Un **lenguaje de programación** puede definirse como un idioma artificial diseñado para que sea fácilmente entendible por un humano e interpretable por una máquina. Consta de una serie de reglas y de un conjunto de órdenes o instrucciones.

Cada una de las **instrucciones** realiza una tarea determinada. A través de una **secuencia** de instrucciones podemos indicar a una computadora el **algoritmo** que debe seguir para solucionar un problema dado. A un algoritmo escrito en un lenguaje de programación se le denomina **programa**.

La **programación** es la acción de determinar la secuencia de instrucciones de un programa informático, con el fin de resolver un problema.

Un lenguaje de programación está diseñado para que una persona escriba fácilmente algoritmos, pero un ordenador no comprende ningún lenguaje distinto al sistema binario (0 1). Para conseguir que un ordenador comprenda lo que se ha escrito mediante un lenguaje de programación se usa una herramienta llamada **compilador**, que transforma las instrucciones escritas en un lenguaje de programación (también llamado código fuente) en los ceros y unos que son comprensibles para la máquina (código máquina).

Existen dos maneras de realizar el proceso de traducción de código fuente a código máquina. La primera, llamada **compilación**, traduce todas las instrucciones del código fuente y almacena el código máquina generado. La otra manera se llama **interpretación** y consiste en traducir el código

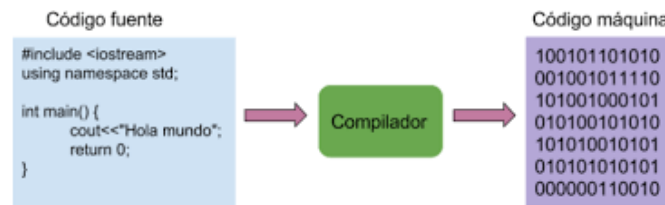


Figura 2.1: Compilador

instrucción a instrucción e ir ejecutando, es decir, solo se traduce y ejecuta las siguiente instrucción.

Historia de los lenguajes de programación https://es.wikipedia.org/wiki/Historia_de_los_lenguajes_de_programaci%C3%B3n#:~:text=En%20los%20cincuenta%2C%20los%20tres,altamente%20influenciado%20por%20Grace%20Hopper.

<https://www.youtube.com/watch?v=TrzZ7YQyXbs> (15:48)

<https://computerhoy.com/reportajes/tecnologia/historia-lenguajes-programacion-42804>

Lenguajes de programación hoy en día <https://www.universia.net/es/actualidad/empleo/lenguajes-programacion-mas-usados-actualidad-1136443.html>

<https://blog.centrodelearning.com/2021/12/15/lenguajes-de-programacion-mas-utiliza>

<https://www.campusmvp.es/recursos/post/los-5-lenguajes-de-programacion-mas-faciles.aspx>

En esta asignatura utilizaremos el lenguaje de programación **JAVA**.

Lenguaje de programación Java <https://www.youtube.com/watch?v=Kcflug9eegw> (23:51)

Java es un lenguaje de programación orientado a objetos creado en 1991 y publicado en 1995 por Sun Microsystem (adquirida por Oracle en 2010). Se concibió como un lenguaje para internet y necesita ser multiplataforma para que un programa se compile una vez y pueda ser ejecutado en multitud de ordenadores y sistemas operativos diferentes (WORA - write once, run anywhere). Esto se consigue compilando a un lenguaje binario especial denominado **bytecode**. No es ejecutable en ningún ordenador porque es independiente de cualquier plataforma. Para ejecutarlo es necesario un intérprete (JVM = Máquina virtual java) en cada equipo que traduce de bytecode a código máquina propio de cada plataforma.





Figura 2.2: Los lenguajes de programación más usados.

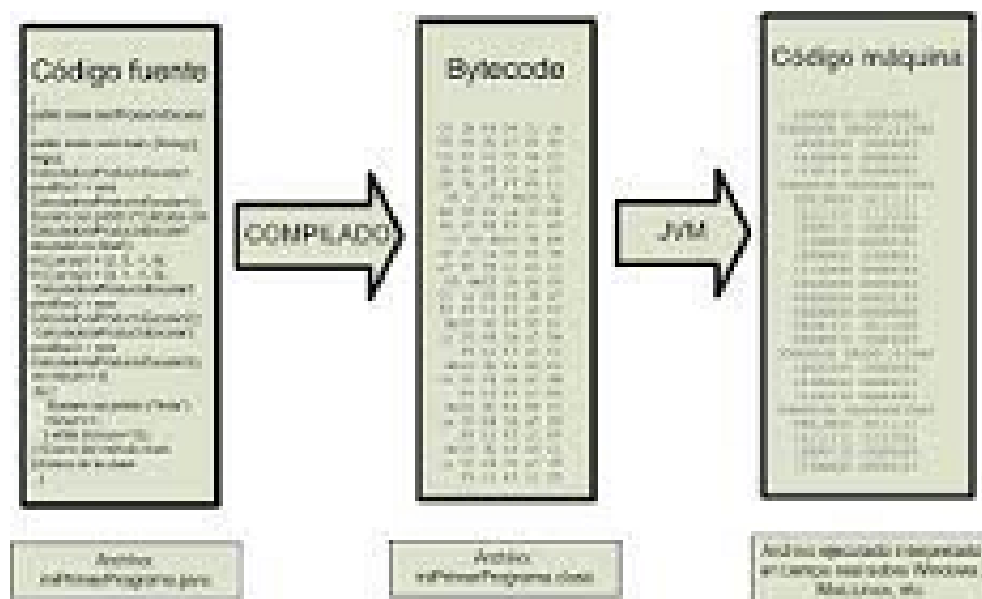


Figura 2.3: Bytecode

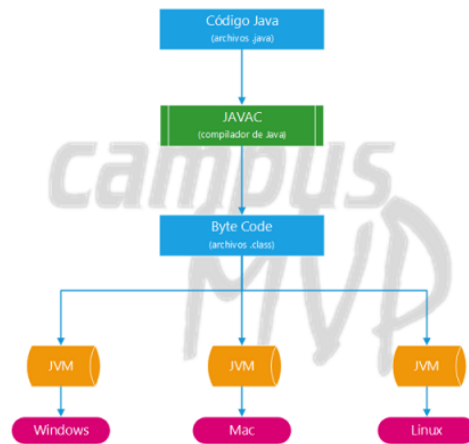


Figura 2.4: JVM

El hecho de que el código java pueda ser ejecutado en cualquier sistema gracias a la máquina virtual hace que se esté utilizando en millones de dispositivos <https://www.campusmvp.es/recursos/post/es-cierto-que-3-mil-millones-de-dispositivos> [aspx](https://www.campusmvp.es/recursos/post/es-cierto-que-3-mil-millones-de-dispositivos)

El JDK (java development kit) es el entorno para crear y ejecutar aplicaciones java. Este JDK posee un compilador que toma nuestro código Java y valida la sintaxis. Si el compilador encuentra algún error en nuestro código nos mostrará un mensaje (errores de compilación) y si todo está bien el compilador nos creará un archivo con código bytecode. Este archivo será ejecutado por la máquina virtual java. El jdk se puede descargar desde <http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Un **entorno de desarrollo integrado** o entorno de desarrollo interactivo, en inglés Integrated Development Environment (IDE), es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software. Normalmente, un IDE suele estar formado por un editor de código fuente, herramientas de construcción automáticas y un depurador.



El entorno de desarrollo integrado que usaremos en clase será IntelliJ Idea.

https://es.wikipedia.org/wiki/IntelliJ_IDEA

<https://www.jetbrains.com/es-es/idea/>

2.2. Crear un proyecto Java con IntelliJ Idea

Para empezar a programar en Java vamos a utilizar el IDE IntelliJ Idea.



En el aula está ya instalado y en la carpeta software del curso moodle de programación tenéis el ejecutable.

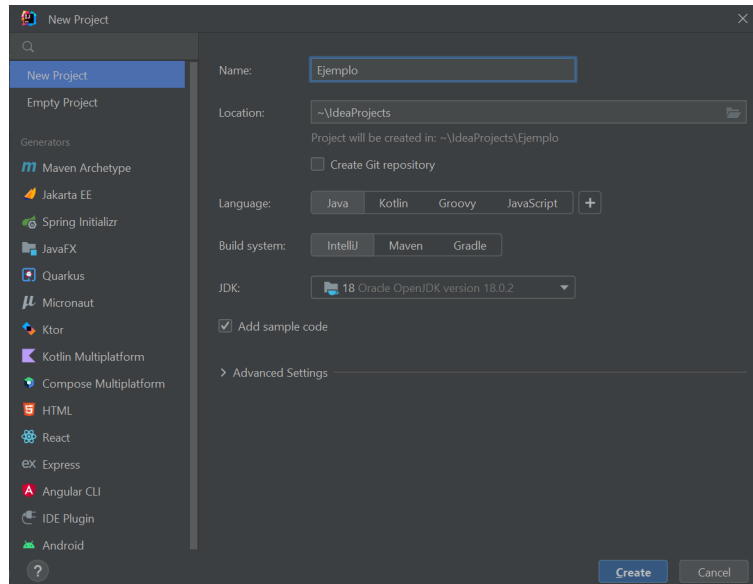


Figura 2.5: Crear un proyecto nuevo java

2.3. Estructura básica de un programa

Objetivo: Escribir en pantalla Hola Mundo.

Datos de entrada: No hay.

Datos de salida: Hola Mundo

```
1 Algoritmo holaMundo
2   Escribir 'Hola Mundo'
3 FinAlgoritmo
4
```

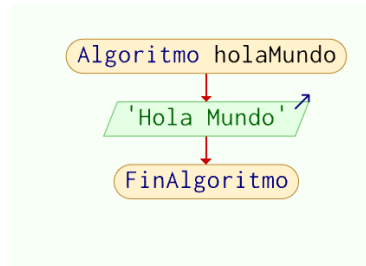
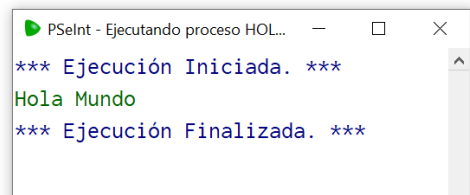


Figura 2.6: Pseudocódigo, diagrama de flujo y ejecución

Codificación en java:

```
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println("Hola Mundo");
4     }
5 }
```

El diagrama de flujo y el pseudocódigo lo he desarrollado con una herramienta denominada PseInt que también se encuentra en la carpeta software.

2.4. Comentarios (documentación)

```
1
2 public class Main {
3     /* Este programa sirve para .....
4     Fecha:
5     Autor:
6     .....
7     */
8     public static void main(String[] args) {
9         // Escribir "Hola Mundo" en inglés.
10        System.out.println("Hello world!");
11    }
12 }
```

Un comentario es una construcción destinada a integrar información adicional en el código fuente de un programa. Cuando el código fuente es procesado por un compilador o intérprete, los comentarios no se toman en cuenta.

Los comentarios tienen una amplia gama de posibles usos: desde la mejora del código fuente con descripciones básicas hasta la generación de documentación externa (javadoc). También se utilizan para la integración con sistemas de control de versiones y otros tipos de herramientas de programación externas.

En Java hay tres tipos de comentarios:

// comentarios para una sola línea

/* comentarios de una o

más líneas */

/** comentario de documentación, de una o más líneas */

Los comentarios de documentación, indican que han de ser colocados en la documentación que se genera automáticamente cuando se use la herramienta *javadoc*.

Los comentarios que llevan la palabra TODO (por hacer) nos permiten estructurar un programa (decir que hay que hacer) y posteriormente hacerlo. Nos permiten documentar las tareas futuras.

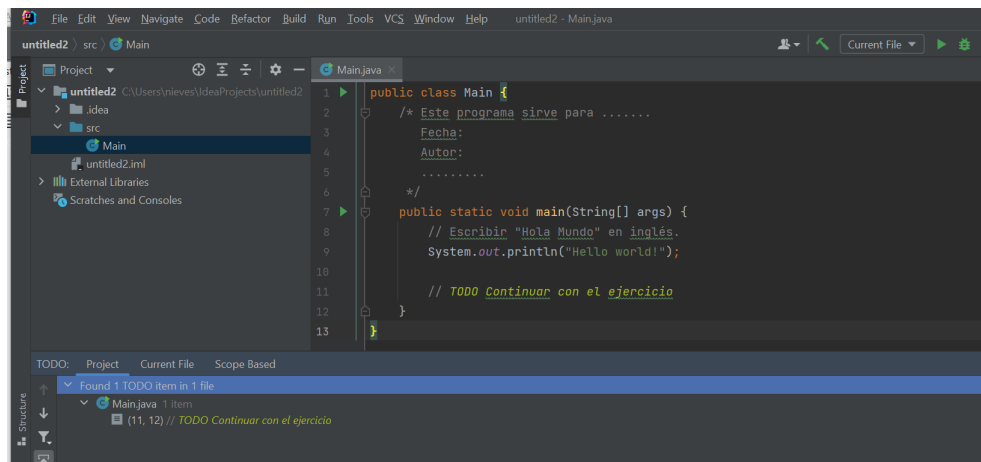


Figura 2.7: Comentarios de tipo TODO

2.5. Variables y constantes

Una **variable** es una especie de caja o celda de memoria, que se distingue por un nombre (identificador), en la que se almacena un dato que puede ser modificado. Una variable no es más que un nombre simbólico que identifica una o más direcciones de memoria.

Una variable nos permite guardar un dato en un programa.

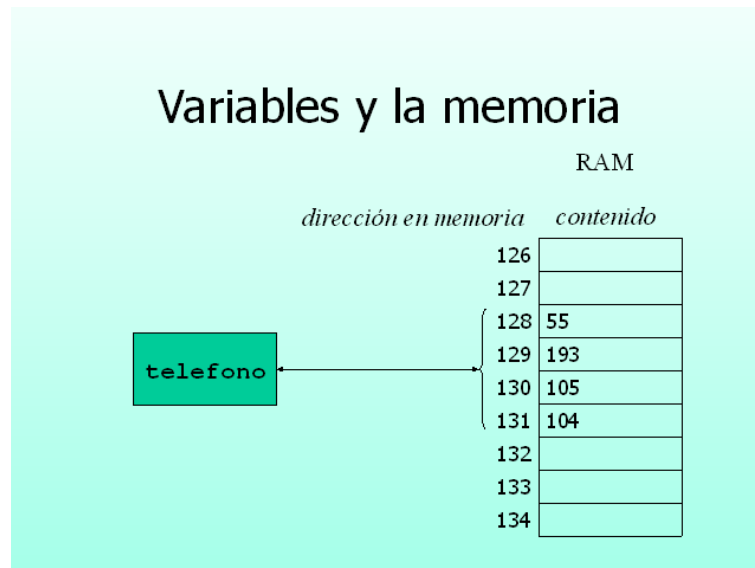


Figura 2.8: Variable = una o más celdas en memoria

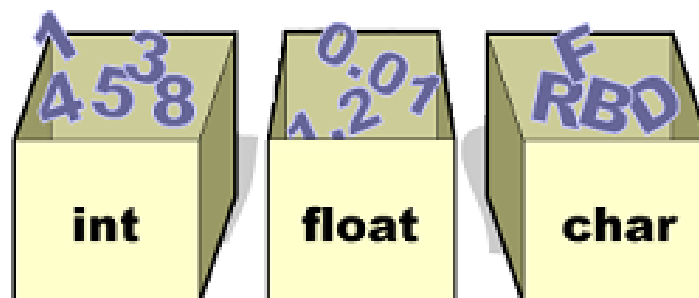


Figura 2.9: Guardar datos en variables

Para usar una variable en un programa codificado en Java hay que **declararla**. Al declarar una variable, se reserva el espacio de memoria necesario para almacenar un valor del tipo de la variable. El identificador o nombre asociado a la variable se puede utilizar para acceder al dato almacenado en memoria y para modificarlo.

Una **constante** es una variable en la que se almacena un dato que **NO** puede ser modificado. Se definen igual que cuando se declara una variable y se inicializa su valor. Con la palabra reservada *final* se impide la modificación del valor almacenado.

La **declaración de variables** en java consta de dos partes: el tipo de datos que va a contener (determina la cantidad de espacio que se reserva) y el identificador o nombre. En una misma declaración se pueden declarar varias variables, siempre que sean del mismo tipo. En este caso, los nombres de las variables se separan por comas.

```
1
2 tipoDato identificador1, identificador2;
3
4 tipoDato2 identificador3;
```

2.5.1. Identificadores o nombres de variables

En Java, un identificador comienza con una letra, un subrayado o un símbolo de dólar. Los siguientes caracteres pueden ser letras o dígitos. Se distinguen las mayúsculas de las minúsculas y no hay longitud máxima.

Las siguientes palabras son las palabras clave que están definidas en Java y que no se pueden utilizar como identificadores: *abstract continue for new switch boolean default goto null synchronized break do if package this byte double implements private threadsafe byvalue else import protected throw case extends instanceof public transient catch false int return true char final interface short try class finally long static void const float native super while*

Los identificadores deben ser **autodescriptivos**, es decir, deben hacer referencia al contenido.

Se usa la notación camelCase, es decir, los nombres asociados a las variables se ponen en minúsculas. Cuando está formado por varias palabras, la primera palabra va en minúsculas y el resto de palabras se inician con una letra mayúscula. Para el identificador de las constantes se suelen utilizar letras mayúsculas separadas por el carácter subrayado.



Hay que tener en cuenta que a veces la letra ñ da problemas.

2.5.2. Tipos de datos en Java

Lógicos	boolean	true, false
Caracteres	char	caracteres unicode
Numéricos	byte	entero de 8 bits
	short	entero de 16 bits
	int	entero de 32 bits
	long	entero de 64 bits
	float	real de 32 bits
	double	real de 64 bits

Figura 2.10: Tipos de datos

- Tipos de datos **numéricos enteros**

Se usan para representar números enteros con signo. Hay cuatro tipos: **byte** (8 bits) (valores numéricos de -128 a 127), **short** (16 bits) (de -32.768 a 32.767), **int** (32 bits) (de -2.147.483.648 a 2.147.483.647) y **long** (64 bits) (sin límite).

- Tipos de datos en **numéricos reales o coma flotante**

Se usan para representar números con partes fraccionarias. Hay dos tipos de coma flotante: **float** y **double**. El primero reserva almacenamiento para un número de precisión simple de 4 bytes (valores numéricos hasta 38 cifras) y el segundo lo hace para un número de precisión doble de 8 bytes (valores numéricos hasta 308 cifras).

- Tipo de datos **boolean**

Se usa para almacenar variables que presenten dos estados que serán representados por los valores true y false.

- Tipo de datos **char**.

Se usa para almacenar caracteres Unicode simples. Debido a que el conjunto de caracteres Unicode se compone de valores de 16 bits, el tipo de datos char se almacena en un entero sin signo de 16 bits.

Representan un único carácter y aparecen dentro de un par de comillas simples.

- Adicionalmente también vamos a trabajar con cadenas de caracteres. No son un tipo básico. Se tratan como una clase especial llamada **java.lang.String**.

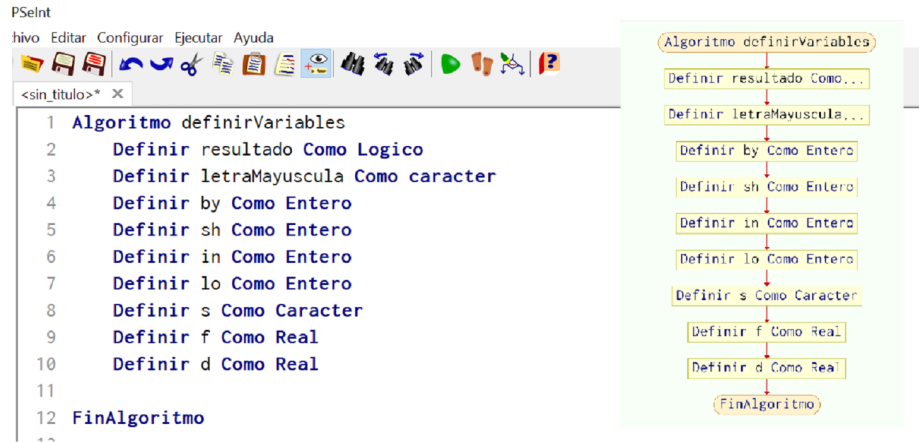


Figura 2.11: Definir variables en pseudocódigo y en diagramas de flujo

```

1
2 // Declaración de variables.
3
4   boolean resultado;
5
6   char letraMayuscula;
7
8   byte by;
9
10  short sh;
11
12  int in;
13
14  long lo;
15
16  String s;
17
18  float f;
19
20  double d;
  
```

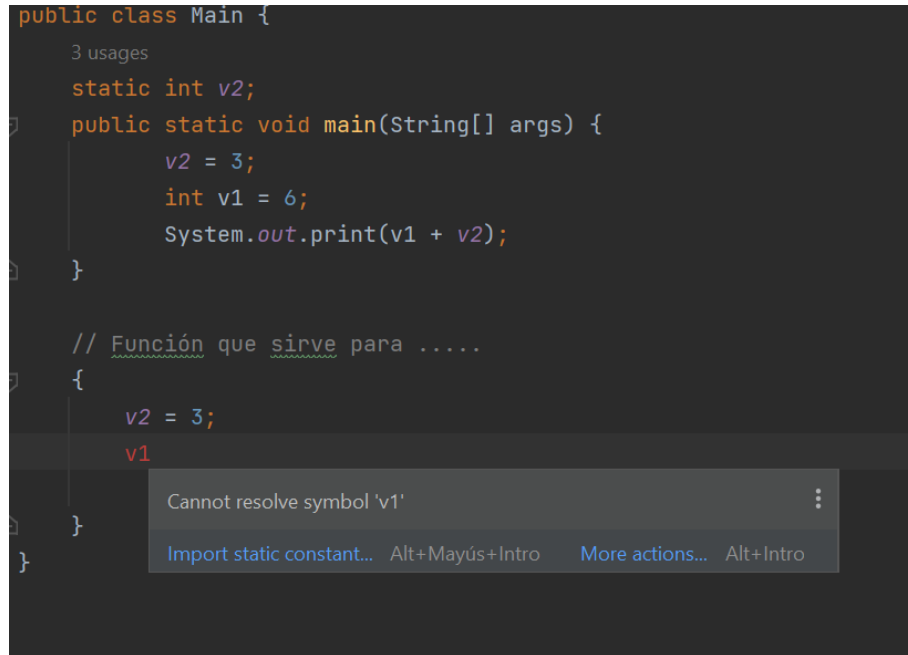


En java 10 se pueden crear variables en algunos casos con **var** sin indicar el tipo

2.5.3. Ámbito de una variable

El ámbito de una variable es el período de tiempo o zona en la que el programa almacena dicho valor en memoria. Como norma general, el ámbito de una variable es el bloque donde ha sido definida.

Los bloques siempre vienen delimitados por las llaves.



```

public class Main {
    3 usages
    static int v2;
    public static void main(String[] args) {
        v2 = 3;
        int v1 = 6;
        System.out.print(v1 + v2);
    }

    // Función que sirve para .....
    {
        v2 = 3;
        v1
    }
}

```

Cannot resolve symbol 'v1'

Import static constant... Alt+Mayús+Intro More actions... Alt+Intro

Figura 2.12: Ambito de las variables

2.6. Instrucciones de asignación

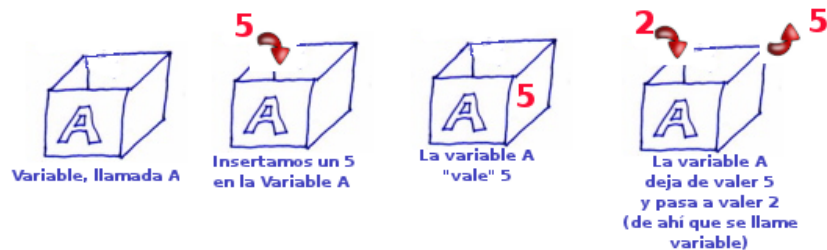


Figura 2.13: Asignar un valor

La asignación de un valor a una variable o constante se realiza a través del operador igual (=).

En el momento de la declaración es obligatorio inicializar una constante. En el caso de las variables es opcional, es decir, se puede realizar en el momento de la declaración o posteriormente dependiendo de la operación que se vaya a realizar con ella.

Ejemplos:

```

1
2      public static void main(String[] args) {
3
4          // Declaración de una variable de tipo byte sin asignación
5          byte primeraVariable;
6
7          // Declaración de una variable de tipo short sin asignación
8          short enteroPequeño;
9
10         // Declaración de una variable de tipo int asignando valor
11         int enteroNormal=655;
12
13         // Declaración de una variable de tipo long asignando valor
14         long enteroLargo=7000;
15
16         // Asignamos valor a variables después de haberla declarado
17         primeraVariable = 1;
18         enteroPequeño = 3;
19
20         // Declaramos tres variables de tipo int en una línea
21         int numeroUno, numeroDos, numeroTres;
22

```

```
23      // Declaramos dos variables de tipo int e inicializamos sólo una
24      int n1=3, n2 = 4;
25
26      // Declaración e inicialización de una variable de tipo float
27      // sufijo f para que considere el valor float y no double que es
        lo que supone por defecto
28      float decimalesPequeño = 2.3f;
29
30      // Declaración e inicialización de una variable de tipo double
31      double decimalLargo = 1234765.34;
32
33      // Declaración e inicialización de una variable booleana
34      boolean entero = false;
35
36      // Modificación del contenido de una variable
37      entero = true;
38
39      // Declaración e inicialización de una variable de tipo char
40      char letra = 's';
41
42      // Declaración e inicialización de una variable de tipo String
43      String nombre = "abc def";
44
45      // Declaración e inicialización de una constante de tipo float
46      final float DESCUENTO = 0.25f;
47
48      // DIFERENTES SISTEMAS NUMERICOS
49      //Valor 26, en decimal
50      int decVal = 26;
51
52      //Valor 26, en hexadecimal
53      int hexVal = 0x1a;
54
55      //Valor 26, en binario
56      int binVal = 0b11010;
57
58      // Desde Java 7 se puede usar el guión bajo para delimitar las partes
        de un literal
59      // Java no lo procesará como parte del literal
60      // El sufijo L significa Long
61      long l = 3_000_000_000L;
62      long numeroTarjeta = 1234_5678_9012L;
63
64      // VAR
65      var nuevaVariable = 6;
66
67      var otraVariable = "Pepe";
68    }
69 }
```

2.7. Operadores aritméticos

Tipo	Operador	Precedencia	Operación realizada
Prefix, postfix	-- ++	expr++ expr-- ++expr --expr	Incremento/Decremento en una unidad.
Unarios	+ -	+ -	Cambio de signo
Multiplicativos	* / %	* / %	Multiplicación, división y resto
Aditivos	+ -	+ -	Suma, resta
De movimiento	<< >> >>>	<< >> >>>	Desplazamiento a nivel de bits.

Figura 2.14: Operadores aritméticos

Un operador es un símbolo especial que indica que se debe efectuar una operación matemática o lógica.

Java reconoce los siguientes operadores aritméticos:

- + Suma
- - Resta
- * Multiplicación
- / División
- Resto o módulo %

Para resolver operaciones más complejas en java existe toda una librería de instrucciones o funciones matemáticas llamada **Math**.

Ejemplo:

```

1
2 public static void main(String[] args) {
3     //Declaración e inicialización de las variables numéricas
4     int n1 = 20;
5     int n2 = 3;
6
7     int suma = n1 + n2;
8     System.out.println("El resultado de la suma es: " + suma);
9
10
11     int resta = n1 - n2;
```



```
12      System.out.println("El resultado de la resta es: " + resta);
13
14      /*
15       * El resultado de la operación lo guardamos en una variable,
16       * sólo si nos interesa
17       */
18
19      System.out.println("El resultado de la multiplicación es: " + (n1
20                          * n2));
21
22      System.out.println("El resultado de la división es: " + (n1/n2));
23
24      System.out.println("El resto de la division es: " + (n1%n2));
25
26      System.out.println("El resultado de la potencia es: " +
27                          Math.pow(n1, n2));
28  }
```

El principal operador de asignación es el igual pero hay más operadores de asignación unidos a los aritméticos:

- `'+='`
op1 += op2 es lo mismo que op1 = op1 + op2
- `'-='`
op1 -= op2 es lo mismo que op1 = op1 - op2
- `'*='`
op1 *= op2 es lo mismo que op1 = op1 * op2
- `'/='`
op1 /= op2 es lo mismo que op1 = op1 / op2

2.7.1. Operadores aritméticos incrementales

Los operadores aritméticos incrementales son operadores unarios (un único operando). El operando puede ser numérico o de tipo char y el resultado es del mismo tipo que el operando. Estos operadores pueden emplearse de dos formas dependiendo de su posición con respecto al operando.

- `++` Incremento
i++ primero se utiliza la variable y luego se incrementa su valor
++i primero se incrementa el valor de la variable y luego se utiliza

- – decremento

```
1
2 int n1 = 5;
3 int n2 = ++n1;
4 System.out.println(n1 + " " + n2); // 6 6
5 int n3 = n1++;
6 System.out.println(n1 + " " + n3); // 7 6
```

2.7.2. Operadores a nivel de bits

Nosotros no los vamos a usar pero también existen operadores que desplazan los bits a la derecha o a la izquierda.

https://es.wikibooks.org/wiki/Programaci%C3%B3n_en_Java/Operadores_de_bits

2.8. Instrucciones de entrada/salida

En java para poder escribir (mostrar datos al usuario) se emplea **System.out** y para leer del teclado es necesario emplear **System.in**.

```
1
2  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
3
4  // Salida
5  System.out.println("Teclea lo que quieras");
6
7  // Entrada
8  String línea = br.readLine();
9
10 // Salida
11 System.out.println("Has tecleado " + línea);
```

Se crea un `InputStreamReader` a partir de `System.in` y se pasa al constructor de `BufferedReader`. Se crea un objeto `br` que nos permitirá leer las líneas tecleadas. Es necesario realizar un `import` de `java.io` para poder emplear esta lectura de líneas. Además la línea del `readLine` puede lanzar excepciones, es por ello que hay que meterla entre instrucciones `try/catch` para poder gestionar el posible error o relanzarla.

Otra manera mas simple de hacer más o menos lo mismo, se consigue utilizando la clase **Scanner** de la siguiente manera:

1. Importar la clase Scanner

```
1
2  import java.util.Scanner;
```

2. Crear un objeto Scanner

```
1
2  Scanner sc = new Scanner(System.in);
```

3. Usar métodos para leer diferentes tipos de datos. Los métodos más comunes del objeto Scanner son:

nextLine() Lee una línea completa de texto.

nextInt() Lee un número entero.

nextDouble() Lee un número con decimales (double).

nextBoolean() Lee un valor booleano (true o false).

4. Cerrar el scanner

```
1
2  sc.close();
```

Ejemplo:

```
1
2  // PRIMERA FORMA -- System.in
3
4  //1. creamos un objeto llamado "br" de la clase BufferedReader
5  BufferedReader br = new BufferedReader(new InputStreamReader (System.in));
6
7  //2. leemos una entrada con el metodo readLine(), de tipo String
8  System.out.println("Teclea tu apellido");
9  String apellido = br.readLine();
10 System.out.println(apellido);
11
12 //3. Tambien podemos realizar conversiones para trabajar con otros tipos
    de datos.
13 System.out.println("Teclea tu edad");
14 int edad = Integer.parseInt (br.readLine());
15 System.out.println(edad);
16
17
18 // SEGUNDA FORMA -- Clase Scanner
19
20 Scanner sc = new Scanner(System.in); //Se crea el lector
21
22 System.out.println("Teclea tu nombre");//Se pide un dato al usuario
23 String nombre = sc.nextLine(); //Se lee el nombre con nextLine()
24
25 System.out.println("Teclea tu edad");
26 edad = sc.nextInt(); // Se guarda la edad directamente con nextInt()
27
28 System.out.println("Hola " + nombre + " tienes " + edad + " años");
29
30 sc.close();
```

Existen formas más gráficas de llevar a cabo la entrada y salida de datos utilizando, por ejemplo, los cuadros de diálogo que nos proporciona **javax.swing.JOptionPane**. El método **showMessageDialog** nos permite mostrar una cadena de caracteres y el método **showInputDialog** recoger una cadena de caracteres.

Ejemplo:

```
1
2  import javax.swing.JOptionPane;
3
4  public class Main {
5
6      public static void main(String[] args) {
7          String datosEntrada;
8          datosEntrada = JOptionPane.showInputDialog(null,"Sumando uno: ");
9          int sumandoUno = Integer.parseInt(datosEntrada);
```

```
10
11     int sumandoDos=
        Integer.parseInt(JOptionPane.showInputDialog("Sumando dos:
        "));
12
13     int resultado = sumandoUno + sumandoDos;
14
15     JOptionPane.showMessageDialog(null, "El resultado de la suma es:
        "+ resultado);
16 }
17 }
```

2.9. Conversiones de tipo de datos (Castings)

En Java es posible transformar el tipo de dato de una variable u objeto en otro diferente al original. En muchos casos estos cambios se realizan de forma implícita, pero también podemos realizarlos de forma explícita. La conversión se lleva a cabo colocando el tipo destino entre paréntesis a la izquierda del valor que queremos convertir.

Ejemplo:

```
1
2 System.out.println("Teclea tu edad");
3 int edad = Integer.parseInt (br.readLine());
4 System.out.println(edad);
```

Además de los tipos de datos existen los métodos codificados dentro de cada una de las clases asociadas (también denominadas clases envoltorio) a los tipos de datos.

- **Tipo - Clase**
- byte – Byte
- short – Short
- int – Integer
- long – Long
- float – Float
- double – Double
- boolean – Boolean
- char – Character



Una Clase es un elemento de la programación orientada a objetos que ya veremos. De momento lo único que nos interesa es que nos aporta funcionalidades

```
1
2 int numeroUno = 7;
3 numeroUno. // No se puede poner nada tras el punto
4 int.       // No se puede poner nada tras el punto
```

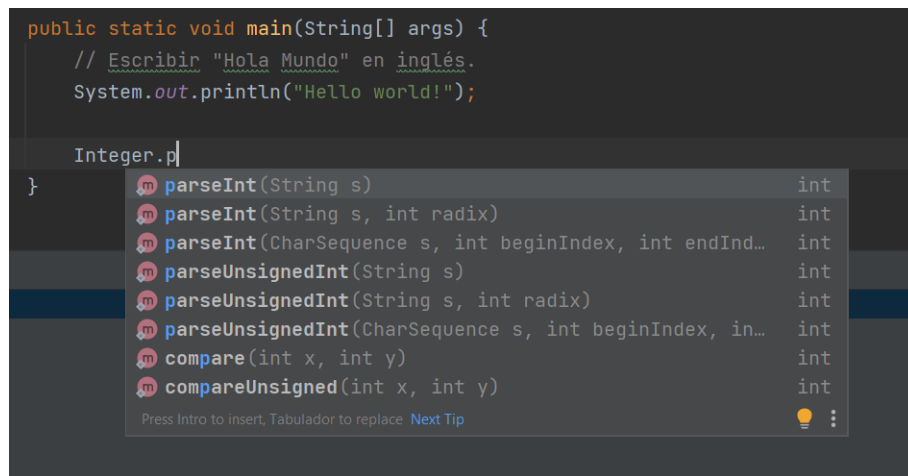


Figura 2.15: Métodos de la clase Integer