**Objetos y clases: Introducción**

**El concepto de clase**

Si pensamos en las **clases de objetos** que vemos, tenemos coches, bares, semáforos, personas…

Si simplificamos esta definición de clases de objetos por simplemente clases:

¿Qué **clases** vemos? Coches, bares, semáforos, personas…

Y ya estaremos usando terminología de la **programación orientada a objetos**:

Tenemos la **clase** Coche, la clase Bar, la clase Semáforo, la clase Persona…

No nos importa si esa persona es Marta, está en el bar “Casa Manolo”, si tiene un Seat Panda o si el semáforo está en rojo. Estamos pensando en términos **clasificatorios**.

Vamos a pensar ahora en cada una de estas clases:

¿Cómo son los coches? ¿Qué **características** pueden diferenciar uno de otro? Por ejemplo, podríamos indicar la marca, el modelo, el color, la cilindrada, etc.

A las características las vamos a denominar **atributos**, término muy utilizado en la **programación orientada a objetos**.

|  |
| --- |
| ¿Qué atributos tiene la clase Bar? Todo bar tiene un nombre, una ubicación, un estado (si está abierto o cerrado), una lista de precios,… |
| Como atributos de la clase Semáforo podríamos indicar su ubicación y estado (rojo, verde o ámbar). |
| La clase Persona podría definir como atributos el nombre, sexo, edad, estado civil, altura, etc. |

A continuación vamos a pensar en el **comportamiento** de estas clases. Nos preguntaremos qué cosas hacen, qué tipo de acciones pueden realizar...

|  |
| --- |
| Un coche puede arrancar, detenerse, girar a la izquierda, acelerar, frenar, encender sus luces. |
| Un semáforo puede cambiar de estado. |
| Un bar puede abrir, cerrar, servirte una cerveza, cobrarla, modificar la lista de precios.  Una persona puede hablar, dormir, conducir un coche, tomarse una cerveza en un bar. |

En terminología de la **programación orientada a objetos**, a estas funciones que determinan el comportamiento de una clase se las conoce como **métodos**.

|  |
| --- |
| Sabemos lo que es una **clase** y que está compuesta de **atributos** y **métodos**. Nuestra labor ahora consistirá en **modelar** en Java estas clases. |

**Modelar** no es otra cosa sino crear una **abstracción** que represente de algún modo una determinada realidad.

Para crear en Java la clase Coche procederíamos del siguiente modo:

|  |
| --- |
| class Coche |
| { |
| } |

Entre las llaves introduciremos los atributos y métodos de que consta la clase. Usamos para ello la palabra reservada **class**.

El nombre Coche lo escribimos con la primera letra en mayúsculas, pues es de común acuerdo entre los programadores en Java que los nombres de clases empiecen así.

Introduzcamos algunos atributos:

|  |
| --- |
| class Coche |
| { |
| String marca; |
| String modelo; |
| String color; |
| int numeroDePuertas; |
| int cuentaKilometros; |
| int velocidad; |
| boolean arrancado; |
| } |

**Declarar atributos** es algo similar a declarar una variable normal. El nombre del atributo se precede por su tipo. Así, en el ejemplo, tenemos tres atributos de tipo String que contendrán cadenas de caracteres; otros tres de tipo int para almacenar valores enteros; finalmente, el atributo arrancado, que utilizaremos para indicar si el coche está en marcha o no, es de tipo boolean, admitiendo como posibles valores true o false.

Los tipos int y boolean forman parte de los tipos básicos de Java, conocidos como tipos **primitivos**. Los presentaremos formalmente a su debido momento; por ahora es suficiente con que conozcamos su existencia y cómo los utilizamos.

El tipo String, que escribimos con la primera letra en mayúsculas (lo que debería darte una pista), no es más que otra clase, como lo son las clases Coche y Persona. Es una clase muy importante en Java que **encapsula** un buen conjunto de métodos para trabajar con cadenas de caracteres.

El concepto importante que debes entender es que los *atributos no necesariamente son siempre de tipos básicos*, sino que también pueden ser de cualquier clase, incluso de una propia que nosotros mismos hayamos creado.

Por ejemplo, podríamos definir un nuevo atributo de la clase Coche, llamado conductor, en el que figure la persona (de tipo Persona) que lo conduce:

|  |
| --- |
| Persona conductor; |
|  |

Fíjate también en la convención utilizada para nombres de variables compuestos de varias palabras. Se escriben todas juntas, pero iniciando cada palabra en mayúsculas, a excepción de la primera. Esto forma parte también del estilo de escritura Java.

Ten en cuenta que Java distingue mayúsculas de minúsculas. No es lo mismo numeroDePuertas que numerodepuertas.

Definamos ahora los métodos de la clase Coche:

|  |
| --- |
| class Coche |
| { |
| String marca; |
| String modelo; |
| String color; |
| int numeroDePuertas; |
| int cuentaKilometros; |
| int velocidad; |
| boolean arrancado; |
|  |
| void arrancar() |
| { |
| } |
|  |
| void parar() |
| { |
| } |
|  |
| void acelerar() |
| { |
| } |
|  |
| void frenar() |
| { |
| } |
|  |
| void pitar() |
| { |
| } |
|  |
| int consultarCuentaKilometros() |
| { |
| } |
| } |

El nombre de método viene seguido por un par de paréntesis, pues en ocasiones los métodos podrán recibir argumentos que luego se utilizarán en el cuerpo del método. Aunque el método no requiera argumentos los paréntesis son absolutamente necesarios.

Por otro lado, el nombre del método va precedido por el tipo del valor que devuelve. Hay que indicarlo incluso si el método no devuelve explícitamente ningún valor. Ese caso se indica con el tipo **void**.

Entre el par de llaves { } introduciremos el cuerpo del método, las instrucciones que indican su operatividad.

Escribamos algo de código básico en cada uno de ellos:

|  |
| --- |
| class Coche |
| { |
| String marca; |
| String modelo; |
| String color; |
| int numeroDePuertas; |
| int cuentaKilometros; |
| int velocidad; |
| boolean arrancado; |
|  |
| void arrancar() |
| { |
| arrancado = true; |
| } |
|  |
| void parar() |
| { |
| arrancado = false; |
| } |
|  |
| void acelerar() |
| { |
| velocidad = velocidad + 1; |
| } |
|  |
| void frenar() |
| { |
| velocidad = velocidad - 1; |
| } |
|  |
| void pitar() |
| { |
| System.out.println("Piiiiiiiiiiiiiiiiii"); |
| } |
|  |
| int consultarCuentaKilometros() |
| { |
| return cuentaKilometros; |
| } |
| } |
|  |

Los métodos arrancar() y parar() establecen el atributo booleano arrancado a true y false, respectivamente. Los métodos acelerar() y frenar() incrementan y decrementan en una unidad, respectivamente, el atributo velocidad. El método pitar() imprime en consola una cadena de caracteres. El método consultarCuentaKilometros() devuelve **a quien lo invoca** (fíjate en el return) lo que contiene el atributo cuentaKilometros. Observa que es el único que devuelve explícitamente un valor (de tipo entero); los restantes, aunque algunos modifican los atributos de la misma clase, no devuelven ningún valor a quien los llama.

Podríamos modelar la clase Persona del siguiente modo:

|  |
| --- |
| class Persona |
| { |
| char sexo; |
| String nombre; |
| int edad; |
| Coche coche;   // El coche que conduce esa persona |
|  |
| void saludar() |
| { |
| System.out.println("Hola, me llamo " + nombre); |
| } |
|  |
| void dormir() |
| { |
| System.out.println("Zzzzzzzzzzz"); |
| } |
|  |
| int obtenerEdad() |
| { |
| return edad; |
| } |
| } |

Fíjate que uno de los atributos es precisamente de la clase Coche que acabamos de definir:

|  |
| --- |
| Coche coche |
|  |

**El concepto de objeto**

Una vez hemos entendido el concepto de **Clase** en el paradigma de la Programación Orientada a Objetos, es momento de matizar qué es un **Objeto** en su sentido más práctico.

Una **Clase**, como hemos visto, no es más que una especificación que define las características y el comportamiento de un determinado tipo de objetos. Piensa en ella como si se tratara de una plantilla, molde o esquema a partir del cual podremos construir **objetos concretos**.

Consideremos la clase Coche que definimos en el artículo anterior:

Con esta plantilla, vamos a crear coches **concretos** de la marca, modelo y color que nos apetezca. Cada uno que creemos será un **objeto**, o **instancia**, de la clase Coche.

Fíjate, en esta terminología, en la equivalencia de los términos objeto e instancia para referirnos, ahora sí, a **entidades concretas (objetos)** de una determinada clase.

Fabriquemos, entonces, nuestro primer coche…

Cada objeto, como todas las variables en Java, ha de ser **declarado** antes de ser utilizado:

|  |
| --- |
| Coche coche1 |
|  |

Con esta instrucción declaramos la variable coche1 de tipo Coche. En cuanto creemos, con el comando que escribiremos a continuación, el objeto concreto, coche1 contendrá una **referencia** a ese objeto, es decir, almacenará la dirección de memoria en la que realmente se halla el objeto propiamente dicho. **Esto es muy importante**: coche1 no contendrá el objeto en sí, sino una dirección de memoria que apunta a él.

Materialicemos nuestro primer coche del siguiente modo:

|  |
| --- |
| coche1 = new Coche() |
|  |

La palabra reservada **new** se emplea para **crear nuevos objetos**, instancias de una determinada clase que indicamos a continuación, seguida de un par de paréntesis.

Veremos esto a su debido momento, pero por ahora tenemos bastante con retener que se está invocando a un método especial que tienen todas las clases, que sirve para **construir** el objeto en cuestión facilitándole sus valores iniciales. A este método se le conoce como **constructor** de la clase.

Podríamos haber realizado la declaración y la creación en una sola instrucción:

|  |
| --- |
| Coche coche1 = new Coche() |

Ahora, vamos a decir de qué marca y modelo es:

|  |
| --- |
| coche1.marca = "Seat" |
| coche1.modelo = "Panda" |

Consulta el cuadro de arriba con la definición de la clase y observa que tanto marca como modelo son dos atributos de tipo String, por lo que referencian cadenas de caracteres que escribimos entre comillas.

Observa con cuidado la **notación punto**. Separamos el nombre de la variable que referencia al objeto del atributo empleando un **punto como separador**.

Pintemos de azul nuestro vehículo:

|  |
| --- |
| coche1.color = "Azul" |

Además tiene tres puertas, el cuenta kilómetros indica 250.000 y su velocímetro refleja 0 Km/h. Valores correspondientes a los atributos numeroDePuertas, cuentaKilometros y velocidad, respectivamente, todos de tipo entero.

|  |
| --- |
| coche1.numeroDePuertas = 3 |
|  |
| coche1.cuentaKilometros = 250000 |
|  |
| coche1.velocidad = 0 |
|  |

Su motor está detenido, hecho que representamos a través de la variable booleana arrancado:

|  |
| --- |
| coche1.arrancado = false |

Ahora vamos a experimentar con los métodos. Arranquemos el Panda:

|  |
| --- |
| coche1.arrancar() |

De nuevo, empleamos también la notación punto para separar la variable del método.

Si observas el código verás que este método se limita a hacer que la variable booleana arrancado valga ahora true. Podrías decir que hubiéramos logrado el mismo resultado actuando sobre el atributo directamente, en lugar de invocar al método:

|  |
| --- |
| coche1.arrancado = true |
|  |

En efecto, es así; pero, como comprenderás más adelante, no suele ser buena idea dejar que los programas campen a sus anchas y modifiquen arbitrariamente los atributos de un objeto, siendo preferible que sean los métodos los que se ocupen de esa labor. Imagina que el programa intenta hacer que el cuenta kilómetros marque una cantidad negativa, o que el número de puertas sea igual a cincuenta. Un método correctamente diseñado podría gestionar que los valores estuvieran dentro del rango adecuado y asegurarse de que se cumplen las condiciones que permitirían la modificación del atributo.

La Programación Orientada a Objetos implementa un mecanismo, denominado **encapsulación**, que nos permite **ocultar** determinadas facetas de nuestro objeto y dejar solo accesibles aquellas partes que nos interesen. Pero vayamos por orden, todo a su momento…

Vamos a acelerar nuestro coche:

|  |
| --- |
| coche1.acelerar() |
|  |

Lo que provocará, si consultas el código, que el velocímetro incremente en una unidad su valor.

|  |
| --- |
| coche1.pitar() |

Lo que ocasionará un pitido.

Puedes crear todos los objetos de la clase Coche que desees:

|  |
| --- |
| Coche coche2 = new Coche() |

Cada uno con su propia colección de atributos:

|  |
| --- |
| coche2.marca = "Ford" |
| coche2.modelo = "Fiesta" |
| coche2.color = "Negro" |

Incluso podrías crear otros Seat Pandas, por supuesto. Aunque, en un instante dado, compartieran los mismos valores en sus atributos, se trataría de objetos distintos, **ubicados en direcciones de memoria diferentes** y cada uno podría seguir su propia trayectoria vital.

Extraído de:

<http://elclubdelautodidacta.es/wp/indice-java/>