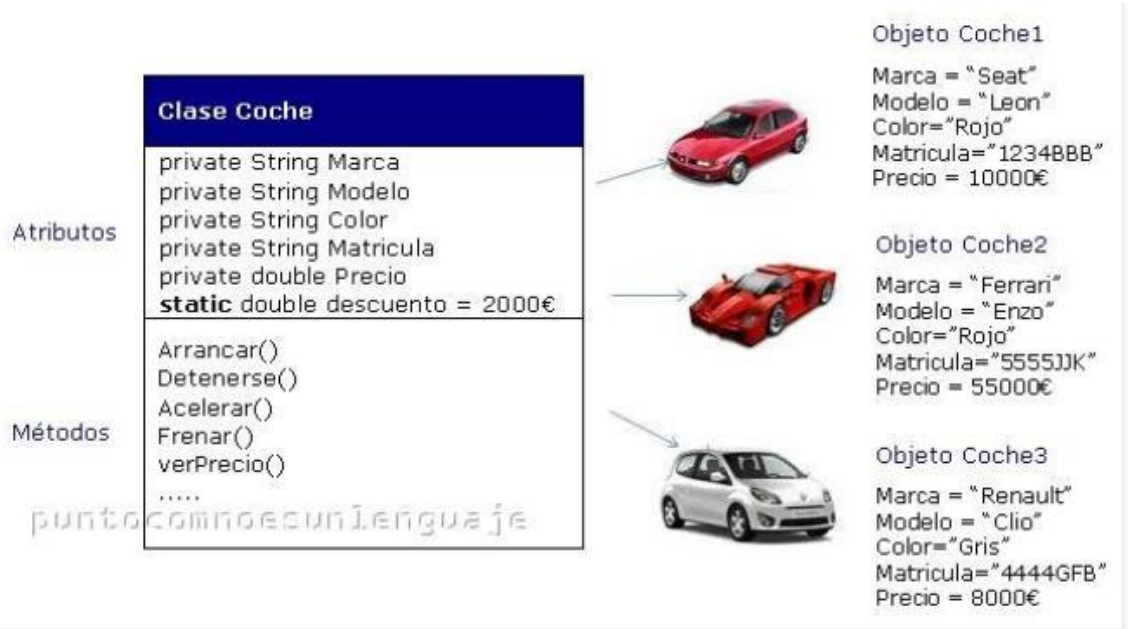
Java static. Atributos y métodos estáticos o de clase

Los atributos y métodos estáticos también se llaman **atributos de clase** y **métodos de clase**. Se declaran como **static**.

Supongamos una clase Coche sencilla que se utiliza en un programa de compra-venta de coches usados y de la que se crean 3 objetos de tipo Coche.

La clase contiene 6 atributos: marca, modelo, color, matrícula, precio y descuento. Supongamos que el descuento es una cantidad que se aplica a todos los coches sobre el precio de venta. Como este dato es el mismo para todos los coches y es un valor que se puede modificar en cualquier momento no debe formar parte de cada coche sino que es un dato que deben compartir todos. Esto se consigue declarándolo como **static**.



Atributos static:

Son **propios** únicamente **de la clase y no de los objetos que pueden crearse de la misma**, por lo tanto, sus valores son compartidos por todos los objetos de la clase. Van **precedidos del modificador static**.

Un atributo static:

- No es específico de cada objeto. Solo hay una copia del mismo y su valor es compartido por todos los objetos de la clase.
- Podemos considerarlo como una variable global a la que tienen acceso todos los objetos de la clase.
- Existe y puede utilizarse aunque no existan objetos de la clase.

Para invocar a una variable estática **no se necesita crear un objeto de la clase en la que se define**:

- Si se invoca **desde la clase** en la que se encuentra definido, basta con escribir su nombre.
- Si se le invoca **desde una clase distinta**, debe anteponerse a su nombre, el de la clase en la que se encuentra definido seguido del operador punto (.) <NombreClase>.variableEstatica.

Ejemplo1:

```
public class SerHumano{
        String nombre;
        String colorOjos;
        int edad;

        /*
         * Declaración e inicialización de una variable de instancia estática
         * Tiene sentido declararla estática pues todos los objetos
         * de la clase, teniendo en cuenta que esta modela a un ser humano,
         * habitan en el mismo planeta
         */
        static String planeta="Tierra";

        void mostrarCaracteristicas(){
                System.out.println(nombre+" tiene "+edad+" años");
                System.out.println("Sus ojos son "+colorOjos);
                System.out.println("Su planeta es "+planeta);
        }
        void esMayorEdad(){
                if(edad>=18){
                        System.out.println(nombre+" es mayor de edad");
                        System.out.println("Tiene "+edad+" años");
                }
                else{
                        System.out.println(nombre+" es menor de edad");
                        System.out.println("Tiene "+edad+" años ");
                }
        }
        public static void main(String args[]){
                SerHumano sh1=new SerHumano ();
                sh1.nombre="Jesus";
                sh1.colorOjos="azules";
                sh1.edad=28;
                sh1.mostrarCaracteristicas();
                sh1.eresMayorEdad();

                System.out.println("----------------------");
```

```
                SerHumano sh2=new SerHumano ();
                sh2.nombre="Rebeca";
                sh2.colorOjos="verdes";
                sh2.edad=27;
                sh2.mostrarCaracteristicas();
                sh2.esMayorEdad();

                System.out.println("---------------------");
                System.out.println("FIN DEL PROGRAMA");
        }
}
```

Métodos static:

Van **precedidos del modificador static**.

Para invocar a un método estático **no se necesita crear un objeto de la clase en la que se define**:

- Tienen acceso solo a los **atributos estáticos de la clase**.
- No es necesario instanciar un objeto para poder utilizarlo.
  Si se le invoca desde una clase distinta, debe anteponerse a su nombre, el de la clase en la que se encuentra seguido del operador punto (.)
  <NombreClase>.metodoEstatico

Ejemplo:

Vamos a escribir una clase Persona que contendrá un atributo contadorPersonas que indique cuántos objetos de la clase se han creado.
contadorPersonas debe ser un atributo de clase ya que no es un valor que se deba guardar en cada objeto persona que se crea, por lo tanto se debe declarar static:

```
public static int contadorPersonas;
```

Si lo declaramos como privado:

```
private static int contadorPersonas;
```

Desde fuera de la clase Persona solo podremos acceder al atributo a través de métodos static:

```
public static void incrementarContador(){
      contadorPersonas++;
}
public static int getContadorPersonas() {
```

```
        return contadorPersonas;
    }
```

En este caso un ejemplo de uso puede ser:

```
System.out.println(Persona.getContadorPersonas());
```

Cada vez que se crea una persona se incrementará su valor.

Si no es private, desde otra clase podemos hacerlo así:

```
Persona.contadorPersonas++;
```

Si es private, desde otra clase debemos incrementarlo así:

```
Persona.incrementarContador();
```

```
//Clase Persona
public class Persona {

   private String nombre;
   private int edad;
   private static int contadorPersonas;

   public Persona() {
    }

   public Persona(String nombre, int edad) {
      this.nombre = nombre;
      this.edad = edad;
   }

   public void setNombre(String nom) {
      nombre = nom;
   }

   public String getNombre() {
      return nombre;
   }

   public void setEdad(int ed) {
      edad = ed;
   }
```

```
    public int getEdad() {
        return edad;
    }

    public static int getContadorPersonas() {
        return contadorPersonas;
    }

    public static void incrementarContador() {
        contadorPersonas++;
    }
}


//Clase Principal
public class Estatico1 {
    public static void main(String[] args) {
        Persona p1 = new Persona("Tomás Navarra", 22);
        Persona.incrementarContador();
        Persona p2 = new Persona("Jonás Estacio", 23);
        Persona.incrementarContador();
        System.out.println("Se han creado: " + Persona.getContadorPersonas()
+ " personas");
    }
}
```

En lugar de utilizar el método incrementarContador() cada vez que se crea un objeto, podemos hacer el incremento de la variable estática directamente en el constructor.

El código de la clase Persona y de la clase principal quedaría ahora así:

```
//Clase Persona

public class Persona {

    private String nombre;
    private int edad;
    private static int contadorPersonas;

    public Persona() {
        contadorPersonas++;
```

```
    }

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
        contadorPersonas++;
    }

    public void setNombre(String nom) {
        nombre = nom;
    }

    public String getNombre() {
        return nombre;
    }

    public void setEdad(int ed) {
        edad = ed;
    }

    public int getEdad() {
        return edad;
    }

    public static int getContadorPersonas() {
        return contadorPersonas;
    }
}

//Clase Principal
public class Estatico1 {

    public static void main(String[] args) {
        Persona p1 = new Persona("Tomás Navarra", 22);
        Persona p2 = new Persona("Jonás Estacio", 23);
        System.out.println("Se han creado: " + Persona.getContadorPersonas()
+ " personas");
    }
}
```

No conviene usar muchos métodos estáticos, pues si bien se aumenta la rapidez de ejecución, se pierde flexibilidad, no se hace un uso efectivo de la memoria y no se trabaja según los principios de la Programación Orientada a Objetos.

NOTA: muchas clases de la API disponen de métodos estáticos. Por ejemplo, la **clase Math** del paquete java.lang cuenta con multitud de ellos. Estos métodos se emplean para realizar operaciones matemáticas. La **clase Thread**, del mismo paquete, cuenta con varios: uno que se emplea para retardar la ejecución de código es "**void sleep(long retardo)**". Consultar la API. Lo importante de estos métodos es que para su utilización no es necesario instanciar un objeto de las clases en las que se encuentran ya que son estáticos.

```
public class MetodosEstaticosAPI{
        public static void main(String args[]){
                int num=100;
                System.out.println("La raiz cuadrada de "+num+
                " es "+Math.sqrt(num));

                //Bloque try ... catch. Se estudiará más adelante. A nivel de
                //ejecución no afecta.
                //Se introduce un retardo en la ejecución del código de 3 sg
                try{
                        Thread.sleep(3000);
                }catch(InterruptedException e){}
                System.out.println("La potencia de 2 elevado a 8 es
                "+Math.pow(2,8));
        }
}
```

NOTA: **this** no se puede utilizar en un método estático

Bloques estáticos:

La estructura siguiente:

```
static {
  // ....
}
```

se llama un **bloque de inicialización estático** (**static initialization block**).

En cierta forma, actúa como un constructor de clase, pero en vez de inicializar los miembros de *instancia* de una clase, el bloque de inicialización estático permite

inicializar los miembros *estáticos* de la clase, cosa que no se puede hacer con los constructores normales.

El bloque de inicialización estático se ejecuta una sola vez para una clase dada, sin importar cuantas instancias se crean. Se ejecutará antes de cualquier constructor de instancia.

En resumen:

El bloque static es un bloque de instrucción dentro de una clase Java que se ejecuta cuando una clase se carga por primera vez en la JVM. Básicamente un bloque static inicializa variables de tipo static dentro de una clase, al igual que los constructores ayudan a inicializar las variables de instancia, un bloque static inicializa las variables de tipo static en una clase.

```
public class Bloque {
	static int a;
	static int b;
	static {
		a = 10;
		b = 20;
	}
	public static void main(String[] args) {
		System.out.println("Valor de a = " + a);
		System.out.println("Valor de b = " + b);
	}
}
```

Como podemos ver en este caso, las variables static a y b se inicializan automáticamente puesto que el bloque static junto con las variables static **es lo primero que carga la clase cuando se ejecuta**.

Finalmente es importante recordar que en un bloque static solo se puede inicializar variables de tipo static no variables de instancia.