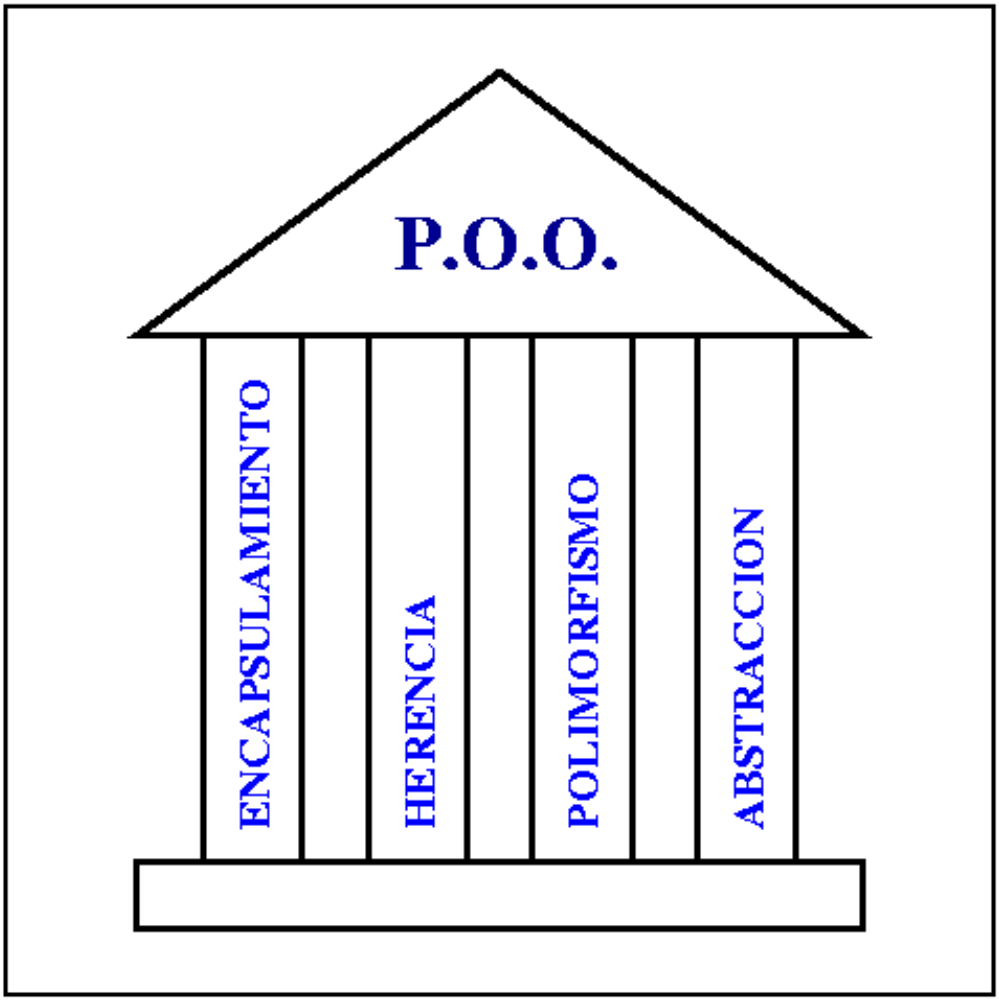
${title}



1. Introducción a la POO

Orientado a objetos hace referencia a una forma diferente de acometer la tarea del desarrollo de software, frente a otros modelos como el de la programación imperativa, la programación funcional o la programación lógica. Supone una reconsideración de los métodos de programación, de la forma de estructurar la información y, ante todo, de la forma de pensar en la resolución de problemas.

La **programación orientada a objetos (POO)** es un modelo para la elaboración de programas que ha impuesto en los últimos años. Este auge se debe, en parte, a que esta forma de programar está fuertemente basada en la representación de la realidad; pero también a que refuerza el uso de buenos criterios aplicables al desarrollo de programas.

> La orientación a objetos no es un tipo de lenguaje de programación. Es una metodología de trabajo para crear programas.

En POO, un programa es una colección de objetos que se relacionan entre sí de distintas formas.

2. Características de la POO

Cuando hablamos de Programación Orientada a Objetos, existen una serie de características que se deben cumplir. Cualquier lenguaje de programación orientado a objetos las debe contemplar. Las características más importantes del paradigma de la programación orientada a objetos son:

- **Abstracción**. Es el proceso por el cual definimos las características más importantes de un objeto, sin preocuparnos de cómo se escribirán en el código del programa, simplemente lo definimos de forma general. En la Programación Orientada a Objetos la herramienta más importante para soportar la abstracción es la clase. Básicamente, una clase es un tipo de dato que agrupa las características comunes de un conjunto de objetos. Poder ver los objetos del mundo real que deseamos trasladar a nuestros programas, en términos abstractos, resulta de gran utilidad para un buen diseño del software, ya que nos ayuda a comprender mejor el problema y a tener una visión global de todo el conjunto. Por ejemplo, si pensamos en una clase Vehículo que agrupa las características comunes de todos ellos, a partir de dicha clase podríamos crear objetos como Coche y Camión. Entonces se dice que Vehículo es una abstracción de Coche y de Camión.
- **Modularidad**. Una vez que hemos representado el escenario del problema en nuestra aplicación, tenemos como resultado un conjunto de objetos software a utilizar. Este conjunto de objetos se crean a partir de una o varias clases. Cada clase se encuentra en un archivo diferente, por lo que la modularidad nos permite modificar las características de la clase que define un objeto, sin que esto afecte al resto de clases de la aplicación.
- **Encapsulación**. También llamada "ocultamiento de la información". La encapsulación o encapsulamiento es el mecanismo básico para ocultar la información de las partes internas de un objeto a los demás objetos de la aplicación. Con la encapsulación un objeto puede ocultar la información que contiene al mundo exterior, o bien restringir el acceso a la misma para evitar ser manipulado de forma inadecuada. Por ejemplo, pensemos en un programa con dos objetos, un objeto Persona y otro Coche. Persona se comunica con el objeto Coche para llegar a su destino, utilizando para ello las acciones que Coche tenga definidas como por ejemplo conducir. Es decir, Persona utiliza Coche pero no sabe cómo funciona internamente, sólo sabe utilizar sus métodos o acciones.
- **Jerarquía**. Mediante esta propiedad podemos definir relaciones de jerarquías entre clases y objetos. Las dos jerarquías más importantes son la jerarquía "es un" llamada generalización o especialización y la jerarquía "es parte de", llamada agregación. Conviene detallar algunos aspectos:
    - La generalización o especialización, también conocida como herencia, permite crear una clase nueva en términos de una clase ya existente (herencia simple) o de varias clases ya existentes (herencia múltiple). Por ejemplo, podemos crear la clase CochedeCarreras a partir de la clase Coche, y así sólo tendremos que definir las nuevas características que tenga.
    - La agregación, también conocida como inclusión, permite agrupar objetos relacionados entre sí dentro de una clase. Así, un Coche está formado por Motor, Ruedas, Frenos y Ventanas. Se dice que Coche es una agregación y Motor, Ruedas, Frenos y Ventanas son agregados de Coche.
- **Polimorfismo**. Esta propiedad indica la capacidad de que varias clases creadas a partir de una antecesora realicen una misma acción de forma diferente. Por ejemplo, pensemos en la clase Animal y la acción de expresarse. Nos encontramos que cada tipo de Animal puede hacerlo de manera distinta, los Perros ladran, los Gatos maullan, las Personas hablamos, etc. Dicho de otra manera, el polimorfismo indica la posibilidad de tomar un objeto (de tipo Animal, por ejemplo), e indicarle que realice la acción de expresarse, esta acción será diferente según el tipo de mamífero del que se trate.

3. Objetos y Clases

3.1. Características de los objetos

En este contexto, un objeto de software es una representación de un objeto del mundo real, compuesto de una serie de características y un comportamiento específico. Pero ¿qué es más concretamente un objeto en Programación Orientada a Objetos? Veámoslo.

> Un objeto es un conjunto de datos con las operaciones definidas para ellos. Los objetos tienen un estado y un comportamiento.

Por tanto, estudiando los objetos que están presentes en un problema podemos dar con la solución a dicho problema. Los objetos tienen unas características fundamentales que los distinguen:

- **Identidad**. Es la característica que permite diferenciar un objeto de otro. De esta manera, aunque dos objetos sean exactamente iguales en sus atributos, son distintos entre sí. Puede ser una dirección de memoria, el nombre del objeto o cualquier otro elemento que utilice el lenguaje para distinguirlos. Por ejemplo, dos vehículos que hayan salido de la misma cadena de fabricación y sean iguales aparentemente, son distintos porque tienen un código que los identifica.
- **Estado**. El estado de un objeto viene determinado por una serie de parámetros o atributos que lo describen, y los valores de éstos. Por ejemplo, si tenemos un objeto `Coche`, el estado estaría definido por atributos como `Marca`, `Modelo`, `Color`, `Cilindrada`, etc.
- **Comportamiento**. Son las acciones que se pueden realizar sobre el objeto. En otras palabras, son los métodos o procedimientos que realiza el objeto. Siguiendo con el ejemplo del objeto Coche, el el comportamiento serían acciones como: `arrancar()`, `parar()`, `acelerar()`, `frenar()`, etc. Definición de clases.

Una clase java se escribe en un fichero con extensión `.java` que tiene el mismo nombre que la clase. Por ejemplo la clase `Vehículo` se escribiría en el fichero `Vehiculo.java`.

Cuando la clase se compila se obtiene un fichero con el mismo nombre que la clase y extensión `.class`. Ej.: `Vehiculo.class`.

> Los identificadores de clase siguen las mismas reglas que otros identificadores de Java (contienen carácter alfanuméricos y especiales, no pueden comenzar por un dígito, no pueden coincidir con una palabra reservada, etc.). Por convenio los identificadores de las clases comienzan por mayúsculas.

3.2. Propiedades y métodos de los objetos

Como acabamos de ver todo objeto tiene un estado y un comportamiento. Concretando un poco más, las partes de un objeto son:

- **Campos, Atributos o Propiedades**: Parte del objeto que almacena los datos. También se les denomina Variables Miembro. Estos datos pueden ser de cualquier tipo primitivo (`boolean`, `char`, `int`, `double`, etc) o ser a su vez otro objeto. Por ejemplo, un objeto de la clase `Coche` puede tener un objeto de la clase `Ruedas` (o más concretamente cuatro).
- **Métodos o Funciones Miembro**: Parte del objeto que lleva a cabo las operaciones sobre los atributos definidos para ese objeto.

La idea principal es que el objeto reúne en una sola entidad los datos y las operaciones, y para acceder a los datos privados del objeto debemos utilizar los métodos que hay definidos para ese objeto.

La única forma de manipular la información del objeto es a través de sus métodos. Es decir, si queremos saber el valor de algún atributo, tenemos que utilizar el método que nos muestre el valor de ese atributo. De esta forma, evitamos que métodos externos puedan alterar los datos del objeto de manera inadecuada. Se dice que los datos y los métodos están encapsulados dentro del objeto.

3.2.1. Atributos

Los atributos representan la información que almacenan los objetos de la clase. Los atributos son declaraciones de variables dentro de la clase.

Se sigue la siguiente sintaxis (los corchetes indican opcionalidad):

```
[ámbito] tipo nombreDelAtributo;
[ámbito] tipo nombreDelAtributo1, nombreDelAtributo2, ...;
```

donde ...

- ***ámbito*** permite indicar desde qué clases es accesible el atributo.
- ***tipo*** indica el tipo de dato del atributo
- ***nombreDelAtributo*** es el identificador del atributo

3.2.2. Métodos

Los métodos determinan qué puede hacer un objeto de la clase, es decir, su comportamiento.

Los métodos realizan algún tipo de acción o tarea y, en ocasiones, devuelven un resultado.

Para realizar su trabajo puede ser necesario que pasemos al método cierta información. Por ejemplo, cuando llamamos al método `round` de la clase `Math`, para redondear un número real, debemos indicar al método cual es el número que queremos redondear. A esa información que pasamos a los métodos se le llama **parámetros** o **argumentos**.

```
//Al llamar a Math.round, pasamos al método un parámetro
int redondeado1 = Math.round(numero);
int redondeado2 = Math.round(125.687);
...
//Al llamar a Math.pow, pasamos al método dos parámetros
int pot1 = Math.pow(a,b);
int pot2 = Math.pow(a,6);
...
```

En la definición de un método se distinguen dos partes

- **La cabecera**, en la cual se indica información relevante sobre el método
- **El cuerpo**, que contiene las instrucciones mediante las cuales el método realiza su tarea

Para definirlos, se sigue la siguiente sintaxis (los corchetes indican opcionalidad):

```
[ámbito] [static] tipoDevuelto nombreDelMetodo ([parámetros]){
    //Cuerpo del método (instrucciones)
    ...
    ...
    ...
}
```

donde ...

- ***ámbito*** permite indicar desde qué clases es accesible el método.
- ***static***, cuando aparece, indica que el método es estático.
- ***tipoDevuelto*** indica el tipo de dato que devuelve el método. La palabra reservada *void* (que no es ningún tipo de dato), indicaría que el método no devuelve nada.
- ***nombreDelMetodo*** es el identificador del método
- ***parámetros*** es una lista, separada por comas, de los parámetros que recibe el método. De cada parámetro se indica el **tipo** y un **identificador**.

3.3. Interacción entre objetos.

Dentro de un programa los objetos se comunican llamando unos a otros a sus métodos. Los métodos están dentro de los objetos y describen el comportamiento de un objeto cuando recibe una llamada a uno de sus métodos. En otras palabras, cuando un objeto, `objeto1`, quiere actuar sobre otro, `objeto2`, tiene que ejecutar uno de sus métodos. Entonces se dice que el `objeto2` recibe un mensaje del `objeto1`.

Un mensaje es la acción que realiza un objeto. Un método es la función o procedimiento al que se llama para actuar sobre un objeto.

Los distintos mensajes que puede recibir un objeto o a los que puede responder reciben el nombre de **protocolo** de ese objeto.

El proceso de interacción entre objetos se suele resumir diciendo que se ha "enviado un mensaje" (hecho una petición) a un objeto, y el objeto determina "qué hacer con el mensaje" (ejecuta el código del método). Cuando se ejecuta un programa se producen las siguientes acciones:

- **Creación** de los objetos a medida que se necesitan.
- **Comunicación** entre los objetos mediante el envío de mensajes unos a otros, o el usuario a los objetos.
- **Eliminación** de los objetos cuando no son necesarios para dejar espacio libre en la memoria del computador.

> Los objetos se pueden comunicar entre ellos invocando a los métodos de los otros objetos.

3.4. Clases

Hasta ahora hemos visto lo que son los objetos. Un programa informático se compone de muchos objetos, algunos de los cuales comparten la misma estructura y comportamiento. Si tuviéramos que definir su estructura y comportamiento del objeto cada vez que queremos crear un objeto, estaríamos utilizando mucho código redundante. Por ello lo que se hace es crear una clase, que es una descripción de un conjunto de objetos que comparten una estructura y un comportamiento común.
Y a partir de la clase, se crean tantas "copias" o "instancias" como necesitemos. Esas copias son los objetos de la clase.

> Las clases constan de datos y métodos que resumen las características comunes de un conjunto de objetos. Un programa informático está compuesto por un conjunto de clases, a partir de las cuales se crean objetos que interactúan entre sí.

En otras palabras, una clase es una plantilla o prototipo donde se especifican:

- Los **atributos** comunes a todos los objetos de la clase.
- Los **métodos** que pueden utilizarse para manejar esos objetos.

Para declarar una clase en Java se utiliza la palabra reservada `class`. La declaración de una clase está compuesta por:

- **Cabecera de la clase**. La cabecera es un poco más compleja que como aquí definimos, pero por ahora sólo nos interesa saber que está compuesta por una serie de modificadores, en este caso hemos puesto

public que indica que es una clase pública a la que pueden acceder otras clases del programa, la palabra reservada class y el nombre de la clase.
- **Cuerpo de la clase**. En él se especifican encerrados entre llaves los atributos y los métodos que va a tener la clase.

Ejemplo:

```
//Paquete al que pertenece la clase
package NombreDePaquete;

//Paquetes que importa la clase
import ...

...

public class NombreDeLaClase {
    // Atributos de la clase
    ...
    ...
    ...
    // Métodos de la clase
    ...
    ...
    ...
}
```

> En la unidad anterior ya hemos utilizado clases, aunque aún no sabíamos su significado exacto. Por ejemplo, en los ejemplos de la unidad o en la tarea, estábamos utilizando clases, todas ellas eran clases principales, no tenían ningún atributo y el único método del que disponían era el método `main()`.

> También es una clase `Math` y su método era `random()`, el que nos permitía usar números aleatorios.

> El método `main()` se utiliza para indicar que se trata de una clase principal, a partir de la cual va a empezar la ejecución del programa. Este método no aparece si la clase que estamos creando no va a ser la clase principal del programa.

3.4.1. ¿Qué significa `public class`?

Significa que la clase que se define es pública. Una clase pública es una clase accesible desde otras clases o, dicho de otra forma, que puede ser utilizada por otras clases. Ya hemos dicho que un programa, de alguna manera, consiste en la creación de objetos de distintas clases, que se relacionan entre sí. Lo más común es que las clases que definimos sean públicas y que en cada fichero de extensión `.java` se defina una única clase.

Sin embargo, en ocasiones se definen clases (A) que solo van a ser utilizadas por una clase determinada (B). En ese caso, decimos que la clase A es una clase privada de la clase B. Las clases A y B se definen en el mismo fichero `.java`. En un fichero pueden definirse varias clases pero solo una de ellas puede ser pública. De esta forma, si en un fichero se definen varias clases, una de ellas sería pública y el resto serían clases privadas de la primera, a las que solo ésta tendría acceso.

3.4.2. Herencia.

La herencia es un mecanismo de la POO que permite definir unas clases de manera que hereden los atributos y los métodos de otras. Esto es interesante para "agrupar" características de diferentes clases con sus características comunes, tanto de atributos como de métodos. En el ejemplo del objeto `Coche` podriamos definir una clase superior del la que heredar llamada `Vehiculo` en la que definiríamos las características comunes de todos los vehículos (`Coche`, `Moto`, `Camion`, etcétera) así como sus métodos comunes (`inchar_rueda()`, `pintar()`, etcétera). Y así las clases hija solo deben especificar aquellos atributos o métodos específicos.

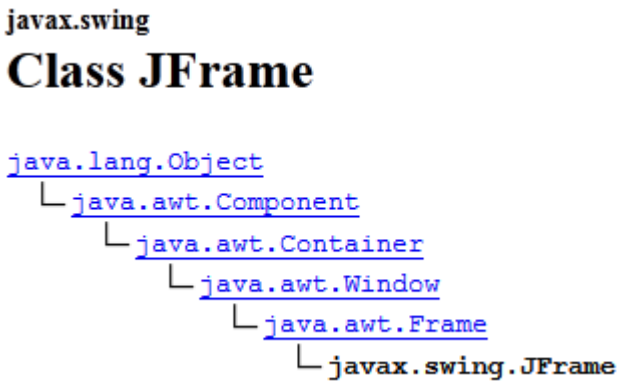
Si una clase H hereda de otra clase P, se dice que

- P es la clase padre o superclase de H
- H es la clase hija o subclase de P

En Java para indicar que una clase hereda de otra se utiliza la palabra reservada ***extends.***

```
1 public class Coche extends Vehiculo {
2     ...
3 }
```

Todas las clases de java tienen una clase padre común que es la clase `Object`. Podemos comprobarlo si miramos la documentación en línea de cualquier clase de Java. En la documentación veremos cuál es su clase padre, y el padre del padre, y así sucesivamente hasta llegar a la clase `Object`, que siempre es el primer elemento de la jerarquía.



3.4.3. Tipos de clases

Aunque la sintaxis es básicamente la misma, en Java suelen crearse clases para tres cosas distintas, lo que nos permite hacer la siguiente clasificación en función de para qué se utilizan:

- **Clases de datos**: Se utilizan para crear, a partir de ellas, objetos que representan elementos que aparecen en el contexto de la aplicación. Por ejemplo en una aplicación de gestión empresarial podrían aparecer clases como `Factura`, `Albarán`, `Pedido`, `Cliente`, etc. Estas clases recogen la información (atributos) y comportamiento (métodos) abstraídos de los objetos del mundo real en el contexto del problema que se esté resolviendo.

  La estructura de este tipo de clases suele ser la que hemos visto anteriormente, es decir, aparecen los atributos de la clase y sus métodos, entre los que se encuentran uno o varios constructores. Solemos encontrar en estas clases tanto métodos estáticos (`static`) como no estáticos. (*Los métodos constructores y estáticos se explican más adelante*)

- **Clases de programa**: Se utilizan para contener al método `main` por el que comienzan los programas. Un programa Java es una clase que contiene, entre otros, al método

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

  Cuando iniciamos la ejecución, ésta comienza siempre por la primera instrucción de su método `main`. Una clase que no tiene método `main` no es un programa Java.

  En estas clases no suelen aparecer atributos, pues su objetivo no es representar algún elemento del mundo real. Suelen estar compuestos por varios métodos estáticos (`static`) entre los que se encuentra el método `main`.

- **Clases de librería**: Se utilizan para colocar en ellas métodos útiles que están relacionados entre sí por su temática. Por ejemplo, la clase `Math`, es una colección de métodos para hacer cálculos matemáticos. Los métodos están en esa clase porque están relacionados con el ámbito de las matemáticas.

  En estas clases tampoco suelen aparecer atributos. Sí puede haber definiciones de constantes. Están formados por uno o varios métodos estáticos (`static`)

4. Utilización de Objetos

Una vez que hemos creado una clase, podemos crear objetos en nuestro programa a partir de esas clases.

Cuando creamos un objeto a partir de una clase se dice que hemos creado una "instancia de la clase". A efectos prácticos, "objeto" e "instancia de clase" son términos similares. Es decir, nos referimos a objetos como instancias cuando queremos hacer hincapié que son de una clase particular.

Los objetos se crean a partir de las clases, y representan casos individuales de éstas.

> Para entender mejor el concepto entre un objeto y su clase, piensa en un molde de galletas y las galletas. El molde sería la clase, que define las características del objeto, por ejemplo su forma y tamaño. Las galletas creadas a partir de ese molde son los objetos o instancias.

Otro ejemplo, imagina una clase Persona que reúna las características comunes de las personas (`color de pelo`, `ojos`, `peso`, `altura`, etc.) y las acciones que pueden realizar (`crecer`, `dormir`, `comer`, etc.). Posteriormente dentro del programa podremos crear un objeto `Trabajador` que esté basado en esa clase Persona. Entonces se dice que el objeto `Trabajador` es una instancia de la clase `Persona`, o que la clase `Persona` es una abstracción del objeto `Trabajador`.

Cualquier objeto instanciado de una clase contiene una copia de todos los atributos definidos en la clase. En otras palabras, lo que estamos haciendo es reservar un espacio en la memoria del ordenador para guardar sus atributos y métodos. Por tanto, cada objeto tiene una zona de almacenamiento propia donde se guarda toda su información, que será distinta a la de cualquier otro objeto. A las variables miembro instanciadas también se les llama variables instancia. De igual forma, a los métodos que manipulan esas variables se les llama métodos instancia.

En el ejemplo del objeto Trabajador, las variables instancia serían `color_de_pelo`, `peso`, `altura`, etc. Y los métodos instancia serían `crecer()`, `dormir()`, `comer()`, etc.

4.1. Ciclo de vida de los objetos.

Todo programa en Java parte de una única clase, que como hemos comentado se trata de la clase principal.

Esta clase ejecutará el contenido de su método `main()`, el cual será el que utilice las demás clases del programa, cree objetos y lance mensajes a otros objetos.

Las instancias u objetos tienen un tiempo de vida determinado. Cuando un objeto no se va a utilizar más en el programa, es destruido por el recolector de basura para liberar recursos que pueden ser reutilizados por otros objetos.

A la vista de lo anterior, podemos concluir que los objetos tienen un ciclo de vida, en el cual podemos distinguir las siguientes fases:

- **Creación**, donde se hace la reserva de memoria e inicialización de atributos.
- **Manipulación**, que se lleva a cabo cuando se hace uso de los atributos y métodos del objeto.
- **Destrucción**, eliminación del objeto y liberación de recursos.

4.2. Declaración.

Para la creación de un objeto hay que seguir los siguientes pasos:

- **Declaración**: Definir el tipo de objeto.
- **Instanciación**: Creación del objeto utilizando el operador `new`.
  Pero ¿en qué consisten estos pasos a nivel de programación en Java? Veamos primero cómo declarar un objeto. Para la definición del tipo de objeto debemos emplear la siguiente instrucción:

```
<tipo> nombre_objeto;
```

Donde:

- **tipo** es la case a partir de la cual se va a crear el objeto, y
- **nombre_objeto** es el nombre de la variable referencia con la cual nos referiremos al objeto.

Los tipos referenciados o referencias se utilizan para guardar la dirección de los datos en la memoria del ordenador.

Nada más crear una referencia, ésta se encuentra vacía. Cuando una referencia a un objeto no contiene ninguna instancia se dice que es una referencia nula, es decir, que contiene el valor `null`.

Esto quiere decir que la referencia está creada pero que el objeto no está instanciado todavía, por eso la referencia apunta a un objeto inexistente llamado "nulo".

Para entender mejor la declaración de objetos veamos un ejemplo. Cuando veíamos los tipos de datos, decíamos que Java proporciona un tipo de dato especial para los textos o cadenas de caracteres que era el tipo de dato `String`. Veíamos que realmente este tipo de dato es un tipo referenciado y creábamos una variable mensaje de ese tipo de dato de la siguiente forma:

```
String mensaje;
```

Los nombres de la clase empiezan con mayúscula, como `String`, y los nombres de los objetos con minúscula, como `mensaje`, así sabemos qué tipo de elemento utilizando.

Pues bien, `String` es realmente la clase a partir de la cual creamos nuestro objeto llamado mensaje (🤯).

Si observas, poco se diferencia esta declaración de las declaraciones de variables que hacíamos para los tipos primitivos. Antes decíamos que `mensaje` era una variable del tipo de dato `String`. Ahora realmente vemos que `mensaje` es un objeto de la clase `String`. Pero mensaje aún no contiene el objeto porque no ha sido instanciado, veamos cómo hacerlo.

Por tanto, cuando creamos un objeto estamos haciendo uso de una variable que almacena la dirección de ese objeto en memoria. Esa variable es una referencia o un tipo de datos referenciado, porque no contiene el dato si no la posición del dato en la memoria del ordenador.

```
String saludo = new String ("Bienvenido a Java");
String s; //s vale null
s = saludo; //asignación de referencias
```

En las instrucciones anteriores, las variables `s` y `saludo` apuntan al mismo objeto de la clase `String`. Esto implica que cualquier modificación en el objeto saludo modifica también el objeto al que hace referencia la variable `s`, ya que realmente son el mismo.

4.3. Instanciación.

Una vez creada la referencia al objeto, debemos crear la instancia u objeto que se va a guardar en esa referencia. Para ello utilizamos la orden `new` con la siguiente sintaxis:

```
nombre_objeto = new <Constructor_de_la_Clase>([<par1>, <par2>, ..., <parN>]);
```

Donde:

- **nombre_objeto** es el nombre de la variable referencia con la cual nos referiremos al objeto.

- **new** es el operador para crear el objeto.
- **Constructor_de_la_Clase** es un método especial de la clase, que se llama igual que ella, y se encarga de inicializar el objeto, es decir, de dar unos valores iniciales a sus atributos.
- **par1-parN**, son parámetros que puede o no necesitar el constructor para dar los valores iniciales a los atributos del objeto.

Durante la instanciación del objeto, se reserva memoria suficiente para el objeto. De esta tarea se encarga Java y juega un papel muy importante el `recolector de basura`, que se encarga de eliminar de la memoria los objetos no utilizados para que ésta pueda volver a ser utilizada.

De este modo, para instanciar un objeto `String`, haríamos lo siguiente:

```
1 mensaje = new String;
```

Así estaríamos instanciando el objeto mensaje. Para ello utilizaríamos el operador `new` y el constructor de la clase `String` a la que pertenece el objeto según la declaración que hemos hecho en el apartado anterior. A continuación utilizamos el constructor, que se llama igual que la clase, `String`.

En el ejemplo anterior el objeto se crearía con la cadena vacía (`""`), si queremos que tenga un contenido debemos utilizar parámetros en el constructor, así:

```
1 mensaje = new String ("El primer programa");
```

Java permite utilizar la clase `String` como si de un tipo de dato primitivo se tratara, por eso no hace falta utilizar el operador `new` para instanciar un objeto de la clase `String` (pero no es lo habitual en el resto de clases).

```
1 mensaje = "El primer programa";
```

La declaración e instanciación de un objeto puede realizarse en la misma instrucción, así:

```
1 String mensaje = new String ("El primer programa");
```

o para la clase `String`:

```
1 String mensaje = "El primer programa";
```

4.4. Manipulación.

Una vez creado e instanciado el objeto ¿cómo accedemos a su contenido? Para acceder a los atributos y métodos del objeto utilizaremos el nombre del objeto seguido del operador punto (`.`) y el nombre del **atributo** o **método** que queremos utilizar. Cuando utilizamos el operador `punto` se dice que estamos enviando un mensaje al objeto. La forma general de enviar un mensaje a un objeto es:

```
1 nombre_objeto.mensaje
```

Por ejemplo, para acceder a las variables instancia o atributos se utiliza la siguiente sintaxis:

```
1 nombre_objeto.atributo
```

Y para acceder a los métodos o funciones miembro del objeto se utiliza la sintaxis es:

```
nombre_objeto.método( [par1, par2, ..., parN] )
```

En la sentencia anterior `par1`, `par2`, etc. son los parámetros que utiliza el método. (*Aparece entre corchetes para indicar son opcionales*).

Para entender mejor cómo se manipulan objetos vamos a utilizar un ejemplo. Para ello necesitamos la Biblioteca de Clases Java o API (Application Programming Interface - Interfaz de programación de aplicaciones). Uno de los paquetes de librerías o bibliotecas es `java.awt`. Este paquete contiene clases destinadas a la creación de objetos gráficos e imágenes. Vemos por ejemplo cómo crear un rectángulo.

En primer lugar instanciamos el objeto utilizando el método constructor, que se llama igual que el objeto, e indicando los parámetros correspondientes a la posición y a las dimensiones del rectángulo:

```
Rectangle rect = new Rectangle(50, 50, 150, 150);
```

Una vez instanciado el objeto rectángulo si queremos cambiar el valor de los atributos utilizamos el operador punto. Por ejemplo, para cambiar la dimensión del rectángulo:

```
rect.height=100;
rect.width=100;
```

O bien podemos utilizar un método para hacer lo anterior:

```
rect.setSize(200, 200);
```

A continuación puedes acceder al código del ejemplo:

```
/*
 * Muestra como se manipulan objetos en Java
 */
import java.awt.Rectangle;

public class Manipular {
    public static void main(String[] args) {
        // Instanciamos el objeto rect indicando posicion y dimensiones
        Rectangle rect = new Rectangle( 50, 50, 150, 150 );
        //Consultamos las coordenadas x e y del rectangulo
        System.out.println( "------- Coordenadas esquina superior izqda. -------");
        System.out.println("\tx = " + rect.x + "\n\ty = " + rect.y);
        // Consultamos las dimensiones (altura y anchura) del rectangulo
        System.out.println( "\n----------------- Dimensiones -------------------");
        System.out.println("\tAlto = " + rect.height );
        System.out.println( "\tAncho = " + rect.width);
        //Cambiar coordenadas del rectangulo
        rect.height=100;
        rect.width=100;
        rect.setSize(200, 200);
        System.out.println( "\n-- Nuevos valores de los atributos --");
        System.out.println("\tx = " + rect.x + "\n\ty = " + rect.y);
        System.out.println("\tAlto = " + rect.height );
        System.out.println( "\tAncho = " + rect.width);
    }
}
```

4.5. Destrucción de objetos y liberación de memoria.

Cuando un objeto deja de ser utilizado, es necesario liberar el espacio de memoria y otros recursos que poseía para poder ser reutilizados por el programa. A esta acción se le denomina destrucción del objeto.

En Java la destrucción de objetos corre a cargo del **recolector de basura** (**garbage collector**). Es un sistema de destrucción automática de objetos que ya no son utilizados. Lo que se hace es liberar una zona de memoria que había sido reservada previamente mediante el operador `new`. Esto evita que los programadores tengan que preocuparse de realizar la liberación de memoria.

El recolector de basura se ejecuta en modo segundo plano y de manera muy eficiente para no afectar a la velocidad del programa que se está ejecutando. Lo que hace es que periódicamente va buscando objetos que ya no son referenciados, y cuando encuentra alguno lo marca para ser eliminado.

Después los elimina en el momento que considera oportuno.

Justo antes de que un objeto sea eliminado por el recolector de basura, se ejecuta su método `finalize()`. Si queremos forzar que se ejecute el proceso de finalización de todos los objetos del programa podemos utilizar el método `runFinalization()` de la clase `System`. La clase `System` forma parte de la Biblioteca de Clases de Java y contiene diversas clases para la entrada y salida de información, acceso a variables de entorno del programa y otros métodos de diversa utilidad. Para forzar el proceso de finalización ejecutaríamos:

```
1 System.runFinalization();
```

5. Utilización de Métodos

Los métodos, junto con los atributos, forman parte de la estructura interna de un objeto. Los métodos contienen la declaración de variables locales y las operaciones que se pueden realizar para el objeto, y que son ejecutadas cuando el método es invocado. Se definen en el cuerpo de la clase y posteriormente son instanciados para convertirse en métodos instancia de un objeto.

Para utilizar los métodos adecuadamente es conveniente conocer la estructura básica de que disponen.

Al igual que las clases, los métodos están compuestos por una cabecera y un cuerpo. La cabecera también tiene modificadores, en este caso hemos utilizado `public` para indicar que el método es público, lo cual quiere decir que le pueden enviar mensajes no sólo los métodos del objeto sino los métodos de cualquier otro objeto externo.

Dentro de un método nos encontramos el cuerpo del método que contiene el código de la acción a realizar. Las acciones que un método puede realizar son:

- **Inicializar** los atributos del objeto
- **Consultar** los valores de los atributos
- **Modificar** los valores de los atributos
- **Llamar a otros métodos**, del mismo del objeto o de objetos externos

5.1. Parámetros y valores devueltos.

Los métodos se pueden utilizar tanto para consultar información sobre el objeto como para modificar su estado. La información consultada del objeto se devuelve a través de lo que se conoce como valor de retorno, y la modificación del estado del objeto, o sea, de sus atributos, se hace mediante la lista de parámetros.
En general, la lista de parámetros de un método se puede declarar de dos formas diferentes:

- **Por valor**. El valor de los parámetros no se devuelve al finalizar el método, es decir, cualquier modificación que se haga en los parámetros no tendrá efecto una vez se salga del método. Esto es así porque cuando se llama al método desde cualquier parte del programa, dicho método recibe una copia de los argumentos, por tanto cualquier modificación que haga será sobre la copia, no sobre las variables originales.
- **Por referencia**. La modificación en los valores de los parámetros sí tienen efecto tras la finalización del método. Cuando pasamos una variable a un método por referencia lo que estamos haciendo es pasar la dirección del dato en memoria, por tanto cualquier cambio en el dato seguirá modificado una vez que salgamos del método.

> En el lenguaje Java, todas las variables se pasan por valor, excepto los objetos que se pasan por referencia.

En Java, la declaración de un método tiene dos restricciones:

- **Un método siempre tiene que devolver un valor (no hay valor por defecto)**. Este valor de retorno es el valor que devuelve el método cuando termina de ejecutarse, al método o programa que lo llamó. Puede ser un tipo primitivo, un tipo referenciado o bien el tipo `void`, que indica que el método no devuelve ningún valor.
- **Un método tiene un número fijo de argumentos**. Los argumentos son variables a través de las cuales se pasa información al método desde el lugar del que se llame, para que éste pueda utilizar dichos valores durante su ejecución. Los argumentos reciben el nombre de parámetros cuando aparecen en la declaración del método.

> El valor de retorno es la información que devuelve un método tras su ejecución.

Según hemos visto en el apartado anterior, la cabecera de un método se declara como sigue:

```
public tipo_de_dato_devuelto nombre_metodo (lista_de_parametros);
```

Como vemos, el tipo de dato devuelto aparece después del modificador `public` y se corresponde con el valor de retorno.

La lista de parámetros aparece al final de la cabecera del método, justo después del nombre, encerrados entre signos de paréntesis y separados por comas. Se debe indicar el tipo de dato de cada parámetro así:

```
(tipo_parámetro1 nombre_parámetro1, ..., tipo_parámetroN nombre_parámetroN)
```

> Cuando se llame al método, se deberá utilizar el nombre del método, seguido de los argumentos que deben coincidir con la lista de parámetros.

La lista de argumentos en la llamada a un método debe coincidir en número, tipo y orden con los parámetros del método, ya que de lo contrario se produciría un error de sintaxis.

5.2. Constructores.

¿Recuerdas cuando hablábamos de la creación e instanciación de un objeto? Decíamos que utilizábamos el operador `new` seguido del nombre de la clase y una pareja de abrir-cerrar paréntesis.

Además, el nombre de la clase era realmente el constructor de la misma, y lo definíamos como un método especial que sirve para inicializar valores. En este apartado vamos a ver un poco más sobre los constructores.

Un constructor es un método especial con el mismo nombre de la clase y que no devuelve ningún valor tras su ejecución.

Cuando creamos un objeto debemos instanciarlo utilizando el constructor de la clase. Veamos la clase `Date` proporcionada por la Biblioteca de Clases de Java. Si queremos instanciar un objeto a partir de la clase `Date` tan sólo tendremos que utilizar el constructor seguido de una pareja de abrir-cerrar paréntesis:

```
Date fecha = new Date();
```

Con la anterior instrucción estamos creando un objeto fecha de tipo `Date`, que contendrá la fecha y hora actual del sistema.

La estructura de los constructores es similar a la de cualquier método, salvo que no tiene tipo de dato devuelto porque no devuelve ningún valor. Está formada por una cabecera y un cuerpo, que contiene la inicialización de atributos y resto de instrucciones del constructor.

El método constructor tiene las siguientes particularidades:

- **El constructor es invocado** automáticamente **en la creación** de un objeto, **y sólo esa vez**.
- **Los constructores no empiezan con minúscula**, como el resto de los métodos, ya que se llaman igual que la clase y las clases empiezan con letra mayúscula.
- **Puede haber varios** constructores para una clase.
- Como cualquier método, **el constructor puede tener parámetros** para definir qué valores dar a los atributos del objeto.
- **El constructor por defecto es aquél que no tiene argumentos o parámetros**. Cuando creamos un objeto llamando al nombre de la clase sin argumentos, estamos utilizando el constructor por defecto.
- **Es necesario que toda clase tenga al menos un constructor.** Si no definimos constructores para una clase, y sólo en ese caso, el compilador crea un constructor por defecto vacío, que inicializa los atributos a sus valores por defecto, según del tipo que sean: `0` para los tipos numéricos, `false` para los `boolean` y `null` para los tipo carácter y las referencias. Dicho constructor lo que hace es llamar al constructor sin

argumentos de la superclase (clase de la cual hereda); si la superclase no tiene constructor sin argumentos se produce un error de compilación.

> Cuando definimos constructores personalizados, el constructor por defecto deja de existir, y si no definimos nosotros un constructor sin argumentos cuando intentemos utilizar el constructor por defecto nos dará un error de compilación.

5.3. El operador `this`.

Los constructores y métodos de un objeto suelen utilizar el operador `this`. Este operador sirve para referirse a los atributos de un objeto cuando estamos dentro de él. Sobre todo se utiliza cuando existe ambigüedad entre el nombre de un parámetro y el nombre de un atributo, entonces en lugar del nombre del atributo solamente escribiremos `this.nombre_atributo`, y así no habrá duda de a qué elemento nos estamos refiriendo.

5.4. Métodos estáticos.

Cuando trabajábamos con cadenas de caracteres utilizando la clase `String`, veíamos las operaciones que podíamos hacer con ellas: obtener longitud, comparar dos cadenas de caracteres, cambiar a mayúsculas o minúsculas, etc. Pues bien, sin saberlo estábamos utilizando métodos estáticos definidos por Java para la clase `String`. Pero ¿qué son los métodos estáticos? Veámoslo.

Los métodos estáticos son aquellos métodos definidos para una clase que se pueden usar directamente, sin necesidad de crear un objeto de dicha clase. También se llaman métodos de clase.

Para llamar a un método estático utilizaremos:

- **El nombre del método**, si lo llamamos desde la misma clase en la que se encuentra definido.
- **El nombre de la clase**, seguido por el operador punto (`.`) más el nombre del método estático, si lo llamamos desde una clase distinta a la que se encuentra definido:

```
1 nombre_clase.nombre_metodo_estatico
```

- **El nombre del objeto**, seguido por el operador punto (`.`) más el nombre del método estático. Utilizaremos esta forma cuando tengamos un objeto instanciado de la clase en la que se encuentra definido el método estático, y no podamos utilizar la anterior:

```
1 nombre_objeto.nombre_metodo_estatico
```

Los métodos estáticos no afectan al estado de los objetos instanciados de la clase (variables instancia), y suelen utilizarse para realizar operaciones comunes a todos los objetos de la clase. Por ejemplo, si necesitamos contar el número de objetos instanciados de una clase, podríamos utilizar un método estático que fuera incrementando el valor de una variable entera de la clase conforme se van creando los objetos.

En la Biblioteca de Clases de Java existen muchas clases que contienen métodos estáticos. Pensemos en las clases que ya hemos utilizado en unidades anteriores, como hemos comentado la clase `String` con todas las operaciones que podíamos hacer con ella y con los objetos instanciados a partir de ella. O bien la clase `Math` para la conversión de tipos de datos. Todos ellos son métodos estáticos que la API de Java define para esas clases. Lo importante es que tengamos en cuenta que al tratarse de métodos estáticos, para utilizarlos no necesitamos crear un objeto de dichas clases.

Fijémonos en esta secuencia de instrucciones

```
//Creamos dos círculos de radio 100  en distintas posiciones
Circulo c1 = new Circulo(50,50,100);
Circulo c2 = new Circulo(80,80,100);
...
//Aumentamos el radio del primer círculo a 200
c1.setRadio(200);
```

y en esta otra

```
System.out.println(Math.sqrt(4));
```

En el primer ejemplo, `.setRadio(200)` va precedido por un objeto. La variable `c1` es un objeto de la clase Círculo, por tanto, la instrucción está modificando el radio de un círculo concreto, el que se encuentra en la posición (50,50) . El método `setRadio` es un método no estático. Los métodos no estáticos actúan siempre sobre algún objeto (el que figura a la izquierda del punto).

En el segundo ejemplo, en cambio, a la izquierda de `.sqrt(4)` no se ha puesto el nombre de un objeto, sino el de una clase, la clase `Math`. El método `sqrt` no está actuando sobre un objeto concreto: no tiene sentido hacerlo, solo pretendemos calcular la raíz cuadrada de `4`. `Sqrt` es un método estático. Los métodos estáticos se usan poniendo delante del punto el nombre de la clase en que se encuentran definidos.

5.5. Métodos sobrecargados

Se dice que un operador o un método están **sobrecargados** cuando se utiliza para varias cosas distintas. Por ejemplo,

- El símbolo + está sobrecargado. Se utiliza para:
  - Operador aritmético suma:

    ```
    a = b + c;
    ```

  - Concatenación de cadenas de caracteres:

    ```
    System.out.println("Edad:" + edad);
    ```

- El símbolo **-** está sobrecargado. Se utiliza para:
  - Operador aritmético resta:

    ```
    a = b - c;
    ```

  - Cambio de signo:

    ```
    a = -b;
    ```

- El símbolo **/** está sobrecargado. Se utiliza para:
  - División entera:

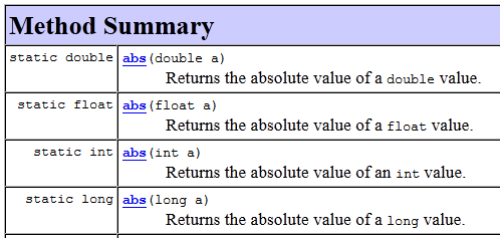
    ```
    a = 5 / 2;
    ```

  - División real:

```
a = 5 / 2.0;
```

Del mismo modo, los métodos también pueden estar sobrecargados de forma que, en una misma clase puede haber dos métodos distintos que tengan el mismo nombre. Para que esto sea posible Java los métodos se tienen que diferenciar en el tipo o número de parámetros que recibe.

En la siguiente imagen de la documentación de la clase `Math` vemos como hay cuatro métodos distintos llamados `abs`. Estos métodos se diferencian en el tipo del número para el cual calculan el valor absoluto. Aunque los cuatro métodos realizan la misma tarea (calcular el valor absoluto de un número), se trata de métodos distintos.



Method Summary

static double	abs(double a) Returns the absolute value of a double value.
static float	abs(float a) Returns the absolute value of a float value.
static int	abs(int a) Returns the absolute value of an int value.
static long	abs(long a) Returns the absolute value of a long value.

> Es muy habitual que el método constructor esté sobrecargado. Por ejemplo un método sin parámetros (que seria el constructor por defecto), y otro con diferentes cantidades de parámetros que permitirán instanciar objetos con más o menos atributos.

6. Librerías de Objetos (Paquetes).

Conforme nuestros programas se van haciendo más grandes, el número de clases va creciendo.
Meter todas las clases en único directorio no ayuda a que estén bien organizadas, lo mejor es hacer grupos de clases, de forma que todas las clases que estén relacionadas o traten sobre un mismo tema estén en el mismo grupo.

Un **paquete** de clases es una agrupación de clases que consideramos que están relacionadas entre sí o tratan de un tema común.

Las clases de un mismo paquete tienen un acceso privilegiado a los atributos y métodos de otras clases de dicho paquete. Es por ello por lo que se considera que los paquetes son también, en cierto modo, unidades de encapsulación y ocultación de información.

Java nos ayuda a organizar las clases en paquetes. En cada fichero `.java` que hagamos, al principio, podemos indicar a qué paquete pertenece la clase que hagamos en ese fichero.

Los paquetes se declaran utilizando la palabra clave `package` seguida del nombre del paquete.

Para establecer el paquete al que pertenece una clase hay que poner una sentencia de declaración como la siguiente al principio de la clase:

```
1 package Nombre_de_Paquete;
```

Por ejemplo, si decidimos agrupar en un paquete `ejemplos` un programa llamado `Bienvenida`, pondríamos en nuestro fichero `Bienvenida.java` lo siguiente:

```
1 package ejemplos;
2
3 public class Bienvenida {
4     [...]
5 }
```

El código es exactamente igual que como hemos venido haciendo hasta ahora, solamente hemos añadido la línea `package ejemplos;` al principio.

6.1. Sentencia `import`.

Cuando queremos utilizar una clase que está en un paquete distinto a la clase que estamos utilizando, se suele utilizar la sentencia `import`. Por ejemplo, si queremos utilizar la clase `Scanner` que está en el paquete `java.util` de la Biblioteca de Clases de Java, tendremos que utilizar esta sentencia:

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

Se pueden importar todas las clases de un paquete, así:

```
1 import java.awt.*;
```

Esta sentencia debe aparecer al principio de la clase, justo después de la sentencia package, si ésta existiese.

También podemos utilizar la clase sin sentencia `import`, en cuyo caso cada vez que queramos usarla debemos indicar su ruta completa:

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

Hasta aquí todo correcto. Sin embargo, al trabajar con paquetes, Java nos obliga a organizar los directorios, compilar y ejecutar de cierta forma para que todo funcione adecuadamente.

6.2. Compilar y ejecutar clases con paquetes.

Si hacemos que `Bienvenida.java` pertenezca al paquete ejemplos, debemos crear un subdirectorio `ejemplos` y meter dentro el archivo `Bienvenida.java`.
Por ejemplo, en GNU/Linux tendríamos esta estructura de directorios:

```
/<directorio_usuario>/Proyecto_Bienvenida/ejemplos/Bienvenida.java
```

Debemos tener cuidado con las mayúsculas y las minúsculas, para evitar problemas, tenemos que poner el nombre en `package` exactamente igual que el nombre del subdirectorio.

Para compilar la clase `Bienvenida.java` que está en el paquete ejemplos debemos situarnos en el directorio padre del paquete y compilar desde ahí:

```
$ cd /<directorio_usuario>/Proyecto_Bienvenida
$ javac ejemplos/Bienvenida.java
```

Si todo va bien, en el directorio ejemplos nos aparecerá la clase compilada `Bienvenida.class`.

Para ejecutar la clase compilada `Bienvenida.class` que está en el directorio ejemplos, debemos seguir situados en el directorio padre del paquete. El nombre completo de la clase es `paquete.clase`, es decir `ejemplos.Bienvenida`. Los pasos serían los siguientes:

```
$ cd /<directorio_usuario>/Proyecto_Bienvenida
$ java ejemplos.Bienvenida
Bienvenido a Java
```

Si todo es correcto, debe salir el mensaje `Bienvenido a Java` por la pantalla.

6.3. Jerarquía de paquetes.

Para organizar mejor las cosas, un paquete, en vez de clases, también puede contener otros paquetes. Es decir, podemos hacer subpaquetes de los paquetes y subpaquetes de los subpaquetes y así sucesivamente. Esto permite agrupar paquetes relacionados en un paquete más grande. Por ejemplo, si quiero dividir mis clases de ejemplos en ejemplos básicos y ejemplos avanzados, puedo poner más niveles de paquetes separando por puntos:

```
package ejemplos.basicos;
package ejemplos.avanzados;
```

A nivel de sistema operativo, tendríamos que crear los subdirectorios `basicos` y `avanzados` dentro del directorio `ejemplos`, y meter ahí las clases que correspondan.
Para compilar, en el directorio del proyecto habría que compilar poniendo todo el path hasta llegar a la clase. Es decir, el nombre de la clase va con todos los paquetes separados por puntos, por ejemplo `ejemplos.basicos.Bienvenida`.

La estructura de directorios en el sistema operativo cuando usamos subpaquetes sería:

```
/<directorio_usuario>/Proyecto_Bienvenida/ejemplos/basicos/HolaMundo.java
```

Y la compilación y ejecución sería:

```
$ cd /<directorio_usuario>/Proyecto_Bienvenida
$ javac ejemplos/basicos/Bienvenida.java
$ java ejemplos/basicos/Bienvenida
Hola Mundo
```

La Biblioteca de Clases de Java se organiza haciendo uso de esta jerarquía de paquetes. Así por ejemplo, si quiero acceder a la clase `Date`, tendré que importarla indicando su ruta completa, o sea, `java.util.Date`, así:

```
import java.util.Date;
```

6.4. Importar paquetes de usuario

Para encontrar una clase Java necesitamos el nombre del paquete y la ruta donde están situados los paquetes y las clases.

La variable de entorno `CLASSPATH` sirve para localizar las clases creadas por el usuario y que no forman parte de la plataforma Java. Debemos establecer el valor de la variable cuando el paquete o la clase no se encuentre en la misma carpeta donde se está trabajando o en ningún lugar definido en la `CLASSPATH`.

```
CLASSPATH = c:\mijava\paquetes; export CLASSPATH
```

En el ejemplo anterior añadimos la ruta **c:\mijava\paquetes** a la variable de entorno `CLASSPATH` para que localice paquetes y clases definidas por el usuario en dicha ruta.

También podemos especificar el `CLASSPATH` en el momento de ejecutar `java` con el parámetro `-cp`:

```
java -cp c:\mijava\paquetes MiClase.java
```

6.5. Librerías Java.

Cuando descargamos el entorno de compilación y ejecución de Java, obtenemos la API de Java. Como ya sabemos, se trata de un conjunto de bibliotecas que nos proporciona paquetes de clases útiles para nuestros programas.
Utilizar las clases y métodos de la Biblioteca de Java nos va ayudar a reducir el tiempo de desarrollo considerablemente, por lo que es importante que aprendamos a consultarla y conozcamos las clases más utilizadas.
Ejemplo:

```
import java.lang.System; // Se importa la clase System.
import java.awt.*;       // Se importa todas las clases del paquete awt;
```

Los paquetes más importantes que ofrece el lenguaje Java son:

Paquete o librería	Descripción
java.io	Contiene las clases que gestionan la entrada y salida, ya sea para manipular ficheros, leer o escribir en pantalla, en memoria, etc. Este paquete contiene por ejemplo la clase BufferedReader que se utiliza para la entrada por teclado.
java.lang	Contiene las clases básicas del lenguaje. Este paquete no es necesario importarlo, ya que es importado automáticamente por el entorno de ejecución. En este paquete se encuentra la clase Object, que sirve como raíz para la jerarquía de clases de Java, o la clase System que ya hemos utilizado en algunos ejemplos y que representa al sistema en el que se está ejecutando la aplicación. También podemos encontrar en este paquete las clases que "envuelven" los tipos primitivos de datos. Lo que proporciona una serie de métodos para cada tipo de dato de utilidad, como por ejemplo las conversiones de datos.
java.util	Biblioteca de clases de utilidad general para el programador. Este paquete contiene por ejemplo la clase Scanner utilizada para la entrada por teclado de diferentes tipos de datos, la clase Date, para el tratamiento de fechas, etc.
java.math	Contiene herramientas para manipulaciones matemáticas.
java.awt	Incluye las clases relacionadas con la construcción de interfaces de usuario, es decir, las que nos permiten construir ventanas, cajas de texto, botones, etc. Algunas de las clases que podemos encontrar en este paquete son Button, TextField, Frame, Label, etc.
java.swing	Contiene otro conjunto de clases para la construcción de interfaces avanzadas de usuario. Los componentes que se engloban dentro de este paquete se denominan componentes Swing, y suponen una alternativa mucho más potente que AWT para construir interfaces de usuario.
java.net	Conjunto de clases para la programación en la red local e Internet.
java.sql	Contiene las clases necesarias para programar en Java el acceso a las bases de datos.
java.security	Biblioteca de clases para implementar mecanismos de seguridad.

Como se puede comprobar Java ofrece una completa jerarquía de clases organizadas a través de paquetes.

7. Cadenas de caracteres. La clase String

7.1. Cadenas de caracteres.

Hasta ahora hemos utilizado literales de cadenas de caracteres que, como sabemos, se ponen entre comillas dobles, como en la siguiente expresión

```
System.out.println("Hola");
```

Para almacenar cadenas de caracteres en variables se utiliza la clase `String`. `String` se encuentra definida en el paquete `java.lang`. *Recordemos que no es necesario importar este paquete para utilizar sus clases.*

La forma de `String` es la siguiente:

```
String variable = new String("texto");
```

Ejemplo:

```
String nombre = new String("Javier");
System.out.println("Mi nombre es " + nombre);
```

Sin embargo, debido a que es una clase que se utiliza ampliamente en los programas, Java permite una forma abreviada de crear objetos String:

```
String nombreVariable = "texto";
```

Ejemplo:

```
String nombre = "Javier";
System.out.println("Mi nombre es " + nombre);
```

7.2. Leer cadenas desde teclado.

7.2.1. Clase `Scanner`

Para leer cadenas de caracteres desde teclado podemos utilizar la clase `Scanner`. Ésta dispone de dos métodos para leer cadenas:

- `next()`: Lee desde la entrada estándar (teclado) una secuencia de caracteres hasta encontrar un delimitador (un espacio). Devuelve un `String`.
- `nextLine()`: Lee desde la entrada estándar (teclado) una secuencia de caracteres hasta encontrar un salto de línea. Devuelve un `String`.

Ejemplo:

```
Scanner tec = new Scanner(System.in);
//De lo que introduce el usuario, lee la 1º palabra.
String nombre = tec.next();
//Lee lo que introduce el usuario hasta que pulsa intro.
String nombreCompleto = tec.nextLine();
```

7.2.2. Ejemplos de la UD01 pero utilizando `Scanner` (compatible con los IDE's)

A continuación vamos a ver los mismos ejemplos de la UD01, pero utilizando la clase `Scanner` que si es compatible con los IDE's. Para poder usar la clase `Scanner` necesitamos importar el paquete: `java.util.Scanner`.

```java
import java.util.Scanner;

public class EjemploUD02 {

    public static void main(String[] args) {

        Scanner teclado = new Scanner(System.in);

        //Introducir texto desde teclado
        String texto;
        System.out.print("Introduce un texto: ");
        texto = teclado.nextLine();
        System.out.println("El texto introducido es: "+ texto);

        //Introducir un número entero desde teclado
        String texto2;
        int entero2;
        System.out.print("Introduce un número: ");
        texto2 = teclado.nextLine();
        entero2 = Integer.parseInt(texto2);
        System.out.println("El número introducido es:"+entero2);

        //Introducir un número decimal desde teclado
        String texto3;
        double doble3;
        System.out.print("Introduce un número decimal: ");
        texto3 = teclado.nextLine();
        doble3 = Double.parseDouble(texto3); // convertimos texto a doble
        System.out.println("Número decimal introducido es: "+doble3);
    }
}
```

7.3. La clase `String`.

Además de permitir almacenar cadenas de caracteres, `String` tiene métodos para realizar cálculos u operaciones con ellas.

Así por ejemplo, la clase tiene un método `toUpperCase()` que devuelve el `String` convertido a mayúsculas. El siguiente ejemplo ilustra su uso:

```java
String nombre = "Javier";
System.out.println(nombre.toUpperCase()); // Se muestra JAVIER por pantalla
```

Accede a la documentación en línea de Java y estudia los siguientes métodos de la clase:

- `charAt`
- `indexOf`

- `subString`
- `toLowerCase`
- `trim`

7.4. `printf` o `format`

El método `printf()` o `format()` (son sinónimos) utilizan unos códigos de conversión para indicar si el contenido a mostrar de qué tipo es. Estos códigos se caracterizan porque llevan delante el símbolo %, algunos de ellos son:

- `%c`: Escribe un carácter.
- `%s`: Escribe una cadena de texto.
- `%d`: Escribe un entero.
- `%f`: Escribe un número en punto flotante.
- `%e`: Escribe un número en punto flotante en notación científica.

Por ejemplo, si queremos escribir el número float `12345.1684` con el punto de los miles y sólo dos cifras decimales la orden sería:

```
1 System.out.printf("%,.2f\n", 12345.1684);
```

Esta orden mostraría el número `12.345,17` por pantalla.

Otro ejemplo seria:

```
1 System.out.format("El valor de la variable float es" +
2      "%f, mientras que el valor del entero es %d" +
3      "y el string contiene %s", variableFloat, variableInt, variableString);
```

Puedes investigar más sobre `printf` o `format` en este enlace

7.5. Salida de error.

La salida de error está representada por el objeto `System.err`. No parece muy útil utilizar `out` y `err` si su destino es la misma pantalla, o al menos en el caso de la consola del sistema donde las dos salidas son representadas con el mismo color y no notamos diferencia alguna. En cambio en la consola de varios entornos integrados de desarrollo como NetBeans o Eclipse la salida de `err` se ve en un color diferente. Teniendo el siguiente código:

```
1 System.out.println("Salida estándar por pantalla");
2 System.err.println("Salida de error por pantalla");
```

Tanto NetBeans como Eclipse mostraran el mensaje `err` en color rojo.

8. Ejemplo UD02

```java
package UD02;

import java.util.Scanner;

public class EjemploUD02 {

    public static void main(String[] args) {

        Scanner teclado = new Scanner(System.in);

        //Introducir texto desde teclado
        String texto;
        System.out.print("Introduce un texto: ");
        texto = teclado.nextLine();
        System.out.println("El texto introducido es: " + texto);

        //Introducir un número entero desde teclado
        String texto2;
        int entero2;
        System.out.print("Introduce un número: ");
        texto2 = teclado.nextLine();
        entero2 = Integer.parseInt(texto2);
        System.out.println("El número introducido es:" + entero2);

        //Introducir un número decimal desde teclado
        String texto3;
        double doble3;
        System.out.print("Introduce un número decimal: ");
        texto3 = teclado.nextLine();
        doble3 = Double.parseDouble(texto3); // convertimos texto a doble
        System.out.println("Número decimal introducido es: " + doble3);

        System.out.println("La clase String");
        String nombre = "Javier "; //Observa que hay un espacio final
        System.out.println(nombre.toUpperCase()); //JAVIER
        System.out.println(nombre.charAt(4)); //E
        System.out.println(nombre.indexOf("i"));//3
        System.out.println(nombre.substring(0, 3));//JAVI
        System.out.println(nombre.toLowerCase());//javier
        System.out.println(nombre.trim());//Javier sin espacios finales
        System.out.printf("%,.2f\n", 12345.1684);

        System.out.format("El valor de la variable float es %f"
                + ", mientras que el valor del entero es %d"
                + " y el string contiene %s", doble3, entero2, texto);

        System.err.println("Salida de error por pantalla");
    }
}
```

8.1. Clase `Pajaro`

Vamos a ilustrar mediante un ejemplo la utilización de objetos y métodos, así como el uso de parámetros y el operador `this`. Aunque la creación de clases la veremos en las siguientes unidades, en este ejercicio creamos una pequeña clase para que podamos instanciar el objeto con el que vamos a trabajar.

Las clases se suelen representar como un rectángulo, y dentro de él se sitúan los atributos y los métodos de dicha clase.

En la imagen, la clase `Pajaro` está compuesta por tres atributos, uno de ellos el nombre y otros dos que indican la posición del ave, `posX` y `posY`. Tiene dos métodos constructores y un método `volar()`. Como sabemos, los métodos constructores reciben el mismo nombre de la clase, y puede haber varios para una misma clase, dentro de ella se diferencian unos de otros por los parámetros que utilizan.

Enunciado:

> Dada una clase principal llamada `Pajaro`, se definen los atributos y métodos que aparecen en la imagen. Los métodos realizan las siguientes acciones:
>
> - `pajaro()`. Constructor por defecto. En este caso, el constructor por defecto no contiene ninguna instrucción, ya que Java inicializa de forma automática las variables miembro, si no le damos ningún valor.
> - `pajaro(String nombre, int posX, int posY)`. Constructor que recibe como argumentos una cadena de texto y dos enteros para inicializar el valor de los atributos.
> - `volar(int posX, int posY)`. Método que recibe como argumentos dos enteros: `posX` y `posY`, y devuelve un valor de tipo `double` como resultado, usando la palabra clave `return`. El valor devuelto es el resultado de aplicar un desplazamiento de acuerdo con la siguiente fórmula:
>
> $$desplazamiento = \sqrt{posX \cdot posX + posY \cdot posY}$$
>
> Diseña un programa que utilice la clase `Pajaro`, cree una instancia de dicha clase y ejecute sus métodos.

Lo primero que debemos hacer es crear la clase `Pajaro`, con sus métodos y atributos. De acuerdo con los datos que tenemos, el código de la clase sería el siguiente:

```
1  public class Pajaro {
2
3      String nombre;
4      int posX, posY;
5
6      public Pajaro() {
7      }
8
9      public Pajaro(String nombre) {
10         this.nombre = nombre;
11     }
12
13     public Pajaro(String nombre, int posX, int posY) {
14         this.nombre = nombre;
15         this.posX = posX;
16         this.posY = posY;
17     }
18
19     double volar(int posX, int posY) {
20         double desplazamiento = Math.sqrt(posX * posX + posY * posY);
21         this.posX = posX;
22         this.posY = posY;
23         return desplazamiento;
24     }
```

```
    //método main()
    [...]
}
```

Debemos tener en cuenta que se trata de una clase principal, lo cual quiere decir que debe contener un método `main()` dentro de ella. En el método `main()` vamos a situar el código de nuestro programa. El ejercicio dice que tenemos que crear una instancia de la clase y ejecutar sus métodos, entre los que están el constructor y el método `volar()`.

También es conveniente imprimir el resultado de ejecutar el método `volar()`. Por tanto, lo que haría el programa sería:

- Crear un objeto de la clase e inicializarlo.
- Invocar al método volar.
- Imprimir por pantalla la distancia recorrida.

Para inicializar el objeto utilizaremos el constructor con parámetros, después ejecutaremos el método `volar()` del objeto creado y finalmente imprimiremos el valor que nos devuelve el método.

Luego crearemos otro pajaro2 usando el constructor por defecto (sin parámetros). Le asignaremos el nombre y la posición manualmente, y calcularemos su desplazamiento llamando al método, pero usando los atributos del objeto (pajaro2.posX y pajaro2.posY) en lugar de constantes. El código del método `main()` quedaría como sigue:

```
public static void main(String[] args) {
    //creamos el objeto con parámetros
    Pajaro pajaro1 = new Pajaro("WoodPecker", 50, 50);
    double d1 = pajaro1.volar(50, 50);
    System.out.println("El desplazamiento de " + pajaro1.nombre + " ha sido " + d1);

    Pajaro pajaro2 = new Pajaro();
    //damos nombre y cambiamos la posición de "Piolin" a mano
    pajaro2.nombre="Piolín";
    pajaro2.posX=30;
    pajaro2.posY=30;
    double d2 = pajaro2.volar(pajaro2.posX, pajaro2.posY);
    System.out.println("El desplazamiento de " + pajaro2.nombre + " ha sido " + d2);
}
```

Si ejecutamos nuestro programa el resultado sería el siguiente:

```
El desplazamiento de WoodPecker ha sido 70.71067811865476
El desplazamiento de Piolín ha sido 42.42640687119285
```

9. Píldoras informáticas relacionadas

- Curso Java. Manipulación de cadenas. Clase String I. Vídeo 11
- Curso Java. Manipulación de cadenas. Clase String II. Vídeo 12
- Curso Java. Entrada Salida datos I. Vídeo 14
- Curso Java. Entrada Salida datos II. Vídeo 15
- Curso Java. POO I. Vídeo 27
- Curso Java. POO II. Vídeo 28
- Curso Java. POO III. Vídeo 29
- Curso Java POO VI. Construcción objetos. Vídeo 32
- Curso Java POO VII. Construcción objetos II. Vídeo 33
- Curso Java POO VIII. Construcción objetos III. Vídeo 34
- Curso Java POO IX. Construcción objetos IV. Vídeo 35
- Curso Java. Métodos static. Vídeo 38
- Curso Java. Sobrecarga de constructores. Vídeo 39

10. Fuentes de información

- Wikipedia
- Programación (Grado Superior) - Juan Carlos Moreno Pérez (Ed. Ra-ma)
- Apuntes IES Henri Matisse (Javi García Jimenez?)
- Apuntes AulaCampus
- Apuntes José Luis Comesaña
- Apuntes IOC Programació bàsica (Joan Arnedo Moreno)
- Apuntes IOC Programació Orientada a Objectes (Joan Arnedo Moreno)