

Tema 2

Introducción a la programación en Java

Programación (Desarrollo de aplicaciones web) Carlos Alberto Cortijo Bon



Esta obra está bajo una [Licencia Creative Commons Atribución-NoComercial-SinDerivadas 4.0 Internacional](https://creativecommons.org/licenses/by-nc-nd/4.0/)

Índice

1. El lenguaje de programación Java	1
2. Programa mínimo en Java	2
3. Comentarios	3
4. Literales y variables	3
5. Tipos básicos en Java	4
6. Cadenas de caracteres	6
7. Operadores de Java	6
7.1. Operadores aritméticos	7
7.2. Operadores relacionales	8
7.3. Operadores lógicos	9
8. Entrada y salida	9
8.1. Lectura de datos desde la entrada estándar con la clase Scanner	10
8.2. Escritura de datos a la salida estándar	11
9. Gestión de excepciones	12
10. Argumentos de línea de comandos	15
11. Obtención de datos de diversos tipos a partir de argumentos de línea de comandos	17
12. Programación estructurada	20
12.1. Variantes de las sentencias condicionales	24
12.2. Sentencia de selección múltiple	26
12.3. Bucle for (para)	29
12.4. Bucle do ... while (Hacer ... Mientras)	32
13. Arrays en Java	34

En el tema anterior se han aprendido y utilizado los fundamentos de programación estructurada con diagramas de flujo y pseudocódigo, utilizando PSeInt. La programación estructurada se basa en la utilización de determinadas estructuras (a saber, secuencias, sentencias condicionales y bucles *while* o mientras) para construir cualquier programa.

En este tema se verá la programación estructurada utilizando el lenguaje Java.

En temas posteriores se aprenderá programación orientada a objetos en Java.

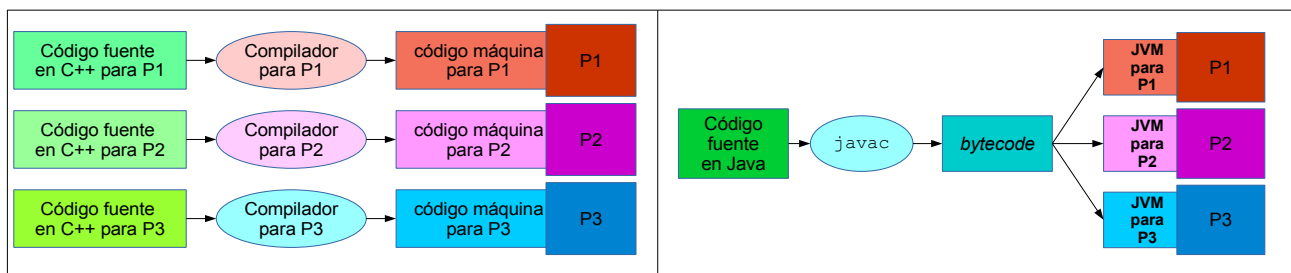
1. El lenguaje de programación Java

Java fue el primer lenguaje de programación de máquina virtual que tuvo un éxito masivo. Desde su introducción a mediados de los años 90, siempre ha sido uno de los lenguajes más ampliamente utilizados.

Java fue creado por la empresa Sun Microsystems. Hoy es propiedad de Oracle, que adquirió Sun Microsystems en 2010.

Java se creó para que los programas, una vez compilados, pudieran ejecutarse sin cambios en cualquier plataforma (esto es, en cualquier sistema operativo y sobre cualquier microprocesador). Esto se resume en las siglas WORA (*Write Once, Run Anywhere*, es decir: escribir una vez, ejecutar en cualquier lugar).

El lenguaje Java se compila a *bytecode*. Pero el *bytecode* no se ejecuta directamente en un procesador físico, sino en una máquina virtual de Java (JVM, *Java Virtual Machine*).



Compilación de programa en C++ y ejecución en diversas plataformas Compilación de programa en Java y ejecución en diversas plataformas

Se puede considerar, por tanto, que existe una **plataforma Java**, además del propio lenguaje de programación Java. Un programa en C++ se compila para producir código de máquina que se ejecuta en una plataforma particular (es decir, en un sistema operativo y un microprocesador determinado). Un programa que se escribe en Java, en cambio, se compila a *bytecode* que se ejecuta en una máquina virtual de Java.

Para saber más

La máquina virtual de Java es un programa desarrollado en lenguaje C. Por ello, es relativamente sencillo disponer de la máquina virtual de Java en cualquier plataforma. Basta con compilar la máquina virtual de Java para ella.

El *bytecode* es muy similar a un lenguaje ensamblador. Por ello, su ejecución es muy rápida, aunque lo ejecute un programa (la máquina virtual) y no un procesador físico. Se puede decir que el *bytecode* es un lenguaje ensamblador pero no para un microprocesador físico, sino para la máquina virtual de Java.

De hecho, se ha llegado a desarrollar microprocesadores físicos que ejecutan directamente *bytecode*.

La plataforma Java incluye:

- La máquina virtual de Java, que ejecuta *bytecode*.
- La biblioteca estándar de clases de Java, que incluye muchas clases de utilidad general.

Java se inspiró, en muchos aspectos, en el lenguaje de programación C++. Este se creó en los años 80 como una extensión del lenguaje C para incluir clases y permitir la programación orientada a objetos. La posibilidad de utilizar antiguo código en C en nuevos desarrollos facilitó la adopción masiva de la programación orientada a objetos. Pero muchos de los aspectos más complejos y problemáticos de C++ no están presentes en Java. Por ello, Java es un lenguaje más simple que C++.

La empresa Oracle es propietaria de la plataforma Java y del lenguaje de programación Java. Pero la plataforma Java es una plataforma abierta a la comunidad. El código fuente de la máquina virtual de Java (que está escrita en lenguaje C), está a disposición del público en general. Oracle proporciona implementaciones de la plataforma Java bajo licencia GPL en <https://jdk.java.net>. También hay implementaciones de la plataforma Java de otras empresas u organizaciones distintas de Oracle, y algunas de ellas son software libre. Las modificaciones se realizan de acuerdo a un procedimiento de la comunidad de Java o *Java Community Process*, abierto a la participación y a las contribuciones de otras empresas y organizaciones (<https://jcp.org/en/procedures/jcp2>).

Especificaciones de Java

Las especificaciones de la máquina virtual de Java están disponibles en <https://docs.oracle.com/javase/specs/>

Con cada nueva versión de la especificación se ha creado una nueva versión del lenguaje Java, cuyas especificaciones también están disponibles en la anterior dirección.

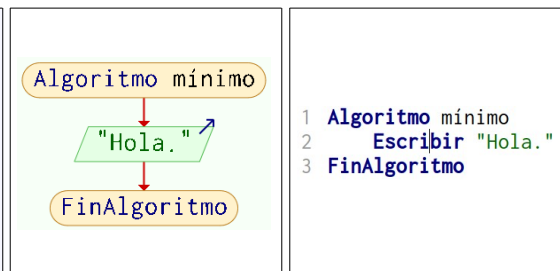
Estos documentos son muy extensos y detallados. Normalmente no se necesitará consultarlos.

Las especificaciones de la biblioteca estándar de clases para la versión 17 están en <https://docs.oracle.com/en/java/javase/17/docs/api>. Su consulta sí será útil con frecuencia, cuando sea necesario utilizar clases estándares de Java en los programas que se vayan desarrollando.

2. Programa mínimo en Java

El siguiente programa en Java es el típico programa que simplemente escribe un texto en la salida estándar. Se muestra junto a un diagrama de flujo que hace lo mismo, y con el pseudocódigo de PSeInt correspondiente.

```
package minimo;
public class Minimo {
    public static void main(String[] args) {
        System.out.println("Hola.");
    }
}
```



Cuando se ejecuta un programa en Java, se ejecuta lo que hay en el método **main**. La única línea de código que en este programa en Java hace realmente algo es **System.out.print("Hola.")**. Con ella, se envía hacia la salida estándar (**System.out**) el texto **"Hola."**.

El resto de las líneas definen unidades superiores: una clase a la que pertenece el método **main** y un paquete al que pertenece esta clase. El paquete se declara en una línea al principio. La clase y el método comienzan con una declaración a la que sigue el cuerpo, que está delimitado por una llave de inicio **{** y una llave de fin **}**. Más adelante se explicará el propósito de cada una de estas unidades.

1	<code>package minimo;</code>	Paquete al que pertenece la clase principal.
2	<code>public class Minimo {</code>	Clase principal, declaración e inicio.
3	<code> public static void main(String[] args) {</code>	Método main de la clase principal, declaración e inicio.
4	<code> System.out.println("Hola.");</code>	Instrucciones del método main de la clase principal
5	<code> }</code>	Fin de método main de la clase principal.
6	<code>}</code>	Fin de clase principal.

El método `main`, punto de entrada a un programa en Java

El método `main` es el punto de entrada a un programa de Java. Cuando este se ejecuta, se ejecuta el método `main`.

Debe tener un parámetro `String[] args`, que es un *array* de cadenas de caracteres (`String`) que contiene los argumentos de línea de comandos.

El programa anterior podría escribir un mensaje personalizado si se pasa un nombre como argumento de línea de comandos. Se muestra en el siguiente ejemplo. Por sencillez se ha omitido la declaración del paquete (`package`).

`args.length` contiene el número de parámetros de línea de comandos. Cuando hay al menos uno, se toma el primero como nombre para mostrar en un saludo personalizado. Las dos ramas de una sentencia condicional (`if ... else`) están encapsuladas entre una llave de apertura `{` y una de cierre `}`.

```
public class Minimo {  
  
    public static void main(String[] args) {  
        if(args.length < 1) {  
            System.out.println("Hola.");  
        } else {  
            System.out.printf("Hola, %s\n.", args[0]);  
        }  
    }  
}
```

Cuando se proporciona algún parámetro de línea de comandos (si `args.length` es mayor o igual que 1), se utiliza `printf` para escribir un texto en el que se sustituye `%s` por el argumento proporcionado a continuación, es decir, `args[0]`, que contiene el valor del primer parámetro de línea de comandos. El carácter `'\n'` representa un salto de línea. `println` escribe siempre un salto de línea al final. Pero `printf` no lo hace, y por eso se añade al final `'\n'`.

3. Comentarios

Existen dos tipos de comentarios en Java.

- Los del primer tipo comienzan con `/*` y terminan con `*/`. Pueden incluir más de una línea.
- Los del segundo tipo comienzan con `//` y llegan hasta el final de la línea.

Se pueden añadir comentarios de ambos tipos al programa mínimo anterior.

```
public class Minimo {  
  
    /*  
    Este es un programa mínimo que escribe un texto.  
    El método main es el punto de entrada del programa.  
    */  
  
    public static void main(String[] args) {  
        System.out.println("Hola."); // System.out es la salida estándar  
    }  
}
```

4. Literales y variables

Los principales lenguajes de programación permiten utilizar los siguientes tipos de datos:

- Números enteros. Por ejemplo: 4, 0 ó -10.

- Números reales. Por ejemplo: 5.43 ó 3.1416.
- Caracteres. Por ejemplo: 'B', 'e', '4', '+', 'é', 'ñ', 'è', 'â'.
- Cadenas de caracteres. Por ejemplo: "Buenos días" o "verde".

En cualquier lenguaje de programación se puede distinguir entre literales y variables.

- **Literales.** Un literal es un valor constante. Por ejemplo: 4, 5.43 o "Buenos días". Existen literales de todos los tipos básicos anteriores. El valor 4, por ejemplo, es un valor entero, es decir, un número sin parte decimal. El valor 4.53, en cambio, es un valor real, es decir, un número que tiene una parte decimal. También sería un valor real 4.0, y su parte decimal sería 0. El valor "Buenos días" es un literal que representa una cadena de caracteres, que consiste en una secuencia de caracteres individuales: 'B', 'u', 'e', 'n', 'o', 's', ' ', 'd', 'í', 'a', 's'.
- **Variables.** Una variable tiene asociado un nombre y contiene un valor. Por ejemplo, una variable peso puede tener el valor 89.26, y una variable edad el valor 32.



Ilustración 2.1: Almacenamiento de variables en memoria

Cada nombre de variable tiene asociada una dirección de memoria, en la que se almacena el valor de la variable.

5. Tipos básicos en Java

Los tipos básicos de Java permiten representar valores que ocupan un espacio fijo en memoria, diferente para cada tipo básico. El espacio que ocupa un dato en memoria se mide en bits o bytes (1 byte = 8 bits).

Ni en Java ni en ningún otro lenguaje de programación existe un tipo básico para las cadenas de caracteres como, por ejemplo, "Buenos días". Esto es porque no tienen una longitud fija y, por tanto, no siempre ocupan el mismo espacio en memoria. Para representar cadenas de caracteres, en Java existe la clase **String**.

Los tipos básicos de Java se pueden clasificar de la siguiente manera.

- **Numéricos.**
 - **Enteros.** Los valores de este tipo representan un número entero, que puede ser positivo, cero o negativo. Los diferentes tipos se diferencian en el número de bits que ocupan, y de esto depende el rango de números enteros que pueden representar. Con n bits se pueden representar números en el rango de valores de -2^{n-1} a $2^{n-1}-1$. Por ejemplo, con un byte (8 bits) se pueden representar los números de -2^7 a 2^7-1 , es decir, de -128 a 127. Los tipos enteros son los siguientes.
 - Entero (**int**). Los valores de este tipo ocupan 4 bytes (32 bits).
 - Entero largo (**long**). Los valores de este tipo ocupan 8 bytes (64 bits).
 - Entero corto (**short**). Los valores de este tipo ocupan 2 bytes (16 bits).
 - Byte (**byte**). Los valores de este tipo ocupan 1 bytes (8 bits).
 - **Reales.** Los valores de este tipo representan números no enteros, es decir, con decimales. El rango de números que pueden representar y la precisión con que los pueden representar dependen del número de bits que se utilizan para ello. Los tipos reales son los siguientes.
 - Real de precisión simple (**float**). Los valores de este tipo ocupan 4 bytes (32 bits).

- Real de precisión doble (**double**). Los valores de este tipo ocupan 8 bytes (64 bits).
- **Carácter** (**char**). Los valores de este tipo representan un carácter. Ocupan 2 bytes (16 bits).
- **Booleano** (**boolean**). Los valores de este tipo representan un valor de verdad y son solo dos: **true** (cierto) y **false** (falso). Cada uno de ellos se representa internamente como un valor **int**. Pero para un *array* de elementos de este tipo se puede utilizar una representación más compacta, que utiliza menos bits.

En Java hay que declarar todas las variables. Cada variable tiene un tipo determinado, que se especifica en su declaración. En la declaración de una variable se puede, además, asignar un valor inicial. El valor de la variable puede cambiar a lo largo del tiempo, pero siempre debe ser del mismo tipo.

Para asignar un valor a una variable, se escribe el nombre de la variable, seguido del carácter = y del valor que se le asigna, como se muestra en los siguientes ejemplos.

Se podrían declarar variables para el peso y la edad de una persona de la siguiente forma en Java.

```
double peso = 89.26;  
int edad = 32;
```

Otra posibilidad sería declarar primero las variables y después asignarles un valor.

```
double peso;  
int edad;  
peso = 89.26;  
edad = 32;
```

Una vez que una variable se define de un tipo determinado este no puede cambiar, y solo se le pueden asignar valores de ese tipo.

Con el lenguaje Java se asigna un valor por defecto a todas las variables, en caso de que no se les asigne un valor al declararlas. Este valor es el siguiente:

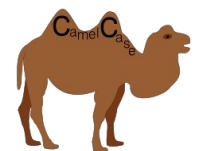
- Para tipos numéricos, el valor 0.
- El tipo **char** es en realidad un tipo numérico y, por tanto, su valor por defecto es 0, cuya representación como literal es `'\u0000'`.
- Para el tipo **boolean**, el valor **false**.
- Para la clase **String**, el valor **null**. Este es un valor especial que no representa ninguna cadena de caracteres, sino el hecho de que no se ha asignado un valor a la variable.

Nombres de paquetes, clases y variables en Java

Para la programación en Java se siguen unos convenios de nomenclatura para todos los identificadores.

Deben empezar con una letra o con el carácter especial `_`. Los siguientes caracteres pueden ser también números.

- Los nombres de paquetes deben contener solo letras minúsculas.
- Los nombres de las clases (en el ejemplo anterior `Minimo`, `String` y `System`) deben seguir la notación *UpperCamelCase* o en forma de joroba de camello. Otros nombres válidos serían `InputStream`, `RedNeuronal`, o `FichaDomino`. Más adelante se aprenderá más acerca de las clases.



UpperCamelCase

- Los nombres de las variables deben seguir la notación *lowerCamelCase* o en forma de joroba de dromedario.
 - El primer carácter, como ya se ha dicho, es una letra. Este convenio establece que debe ser minúscula. Nombres válidos de variables serían `a`, `b`, `x`, `id`, `nom`, `nombre`.
 - Si en el nombre de la variable hay más de una palabra, la letra inicial debe estar en mayúsculas a partir de la segunda palabra. Nombres válidos de variables serían `xMin`, `iCliente`, `numClientes`, `nClientes`, `maxVol`.



Se pueden utilizar abreviaturas para hacer más cortos los nombres: `numClientes` o `nClientes` en lugar de, y preferiblemente `a`, `numeroClientes`, por ejemplo.

Los identificadores deben ser descriptivos para facilitar la comprensión del programa. El nombre de las variables debe describir lo más exacta y específicamente que sea posible el contenido de las variables.

6. Cadenas de caracteres

Las cadenas de caracteres se representan en Java con la clase **String**. Los valores literales de este tipo se representan entre comillas dobles. Por ejemplo, `"Hola."` es una cadena de caracteres formada por los caracteres `'H'`, `'o'`, `'l'`, `'a'` y `'.'`.

El siguiente programa de ejemplo utiliza la operación de concatenación de cadenas `+` para construir una cadena a partir de otras preexistentes.

```
public static void main(String[] args) {  
    String saludo = "Hola";  
    String nombre = "Ramón";  
  
    String saludoPers = saludo + ", " + nombre + ".";  
  
    System.out.println(saludoPers);  
    System.out.println(saludoPers.length());  
}
```

El anterior programa utiliza también el método `length` de la clase **String** para obtener la longitud de la cadena `saludoPers`. Más adelante se aprenderá más acerca de las clases y sus métodos.

7. Operadores de Java

Existen en Java, como en general en todos los lenguajes de programación, distintos tipos de operadores, con los que se pueden construir expresiones que se evalúan para obtener un valor. Las expresiones aparecen:

- a) A la derecha de una sentencia de asignación. La expresión debe dar como resultado un dato de un tipo compatible con el de la variable a la que se le asigna el valor.

```
double desc = precio * porcDesc / 2;  
String saludoPers = saludo + ", " + nombre + ".";
```

- b) Con sentencias condicionales y en bucles. La expresión debe dar como resultado un valor de tipo `boolean` (`true` o `false`).

```
if (args.length < 1) { ... }
```



```
while (i < n) { ... }
```

En los siguientes apartados se explican los siguientes tipos de operadores.

7.1. Operadores aritméticos

Los operadores aritméticos permiten realizar operaciones matemáticas elementales.

Operador	Descripción	Ejemplos de uso
+	Suma	<code>horasTot = horasNorm + horasExtra;</code>
-	Resta	<code>prFinal = precio - desc;</code>
*	Multiplicación	<code>precio = prUnit * cant;</code>
/	División	<code>prPers = precio / numPers;</code>
%	Resto o módulo de una división entera	<code>resto = dividendo % divisor;</code>

No existe un operador para la potencia en Java que permita calcular directamente, por ejemplo, 2^8 o $1,5^3$. Esto no significa, por supuesto, que no se pueda hacer fácilmente. Más adelante se verá cómo.

Para todos los operadores anteriores se puede utilizar una forma abreviada cuando el primero de los dos operandos es la misma variable a la que se le está asignando el resultado de la operación. Se ponen a continuación algunos ejemplos.

Expresión	Equivalente a
<code>nImpar += 2;</code>	<code>i = i + 2;</code>
<code>precio -= desc;</code>	<code>precio = precio - desc;</code>
<code>pot2 *= 2;</code>	<code>pot2 = pot2 * base;</code>
<code>n /= 2;</code>	<code>n = n / 2;</code>
<code>p %= q;</code>	<code>p = p % q;</code>

Los **operadores de incremento y decremento** se utilizan mucho en la práctica para incrementar y decrementar variables numéricas de tipo entero.

Expresión	Equivalente a
<code>c++;</code>	<code>c = c + 1;</code>
<code>i--;</code>	<code>i = i - 1;</code>

También se pueden utilizar dentro de expresiones, como en el siguiente ejemplo.

```
int factorial = 1;
while (n > 1) {
    factorial *= n--;
}
```

En cada iteración del bucle, se hace lo siguiente, y en este orden:

1. Se multiplica el valor de `factorial` por el valor de `n`.
2. Se decrementa el valor de `n`.

Los siguientes programas de ejemplo realizan diversos cálculos con datos numéricos y escriben el resultado. El primero con datos de tipo entero (**int**) y el segundo con datos de tipo real (**double**). Por sencillez se ha omitido la declaración del paquete y de la clase que contiene el método **main**, y se hará en los siguientes ejemplos, en general.

De igual manera que en el primero se utiliza el tipo **int** (entero) se podría haber utilizado el tipo **long** (entero largo) o **short** (entero corto). Y de igual manera que en el segundo se ha utilizado el tipo **double** (número real con doble precisión) se podría haber utilizado el tipo **float** (número real con precisión simple).

```
public static void main(String[] args) {
    int filas = 12;
    int columnas = 14;

    int casillas = filas * columnas;

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

```
public static void main(String[] args) {
    double base = 5.4;
    double altura = 6.25;

    double areaTriang = base * altura / 2;

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

Hay que tener en cuenta que la división entre dos números enteros da como resultado el cociente entero entre ambos. Si se quiere obtener el cociente con decimales, debe convertirse antes de realizar la división a un número de tipo **double** o **float**. Los siguientes ejemplos ilustran todo esto. Se muestra resaltado el código que realiza la conversión a un número de tipo **double** antes de realizar la división. Esta manera de convertir un valor de un tipo al valor correspondiente en otro tipo se llama *casting* en inglés.

```
int pizzas = 4;
int pers = 5;

System.out.println(pizzas / pers);

double racionPers = pizzas / pers;
System.out.println(racionPers);
```

0
0.0

```
int pizzas = 4;
int pers = 5;

double racionPers = (double) pizzas / pers;
System.out.println(racionPers);
```

0.8

7.2. Operadores relacionales

Los operadores relacionales permiten verificar si se cumplen determinadas relaciones entre dos números. Se suelen utilizar en sentencias condicionales **if**, como ya se ha visto en un ejemplo anterior:

```
if (args.length < 1) {
    System.out.println("Hola.");
} else {
    System.out.printf("Hola, %s\n.", args[0]);
}
```

Operador	Descripción	Ejemplo (en sentencia condicional if o en bucle while)
==	Igual	if (pos == 0) { ... }
<	Menor	if (n < min) { ... }
<=	Menor o igual	if (n <= max) { ... }
>	Mayor	if (n > max) { ... }
>=	Mayor o igual	if (n >= min) { ... }
!=	Distinto	if (ind != -1) { ... }

Atención

El operador de igualdad es ==, no =.

7.3. Operadores lógicos

Los operadores lógicos permiten crear nuevas condiciones a partir de otras.

Operador	Descripción	Ejemplo (con sentencia condicional <code>if</code> o con bucle <code>while</code>)
<code>&&</code>	Conjunción lógica. <code>c1 && c2</code> se cumple si se cumplen ambas condiciones <code>c1</code> y <code>c2</code> .	<code>if (x >= xMin && x <= xMax) { ... }</code>
<code> </code>	Disyunción lógica. <code>c1 c2</code> se cumple si se cumple cualquiera de las condiciones <code>c1</code> o <code>c2</code> .	<code>if (x < xMin x > xMax) { ... }</code>
<code>!</code>	Negación lógica. <code>!cond</code> se cumple si no se cumple la condición <code>cond</code> . <code>cond</code> puede ser cualquier condición, pero suele ser una variable de tipo <code>boolean</code> , es decir, que puede tomar valores <code>true</code> o <code>false</code> .	<pre>boolean fin = false; while(!fin) { ... if (...) { fin = true; } ... }</pre>

8. Entrada y salida

Los programas de ejemplo vistos hasta ahora muestran información en la salida estándar con `System.out.println` o `System.out.printf`. Por ejemplo el siguiente. La salida estándar suele estar asociada a la pantalla, con lo que cualquier cosa que se escriba en ella se mostrará en la pantalla.

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

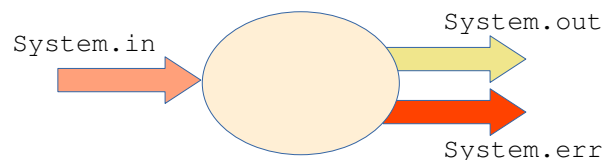
    double base = 5.4;
    double altura = 6.25;

    double areaTriang = base * altura / 2;

    System.out.println(areaTriang);

}
```

Pero este programa es bastante limitado. Lo interesante es que el usuario pueda introducir los valores de la base y de la altura. De igual manera que un programa puede escribir en su salida estándar (**System.out**), también puede leer de su entrada estándar (**System.in**). La entrada estándar suele estar asociada, por defecto, al teclado.



Para saber más: la salida de error

La salida de error (**System.err**) se suele utilizar para escribir mensajes de error cuando, por algún motivo que escape al control del usuario y del programa, se produce un fallo en una operación que se está intentando realizar. Por ejemplo, cuando se está intentando escribir información en un fichero y se agota el espacio en el sistema de ficheros, o cuando se está intentando obtener datos de una página web pero no se puede porque falla la conexión de red. No se utilizará en los programas sencillos con los que se va a trabajar a continuación. Suele estar asociada, por defecto, a la pantalla, al igual que la salida estándar.

8.1. Lectura de datos desde la entrada estándar con la clase Scanner

Para leer de manera sencilla datos de diferentes tipos desde la entrada estándar (**System.in**), se puede construir un **Scanner** sobre ella. Esta clase tiene métodos que permiten obtener datos de los tipos básicos antes explicados y de la clase **String**.

Tipo	Método
String	<code>nextLine()</code>
int	<code>nextInt()</code>
long	<code>nextLong()</code>
float	<code>nextFloat()</code>
double	<code>nextDouble()</code>
boolean	<code>nextBoolean()</code>

El siguiente programa lee un número entero desde la entrada estándar (**System.in**) y escribe el doble del número introducido. Para ello, primero construye un **Scanner** asociado a ella con `new Scanner(System.in)`. La sentencia `import java.util.Scanner` del principio es necesaria para poder utilizar la clase **Scanner**. Más adelante, cuando se explique la programación orientada a objetos, se explicarán todos estos detalles. Pero por ahora no son necesarios para entender el funcionamiento de este programa y de los siguientes ni para crear programas similares.

Con `num = s.nextInt()` se lee un número entero de la entrada estándar (normalmente asociada al teclado), y se asigna su valor a la variable `num`.

```
package leeentero;

import java.util.Scanner;

public class LeeEntero {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        System.out.printf("Introduce número entero: ");
        int num = s.nextInt();
        System.out.printf("Número: %d\n", num);

    }

}
```

Atención: separadores de decimales y de miles para la clase Scanner

En los lenguajes de programación se utiliza un punto como separador de decimales. Se ha visto en ejemplos anteriores (por ejemplo, `double base = 5.4`). En los países de habla inglesa se utiliza el punto como separador de decimales. Cuando se lee una cadena de caracteres de la entrada estándar y se intenta convertir a un tipo `double` o `float`, se asume un separador de decimales que depende de la configuración regional del sistema operativo. En el caso de España, por ejemplo, el separador de decimales es una coma. Por lo tanto, debe introducirse 5,54 y no 5.54 para un valor de 5 unidades y 54 centésimas.

Además hay que tener en cuenta que, según la configuración regional para España, un punto se interpretará como separador de miles. Si está en un lugar apropiado para ello, sencillamente se ignorará. Pero si no, se lanzará una excepción de tipo `InputMismatchException`. 5.542 se interpretará como 5542, mientras que 5.54 provocará una excepción de tipo `InputMismatchException`.

Actividad 2.1

Crea un programa que lea dos números enteros y escriba su suma.

Actividad 2.2

Crea un programa que lea dos números reales y escriba su suma. Debe funcionar correctamente cuando se introducen números con dígitos decimales.

8.2. Escritura de datos a la salida estándar

Existen varios métodos para escribir datos en la salida estándar (`System.out`).

print	<p>Tiene un único argumento, que es un dato que se escribe en la salida estándar. No realiza un salto de línea al final. Lo que se escriba a continuación aparecerá en la misma línea. Se muestra un ejemplo a continuación.</p> <div data-bbox="288 1155 1434 1288"> <pre>System.out.print(" (" + a + ", " + b + ")");</pre> (5.4, 8.0) </div> <p>Es bastante engorroso escribir cada parte por separado. Se puede utilizar el operador <code>+</code> para concatenación de cadenas para formar una única cadena de caracteres y después escribirla. De todas formas, hay una solución mejor, y es utilizar printf, que se verá en breve.</p> <div data-bbox="288 1413 1434 1451"> <pre>System.out.print("(" + a + ", " + b + ")");</pre> (5.4, 8.0) </div> <p>Si se quisiera escribir otro par de números en la siguiente línea, habría que escribir antes un salto de línea, representado por el carácter <code>'\n'</code>, como se muestra en el siguiente ejemplo.</p> <div data-bbox="288 1547 1434 1727"> <pre>double a = 5.4; double b = 8; System.out.print("(" + a + ", " + b + ")"); System.out.print("\n"); double c = 4.3; double d = 6.73; System.out.print("(" + c + ", " + d + ")");</pre> (5.4, 8.0) ↵ (4.3, 6.73) </div>
println	<p>Funciona igual que print, con la única diferencia de que se escribe un salto de línea al final, de manera que lo siguiente que se escriba, se escribirá en la línea siguiente.</p> <div data-bbox="288 1805 1434 1841"> <pre>System.out.println("(" + a + ", " + b + ")");</pre> (5.4, 8.0) ↵ </div>

printf	<p>Permite especificar un formato o patrón para la escritura de la información. Este es una cadena de caracteres en la cual se insertan determinados marcadores o <i>placeholders</i>. Estos especifican lugares en el que se escribirá determinada información, cuyo valor se proporciona a continuación, en argumentos adicionales. Debe haber un argumento a continuación para cada marcador, y debe ser del tipo especificado en el marcador. Los principales marcadores, cada uno para un tipo de datos distinto, son los siguientes:</p> <ul style="list-style-type: none"> %s Cadena de caracteres %d Número entero. Puede usarse con los tipos int y long. %f Número real, con decimales. Puede usarse con los tipos double y float. %c Carácter. Puede utilizarse con el tipo char. %b Valor booleano, que puede tomar valores true o false. <p>Como ejemplo de uso, valga el siguiente.</p> <pre>int i = 102; String t = "Hola"; System.out.printf("Número: %d, texto: %s%s\n", i, t, ".");</pre> <p>Número: 102, texto: Hola.</p> <p>Es preferible utilizar printf para escribir un texto compuesto de varias partes, algunas de las cuales son valores de variables. Para escribir un valor literal o el valor de una variable, se puede utilizar println o print, dependiendo de si se quiere un salto de línea después.</p>
---------------	---

9. Gestión de excepciones

Si al ejecutar el programa de ejemplo anterior, que leía un número entero y lo mostraba, no se introduce un número entero, `s.nextInt()` lanza una excepción de tipo `InputMismatchException`. En general, todos los métodos `nextXXX()` lanzan una excepción de tipo `InputMismatchException` si no se introduce un texto que representa un dato del tipo `XXX`.

```
run:
Introduce número entero: dos
Exception in thread "main" java.util.InputMismatchException (1)
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)
    at java.base/java.util.Scanner.next(Scanner.java:1594)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
    at leeentero.LeeEntero.main(LeeEntero.java:12) (2)
/home/carlos/.cache/netbeans/19/executor-snippets/run.xml:111: The following error occurred while executing this line:
/home/carlos/.cache/netbeans/19/executor-snippets/run.xml:68: Java returned: 1
BUILD FAILED (total time: 4 seconds)
```

Ilustración 2.2: Excepción de tipo `InputMismatchException` cuando no se introduce un valor del tipo esperado

En la línea (1) se muestra el tipo de excepción que se ha producido (`InputMismatchException`).

En la línea (2) se indica la línea del código fuente de nuestro programa en la que se ha producido la excepción. Se puede pulsar en el enlace para que se muestre dicha línea.

```

7  |
8  |
9  |
10 |
11 |
12 | System.out.printf("Introduce número entero: ");
13 | int num = s.nextInt();
14 | System.out.printf("Número: %d\n", num);
15 | }
```

Esta excepción se puede capturar y gestionar para mostrar un mensaje de error apropiado y terminar la ejecución del programa. Para ello se incluye la instrucción donde se puede producir esta excepción dentro de un bloque `try { ... } catch (InputMismatchException ex) { ... }`. En el bloque

catch se muestra el valor incorrecto, que se obtiene con `s.next()`, y después se termina la ejecución del programa con **return**.

Nota: cuando se produce una excepción de tipo `InputMismatchException`, la cadena de caracteres que se ha leído que no representa un dato del tipo esperado (`int` para `nextInt()`, `double` para `nextDouble()`, etc), se devuelve a la entrada estándar. De esa manera, se puede recuperar para utilizarla para lo que se quiera, por ejemplo para mostrarla en un mensaje de error.

La sentencia **return**

La sentencia `return` hace que termine inmediatamente la ejecución del método `main` y, por lo tanto, del programa.

```
package leetiposbasicos;

import java.util.InputMismatchException;
import java.util.Scanner;

public class LeeTiposBasicos {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        // Leer texto
        System.out.print("Introduce texto: ");
        String linea = s.nextLine();
        System.out.printf("Texto: %s\n", linea);

        // Leer número entero
        System.out.printf("Introduce número entero: ");
        int num;
        try {
            num = s.nextInt();
        } catch (InputMismatchException e) {
            System.out.printf("ERROR: El valor introducido (%s) no es un entero.\n",
                s.next());
            return;
        }
        System.out.printf("Número entero: %d\n", num);

        // Leer número real
        System.out.printf("Introduce número real: ");
        double numRealD;
        try {
            numRealD = s.nextDouble();
        } catch (InputMismatchException e) {
            System.out.printf("ERROR: El valor introducido no es un número real.\n",
                s.next());
            return;
        }
        System.out.printf("Número real: %f\n", numRealD);
    }
}
```

Ámbito de las variables

El contenido de un fichero de código fuente de Java consiste, excepto la parte del principio en la que están las directivas **package** e **import**, en bloques de código dentro de otros bloques de código. Los bloques de código empiezan por una palabra reservada (como **class**, **if**, **while**, **for**, etc) que es el principio de una primera parte, a la que sigue un cuerpo delimitado entre por una llave de apertura (**{**) y una de cierre (**}**).

Los bloques de código de más alto nivel son las clases (**class**). Dentro de un bloque de código pueden haber otros. De esta forma, el código fuente de una clase se puede ver como una caja dentro de la que pueden haber más cajas, dentro de las que, a su vez, pueden haber más cajas.

El **ámbito de una variable** es el bloque de código en el que la variable está definida. Una variable solo existe dentro de su ámbito, y por tanto solo se puede dentro de él.

Este es el motivo por el que se ha tenido que sacar la declaración de las variables **num** y **numReal** fuera del bloque **try** en el que se les asigna un valor, para que se puedan utilizar cuando más adelante se necesitan, fuera de él, para escribir su valor con **printf**.

```
package leetiposbasicos;

import java.util.InputMismatchException;
import java.util.Scanner;

public class LeeTiposBasicos {

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        // Leer texto
        System.out.print("Introduce texto: ");
        String linea = s.nextLine();
        System.out.printf("Texto: %s\n", linea);

        // Leer número entero
        System.out.printf("Introduce número entero: ");
        int num;
        try {
            num = s.nextInt();
        }
        catch (InputMismatchException e) {
            System.out.printf("ERROR: El valor introducido (%s) no es un entero.\n", s.next());
            return;
        }
        System.out.printf("Número entero: %d\n", num);

        // Leer número real
        System.out.printf("Introduce número real: ");
        double numRealD;
        try {
            numRealD = s.nextDouble();
        }
        catch (InputMismatchException e) {
            System.out.printf("ERROR: El valor introducido no es un número real.\n",
                s.next());
            return;
        }
        System.out.printf("Número real: %f\n", numRealD);
    }
}
```


Actividad 2.3

Modifica los programas creados para las dos actividades anteriores (el que pide dos números enteros y muestra su suma y el que pide dos números reales y muestra su suma) para que funcionen correctamente cuando se introduce cualquier dato incorrecto. En ese caso, debe mostrarse un mensaje de error y terminar inmediatamente la ejecución del programa.

Actividad 2.4

Crear un programa en Java que pida un número de horas trabajadas y la paga por hora, y que calcule la paga total para el número de horas trabajadas.

Actividad 2.5

Crear un programa en Java que pida un número entero y que escriba un mensaje indicando si es par o impar. Para ello, debe calcular el resto de dividir entre dos.

Actividad 2.6

Crear un programa en Java que lea una cantidad en euros (€) y muestre la cantidad equivalente en pesetas (ptas). La equivalencia es 1€=166,386 ptas. Para la cantidad en euros hay que admitir decimales. Para no complicar el programa, se permitirá introducir cantidades con más de dos dígitos decimales. El resultado se debe redondear a una cantidad entera. Si la parte decimal es mayor o igual que 0,5 debe redondearse por exceso, y si no por defecto. Por ejemplo:

$2€ = 2 \times 166,386 \text{ ptas} = 332,772 \text{ ptas} \approx 333 \text{ ptas}$ (redondeando por exceso)

$3€ = 3 \times 166,386 \text{ ptas} = 499,158 \text{ ptas} \approx 499 \text{ ptas}$ (redondeando por defecto)

El programa debe mostrar la equivalencia en pesetas antes y después del redondeo. Si por ejemplo se da una cantidad de 3,52€, se mostraría una línea como la siguiente, salvo pequeñas variaciones en el formato, como el número de decimales que se muestran:

$3,52€ = 3,52 \times 166,386 \text{ ptas} = 585,67872 \text{ ptas} \approx 586 \text{ ptas}$

Nota: El redondeo a una cantidad entera se puede hacer con un código como este:

```
int ptasRed = (int) Math.round(ptas);
```

Con el siguiente código, se obtendría en `ptasRed` el resultado de truncar `ptas`. Es decir, de eliminar la parte decimal. No es lo que se quiere.

```
int ptasRed = (int) ptas;
```

Por cierto, una manera ingeniosa de redondear, que hace innecesario el uso de `Math.round`, es la siguiente.

```
int ptasRed = (int) (ptas + 0.5);
```

Actividad 2.7

Crear un programa que pida un número de horas trabajadas en días laborables y un número de horas trabajadas en días festivos, y la paga por hora trabajada en día laborable. Para la paga deben admitirse decimales. El programa debe calcular y mostrar la paga total. Por las horas trabajadas en festivo se paga un 60% más que en día laborable.

10. Argumentos de línea de comandos

Un programa puede también obtener información de los argumentos de línea de comandos.

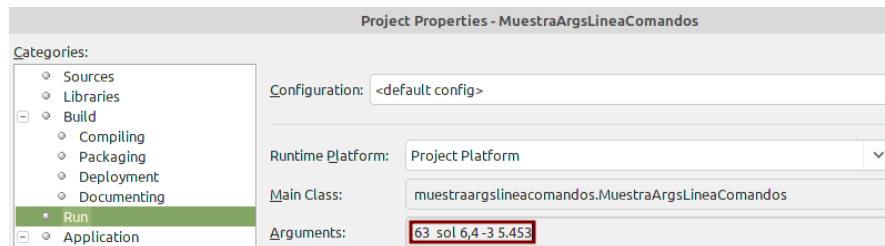
Cuando se ejecuta un programa desde línea de comandos, los argumentos de línea de comandos se especifican después del nombre del programa. Por ejemplo, cuando se ejecuta en un intérprete de línea de comandos de Linux el siguiente comando:

```
$ ls -ltrc -F /etc
```

El programa ejecutado es `ls`. El resto (`-ltrc -F`) son argumentos de línea de comandos.

Los programas de Java también pueden obtener los valores de los argumentos de línea de comandos que se les pasan. Cuando se ejecuta el programa desde un IDE o entorno integrado de desarrollo hay opciones en el entorno para indicar valores para estos argumentos.

En NetBeans se pueden indicar valores para los argumentos de línea de comandos desde las propiedades del proyecto. Para ello, se pulsa con el botón derecho del ratón sobre el proyecto, y se selecciona la opción *Properties*. Dentro de la caja de diálogo que aparece, se pueden indicar los argumentos de línea de comandos en el apartado *Run*, en la opción *Arguments*, como se muestra en el siguiente ejemplo.



El siguiente programa de ejemplo muestra los valores de todos los argumentos de línea de comandos. Se puede obtener el número de argumentos de línea de comandos de `args.length`. Sus valores se pueden obtener de `args[0]`, `args[1]`, ..., `args[args.length-1]`.

```
package muestraargslíneacomandos;

public class MuestraArgsLineaComandos {

    public static void main(String[] args) {

        System.out.printf("%d argumentos de línea de comandos.\n", args.length);

        int i = 0;

        while(i < args.length) {
            System.out.printf("Argumento %d: %s\n", i, args[i]);
            i++;
        }

    }

}
```

La salida del anterior programa, cuando se proporcionan los argumentos de líneas de comandos mostrados en el ejemplo anterior, es la siguiente.

The image shows the 'Output' window of NetBeans. It displays the output of the program 'MuestraArgsLineaComandos (run)'. The output is as follows:
run:
5 argumentos de línea de comandos.
Argumento 0: 63
Argumento 1: sol
Argumento 2: 6,4
Argumento 3: -3
Argumento 4: 5.453
BUILD SUCCESSFUL (total time: 0 seconds)

Actividad 2.8

Prueba el programa anterior. Después, haz las modificaciones apropiadas para que escriba los parámetros de línea de comandos en orden inverso. Es decir, del último al primero en lugar del primero al último.

11. Obtención de datos de diversos tipos a partir de argumentos de línea de comandos

En el siguiente ejemplo, se indica un valor 5 como argumento de línea de comandos para un programa que calcula el factorial de un número. El programa calcula el valor del número especificado como primer parámetro de línea de comandos, si se ha especificado alguno. Si no, pide que se introduzca un número. Finalmente, calcula el factorial del número.

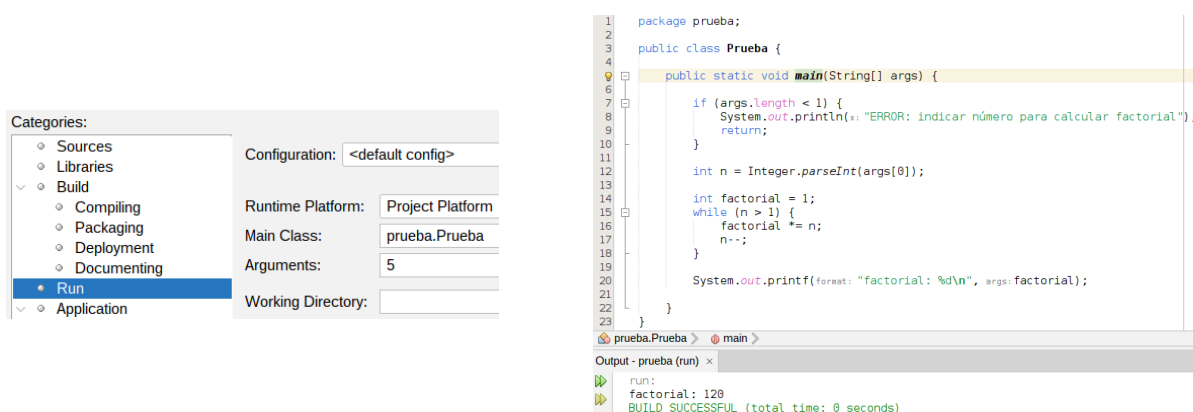


Ilustración 2.3: Especificación de valores para argumentos de línea de comandos, y uso en un programa de Java

Se obtiene un valor de tipo **int** a partir del primer argumento de línea de comandos, `args[0]`, de tipo **String**, con el siguiente código:

```
int n = Integer.parseInt(args[0])
```

Más adelante se aprenderá más acerca de las clases. Por ahora baste decir que **Integer** es una clase que permite representar valores del tipo básico **int**. Se dice que **Integer** es una clase *wrapper* del tipo **int**.

Hay más clases *wrapper* de tipos básicos, cada una con su correspondiente método `parseXXX`, donde **XXX** es el nombre de la clase. Se muestran en la siguiente tabla.

	Tipo básico	Clase wrapper
Entero	int	Integer
Entero largo	long	Long
Entero corto	short	Short
Número real	float	Float
Número real con doble precisión	double	Double
Byte	byte	Byte
Carácter Unicode	char	Character
Booleano	boolean	Boolean

Los métodos `parseXXX` de todas estas clases lanzan una excepción de tipo `NumberFormatException` cuando el dato no se puede interpretar como el tipo **XXX** en cuestión. Se puede probar con el anterior programa de ejemplo, introduciendo un argumento de línea de comandos que no representa un número entero. Esta excepción se puede capturar y gestionar, como hace el siguiente programa. El código en el que puede saltar la excepción se mete en un bloque `try { ... }`. Si el número no se puede interpretar como un número entero, se captura la excepción y, en el bloque `catch { ... }` que sigue, se muestra un mensaje de error y se termina la ejecución del programa con `return`.

```
package factorial;

public class Factorial {
```

```

public static void main(String[] args) {

    if (args.length < 1) {
        System.out.println("ERROR: indicar número para calcular factorial");
        return;
    }

    int n;
    try {
        n = Integer.parseInt(args[0]);
    } catch (NumberFormatException ex) {
        System.out.printf("ERROR: El valor introducido (%s) no es un entero.\n", args[0]);
        return;
    }

    int factorial = 1;
    while (n > 1) {
        factorial = factorial * n;
        n--;
    }

    System.out.printf("factorial: %d\n", factorial);

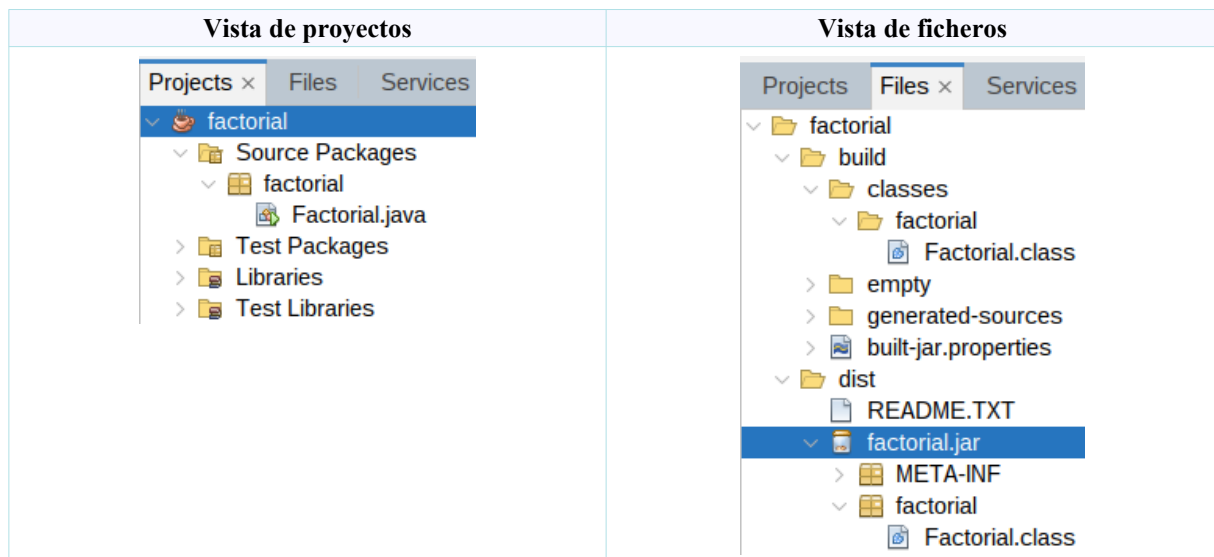
}
}

```

Ejecución de un programa de Java desde línea de comandos

Para probar un programa con distintos valores para los argumentos de línea de comandos, puede ser más cómodo ejecutarlo directamente desde línea de comandos.

En la vista de ficheros de un proyecto se pueden ver los ficheros que el entorno de desarrollo NetBeans crea cuando se construye un programa, con la opción “Build” (construir).



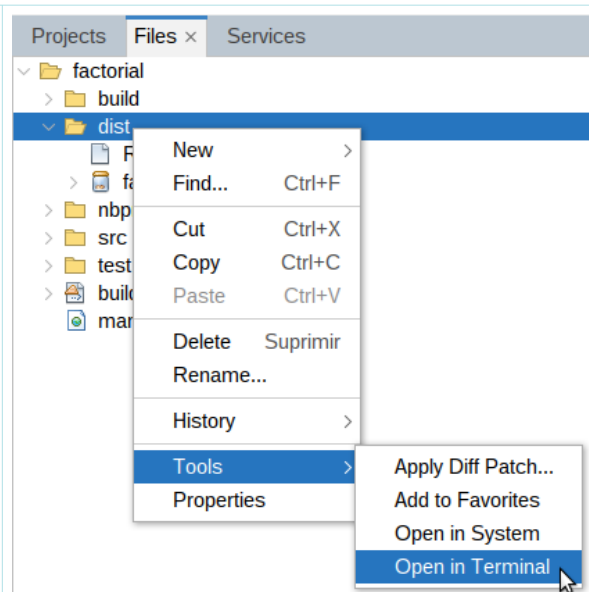
En un directorio “classes” hay un fichero `Factorial.class` que contiene el bytecode para la clase `Factorial`.

En un directorio “dist” hay un fichero `factorial.jar`, que es un fichero comprimido que contiene, entre otras cosas, el mismo fichero `Factorial.class`.

Se puede abrir una terminal de línea de comandos en este último directorio, pulsando con el botón derecho y seleccionando las opciones “Tools”, “Open in Terminal”. La terminal de línea de comandos se abre en este directorio, y en ella se puede ejecutar el programa con el comando `java -jar Factorial.jar`.

(Nota: el programa `java` es la máquina virtual de Java)

A esto se le pueden añadir después los parámetros de línea de comandos. Esta es una manera más cómoda cuando se quiere probar a ejecutar el programa con diferentes opciones.



```
Terminal - ...: ~/NetBeansProjects/factorial/dist x Output - factorial (clean.jar)
(base) carlos@babel:~/NetBeansProjects/factorial/dist$ java -jar factorial.jar
ERROR: indicar número para calcular factorial
(base) carlos@babel:~/NetBeansProjects/factorial/dist$ java -jar factorial.jar cuatro
ERROR: El valor introducido (cuatro) no es un entero.
(base) carlos@babel:~/NetBeansProjects/factorial/dist$ java -jar factorial.jar 4
factorial: 24
(base) carlos@babel:~/NetBeansProjects/factorial/dist$ java -jar factorial.jar 10
factorial: 3628800
(base) carlos@babel:~/NetBeansProjects/factorial/dist$
```

También se puede compilar y ejecutar el programa sin un entorno de desarrollo, directamente desde la línea de comandos.

La clase `Factorial.java` está en un paquete `factorial`. Para poder compilar y ejecutar desde línea de comandos, debe crearse un directorio `factorial` y dentro del él el fichero `Factorial.java`, con el código fuente de la clase.

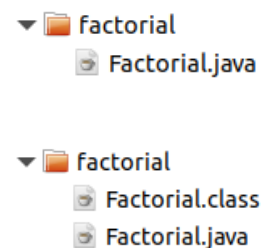
Se puede compilar el programa con el comando siguiente.

```
$ javac factorial.Factorial
```

Con ello, se crea un fichero `Factorial.class` que tiene el bytecode para la clase `Factorial` cuyo código fuente está en el fichero `Factorial.java`.

Se puede ejecutar el programa con el comando `java factorial.Factorial`, como se muestra a continuación.

```
$ java factorial.Factorial
ERROR: indicar número para calcular factorial
$ java factorial.Factorial cinco
ERROR: El valor introducido (cinco) no es un entero.
$ java factorial.Factorial 5
factorial: 120
```



Actividad 2.9

Crea un programa que lea dos números enteros de argumentos de línea de comandos y escriba su suma. Los valores deben almacenarse antes en variables de tipo **int**.

Actividad 2.10

Crear un programa que obtenga dos números reales a partir de argumentos de línea de comandos y muestre su producto. Los valores de ambos números reales deben almacenarse en variables de tipo **double**. Si no se introducen al menos dos argumentos de línea de comandos, el programa debe mostrar un mensaje de error y terminar su ejecución. Si cualquiera de los dos primeros argumentos de línea de comandos no representa un número real, debe capturarse la excepción de tipo `NumberFormatException` para mostrar un mensaje de error y terminar la ejecución.

Actividad 2.11

Crea un programa que obtenga la suma de todos los números enteros introducidos como argumentos de línea de comandos. Si algún argumento de línea de comandos no es un número entero, debe mostrarse un mensaje de error y terminar inmediatamente la ejecución del programa.

Actividad 2.12

Crea un programa que obtenga la suma de todos los números enteros introducidos como argumentos de línea de comandos. Si algún argumento de línea de comandos no es un número entero, debe ignorarse. Para ello, basta con capturar la excepción con un bloque `try {...} catch (NumberFormatException ex) {}`, en el que se deja vacía la parte del `catch`.

Actividad 2.13

Crea un programa que obtenga la suma de todos los números enteros positivos introducidos como argumentos de línea de comandos. Si algún argumento de línea de comandos no es un número entero, o si es un número entero negativo, debe ignorarse y seguir con el siguiente.

Actividad 2.14

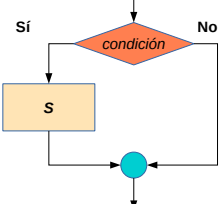
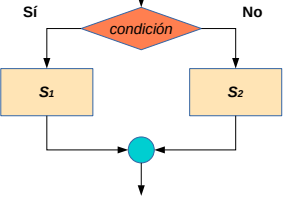
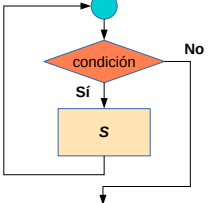
Modifica el programa de ejemplo anterior que calcula el factorial de un número introducido como argumento de línea de comandos para que, si no se proporciona un número, en lugar de mostrar un mensaje de error y terminar, lea el número de la entrada estándar, utilizando un `Scanner`.

12. Programación estructurada

Java es un lenguaje imperativo, estructurado y orientado a objetos.

Un lenguaje estructurado tiene los siguientes tipos de estructuras para construir programas.

	Diagrama de flujo	Pseudocódigo	Código en Java
Secuencia	<pre> graph TD Start(()) --> A[A] A --> B[B] B --> End(()) </pre>	Se ejecutan las sentencias una tras otra: primero A y después B.	A; B;

	Diagrama de flujo	Pseudocódigo	Código en Java
Sentencia condicional (variante 1)		Si <i>condición</i> Entonces <i>S</i> FinSi	if (<i>condición</i>) { <i>S</i> ; }
Sentencia condicional (variante 2)		Si <i>condición</i> Entonces <i>S</i> ₁ SiNo <i>S</i> ₂ FinSi	if (<i>condición</i>) { <i>S</i> ₁ ; } else { <i>S</i> ₂ ; }
Bucle while (mientras)		Mientras <i>condición</i> Hacer <i>S</i> FinMientras	while (<i>condición</i>) { <i>S</i> ; }

Estas estructuras son como bloques de construcción. Cualquier caja se puede sustituir o bien por una instrucción directamente ejecutable, o bien por una estructura de cualquier tipo. De esta forma, se puede tener una secuencia de instrucciones dentro de una sentencia condicional, o una sentencia condicional dentro de un bucle, o un bucle dentro de cualquiera de las dos ramas de ejecución de una sentencia condicional.

Indicaciones generales para actividades de programación en Java

Con los programas realizados hasta ahora con PSeInt se han omitido las **validaciones**. Es decir, el programa tenía que funcionar correctamente cuando los valores de las entradas eran correctos. Pero ya se ha aprendido a validar los valores de las entradas con Java, y deben validarse en todos los nuevos programas que se creen. Al igual que en los programas de ejemplo, las validaciones deben realizarse al principio. Si el valor de cualquier entrada es incorrecto, la ejecución del programa debe terminar inmediatamente.

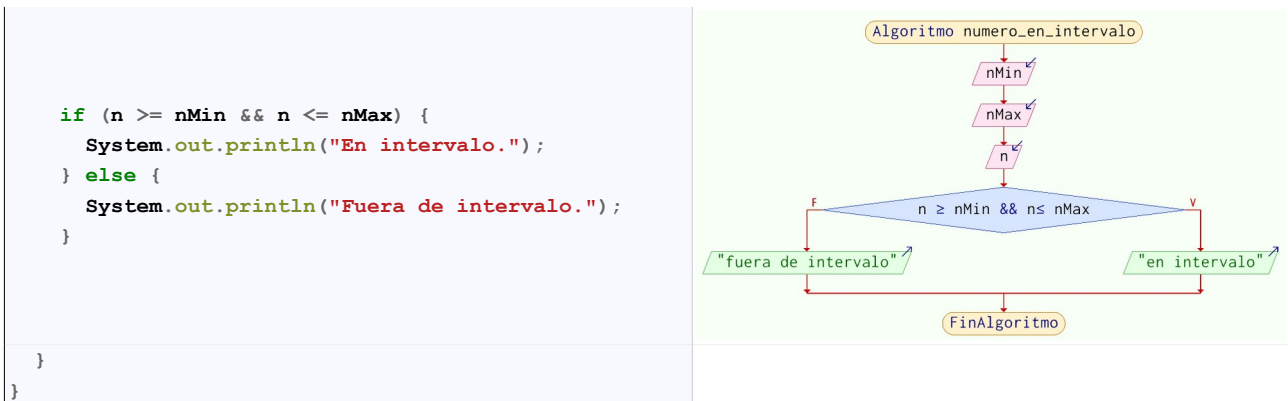
De esta manera, para crear un programa de Java basado en uno ya creado de PSeInt, solo hay que:

1. **Leer los valores de las entradas.** Con sentencias de Java equivalentes a las sentencias `Leer` de PSeInt.
2. **Añadir las validaciones al principio.** Justo después de leer los valores de las entradas. Si cualquier entrada tiene un valor incorrecto, se debe terminar inmediatamente la ejecución del programa, como se hace en los programas de ejemplo vistos anteriormente. Algunas validaciones pueden implicar a más de una entrada. Por ejemplo: si se introducen dos números, de los que el segundo no puede ser menor que el primero.
3. **Añadir el código en Java para el algoritmo propiamente dicho.** Con este, se obtendrá el valor de las salidas a partir del valor de las entradas.
4. **Escribir los valores de las salidas.** Con sentencias de Java equivalentes a las sentencias `Escribir` de PSeInt.

El código para lectura y validación de los valores para las entradas está al principio del programa, y es completamente independiente del resto. Sigue siempre los mismos patrones y es, por tanto, muy fácil de programar. Si se termina de ejecutar, se puede contar con que el valor de todas las entradas es correcto.

En el siguiente ejemplo, se piden los extremos inferior y exterior de un intervalo $[nMin, nMax]$, y un número n . Estos tres datos deben ser numéricos reales (pueden tener decimales). Además, debe cumplirse que $nMin \leq nMax$.

<pre>package NumeroEnInt; import java.util.Scanner; import java.util.InputMismatchException; public class NumeroEnInt {</pre>	
<pre> public static void main(String[] args) { double nMin; double nMax; double n;</pre>	Declaración de variables para las entradas.
<pre> Scanner s = new Scanner(System.in); System.out.print("Introducir nMin: "); try { nMin = s.nextDouble(); } catch (InputMismatchException ex) { System.out.printf("ERROR: Valor no numérico para nMin (%s).\n", s.next()); return; } System.out.print("Introducir nMax: "); try { nMax = s.nextDouble(); } catch (InputMismatchException ex) { System.out.printf("ERROR: Valor no numérico para nMax (%s).\n", s.next()); return; } System.out.print("Introducir n: "); try { n = s.nextDouble(); } catch (InputMismatchException ex) { System.out.printf("ERROR: Valor no numérico para n (%s).\n", s.next()); return; } }</pre>	Lectura y validación de valores de entradas individuales.
<pre> if (nMin > nMax) { System.out.printf("ERROR: extremo inferior del intervalo (%f) mayor que extremo superior (%f).\n", nMin, nMax); return; }</pre>	Validaciones cruzadas, en las que se verifica conjuntamente el valor de más de una entrada.

**Actividad 2.15**

(No hacer)

El siguiente ejemplo muestra un bucle **while** que escribe los números del 1 al 20.

```

public static void main(String[] args) {
    int i = 0;
    while(i < 20) {
        System.out.println(i);
        i++;
    }
}

```

Al igual que con PSeInt, se pueden utilizar variables adicionales en el bucle, inicializándolas antes de él y actualizando su valor al final de cada iteración. En el siguiente ejemplo se calcula la suma de los números del 1 al 20 utilizando una variable adicional suma, a la que se le asigna valor 0 antes del bucle y se le suma el valor de i en cada iteración del bucle.

```

public static void main(String[] args) {
    int suma = 0;
    int i = 0;
    while(i < 20) {
        suma += i;
        i++;
    }
    System.out.println(suma);
}

```

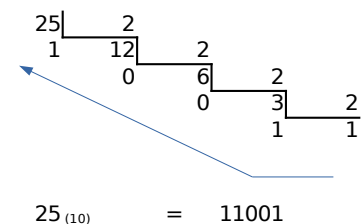
Actividad 2.16

Escribe un programa que lea un número entero no negativo y escriba el número binario equivalente, utilizando un bucle **while**.

En cada iteración del bucle se dividirá por 2 para obtener un nuevo dígito decimal en el resto, mientras el número sea mayor que 1. En la siguiente iteración, se hará lo mismo, sustituyendo el número por el cociente entero del número entre 2. El número equivalente será un 1 seguido de todos los restos que se han ido obteniendo, en orden inverso.

Es más fácil escribir el número binario equivalente al revés. Lo puedes dejar así, si no se te ocurre como mostrar los dígitos en el orden correcto.

Puedes ir concatenando los dígitos en un `String` utilizando el operador de concatenación de cadenas `+`.



Aunque las estructuras o bloques de construcción que se han visto sean suficientes para implementar cualquier algoritmo, tanto Java como en general todos los lenguajes de programación estructurados tienen algunos tipos más de estructuras que no son estrictamente necesarias, pero sí muy útiles. Esto significa que, para cualquier programa que las utilice, existe un programa equivalente (es decir, que hace lo mismo) y que solo usa las estructuras vistas hasta ahora. Pero con ellas algunos programas se pueden expresar de manera más sencilla. Estas se verán en los apartados siguientes.

12.1. Variantes de las sentencias condicionales

Las dos variantes ya vistas de las sentencias condicionales son suficientes para cualquier programa que se quiera construir.

	Diagrama de flujo	Código en Java
Sentencia condicional (variante 1)	<pre> graph TD Start(()) --> Cond{condición} Cond -- Sí --> S[S] Cond -- No --> Merge(()) S --> Merge Merge --> End(()) </pre>	<pre> if (condición) { S; } </pre>
Sentencia condicional (variante 2)	<pre> graph TD Start(()) --> Cond{condición} Cond -- Sí --> S1[S1] Cond -- No --> S2[S2] S1 --> Merge(()) S2 --> Merge Merge --> End(()) </pre>	<pre> if (condición) { S1; } else { S2; } </pre>

Si solo se utilizan estas sentencias, para algunos programas que tienen que considerar una casuística muy variada y hacer cosas diferentes para cada caso, es necesario utilizar muchas sentencias de este tipo anidadas unas dentro de otras. Esto puede hacer que el código sea muy complicado y engorroso. En Java, y en general en todos los lenguajes imperativos estructurados, existen algunos tipos de sentencias condicionales que permiten simplificar el código para sentencias condicionales anidadas.

	Diagrama de flujo	Código en Java
Condicional compuesta (variante 3)	<pre> graph TD Start(()) --> Cond1{condición1} Cond1 -- Sí --> S1[S1] Cond1 -- No --> Cond2{condición2} Cond2 -- Sí --> S2[S2] Cond2 -- No --> Merge2(()) S2 --> Merge2 S1 --> Merge1(()) Merge2 --> Merge1 Merge1 --> End(()) </pre>	<pre> if (condición1) { S1; } else if (condición2) { S2; } </pre>

	Diagrama de flujo	Código en Java
Condicional compuesta (variante 4)	<pre> graph TD Start(()) --> Cond1{condición1} Cond1 -- Sí --> S1[S1] Cond1 -- No --> Cond2{condición2} Cond2 -- Sí --> S2[S2] Cond2 -- No --> S3[S3] S1 --> Join(()) S2 --> Join S3 --> Join Join --> End[] </pre>	<pre> if (condición1) { S1; } else if (condición2) { S2; } else { S3; } </pre>
Condicional compuesta (variante 5)	<pre> graph TD Start(()) --> Cond1{condición1} Cond1 -- Sí --> S1[S1] Cond1 -- No --> Cond2{condición2} Cond2 -- Sí --> S2[S2] Cond2 -- No --> Cond3{condición3} Cond3 -- Sí --> S3[S3] Cond3 -- No --> Cond4{condición4} Cond4 -- Sí --> S4[S4] Cond4 -- No --> S5[S5] S1 --> Join1(()) S2 --> Join1 S3 --> Join1 S4 --> Join1 S5 --> Join1 Join1 --> End[] </pre>	<pre> if (condición1) { S1; } else if (condición2) { S2; } else if (condición3) { S3; } else if (condición4) { S4; } else { S5; } </pre>

Un programa de ejemplo anterior pide los extremos inferior y exterior de un intervalo $[nMin, nMax]$, y un número n , y escribe un mensaje distinto dependiendo de si n está dentro o fuera del intervalo. Se puede modificar para que escriba un mensaje distinto dependiendo de si n está antes del intervalo ($n < nMin$), dentro ($nMin \leq n \leq nMax$), o después ($n > nMax$). Se muestran lado a lado las diferencias entre ambos programas.

```

if (n >= nMin && n <= nMax) {
    System.out.println("En intervalo.");
} else {
    System.out.println("Fuera de intervalo.");
}
        
```

```

if (n < nMin) {
    System.out.println("Antes del intervalo.");
} else if (n > nMax) {
    System.out.println("Después del intervalo.");
} else {
    System.out.println("Dentro del intervalo.");
}
        
```

Actividad 2.17

Escribe un programa que lea la edad de un comprador y el importe de la compra en euros. Si el cliente es menor de edad, tiene un descuento del 10%. Si es mayor de edad, tiene un descuento del 5% si el importe de la compra es menor (estrictamente) de 100€, y de un 7,5% si es mayor o igual. El programa debe escribir el precio final que tiene que pagar el cliente, una vez aplicados los descuentos oportunos.

Actividad 2.18

Escribe un programa que lea un año y diga si es año bisiesto. En general, son bisiestos los años múltiplos de cuatro. Hay una excepción para esta regla, y es que no son bisiestos los años múltiplos de 100. Y hay una excepción para la anterior excepción, y es que sí son bisiestos los años múltiplos de 400.

12.2. Sentencia de selección múltiple

La sentencia de selección múltiple permite realizar distintas acciones según los distintos valores que tenga una variable.

	Diagrama de flujo	Pseudocódigo	Código en Java
Sentencia de selección múltiple	<pre> graph TD v((v)) -- valor1 --> S1[S1] v -- "valor2, valor3" --> S2[S2] v -- valor4 --> S3[S3] v -- Otro --> S4[S4] S1 --> Fin(()) S2 --> Fin S3 --> Fin S4 --> Fin </pre>	<p>Según <i>v</i> Hacer</p> <p><i>valor1</i>: <i>S</i>₁</p> <p><i>valor2</i>, <i>valor3</i>: <i>S</i>₂</p> <p><i>valor4</i>: <i>S</i>₃</p> <p>De Otro Modo: <i>S</i>₄</p> <p>FinSegún</p>	<pre> switch (v) { case valor1: S1; break; case valor2: case valor3: S2; break; case valor4: S3; break; default: S4; } </pre>

En el siguiente ejemplo, se lee una nota musical en notación tradicional y se da su equivalencia en notación anglosajona.

En este caso, no se puede saber que el valor de la entrada es incorrecto hasta que no se ha verificado que no coincide con ninguno de los valores correctos. Por ello, al final, o bien se escribe el valor de la salida, o bien se muestra un mensaje de error indicando que el valor de la entrada es incorrecto.

Se utiliza un valor especial de la salida, el valor **null**, para indicar que el valor de la entrada es incorrecto. Este es un valor especial, que no representa ninguna cadena de caracteres, sino el hecho de que a la variable no se le ha asignado ninguna cadena de caracteres. Por lo demás, este programa funcionaría perfectamente si se utilizara el valor "" (una cadena de caracteres vacía) en lugar del valor **null**.

<pre> package equivnotas; import java.util.Scanner; public class EquivNotas { public static void main(String[] args) { // Entradas String nota; // Salidas String na = null; System.out.print("Introducir nota: "); Scanner s = new Scanner(System.in); nota = s.nextLine(); </pre>	<p>Declaración de entradas y salidas. A la salida <i>na</i> se le asigna por defecto el valor null. Si al final sigue teniendo este valor, significa que no se ha introducido un valor correcto para la entrada <i>nota</i>.</p> <p>Lectura de entradas. En este caso, solo <i>nota</i>. No se realiza una validación previa, porque puede tomar uno de entre una lista de valores correctos.</p>
---	--

<pre> switch(nota) { case "Do": na = "C"; break; case "Re": na = "D"; break; case "Mi": na = "E"; break; case "Fa": na = "F"; break; case "Sol": na = "G"; break; case "La": na = "A"; break; case "Si": na = "B"; break; } </pre>	<p>Asignación de valor a la salida na dependiendo del valor de la salida nota.</p>
<pre> if(na == null) { System.out.printf("ERROR: Nota incorrecta (%s).\n", nota); } else { System.out.println(na); } } } </pre>	<p>Mostrar valor de la salida na, o bien indicar que el valor de la entrada nota es incorrecto.</p>

Otra posibilidad, utilizando la opción **default** de la sentencia **switch**, es el siguiente programa. En la opción **default** se escribe un mensaje de error y se termina la ejecución del programa. Si el valor de la entrada es válido, se asigna un valor a la salida, que se muestra al final del programa.

Por simplicidad se muestra solo el método **main**. Y se hará lo mismo, en general, a partir de ahora.

```

public static void main(String[] args) {

    // Entradas
    String nota;

    // Salidas
    String na;

    System.out.print("Introducir nota: ");
    Scanner s = new Scanner(System.in);

    nota = s.nextLine();

    switch (nota) {
        case "Do":
            na = "C";
            break;
        case "Re":
            na = "D";
            break;
        case "Mi":

```

```
        na = "E";
        break;
    case "Fa":
        na = "F";
        break;
    case "Sol":
        na = "G";
        break;
    case "La":
        na = "A";
        break;
    case "Si":
        na = "B";
        break;
    default:
        System.out.printf("ERROR: Nota incorrecta (%s).\n", nota);
        return;
    }
    System.out.println(na);
}
```

Por último, con el siguiente programa se escribe el tipo de vocal (abierta, cerrada o media).

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

    // Entradas
    String vocal;

    // Salidas
    String tipo;

    System.out.print("Introducir vocal: ");
    Scanner s = new Scanner(System.in);
    vocal = s.nextLine();

    switch (vocal) {
        case "a":
            tipo = "abierta";
            break;
        case "i":
        case "u":
            tipo = "cerrada";
            break;
        case "e":
        case "o":
            tipo = "media";
            break;
        default:
            System.out.printf("ERROR: Nota no es una vocal (%s).\n", vocal);
            return;
    }
    System.out.println(tipo);
}
```

Actividad 2.19

Crear un programa en Java que pida una cantidad numérica y un tipo de cliente. Debe utilizarse una sentencia **switch**. Si el tipo de cliente es A, tiene un descuento de 8%. Si es B, tiene un descuento de un 16%, si es C de un 18%, y si es D de un 20%. El programa debe escribir la cantidad final que debe pagar el cliente, una vez aplicado el descuento que, en su caso, corresponda.

Actividad 2.20

Crear un programa para el cálculo del salario de un trabajador según el número de horas trabajadas en una semana y el tipo de salario. Debe utilizarse una sentencia **switch**.

Si el salario es de tipo A, se pagan 6€ por hora, hasta 40 horas, y a partir de ellas a 9€ por hora.

Si el salario es de tipo B, se pagan 7€ por hora, hasta 40 horas, y a partir de ellas a 9,5€ por hora.

Si el salario es de tipo C, se pagan 8€ por hora, hasta 40 horas, y a partir de ellas a 10€ por hora.

El programa debe pedir el número de horas trabajadas y el tipo de salario y calcular y escribir el salario total.

12.3. Bucle **for** (para)

Existen otros tipos de bucles además del bucle *while* (mientras). Cualquier cosa que se haga con ellos se puede hacer con un bucle *while*. Pero muchos programas se pueden realizar de manera más simple utilizándolos.

Diferencias con PSeInt

Los bucles Mientras *condición* Hacer de PSeInt son equivalentes a los bucles *while* (*condición*) de Java que se verán a continuación, y que también existen en general en todos los lenguajes imperativos estructurados, que son la práctica totalidad de los lenguajes utilizados en la práctica.

En cambio, los bucles Para y los bucles Repetir ... Hasta Que de PSeInt tienen algunas diferencias con respecto a los bucles *for* y los bucles *do ... while*(*condición*) existentes en Java, y en general en todos los lenguajes imperativos estructurados.

El bucle *for* de Java es más potente que el bucle Para de PSeInt. Permite hacer lo mismo, pero también otras cosas que no son posibles con el bucle Para de PSeInt.

Con el bucle Repetir ... Hasta Que *condición* de PSeInt se ejecuta el bucle hasta que se cumple la condición. Con el bucle *do ... while*(*condición*) de Java, en cambio, se ejecuta el bucle mientras que (es decir, hasta que no) se cumple la condición.

El bucle *for* (para) es más potente que el disponible en PSeInt. Es decir, permite hacer lo mismo, pero también otras cosas que no son posibles con PSeInt, como se verá más adelante.

	Código en Java	Diagrama de flujo equivalente con bucle <i>while</i> (mientras)	Código en Java equivalente
Bucle <i>for</i> (para)	<pre>for(Inicialización; condición; Fin) { S; }</pre>	<pre> graph TD Inicio(()) --> Inicialización[Inicialización] Inicialización --> Condición{condición} Condición -- Sí --> S[S] S --> Fin[Fin] Fin --> Condición Condición -- No --> Salida(()) </pre>	<pre>Inicialización while(condición) { S; Fin; }</pre>

El siguiente programa, por ejemplo, escribe todos los números del 1 al 20 utilizando un bucle **for**. Se muestra junto a otro que utiliza un bucle **while** equivalente. El código que utiliza un bucle **for** es más

compacto, pero no por ello menos claro. Para bucles como este, controlados por una única variable numérica, se prefiere siempre un bucle **for** a un bucle **while**.

```
public static void main(String[] args) {  
    for(int i=1; i<=20; i++) {  
        System.out.println(i);  
    }  
}
```

```
public static void main(String[] args) {  
    int i = 1;  
    while(i <= 20) {  
        System.out.println(i);  
        i++;  
    }  
}
```

Con una pequeña variación de este bucle **for** se pueden escribir solo los números impares entre 1 y 20.

```
public static void main(String[] args) {  
    for(int i=1; i <= 20; i += 2) {  
        System.out.println(i);  
    }  
}
```

Abreviaturas de asignaciones con operadores aritméticos

La sentencia `i += 2` del ejemplo anterior es equivalente a `i = i+2`.

Se pueden utilizar estas abreviaturas con otros operadores.

`i -= 2` del ejemplo anterior es equivalente a `i = i-2`.

`i *= 2` del ejemplo anterior es equivalente a `i = i*2`.

`i /= 2` del ejemplo anterior es equivalente a `i = i/2`.

Actividad 2.21

Escribe un programa que escriba los números entre 20 y 1, en orden descendiente, utilizando un bucle **for**.

Actividad 2.22

Escribe un programa que escriba los números pares entre 20 y 2, en orden descendiente, utilizando un bucle **for**.

Actividad 2.23

(No hacer)

Escribe un programa que escriba los números entre 20 y 1, en orden descendiente, utilizando un bucle **for**.

Se pueden añadir inicializaciones y variables adicionales a los bucles **for**, de manera análoga a como se vio que se podía hacer con un bucle **for**. Al fin y al cabo, un bucle **for** no es sino un bucle **while** expresado con una notación distinta.

En el siguiente ejemplo se utiliza un bucle **for** para calcular la suma de los números del 1 al 20, utilizando una variable adicional `suma`.

```
public static void main(String[] args) {  
    int suma = 0;  
    for (int i = 1; i <= 20; i ++ ) {  
        suma += i;  
    }  
    System.out.println(suma);  
}
```


En el siguiente ejemplo se utiliza un bucle **for** para calcular la suma de las primeras 20 potencias de 2. Para ello se utiliza una variable adicional `pot2` para generar las sucesivas potencias de 2, y una variable adicional `suma` para guardar su suma.

```
public static void main(String[] args) {  
    int suma = 0;  
    int pot2 = 1;  
    for (int i = 1; i <= 20; i++) {  
        suma += pot2;  
        pot2 *= 2;  
    }  
    System.out.println(suma);  
}
```

Se puede utilizar el método `charAt` de la clase **String** para obtener el carácter que hay en una posición dada de un **String**. La posición se especifica con un número entero, y la posición del primer carácter es 0. En el siguiente ejemplo se utiliza un bucle **for** para escribir todos los caracteres de un **String**.

```
public static void main(String[] args) {  
    Scanner s = new Scanner(System.in);  
  
    System.out.println("Introducir cadena de caracteres: ");  
    String str = s.nextLine();  
  
    for (int i = 0; i < str.length(); i++) {  
        System.out.printf("(%d):%c ", i, str.charAt(i));  
    }  
    System.out.println();  
}
```

Actividad 2.24

Escribe un programa que lea una cadena de caracteres y escriba el número de vocales que contiene. No es necesario que tenga en cuenta las vocales acentuadas o con diéresis o con otros diacríticos, pero sí que tenga en consideración por igual tanto las mayúsculas como las minúsculas. Utiliza un bucle para obtener los caracteres y una sentencia **switch** para aumentar la cuenta de vocales cuando el carácter es una vocal.

Actividad 2.25

Escribe un programa que lea un número binario y escriba su equivalencia en decimal, utilizando un bucle **for**. Te puedes basar en los dos programas de ejemplo anteriores. Uno de ellos obtiene la suma de potencias crecientes de 2. El otro obtiene todos los caracteres de una cadena de caracteres con el método `charAt()`. Para obtener la equivalencia en decimal, habría utilizar un bucle **for** que obtenga en cada iteración una nueva potencia de 2 y un nuevo carácter de la cadena de caracteres, empezando por el último. La potencia se sumará solo si el carácter es **'1'**. Si la cadena de caracteres representa un número binario, solo puede contener los caracteres **'0'** y **'1'**. Si contiene cualquier otro carácter, se debe mostrar un mensaje de error y terminar inmediatamente.

Actividad 2.26

Escribe un programa que lea un número octal (en base 8) y escriba su equivalencia en decimal, utilizando un bucle **for**. El programa será similar al programa anterior, pero dado que hay más de dos posibles dígitos (de 0 a 7), se utilizará una sentencia **switch** en lugar de una sentencia **if**.

Actividad 2.27

Escribe un programa que lea un número entero no negativo y escriba su equivalencia en binario, utilizando un bucle **while**.

En cada iteración del bucle se dividirá por 2 para obtener un nuevo dígito decimal en el resto, mientras el número sea mayor que 1. En la siguiente iteración, se hará lo mismo, sustituyendo el número por el cociente entero del número entre 2.

El siguiente programa utiliza dos bucles anidados, uno dentro de otro, para escribir las tablas de multiplicar del 1 al 10. Se muestra al lado su salida.

```
public static void main(String[] args) {
    for (int i = 1; i < 10; i++) {
        for (int j = 1; j < 10; j++) {
            System.out.printf("%2d ", i*j);
        }
        System.out.println();
    }
}
```

```
2  4  6  8 10 12 14 16 18
3  6  9 12 15 18 21 24 27
4  8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

Actividad 2.28

Cambiar el programa anterior para que su salida sea la siguiente. Hay que utilizar un bucle también para escribir la segunda línea a partir del carácter +.

```
x | 1  2  3  4  5  6  7  8  9 10
---+-----
2 | 2  4  6  8 10 12 14 16 18 20
3 | 3  6  9 12 15 18 21 24 27 30
4 | 4  8 12 16 20 24 28 32 36 40
5 | 5 10 15 20 25 30 35 40 45 50
6 | 6 12 18 24 30 36 42 48 54 60
7 | 7 14 21 28 35 42 49 56 63 70
8 | 8 16 24 32 40 48 56 64 72 80
9 | 9 18 27 36 45 54 63 72 81 90
```

Actividad 2.29

Cambiar el programa anterior para que su salida sea la siguiente. Solo hay que escribir el producto de dos números *i* y *j* cuando *j* no es mayor que *i* (columna).

```
1 | 1
2 | 2  4
3 | 3  6  9
4 | 4  8 12 16
5 | 5 10 15 20 25
6 | 6 12 18 24 30 36
7 | 7 14 21 28 35 42 49
8 | 8 16 24 32 40 48 56 64
9 | 9 18 27 36 45 54 63 72 81
---+-----
x | 1  2  3  4  5  6  7  8  9
```

12.4. Bucle **do ... while** (Hacer ... Mientras)

Con el bucle *do ... while* (repetir ... mientras), el cuerpo del bucle siempre se ejecuta al menos una vez, y se ejecuta mientras se cumple la condición que se especifica al final del bucle.

	Código en Java	Diagrama de flujo equivalente con bucle <i>while</i> (mientras)	Código en Java equivalente
Bucle <i>do ... while</i> (hacer ... mientras)	<pre>do { S; } while (condición);</pre>	<pre> graph TD Start(()) --> S1[S] S1 --> Cond{condición} Cond -- Sí --> S2[S] S2 --> Cond Cond -- No --> Exit(()) </pre>	<pre>S; while (condición) { S; }</pre>

El siguiente programa lee números hasta que se introduce un 0. Cuando se introduce un 0, muestra la suma de los números positivos y la suma de los números negativos que se han introducido. Cuando se introduce cualquier cosa que no es un número, escribe un mensaje informativo.

En este programa, un bucle `do ... while` permite simplificar la programación. Si no se utilizara, habría que leer el primer valor antes de entrar en el bucle y el siguiente justo al final del bucle, antes de la siguiente iteración. Además habría que gestionar la excepción que se produce cuando no se introduce un número entero. Se utiliza una variable auxiliar `fin` de tipo **boolean** para salir del bucle, a la que inicialmente se le da valor **false**. Cuando se introduce un cero, se le asigna valor **true**.

```
public static void main(String[] args) {
    Scanner s = new Scanner(System.in);
    double sumaPos = 0;
    double sumaNeg = 0;
    double num;
    boolean fin = false;
    do {
        try {
            System.out.print("Introducir numero: ");
            num = s.nextDouble();
            if (num > 0) {
                sumaPos += num;
            } else if (num < 0) {
                sumaNeg += num;
            } else {
                fin = true;
            }
        } catch (InputMismatchException ex) {
            System.out.printf("%s no es un número.\n", s.next());
        }
    } while (!fin);
    System.out.printf("Suma de positivos: %f.\n", sumaPos);
    System.out.printf("Suma de negativos: %f.\n", sumaNeg);
}
```

Actividad 2.30

(x, y)	0	1	2	3	4
0	(0,0)	(1,0)	(2,0)	(3,0)	(4,0)
1	(0,1)	(1,1)	(2,1)	(3,1)	(4,1)
2	(0,2)	(1,2)	(2,2)	(3,2)	(4,2)
3	(0,3)	(1,3)	(2,3)	(3,3)	(4,3)
4	(0,4)	(1,4)	(2,4)	(3,4)	(4,4)

Escribe un Programa que pide las coordenadas iniciales dentro de una cuadrícula de 5x5 y después movimientos a realizar, hasta que se introduzca un punto ("."). Las coordenadas vienen dadas en la forma (x, y).
 El programa pedirá un movimiento a realizar hasta que se introduzca un punto (.), y entonces terminará. Los posibles movimientos y su significado son:

- x a la izquierda.
- c a la derecha.
- p arriba.
- l abajo.

No se puede salir del tablero. Por ejemplo, si la posición actual es en la primera columna (x=0), con un movimiento a la izquierda no se cambiará la posición. Después de cada movimiento, el programa escribirá las coordenadas de la posición actual, haya cambiado o no. Si se introduce un movimiento incorrecto, se mostrará un mensaje de error.

13. Arrays en Java

Los *arrays* en Java son muy similares a los vistos con PSeInt en lo que respecta al acceso a sus elementos, tanto para obtener su valor como para asignarlo. Si *arr* es un *array*, entonces:

- `arr[i]` es el elemento del *array* en la posición *i*. Las posiciones del *array* se identifican con un número entero, que va desde 0 (para la primera posición) hasta `arr.length-1`.
- Se puede obtener la longitud de un *array* *arr* con `arr.length`.
- Se puede crear un *array* en memoria de manera similar a como se hace con PSeInt. Se muestran las diferencias a continuación.

PSeInt	Java
Dimensionar <code>arr(10)</code>	<pre>String[] arrStr = new String[10]; int[] arrNum = new int[10];</pre>

En Java hay que declarar el tipo de los elementos de un *array*, de manera análoga a como hay que declarar el tipo de todas las variables. A todos los elementos del *array* se les asigna el valor por defecto correspondiente a su tipo. Como ya se vio, este es 0 para los tipos numéricos (como por ejemplo `int`) y `null` para la clase `String`.

Con Java se puede, además, asignar directamente valores a todas las posiciones de un *array*, como se muestra en el siguiente ejemplo. Se compara con un programa equivalente en pseudocódigo de PSeInt.

<pre>public static void main(String[] args) { String[] diasSem = {"Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"}; for(int i=0; i<diasSem.length; i++) { System.out.println(diasSem[i]); } }</pre>	<pre>NUM_DIAS_SEM <- 7 Dimensionar diasSem(NUM_DIAS_SEM) diasSem[0] <- 'Lunes' diasSem[1] <- 'Martes' diasSem[2] <- 'Miercoles' diasSem[3] <- 'Jueves' diasSem[4] <- 'Viernes' diasSem[5] <- 'Sábado' diasSem[6] <- 'Domingo' i <- 0 Mientras i<NUM_DIAS_SEM Hacer Escribir diasSem[i] i <- i+1 FinMientras</pre>
--	--

En el ejemplo anterior se puede ver cómo se puede utilizar un bucle `for` para iterar sobre un *array*. Esto es más cómodo que utilizar un bucle `while` equivalente al utilizado en el programa equivalente en PSeInt. Pero existe una manera aún más cómoda para iterar sobre todos los elementos de un *array*, y es un bucle `for` mejorado, como se muestra en el siguiente ejemplo.

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

    String[] diasSem = {"Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"};

    for (String dia: diasSem) {
        System.out.println(dia);
    }

}
```

Por último, un ejemplo más de un programa en Java equivalente a un programa de PSeInt del tema anterior. El siguiente programa crea un *array* y lo rellena con tantos números pares como indica el usuario. Se muestra resaltado el código que hace lo mismo que el programa de PSeInt. En el programa en Java se incluyen al principio validaciones con las que se termina inmediatamente la ejecución del programa si no se introduce un dato correcto, a saber, un número entero mayor que cero.

<pre>public static void main(String[] args) { Scanner s = new Scanner(System.in); int n; System.out.print("Número de números pares que hay que generar: "); try { n = s.nextInt(); } catch (InputMismatchException ex) { System.out.printf("ERROR: %s no es un entero.\n", s.next()); return; } if (n < 1) { System.out.printf("ERROR: %s no es mayor que 0.\n", n); return; } int[] nums = new int[n]; int nPar = 2; for (int i = 0; i < nums.length; i++) { nums[i] = nPar; nPar += 2; } for (int i = 0; i < nums.length; i++) { System.out.printf("[%d]: %d\n", i, nums[i]); } }</pre>	<pre>1 Algoritmo num_pares 2 Leer n 3 Dimensionar nums(n) 4 i <- 0 5 nPar = 2 6 Mientras i<n Hacer 7 nums[i] <- nPar 8 nPar = nPar + 2 9 i <- i+1 10 FinMientras 11 i <- 0 12 Mientras i<n Hacer 13 Escribir '[', i, ']: ', nums[i] 14 i <- i+1 15 FinMientras 16 FinAlgoritmo 17</pre>
--	--

Para algunos ejercicios posteriores será útil el siguiente programa que rellena un *array* con números aleatorios. Antes se muestra la manera de generar números aleatorios.

Generación de números aleatorios con la clase Random

El siguiente código de programa genera un numero aleatorio entre 0 y 10. Hay que pasar a `nextInt` el valor máximo que se quiere obtener mas 1.

```
int b = 10;
Random r = new Random();
int aleat = r.nextInt(b + 1);
System.out.println(aleat);
```

El siguiente código de programa genera un numero aleatorio entre 10 y 30.

```
int b = 10;
int b = 30;
Random r = new Random();
int aleat = r.nextInt(a, b + 1);
System.out.println(aleat);
```

El siguiente código de programa rellena un *array* con números entre 1 y 12, y después los muestra en forma de lista separada por comas. Se puede utilizar este código para algunos de los ejercicios siguientes.

```
int nums[] = new int[20];
int valMax = 12;

Random r = new Random();

for(int i=0; i<nums.length; i++) {
    nums[i] = r.nextInt(1, valMax + 1);
}

for(int i=0; i<nums.length; i++) {
    if(i > 0) {
        System.out.print(", ");
    }
    System.out.print(nums[i]);
}
System.out.println("");
```

Actividad 2.31

Rellenar un *array* de números enteros de 30 elementos con números aleatorios entre 1 y 100. Después escribir sus contenidos y, finalmente, obtener la media aritmética de los elementos que contiene. La media aritmética debe calcularse con decimales, no debe hacerse una división entera para obtenerla.

Actividad 2.32

Rellenar un *array* de números enteros de 30 elementos con números aleatorios entre 1 y 100. Después escribir sus contenidos y, finalmente, obtener los valores máximo y mínimo que contiene.

Actividad 2.33

Rellenar dos *arrays* de números enteros de 30 elementos con números aleatorios entre 1 y 100. Después escribir sus contenidos y, finalmente, crear un nuevo *array* con la suma de los dos anteriores, de manera que en cada posición suya esté la suma de los elementos hay en la misma posición en los otros dos.

Actividad 2.34

Rellenar un *array* de 30 elementos con los números del 1 al 30. Después crear un nuevo *array* con los mismos elementos, pero en orden inverso y, por último, mostrar los contenidos de este nuevo *array*.

Actividad 2.35

Rellenar un *array* de 30 elementos con los números del 1 al 30. Después invertir los contenidos del *array* en el mismo *array*. Para ello, intercambiar los contenidos de la primera y la última posición, los de la segunda y penúltima, y así en adelante. Debe hacerse en un bucle en la que se mantienen dos índices, cada uno de los cuales hace referencia a una posición del *array*. El primero empieza con valor 0, referenciando al primer elemento del *array*, y el segundo empieza referenciando al último elemento del *array*. En cada iteración, se intercambian los elementos del *array* referenciados por ambos índices, se incrementa el valor del primero, y se decrementa el valor del último. El bucle se ejecuta mientras que el valor del primero sea menor que el valor del último.

Al final deben mostrarse los contenidos del *array*, de la primera a la última posición. Si todo ha funcionado bien, deben mostrarse los números del 30 al 1.

Actividad 2.36

Pedir el número de caras de un dado (los hay de 4, de 6, de 12 y de 20 caras, por ejemplo) y el número de lanzamientos del dado. Simular el lanzamiento del dado con el número de caras dado y el número de veces dado. El número de veces que ha salido cada número debe anotarse en una posición de un *array*, con tantos elementos como el número de caras. Finalmente, mostrar la frecuencia con la que ha salido cada cara, tanto absoluta (número de veces que ha salido) como relativa en % (número de veces que ha salido la cara dividido por el número de lanzamientos y multiplicado por 100).

Actividad 2.37

Crear un *array* con cinco sustantivos, otro con 10 verbos, y otro con cuatro objetos directos. Generar y escribir 40 oraciones al azar, seleccionando de estos *arrays* un sustantivo al azar, un verbo al azar, y un objeto directo al azar.

Actividad 2.38

Simular una partida de bingo. Deben salir todos los números del 1 al 90 sin repetir. Se utilizarán dos *arrays*. En uno de tipo **boolean** y con 90 posiciones debe anotarse cuándo un número ya ha salido. En otro de tipo *int* y con el mismo número de posiciones deben anotarse los números que van saliendo, de la primera a la última. Para elegir cada número se genera un número aleatorio que representa una posición *p* del primer *array*, correspondiente al un número *p*+1. Si esa posición contiene **false**, se asigna **true** y se anota el número en el otro *array*. Si no, se comprueba el contenido de posiciones sucesivas del primer *array*, hasta que se encuentre una que contenga **false**. Si se llega al final del *array*, se continúa a partir de la primera posición.

Actividad 2.39

Crear un programa que juegue al ahorcado. El programa elegirá una palabra al azar de un *array* de tipo **String**, en el que se introducirán unas cuantas palabras. Cuantas más palabras mejor, pero para empezar bastarían con cinco palabras. No se distinguirán como letras diferenciadas las letras con acentos o con diéresis. Es decir, que las únicas vocales serán A, E, I, O y U.

Inicialmente, el jugador no habrá acertado ninguna letra de la palabra. El jugador introducirá una letra en cada jugada. En realidad, introducirá una cadena de caracteres. Pero si introduce alguna que tenga una longitud mayor que 1, se mostrará un mensaje de error y se pedirá de nuevo una letra.

Una vez leída una letra, se anotarán en un *array* **boolean[]** **aciertos** las posiciones en las que está la letra que ha introducido, asignándoles un valor **true**, y se mostrará el estado del juego. Este *array* se debe crear al principio con igual longitud que la palabra elegida para jugar. Por cada letra de la palabra, se mostrará la letra si ya la ha introducido el usuario, o un carácter '_' si no. El jugador gana cuando ha acertado todas las letras de la palabra, y pierde si no las ha acertado todas después de un número máximo de intentos. El número máximo de intentos se debe asignar inicialmente a una variable **int** **MAX_FALLOS**, y su valor sería en principio 6. Tras cada jugada, debe mostrarse también el número de intentos que quedan. Se muestran dos ejemplos a continuación.

(Palabra: SENDERO)

_ _ _ _ _ _ (0 de 6 fallos)

Introduce letra: A

_ _ _ _ _ _ (1 de 6 fallos)

Introduce letra: E

_ E _ _ _ E _ _ (1 de 6 fallos)

Introduce letra: O

_ E _ _ _ E _ O (1 de 6 fallos)

Introduce letra: L

_ E _ _ _ E _ O (2 de 6 fallos)

Introduce letra: M

_ E _ _ _ E _ O (3 de 6 fallos)

Introduce letra: N

_ E N _ _ E _ O (3 de 6 fallos)

Introduce letra: R

_ E N _ _ E R O (3 de 6 fallos)

Introduce letra: D

_ E N D E R O (3 de 6 fallos)

Introduce letra: T

_ E N D E R O (3 de 6 fallos)

Introduce letra: S

S E N D E R O

¡ HAS GANADO !

(Palabra: SEÑALECTICA)

_ _ _ _ _ _ _ _ _ _ (0 de 6 fallos)

Introduce letra: A

_ _ _ A _ _ _ _ _ A (0 de 6 fallos)

Introduce letra: E

_ E _ A _ E _ _ _ A (0 de 6 fallos)

Introduce letra: I

_ E _ A _ E _ _ I _ A (0 de 6 fallos)

Introduce letra: F

_ E _ A _ E _ _ I _ A (1 de 6 fallos)

Introduce letra: P

_ E _ A _ E _ _ I _ A (2 de 6 fallos)

Introduce letra: N

S E _ A _ E _ _ I _ A (3 de 6 fallos)

Introduce letra: D

S E _ A _ E _ _ I _ A (4 de 6 fallos)

Introduce letra: L

S E _ A L E _ _ I _ A (4 de 6 fallos)

Introduce letra: R

S E _ A L E _ _ I _ A (5 de 6 fallos)

Introduce letra: M

¡ HAS PERDIDO ! La palabra es:

S E Ñ A L E C T I C A