

# Programación orientada a objetos

Comunidad de objetos

***Utilizar principios básicos  
de diseño orientado a  
objetos para la  
implementación de una  
pieza de software acorde al  
lenguaje Java para resolver  
un problema de baja  
complejidad***

- Unidad 1: Flujo, ciclos y métodos
- Unidad 2: Arreglos y archivos
- Unidad 3: Programación orientada a objetos
- Unidad 4: Pruebas unitarias y TDD



Te encuentras aquí



## ¿Qué aprenderás en esta sesión?

- *Comprender la sobrecarga de métodos.*
- *Comprender las variables de instancia, local y de clase.*

¿Qué entendemos por  
sobrecarga?



**/\* Sobrecarga de métodos \*/**

# Sobrecarga de métodos

- Permite crear varias versiones (ya que se utiliza el mismo nombre de método, pero se cambian los parámetros).
- Podemos mantener el código limpio, ya que podemos usar el mismo nombre (que define exactamente lo que el método hace) como es el caso del constructor estándar, que no recibe parámetros, y el constructor con parámetros, el cual es una sobrecarga del estándar.

# La sobrecarga de toString()

Hay un método muy útil que nos provee la clase padre y es `toString()`

- Devuelve la representación de una instancia en forma de String.
- De esta forma, podemos imprimirla con el método `System.out.println()`.

# Ejercicio guiado





# Vehículo

*Paso 1: Crearemos una clase Auto con sus respectivos parámetros.*

```
public class Auto(){  
    private String marca;  
    private String modelo;  
    private String color;  
    private int velocidadActual;  
    private boolean motorEncendido;  
}
```

# Vehículo

*Paso 2: Luego, crearemos el constructor respectivo.*

```
public Auto(){  
}  
public Auto(String marca, String modelo, String color, int velocidadActual,  
boolean motorEncendido){  
    this.marca = marca;  
    this.modelo = modelo;  
    this.color = color;  
    this.velocidadActual = velocidadActual;  
    this.motorEncendido = motorEncendido;  
}
```



# Vehículo

***Paso 3: Vamos a hacer una sobrecarga del método aumentarVelocidad para que, en caso de no recibir la velocidad por parámetro, aumente la velocidad en 10 y otra sobrecarga que reciba dos valores booleanos.***

```
public void aumentarVelocidad(int velocidad){
    velocidadActual = velocidadActual + velocidad;
}
public void aumentarVelocidad(){
    velocidadActual = velocidadActual + 10;
}
public void aumentarVelocidad(boolean maximoCiudad, boolean maximoCarretera){
    if(maximoCiudad) {
        velocidadActual = velocidadActual + 50;
    }
    if(maximoCarretera) {
        velocidadActual = velocidadActual + 100;
    }
}
```

# Vehículo

*Paso 4: Vamos a utilizarlo con la instancia de Auto al final del método main para probarlo:*

```
Auto instanciaAuto = new Auto();  
System.out.println("Auto creado");  
instanciaAuto.aumentarVelocidad();  
System.out.println(instanciaAuto.toString());
```

-----

Impresión en pantalla:

[Auto creado Modelo.Auto@15db9742]

# Vehículo

*Paso 5: Al final de la clase Auto, hacemos clic derecho y elegimos la opción Source -> Generate toString(), tal como se muestra en la imagen a continuación.*



# Vehículo

*Paso 6: Guardamos los cambios y al dar clic en “Play” se ve que el resultado de `System.out.println(instanciaAuto.toString())` cambió.*

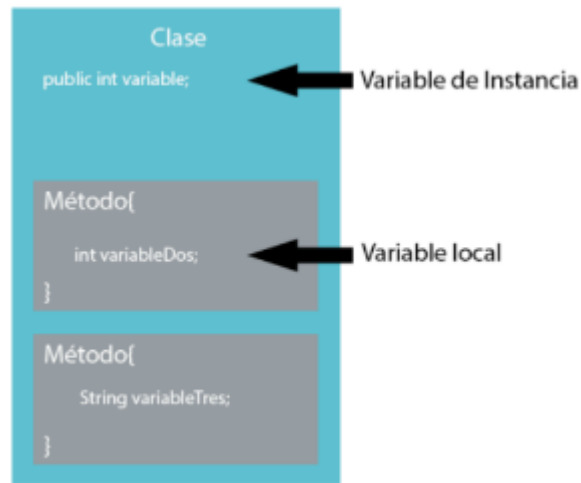
```
Auto creado  
Auto [marca=null, modelo=null, color=null, velocidadActual=0, motorEncendido=false,  
]
```

***/\* Variables y métodos \*/***

# Las variables de instancia y las variables locales

**Variables de instancia:** se mantienen "almacenadas" dentro de las instancias.

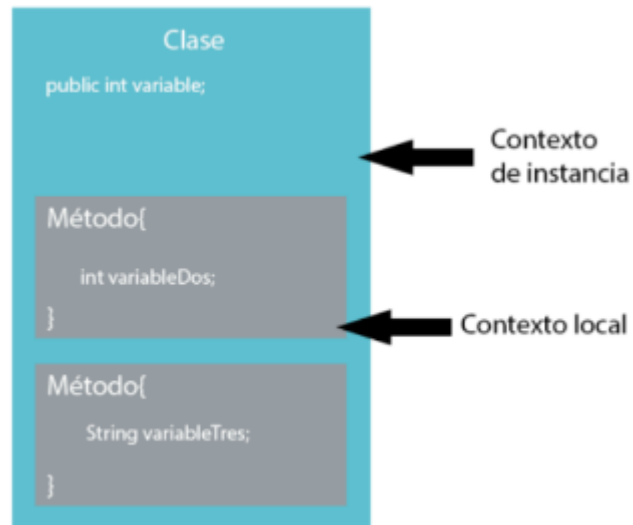
**Variables locales:** se declaran dentro de los métodos y, al terminar la ejecución del método, se descartan.





# Las variables de instancia y las variables locales

En Java existen diferentes contextos o scopes. Como podemos ver, las variables de instancia están en el contexto de la instancia y las variables locales están en el contexto de los métodos.



# Variables de clase

Se caracteriza porque los valores se mantienen estáticos para todas las partes del programa, ya que no necesitan ser instanciados para ser utilizados, basta solamente con llamar al elemento en cuestión utilizando la clase que lo contiene.

Por ejemplo, si creamos una variable de clase dentro de la clase Auto y la nombramos "pruebaEstatica", para usarla en otras partes del programa deberíamos llamarla como "Auto.pruebaEstatica". Esta nos devolvería el valor de la variable en cuestión, asimismo, cuando la llamamos desde otra parte del programa, la variable tendrá ese valor, sin necesidad de crear una instancia de Auto para utilizar la variable.

# Métodos de clase

Sirven, por ejemplo, para hacer cálculos genéricos y, si solamente queremos llamar al método una vez y luego no usaremos más la clase que contiene el método, vamos a tener que crear una instancia solo para eso, lo que hará que el código se vea sucio y se utilice memoria que podría ahorrarse.

Otro ejemplo sería el que entrega Java con sus métodos de clases más útiles:

**`Math.random()`** ; : El cual genera un número decimal aleatorio entre 0 y 1.

Esto es posible en cualquier parte de la aplicación gracias al modificador de acceso public del método, si hubiera otro modificador también debe tenerse en cuenta en el contexto de clase.

**/\* Composición \*/**

# Composición

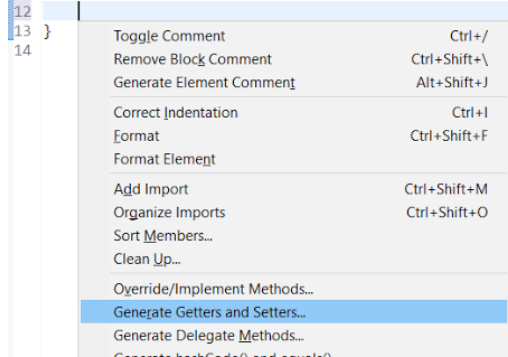
Así como en nuestra comunidad de objetos tenemos la herencia, donde una clase es hija de otra, también tenemos la posibilidad de almacenar una clase dentro de otra, en forma de atributo, en estos casos, se habla de que un objeto está compuesto por otro menor.




# Composición

Creamos la clase SistemaSonido y luego se la agregamos al Auto como uno de sus atributos y creamos su encapsulamiento gracias a las sugerencias de eclipse. En este ejemplo se muestra la forma de crear el encapsulamiento con las sugerencias.

```
1 package Modelo;
2
3 import java.util.List;
4
5 public class SistemaSonido {
6
7     private Integer volumenMaximo;
8     private Integer volumenActual;
9     private List<String> funciones;
10    private String funcionActual;
11
12
13 }
14
```



```
3 public class Auto extends Vehiculo {
4
5     private String marca;
6     private String modelo;
7     private String color;
8     private Integer velocidadActual;
9     private Boolean motorEncendido;
10    private SistemaSonido sistemaSonido;
11
12    public void setMarca(String marca) {
13        this.marca = marca;
14    }
15
```



# Composición

Se puede probar la composición en el método main:

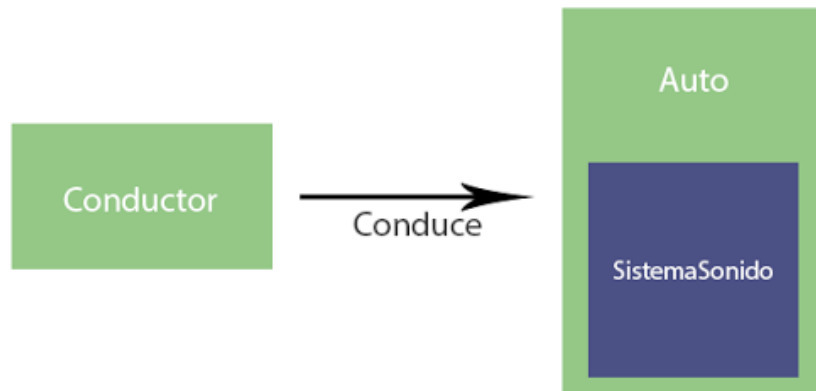
```
import Modelo.SistemaSonido;
public class MainClass {
    public static void main(String[] args) {
        Auto instancia = new Auto();
        System.out.println("Auto creado");
        instancia.encenderMotor();
        System.out.println("Sistema de sonido creado");
        SistemaSonido sistemaSonido = new SistemaSonido();
        sistemaSonido.setVolumenMaximo(100);
        instancia.setSistemaSonido(sistemaSonido);
        System.out.println(instancia.toString());
    }
}
```

***/\* Asociación \*/***



# Asociación

Así como una clase puede estar compuesta por otras, también se puede hacer una relación independiente entre dos clases, a este concepto, se le llama asociación.



# Asociación

Se puede probar la asociación en el método main:

```
public static void main(String[] args) {  
    Auto instancia = new Auto();  
    instancia.setMarca("Opel");  
    instancia.setModelo("Corsa");  
    instancia.setColor("Blanco");  
    Conductor conductor = new Conductor();  
    conductor.setAutoConducido(instancia);  
    conductor.setNombre("Juan");  
    System.out.println("Auto y conductor creados");  
    System.out.println(conductor.toString());  
}
```

¿Qué es lo que hace la  
sobrecarga de toString()?



¿Qué sucede con los valores  
de una variable de clase?



# ¿Cuál de las siguientes definiciones corresponde al concepto "sobrecarga de métodos"?

- 1. Un método que permite crear varias versiones que utilizan el mismo nombre de método, pero cambiando los parámetros.
- 1. Un método que permite crear varias copias de sí mismo, pero sin cambiar los parámetros.
- 1. Un método que devuelve la representación de una instancia en forma de String.



## Próxima sesión...

- *Comprender las excepciones de Java para optimizar nuestras aplicaciones.*
- *Aplicar Excepciones utilizando Try-Catch y “throw” para controlar y capturar las excepciones.*

**{desafío}**  
**latam\_**

*Academia de  
talentos digitales*

