1. Bibliotecas de clases

En anteriores unidades hemos conocido los fundamentos del lenguaje de programación Java. Así pues, ya sabemos que existen palabras reservadas que hacen referencia a tipos de datos (`int`, `float`, etcétera) o a instrucciones del sistema (`for`, `while`, `if`…).

Aunque de momento solo estamos trabajando a un nivel muy básico con el lenguaje, quizás nos hayan llamado la atención otras expresiones como, por ejemplo, `System.out.print(…)` —para mostrar algo por pantalla—, o `Scanner teclado = new Scanner(System.in)` —para crear un buffer de entrada que permita introducir datos desde el teclado—. También hemos visto que, al declarar una variable de tipo entero, usamos `int` (con i minúscula), pero para las cadenas de caracteres usamos `String` (con s mayúscula). Esto se debe a que `int` es un tipo de datos primitivo, mientras que **`String`** es una clase, como también lo son **`Scanner`** o **`System`**.

IMPORTANTE

Una **clase** es, según la RAE, un conjunto de elementos con caracteres comunes. En informática se usa precisamente para agrupar en un solo tipo especial de datos todos aquellos elementos con características comunes. Volveremos sobre ello más adelante.

De momento, todo lo que necesitamos saber sobre las clases es que el lenguaje Java ya lleva implementadas muchas de ellas y que las podemos usar en nuestros programas. Para ello, debemos conocer su nombre, sus propiedades (características) y sus métodos (qué cosas podemos hacer con un elemento de la clase). Empecemos por las clases más sencillas, las llamadas clases envoltorio o *wrapper*.

1.1. Clases envoltorio

Las clases envoltorio se usan para «envolver» en ellas a los tipos primitivos de datos como `int`, `float`, `boolean`, `char`, etcétera. Al tratarse de una clase, permite lanzar instrucciones que de otra manera no podríamos. Las clases envoltorio tienen, en general, el mismo nombre del tipo al que envuelven pero con la inicial en mayúscula. Como excepciones, la clase envoltorio de `char` se llama **`Character`**, y la de `int` se llama **`Integer`**.

Tipo básico	Clase envotorio
int	Integer
char	Character
boolean	Boolean
long	Long
double	Double
float	Float
short	Short
byte	Byte

Tabla 4. 1. Tipos primitivos de datos y sus correspondientes clases envoltorio

Para trabajar con tipos de datos existe también la clase **`String`**, que permite manipular cadenas de caracteres. Por sus características diferenciales, la trataremos más adelante en un apartado específico.

La clase **`Integer`**, por su parte, tiene métodos (funcionalidades especiales) que permiten, por ejemplo, convertir un número entero a una cadena de caracteres, o viceversa.

```
int numero = 1;  // el valor de numero es 1
String cadena = Integer.toString(x);
// con el método toString() de la clase Integer convertimos el entero a cadena
String cadena = "12345";   // una cadena de caracteres numéricos
int numeroEntero = Integer.parseInt(cadena);
// a través del método parseInt convertimos la cadena a un número entero
```

La conversión entre los tipos primitivos y sus clases envoltorio es, desde la versión 5 de Java, automática. Anteriormente debíamos realizar la conversión de manera explícita:

```
int numero = 1;  // numero es una variable de tipo entero cuyo valor es 1
Integer numeroObjeto = new Integer(numero); // convertimos la variable a objeto
```

IMPORTANTE

Los puntos separan un objeto de sus métodos y atributos, así como también una clase de sus objetos y métodos. Es el caso de **`Integer.toString`**, o **`Integer.parseInt`**.

La clase `Character` tiene métodos que permiten, por ejemplo, detectar si un carácter es numérico o alfabético, si está en mayúsculas o minúsculas, y también hacer conversiones entre mayúsculas y minúsculas. Por ejemplo:

```
Character.isDigit('A');  // devolverá False, ya que la A no es un dígito numérico
Character.isDigit('2'); // devolverá True, ya que el 2 sí que es un dígito numérico
```

```
Character.isUpperCase('A'); // devolverá True, ya que el carácter 'A' está en mayúsculas
Character.isLowerCase('a'); // devolverá True, ya que el carácter 'a' está en minúsculas
char caracterMinusculas = Character.toLowerCase('A');
                                    // guarda en la variable la letra 'a' ('A' en minúsculas)
```

1.2. Otras clases

Hay otras clases nativas de Java, aunque no todas se pueden usar directamente en nuestras aplicaciones. En ocasiones hace falta hacer una importación de la clase en cuestión para poder utilizarla.

Por ejemplo, la **clase Random** permite crear objetos que generan números aleatorios.

```
Random semillaAleatorio = new Random();  // creación de un objeto de la clase Random
int numeroAleatorio = semillaAleatorio.nextInt(); // genera un número aleatorio entero
float numeroAleatorioDecimal= semillaAleatorio.nextFloat(); // crea núm. aleat. tipo float entre 0 y 1
```

Sin embargo, si ejecutamos ese código en nuestra aplicación, veremos que el sistema nos muestra un error avisando de que la clase `Random` no se encuentra. Para que desaparezca el mensaje de error, debemos incluir la siguiente línea en el código, después del package:

```
import java.util.Random;
```

Para generar un número aleatorio dentro de un intervalo, lo podemos hacer usando `nextDouble()`, que nos devuelve un número con decimales entre 0 y 1, multiplicando el resultado por la cantidad de valores del intervalo y finalmente sumándole el primer valor de dicho intervalo. Por ejemplo, para generar un número aleatorio entre 1 y 10 a partir de nuestro objeto `Random` llamado `semillaAleatorio`, haríamos lo siguiente:

```
int aleatorioEntrely10 = (int) semillaAleatorio.nextDouble() * 10 + 1;
```

Veremos un ejemplo completo de generación de números aleatorios en el material complementario.

La **clase `Math`** contiene una serie de métodos y constantes que nos pueden ayudar a realizar operaciones matemáticas más complejas, como raíces cuadradas, potencias, logaritmos, etcétera, además de otras utilidades como las constantes PI o E que ya hemos utilizado en unidades anteriores. Por ejemplo:

`int numeroRedondeado = (int) Math.round(5.2)`	redondea 5.2 y devuelve 5
`int numeroRedondeado = (int) Math.round(5.9)`	redondea 5.9 y devuelve 6
`int mayor = Math.max(10, 2)`	devuelve 10, el valor mayor de los dos
`int menor = Math.min(10, 2)`	devuelve 2, el valor menor de los dos
`double raiz = Math.sqrt(36)`	devuelve 6.0, la raíz cuadrada de 36
`double potencia = Math.pow(2, 5)`	devuelve 32, el resultado de 2 elevado a 5

Tabla 4. 2. Ejemplos de uso de la clase Math

Como hemos comentado, la clase `Math` permite también utilizar algunas constantes ampliamente usadas en matemáticas como, por poner un ejemplo que ya hemos usado anteriormente, el número pi.

```
double area = 2*Math.PI*radio;   // calcula la superficie de una circunferencia
```

IMPORTANTE

Puedes encontrar más información sobre todas las utilidades de la clase **Math** en la documentación oficial de Java que aparece en la web de Oracle.

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/lang/Math.html

La clase **`Scanner`**, como hemos visto en unidades anteriores, se usa para permitir la introducción de datos desde el teclado. Para poder usarla debe importarse:

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

Posteriormente, en nuestra aplicación podemos crear un objeto de la clase `Scanner` para poder pedir datos desde el teclado:

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

También es posible usar sus métodos `nextInt()`, `nextDouble()`, `next()`, `nextLine()`, etcétera, para pedir un dato desde el teclado.

```
System.out.println("Introduce la base: ");
int base=teclado.nextInt();
System.out.println("Introduce el exponente: ");
int exponente=teclado.nextInt();
System.out.println("El valor de " + base + " elevado a " + exponente + " es igual a " + Math.pow(base,exponente));
```

IMPORTANTE

Puedes encontrar más información sobre todas las utilidades de la clase **Scanner** en la documentación oficial de Java que aparece en la web de Oracle.

https://docs.oracle.com/en/java/javase/16/docs/api/java.base/java/util/Scanner.html

La clase **`Date`** permite trabajar con fechas. Una operación muy sencilla es obtener la fecha de hoy:

```
Date hoy = new Date(); // crea un objeto Date con la fecha de hoy
System.out.println("Hoy es " + hoy); // muestra la fecha de hoy
```

Para escoger el formato en el que mostramos la fecha disponemos de la clase `SimpleDateFormat`:

```
SimpleDateFormat formatea = new SimpleDateFormat("dd/mm/yyyy");
System.out.println("Hoy es " + formatea.format(hoy));
```

O bien, podemos mostrar el nombre del mes en castellano:

```
Date hoy = new Date();

SimpleDateFormat formatea = new SimpleDateFormat("dd 'de' MMMM 'de' yyyy",
new Locale("ES"));

System.out.println("Hoy es " + formatea.format(hoy));
```

Tanto Date como SimpleDateFormat y Locale deben importarse:

```
import java.text.SimpleDateFormat;

import java.util.Date;

import java.util.Locale;
```

Puedes encontrar más información sobre estas y otras clases predefinidas de Java en la documentación oficial en la web de Oracle.

https://docs.oracle.com/en/java/javase/16/docs/api/index.html