FUNDACIÓN ACCENTURE accenture

JAVA I

2. LAS CLASES EN JAVA

Aprende a crear una clase en Java junto a sus diferentes elementos (atributos y métodos) y conoce cómo funciona el método main y la api de Java.

TABLA DE CONTENIDOS

JAVA I

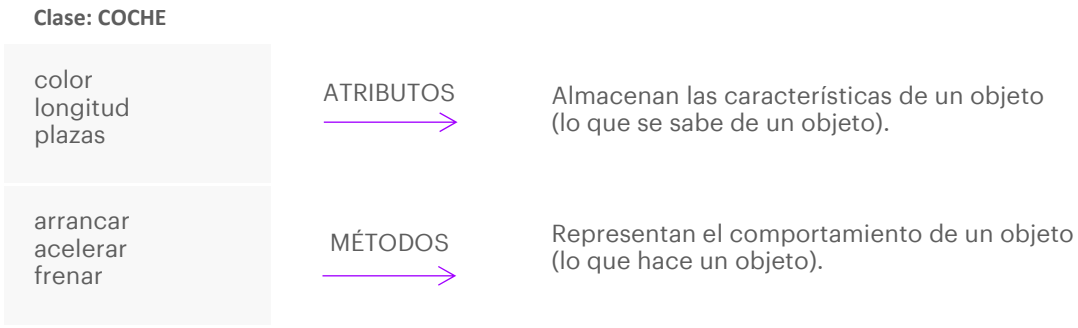
2. LAS CLASES EN JAVA

2.1. ¿QUÉ ES UNA CLASE?

¿QUÉ ES UNA CLASE JAVA?

Una Clase Java es una plantilla que **define la forma que tendrán** los objetos creados en la misma. Esto incluye el tipo de características (atributos) y los comportamientos (métodos) comunes que tendrán **los objetos que se crearán a partir de ella.**

Vamos a verlo con un ejemplo. Un coche (clase) tiene una serie de características (atributos) y comportamientos (métodos) comunes a todos los coches, pero también tiene características y comportamientos que diferencian un modelo de coche (objeto) de otro. Lo que hace diferente a cada modelo de coche (objeto), que creemos a partir de esta clase, serán sus características particulares (ATRIBUTOS: tamaño, color, número de plazas…) y comportamientos (MÉTODOS: arrancar, acelerar, frenar…).



APLICACIÓN EN JAVA

La **declaración** de una clase se realiza mediante la palabra reservada *class* seguida de su nombre.

La palabra clave class puede ir precedida de un modificador de acceso:

public → se puede acceder a la clase desde cualquier lugar del programa.
protected → se puede acceder a la clase desde el paquete o subclase de esta.
private → sólo se puede acceder a la clase desde esta misma.

Veamos su sintaxis: `[modificadores] class NombreClase {`

La **implementación** incluye los atributos y métodos de la clase, después de su declaración.

Ejemplo CLASE:

En nuestro ejemplo hemos definido la clase como public. A continuación, definimos los atributos y métodos que explicaremos en los siguientes apartados.

```
public class Coche {

    //declaración de atributos
    private String color;
    private double longitud;
    private int plazas;

    //declaración de métodos
    public void arrancar (){};
    public void acelerar (){};
    public void frenar (){};
```

2.2. ATRIBUTOS DE LA CLASE

ATRIBUTOS DE LA CLASE

Los atributos de la clase definen las características de esta, en nuestro ejemplo vimos que todos los coches tienen un color, una longitud y un número de plazas. Pues bien, estos atributos podrán ser datos de distinto **tipo**:

- **Primitivo:** int (números enteros), float (decimales de hasta 7 dígitos), double (decimales más precisos de hasta 16 dígitos).
- **Complejo:** String (texto o cadenas de texto), Date(fecha). También podemos crear nuestros propios tipos complejos.

Se definen dentro de la clase, pero en un apartado diferente a los métodos, y se inicializan de forma automática para cada objeto creado a partir de esa clase.

El **nombre** de los atributos por convención se escribe **empezando por minúscula**, y delante de cada atributo es necesario indicar el **tipo de dato seguido del nombre del atributo.**

De forma adicional, pueden ir precedidos de un **modificador de acceso**:

- **public**: Cuando se declara un atributo público se está permitiendo que desde cualquier otra clase pueda referenciarse directamente.
- **private**: Si se declara un atributo privado no se podrá acceder directamente a él desde otra clase distinta. Por tanto, será necesario recurrir a algún método que nos proporcione o nos modifique su valor.
- **protected**: Los atributos declarados así son accesibles directamente sólo desde la propia clase, de las subclases de esta (herencia) y las clases que se encuentran dentro del mismo paquete (se explicará posteriormente).

Datos complejos: datos complejos, que pueden estar formados por varias variables, otros datos complejos y/o comportamientos (cadenas de textos, fechas...)

Datos primitivos: datos simples que no necesitan ser creados, como números, caracteres, verdadero/falso.

Modificador de acceso: los modificadores de acceso en Java ayudan a restringir el alcance de una clase, constructor, variable, método o miembro de datos.

Ejemplo ATRIBUTOS de la clase:

Siguiendo con el ejemplo anterior, para nuestra clase Coche, los atributos serán **String color** (ya que es de tipo texto), **double longitud** (tipo número decimal preciso) e **int plazas** (número entero).

En nuestro ejemplo hemos definido todos los atributos como **private**. Tal y cómo hemos dicho, estos están dentro de la clase, pero en un apartado diferente a los métodos.

```
public class Coche {

    //declaración de atributos
   private String color;
   private double longitud;
   private int plazas;

    //declaración de métodos
    public void arrancar (){};
    public void acelerar (){};
    public void frenar (){};
```

2.3. MÉTODOS DE LA CLASE

MÉTODOS DE LA CLASE

Un **método** es un grupo de declaraciones o sentencias que realizan una tarea específica. Se puede llamar a un método tantas veces como lo necesitemos. Los métodos de la clase definen el comportamiento de los objetos de la clase. Siguiendo nuestro ejemplo, para la clase coche los métodos serán: arrancar, acelerar, frenar… es decir, todo lo que podemos hacer con un coche.

La declaración o implementación de un método incluye, la **cabecera** de dicho método y el **cuerpo** (grupo de sentencias) definido entre llaves "{ }".

Los componentes de la cabecera del método son:

- **Modificadores** (opcional).
- Tipo de **retorno** (usaremos ***void*** en caso de no retornar nada).
- **Nombre:** Se debe dar un nombre al método para poder ser invocado o usado cuando lo necesitemos; dicho nombre se escribe en minúsculas, pero cuando tienen más de una palabra, la inicial de la segunda palabra se escribe en mayúscula.
  Ejemplo: `arrancarSinRuido`, `acelerarRapido`.
- Lista de **parámetros**: Nos sirven para pasar información específica cuando sea necesario para la ejecución del método. La lista de parámetros se separa por comas y se indica, por cada parámetro, el tipo y el nombre.

Ten en cuenta que…
Podemos crear métodos dentro de cualquier clase, sin necesidad de ser una clase "plantilla" para crear instancias, de tal forma que podemos tener una clase donde sólo tengamos la implementación de unos métodos que podamos usar desde otras clases.

De forma adicional, pueden ir precedidos de uno o varios modificadores:

MODIFICADORES DE ACCESO

Especifican las **restricciones de acceso** o el **alcance** de una clase, método o variable:

→ **public**: Cuando se declara un método como público se está permitiendo que desde cualquier otra clase pueda referenciarse directamente.

→ **private**: Si se declara un método como privado no podrá ser invocado desde ninguna otra clase.

→ **protected**: Los métodos así declarados son accesibles directamente sólo desde la propia clase, de las subclases de esta (herencia) y las clases que se encuentran dentro del mismo paquete.

MODIFICADORES DE NO ACCESO

No cambian el alcance de la clase, método o variable, sino que les proporciona **funciones adicionales**:

- **static**: Se usa para definir un método que está vinculado a una clase, en lugar de a su instancia, es decir, sirve para crear métodos que pertenecen a la clase y no a la instancia de la clase. ¿Esto qué implica?, que **no es necesario crear una instancia u objeto de la clase para poder acceder o utilizar ese método**, simplemente se pueden invocar por su nombre siempre que se use en la misma clase donde se defina o, si se usa desde otra clase, anteponiendo el nombre de la clase donde se ha creado: nombreClase.nombreMetodo

- **final**: Un método declarado como final no puede ser sobrescrito por las subclases de la clase a la que pertenece (se explicará en el tema de herencia). Solo los métodos static pueden ser final.

- **abstract**: Define un método abstracto, es decir, es un método que **NO tiene una implementación**, sino que la implementación vendrá dada por las subclases de la clase a la que pertenece (se explicará en el tema de herencia). Es un método incompleto, no tiene cuerpo o implementación, por eso no tiene laves "{}" y termina con punto y coma: public abstract void saludar();

Ejemplo MÉTODOS de la clase:

En nuestro ejemplo hemos definido todos los métodos como **public** (hemos puesto **void** porque ninguno de ellos tiene que devolver un valor de retorno).

```
public class Coche {

    //declaración de atributos
    private String color;
    private double longitud;
    private int plazas;

    //declaración de métodos
    public void arrancar (){};
    public void acelerar (){};
    public void frenar (){};
```

RETORNO DEL MÉTODO:

Cuando se crea un método, otra de las características que le define es si queremos que nos devuelva algún valor o no.

En caso de no querer que se retorne ningún valor, en la declaración del método se debe poner la palabra **void** delante del nombre del método, como acabamos de ver en el ejemplo anterior.

En el caso de necesitar que el método devuelva un valor, se tiene que indicar el tipo de dato que esperamos que devuelva el método, además, debemos usar dentro del cuerpo del método la palabra reservada **return** para hacer que devuelva el valor. Por ejemplo, si quisiéramos crear un método que nos devuelva el color de nuestro coche, en este caso tenemos que indicar delante del nombre del método (getColor) el tipo de dato que será el valor de retorno (primitivo o complejo), en nuestro caso nos devolverá un color, por tanto, definiremos el método precedido de la palabra String.

Ten en cuenta que...

Dentro de un método con retorno de datos, **las líneas de código que existan después del return no se ejecutarán** ya que, al ejecutarse esta sentencia, se sale del método.

Ejemplo:

```
public class Coche {
      //declaración de atributos
      private String color;
      private double longitud;
      private int plazas;

      //declaración de método que devuelve el color del coche
      public String getColor () {
            return color;
      }
}
```

CONSTRUCTORES DE LA CLASE

Los constructores son un tipo de métodos especiales que inicializan los atributos de la clase cada vez que se crea un objeto con valores coherentes. Imaginemos que creamos un tipo de coche que es nuevo y uno de sus atributos es tener una longitud de 0 metros (no tiene sentido). El constructor sirve para **establecer el estado inicial del objeto** (mínimo 2 metros y máximo 4 metros). También podemos crear un constructor para definir que los coches que creemos sean rojos.

- **Uso**: para crear o establecer el estado inicial de un objeto.
- **Nombre**: deben tener el mismo nombre que la clase.
- **Estructura**: tienen la estructura de un método.
- **Valor**: no devuelven valor.

Hay cuatro **tipos de constructores**:

- Constructor **vacío/por defecto**: no tiene parámetros de entrada ni instrucciones a ejecutar. Sin embargo, sirve para inicializar los atributos de la clase con valores predeterminados por el sistema (por ejemplo, longitud inicial del coche se inicializaría a cero ya que es de tipo numérico). En nuestro ejemplo, dejamos que el sistema estipule valores predeterminados para los atributos:

```
public Coche() {
 }
```

- Constructor **de copia**: sólo tiene un parámetro de entrada, el cual será otro objeto de la misma clase. Sirve para clonar objetos, de tal forma que debemos asignar a cada uno de nuestros atributos el valor de los atributos del objeto a copiar. En el ejemplo asignamos a nuestro propio atributo (color) el valor del atributo (color) del objeto que estamos copiando:

```
public Coche (Coche nombreObjeto) {
      this.color = nombreObjeto.color;
}
```

→ Constructor **común**: es aquel que no recibe parámetros y se utiliza para asignar valores iniciales a los atributos de la clase. En nuestro ejemplo, le asignamos un valor inicial de 2 metros al atributo longitud:

```
public class Coche{
     double longitud;

     public Coche(){
          longitud = 2;
     }
}
```

→ Constructor **de parámetros**: recibe como parámetros los distintos valores para cada uno de los atributos del objeto a crear. En el ejemplo, asignamos una longitud concreta al atributo del objeto:

```
public class Coche{
     double longitud;

     public Coche(){
          longitud = 2;
     }

     public Coche (double long){
          this.longitud = long;
     }
}
```

Ten en cuenta que...

- Los constructores son llamados automáticamente cuando se crea un objeto.
- En todas las clases Java, siempre hay un constructor. Si no está definido por el usuario, existe el constructor por defecto o implícito, que será el constructor vacío.

PALABRA RESERVADA THIS

Como hemos visto en los ejemplos de los constructores, la palabra reservada *this* se usa para hacer referencia a las **propiedades** (atributos) del objeto, pero también podemos usarlo para acceder a otros **métodos o constructores** del objeto actual.

- **this.atributo**: accede a un atributo de la misma clase.
- **this.método():** accede a un método de la misma clase.
- **this():** llamamos a otro constructor de la misma clase.

En realidad, no es obligatorio siempre el uso de **this**, sólo será necesario utilizarlo en el momento en que estamos asignando a un atributo un parámetro en el que coincida con el nombre, porque así el compilador puede distinguir si estamos haciendo referencia al atributo (color) o al parámetro (color), porque si el atributo y el parámetro tienen distinto identificador no te haría falta, como puede verse en el ejemplo.

Ejemplo:

```
public class Coche{
    //declaración de atributos
    private String color;
    private double longitud;
    private int plazas;

    //Constructor de parámetros
    public Coche(String color, double long, int plazas){
          this.color = color;
          longitud = long; //Aquí no es necesario usar this
          this.plazas = plazas;
    }
}
```

SETTERS Y GETTERS

Cuando hemos declarado nuestros atributos como **private**, como en el ejemplo de la clase Coche, necesitamos los **métodos públicos** conocidos como **getters** y **setters** que nos permiten manipular o modificar los valores de nuestros atributos o recuperar los valores de dichos atributos.

En caso de que los atributos fueran públicos, no haría falta usarlos pues se tendría acceso a ellos sin ningún tipo de control.

SET

Permiten modificar los atributos privados.
Ejemplo: Asignarle un color específico al atributo (color) del coche.

GET

Permiten a los objetos retomar los valores de sus atributos privados.
Ejemplo: Devolver el color del coche

Ten en cuenta que...

En caso de que el atributo sea de tipo boolean (verdadero/falso), se usará la palara "**is**" en lugar de "get"

El **nombre** de estos métodos empezará por las palabras **get** o **set**, según corresponda y seguido del nombre del atributo en mayúsculas.

En nuestro ejemplo de la clase Coche, al ser todos los atributos privados, necesitamos los métodos setters y getters para cada uno de ellos, de tal forma, que necesitamos el método **get** que nos permita devolver el valor de un atributo usando el return, cuyo nombre sería *getColor*. Igualmente, necesitamos el método **set** que nos permita asignar un valor a al atributo, cuyo nombre del método sería *setColor*.

Ejemplo:

```
public class Coche {
    //declaración de atributos
    private String color;
    private double longitud;
    private int plazas;

    //declaración de método get que devuelve el color del coche
    public String getColor(){
           return color;
    }

    //declaración de método set que modifica el color del coche
    public void setColor(String rojo){
           this.color = rojo;
    }
}
```

2.4. MÉTODO MAIN

Ya hemos visto que los métodos son un conjunto de **instrucciones o comportamientos** definidos dentro de una clase, **que realizan una determinada tarea** (arrancar, frenar, imprimir, devolver un valor) para los objetos que crearemos a partir de esta.

Aunque, por otro lado, una clase también puede estar compuesta por métodos estáticos que no necesitan de objetos, como es el caso del **método Main.**

MÉTODO MAIN

El método main sirve para iniciar la ejecución de cualquier programa en Java. Dicho método se conoce como **punto de entrada de una clase.**

PRINCIPIOS DEL MÉTODO MAIN:

- Siempre debe incluir los modificadores: ***public*** (el método será visible para todas las clases en todas partes) y ***static*** (el método es estático, por tanto, se puede llamar o usar desde cualquier clase sin necesidad de crear una instancia de la clase en la que se ha creado dicho método).
- No puede retornar un valor como resultado, debe indicar el valor ***void*** como retorno.
- Su parámetro de entrada siempre será un array de tipo ***String*** que debe aparecer obligatoriamente como argumento de este método.

Ejemplo: Imagina que queremos crear un programa que únicamente imprima un texto de saludo "Hola a todos!!!". Para esta clase no tenemos objetos, sólo crearemos la clase, en nuestro caso le hemos llamado Robot:

```
public class Robot {
      public static void main(String[] args){
            System.out.println("Hola a todos!!!");
      }
}
```

¿Sabías que en Java se puede escribir el método Main de tres maneras diferentes sin que esto produzca un error en la ejecución del programa?:

- public static void main(String args[]) {}
- public static void main(String[] args){}
- public static void main(String... args){}

2.5. API DE JAVA: IMPORT

¿QUÉ ES LA API DE JAVA?

La **API de Java** es una **Interfaz de Programación de Aplicaciones** (API, por sus siglas en inglés: *Application Programming Interface*) provista por los creadores del lenguaje de programación Java, que da los medios para desarrollar aplicaciones Java, es decir, proporciona un conjunto de clases e interfaces predefinidas, que se copian durante la instalación de Java, para realizar nuestras aplicaciones, y podemos utilizarlas cuando sea necesario.

IMPORT

La API Java está organizada en **paquetes** que contienen un conjunto de clases relacionadas semánticamente.

Si desde nuestro programa se necesita usar algún recurso o recursos de la API de Java, los importaremos a nuestra clase para poder utilizarlos.

→ ¿Cómo importamos paquetes?

```
import nombre.paquete.*; //Con esto importaríamos todas las clases de este paquete.
import nombre.paquete.NombreClase; //solo importamos la clase nombrada
```

Ejemplo: A continuación, vemos un ejemplo donde se indica en primer lugar el package (que ya vimos en el primer tema de Introducción a Java) y es el lugar donde guardaremos la clase que estamos creando. Además, como queremos definir uno de nuestros atributos de tipo fecha, podemos utilizar el tipo Date, que es una clase propia de la API de Java, con lo que debemos incluir un **import** de dicha clase para usarla.

```
package es.ejemplos;

import java.until.Date;

public class Persona {
      private String nombre;
      public Date fechaNacimiento;

      public void saludo() {
            System.out.printIn("Hola " + nombre);
      }
}
```

2.6. PREGUNTAS FRECUENTES

- **¿Las declaraciones en Java admiten tildes?**

  No, debemos escribir las palabras que tengan tilde sin esta. Tampoco es recomendable el uso de la “ñ”.

- **¿Qué quiere decir que el método no tiene retorno?**

  Quiere decir que tenemos un método que no nos devuelve ningún valor, es decir, se limita a ejecutar una serie de instrucciones. Este se declara con la palabra “void”.

FUNDACIÓN ACCENTURE

accenture