



UF4: Programación orientada a objetos. Fundamentos.

DESARROLLO DE APLICACIONES

MARZO DE 2023



Contenido



1. Introducción.
2. ¿Qué es una clase?.
3. Estructura de una clase: Atributos.
4. Estructura de una clase: Métodos.
5. Estructura de una clase: Constructores.
6. Visibilidad de atributos y métodos.
7. Encapsulamiento y el apuntador "this".
8. ¿Qué es un objeto?. Características. Definición.
9. Atributos de clase y de objetos.
10. Métodos de clase y de objetos.
11. Sobrecarga de métodos.
12. Tipos primitivos vs. Objetos.
13. Características POO: Composición, herencia y abstracción.
14. Herencia y constructores.
15. Sobreescritura de métodos.
16. Clases y métodos abstractos.
17. Interfaces.
18. Destructores, finalización de objetos y liberación de memoria.
19. Otras características de la POO.
20. Uso avanzado de clases: Polimorfismo

Introducción a la POO.



La programación orientada a objetos (en adelante, POO) es un paradigma de programación que permite realizar una abstracción de la realidad.

¿Qué significa esto? Pues que tratemos la solución a implementar como si lo que codificamos fueran objetos que tienen unas características determinadas y un comportamiento determinado.

Por ejemplo: Si queremos tratar información relativa a las personas (una gestión de personas para acceder a una web, por ejemplo), podemos pensar en simular nuestro problema con un objeto que sea un objeto *Persona*. Una persona tiene una serie de características (nombre, apellidos, dni, etc) y una serie de acciones relacionadas o no con estas características.

Introducción a la POO.

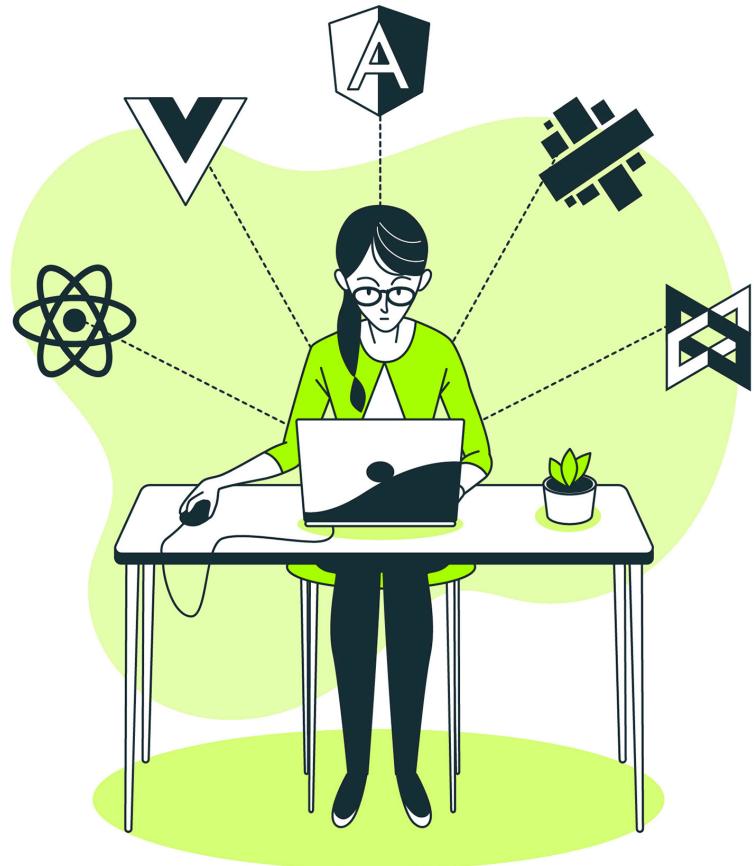


Imagen de storyset en Freepik

Una persona tiene características:

- DNI
- Nombre
- Apellidos
- Color pelo
- Color ojos
- Etc.

Las características es lo que llamamos **atributos o propiedades** en una clase

Introducción a la POO.

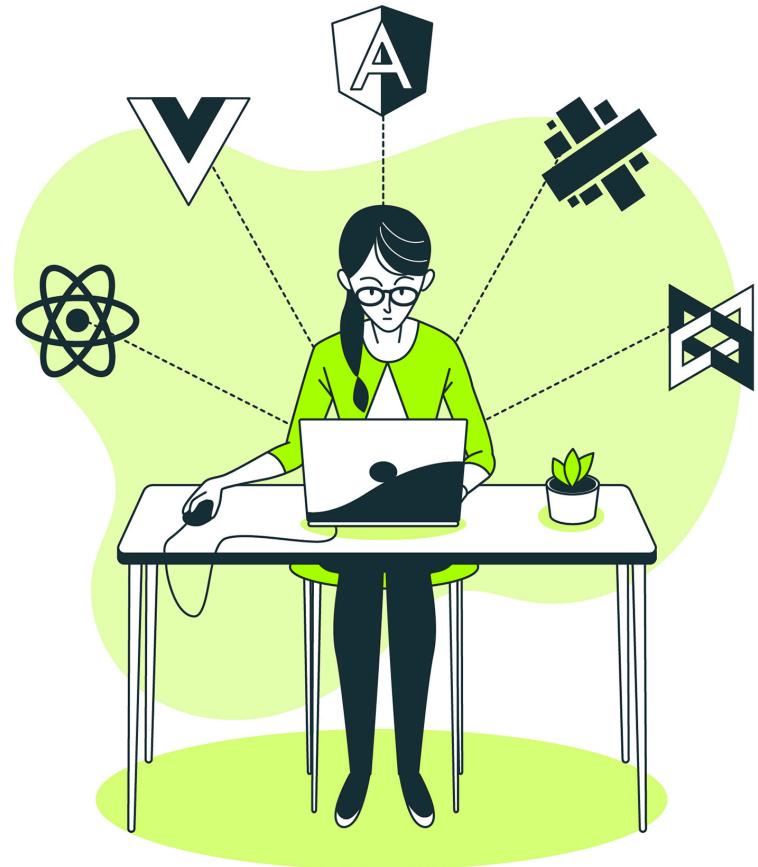


Imagen de storyset en Freepik

Y también tiene acciones o habilidades:

- Mirar
- Recoger
- Caminar
- Etc.

Las acciones o habilidades es lo que llamamos **métodos** en una clase

Los atributos y los métodos lo denominamos miembros de la clase

¿Qué es una clase?



Podemos entender una clase como **la definición de un tipo abstracto de dato que contiene atributos y métodos**.

A través de una clase podemos representar un objeto abstraído de la realidad.

Una clase es solo la **DEFINICION** de un objeto con los que el programa que estamos desarrollando tiene que tratar.

Estructura de una clase: Atributos.

Los atributos de una clase se especifican mediante declaraciones de los datos.

Por ejemplo: La clase **Persona** podría tener las siguientes declaraciones para reflejar su nombre, apellidos y edad:

```
String nombre;  
String apellidos;  
int edad;
```

Cuando hacemos declaraciones, **podemos declarar también otros tipos de datos**. Podría tener una clase **EquipoFutbol** que entre sus atributos tenga un tipo de dato **Persona**.

```
Public class EquipoFutbol {  
    Persona jugador;  
    String nombreEquipo;  
}
```

Recordemos la convención para la definición de los datos:

Consta de letras, símbolos de subrayado y dígitos, empezar por una letra, no se permiten espacios, son Case sensitive, el uso de mayúsculas en las palabras interiores cuando son compuestas.

Estructura de una clase: Métodos.



Las acciones o habilidades de una clase se expresan con uno o más métodos. Un **método** es una secuencia de instrucciones a las que se da un nombre único.

La llamada a un método hace que se ejecute el conjunto de instrucciones dentro de ese método. Lo que hasta ahora hemos llamado una función o un procedimiento. Por tanto, un método puede tener uno o varios datos de entrada, y también puede tener un dato de salida.

```
public void mirar() {  
    System.out.println("Está mirando");  
}
```

Un método puede declarar sus propios campos (variables) que son **locales**, y por ello no son accesibles desde otros métodos aunque estén definidos en la misma clase. Pero **sí puede modificar los atributos de la clase**.

Estructura de una clase: Constructores.

El programador que escribe la clase puede definir uno o más constructores que tengan diferente número de parámetros.

Los constructores tienen el mismo nombre que la clase. Por lo tanto, podríamos tener los siguientes constructores para la clase *Persona*:

```
//Constructor por defecto: no tiene parámetros  
Persona () {  
    nombre = "";  
    apellidos = "";  
    edad = 0;  
}  
//Constructor con parámetros  
Persona (String nom, String ape, int ed) {  
    nombre = nom;  
    apellidos = ape;  
    edad = ed;  
}
```

Los objetos que no han pasado por el proceso de la construcción reciben un valor especial (el objeto nulo) de la JVM. Pudiendo comprobar si un objeto es nulo. Si tratamos de utilizar los miembros de un objeto nulo, Java lanza la excepción *NullPointerException*.

Visibilidad de atributos y métodos.

Con visibilidad nos referimos al nivel de accesibilidad de los atributos y métodos.

En Java los niveles de accesibilidad se dan por las siguientes palabras reservadas:

public *Se puede acceder desde un método implementado desde cualquier clase.*

private *Sólo se puede acceder desde un método implementado en la propia clase.*

protected *Se puede acceder desde un método implementado en una clase que "hija", esto es, que herede la clase que contiene esta visibilidad y desde clases que se encuentren en el mismo paquete.*

Encapsulamiento y el apuntador "this".

Encapsulamiento: es una característica que indica que los atributos que definen las propiedades de la clase deben tener visibilidad *private*. De esta forma se ofrece seguridad a la información que se encuentra en estos atributos.

La **norma** es definir los atributos como *private* precisamente para ofrecer esta seguridad en la integridad de la información contenida en estos atributos.

El **apuntador "this"** permite acceder a los atributos y métodos de la clase. No es obligatorio su uso siempre, pero se recomienda como buena práctica. Pero sí es obligatorio cuando desde un método queremos acceder a un atributo de la clase que tiene el mismo nombre que un parámetro de dicha función.

Es una forma de acceder directamente a cualquiera de sus miembros.

¿Qué es un objeto?. Características y definición.

Un **objeto** es una realización **concreta** de una descripción de clase.

Podemos entender una clase como un fichero en el que se define (y solo se define) los atributos y los métodos de un objeto. Pero no tiene "vida", esto es, no tiene recursos de memoria asignados y no podemos hacer nada con él, ni invocar sus métodos ni acceder a sus atributos.

Para pasar de esta definición a una realización concreta de esta descripción necesitamos "construir" este elemento. Es lo que denominamos **constructor**.

Para poder llamar a sus métodos es necesario que esté *instanciado* (esté creado). Por tanto ya no es una clase sino que pasa a tener unas características determinadas.

Por ejemplo:

Podemos tener definida una clase *Persona*, con sus atributos y métodos, pero no tenemos ninguna realización concreta hasta que no creamos uno o varios objetos.

```
Persona juan = new Persona ("Juan", "Sánchez", 55);  
Persona ana = new Persona ("Ana", "Lamas", 54);  
//Creación de dos objetos llamando a sus constructores.
```

En las instrucciones anteriores estamos creando dos objetos, uno denominado *juan* y otro denominado *ana* (lo que hasta ahora hemos llamado variables). Estos objetos son de tipo *Persona*. Y los hemos creado invocando un constructor utilizando la palabra reservada *new*.

Cada uno de estos objetos tiene todas los atributos y métodos definidos en la clase *Persona*. Y cada objeto tiene una copia de todos los campos definidos para esa clase.

Estrictamente, las variables son de un tipo y los **objetos** pertenecen a una clase.

Crear un objeto e invocar a un método de su clase

Crear un objeto.

```
Persona ana = new Persona ("Ana", "Lamas", 54);
```

Invocar un método de su clase.

La forma de indicar a quién le queremos aplicar las instrucciones de un método es anteponer el nombre del objeto al nombre del método.

Por ejemplo:

```
ana.mirar();
```

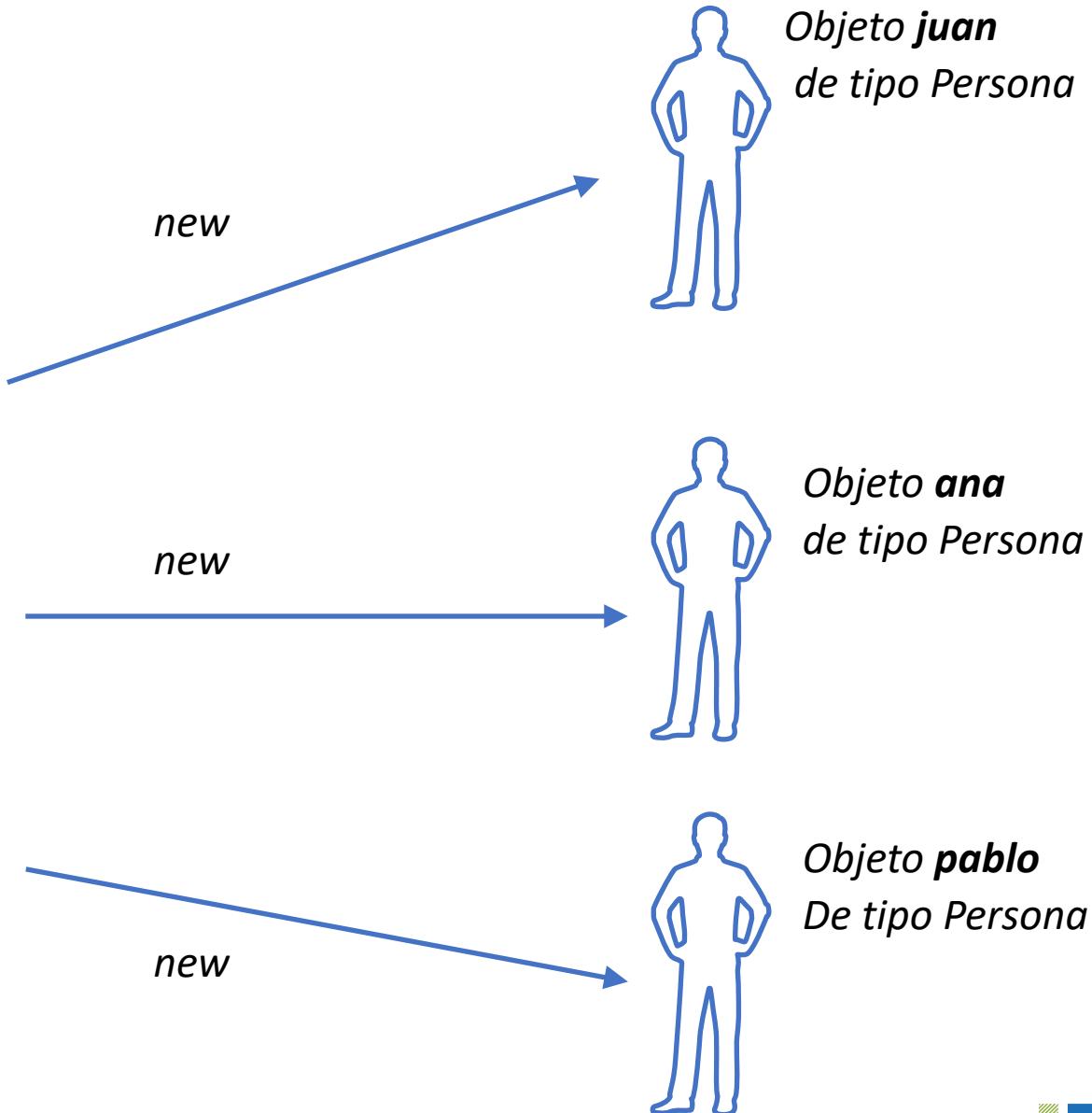
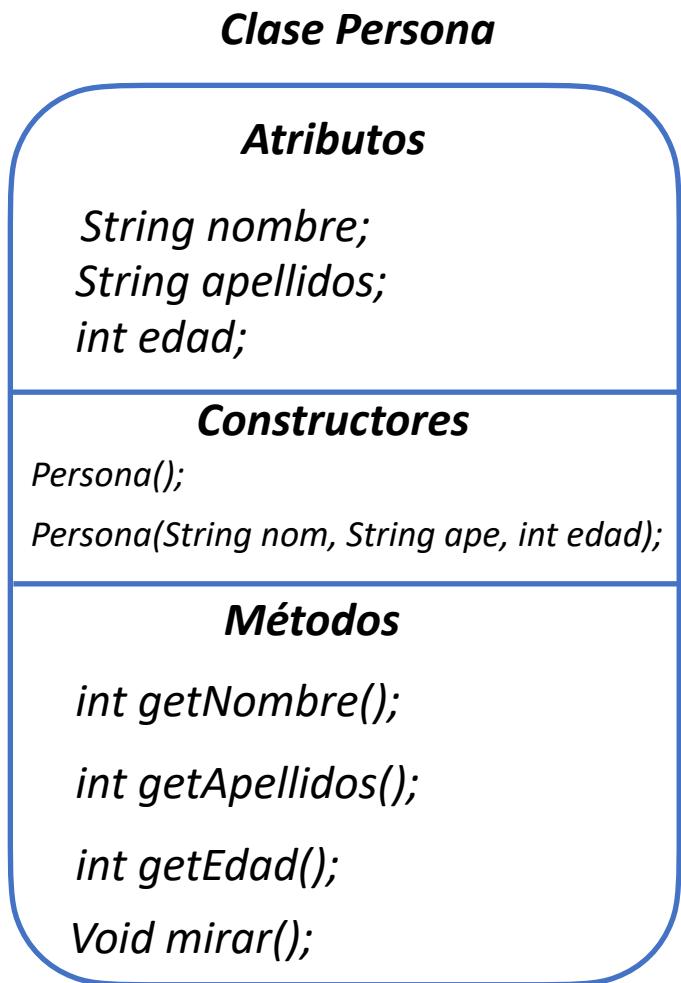
Un programa y sus clases

¿Qué relación tiene un programa principal con las clases?

Pues un programa realizado en un lenguaje de programación orientado a objetos, como Java, consta de una o más clases interdependientes. Pueden estar relacionadas entre sí o no.

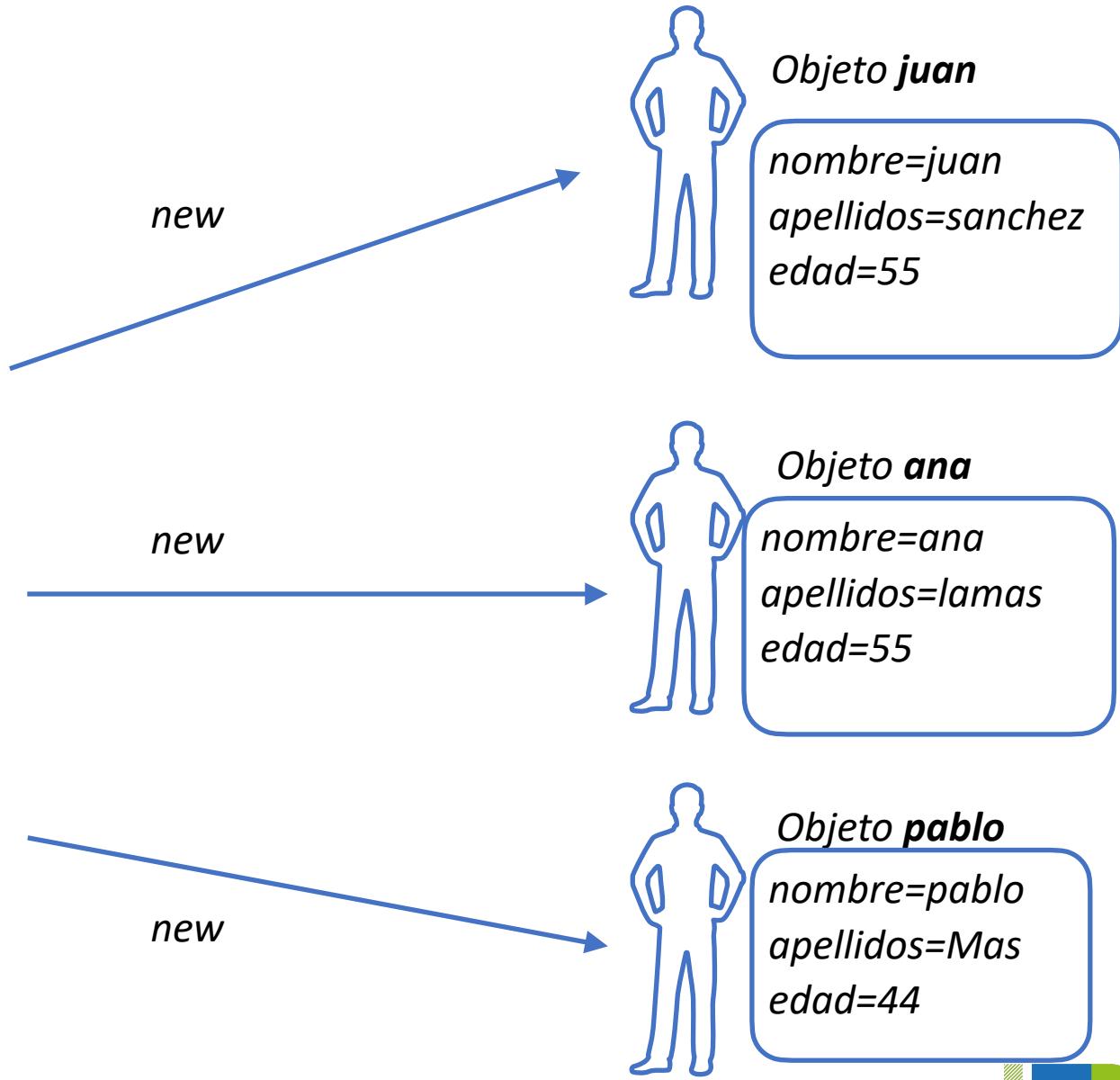
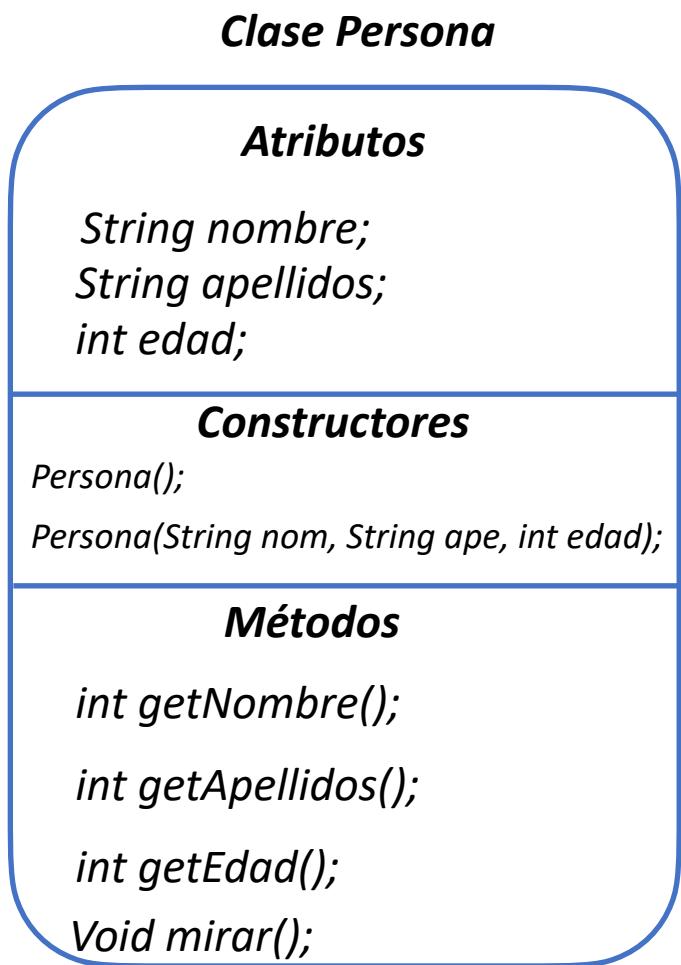
En mi programa principal tendré que definir una o varias clases para resolver el problema que se plantea.

Clases vs. Objetos



Al llamar al constructor con la palabra reservada **new** creamos un objeto

Clases vs. Objetos



Encapsulamiento: definimos los atributos como privados y accedemos a ellos a través de un método get

Atributos de clase y de objetos.

Decíamos anteriormente que al crear un objeto llamando a su constructor, se crea una nueva copia de todos los campos (atributos) declarados para esa clase. Con lo que cada valor almacenado en cada atributo es único de ese objeto. Es lo que denominamos **atributo de objeto**.

En algún caso conviene tener un atributo que no sea único a ese objeto, sino que sea común a todos los objetos de su clase. Esto es lo que denominamos **atributo de clase**.

Para crear un atributo de clase debemos utilizar la palabra reservada **static**. Indica que el almacenamiento para el campo se crea una única vez para la clase, y no cada vez que se crea un nuevo objeto.

¿Cómo nos referimos a los atributos de clase y de objetos? Aplicando la siguiente regla:

- Dentro de su clase, nos referiremos a los campos por sus nombres únicamente.
- Cuando se usan desde otra clase u objeto, anteponemos al nombre del campo el nombre de su clase (para campos de clase) o el de su objeto (para campos de objeto).

Métodos de clase y de objetos.

Al igual que para los atributos, podemos diferenciar entre métodos que no dependan de un objeto particular para funcionar.

El ejemplo que hemos visto hasta ahora es el método de punto de arranque de nuestro programa.

Siguiendo la convención de los campos de clase, un método de clase se define con el modificador *static* y se llama con el nombre de la clase como prefijo.

Sobrecarga de métodos y constructores.

La sobrecarga de métodos es la creación de varios métodos con el mismo nombre pero con diferente número de tipos de parámetros.

¿Cómo sabe la máquina virtual qué método ejecutar entonces? Java utiliza el número y tipo de parámetros para seleccionar cuál definición de método ejecutar.

Al ser un tipo especial de método, también se aplica lo anterior, esto es, podemos tener diferentes constructores siempre y cuando tengamos diferente número o tipo de parámetros.

Tipos de datos en Java: tipos primitivos vs. Objetos.

Tipos primitivos
(sin métodos, no
son objetos, no
necesitan una
llamada al
constructor)

byte
int
long
float
double
char
boolean

Tipos Objetos
(con métodos,
necesitan una
llamada al
constructor para
ser creados)

Tipos de la biblioteca estándar de Java: *String*, *Scanner*, *ArrayList*, etc.
Tipos definidos por el programador: Persona, Taxi, Tren, etc.

Arrays: Serie de elementos o formación tipo vector o matriz. Se considera un
objeto especial que carece de métodos.
Tipos envoltorio o *wrapper*: *Byte*, *Short*, *Integer*, *Long*, *Float*, *Double*, *Character*,
Boolean.
Se consideran equivalentes a los tipos primitivos pero como objetos.

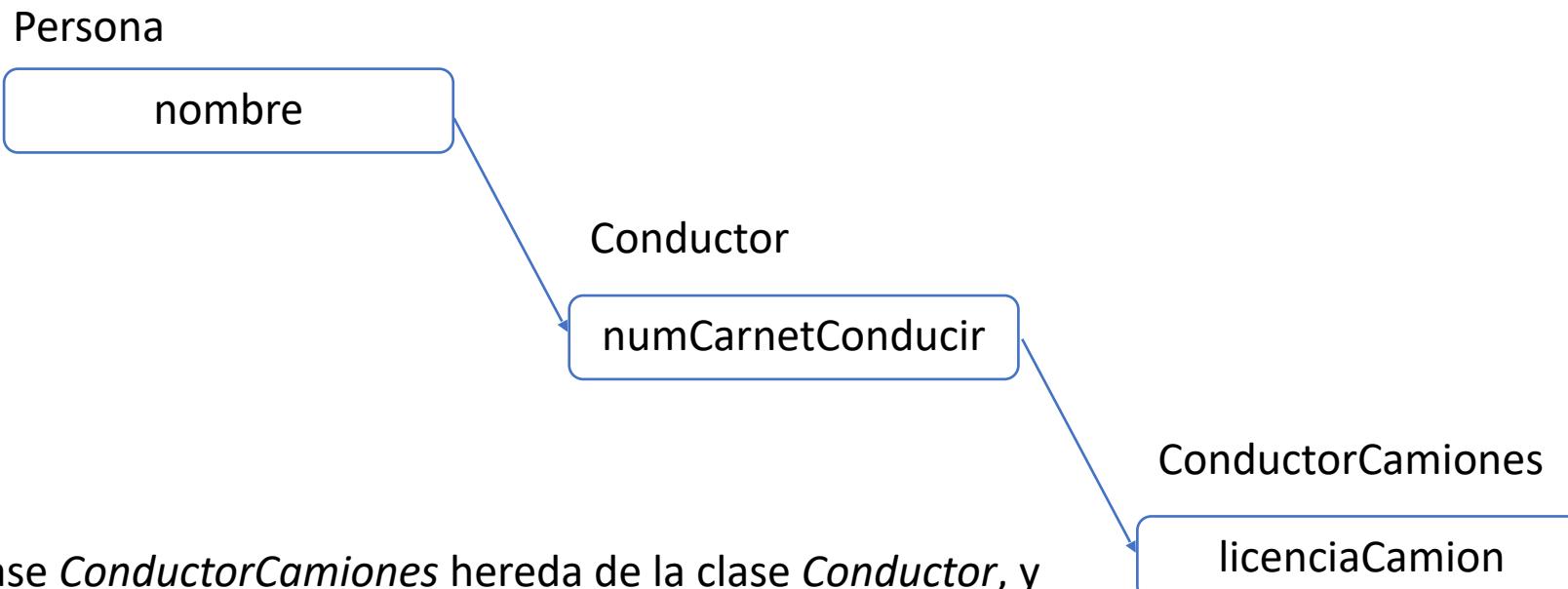
Características POO: Composición, herencia y abstracción.

Composición: Es cuando definimos un atributo de una clase de un tipo que se corresponde con otra clase. Esto es, crear un objeto de una clase basada en otra clase.

Herencia: Nos concentramos en definir una clase que conocemos bien y dejamos abierta la opción de definir más tarde nuevas versiones de ella. Estas nuevas versiones "heredarán" los atributos de la clase original, y por tanto pueden ser más pequeñas y más limpias.

Abstracción: Permite que una clase o método se centre en lo esencial de lo que está haciendo (su conducta e interfaz con el mundo) confiando en que los detalles se resolverán posteriormente.

Herencia. Ejemplo.



La pregunta que nos tenemos que hacer para saber si podemos aplicar herencia es: ¿Un conductor de Camiones es un conductor?, sí. Luego podemos aplicar herencia.

Y, ¿un conductor es una persona?, sí. Luego podemos aplicar también herencia entre la clase *Conductor* y la clase *Persona*.

Herencia. Ejemplo.



Persona

nombre

Conductor

numCarnetConducir

ConductorCamiones

licenciaCamion

En las clases hijas pondremos las características peculiares a esa clase hija, en este caso en la clase *Conductor* pondremos el atributo *numCarnetConducir*. Al igual para la clase *ConductorCamiones* donde tenemos una característica particular reflejada en el atributo *licenciaCamion*.

La clase "padre" la llamamos superclase y la "hija", subclase.

En Java utilizamos la palabra reservada "*extends*" para reflejar que una clase hereda (extiende) de otra.

La herencia tiene la propiedad transitiva: si *a* es una clase, *b* la extiende y *c* extiende a *b*,
23 entonces *c* también hereda de *a*.

Herencia y constructores

Una subclase puede llamar al constructor de su clase padre para realizar la inicialización de las variables que están bajo la responsabilidad de la clase padre. De esta manera, el trabajo queda dividido de forma más limpia.



Sobreescritura de métodos

Una subclase puede definir su propia versión de un método que ya está implementado en una de sus superclases.

Se trata de, a medida que descendemos en la jerarquía, ir añadiendo versiones más especializadas.

La sobreescritura de métodos utiliza la ligadura dinámica para seleccionar el método correcto. Cada objeto lleva asociada una tabla de sus métodos, y Java busca la versión correcta de los métodos sobreescritos en el momento de la ejecución.

La sobreescritura no es lo mismo que la sobrecarga de métodos. En el primero, el número y tipo de parámetros es idéntico.

Clases y métodos abstractos

Junto con las interfaces y la herencia proporciona una manera clara y comprensible de construir sistemas muy grandes.



Métodos abstractos.

Son "casillas" para métodos que se deben mencionar en cierto nivel, pero sólo se implementarán más abajo en la jerarquía de clases.

Clases abstractas.

Es aquella clase que contiene al menos un método abstracto.

Una clase abstracta no se puede utilizar para declarar objetos, la presencia del método abstracto significa que está incompleta.

Herencia de clases: superclases y subclases.

La superclase *Object*

Java alcanza la generalidad utilizando objetos de diferentes clases en las mismas partes del programa, pero todos ellos pertenecen a la clase *Object*.

Esto significa también que los métodos que están definidos en esta clase los podemos sobreescribir en las clases hijas.

Para **comparar** los valores de dos objetos (todos juntos) debemos utilizar el método *equals*, que es preciso definir para cada clase nueva de objetos. Esto supone que el programador debe decidir cómo quiere definir la igualdad. Por ejemplo, en el ejemplo que estamos realizando, diremos que dos personas son iguales si sus nombres, apellidos y edad coinciden.

Otra operación que tiene que ser tenida en cuenta es la **asignación** entre objetos. Al asignar dos objetos, por ejemplo: *ana = juan;*

Se copiarán las referencias. Para copiar los valores, es necesario clonar el objeto utilizando un método que también ha de ser definido para cada nueva clase. Este método es *clone()*. Por ejemplo, clonar un objeto de tipo *Persona* supone copiar todos los datos que contiene.

Por último, existen métodos definidos en la superclase *Object* de Java, que otros objetos pueden utilizar. Sin embargo, antes de que se asignen los resultados, es preciso realizar un cambio explícito de tipo (*cast*).

Interfaces

Una interfaz es un tipo especial de clase que define (y solo define) la especificación de un conjunto de métodos. Eso es todo.

Esto garantiza que cualquier clase que implemente la interfaz tiene que proporcionar esos métodos. Podemos pensar que una interfaz define un estándar, y que la clase que la implementa recibe el "sello de aprobación" de estar conforme a ese estándar. Por consiguiente, los objetos de esa clase tienen acceso a cualquier de los métodos que requiera un objeto adaptado a ese estándar.

Definición de una interfaz:

```
Interface nombreDeInterfaz{  
    Especificaciones de los métodos  
}
```

Implementación de una interfaz:

```
Class nombreDeClase implements nombreDeInterfaz{  
    Cuerpo de los métodos de la interfaz datos y métodos propios  
}
```

Destructores, finalización de objetos y liberación de memoria.

Un destructor es un método opuesto a un constructor, este método en lugar de crear un objeto lo destruye liberando la memoria de nuestra computadora para que pueda ser utilizada por alguna otra variable u objeto.

En java no existen los destructores, esto es gracias al recolector de basura (garbage collector) de la máquina virtual de java.

En cambio en otros lenguajes de programación (por ejemplo, C) es necesario implementarlo ya que en caso contrario se quedan espacios en la memoria ocupados que no están siendo utilizados por nuestro programa.

El recolector de basura recolecta todas las variables u objetos que no se estén utilizando y que no haya ninguna referencia a ellos por una clase en ejecución, liberando así automáticamente la memoria de nuestra computadora.

Aunque Java maneja de manera automática el recolector de basura, el usuario también puede decir en qué momento Java pase el recolector de basura con la instrucción.

`System.gc();`

Otras características de la POO



Bajo acoplamiento

Indica que los diferentes subsistemas deben estar unidos de forma mínima. Esto indica, que las clases que se construyan deben ser lo más reducidas. Ejemplo de sistema con bajo acoplamiento:
Una memoria USB.

El caso contrario sería un disco duro.

Alta cohesión

Indica que, los atributos y métodos de una clase deben ser conscientes con el concepto que abstrae dicha clase.

O sea, los atributos y métodos de dicha una clase deben estar referidos a las características y habilidades de dicha clase.

Polimorfismo: En POO, y siendo rigurosos, el polimorfismo se refiere a la capacidad que poseen las clases “hijas” de utilizar un mismo método heredado de forma distinta. Pueden inducir a confusión con la sobrecarga, pero en este caso hay herencia.

Otras características de la POO



Polimorfismo: En POO, y siendo rigurosos, el polimorfismo se refiere a la capacidad que poseen las clases “hijas” de utilizar un mismo método heredado de forma distinta.

Puede inducir a confusión con la sobrecarga, pero hay una diferencia: la sobrecarga de métodos no incluye la característica de la herencia, como sí hace el polimorfismo