



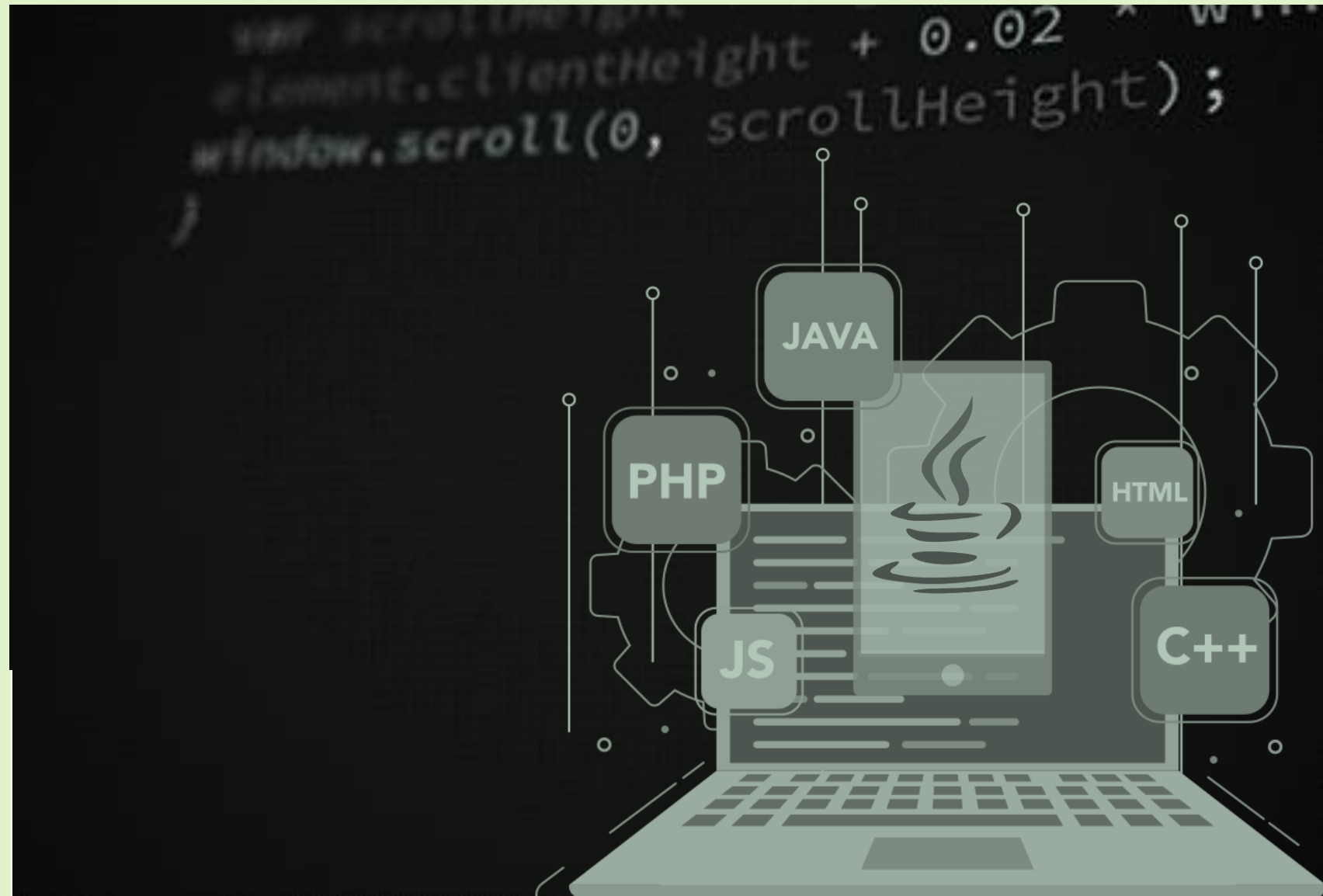
EFA
MORATALAZ

*1º CFGS Desarrollo de
Aplicaciones Web*

PROGRAMACIÓN

*DANIEL GONZÁLEZ-CALERO
JIMÉNEZ*

UT5 – INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS





EFA
MORATALAZ

*1º CFGS Desarrollo de
Aplicaciones Web*

PROGRAMACIÓN

INDICE

UT5 – INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS

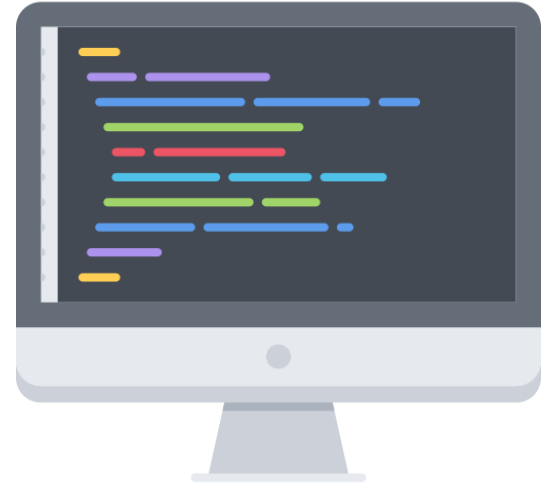
1. INTRODUCCIÓN
2. EL CONCEPTO DE CLASE
3. EL CONCEPTO DE OBJETO
4. CONSTRUCTOR
5. GETTERS/SETTERS
6. OBJETOS
7. ENCAPSULACIÓN
8. HERENCIA
9. POLIMORFISMO
10. ABSTRACCIÓN

INTRODUCCIÓN

1

La Programación Orientada a Objetos (POO) es un paradigma de programación que usa objetos para crear aplicaciones. Está basada en tres pilares fundamentales:

- Herencia
- Polimorfismo
- Encapsulación.



Su uso se popularizó en la década de los 90 y desde entonces han ido surgiendo una gran variedad de lenguajes de programación que soportan programación orientada a objetos.

Uno de los lenguajes más utilizados para la POO es Java.

Lo interesante de la POO es que proporciona conceptos y herramientas con las cuales se modela y se representa el mundo real, tan fielmente como sea posible.

Alguna de las ventajas de la POO son:

- Fomenta la reutilización y ampliación del código
- Permite crear sistemas más complejos
- La programación se asemeja al mundo real
- Agiliza el desarrollo de software
- Facilita el trabajo en equipo

Existen dos conceptos fundamentales en la POO:

- Las clases
- Los objetos

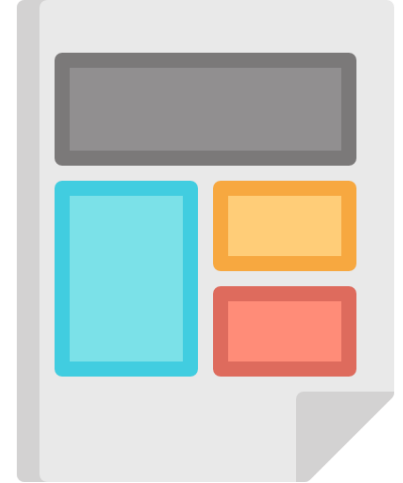


EL CONCEPTO DE CLASE

A large, stylized number 2 in a light green color, centered within a dark green rounded square. The number has a slight shadow effect, giving it a 3D appearance.

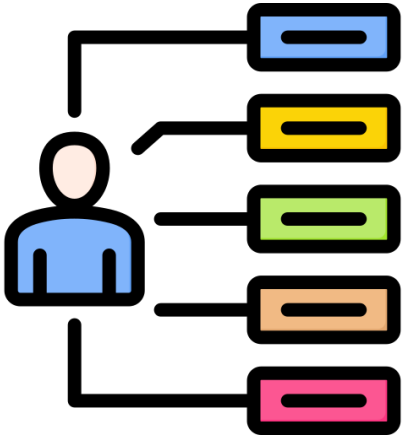
Definición teórica: La **clase** es un modelo o prototipo que define las **variables** y **métodos** comunes a todos los objetos de cierta clase.

También se puede decir que una **clase** es una plantilla genérica para un conjunto de **objetos** de similares características.



Por otro lado, una instancia de una **clase** es otra forma de llamar a un **objeto**. En realidad no existe diferencia entre objeto y una instancia. Sólo que el objeto es un término más general, pero los objetos y las instancias son ambas representaciones de una **clase**.

En el mundo real, normalmente tenemos muchos objetos del mismo tipo. Por ejemplo, nuestro teléfono móvil es sólo uno de los miles que hay en el mundo. Si hablamos en términos de la POO, podemos decir que nuestro **objeto** móvil es una **instancia** de la clase <<móvil>>. Los móviles tienen características (marca, modelo, sistema operativo, pantalla, teclado, etc.) y comportamientos (hacer y recibir llamadas, enviar mensajes multimedia, transmisión de datos, etc.)



Cuando se fabrican los móviles, los fabricantes aprovechan el hecho de que los móviles comparten esas características comunes y construyen modelos o plantillas comunes, para que a partir de esas se puedan crear muchos móviles del mismo modelo. A ese modelo o plantilla le llamamos **clase**, y a los equipos que sacamos a partir de ella la llamamos **objetos**.

Esto mismo se aplica a los objetos de software, se pueden tener muchos objetos del mismo tipo y mismas características.



EL CONCEPTO DE OBJETO



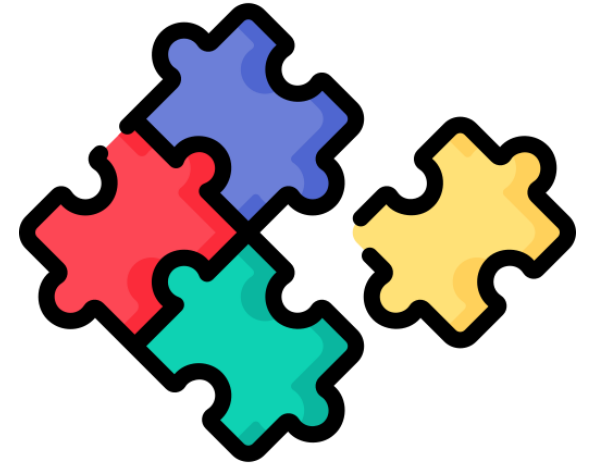
Entender qué es un **objeto** es la clave para entender cualquier **lenguaje de programación**. Tomemos como ejemplo un ordenador, el cual se compone de varios componentes: placa base, procesador, disco duro, tarjeta de video, etc.



El trabajo en conjunto de todos estos componentes hace funcionar un ordenador, sin embargo no necesitamos saber cómo trabajan cada uno de estos componentes. Cada componente es una unidad autónoma, y todo lo que necesitamos saber es **cómo interactúan entre sí los componentes**, es decir, saber si el procesador y las memorias son compatibles con la placa base...

¿Y qué tiene que ver todo esto con la programación?

Todo **programa** está constituido en base a diferentes **componentes** (objetos), cada uno tiene un rol específico en el *programa* y todos los *componentes* pueden comunicarse entre ellos de formas predefinidas.



Todo objeto tiene 2 componentes: **variables de clase** y **métodos**.

Por ejemplo, los automóviles pueden tener como variables de clase (marca, modelo, color, velocidad máxima, etc.) y como métodos (frenar, acelerar, retroceder, llenar combustible, cambiar llantas, etc.)

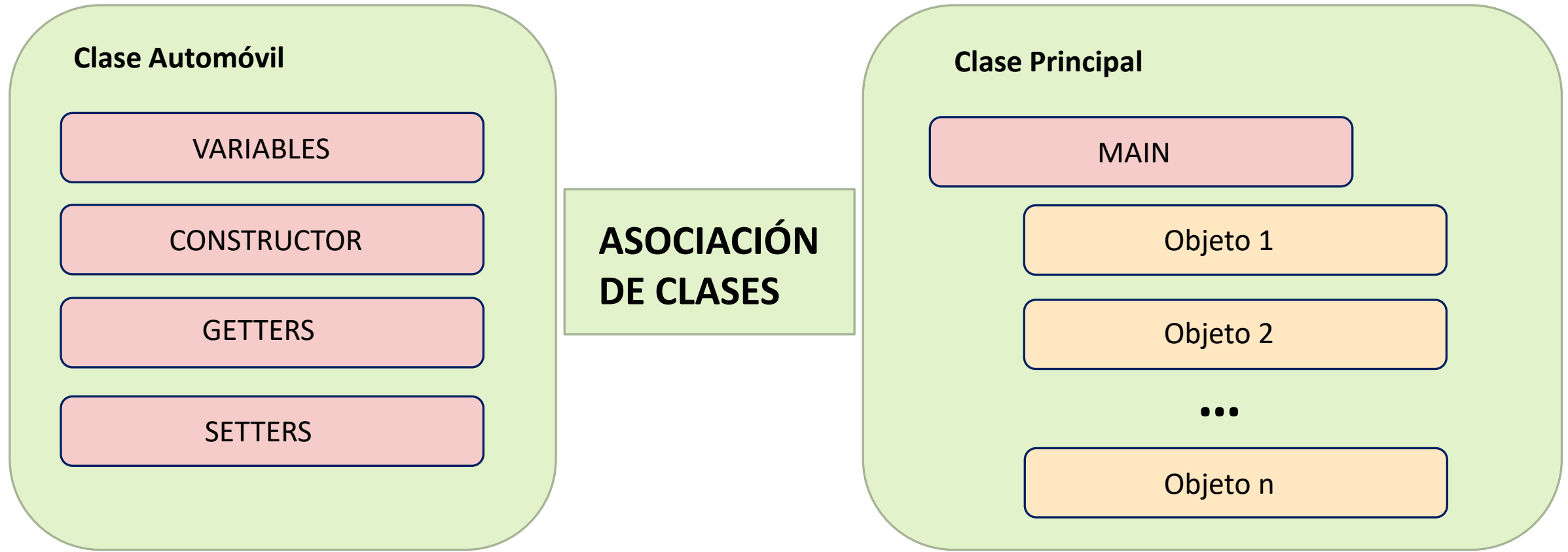
Los **objetos de software**, al igual que los objetos del mundo real, también tienen variables de clase y métodos. Un **objeto** de software mantiene sus características en una o más *variables*, e implementa su comportamiento con *métodos*. Un método es una función o subrutina asociada a un **objeto**.



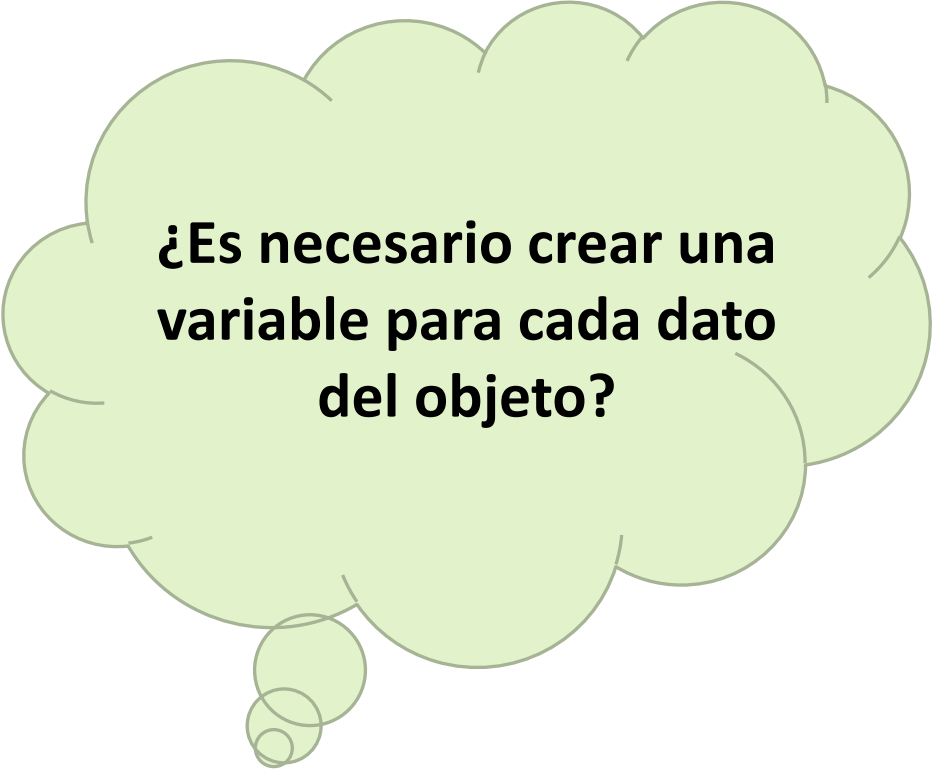
Supongamos un objeto automóvil con las siguientes características:

- Marca: Cupra
- Modelo: León
- Color: Gris Magnetic
- Transmisión: Automático
- Velocidad máxima: 230 km/h

Una clase en Java debe tener los siguientes elementos:



```
class Automovil {  
  
    // VARIABLES DE CLASE  
  
    private String marca;  
    private String modelo;  
    private String color;  
    private boolean transmision;  
    private String velocidadMaxima;  
}
```



¿Es necesario crear una variable para cada dato del objeto?

CONSTRUCTOR



```
// CONSTRUCTOR QUE INICIALIZA LAS VARIABLES DE CLASE

public Automovil (String marca, String modelo, String color, boolean transmision, String velocidadMaxima) {
    this.marca = marca;
    this.modelo = modelo;
    this.color = color;
    this.transmision = transmision;
    this.velocidadMaxima = velocidadMaxima;
}
```

Un método constructor en Java es un método especial que se invoca cada vez que se genera un objeto de la clase a la que pertenece. Dependiendo de las necesidades, una clase puede tener uno o varios constructores.

Características de un constructor en Java:

- Debe tener el mismo nombre de la clase a la que pertenece
- Se declaran como **públicos** para que puedan ser invocados desde fuera de la clase
- Puede tener parámetros o no
- Solo se ejecuta cuando es invocado para crear una instancia de un objeto



GETTERS Y SETTERS



GETTER

```
// METODOS GETTER Y SETTER PARA RECUPERAR O CAMBIAR LOS DATOS DE LAS VARIABLES DE CLASE
```

```
public String getMarca() {  
    return marca;  
}
```

```
public String getModelo() {  
    return modelo;  
}
```

...

```
public String getVelocidadMaxima() {  
    return velocidadMaxima;  
}
```

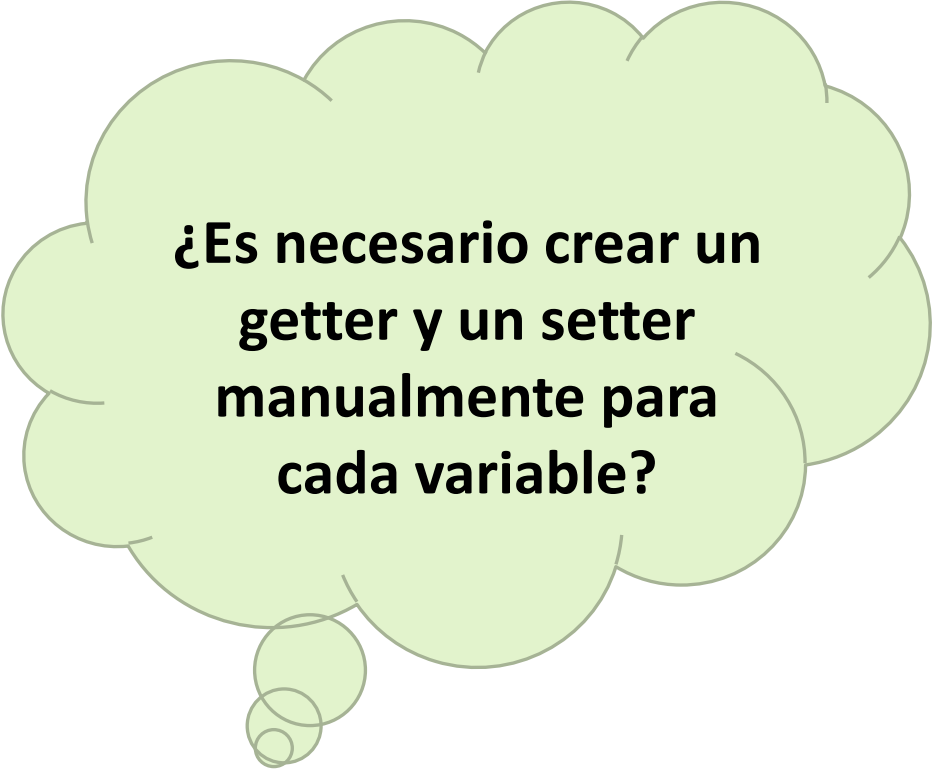
SETTER

```
public void setMarca(String marca) {  
    this.marca = marca;  
}
```

```
public void setModelo(String modelo) {  
    this.modelo = modelo;  
}
```

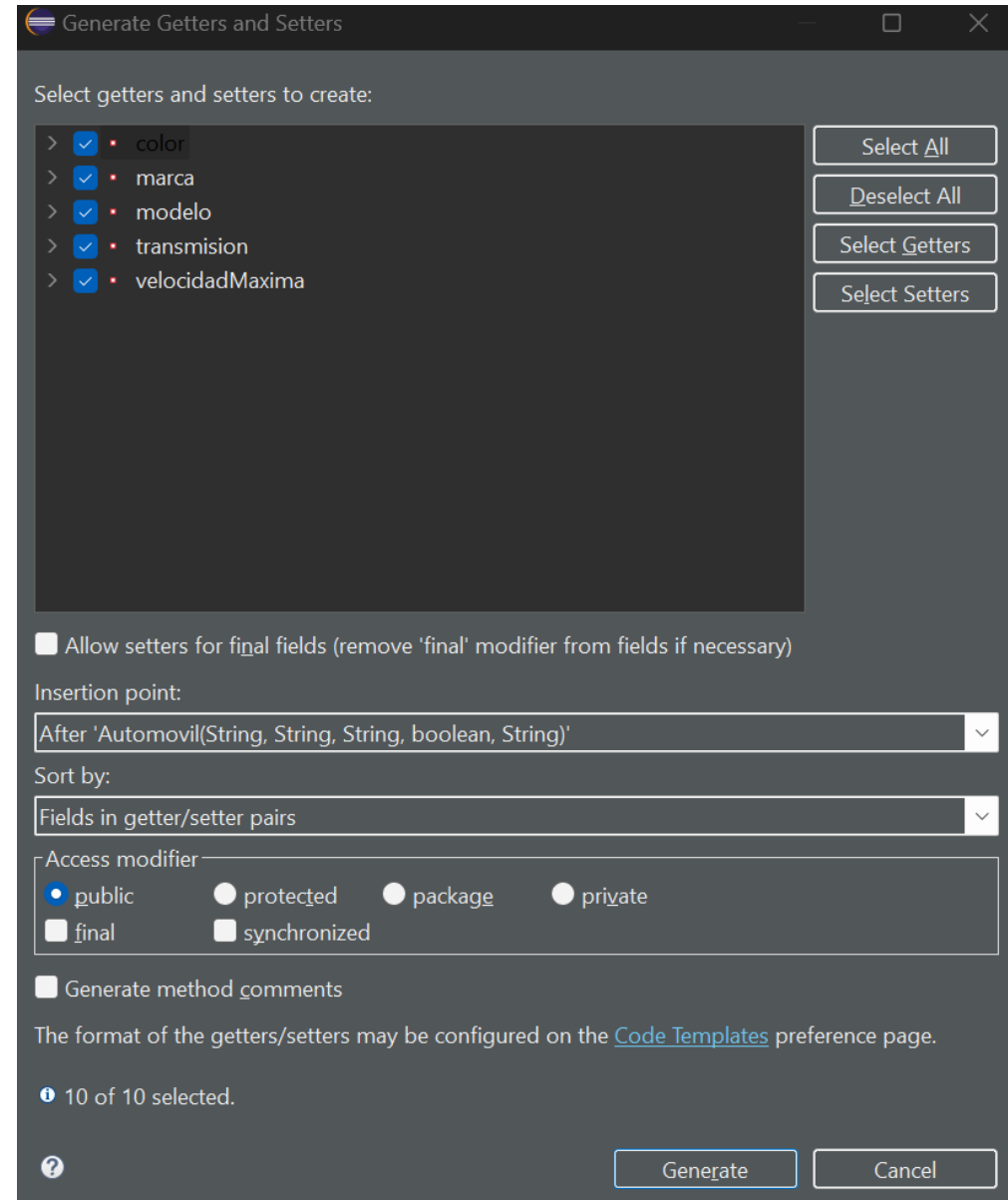
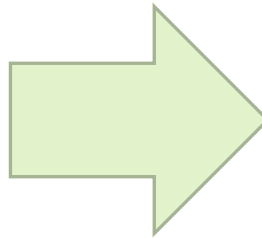
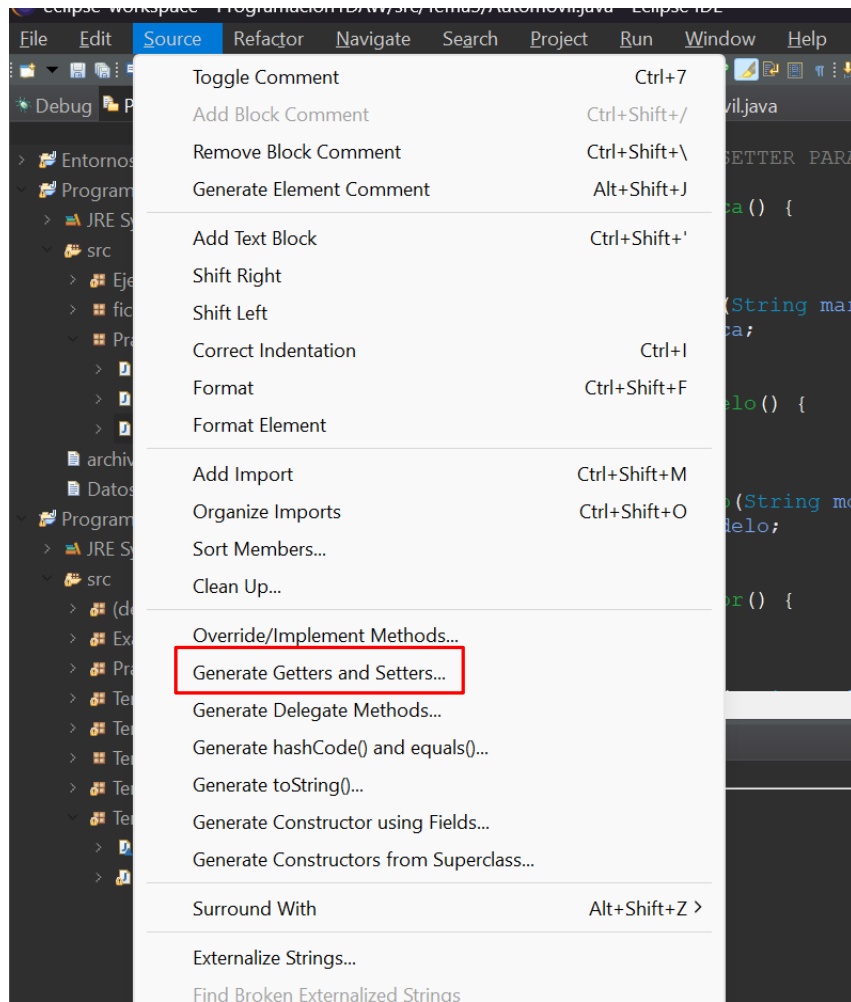
```
public void setColor(String color) {  
    this.color = color;  
}
```

```
public void setTransmision(boolean transmision) {  
    this.transmision = transmision;  
}
```



**¿Es necesario crear un
getter y un setter
manualmente para
cada variable?**

Para ganar algo de tiempo...



OBJETOS



En la clase principal, podemos crear una instancia de la clase Automovil (un objeto automóvil) sin necesidad de importar la clase, simplemente tendremos que declararlo de la siguiente forma:

```
public class PrincipalAutomovil {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        Automovil cupra_leon = new Automovil("Cupra", "Leon", "Gris Magnetic", true, "220 km/h");  
    }  
}
```

Y se pueden crear todos los objetos que queramos:

```
Automovil c4 = new Automovil("Citroen", "C4", "Negro", false, "190 km/h");  
Automovil i20N = new Automovil("Hyundai", "i20N", "AzulCielo/Rojo", true, "235 km/h");  
Automovil CLA180 = new Automovil("Mercedes", "CLA180", "Negro", true, "205 km/h");
```

Mostrar un objeto

```
System.out.println(cupra_leon);
```

```
Tema5.Automovil@626b2d4a
```

Es necesario acceder a una variable en concreto del objeto:

```
System.out.println("DATOS DEL AUTOMOVIL");  
System.out.println(cupra_leon.get);  
}  
}
```

- **getColor()** : String - Automovil
- **getMarca()** : String - Automovil
- **getModelo()** : String - Automovil
- **getVelocidadMaxima()** : String - Automovil
- **getClass()** : Class<?> - Object

onsole ×
minated> PrincipalAutomovil [Java Application] C:\Users\Dan
6_64_17.0

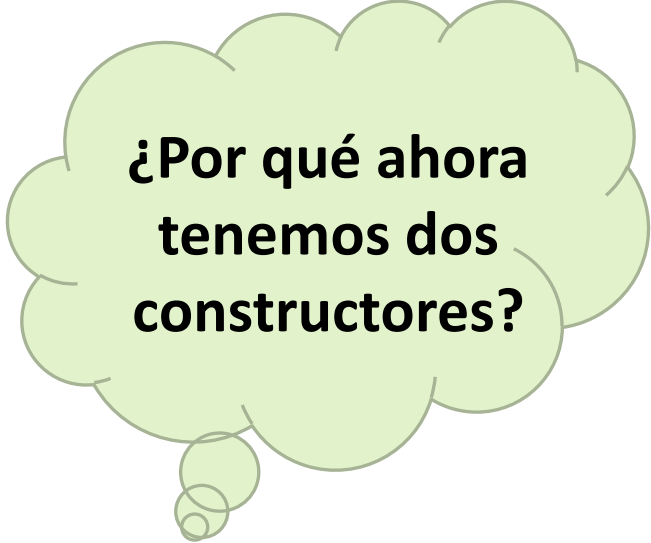
Mostrar un objeto

```
System.out.println("DATOS DEL AUTOMOVIL");  
System.out.println("Marca: " + cupra_leon.getMarca());  
System.out.println("Modelo: " + cupra_leon.getModelo());  
System.out.println("Color: " + cupra_leon.getColor());  
System.out.println("¿Es automático?: " + cupra_leon.isTransmision());  
System.out.println("Velocidad maxima: " + cupra_leon.getVelocidadMaxima());
```

```
DATOS DEL AUTOMOVIL  
Marca: Cupra  
Modelo: Leon  
Color: Gris Magnetic  
¿Es automático?: true  
Velocidad maxima: 220 km/h
```


Dar valor a un objeto

```
Automovil ateca = new Automovil();  
ateca.setMarca("Seat");  
ateca.setModelo("Ateca");  
ateca.setColor("Blanco");  
ateca.setTransmision(false);  
ateca.setVelocidadMaxima("200 km/h");
```



¿Por qué ahora
tenemos dos
constructores?

Para crear un objeto vacío, será necesario crear un nuevo constructor vacío (sin variables). A partir de ahí, podemos añadirle variables mediante los **setters**, siempre y cuando el setter exista en la clase Automovil.

ENCAPSULACIÓN

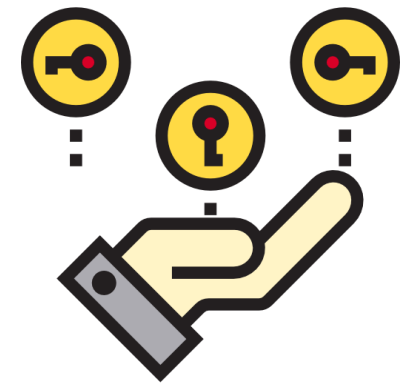


La **encapsulación** consiste en tener las variables de clase declaradas como **private** para evitar que cuando instanciamos un objeto podamos acceder a ellas (por seguridad).



Entonces, ¿cómo accedemos a los valores de esas variables de clase?

Mediante los métodos **getter** y **setter** que hemos declarado anteriormente.



HERENCIA

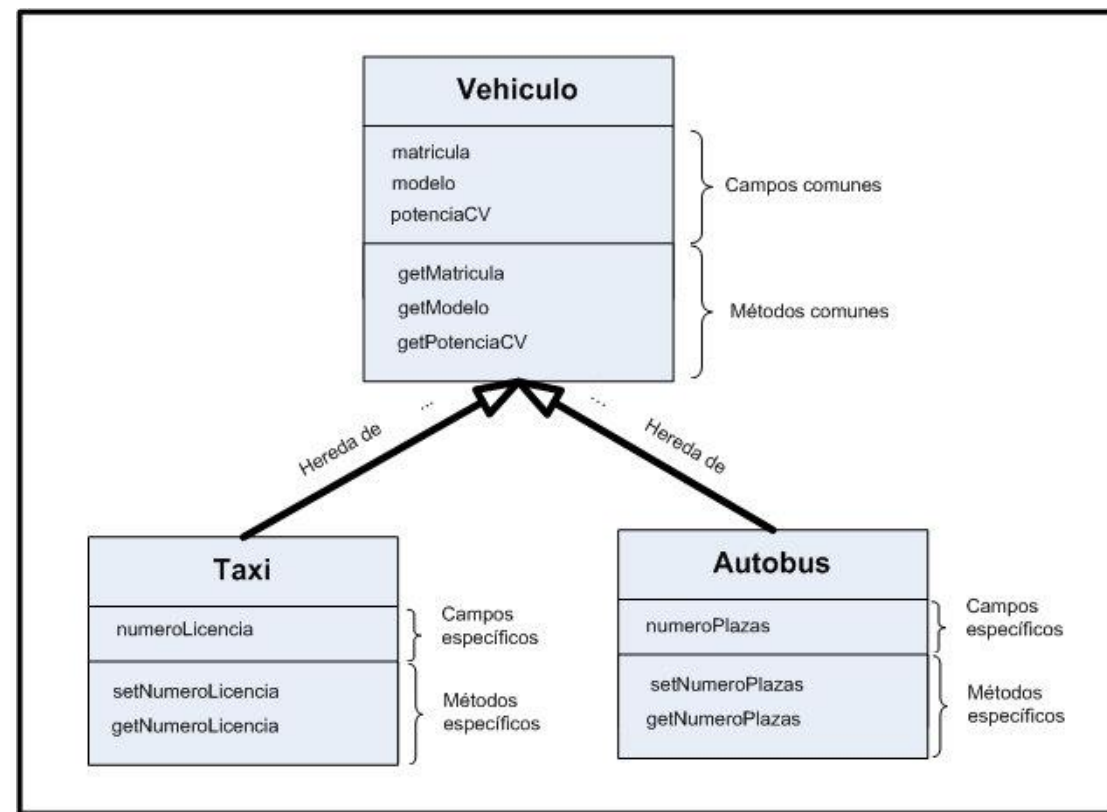


HERENCIA

La **herencia** en Java implica que una superclase hereda sus funciones y atributos a una subclase. La palabra reservada que nos permite realizar herencia entre clases es **extends**.

```
class Persona {  
  
    // atributos de la clase padre  
    String nombre;  
    int edad;  
    int telefono;  
  
}
```

```
class Cliente extends Persona{  
  
    //atributo de la clase hija  
    int credito;  
  
}
```



ABSTRACCIÓN



La **abstracción** es un concepto muy importante y se utiliza para reducir la complejidad de un programa, a través del uso de clases e interfaces abstractas, que permite la reutilización de código y la simplificación del mismo a través de la herencia.

Una **clase abstracta** es una clase base para otras clases llamadas “clases concretas”. La creación del objeto se hará a través de sus clases hijas, las cuales están obligadas a implementar los métodos abstractos definidos en la clase principal.

```
public abstract class Felino {  
    // clase abstracta  
  
    public abstract void Alimentarse(); //metodo abstracto  
}
```

```
public class Gato extends Felino  
{  
    public void Alimentarse()  
    {  
        System.out.println("El Gato come croquetas");  
    }  
}
```

POLIMORFISMO



El **polimorfismo** es la capacidad que tienen los objetos de una clase en ofrecer respuesta distinta e independiente en función de los parámetros (diferentes implementaciones) utilizados durante su invocación. Dicho de otro modo, el objeto como entidad puede contener valores de diferentes tipos durante la ejecución del programa.

```
class Animal {  
    public void makeSound() {  
        System.out.println("Grr...");  
    }  
}  
  
class Cat extends Animal {  
    public void makeSound() {  
        System.out.println("Meow");  
    }  
}  
  
class Dog extends Animal {  
    public void makeSound() {  
        System.out.println("Woof");  
    }  
}
```

```
public static void main(String[] args) {  
    Animal a = new Dog();  
    Animal b = new Cat();  
}
```

```
a.makeSound();  
//Outputs "Woof"  
  
b.makeSound();  
//Outputs "Meow"
```